Utrecht University

BACHELOR THESIS
BSC ARTIFICIAL INTELLIGENCE

Solving n-queens on non-squares

Author:
Jan Anthonie de Groot

Student Number:
6268188

Supervisor:
prof. dr. Hans L. Bodlaender

Second reader:
drs. Jeroen D. Fokker

A 7.5 ECTS thesis submitted in fulfilment of the requirements for the degree of Bachelor of Science

in

Artificial Intelligence

at the

Faculty of Humanities

17 July 2020

Abstract

Jan Anthonie de Groot

Solving n-queens on non-squares

Chess is a hugely popular game, and has been for ages. But it is not just the game itself, it has also give rise to many related games, problems and puzzles. One of the most famous is the n-queens problem. This classic puzzle tries to find the number of different arrangement of n queens on an $n \times n$ board, such that no queen attacks any other queen. On an 8×8 board, 8 queens can be placed in 12 different ways, extended to 92 different placings when counting each rotation and reflection separately.

This thesis has a similar goal, but without the restriction of having a square board. Guided by a hexagonal board and a three-player board, the n-queens problem is tackled by reducing it to the maximum independent set problem. This, in turn, is solved by computing the number of maximum cliques in the complement graph. The algorithm at the core of the accompanying computer program is the Bron–Kerbosch algorithm.

Keywords: N-queens problem, maximal clique problem, optimisation

Contents

Chapter 1

Introduction

Chess is an incredibly popular game that people have enjoyed playing for hundreds of years. Not only has a plethora of game variants emerged, the game is also used as the basis of numerous mathematical puzzles. One such puzzle is the eight queens puzzle, an instance of what would later be generalised to the n queens problem. This classic puzzle was first published in 1848 by the German chess composer Max Bezzel,[1] and then it has since become has been researched so extensively that it has become a teaching tool for various aspects of algorithmics, such as (linear) constraint programming.

Usually this problem is presented on a standard, square, $n \times n$ chessboard. The goal is easy: place as many queens as possible on a chessboard, such that no queen attacks any other queen. On an 8×8 board, 8 queens can be placed in 12 different ways, which can be extended to 92 different placings when counting each rotation and reflection separately.

A total amount of $({}_{\text{size}}\text{nCr}_{\text{queens}})$ possible placements means that the n-queens problem could potentially be computationally expensive. Reducing the n-queens problem to the maximum independent set problem by extension of the maximum clique problem means it is NP-hard, so probably no polynomial time algorithm will be possible.

The relevance of this thesis can be found in multiple locations. The first and most obvious place is in recreational mathematics, which concerns itself with mathematical games and puzzles, amongst other things. Another area where the implications could be found is in games. Strategy games in particular are set on a hexagonal grid—Civilization VI, Settlers of Catan, to name just two—because they provide 50% more movement options than square-based grids. In games like X-COM, where line-of-sight is an important game mechanic, one could see the similarity with the non-attacking queens. Combining these two ideas could prove useful in future games.

The aim of this research is similar to that of the classic problem, but applied to boards with sides $\neq 4$. Throughout the thesis, two shapes will serve as examples, or *guides* if you will, to aid in the general understanding. In an attempt to solve the puzzle, the n-queens problem will be reduced to the maximum independent set problem, which in turn will be solved by computing the maximum clique problem on the complement graph of the target board. How this works will be made clear in chapter 3.

Structure
In chapter 2 of this thesis, the n-queens problem and different types of boards are introduced, along with some theoretical background concerning graph theory. After that, chapter 3 presents the methodology and the core algorithm which we will use to solve the problem. The results are presented and discussed in chapter 4. The concluding chapter discusses the pros and cons of the method used, and the applications of the results.

Chapter 2

Theoretical background

This chapter introduces some definitions of the two main concepts. First, section 2.1 briefly goes into a selection of valid chess moves on different types of boards and formally defines the n-queens problem. Section 2.2 provides the necessary concepts and terminology of graph theory, which is the foundation of the method used to answer the research question.

2.1 N-queens problem

Standard chess is played on a square grid of size 8×8. We will call each of these 64 squares *fields*. In standard chess, the *queen* is the most powerful piece, as out of all chess pieces, she can reach the highest number of fields. From anywhere on the board, she can move either horizontally (2), vertically (2) or diagonally (4), so 8 directions in total. She may move as far as possible within the limits of the board, without changing direction during a move.

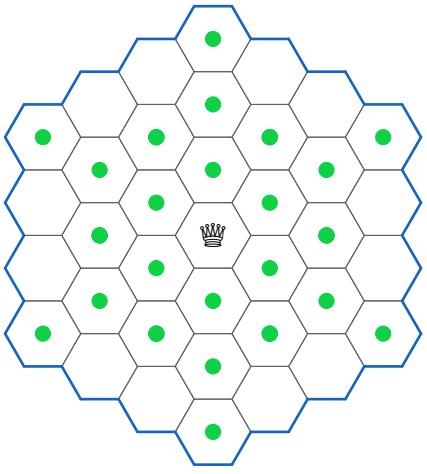


FIGURE 2.1: Hexagonal board with all possible moves from the centre. ($n = 4, h = 27$)

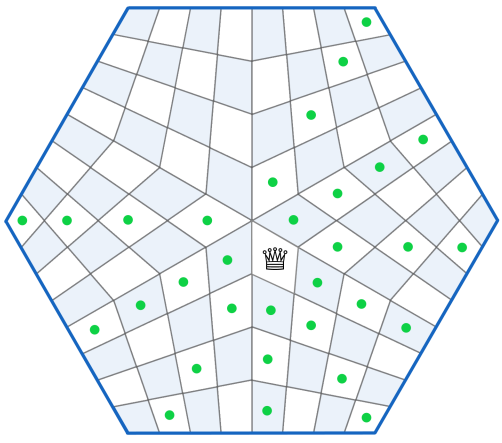


FIGURE 2.2: Three-player board with all possible move from one of the central fields.

A *hexagonal chessboard* is a regular hexagonal grid consisting of smaller hexagons, arranged in a regular tiling (honeycomb structure). We get the size h of such a board by the centered hexagonal number. This is calculated by the formula $h = 3n(n-1)+1$,[14] with n being the number of hexagons/fields at one side, as shown in 2.1. Using the rules of Gliński's hexagonal chess,[2] the queen can move in the direction of any side (orthogonally), as well as in the direction of any corner (diagonally). In the latter case, the resulting field is not directly adjacent to whence she came, but one field beyond that. Summing

the directions gives $6 + 6 = 12$ possible directions. As in standard chess, the distance is unrestricted within the limits of the board and changing course mid-move is not allowed.

The *three-player chessboard*, shown in Figure 2.2, is also a regular hexagonal grid, but the fields are quadrilateral polygons of varying shape and size. The common version is effectively three square chessboard halves fused together, and has 96 fields. It is not surprising, then, that the queen's movement is rather similar to the standard game.[4, 10] The only part where it differs is the centre: diagonal lines split in two. This means that the queen can move in a different number of directions depending on the position—apart from the edges of course, but that also applies to the other boards. On a non-centre field the queen can move in the regular 8 directions, whereas a central field facilitates 9 directions of movement.

Having described the boards and legal moves, we can now formally define the n-queens problem.

Definition 2.1.1 (N-queens problem)**.** Given a board of any size and shape, determine how to position the maximum number of queens on said board, such that no two queens attack each other.

We can, however, add another constraint.

Definition 2.1.2 (Optimal n-queens)**.** A solution to the n-queens problem is optimal iff every row and every column contains exactly one (1) queen.

Said differently, in the case of the hexagon: in an optimal solution, the number of queens is equal to the width of the board $(2n + 1)$.

2.2 Graph terminology

In this section we review the fundamentals of graph theory, including some higher-level graph terminology. A *graph* G consists of a finite nonempty set of *vertices* V and a set of *edges* E. Note: $\forall(u, v) \in E, u \neq v$. The edges are *unordered*, meaning they are bidirectional. Vertices u and v are *adjacent/neighbours* iff $u, v \in V$ and $(u, v) \in E$. The *degree* of a vertex is the number of neighbours it has. Let $N(u) = \{v \mid (u, v) \in E\}$ be the *neighbourhood* of a vertex u.

The *complement graph* $\overline{G}$ of a graph G is a graph with the same V as G, with edges between vertices that are not adjacent in G, and no edge if they are. A *clique* is a subset $V' \subseteq V$ of vertices of G, such that the induced subgraph is *complete*, i.e. every vertex is adjacent to every other vertex. A clique is *maximal* iff it ceases to be a clique when another vertex is added. A *maximum clique* is a maximal clique with the highest number of vertices, indicated by the *clique number* $\omega(G)$. Every clique in G constitutes an independent set in $\overline{G}$ and vice versa.[5] An *independent set* is a subset $V' \subseteq V$ of G, such that no vertex is adjacent to any other vertex.

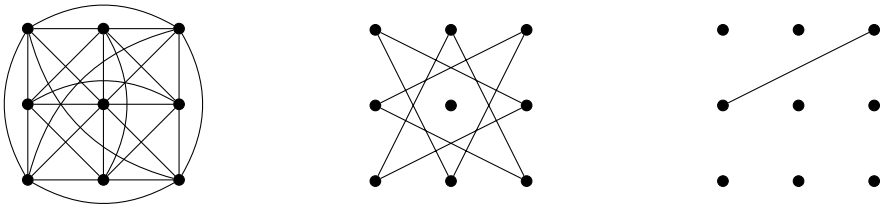
Chapter 3

Methodology

3.1 Maximum cliques

We will attempt to solve the n-queens problem by translating it to the maximum clique problem, as follows. Consider an arbitrary $n \times n$ chessboard. Each field on the board is represented by a vertex in a graph G. The edges in G represent all possible moves from any field/vertex. The goal of the n-queens problem (see 2.1.1) is to find the maximum number of *non-attacking queens*, in other words: to find the maximum number of *non-adjacent vertices*. The number of queens then corresponds to the clique number $\omega(G)$. The problem is now reduced to finding the maximum independent set in G. This is complementary to finding the maximum clique in $\overline{G}$.[5]

As the physical shape of the board has disappeared in the more abstract graph structure, we are not restricted to any shape, as demonstrated by the irregularly shaped three-player board.



(a) (b) (c)

FIGURE 3.1: (a): Graph containing all possible moves on a 3×3 board; (b): The complement graph of (a); (c): One maximal clique.[9]

3.1.1 Bron–Kerbosch

Because we want to find all possible solutions, not just one, the obvious algorithm to use is the Bron–Kerbosch algorithm[3, 17, 7, 8]. The pseudocode can be found in Algorithm 1. This backtracking algorithm is a two-parter, with an initialisation method and an execution method. The initialisation method creates three empty sets R, X, S and G's vertex set P. R is a potential clique and the vertices in P are candidates for clique R. Vertices in X must not be added to the clique, as that would result in a clique that has already been found. Maximal cliques will be collected in S.

R, X, and P are passed to the recursive execution method, where the algorithm chooses a *pivot* u from $P \cup X$, such that u is the vertex with the highest degree. The idea behind this heuristic is that for every u in $P \cup X$, because $P \cup X = N(R)$, any maximal clique must include u or one of u's non-neighbours.[17] The algorithm then selects a vertex v

Algorithm 1: Listing all maximal cliques

Data: $G = (V, E)$
Result: All maximal cliques in G

```
BronKerbosch()
1    set R = ∅
2    set P = G.V
3    set X = ∅
4    set S = ∅
5    BronKerbosch(R, P, X)

BronKerbosch(R, P, X)
1    if P == ∅ and X == ∅
2        S = S ∪ R // R is a maximal clique
3        return
4    vertex u = MaxDegree(P ∪ X)
5    for each vertex v in P \ N(u)
6        BronKerbosch(R ∪ {v}, P ∩ N(v), X ∩ N(v))
7        P = P \ {v}
8        X = X ∪ {v}
```

from P minus the neighbours of u to add to R (line 5). Since non-adjacent vertices have no influence on whether or not R is maximal, only the neighbours of v are included in P and X in the recursive call in line 6. In line 7 and 8, v is moved from P to X to reduce the number of recursive calls. Once P and X are both empty, we know that R is a maximal clique and it will be added to S (lines 1–3).[7]

Since we are only interested in finding the *maximum* cliques—the Bron–Kerbosch algorithm enlists all *maximal* cliques—the C# implementation adds an extra condition so cliques smaller than the (currently) largest clique are not added to S. The code of the C# implementation is included in C.[13]

3.1.2 GfG

To verify that my implementation of the Bron–Kerbosch algorithm outputs the correct data, I deployed a second algorithm. Originally I wanted to use an algorithm such as the one proposed by Östergård[15] or MaxCliqueDyn[11], but I couldn't get it to work in time, so I settled for a less academically appropriate one.[12] This recursive backtracking algorithm returns the size of the maximum clique, but unlike Bron–Kerbosch, does not enumerate all vertices that make up that clique.

3.2 Graph generation

Manually working out all the possible moves for all fields and representing those as edges in an edge list format gets impossibly time-consuming very quickly as n increases. For $n = 3$, there are already 123 edges. So in order to solve n-queens for larger boards, I needed a graph generator.

The generator roughly takes three steps: 1) represent a hexagonal grid as an array, 2) determine the possible moves, and 3) add those moves as edges to a graph.

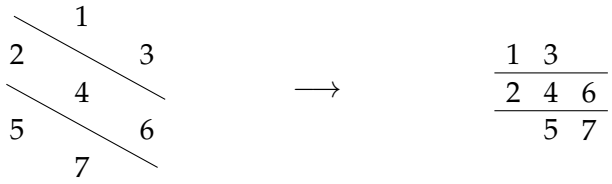


FIGURE 3.2: Hexagon representation in multidimensional array

To represent the hexagonal grid as a multidimensional array, we rotate the hex grid $-60°$ as in 3.2. The empty spaces are null.[16] Now we can use a coordinate structure to simulate movement. Orthogonal movement on the hexagon simply corresponds to an in-/decrease of the row or column in the array. For diagonal movement, it in-/decrements both the row and the column. Since the edges are unordered, it is only necessary to 'look' for vertices that have a higher number. See the method GenerateHexGraph in Appendix C for the complete set of operations.

