CS 517 - Final Project

Tieqiao Wang
Oregon State University
wangtie@oregonstate.edu

Liqiang He
Oregon State University
heli@oregonstate.edu

1 Introduction

In this work, we will address the Hamiltonian Cycle Problem (HCP) using a SAT solver. The problem is defined as: given a graph $G = (V, E)$, check whether the graph G has a Hamiltonian-cycle.

A Hamiltonian Cycle is a graph cycle through a graph that visits each vertex exactly once [9]. This problem was proposed by Sir William Rowan Hamilton in 1895 to find a route along the polyhedron edges of of regular dodecahedron that would pass once and only once through every point [8]. To find a Hamiltonian cycle is a NP-complete problem[4, 5], so the only known way to determine whether a given general graph has a Hamiltonian cycle is to undertake an exhaustive search.

This problem has wide range application in computer science. For example, in computer graphics, we can generate dual graph first and then hamiltonian triangulations can be used for fast rendering [2].

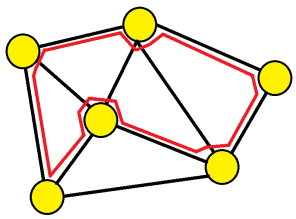

Figure 1: A Hamiltonian cycle around a network of six vertices [1].

2 Method

In this section, we describe how we encode instances of HCP in SAT [6].

2.1 Formulation

Suppose a graph G has n vertices, the index $i, j \in \{1, 2, ..., n\}$. A Hamiltonian cycle can be expressed as a permutation π where

- $\pi(i) = j$ means the vertex j is in the i th position
- $(\pi(i\%n), \pi((i+1)\%n) \in G$ indicates all edges in the cycle are valid

2.2 HCP$\leq_p$SAT

Our goal is to construct a conjunctive normal form (CNF) from our graph G such that the SAT solver accepts if and only if when there exists at least one Hamiltonian cycle in G.

2.2.1 Variables

CNF(G) has n^2 variables represented by $x_{i,j(1\leq i,j\leq n)}$. If the i th position in the cycle is occupied by the j the vertex, then $x_{i,j} = 1$.

2.2.2 Clauses

We can construct 5 kind of clauses based on the definition of HCP.

1. Each vertex j must appear at least once. $O(n^2)$ clauses.
   $\forall j, x_{1j} \vee x_{2j} \vee ... \vee x_{nj} = 1$
2. Each vertex j must appear at most once. $O(n^3)$ clauses.
   $\forall i, j, k, \bar{x}_{ij} \vee \bar{x}_{kj} = 1$
3. Each position i must be occupied by at least one vertex. $O(n^2)$ clauses.
   $\forall i, x_{i1} \vee x_{i2} \vee ... \vee x_{in} = 1$
4. Each position i must be occupied by at most one vertex. $O(n^3)$ clauses.
   $\forall i, j, k, \bar{x}_{ij} \vee \bar{x}_{ik} = 1$
5. Nonadjacent vertex i and j cannot be adjacent in the cycle. $O(n^3)$ clauses.
   $\forall i, j, k, \bar{x}_{k\%n,i} \vee \bar{x}_{(k+1)\%n,j} = 1$

Hence, CNF(G) contains $O(n^3)$ clauses.

2.3 Tricks

We find a method (two product encoding [3]) to optimize at-most-one (AMO) constraint. Let $Y = y_1, y_2, ..., y_n$. The standard SAT encoding of the AMO constraint is the following.

$$AMO(Y) \equiv \{\bar{y}_i \vee \bar{y}_j | y_i, y_j \in Y, i < j\} \tag{1}$$

which requires $O(n^2)$ clauses. Based on the idea of Cartesian products, this method only needs $2n + 4\sqrt{n} + O(\sqrt[4]{n})$ clauses but it also introduces $2n + \sqrt{n} + O(\sqrt[4]{n})$ auxiliary variables.
The big picture is to represent one dimension constraints with two dimensions, i.e., use a grid with n points, one for each variable, and to constrain at most one row and at most one column to be selected, and the points at their intersection is the only variable allowed to be true. In details, we break down n into two parts: p and q. Let $p = \lceil\sqrt{n}\rceil$,$q = \lceil n/p\rceil$ and make one point x_k in one dimension coordinate correspond to one point $< u_i, v_i >$ in two dimension coordinate. After applying a one-to-one mapping, AMO encoding can be recursively defined as

$$AMO(X) \equiv AMO(U) \wedge AMO(V) \bigwedge_{i\leq i\leq p, 1\leq j\leq q}^{1\leq k\leq n, k=(i-1)q+j} ((\bar{x_k} \vee u_i) \wedge (\bar{x_k} \vee v_j)) \tag{2}$$

The dominant term is the last term which has $2n$ clauses and by recursively optimizing first two AMOs we only need $2n + 4\sqrt{n} + O(\sqrt[4]{n})$ to represent the initial at most one constraint in the end.