The three-player board is a bit of a strange beast. Since it is effectively three halves of regular chessboards put together, direction and orientation change depending on the location. For instance, horizontal at the top row is vertical at the bottom . . . 'row'. Casting it to an array similar to the hexagonal board above doesn't quite work, so a different solution was required.

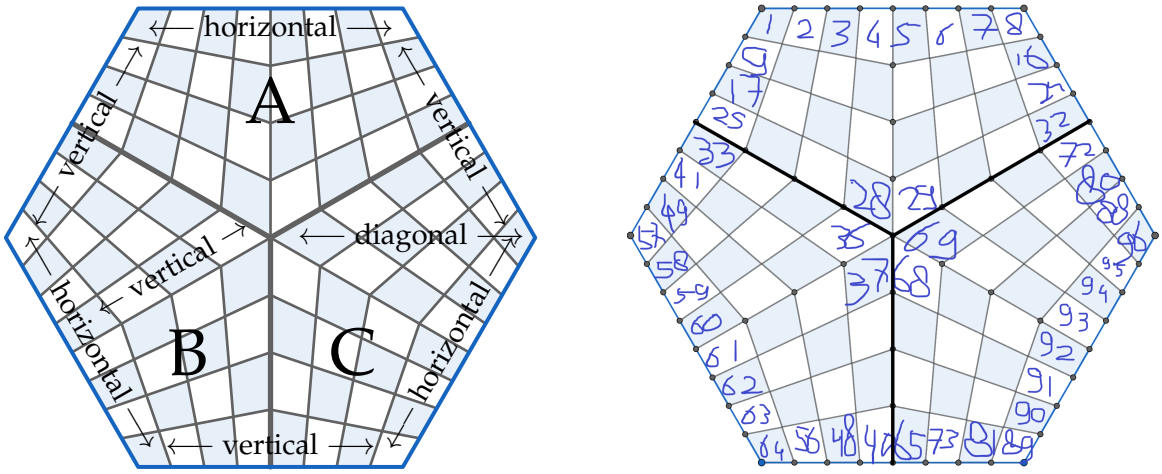


FIGURE 3.3: Left: different directions in different places; Right: numbering of the fields

I divided the board into three sectors, and assigned each of the fields a number 1–96 along the horizontal plane. This makes determining horizontal moves very easy, as you just iterate over all fields and add the other fields in that row. For vertical movement, the field that is 1 below has value +8. Because of the way the fields are numbered, there is a jump from sector A to C, and from B to C. These are all implemented as special cases, making it not the most elegant solution.

Chapter 4

Results

4.1 Counting solutions

In this section, we will explore the results as produced by the program. The complete set of solutions is listed in Appendix B. While the number of queens slowly increases as the board grows, the number of solutions does not. Its irregularity for different n immediately draws attention. This can be explained by looking at the number of queens compared to the width of the board. Boards with $n = 1$, $n = 4$ and $n = 8$ have optimal solutions—the number of queens is equal to the width. For $n = 5$ and $n = 6$, the number of queens is 2 less than the width, allowing a greater number of combinations. Looking further, $n = 7$ and $n = 9$, where the queens-to-width deficit is 1, have fewer solutions even though the board is bigger. As for why some boards yield an optimal solution and others do not, I cannot tell.

If two solutions are identical when rotated or mirrored, then they have the same fundamental solution. On the hexboard, a fundamental solution usually has 12 variants (including itself). These are created by rotating 60°, 120°, 180°, 240° or 300° and mirror each of those. However, if a solution is equivalent to its 60° rotation, it will have just two variants: the original and its reflection. This is the case for $n = 4$, for example.

n	width	size	fundamental	solutions	queens	optimal
1	1	1	1	1	1	yes
2	3	7	2	7	1	no
3	5	19	1	12	3	no
4	7	37	1	2	7	yes
5	9	61	16	172	7	no
6	11	91	35	390	9	no
7	13	127	2	24	12	no
8	15	169	1	6	15	yes
9	17	217		120	16	no

TABLE 4.1: Number of solutions and queens on an n-sized hexagonal board

The C# program has no method to calculate the number of fundamental solutions. For a small enough n, however, it is immediately visible from the total number of solutions. For nr_of_solutions $>$ 12, consider that most solutions are divisible by 12. Explaining this calculation in words is slightly cumbersome, so I will give an example.

For $n = 5$:

1. $\gcd(172, 12) = 4$
2. $172 - 4 = 168$
3. $168/12 = 14$
4. $4/2 = 2$
5. $14 + 2 = 16$

Figure 4.2 shows the results for the three-player board. I had only 'programmed' the three-player board with $n = 8$, but for sizes smaller than that it's not hard to work out. In the figure, I only included solutions for $n =$ even, because the rules pertaining the central fields need to be changed in order to work for odd widths.

n	size	fundamental	solutions	queens	optimal
2	6	1	6	2	yes
4	24	1	12	5	no
6	54	6	60	8	no
8	96	18 ± 1	208	12	yes

TABLE 4.2: Number of solutions and queens on an n-sized three-player board

4.2 Running time

The exponential time is reflected by the real-world running time, shown in 4.1. Using an Intel Core i5-3350P at 3.1GHz, computing maximum cliques took quite some time for $n > 6$. The figure on the right shows the same data as the figure on the left, but its y-axis is on a logarithmic scale. Since that line graph is approximately linear, the running time is indeed exponential.

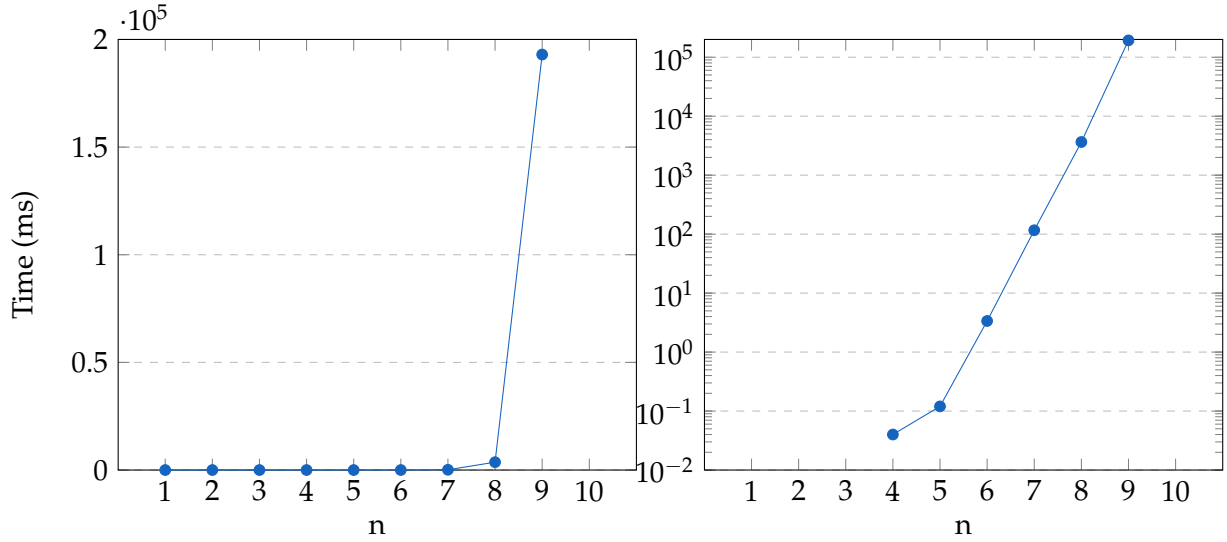


FIGURE 4.1: Running time of finding maximum queens for side n (left: linear y-axis, right: logarithmic y-axis)

Chapter 5

Discussion

In this final chapter, the strengths and weaknesses of solving the n-queens problem by reducing it to maximum cliques are discussed. I will also touch upon some topics that could be examined in further research, as well as the implications for artificial intelligence.

General discussion: algorithm
A strength of solving n-queens using graph theory is the versatility. The shape of the board becomes irrelevant; one could imagine a ♕-shaped board and the algorithm could find the maximum number of queens and their positions.

Right now, the program can't find fundamental solutions, because of the dissociation between graph and chessboard. An additional method that checks for every maximum clique whether a rotation and/or reflection results in another maximum clique, would have to be implemented. This method would then run after the maximum cliques have been found.

The Bron–Kerbosch algorithm could be improved by first sorting the vertices by degeneracy, a measurement of how sparse a graph is. The degeneracy of a graph G is the smallest number d such that every subgraph of G contains a vertex of degree at most d.[7] However, this supposedly only improves the running time for sparse graphs. Calculating the density of the graphs by applying $D = \frac{2|E|}{|V|(|V|-1)}$ shows that the density of both G and $\overline{G}$ increases with n. Adding a degeneracy-based heuristic in this situation would only be beneficial for $n < 5$. (See Appendix B)

Future research
One of the big unanswered questions raised by the results is: why are there no optimal solutions for $n > 1 \wedge n \,\%\, 4 \neq 0$? Does this even hold true for $n > 8$, or is this limited to $n = 4$ and $n = 8$? Using better computers, further research could be done on the number of queens and solutions for $n > 9$.

There are related puzzles that could be applied to the n-queens problem. These include finding the solutions that are symmetrical, or solutions with the property that no three queens are in a straight line (no-three-in-line problem[6]).

I believe that this research is a step for AI in the broadest sense, for mathematicians, and for puzzle fanatics. Recreational mathematics has been enriched, for a new problem has appeared. On a different part of the spectrum, a lot of games tend to use hexagonal grids, because they offer an efficient playing-ground. In strategy games, line-of-sight is sometimes an important factor, which can now be exploited in an unprecedented way.

Bibliography

[1] W. W. Rouse (Walter William Rouse) Ball. *Mathematical recreations and essays*. In collab. with University of California Libraries. London : Macmillan, 1917. 536 pp. URL: http://archive.org/details/mathematicalrecr00ballrich (visited on 06/07/2020).

[2] Hans L. Bodlaender. *Glinski's Hexagonal Chess*. The Chess Variant Pages. In collab. with John William Brown. 4th Jan. 2002. URL: https://www.chessvariants.com/hexagonal.dir/hexagonal.html (visited on 13/07/2020).

[3] Coen Bron and Joep Kerbosch. 'Algorithm 457: Finding All Cliques of an Undirected Graph'. In: *Commun. ACM* 16.9 (Sept. 1973). Place: New York, NY, USA Publisher: Association for Computing Machinery, pp. 575–577. ISSN: 0001-0782. DOI: 10.1145/362342.362367. URL: https://doi.org/10.1145/362342.362367.

[4] *Chess for Three*. The Chess Sets. Nov. 2015. URL: http://thechesssets.com/wp-content/uploads/2015/11/Chess-for-Three-prev.pdf (visited on 13/07/2020).

[5] Thomas H. Cormen et al. *Introduction to Algorithms*. 3rd. The MIT Press, 2009. 1320 pp. ISBN: 0-262-03384-4. URL: https://mitpress.mit.edu/books/introduction-algorithms-third-edition.

[6] Henry Ernest Dudeney. '317.—A puzzle with pawns.' In: *Amusements in mathematics*. In collab. with Boston College Libraries. London, Edinburgh, New York: Thomas Nelson and Sons, Ltd., 1917, p. 94. URL: https://archive.org/details/amusementsinmath00dude (visited on 17/07/2020).

[7] David Eppstein, Maarten Löffler and Darren Strash. 'Listing All Maximal Cliques in Sparse Graphs in Near-optimal Time'. In: *CoRR* abs/1006.5440 (2010). _eprint: 1006.5440. URL: http://arxiv.org/abs/1006.5440.

[8] David Eppstein and Darren Strash. 'Listing All Maximal Cliques in Large Sparse Real-World Graphs'. In: *CoRR* abs/1103.0318 (2011). _eprint: 1103.0318. URL: http://arxiv.org/abs/1103.0318.

[9] L. R. Foulds and D. G. Johnston. 'An Application of Graph Theory and Integer Programming: Chessboard Non-Attacking Puzzles'. In: *Mathematics Magazine* 57.2 (1984). Publisher: Mathematical Association of America, pp. 95–104. ISSN: 0025570X, 19300980. URL: http://www.jstor.org/stable/2689591.

[10] *How to Play Three Player Chess*. Yellow Mountain Imports. URL: https://www.ymimports.com/pages/how-to-play-three-player-chess (visited on 13/07/2020).

[11] Janez Konc and Dušanka Janežić. 'An improved branch and bound algorithm for the maximum clique problem'. In: *MATCH - Communications in Mathematical and in Computer Chemistry* 58 (21st June 2007). ISSN: 0340-6253. URL: https://www.csc2.ncsu.edu/media/colloquia1/unrestricted/Konc/All-Reprints-Konc.pdf.

[12] *Maximal Clique Problem - Recursive Solution*. GeeksforGeeks. 7th Feb. 2020. URL: https://www.geeksforgeeks.org/maximal-clique-problem-recursive-solution/ (visited on 17/07/2020).

[13] *Maximal Cliques (Bron Kerbosch with Pivot) : Java*. Algorithm KnapSack. Library Catalog: algosdotorg.wordpress.com. 28th Sept. 2014. URL: https://algosdotorg.

wordpress.com/maximal-cliquesbron-kerbosch-without-pivot-java/ (visited on 12/07/2020).

[14] OEIS Foundation Inc. *A003215*. The On-Line Encyclopedia of Integer Sequences. URL: https://oeis.org/A003215 (visited on 08/07/2020).

[15] Patric R. J. Östergård. 'A fast algorithm for the maximum clique problem'. In: *Discrete Applied Mathematics* 120.1 (2002), pp. 197–207. ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(01)00290-6. URL: http://www.sciencedirect.com/science/article/pii/S0166218X01002906.

[16] *Red Blob Games: Hexagonal Grids*. Library Catalog: www.redblobgames.com. URL: https://www.redblobgames.com/grids/hexagons/ (visited on 12/07/2020).

[17] Etsuji Tomita, Akira Tanaka and Haruhisa Takahashi. 'The worst-case time complexity for generating all maximal cliques and computational experiments'. In: *Theoretical Computer Science* 363.1 (2006), pp. 28–42. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2006.06.015. URL: http://www.sciencedirect.com/science/article/pii/S0304397506003586.

Appendix A

Additional graphs

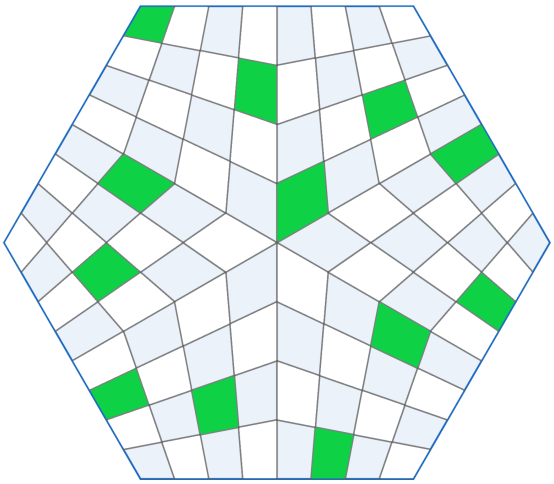


FIGURE A.1: Solution 0 for three-player board

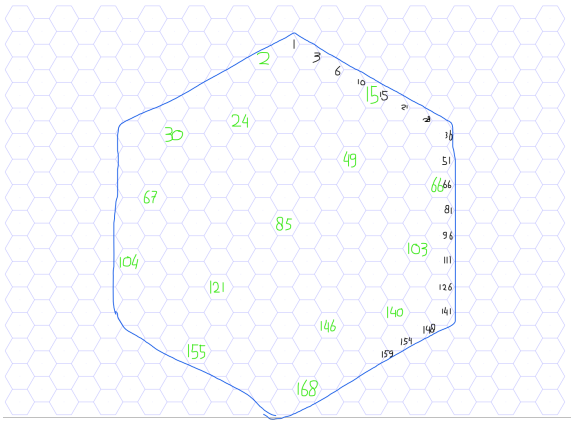


FIGURE A.2: Solution 0 for hexagonal board with $n = 8$

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	n	V	E	max edges	E / V	increment	3^(V/3) (maximal independent sets)	2E/(V(V-1)) (density)	min d	(V-d)3^(d/3)	max edges for d	fundamental	solutions	queens	width	q / V	q / width
2	2	7	21	21	3		13	1	6	9	21	2	7	1	3	0.142857	0.333333
3	3	19	123	171	6.473684	3.473684	1051	0.719298246	18	729	171	1	12	3	5	0.157895	0.6
4	4	37	372	666	10.05405	3.580370	766471	0.558558559	36	531441	666	1	2	7	7	0.189189	1
5	5	61	834	1830	13.67213	3.618077	5028813304	0.455737705	60	3486784401	1830	16	172	7	9	0.114754	0.777778
6	6	91	1575	4095	17.30769	3.635561	296946396793613	0.384615385	90	2.05891E+14	4095	35	390	9	11	0.098901	0.818182
7	7	127	2661	8001	20.95276	3.645064	157809490058396000000	0.332583427	126	1.09419E+20	8001	2	24	12	13	0.094488	0.923077
8	8	169	4158	14196	24.60355	3.650794	754797898855117000000000000	0.292899408	168	5.23348E+26	14196	1	6	15	15	0.088757	1
9	9	217	6132	23436	28.25806	3.654514	32491574563402200000000000000000000	0.261648746	216	2.25284E+34	23436		120	16	17	0.073733	0.941176
10	10	271	8649	36585	31.91513	3.657065	12587901705733200000000000000000000000000000	0.236408364	270	8.72796E+42	36585			18	19	0.066421	0.947368
11																	
12																	
13	n	V	E	max edges	E / V	increment	3^(V/3) (maximal cliques)	density	min d	(V-d)3^(d/3)	max edges for d						
14	2	7	0	21	0		13	0	1	8.653497422	6						
15	3	19	48	171	2.526316	2.5263158	1051	0.280701754	3	48	51						
16	4	37	294	666	7.945946	5.4196302	766471	0.441441441	10	1051.399937	315						
17	5	61	996	1830	16.32787	8.3819229	5028813304	0.544262295	20	62171.62539	1010						
18	6	91	2520	4095	27.69231	11.364439	296946396793613	0.615384615	35	20634914.1	2555						
19	7	127	5340	8001	42.04724	14.354936	157809490058396000000	0.667416573	54	28281695697	5373						
20	8	169	10038	14196	59.39645	17.349206	754797898855117000000000000	0.707100592	78	2.3131E+14	10101						
21	9	217	17304	23436	79.74194	20.345486	32491574563402200000000000000000000	0.738351254	106	8.00954E+18	17331						
22	10	271	27936	36585	103.0849	23.342935	12587901705733200000000000000000000000000000	0.763591636	139	1.6873E+24	27939						

FIGURE A.3: Analysis of the results

Appendix B

All solutions

```
n = 2 (Vertices: 7, Edges: 21)
Running time: 00:00:00.02
Maximum cliques: 7 solutions

Clique Size  Vertices
0      1     1
1      1     2
2      1     3
3      1     4
4      1     5
5      1     6
6      1     7
---------------------------------------
n = 3 (Vertices: 19, Edges: 123)
Running time: 00:00:00.00
Maximum cliques: 12 solutions

Clique Size  Vertices
0      3     1 9 13
1      3     1 11 12
2      3     2 6 15
3      3     2 13 14
4      3     3 4 15
5      3     3 12 16
6      3     4 8 17
7      3     5 14 18
8      3     5 16 17
9      3     6 7 18
10     3     7 11 19
11     3     8 9 19
---------------------------------------
n = 4 (Vertices: 37, Edges: 372)
Running time: 00:00:00.04
Maximum cliques: 2 solutions

Clique Size  Vertices
0      7     2 6 14 24 32 36 19
1      7     3 4 17 21 34 35 19
---------------------------------------
n = 5 (Vertices: 61, Edges: 834)
Running time: 00:00:00.120
Maximum cliques: 172 solutions

Clique Size  Vertices
0      7     1 12 19 29 52 60 41
1      7     1 12 24 29 35 18 41
2      7     1 12 24 29 41 59 18
3      7     1 12 24 29 55 59 18
```

```
4       7     1 12 24 43 41 59 18
5       7     1 12 24 43 55 59 18
6       7     1 12 24 43 58 59 18
7       7     1 12 52 29 18 35 41
8       7     1 16 14 33 55 59 39
9       7     1 16 24 38 27 58 59
10      7     1 16 24 38 35 56 54
11      7     1 16 32 38 35 56 54
12      7     1 20 19 42 26 56 60
13      7     1 20 19 42 36 53 58
14      7     1 20 33 36 17 14 39
15      7     1 20 33 39 60 14 17
16      7     1 20 33 52 60 14 17
17      7     1 20 46 39 60 14 17
18      7     1 20 46 52 60 14 17
19      7     1 20 46 56 60 14 17
20      7     1 30 42 36 58 19 53
21      7     1 55 33 17 14 36 39
22      7     2 6 16 22 39 45 28
23      7     2 6 16 22 39 60 28
24      7     2 6 16 22 47 45 28
25      7     2 6 16 22 47 60 28
26      7     2 6 16 42 48 61 45
27      7     2 6 20 28 51 39 57
28      7     2 6 20 42 36 61 43
29      7     2 6 22 39 51 28 45
30      7     2 10 16 33 47 60 40
31      7     2 10 16 42 48 45 61
32      7     2 10 20 31 42 52 60
33      7     2 10 20 32 51 43 54
34      7     2 10 20 32 51 56 54
35      7     2 10 21 33 43 61 50
36      7     2 10 29 42 48 61 45
37      7     2 15 20 52 57 37 18
38      7     2 15 20 52 58 37 18
39      7     2 15 21 40 58 37 18
40      7     2 15 21 52 57 37 18
41      7     2 15 21 52 58 37 18
42      7     2 15 22 29 52 60 46
43      7     2 15 29 52 18 57 37
44      7     2 15 34 56 60 46 40
45      7     2 18 21 37 40 58 43
46      7     2 23 16 42 45 61 48
47      7     2 23 29 42 48 61 45
48      7     2 23 34 40 56 60 46
49      7     3 4 19 22 41 44 25
50      7     3 4 19 22 41 59 25
51      7     3 4 19 22 51 44 25
52      7     3 4 19 22 51 59 25
53      7     3 4 19 38 50 61 44
54      7     3 4 22 47 25 41 44
55      7     3 4 24 25 47 41 57
56      7     3 4 24 38 35 61 46
57      7     3 7 19 29 51 59 40
58      7     3 7 19 38 50 44 61
59      7     3 7 23 29 46 61 48
60      7     3 7 24 30 47 46 53
61      7     3 7 24 30 47 58 53
62      7     3 7 24 31 38 55 59
63      7     3 7 33 38 50 61 44
64      7     3 11 22 33 55 59 43
65      7     3 11 23 40 56 34 17
66      7     3 11 23 55 56 34 17
```

```
67      7     3 11 23 55 57 34 17
68      7     3 11 24 55 56 34 17
69      7     3 11 24 55 57 34 17
70      7     3 11 33 55 17 57 34
71      7     3 11 37 58 59 43 40
72      7     3 17 23 34 40 56 46
73      7     3 21 19 38 44 61 50
74      7     3 21 33 38 50 61 44
75      7     3 21 37 40 58 59 43
76      7     4 9 15 20 26 32 49
77      7     4 9 15 20 52 46 49
78      7     4 9 15 38 55 49 32
79      7     4 9 15 38 55 59 32
80      7     4 9 15 55 26 32 49
81      7     4 9 42 20 26 49 43
82      7     4 9 42 20 26 61 43
83      7     4 9 43 20 49 32 26
84      7     4 9 43 20 61 32 26
85      7     4 9 43 55 26 32 49
86      7     4 10 20 27 51 52 57
87      7     4 10 20 32 51 43 54
88      7     4 10 25 41 47 60 44
89      7     4 10 25 42 47 60 44
90      7     4 19 22 41 25 44 60
91      7     4 22 47 25 60 41 44
92      7     5 7 15 29 45 26 48
93      7     5 7 15 29 55 26 48
94      7     5 7 15 38 55 56 36
95      7     5 7 23 29 45 48 26
96      7     5 7 23 29 55 48 26
97      7     5 7 28 29 51 59 45
98      7     5 7 28 38 51 45 59
99      7     5 7 28 39 51 59 45
100     7     5 10 11 33 44 27 50
101     7     5 10 11 33 52 27 50
102     7     5 10 11 42 52 58 35
103     7     5 10 21 33 44 50 27
104     7     5 10 21 33 52 50 27
105     7     5 10 25 33 47 60 44
106     7     5 10 25 41 47 60 44
107     7     5 10 25 42 47 44 60
108     7     5 11 23 42 34 56 60
109     7     5 21 15 38 37 58 59
110     7     5 21 27 33 38 44 50
111     7     5 42 29 26 45 48 23
112     7     6 7 24 26 47 55 57
113     7     6 7 24 30 47 46 53
114     7     6 7 28 38 51 59 45
115     7     6 7 28 39 51 59 45
116     7     6 8 11 24 27 30 49
117     7     6 8 11 24 55 43 49
118     7     6 8 11 42 52 49 30
119     7     6 8 11 42 52 60 30
120     7     6 8 11 52 27 30 49
121     7     6 8 38 24 27 49 46
122     7     6 8 38 24 27 61 46
123     7     6 8 46 24 49 30 27
124     7     6 8 46 24 61 30 27
125     7     6 8 46 52 27 30 49
126     7     6 16 22 39 28 59 45
127     7     6 22 39 51 59 28 45
128     7     7 13 19 30 36 58 53
129     7     7 13 19 38 51 56 54
```

```
130     7     7 13 24 30 47 46 53
131     7     7 13 24 30 47 58 53
132     7     7 13 36 30 47 58 53
133     7     7 14 33 39 55 36 57
134     7     7 14 33 47 55 57 36
135     7     7 23 29 45 61 48 26
136     7     8 11 19 42 52 49 30
137     7     8 11 19 42 52 58 30
138     7     8 11 19 42 52 60 30
139     7     8 14 25 31 37 48 54
140     7     9 12 28 31 34 50 53
141     7     9 15 16 38 55 49 32
142     7     9 15 16 38 55 56 32
143     7     9 15 16 38 55 59 32
144     7     10 12 29 41 52 35 57
145     7     10 12 29 51 52 57 35
146     7     10 13 16 32 35 56 54
147     7     10 13 16 42 47 58 53
148     7     10 13 20 32 51 43 54
149     7     10 13 20 32 51 56 54
150     7     10 13 35 32 51 56 54
151     7     10 21 33 44 61 50 27
152     7     11 23 17 34 40 56 60
153     7     11 37 40 58 59 18 43
154     7     13 16 24 38 56 54 35
155     7     13 16 32 38 56 54 35
156     7     13 20 42 53 58 19 36
157     7     13 30 42 19 58 53 36
158     7     13 30 42 47 58 53 36
159     7     13 35 38 51 56 54 32
160     7     15 21 18 40 58 59 37
161     7     15 34 40 56 60 17 46
162     7     21 18 37 40 58 59 43
163     7     21 33 38 27 44 61 50
164     7     23 17 34 40 56 60 46
165     7     23 29 42 26 45 61 48
166     7     47 14 33 55 17 57 36
167     7     51 12 29 52 18 57 35
168     7     57 12 24 29 35 18 41
169     7     57 12 52 29 18 41 35
170     7     57 20 33 36 14 17 39
171     7     57 55 33 14 17 39 36
----------------------------------------
n = 6 (Vertices: 91, Edges: 1575)
Running time: 00:00:03.368
Maximum cliques: 390 solutions

Clique Size  Vertices
0      9     1 12 20 33 41 82 88 70 62
1      9     1 12 20 33 43 47 74 70 61
2      9     1 12 20 33 43 47 90 70 61
3      9     1 12 20 38 65 77 58 90 51
4      9     1 12 20 38 65 77 69 90 73
5      9     1 12 26 33 64 19 88 89 72
6      9     1 12 26 38 65 77 69 19 73
7      9     1 12 26 38 65 77 69 90 73
8      9     1 12 26 38 65 77 88 83 19
9      9     1 12 26 40 49 65 77 69 90
10     9     1 12 26 50 65 77 69 19 73
11     9     1 12 26 50 65 77 69 90 73
12     9     1 12 32 38 81 19 61 83 58
13     9     1 12 32 38 81 19 89 66 58
14     9     1 17 14 38 37 73 45 66 64
```

```
15      9     1 17 14 38 37 89 66 64 45
16      9     1 17 14 40 37 85 86 66 63
17      9     1 17 14 43 60 81 56 89 52
18      9     1 17 14 43 60 81 67 89 74
19      9     1 17 26 40 54 44 90 75 67
20      9     1 17 37 38 88 89 66 64 45
21      9     1 17 43 60 81 67 89 36 74
22      9     1 22 14 37 61 18 86 90 75
23      9     1 22 14 41 54 60 81 67 89
24      9     1 22 14 43 60 81 67 18 74
25      9     1 22 14 43 60 81 67 89 74
26      9     1 22 14 43 60 81 86 84 18
27      9     1 22 14 53 60 81 67 18 74
28      9     1 22 14 53 60 81 67 89 74
29      9     1 22 20 41 49 48 89 72 69
30      9     1 22 29 43 50 81 86 84 36
31      9     1 22 29 43 60 81 67 36 74
32      9     1 22 29 43 60 81 86 84 36
33      9     1 22 43 49 45 85 89 72 36
34      9     1 22 43 60 81 67 89 36 74
35      9     1 27 43 77 18 64 14 84 56
36      9     1 27 43 77 18 90 70 14 56
37      9     1 30 38 53 77 88 26 83 34
38      9     1 30 38 65 77 69 26 34 73
39      9     1 30 38 65 77 88 26 83 34
40      9     1 33 43 20 86 90 70 61 47
41      9     1 38 20 65 77 69 90 34 73
42      9     1 38 54 47 82 90 26 75 34
43      9     1 38 65 77 69 90 26 34 73
44      9     2 6 17 31 70 55 74 24 78
45      9     2 6 17 31 70 55 74 91 78
46      9     2 6 22 31 49 45 68 59 72
47      9     2 6 22 31 49 45 91 59 72
48      9     2 6 22 42 49 65 86 90 68
49      9     2 6 27 41 34 55 85 87 48
50      9     2 6 27 41 34 72 85 48 79
51      9     2 6 27 41 34 77 85 87 48
52      9     2 6 27 41 76 77 87 48 50
53      9     2 6 27 42 65 86 90 50 46
54      9     2 6 27 43 77 50 90 70 24
55      9     2 6 27 43 86 50 90 70 24
56      9     2 10 17 38 36 76 67 87 63
57      9     2 10 17 38 36 76 77 87 63
58      9     2 10 17 38 48 76 77 87 69
59      9     2 10 17 38 54 45 58 91 66
60      9     2 10 22 37 65 71 88 45 68
61      9     2 10 22 37 65 71 88 78 50
62      9     2 10 27 36 23 76 77 84 50
63      9     2 10 27 36 23 76 77 87 50
64      9     2 10 27 37 65 82 55 74 46
65      9     2 10 27 37 65 82 55 90 46
66      9     2 10 27 48 76 77 35 84 56
67      9     2 10 27 48 76 77 84 23 50
68      9     2 10 27 48 76 77 87 23 50
69      9     2 10 63 36 76 77 23 87 50
70      9     2 10 69 23 76 77 87 48 50
71      9     2 15 17 38 54 67 90 75 46
72      9     2 15 17 38 54 77 75 90 46
73      9     2 15 17 43 66 52 86 80 24
74      9     2 15 22 19 42 65 71 69 78
75      9     2 15 22 19 42 65 71 88 68
76      9     2 15 22 19 42 65 71 88 78
77      9     2 15 22 19 65 71 85 78 57
```