3 Experiment

3.1 Dataset

3.1.1 Input and Output

The input is an edge list of the graph.
The output is 0 or 1 indicating whether the graph has a Hamiltonian Cycle.

3.1.2 Data Source

We evaluate our solver using the data from Flinders Hamiltonian Cycle Project (FHCP). This dataset [7] was developed to support research activities relating to the Museums artifacts and administration of the collections, which includes Hamiltonian-cycle and Non-Hamiltonian-cycle graphs with the number of vertices ranging from 4 to 20. Table 1 shows the statistic of the database. For different number of vertices, we have various number of Hamiltonian and Non-Hamiltonian graphs.

Table 1: FHCP graph database used in our experiments.

vertex	Hamiltonian	Non-Hamiltonian
4	1	0
6	2	0
8	5	0
10	17	2
12	80	5
14	474	35
16	3841	219
18	39635	1666
20	0	14498

V	Emin	Emax	Dmin	Dmax	Cmin	Cmax	Tmin	Tmax	Tmean	Tstd
10	15	15	3	3	1520	1520	0.0269	0.1031	0.0650	0.0381
12	18	18	3	3	2760	2760	0.0148	0.0514	0.0238	0.0139
14	21	21	3	3	4536	4536	0.0114	0.0995	0.0303	0.0227
16	24	24	3	3	6944	6944	0.0178	0.2885	0.0629	0.0455
18	27	27	3	3	10080	10080	0.0191	0.7868	0.1107	0.1038
20	30	30	3	3	14040	14040	0.1291	11.2032	2.0187	1.3184

Table 2: Non Hamiltonian cycle graph results. We record the minimal (min), maximum (max), mean (mean) and standard deviation (std) of edge (E), degree of free (D) and time (T) based on different number of vertex (V).

3.2 SAT Solver

We use the SAT solver provided in the **pysat** library to conduct experiments. PySAT integrates several SAT solver (such as CaDiCaL, Glucose, Mergesat and Minisat) and also a number of cardinality encodings (for example, pairwise, bitwise and totalize).

3.3 Setting

We carry out the performance evaluation of solving Hamiltonian Cycle Problem in terms of runtime (seconds) on EECS pelican2 server: Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz and 256GB memory. Several factors that closely related to the graph complexity are considered here: the number of the vertices, the number of edges, and the maximum degree of freedom. But this dataset does not provide a variety of combination variety and hence we propose several augmentation strategies to tackle this drawback.

4 Data Augmentation

As the dataset size is relatively small, we designed some strategies to expand the input dimension:

V	Emin	Emax	Dmin	Dmax	Cmin	Cmax	Tmin	Tmax	Tmean	Tstd
6	9	9	3	3	264	264	0.0004	0.0005	0.0005	0.0000
8	12	12	3	3	720	720	0.0022	0.0025	0.0024	0.0001
10	15	15	3	3	1520	1520	0.0208	0.0483	0.0300	0.0074
12	18	18	3	3	2760	2760	0.0275	0.0763	0.0512	0.0125
14	21	21	3	3	4536	4536	0.0067	0.0164	0.0071	0.0009
16	24	24	3	3	6944	6944	0.0094	0.0297	0.0100	0.0013
18	27	27	3	3	10080	10080	0.0164	0.8551	0.0446	0.0124

Table 3: Hamiltonian cycle graph results. We record the minimal (min), maximum (max), mean (mean) and standard deviation (std) of edge (E), degree of free (D) and time (T) based on different number of vertex (V).