78	9	2 15 22 42 49 65 86 90 68
79	9	2 15 22 65 49 74 91 78 36
80	9	2 15 22 65 49 85 57 78 36
81	9	2 15 27 19 42 23 91 66 80
82	9	2 15 27 19 54 66 23 91 58
83	9	2 15 27 19 54 66 23 91 80
84	9	2 15 27 19 54 72 23 91 58
85	9	2 15 27 19 54 72 23 91 80
86	9	2 15 27 19 65 66 85 86 58
87	9	2 15 27 41 54 72 80 91 34
88	9	2 15 27 43 52 23 91 66 80
89	9	2 15 27 43 52 86 66 80 24
90	9	2 15 27 43 77 85 87 23 36
91	9	2 15 27 53 76 66 86 80 24
92	9	2 15 33 19 42 65 71 88 68
93	9	2 15 33 19 42 65 71 88 78
94	9	2 15 33 43 61 52 86 80 24
95	9	2 15 33 53 61 76 86 80 24
96	9	2 15 33 65 71 52 88 78 24
97	9	2 15 49 43 77 69 90 23 46
98	9	2 15 49 43 77 85 87 23 36
99	9	2 19 22 42 65 71 69 50 78
100	9	2 19 22 42 65 71 88 45 68
101	9	2 19 22 42 65 71 88 50 78
102	9	2 19 27 48 76 77 87 23 50
103	9	2 19 69 23 76 77 48 87 50
104	9	2 21 27 72 85 48 25 34 79
105	9	2 21 27 77 85 87 48 25 34
106	9	2 25 17 38 48 52 79 66 75
107	9	2 25 17 38 48 52 91 66 75
108	9	2 25 17 38 54 67 90 75 46
109	9	2 25 17 38 54 77 90 75 46
110	9	2 80 22 31 49 45 91 59 72
111	9	3 4 20 28 66 59 73 24 80
112	9	3 4 20 28 66 59 73 91 80
113	9	3 4 26 28 54 47 68 55 75
114	9	3 4 26 28 54 47 91 55 75
115	9	3 4 26 39 54 60 88 89 68
116	9	3 4 32 38 81 53 89 66 24
117	9	3 4 32 38 88 53 89 66 24
118	9	3 4 32 39 60 88 89 53 46
119	9	3 4 32 40 36 59 82 87 44
120	9	3 4 32 40 36 75 82 44 79
121	9	3 4 32 40 36 81 82 87 44
122	9	3 4 32 40 71 81 87 44 53
123	9	3 7 20 43 34 71 69 87 62
124	9	3 7 20 43 34 71 81 87 62
125	9	3 7 20 43 44 71 81 87 67
126	9	3 7 20 43 49 47 56 91 70
127	9	3 7 26 33 60 76 86 47 68
128	9	3 7 26 33 60 76 86 80 53
129	9	3 7 32 33 60 85 59 73 46
130	9	3 7 32 33 60 85 59 89 46
131	9	3 7 32 34 25 71 81 83 53
132	9	3 7 32 34 25 71 81 87 53
133	9	3 7 32 44 71 81 35 83 58
134	9	3 7 32 44 71 81 83 25 53
135	9	3 7 32 44 71 81 87 25 53
136	9	3 7 62 34 71 81 25 87 53
137	9	3 7 67 25 71 81 87 44 53
138	9	3 11 20 38 70 51 88 78 24
139	9	3 11 20 43 49 69 89 72 46
140	9	3 11 20 43 49 81 72 89 46

```
141     9     3 11 26 18 39 60 76 67 80
142     9     3 11 26 18 39 60 76 86 68
143     9     3 11 26 18 39 60 76 86 80
144     9     3 11 26 18 60 76 82 80 57
145     9     3 11 26 39 54 60 88 89 68
146     9     3 11 26 60 54 73 91 80 34
147     9     3 11 26 60 54 82 57 80 34
148     9     3 11 32 18 39 25 91 70 78
149     9     3 11 32 18 49 70 25 91 56
150     9     3 11 32 18 49 70 25 91 78
151     9     3 11 32 18 49 75 25 91 56
152     9     3 11 32 18 49 75 25 91 78
153     9     3 11 32 18 60 70 82 88 56
154     9     3 11 32 38 51 25 91 70 78
155     9     3 11 32 38 51 88 70 78 24
156     9     3 11 32 38 81 82 87 25 34
157     9     3 11 32 40 49 75 78 91 36
158     9     3 11 32 50 71 70 88 78 24
159     9     3 11 37 18 39 60 76 86 68
160     9     3 11 37 18 39 60 76 86 80
161     9     3 11 37 38 64 51 88 78 24
162     9     3 11 37 50 71 88 78 24 64
163     9     3 11 37 60 76 51 86 80 24
164     9     3 11 54 38 81 67 89 25 46
165     9     3 11 54 38 81 82 87 25 34
166     9     3 16 32 75 82 44 23 36 79
167     9     3 16 32 81 82 87 44 23 36
168     9     3 18 26 39 60 76 67 53 80
169     9     3 18 26 39 60 76 86 47 68
170     9     3 18 26 39 60 76 86 53 80
171     9     3 18 32 44 71 81 87 25 53
172     9     3 18 67 25 71 81 44 87 53
173     9     3 23 20 43 44 51 79 70 72
174     9     3 23 20 43 44 51 91 70 72
175     9     3 23 20 43 49 69 89 72 46
176     9     3 23 20 43 49 81 89 72 46
177     9     3 78 26 28 54 47 91 55 75
178     9     4 9 15 22 53 61 80 83 57
179     9     4 9 15 22 53 71 57 80 83
180     9     4 9 15 22 54 71 88 68 29
181     9     4 9 15 27 53 77 88 83 46
182     9     4 9 15 27 54 62 58 91 66
183     9     4 9 15 27 54 66 73 91 80
184     9     4 9 15 27 65 77 88 83 46
185     9     4 9 15 39 53 77 88 83 46
186     9     4 9 15 39 54 62 58 91 66
187     9     4 9 15 39 65 77 88 83 46
188     9     4 9 21 22 50 48 75 79 29
189     9     4 9 21 22 60 48 75 79 29
190     9     4 9 21 22 60 75 82 79 29
191     9     4 9 60 36 75 82 22 79 29
192     9     4 10 22 30 59 51 91 72 80
193     9     4 10 22 32 60 81 82 36 74
194     9     4 10 22 32 60 81 89 36 74
195     9     4 10 27 36 62 76 82 87 55
196     9     4 14 21 22 60 81 68 89 42
197     9     4 14 21 27 42 50 90 70 73
198     9     4 14 21 27 42 82 90 70 55
199     9     4 14 21 27 77 90 40 59 68
200     9     4 14 21 27 77 90 59 50 73
201     9     4 14 21 27 77 90 70 50 73
202     9     4 14 21 28 42 81 89 55 68
203     9     4 14 41 22 54 60 81 68 89
```

204	9	4 14 41 22 54 72 81 89 68
205	9	4 14 41 28 54 55 81 89 68
206	9	4 20 29 38 76 48 66 84 57
207	9	4 21 27 47 50 90 70 24 73
208	9	4 21 27 47 82 74 70 55 24
209	9	4 21 27 47 82 90 70 55 24
210	9	4 21 27 77 59 90 24 50 73
211	9	4 21 27 77 73 90 70 24 50
212	9	4 21 38 77 88 70 83 24 63
213	9	5 7 15 43 29 56 64 33 84
214	9	5 7 15 43 29 71 64 33 84
215	9	5 7 15 43 49 77 56 90 69
216	9	5 7 15 58 44 65 51 86 90
217	9	5 7 15 58 44 65 71 90 67
218	9	5 7 15 65 49 77 56 90 69
219	9	5 7 15 78 29 71 64 33 84
220	9	5 7 21 29 60 33 85 86 58
221	9	5 7 21 33 60 52 59 73 80
222	9	5 7 25 29 71 33 64 78 84
223	9	5 7 31 33 60 52 59 80 73
224	9	5 7 37 44 65 51 86 90 58
225	9	5 7 37 49 41 76 80 83 45
226	9	5 10 11 38 30 58 61 37 83
227	9	5 10 11 38 30 76 61 37 83
228	9	5 10 11 38 54 81 58 89 67
229	9	5 10 11 56 48 60 52 88 89
230	9	5 10 11 56 48 60 76 89 69
231	9	5 10 11 60 54 81 58 89 67
232	9	5 10 11 80 30 76 61 37 83
233	9	5 10 16 30 65 37 82 88 56
234	9	5 10 16 37 65 51 55 74 78
235	9	5 10 23 30 76 37 61 80 83
236	9	5 10 28 37 65 51 55 78 74
237	9	5 10 33 48 60 52 88 89 56
238	9	5 10 33 54 40 71 78 84 47
239	9	5 11 21 39 25 74 89 48 67
240	9	5 11 21 39 25 85 89 48 67
241	9	5 11 21 39 30 85 67 89 58
242	9	5 11 21 39 60 52 88 89 48
243	9	5 11 21 39 60 74 89 48 67
244	9	5 11 21 39 60 85 67 89 48
245	9	5 11 21 39 60 85 67 89 58
246	9	5 11 21 49 30 85 72 89 58
247	9	5 11 21 49 48 85 25 89 72
248	9	5 16 15 42 23 73 90 44 69
249	9	5 16 15 42 23 82 90 44 69
250	9	5 16 15 42 29 82 69 90 56
251	9	5 16 15 42 65 51 86 90 44
252	9	5 16 15 42 65 73 90 44 69
253	9	5 16 15 42 65 82 69 90 44
254	9	5 16 15 42 65 82 69 90 56
255	9	5 16 15 54 29 82 75 90 56
256	9	5 16 15 54 44 82 23 90 75
257	9	5 16 25 54 29 75 82 90 56
258	9	5 16 31 54 44 81 72 84 23
259	9	5 23 21 49 72 30 85 89 58
260	9	5 23 31 44 54 81 58 72 84
261	9	5 28 21 49 77 75 48 83 25
262	9	5 28 25 49 48 77 56 75 83
263	9	6 7 27 26 65 77 85 34 73
264	9	6 7 27 26 65 77 90 34 73
265	9	6 7 29 26 55 52 91 75 78
266	9	6 7 32 34 63 71 85 87 59