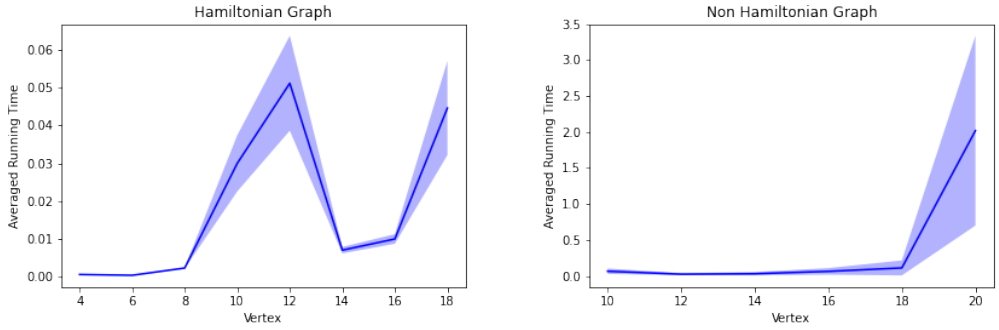

Figure 2: The relationship between the number of vertices and average running time.

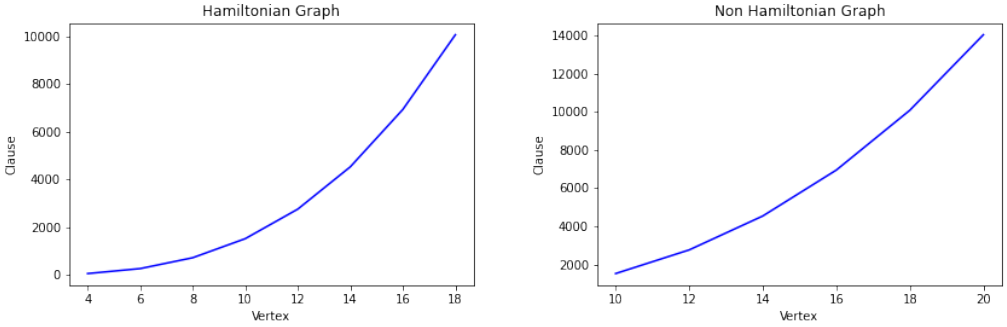

Figure 3: The relationship between the number of vertices and clauses.

1. Non Hamiltonian Cycle Graph.
   Nondeterministically select 2 graphs (G_1, G_2) from our original data set with n_1 and n_2 number of vertices. Then, randomly add $n_1 - 1$ and $n_2 - 1$ edges for these two sub-graphs. Finally, randomly select one vertex (v) from G_1 and two vertices (u_1,u_2) from G_2 and add two edges $< v, u_1 >$ and $< v, u_2 >$.
2. Hamiltonian Cycle Graph.
   Nondeterministically select 2 graphs (G_1, G_2) from our original data set with n_1 and n_2 number of vertices. Then, add a Hamiltonian cycle with a random vertex order.
3. Random Graph.
   Create a graph G_h with n vertices. Then, we randomly generate some edges and add to G_h. At this point, whether the generated graph is a Hamiltonian or Non Hamiltonian cycle is unknown, we can used the same trick to make it Hamiltonian by adding one additional cycle or Non Hamiltonian cycle graph by attaching one extra vertex and an edge. Since previous experiments have shown the correctness of our algorithm, some randomly graphs can be used to obtain an upper bound on the solver's problem-solving ability.

5 Ablation Study

5.1 Vertices, Edges and Clauses

Vertices, edges and clauses are three closely relative factors. Table 2 and 3 illustrates the results where we record the minimal (min), maximum (max), mean (mean) and standard deviation (std) of edge (E), degree of free (D) and time (T) based on different number of vertex (V). From Fig.2-3, when we fixed the degree of freedom and the edge number equals the number of vertices, it usually takes more time for the SAT solver to find the results, especially for Non-Hamiltonian cases. And the number of clauses grows as the number of vertex increases.
Giving the vertex dimension in the initial dataset is relatively small, we create new graphs with 22 to 34 vertices by combining original ones (Table 4). Fig.4-5 illustrates new experimental results. We

number of vertices in G_1	12	12	12	14	14	16	16
number of vertices in G_2	10	12	14	14	16	16	18
number of vertices in G	22	24	26	28	30	32	34

Table 4: We generate G with specific combinations of G_1 and G_2.

can conclude that the clause number grows very fast as the number of vertex increases. And it takes more time for non Hamiltonian cycle graphs than their Hamiltonian counterparts. This is probably because for the former, the solver need to check all possibilities before making a conclusion. While for the latter, the solver will immediately return yes after finding one possible satisfiable assignment. But these results may be derived from our initial graph constraints such as degree of freedom equals three or our construction algorithm. And hence we provide two more ablation studies to explore the dominant runtime factor in the following sections.

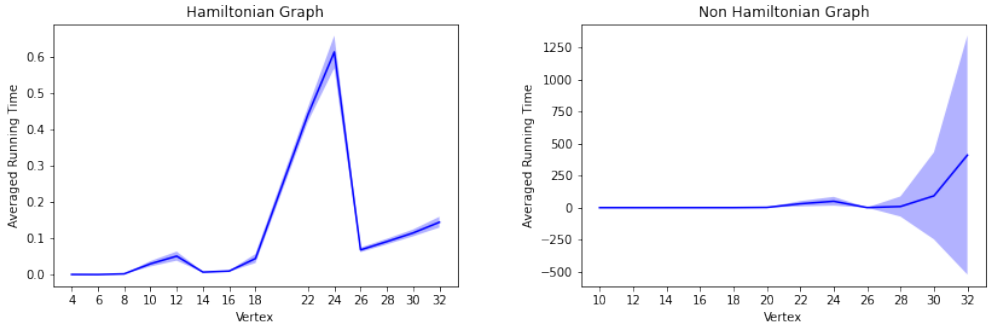

Figure 4: The relationship between the number of vertices and average running time. We provide the averaged running time (the blue line) along with the standard deviation (the light blue region) for graphs with different vertex number.

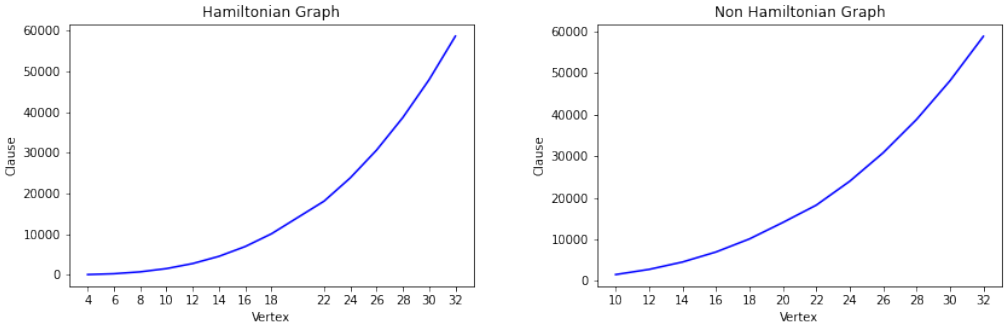

Figure 5: The relationship between the number of vertices and clauses.

5.2 Degree of Freedom

We find that the degree of freedom of the graphs in the FHCP dataset is fixed to 3, which means that each vertex is only connected to 3 other vertices in the graph. In Section 4, we added edges between vertices in the data augmentation, however, the overall degree of freedom does not change too much. To explore the impact of degree of freedom, we generate graphs based on 20-vertex non-Hamiltonian Cycle graphs in FHCP using following procedures:

1. First, select the 1666 graphs from the 18-vertex NH subset in FHCP.
2. Then, add edges between vertices. The average degree of freedom is controlled to be in the range between 4 and 11.
3. Finally, generate new graphs depending on what we need:

- Hamiltonian Cycle Graph.
  Add a Hamiltonian Cycle in above generated graphs with arbitrary topology.
- Non Hamiltonian Cycle Graph.
  Randomly pick up one vertex in the graph and then shrink its degree of freedom to one.

The results are visualized in Fig.6-9. As we can see from Fig.6, the degree of freedom linearly affects the average running of solving HCP in both H graphs and NH graphs. The standard deviation for H graphs is much higher than NH graphs. Because the running time of the previous one is related to how easy the solver to find the solution, while for the latter, the solver has to go through all possible assignments. From time to time, there are large fluctuations in the running time, which indicates some outliers (extremely easier or harder cases) are generate by our algorithm (even though the graph has the same average degree of freedom).
In Fig.7, the number of edges is also linearly contributing to the running time, but it is more flatten comparing to the curves in Fig.6. This is because the number of edges are constrained while we fixed the number of vertices.
In Fig.9, when we increase the average degree of freedom in the graph, the number of clauses linearly decreases in both H and NH graphs. This is due to the additional relationships (edges) added between vertices in the graph. Based on the fifth construction rule in Sec.2.3, the clauses number decreases when the graph becomes denser (the average degree of freedom increases). Therefore, in Fig.8, we can find that the average running time increases while number of clauses decreases (more edges in the graph). But these running time different is trivial compared to the counterpart in previous and following section, indicating the degree of freedom does not significantly affect the running time.