267	9	6 8 11 26 49 76 86 68 30
268	9	6 8 11 26 50 64 78 84 57
269	9	6 8 11 26 50 76 57 78 84
270	9	6 8 11 32 49 63 56 91 70
271	9	6 8 11 32 49 70 74 91 78
272	9	6 8 11 32 50 81 86 84 46
273	9	6 8 11 32 60 81 86 84 46
274	9	6 8 11 42 49 63 56 91 70
275	9	6 8 11 42 50 81 86 84 46
276	9	6 8 11 42 60 81 86 84 46
277	9	6 8 16 26 53 44 72 79 30
278	9	6 8 16 26 65 44 72 79 30
279	9	6 8 16 26 65 72 85 79 30
280	9	6 8 65 34 72 85 26 79 30
281	9	6 12 16 26 65 77 68 90 39
282	9	6 12 16 31 77 90 59 39 68
283	9	6 12 16 32 39 53 89 66 74
284	9	6 12 16 32 39 85 89 66 59
285	9	6 12 16 32 81 89 41 55 68
286	9	6 12 16 32 81 89 55 53 74
287	9	6 12 16 32 81 89 66 53 74
288	9	6 12 40 26 49 65 77 68 90
289	9	6 12 40 26 49 75 77 90 68
290	9	6 12 40 31 49 77 90 59 68
291	9	6 16 32 45 53 89 66 24 74
292	9	6 16 32 45 85 73 66 59 24
293	9	6 16 32 45 85 89 66 59 24
294	9	6 16 32 81 55 89 24 53 74
295	9	6 16 32 81 74 89 66 24 53
296	9	6 16 43 62 81 86 66 84 24
297	9	6 17 30 43 71 44 70 83 57
298	9	7 13 20 27 62 58 86 66 84
299	9	7 13 20 27 62 76 86 66 84
300	9	7 13 20 44 65 51 86 90 58
301	9	7 13 20 44 65 71 90 58 67
302	9	7 13 26 27 53 34 80 55 83
303	9	7 13 26 27 53 76 55 80 83
304	9	7 14 21 27 77 35 90 70 73
305	9	7 14 43 27 77 35 64 84 56
306	9	7 14 43 27 77 35 83 70 56
307	9	7 14 43 27 77 35 90 70 56
308	9	7 15 27 19 65 66 85 86 58
309	9	7 15 27 19 65 77 85 46 73
310	9	7 26 33 60 76 86 19 47 68
311	9	7 32 19 60 33 85 59 73 46
312	9	7 32 19 60 33 85 59 89 46
313	9	8 11 20 38 34 48 61 69 87
314	9	8 11 20 38 76 48 87 61 69
315	9	8 11 26 60 54 82 57 80 34
316	9	8 14 16 42 81 35 86 66 84
317	9	8 14 21 28 67 59 85 87 63
318	9	8 14 21 28 77 85 87 59 63
319	9	8 14 21 33 60 82 70 42 79
320	9	8 14 21 38 45 82 52 87 59
321	9	8 14 28 42 81 86 66 84 35
322	9	8 14 33 42 60 82 56 70 79
323	9	8 15 16 36 65 44 82 90 57
324	9	8 15 16 42 65 82 69 90 44
325	9	8 15 16 42 65 82 69 90 56
326	9	8 15 28 49 65 57 85 78 36
327	9	8 15 28 49 65 78 74 91 36
328	9	8 16 26 54 44 35 88 72 63
329	9	8 28 21 49 77 85 87 59 63

```
330     9     8 28 36 49 59 77 85 87 63
331     9     9 11 21 34 60 48 85 89 57
332     9     9 11 21 39 60 85 67 89 48
333     9     9 11 21 39 60 85 67 89 58
334     9     9 11 31 34 54 60 57 82 80
335     9     9 11 31 34 54 60 80 73 91
336     9     9 12 16 31 55 69 82 87 62
337     9     9 12 16 31 55 81 82 87 62
338     9     9 12 16 37 65 39 85 66 79
339     9     9 12 16 43 47 85 51 87 55
340     9     9 12 21 39 77 35 88 70 83
341     9     9 12 58 65 39 37 85 66 79
342     9     9 12 70 31 77 88 83 39 35
343     9     9 15 17 43 36 44 64 67 87
344     9     9 15 17 43 71 44 87 64 67
345     9     9 15 22 65 49 85 57 78 36
346     9     9 16 31 54 55 81 82 87 62
347     9     9 22 21 49 48 35 86 75 62
348     9     9 55 31 54 81 82 34 87 62
349     9     10 11 32 18 60 70 82 88 56
350     9     10 11 32 18 60 81 82 46 74
351     9     10 12 16 32 81 35 89 66 74
352     9     10 12 38 32 81 35 61 83 58
353     9     10 12 38 32 81 35 84 66 58
354     9     10 12 38 32 81 35 89 66 58
355     9     10 13 17 32 63 56 88 70 83
356     9     10 13 17 32 63 71 88 70 83
357     9     10 13 17 48 60 52 88 89 56
358     9     10 13 17 48 60 76 89 56 69
359     9     10 13 22 32 50 36 78 59 84
360     9     10 13 22 32 50 71 59 78 84
361     9     10 22 37 45 65 71 88 18 68
362     9     10 27 18 65 37 82 55 74 46
363     9     10 27 18 65 37 82 55 90 46
364     9     11 18 32 39 25 91 74 70 78
365     9     11 18 32 49 63 25 91 70 56
366     9     11 18 32 49 70 25 91 74 78
367     9     13 17 26 54 40 90 44 75 67
368     9     13 17 32 44 71 88 70 83 63
369     9     13 17 42 44 71 88 70 83 63
370     9     13 20 22 49 41 89 48 72 69
371     9     13 20 27 48 76 86 66 84 62
372     9     13 20 39 48 76 86 66 84 62
373     9     15 19 27 42 23 91 73 66 80
374     9     15 19 27 54 62 23 91 66 58
375     9     15 19 27 54 66 23 91 73 80
376     9     16 32 39 53 12 25 89 66 74
377     9     16 32 81 53 12 89 25 66 74
378     9     21 27 42 50 23 90 14 70 73
379     9     21 27 77 50 14 90 23 70 73
380     9     24 17 37 38 64 88 89 66 45
381     9     24 33 43 20 61 86 90 70 47
382     9     55 27 37 14 64 82 87 18 41
383     9     55 27 37 14 76 82 87 18 41
384     9     55 28 19 54 26 47 91 75 78
385     9     59 12 32 33 61 85 87 19 40
386     9     59 12 32 33 71 85 87 19 40
387     9     59 22 31 49 45 91 18 72 80
388     9     68 12 26 19 64 33 88 89 72
389     9     68 22 14 18 61 37 86 90 75
----------------------------------------
n = 7 (Vertices: 127, Edges: 2661)
Running time: 00:01:56.367
```

```
Maximum cliques: 24 solutions

Clique Size Vertices
0      12   1 12 20 36 47 61 79 124 125 94 69 84
1      12   1 12 20 36 54 61 94 112 125 79 104 45
2      12   1 17 14 42 40 67 75 122 126 99 72 83
3      12   1 17 14 48 67 107 126 99 40 75 44 102
4      12   2 6 29 53 61 86 45 127 56 111 114 88
5      12   2 15 35 60 65 106 118 94 111 19 115 24
6      12   2 21 29 26 61 80 84 127 111 114 53 88
7      12   3 4 34 49 67 81 44 127 108 116 92 59
8      12   3 11 41 55 63 100 121 99 108 18 115 26
9      12   3 16 34 24 49 67 74 83 127 108 116 92
10     12   4 14 28 35 107 121 60 123 30 92 62 45
11     12   5 7 21 36 54 100 98 68 114 120 31 65
12     12   5 7 21 36 66 93 100 124 68 114 98 83
13     12   5 10 16 48 40 106 95 73 116 119 32 63
14     12   5 10 16 62 40 87 106 122 73 116 95 84
15     12   6 12 22 41 112 118 55 123 33 88 66 44
16     12   7 13 20 28 29 87 102 117 125 73 110 65
17     12   7 13 20 28 29 87 113 73 105 110 37 83
18     12   8 14 30 28 74 107 92 121 123 60 97 63
19     12   9 12 22 33 80 55 112 88 118 123 96 65
20     12   10 13 17 22 34 93 84 117 68 101 109 39
21     12   10 13 17 22 34 93 113 104 126 68 109 63
22     12   11 19 27 35 44 106 89 118 94 60 111 115
23     12   15 18 23 41 45 100 91 121 99 55 108 115
----------------------------------------
n = 8 (Vertices: 169, Edges: 4158)
Running time: 01:08:50.509
Maximum cliques: 6 solutions

Clique Size Vertices
0      15   2 15 30 24 66 104 67 155 49 168 85 103 140 146 121
1      15   3 11 35 59 111 73 159 46 167 85 97 135 26 144 124
2      15   4 10 23 51 52 119 80 90 160 166 85 118 147 33 137
3      15   6 7 44 58 27 126 75 95 163 164 85 112 143 32 138
4      15   8 14 21 22 31 81 89 148 149 85 156 162 139 72 98
5      15   9 12 16 28 34 74 96 142 154 85 158 161 136 68 102
----------------------------------------
n = 9 (Vertices: 217, Edges: 6132)
Running time: 53:36:32.476 (2 days, 5 hours)
Maximum cliques: 120 solutions

Clique Size Vertices
0      16   1 17 35 55 79 88 102 162 196 132 203 165 210 42 176 49
1      16   1 23 35 55 19 78 147 156 189 197 114 151 206 192 170 65
2      16   1 30 20 71 96 158 99 190 137 207 172 209 61 40 178 50
3      16   1 30 27 18 61 72 139 164 182 202 121 152 204 194 167 68
4      16   1 38 53 18 57 113 34 139 197 216 80 154 94 194 167 124
5      16   1 38 53 18 57 113 34 139 197 216 80 180 194 167 94 124
6      16   1 46 44 19 105 31 147 59 128 202 215 87 149 192 170 90
7      16   1 46 44 19 105 31 147 59 128 202 215 87 174 192 170 90
8      16   2 6 23 61 64 96 105 182 179 50 213 138 140 192 99 119
9      16   2 6 23 61 64 96 105 182 202 213 138 140 192 50 99 119
10     16   2 6 30 52 40 87 122 146 181 182 191 201 205 89 169 99
11     16   2 6 54 70 113 146 197 84 40 213 73 114 157 179 51 167
12     16   2 10 17 55 43 87 90 164 173 196 97 204 166 200 120 76
13     16   2 10 30 24 61 42 107 164 190 207 97 121 200 154 184 126
14     16   2 10 30 63 43 96 58 73 140 196 155 198 210 86 178 159
15     16   2 15 17 55 52 25 122 147 182 161 112 201 204 175 115 101
16     16   2 15 46 19 69 122 164 196 208 104 183 205 39 153 76 142
17     16   2 15 46 19 69 123 164 196 104 208 183 205 39 153 76 142
```

```
18      16    2 21 38 24 79 105 94 161 184 51 165 217 138 180 200 124
19      16    2 21 105 79 94 161 64 51 24 165 217 138 180 200 184 124
20      16    2 36 23 55 79 88 110 41 98 207 192 165 180 200 51 120
21      16    3 4 27 55 88 113 189 175 49 69 213 131 146 194 102 116
22      16    3 4 27 55 88 113 189 197 69 213 131 146 194 49 102 116
23      16    3 4 35 47 80 42 130 140 173 189 195 198 205 95 168 102
24      16    3 4 62 63 105 140 175 202 83 42 213 121 163 48 170 77
25      16    3 7 20 39 61 80 94 156 181 190 104 206 171 199 115 74
26      16    3 7 35 39 70 88 58 77 190 148 81 201 209 146 176 161
27      16    3 7 35 55 26 40 111 156 196 203 104 114 199 149 187 126
28      16    3 11 20 47 61 25 130 139 189 159 106 198 206 179 120 100
29      16    3 11 53 18 64 74 129 156 190 97 211 188 205 43 150 144
30      16    3 11 53 18 64 74 130 156 190 211 97 188 205 43 150 144
31      16    3 16 44 71 113 90 159 187 48 26 172 217 131 174 199 128
32      16    3 16 113 71 90 159 69 48 26 172 217 131 174 199 187 128
33      16    3 29 27 71 96 108 41 103 203 194 172 174 61 199 48 115
34      16    4 9 21 38 97 95 130 60 48 189 136 107 177 206 209 166
35      16    4 9 28 38 87 105 76 120 32 208 165 195 205 98 158 153
36      16    4 10 30 62 63 42 140 173 49 207 213 154 95 184 169 107
37      16    4 10 30 62 63 42 140 173 49 207 213 155 95 184 169 107
38      16    4 14 45 71 94 81 66 202 203 213 104 148 163 51 167 186
39      16    4 14 66 38 71 51 104 94 202 203 148 213 163 81 167 186
40      16    4 27 31 149 59 72 105 164 182 202 87 191 205 129 170 100
41      16    4 28 38 87 76 105 120 181 32 208 165 195 153 205 98 158
42      16    5 11 21 63 61 121 139 128 174 212 48 206 33 132 170 151
43      16    5 11 21 63 61 121 139 128 181 212 33 48 206 132 170 151
44      16    5 11 34 45 63 123 156 78 176 49 208 214 155 188 169 111
45      16    5 11 34 45 78 64 156 111 49 208 214 155 188 123 176 169
46      16    5 16 15 55 70 114 147 124 173 214 32 51 204 137 167 152
47      16    5 16 15 55 70 114 147 124 180 214 51 204 32 137 167 152
48      16    5 16 26 36 78 113 122 119 154 212 216 80 195 157 168 99
49      16    5 16 43 55 78 113 97 156 135 214 215 155 48 170 176 141
50      16    5 24 21 29 72 105 130 116 149 214 215 87 191 163 169 102
51      16    5 24 43 29 72 87 105 130 116 149 214 215 191 163 169 102
52      16    5 31 15 37 70 129 164 72 50 178 211 212 148 183 168 107
53      16    5 31 15 37 72 69 164 107 50 211 212 148 183 129 178 168
54      16    5 51 21 39 61 72 105 104 164 134 212 216 148 167 178 145
55      16    5 78 26 36 80 113 122 119 154 212 216 39 195 157 168 99
56      16    6 7 54 70 35 40 146 181 50 203 213 148 89 187 168 111
57      16    6 7 54 70 35 40 146 181 50 203 213 149 89 187 168 111
58      16    6 8 16 44 89 104 122 56 51 182 133 111 177 204 210 171
59      16    6 8 22 44 80 113 74 115 33 150 211 172 191 205 103 162
60      16    6 12 37 79 86 90 67 197 207 213 97 155 157 48 170 185
61      16    6 12 67 44 79 48 97 90 197 207 155 213 157 86 170 185
62      16    6 22 80 74 33 113 115 44 173 211 172 191 150 205 103 162
63      16    6 23 34 154 57 78 113 156 189 197 80 195 205 123 167 101
64      16    7 13 20 36 46 40 77 123 164 196 120 203 209 166 178 135
65      16    7 13 27 45 46 56 105 103 196 115 212 138 174 185 68 144
66      16    7 13 27 68 46 56 105 138 103 196 115 212 210 174 185 144
67      16    7 13 28 30 62 88 165 121 207 215 200 154 175 68 74 144
68      16    7 13 28 30 62 89 68 165 121 207 215 200 154 175 74 144
69      16    7 14 21 46 61 89 67 147 156 189 128 203 201 166 193 151
70      16    7 21 38 62 71 85 26 145 32 180 138 148 115 191 200 159
71      16    8 11 28 62 47 89 84 189 208 165 145 201 205 42 103 176
72      16    8 14 37 28 96 57 189 103 203 165 68 193 47 106 179 150
73      16    8 14 37 28 96 57 189 103 203 165 68 210 47 106 179 150
74      16    8 14 47 36 96 41 107 202 114 212 174 210 129 162 167 85
75      16    8 15 22 53 56 42 130 139 63 116 176 169 183 201 163 86
76      16    8 15 22 53 56 42 130 139 86 116 176 169 217 183 201 163
77      16    8 15 29 53 39 115 122 181 190 161 68 204 210 171 112 150
78      16    8 22 20 132 40 78 59 122 155 208 216 63 188 175 145 160
79      16    9 11 28 46 40 60 122 147 70 119 178 168 188 198 157 81
80      16    9 11 28 46 40 60 122 147 81 119 178 168 217 188 198 157
```