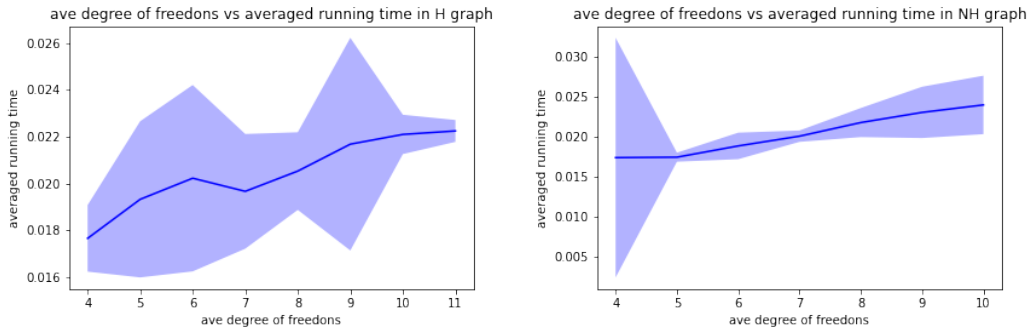

Figure 6: The relationship between the degree of freedom and the average running time. The light blue region represents the standard deviation.

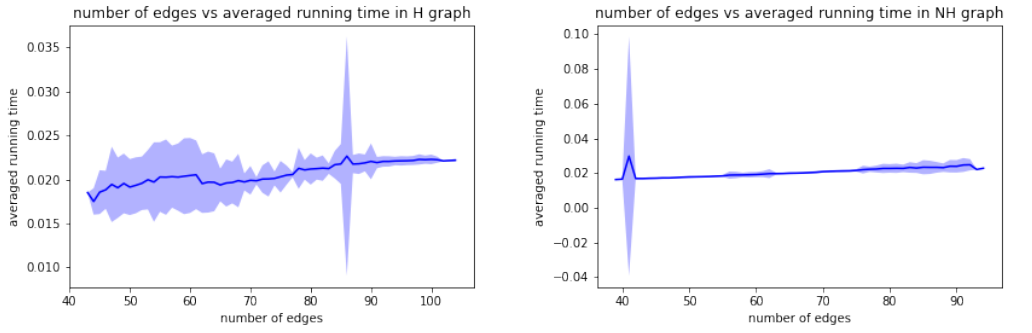

Figure 7: The relationship between the number of edges and the average running time. The light blue region represents the standard deviation.

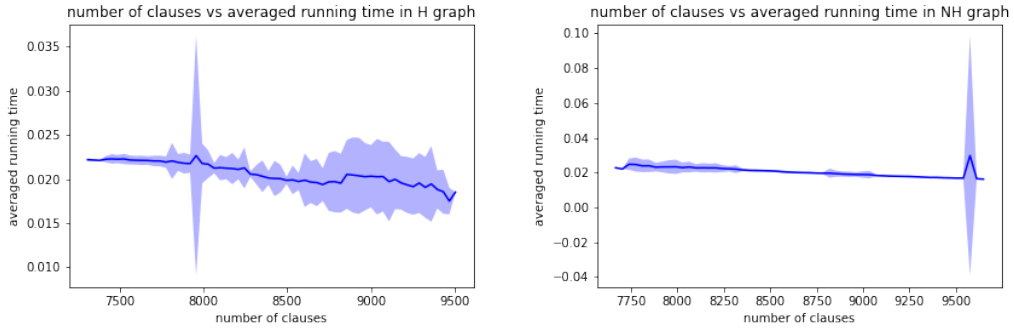

Figure 8: The relationship between the number of clauses and the average running time. The light blue region represents the standard deviation.