81	16	9 11 36 46 43 106 130 173 196 159 65 206 209 166 120 153
82	16	9 12 22 45 88 182 98 207 172 65 193 52 112 175 153 59
83	16	9 12 22 45 88 182 98 207 172 65 209 52 112 175 153 59
84	16	9 12 29 52 88 41 111 197 121 214 180 209 123 158 170 82
85	16	9 15 22 54 52 95 83 182 211 172 141 198 205 40 98 178
86	16	9 17 28 137 72 42 57 130 148 211 215 70 183 179 141 160
87	16	10 12 16 53 55 95 66 139 164 182 124 207 198 171 193 152
88	16	10 13 17 29 53 42 73 129 156 190 115 207 210 171 176 134
89	16	10 13 22 54 95 65 172 35 203 216 114 199 149 179 76 142
90	16	10 13 22 54 96 172 35 203 216 65 114 199 149 179 76 142
91	16	10 13 23 37 53 60 113 190 120 214 131 180 98 186 65 142
92	16	10 13 23 65 53 60 113 131 190 120 214 209 180 98 186 142
93	16	10 16 54 79 82 44 24 131 141 33 174 155 120 195 199 161
94	16	11 18 26 53 38 98 130 139 182 167 216 195 163 120 177 108
95	16	11 18 28 54 121 34 64 97 111 208 157 216 188 194 176 92
96	16	11 20 25 36 54 79 94 152 202 192 165 206 47 163 123 66
97	16	11 20 25 36 54 79 94 152 202 208 165 206 47 163 123 66
98	16	11 36 46 43 25 106 130 173 196 159 65 206 209 166 120 153
99	16	13 16 27 36 54 69 113 89 48 159 214 131 191 146 187 118
100	16	13 17 27 36 37 96 72 119 49 129 212 216 131 188 166 178
101	16	13 20 23 29 45 88 116 50 123 214 215 138 183 78 171 176
102	16	13 21 23 29 62 64 105 95 51 212 138 195 140 184 161 117
103	16	15 17 25 29 62 71 90 151 197 194 172 204 52 157 129 67
104	16	15 17 25 29 62 71 90 151 197 211 172 204 52 157 129 67
105	16	15 19 22 62 114 31 69 104 107 211 163 215 183 192 178 92
106	16	15 19 24 46 44 122 147 189 170 215 191 157 103 115 177 110
107	16	15 29 53 39 25 122 181 190 161 68 204 210 171 112 115 150
108	16	16 24 36 54 79 97 146 157 150 191 217 14 188 200 51 66
109	16	16 26 53 12 55 95 66 139 164 182 124 207 198 171 193 152
110	16	21 24 46 14 61 89 67 147 156 189 128 203 201 166 193 151
111	16	21 26 29 62 71 104 140 153 163 195 217 12 183 199 48 67
112	16	22 45 54 131 18 98 52 128 208 216 121 14 201 163 175 142
113	16	28 37 62 138 19 47 103 124 211 215 114 12 198 157 179 144
114	16	29 39 20 79 88 59 98 207 12 215 118 198 157 171 112 193
115	16	36 43 17 71 96 57 103 203 14 216 117 201 163 166 106 193
116	16	73 38 27 18 59 147 156 197 211 70 80 180 192 103 186 133
117	16	77 23 44 19 57 139 164 98 202 208 63 87 174 194 185 136
118	16	148 30 20 71 96 99 190 137 207 40 172 209 61 158 178 50
119	16	155 17 35 55 79 88 102 196 132 203 165 210 42 162 176 49

```
Three-player chess
n = 8 (Vertices: 96, Edges: 1140)
Running time: 00:02:05.504
Maximum cliques: 208 solutions

Clique Size  Vertices
0      12    1 12 23 34 47 51 62 72 73 84 94 29
1      12    1 13 23 36 48 26 51 61 80 91 66 86
2      12    1 14 24 38 44 61 65 90 79 85 27 50
3      12    1 14 24 39 50 61 73 93 71 83 27 44
4      12    1 15 30 39 54 42 59 72 81 92 20 77
5      12    1 15 30 40 42 59 72 91 53 74 85 20
6      12    2 12 22 32 38 41 63 65 93 51 76 87
7      12    2 12 22 32 39 41 51 62 73 69 95 84
8      12    2 12 23 25 40 45 62 72 94 51 77 82
9      12    2 13 23 25 35 45 55 60 80 67 86 89
10     12    2 13 23 25 35 48 60 80 54 92 66 86
11     12    2 13 24 25 35 45 55 60 81 94 71 75
12     12    2 13 24 28 33 43 53 63 73 94 71 83
13     12    2 13 24 28 33 47 59 53 81 94 71 75
14     12    2 13 24 28 33 48 59 90 53 71 86 75
15     12    2 14 19 31 39 41 52 62 80 81 68 93
16     12    2 14 19 31 40 41 52 62 80 90 85 68
17     12    2 14 24 25 36 47 51 62 65 93 76 87
18     12    2 14 24 27 33 44 55 61 65 75 95 85
19     12    2 14 24 27 33 44 55 61 71 73 93 83
20     12    2 14 24 27 33 47 62 65 93 52 76 87
21     12    2 14 24 27 33 47 62 71 52 81 93 76
22     12    2 15 19 30 39 41 52 62 72 81 92 77
23     12    2 15 19 30 40 45 49 60 72 91 74 85
24     12    2 15 21 27 33 44 55 61 72 73 94 83
25     12    2 15 21 32 33 44 55 59 65 91 86 76
26     12    2 16 22 25 35 48 60 54 92 66 77 87
27     12    3 13 18 32 33 44 55 61 70 73 95 83
28     12    3 13 18 32 33 44 56 61 66 83 94 79
29     12    3 13 18 32 33 44 56 61 90 70 87 75
30     12    3 13 18 32 33 47 62 65 94 52 79 84
31     12    3 13 18 32 33 47 62 70 52 87 76 89
32     12    3 13 18 32 38 44 49 61 65 82 94 79
33     12    3 13 18 32 38 44 55 57 65 94 79 84
34     12    3 13 18 32 39 49 61 73 95 83 44 70
35     12    3 13 23 25 36 48 58 88 91 53 66 78
36     12    3 13 23 26 36 48 49 61 88 91 66 78
37     12    3 13 24 25 36 42 53 64 66 94 79 83
38     12    3 13 24 25 36 48 58 66 53 95 83 78
39     12    3 13 24 26 36 46 49 61 66 81 94 79
40     12    3 13 24 26 36 48 49 61 66 95 83 78
41     12    3 13 24 26 40 49 60 82 45 67 95 78
42     12    3 14 18 33 44 55 61 72 73 95 83 29
43     12    3 14 18 33 47 62 65 88 93 31 52 76
44     12    3 14 18 38 44 57 65 88 55 93 31 76
45     12    3 14 18 40 41 52 62 31 85 74 96 68
46     12    3 14 20 25 40 50 62 72 74 95 85 45
47     12    3 14 20 25 40 50 62 72 90 77 45 87
48     12    3 14 24 25 38 44 63 65 93 76 50 87
49     12    3 14 24 26 37 41 52 63 65 75 95 85
50     12    3 14 24 26 37 41 54 60 66 79 85 89
51     12    3 15 18 32 33 44 55 61 70 73 93 83
52     12    3 15 18 32 33 47 62 70 52 81 93 76
53     12    3 15 18 32 37 47 49 60 73 93 83 70
54     12    3 15 18 32 38 41 60 81 93 70 76 55
55     12    3 15 18 32 39 44 49 61 73 93 83 70
```

```
56      12    3 15 25 40 42 53 63 72 77 94 83 20
57      12    3 15 25 40 58 72 55 77 45 94 83 20
58      12    3 16 18 38 44 49 61 65 93 31 82 78
59      12    3 16 18 39 41 52 62 81 68 93 31 78
60      12    3 16 26 37 41 52 63 65 91 79 85 22
61      12    3 16 26 37 41 56 60 91 66 79 85 22
62      12    3 16 26 40 41 52 63 67 92 77 87 22
63      12    4 9 21 32 34 47 51 62 65 94 79 84
64      12    4 9 21 32 34 47 51 62 70 76 89 87
65      12    4 9 21 32 34 48 59 90 53 70 87 75
66      12    4 9 21 32 38 43 58 65 55 94 79 84
67      12    4 9 23 38 48 26 61 80 93 51 70 82
68      12    4 9 23 40 42 59 80 67 92 86 55 29
69      12    4 10 21 32 33 43 53 63 73 70 95 83
70      12    4 10 21 32 38 41 61 65 82 94 51 79
71      12    4 10 21 32 38 41 63 65 94 51 79 84
72      12    4 10 21 32 39 41 51 61 73 95 83 70
73      12    4 10 21 32 40 41 62 76 95 51 70 82
74      12    4 10 21 32 40 41 63 83 94 51 79 68
75      12    4 10 23 38 27 48 49 61 80 93 70 82
76      12    4 14 17 34 48 59 71 29 74 54 84 96
77      12    4 14 17 38 48 50 61 71 88 74 93 27
78      12    4 14 24 26 38 41 61 65 90 51 79 85
79      12    4 14 24 26 38 41 63 65 93 51 76 87
80      12    4 14 24 26 38 41 64 76 93 51 71 82
81      12    4 14 24 26 39 41 51 61 73 93 71 83
82      12    4 14 24 33 46 56 58 76 93 71 27 82
83      12    4 14 24 33 47 50 62 65 93 27 76 87
84      12    4 14 24 33 47 50 62 69 79 81 92 27
85      12    4 14 24 33 47 50 62 71 76 81 93 27
86      12    4 14 24 33 47 58 53 73 93 71 83 27
87      12    4 15 17 27 38 42 61 70 80 81 93 55
88      12    4 15 17 27 39 56 42 61 80 70 93 83
89      12    4 15 17 30 40 42 59 88 67 92 77 55
90      12    4 15 19 32 34 47 49 62 70 76 81 93
91      12    4 15 21 26 38 41 61 65 88 90 51 78
92      12    4 15 21 26 39 41 51 61 73 70 83 96
93      12    4 15 21 27 33 47 58 53 73 70 83 96
94      12    4 15 21 27 33 48 58 88 66 91 53 78
95      12    4 16 17 34 46 51 63 73 71 29 94 84
96      12    4 16 17 34 46 56 59 71 29 94 74 84
97      12    4 16 17 38 48 50 61 93 70 74 27 87
98      12    4 16 19 34 47 49 62 71 73 29 94 84
99      12    4 16 21 26 38 41 61 65 94 51 79 82
100     12    4 16 21 26 38 41 63 65 94 51 79 84
101     12    4 16 21 33 46 50 61 66 81 94 79 27
102     12    4 16 21 33 47 50 62 65 94 79 84 27
103     12    4 16 21 33 47 50 62 70 73 92 27 87
104     12    5 9 20 35 48 50 61 72 30 91 74 87
105     12    5 9 20 39 56 42 59 72 75 92 30 87
106     12    5 9 20 40 42 59 31 80 55 92 86 67
107     12    5 9 20 40 42 59 31 80 90 53 86 67
108     12    5 9 20 40 42 59 72 30 91 53 74 87
109     12    5 9 22 34 48 59 71 88 74 91 53 28
110     12    5 9 24 34 47 59 53 81 94 71 75 28
111     12    5 9 24 34 48 59 90 53 71 86 75 28
112     12    5 9 24 35 47 50 60 73 92 30 67 87
113     12    5 10 20 30 35 48 54 72 74 95 84 57
114     12    5 10 20 30 39 49 43 62 72 73 84 95
115     12    5 10 20 31 35 48 54 80 66 92 86 57
116     12    5 10 20 31 40 49 43 63 80 92 86 67
117     12    5 10 22 25 35 45 56 60 71 88 74 91
118     12    5 10 24 27 38 44 49 61 65 82 94 79
```

```
119     12    5 10 24 30 35 41 54 60 66 81 92 79
120     12    5 10 24 30 35 41 56 60 92 67 79 82
121     12    5 11 17 34 45 55 60 31 80 67 86 89
122     12    5 11 17 34 45 55 60 72 81 75 30 95
123     12    5 11 17 34 45 56 60 72 30 91 74 87
124     12    5 11 17 34 48 60 31 80 66 92 54 86
125     12    5 11 17 34 48 60 72 30 54 74 95 84
126     12    5 11 17 36 42 56 61 72 30 91 74 87
127     12    5 11 17 40 42 63 31 80 92 52 86 67
128     12    5 11 17 40 50 60 31 80 90 86 45 67
129     12    5 11 17 40 50 60 72 30 91 74 45 87
130     12    5 11 24 34 47 28 53 73 94 71 83 57
131     12    5 11 24 34 47 49 60 73 92 30 67 87
132     12    5 15 18 35 41 55 60 67 73 88 92 30
133     12    5 15 20 25 35 45 55 58 65 80 91 86
134     12    5 15 20 25 35 48 58 66 80 54 92 86
135     12    5 15 20 25 35 48 58 72 90 54 84 78
136     12    5 15 20 25 37 42 54 59 65 80 90 86
137     12    5 15 20 25 40 42 59 67 80 55 92 86
138     12    5 15 20 25 40 42 59 67 80 90 53 86
139     12    5 16 18 35 41 55 60 73 92 31 86 67
140     12    5 16 18 38 41 28 61 65 94 51 79 82
141     12    5 16 20 25 35 45 50 64 91 71 74 86
142     12    5 16 20 25 35 46 50 63 73 71 94 84
143     12    5 16 20 25 40 42 59 67 53 95 82 78
144     12    5 16 20 25 40 42 59 91 53 71 74 86
145     12    6 9 23 37 43 56 60 66 80 91 26 85
146     12    6 9 23 40 26 43 60 67 77 88 92 55
147     12    6 9 31 38 44 61 65 80 90 85 50 19
148     12    6 9 31 39 52 42 62 80 81 68 93 19
149     12    6 9 31 40 42 19 62 80 90 52 85 68
150     12    6 10 25 35 45 56 60 67 80 23 93 82
151     12    6 10 25 35 45 56 60 72 23 91 74 85
152     12    6 10 25 35 48 60 66 77 88 92 23 54
153     12    6 10 25 35 48 60 68 74 88 54 93 23
154     12    6 10 25 35 48 60 72 23 54 92 77 82
155     12    6 10 32 33 44 54 59 65 76 21 95 82
156     12    6 10 32 33 44 54 59 65 90 79 84 21
157     12    6 11 17 31 39 52 42 80 64 68 93 83
158     12    6 11 17 31 40 58 80 90 52 85 46 68
159     12    6 11 17 32 40 50 60 90 45 67 77 87
160     12    6 11 21 32 33 44 50 63 65 91 76 87
161     12    6 11 21 32 33 47 58 65 91 52 76 87
162     12    6 11 23 26 37 47 52 65 80 91 85 57
163     12    6 11 23 26 40 49 60 67 77 45 96 82
164     12    6 11 23 26 40 49 60 72 91 74 85 45
165     12    6 11 23 33 48 58 72 54 92 77 82 28
166     12    6 12 17 31 38 43 58 65 88 55 93 76
167     12    6 12 17 31 39 54 58 73 88 92 43 69
168     12    6 12 17 31 39 56 42 59 88 92 75 69
169     12    6 12 17 32 39 54 42 59 84 79 89 69
170     12    6 12 17 32 39 54 58 73 69 95 84 43
171     12    6 12 18 31 39 49 43 62 73 88 92 69
172     12    6 12 18 32 39 49 43 62 73 69 95 84
173     12    6 12 23 25 35 45 50 64 72 91 74 85
174     12    6 12 23 25 35 46 50 63 72 77 81 92
175     12    6 12 23 25 35 48 58 66 88 54 92 77
176     12    6 12 23 25 35 48 58 72 54 92 77 82
177     12    6 12 23 25 39 54 42 59 72 81 92 77
178     12    6 12 23 25 40 42 59 67 82 53 77 96
179     12    6 12 23 25 40 42 59 67 88 92 77 55
180     12    6 12 23 25 40 42 59 72 91 53 74 85
181     12    7 9 19 32 39 50 61 73 93 83 44 70
```

182	12	7	10	20	25	40	45	62	72	82	94	51	77
183	12	7	10	20	30	33	48	59	72	54	92	77	82
184	12	7	10	22	27	33	44	56	61	80	66	91	85
185	12	7	10	22	27	33	47	62	65	76	88	93	52
186	12	7	11	17	30	34	45	56	60	72	91	74	85
187	12	7	11	17	30	34	48	60	72	54	92	77	82
188	12	7	11	17	30	40	50	60	72	91	74	85	45
189	12	7	11	17	30	40	58	72	46	92	52	77	82
190	12	7	11	17	32	40	50	60	69	91	74	86	45
191	12	7	11	22	26	37	41	52	63	65	80	91	85
192	12	7	11	22	26	37	41	56	60	80	66	91	85
193	12	7	12	17	29	34	46	51	63	72	73	84	94
194	12	7	12	17	29	34	46	56	59	72	74	94	84
195	12	7	12	17	29	34	48	59	72	90	54	84	78
196	12	7	12	17	32	34	46	56	59	76	93	70	82
197	12	7	12	18	32	38	41	64	76	93	51	70	82
198	12	7	12	18	32	39	41	51	61	73	93	83	70
199	12	7	13	18	32	33	44	55	59	65	91	86	76
200	12	7	13	19	25	36	48	58	80	66	91	53	86
201	12	7	13	19	25	40	50	60	80	90	86	45	67
202	12	8	10	27	33	44	56	61	66	83	94	79	21
203	12	8	10	27	33	47	62	65	21	94	52	79	84
204	12	8	11	17	34	48	60	30	54	92	66	77	87
205	12	8	11	17	40	42	63	67	92	52	30	77	87
206	12	8	12	18	39	41	51	62	73	92	31	86	69
207	12	8	13	18	33	48	59	74	91	53	71	86	28

Appendix C

C# implementation

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace Scriptie
{
    class Program
    {
        private static readonly string problem = "";
        private static readonly string incr = "";
        private static readonly string location = Environment.GetFolderPath(Environment.SpecialFolder.Desktop)
        private static readonly string readfile = problem + ".txt";
        private static readonly string writefile = "solution" + incr + ".txt";
        private static readonly bool directed = false;
        private static readonly int index = 1;

        static void Main(string[] args)
        {
            List<Graph> graphs = new List<Graph>();
            GenerateHexGraph(graphs, 2, 6);
            //GenerateThreeChess(graphs);
            //ReadGraphs(graphs);

            /*
            for (int i = 0; i < graphs.Count; i++)
            {
                Graph graph = graphs[i];
                //  PrintMatrix(graph, i != 0);
                graph.Complement(toList: false);
                Stopwatch stopwatch = Stopwatch.StartNew();
                int clique = graph.GfG();
                stopwatch.Stop();
                Console.WriteLine("Size: {0} - Runtime: {1}ms", clique, stopwatch.ElapsedMilliseconds);
            }
            /*/
            MaximumClique(graphs);
            //*/

            //Console.ReadKey();
        }

        private static void GenerateHexGraph(List<Graph> graphs, int n) => GenerateHexGraph(graphs, n, n);
        private static void GenerateHexGraph(List<Graph> graphs, int min, int max)
        {
            if (min < 2)
```

```
49              min = 2;
50          for (int n = min; n <= max; n++)
51          {
52              Stopwatch stopwatch = Stopwatch.StartNew();
53              // Hexagon vertices:  n=1: 1, n=2: 7, n=3: 19, n=4: 37, n=5: 61,
54                  n=6: 91, n=7: 127, n=8: 169, n=9: 217, n=10: 271
55              int dim = n * 2 - 1 + index;
56              int size = 3 * n * (n - 1) + 1;
57              graphs.Add(new Graph(size, index));
58              Graph graph = graphs[n - min];
59
60              // Create board (as vertices)
61              int?[,] hex = new int?[dim, dim];
62              int number = 1;
63              for (int i = index; number <= size; i++)
64              {
65                  // start each diagonal at the top row and from the left
66                  int row = i;
67                  int col = index;
68
69                  while (row >= index)
70                  {
71                      if (Math.Abs(row - col) < n && row < dim && col < dim)
72                          hex[row, col] = number++;
73                      row--;
74                      col++;
75                  }
76              }
77
78              // down: hex[row+1, col]     up right:  hex[row-1, col+1]     left diag: hex[row+2, col+1]
79              // right: hex[row, col+1]    down right: hex[row+1, col+1]    right diag: hex[row+1, col+2]
80
81              // Create all possible movements (as edges)
82              int u, v, ver, hor;
83              for (int row = index; row < dim; row++)
84                  for (int col = index; col < dim; col++)
85                  {
86                      try
87                      {
88                          u = hex[row, col].Value;
89                      }
90                      catch (Exception)
91                      {
92                          continue;
93                      }
94                      // down
95                      ver = row + 1;
96                      while (ver < dim)
97                      {
98                          try
99                          {
100                             v = hex[ver++, col].Value;
101                         }
102                         catch (Exception)
103                         {
104                             break;
105                         }
106                         if (u != v)
107                             graph.Add(u, v);
108                     }
109                     // right
110                     hor = col + 1;
111                     while (hor < dim)
```

```
112                     {
113                         try
114                         {
115                             v = hex[row, hor++].Value;
116                         }
117                         catch (Exception)
118                         {
119                             break;
120                         }
121                         if (u != v)
122                             graph.Add(u, v);
123                     }
124                     // up right
125                     ver = row - 1;
126                     hor = col + 1;
127                     while (ver >= index && hor < dim)
128                     {
129                         try
130                         {
131                             v = hex[ver--, hor++].Value;
132                         }
133                         catch (Exception)
134                         {
135                             break;
136                         }
137                         if (u != v)
138                             graph.Add(u, v);
139                     }
140                     // down right
141                     ver = row + 1;
142                     hor = col + 1;
143                     while (hor < dim && ver < dim)
144                     {
145                         try
146                         {
147                             v = hex[ver++, hor++].Value;
148                         }
149                         catch (Exception)
150                         {
151                             break;
152                         }
153                         if (u != v)
154                             graph.Add(u, v);
155                     }
156                     // diagonal left
157                     ver = row;
158                     hor = col;
159                     while (hor < dim && ver < dim)
160                     {
161                         try
162                         {
163                             ver += 2;
164                             hor++;
165                             v = hex[ver, hor].Value;
166                         }
167                         catch (Exception)
168                         {
169                             break;
170                         }
171                         if (u != v)
172                             graph.Add(u, v);
173                     }
174                     // diagonal right
```

```
175                     ver = row;
176                     hor = col;
177                     while (hor < dim && ver < dim)
178                     {
179                         try
180                         {
181                             ver++;
182                             hor += 2;
183                             v = hex[ver, hor].Value;
184                         }
185                         catch (Exception)
186                         {
187                             break;
188                         }
189                         if (u != v)
190                             graph.Add(u, v);
191                     }
192                 }
193             stopwatch.Stop();
194             Console.WriteLine(stopwatch.ElapsedMilliseconds);
195         }
196     }
197
198     private static void GenerateThreeChess(List<Graph> graphs)
199     {
200         // Three-man chess: 96
201         int size = 96;
202         graphs.Add(new Graph(size, index));
203         Graph graph = graphs[0];
204
205         // horizontal
206         for (int u = 1; u < size; u++)
207             if (u % 8 != 0)
208                 for (int v = u + 1; (v - 1) % 8 != 0; v++)
209                     graph.Add(u, v);
210
211         // vertical
212         for (int u = 1; u < 89; u++)
213         {
214             if (u >= 57 && u <= 60)
215                 continue;
216
217             int v;
218             switch (u)
219             {
220                 case int n when n >= 29 && n <= 32:
221                     v = u + 40;
222                     break;
223                 case 61:
224                     v = 68;
225                     break;
226                 case 62:
227                     v = 67;
228                     break;
229                 case 63:
230                     v = 66;
231                     break;
232                 case 64:
233                     v = 65;
234                     break;
235                 default:
236                     v = u + 8;
237                     break;
```

```
238                 }
239
240                 while (true)
241                 {
242                     graph.Add(u, v);
243                     if (v >= 57 && v <= 60 || v >= 89)
244                         break;
245                     switch (v)
246                     {
247                         case int n when n >= 29 && n <= 32:
248                             v += 40;
249                             graph.Add(u, v);
250                             break;
251                         case 61:
252                             v = 68;
253                             graph.Add(u, v);
254                             break;
255                         case 62:
256                             v = 67;
257                             graph.Add(u, v);
258                             break;
259                         case 63:
260                             v = 66;
261                             graph.Add(u, v);
262                             break;
263                         case 64:
264                             v = 65;
265                             graph.Add(u, v);
266                             break;
267                     }
268                     v += 8;
269                 }
270             }
271
272             // diagonal right
273             List<int> dr = new List<int>() { 8, 16, 24, 32, 72, 80, 88 };
274             for (int u = 1; u < 88; u++)
275             {
276                 if (dr.Contains(u) || (u >= 57 && u <= 63))
277                     continue;
278
279                 int v;
280                 switch (u)
281                 {
282                     case int n when n >= 29 && n <= 31:
283                         v = u + 41;
284                         break;
285                     case 40:
286                         v = 66;
287                         break;
288                     case 48:
289                         v = 67;
290                         break;
291                     case 56:
292                         v = 68;
293                         break;
294                     case 64:
295                         v = 69;
296                         break;
297                     default:
298                         v = u + 9;
299                         break;
300                 }
```

```
301
302                 while (!dr.Contains(v - 9))
303                 {
304                     graph.Add(u, v);
305                     if ((v >= 57 && v <= 63) || v >= 89)
306                         break;
307
308                     switch (v)
309                     {
310                         case int n when n >= 29 && n <= 31:
311                             v += 41;
312                             graph.Add(u, v);
313                             break;
314                         case 40:
315                             v = 66;
316                             graph.Add(u, v);
317                             break;
318                         case 48:
319                             v = 67;
320                             graph.Add(u, v);
321                             break;
322                         case 56:
323                             v = 68;
324                             graph.Add(u, v);
325                             break;
326                         case 64:
327                             v = 69;
328                             graph.Add(u, v);
329                             break;
330                     }
331                     v += 9;
332                 }
333             }
334
335             // diagonal left
336             List<int> dl = new List<int>() { 9, 17, 25, 33, 41, 49, 73, 81, 89 };
337             for (int u = 2; u < 89; u++)
338             {
339                 if (dl.Contains(u) || (u >= 61 && u <= 65))
340                     continue;
341
342                 int v;
343                 switch (u)
344                 {
345                     case int n when n >= 30 && n <= 32:
346                         v = u + 39;
347                         break;
348                     case 57:
349                         v = 68;
350                         break;
351                     case 58:
352                         v = 67;
353                         break;
354                     case 59:
355                         v = 66;
356                         break;
357                     case 60:
358                         v = 65;
359                         break;
360                     default:
361                         v = u + 7;
362                         break;
363                 }
```