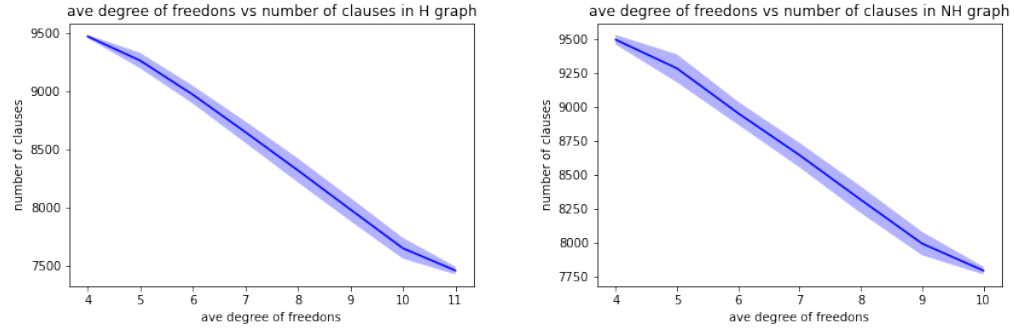

Figure 9: The relationship between the degree of freedom and the number of clauses.The light blue region represents the standard deviation.

5.3 Feasible and Infeasible Input Size

In real-world application of HCP, the graph size could be very large. In this section, we explore the impact of graph size to the running time of the solver for the HCP.

We randomly generate graphs with the vertex number ranging from 50 to 300. And the number of edges are roughly set to 15% of the total possible number of edges (for the fully connected graph). As we can see from Tab 5-6, the number of edges and the degree of freedom linearly increase as the increase of the vertex number. However, the number of clauses and the running time increase exponentially. While the number of vertex increases to 300, both Hamiltonian Cycle graph and Non-Hamiltonian graph cost more than 2 hours to solve. Based on the exponential relationship to the number of vertex in the graph, we think the boundary between feasible and infeasible input graph size is between 300 and 400 (the number of vertices in the graph).

6 Conclusion

We identify a very classic and fascinating NP-hard problem Hamiltonian Cycle Problem (HCP) with a wide range of applications in our final project and succeed in building a tool that solves instances of this problem using **pysat** solver. And we also find some encoding strategies to reduce the clause size. Moreover, we clearly describe how can instances of this problem be encoded in SAT and design a comprehensive ablation study to discover what attributes of the graph may affect the runtime. Among the vertex, edge, degree of freedom and clause, the vertices number is the dominant one. Finally, we dramatically increase the vertex number and create a huge graph to find the boundary of infeasible input size. We also make our code (https://github.com/tqosu/CS-517-Final) available on GitHub to benefit others in our research area.

V	E	D	C	Time
50	228	9.12	222300	1.589
100	829	16.58	1814400	45.163
200	3160	31.60	14656400	1126.590
300	6982	46.54	49631400	8602.258

Table 5: Hamiltonian cycle graph results. We report the number of edges (E), the average degree of freedom (D), the number of clauses (C), and the running time (T) w.r.t. the number of vertex (V) in the graph.

V	E	D	C	Time
50	216	8.64	223500	1.491
100	813	16.26	1817600	41.838
200	3132	31.32	14667600	1119.537
300	6938	46.25	49657800	8499.865

Table 6: Non-Hamiltonian cycle graph results. We report the number of edges (E), the average degree of freedom (D), the number of clauses (C), and the running time (T) w.r.t. the number of vertex (V) in the graph.

References

[1] https://en.wikipedia.org/wiki/hamiltonian_path.

[2] E. M. Arkin, M. Held, J. S. Mitchell, and S. S. Skiena. Hamiltonian triangulations for fast rendering. *The Visual Computer*, 12(9):429–444, 1996.

[3] J. Chen. A new sat encoding of the at-most-one constraint. *Proc. Constraint Modelling and Reformulation*, 2010.

[4] J. Hartmanis. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, 24(1):90, 1982.

[5] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[6] Y.-D. Lyuu. https://www.csie.ntu.edu.tw/ lyuu/complexity/2011/20111018.pdf.

[7] M. Meringer. Fast generation of regular graphs and construction of cages. *Journal of Graph Theory*, 30(2):137–146, 1999.

[8] S. Skiena. *Implementing discrete mathematics: combinatorics and graph theory with Mathematica*. Addison-Wesley Longman Publishing Co., Inc., 1991.

[9] E. W. Weisstein. Hamiltonian cycle. *https://mathworld. wolfram. com/*, 2003.