```
364
365                 while (!dl.Contains(v - 7))
366                 {
367                     graph.Add(u, v);
368                     if ((v >= 61 && v <= 65) || v >= 89)
369                         break;
370
371                     switch (v)
372                     {
373                         case int n when n >= 30 && n <= 32:
374                             v += 39;
375                             graph.Add(u, v);
376                             break;
377                         case 57:
378                             v = 68;
379                             graph.Add(u, v);
380                             break;
381                         case 58:
382                             v = 67;
383                             graph.Add(u, v);
384                             break;
385                         case 59:
386                             v = 66;
387                             graph.Add(u, v);
388                             break;
389                         case 60:
390                             v = 65;
391                             graph.Add(u, v);
392                             break;
393                     }
394                     if (v == 65)
395                         break;
396                     v += 7;
397                 }
398             }
399         }
400
401         private static void MaximumClique(List<Graph> graphs)
402         {
403             Stopwatch stopwatch = new Stopwatch();
404             TimeSpan timeSpan = new TimeSpan();
405
406             using (StreamWriter writer = new StreamWriter(Path.Combine(location, writefile), true))
407             {
408                 for (int i = 0; i < graphs.Count; i++)
409                 {
410                     Console.WriteLine("Graph {0}", i);
411                     Graph graph = graphs[i];
412                     int vertices = graph.V.Count - index;
413                     double n = (3 + Math.Sqrt(12 * vertices - 3)) / 6;
414                     writer.WriteLine("n = {0} (Vertices: {1}, Edges: {2})", n, vertices,
415                         graph.GetNrEdges(directed, true));
416
417                     //PrintMatrix(graph, i != 0);
418
419                     // Calculate the maximum cliques on the complement of the target graph
420                     graph.Complement(toList: true);
421
422                     // Maximal cliques
423                     stopwatch.Start();
424                     List<HashSet<Vertex>> cliques = graph.BronKerbosch();
425                     int aantalCliques = cliques.Count;
426
```

```
427                 // Maximum cliques
428                 int max = cliques.Max(x => x.Count); // Size of maximum clique
429                 cliques = cliques.FindAll(x => x.Count == max); // Maximum cliques
430                 int aantalMaxCliques = cliques.Count;
431
432                 // Print runtime of finding the maximum cliques
433                 stopwatch.Stop();
434                 timeSpan = stopwatch.Elapsed;
435                 string days = string.Format(" ({0:%d} days, {0:%h} hours)", timeSpan);
436                 writer.WriteLine("Running time: {0:00}:{1:mm\\:ss\\.fff}{2}",
437                           (timeSpan.Days * 24) + timeSpan.Hours,
438                           timeSpan,
439                           timeSpan.Days >= 1 ? days : "");
440                 stopwatch.Reset();
441
442                 // Print per graph the maximum cliques and its vertices
443                 writer.WriteLine("Maximal cliques: {0}, Maximum cliques: {1} solutions\r\n",
444                           aantalCliques, aantalMaxCliques);
445                 writer.WriteLine("{0,-7} {1,-5} {2,-8}", "Clique", "Size", "Vertices");
446                 for (int j = 0; j < aantalMaxCliques; j++)
447                 {
448                     HashSet<Vertex> clique = cliques[j];
449                     int size = clique.Count;
450
451                     int[] tempList = new int[size];
452                     for (int k = 0; k < size; k++)
453                     {
454                         Vertex v = clique.ElementAt(k);
455                         tempList[k] = v.Id;
456                     }
457
458                     StringBuilder builder = new StringBuilder();
459                     foreach (int vertex in tempList)
460                         builder.Append(vertex).Append(" ");
461                     builder.Length--;
462                     string result = builder.ToString();
463
464                     writer.WriteLine("{0,-7} {1,-5} {2,-8}", j, max, result);
465                 }
466                 writer.WriteLine(string.Concat(Enumerable.Repeat("-", 40)));
467                 writer.Flush();
468             }
469         }
470     }
471
472     private static void PrintMatrix(Graph graph, bool append)
473     {
474         if (graph.AdjM == null)
475             graph.ConvertToMatrix();
476
477         using (StreamWriter writer = new StreamWriter(Path.Combine(location, "matrix.txt"), append))
478         {
479             writer.Write("\r\n  ");
480             for (int col = index; col < graph.V.Count; col++)
481                 writer.Write("{1}{0} ", col, col < 10 ? " " : "");
482             writer.WriteLine("\r\n" + string.Concat(Enumerable.Repeat("-", 3 * graph.V.Count)));
483             for (int j = index; j < graph.V.Count; j++)
484             {
485                 writer.Write("{1}{0} |", j, j < 10 ? " " : "");
486                 for (int k = index; k < graph.V.Count; k++)
487                     writer.Write(graph.AdjM[j, k] == true ? " 1 " : " 0 ");
488                 writer.WriteLine();
489             }
```

```
490                 writer.WriteLine();
491             }
492         }
493 
494         private static void ReadGraphs(List<Graph> graphs)
495         {
496             int g, v, e;
497             using (StreamReader reader = new StreamReader(Path.Combine(location, readfile)))
498             {
499                 string line = reader.ReadLine();
500                 if (!line.Contains(' '))
501                 {
502                     g = int.Parse(line); // graphs
503                     reader.ReadLine();
504                 }
505                 else
506                 {
507                     g = 1;
508                     reader.DiscardBufferedData();
509                     reader.BaseStream.Seek(0, SeekOrigin.Begin);
510                 }
511 
512                 for (int i = 0; i < g; i++)
513                 {
514                     string[] dimensions = reader.ReadLine().Split(' ');
515                     v = int.Parse(dimensions[0]);
516                     e = int.Parse(dimensions[1]);
517 
518                     graphs.Add(new Graph(v, index));
519 
520                     for (int j = 0; j < e; j++)
521                     {
522                         string[] input = reader.ReadLine().Split(' ');
523                         graphs[i].Add(int.Parse(input[0]),
524                             int.Parse(input[1]),
525                             input.Length == 3 ? int.Parse(input[2]) : 0,
526                             directed);
527                     }
528                     reader.ReadLine();
529                 }
530             }
531         }
532     }
533 
534     class Graph
535     {
536         private int K { get; set; } // number of vertices
537         private int Index { get; }
538         public List<Vertex>[] Adj; // Adjacency List
539         public List<Vertex> V;
540         public List<Edge> E;
541         public bool[,] AdjM; // Adjacency Matrix
542         private List<HashSet<Vertex>> cliques; // Set of cliques (used in BronKerbosch)
543         private int[] temp; // Temporary clique (used in GfG)
544 
545         public Graph()
546         {
547             Adj = new List<Vertex>[1];
548             Adj[0] = new List<Vertex>();
549         }
550 
551         public Graph(int n, int index)
552         {
```

```
553             n += index;
554             K = n;
555             Index = index;
556             Adj = new List<Vertex>[n];
557             for (int i = 0; i < n; i++)
558                 Adj[i] = new List<Vertex>();
559             V = new List<Vertex>(new Vertex[K]);
560             E = new List<Edge>();
561         }
562 
563         public void Add(int u, int v, int weight = 0, bool directed = false)
564         {
565             // if size is unknown
566             if (K == 0)
567             {
568                 int size = Math.Max(u, v) + 1; // +1 als index1
569                 if (Adj.Length <= size)
570                 {
571                     Array.Resize(ref Adj, size);
572                     for (int i = 0; i < Adj.Length; i++)
573                         if (Adj[i] == null)
574                             Adj[i] = new List<Vertex>();
575                 }
576             }
577 
578             Adj[u].Add(new Vertex(v, weight));
579             if (!directed)
580                 Adj[v].Add(new Vertex(u, weight));
581             else
582                 E.Add(new Edge(u, v, weight));
583 
584             if (V[u] == null)
585                 V[u] = new Vertex() { Id = u };
586             if (V[v] == null)
587                 V[v] = new Vertex() { Id = v };
588         }
589 
590         private int GetDegree(int v) => Adj[v].Count();
591         public int GetNrEdges(bool directed = false, bool adjlist = true)
592         {
593             int teller = 0;
594             if (adjlist)
595             {
596                 foreach (List<Vertex> edge in Adj)
597                     teller += edge.Count;
598             }
599             else
600             {
601                 foreach (bool item in AdjM)
602                     if (item == true)
603                         teller++;
604             }
605             return directed ? teller : teller / 2;
606         }
607 
608         /* Adjacency Matrix representation */
609         public void AdjacencyMatrix()
610         {
611             AdjM = new bool[K, K];
612         }
613         public void AdjacencyMatrix(int n)
614         {
615             K = n;
```

```
616             AdjM = new bool[K, K];
617         }
618
619         public void AddEdge(int u, int v)
620         {
621             AdjM[u, v] = true;
622         }
623         public void RemoveEdge(int u, int v)
624         {
625             AdjM[u, v] = false;
626         }
627         private bool HasEdge(int u, int v) => AdjM[u, v];
628
629         public void Complement(bool toList)
630         {
631             if (AdjM == null)
632                 ConvertToMatrix();
633
634             for (int u = Index; u < K; u++)
635                 for (int v = Index; v < K; v++)
636                     if (u != v)
637                         AdjM[u, v] = !AdjM[u, v];
638
639             if (toList)
640                 ConvertToList();
641         }
642
643         public void ConvertToMatrix()
644         {
645             if (AdjM == null)
646                 AdjacencyMatrix();
647
648             for (int u = Index; u < K; u++)
649                 foreach (Vertex v in Adj[u])
650                     AdjM[u, v.Id] = true;
651         }
652         public void ConvertToList()
653         {
654             for (int i = 0; i < K; i++)
655                 Adj[i] = new List<Vertex>();
656
657             for (int u = Index; u < K; u++)
658                 for (int v = Index; v < K; v++)
659                     if (u != v && AdjM[u, v])
660                         Adj[u].Add(new Vertex() { Id = v });
661         }
662         /* Adjacency Matrix representation */
663
664         /* Algorithms */
665         public List<HashSet<Vertex>> BronKerbosch()
666         {
667             List<Vertex> R = new List<Vertex>();
668             List<Vertex> P = new List<Vertex>(V);
669             List<Vertex> X = new List<Vertex>();
670             cliques = new List<HashSet<Vertex>>();
671             BronKerbosch(R, P, X);
672             return cliques;
673         }
674         private void BronKerbosch(List<Vertex> R, List<Vertex> P, List<Vertex> X)
675         {
676             if (P.Count == 0 && X.Count == 0)
677             {
678                 if (cliques.Count == 0 || R.Count >= cliques.Max(x => x.Count))
```

```
679                 cliques.Add(new HashSet<Vertex>(R));
680             return;
681         }
682 
683         // Choose pivot
684         Vertex u = GetMaxDegree(P.Union(X).ToList());
685 
686         // PminNu = P \ N(u)
687         List<Vertex> PminNu = new List<Vertex>(P);
688         foreach (Vertex neighbour in Adj[u.Id])
689             PminNu.Remove(neighbour);
690 
691         foreach (Vertex v in PminNu)
692             if (v != null)
693             {
694                 R.Add(v);
695                 BronKerbosch(R, P.Intersect(Adj[v.Id]).ToList(), X.Intersect(Adj[v.Id]).ToList());
696                 R.Remove(v);
697                 P.Remove(v);
698                 X.Add(v);
699             }
700     }
701     private Vertex GetMaxDegree(List<Vertex> union)
702     {
703         int count, temp = 0;
704         Vertex u = union.Last();
705         for (int i = 0; i < union.Count(); i++)
706             if (union[i] != null)
707             {
708                 count = GetDegree(union[i].Id);
709                 if (count > temp)
710                 {
711                     u = V[union[i].Id];
712                     temp = count;
713                 }
714             }
715         return u;
716     }
717 
718     public int GfG()
719     {
720         temp = new int[GetNrEdges() / K];
721         return GfGMaximumClique(0, 1);
722     }
723     private int GfGMaximumClique(int i, int k)
724     {
725         int maxCliqueSize = 0;
726 
727         // Check if any vertices from i+1 can be inserted
728         for (int j = i + 1; j < K; j++)
729         {
730             // Add the vertex to temporary clique
731             temp[k] = j;
732 
733             // If the graph is not a clique, then it cannot be a clique by adding another edge
734             if (IsClique(k + 1))
735             {
736                 maxCliqueSize = Math.Max(maxCliqueSize, k);
737                 // Check if another edge can be added
738                 maxCliqueSize = Math.Max(maxCliqueSize, GfGMaximumClique(j, k + 1));
739             }
740         }
741         return maxCliqueSize;
```

```
742         }
743         private bool IsClique(int size)
744         {
745             for (int i = 1; i < size; i++)
746                 for (int j = i + 1; j < size; j++)
747                     if (!AdjM[temp[i], temp[j]]) // If any edge is missing
748                         return false;
749             return true;
750         }
751     }
752
753     public class Vertex
754     {
755         public int Id { get; set; }
756         public int Key { get; set; }
757         public int Parent { get; set; }
758         //public int Colour { get; set; }
759         //public int CC { get; set; }
760         //public int D { get; set; } // DFS: discovered, all else: distance
761         //public int Finished { get; set; }
762
763         public Vertex()
764         {
765
766         }
767
768         public Vertex(int id, int key)
769         {
770             Id = id;
771             Key = key;
772         }
773
774         public override int GetHashCode()
775         {
776             return Id;
777         }
778
779         public override bool Equals(object obj)
780         {
781             if (obj is Vertex vertex)
782                 return vertex.Id == Id;
783             return false;
784         }
785     }
786
787     public class Edge
788     {
789         public int U { get; set; }
790         public int V { get; set; }
791         public int Weight { get; set; }
792
793         public Edge(int u, int v, int weight)
794         {
795             U = u;
796             V = v;
797             Weight = weight;
798         }
799     }
800 }
```