We denote the minimum number of queens required to dominate all squares of a chessboard of size $n \times n$ as $\gamma(Q_n)$. we may sometimes refer to this as $\gamma(n)$ for convenience. If n is understood from the context, we may simply use γ.

A theorem states that $\gamma(n) \geq \left\lceil \frac{(n-1)}{2} \right\rceil$. Moreover, a fundamental fact about this problem is that $\gamma(n) = \gamma(n-1) + (0 \; or \; 1)$. Armed with these two facts, the optimization variant of the problem can often be reduced to determining whether $\left\lceil \frac{(n-1)}{2} \right\rceil$ or $\left\lceil \frac{(n-1)}{2} \right\rceil$ + 1.

In the formulas/sat_unsat/cnf folder, CNF formulas are generated for certain values of n. While the method described above would be used in a real-world scenario to solve an open case, the primary goal of this part of the work is to evaluate the performance of different SAT solvers with respect to different encodings. Consequently, we sometimes approximate the search process as follows:

If we know the optimal value of queens for a given n (i.e., it has been solved by other researchers and verified by a SAT solver), we reverse the approach described above. For instance, if we know $\gamma(n)$ is k, we first encode the problem to verify $\gamma(n) = k$ and pass it to the SAT solver, where we expect a SAT result. We then encode $\gamma(n) = k - 1$, which we know is UNSAT, but this still serves as a useful benchmark to monitor SAT solvers' performance on UNSAT instances of the problem.

In addition to simply passing CNF formulas to SAT solvers, we also assess whether using the well-known technique of Cube-and-Conquer with the tool March can improve CPU time. For this purpose, we generated and stored cubes of formulas available in formulas/sat_unsat/cnf within formulas/sat_unsat/icnf. The cubes stored here are the result of calling March once with its default parameters.

Another scenario of interest arises when, after determining the correct value of $\gamma(n)$, we also want to count the number of solutions of that size. CNF formulas for this scenario have also been generated for some values of n and are stored in the formulas/model_enum folder.

All the code discussed is located in the code/ folder. Within that folder, you'll find several Python files. Each file contains a brief, high-level description at the top, outlining the purpose of the code.

Below is a summary of the results.

SAT instances without cube and conquer

n	Gamma	At Most	At Least	Finding a Satisfying Assignment (Seconds)		Model Enumeration (Seconds)	
				CaDiCaL	MapleSAT	CaDiCaL	MapleSAT
10	5	✓		**0.8281**	6.2031	**16.6093**	49.5000
10	5	✓	✓	**2.6875**	10.0546	**11.8593**	45.5937
11	5	✓		**0.1718**	9.9687	**15.6875**	95.2343
11	5	✓	✓	**14.875**	20.7968	**19.1406**	116.4062
12	6	✓		**29.3281**	51.4062	**823.0468**	Time Out
12	6	✓	✓	48.1562	**17.6562**	**728.2031**	6688.2500
13	7	✓		**34.2500**	37.2187	Time Out	Time Out
13	7	✓	✓	115.7656	**43.1406**	Time Out	Time Out
14	8	✓		165.6406	**79.5937**	Time Out	Time Out
14	8	✓	✓	727.3125	**40.2812**	Time Out	Time Out
15	9	✓		**18.4062**	183.0937	Time Out	Time Out
15	9	✓	✓	35.5937	**34.8593**	Time Out	Time Out

UNSAT instances without cube and conquer

n	Gamma	At Most	At Least	Time (Seconds)		UNSAT Proof (Kilobyte)	
				CaDiCaL	MapleSAT	CaDiCaL	MapleSAT
10	4	✓		**1.2656**	3.4375	11,829	**10,583**
10	4	✓	✓	**1.0156**	3.4843	**8,629**	11,526
11	4	✓		**2.0000**	6.3125	**20,401**	21,240
11	4	✓	✓	**1.4687**	6.2187	**13,340**	21,239
12	5	✓		**38.7500**	213.9375	**316,788**	1,093,642
12	5	✓	✓	**34.7187**	150.4218	**227,103**	691,765
13	6	✓		**1662.4062**	Time out	7,366,352	N/A
13	6	✓	✓	**1910.9375**	Time out	7,398,906	N/A

With cube and conquer

n	γ	Inc.	At Most	At Least	Cube Generation Time (Seconds)	Time (Seconds)		UNSAT Proof (Kilobyte)	
						CaDiCaL	MapleSAT	CaDiCaL	MapleSAT
10	4		✓		0.0700	1.0468+0.0156	**0.9687+0.0000**	8,372+**1**	**6,090**+**1**
10	4	✓	✓		0.0700	**0.8594**+0.0156	1.6250+**0.0000**	158,714+**1**	**122,391**+**1**
10	4		✓	✓	0.0500	**1.0625+0.0000**	1.5468+**0.0000**	9,728+**1**	**7,897**+**1**
10	4	✓	✓	✓	0.0500	**0.6562+0.0000**	2.9219+**0.0000**	**25,749+1**	115,662+**1**
11	4		✓		0.1100	1.7656+0.0156	**1.5625+0.0000**	14,262+**1**	**9,506**+**1**
11	4	✓	✓		0.1100	**1.1719**+0.0156	2.8750+**0.0000**	**334,859+1**	335,428+**1**
11	4		✓	✓	0.1100	**1.7187+0.0000**	1.8906+ 0.0156	15,343+**1**	**11,161**+**1**
11	4	✓	✓	✓	0.1100	**1.0938+0.0000**	2.8750+ 0.0156	**76,884**+**1**	88,213+**1**
12	5		✓		0.3600	**33.3593+0.0000**	95.4531+**0.0000**	**308,236+1**	387,960+**1**
12	5	✓	✓		0.3600	**33.0312+0.0000**	63.1250+**0.0000**	**7,420,959+1**	9,937,242+**1**
12	5		✓	✓	0.3700	**36,2031+0.0000**	91.8593+**0.0000**	**336,617+9**	381,587+**1**
12	5	✓	✓	✓	0.3700	**36.2344+0.0000**	54.0938+**0.0000**	**3,904,861+9**	5,685,134+**1**
13	6		✓		4.7200	**905.8438**+0.0312	5225.6406+**0.0156**	**7,075,706**+184	23,428,492+**89**
13	6	✓	✓		4.7200	TimeOut+0.0312	Time Out+**0.0156**	N/A+184	N/A+**89**
13	6		✓	✓	0.7600	**962.5156**+**0.0000**	2583.3750+0.0156	**7,464,258**+**17**	16,894,558+**17**
13	6	✓	✓	✓	0.7600	TimeOut+**0.0000**	Time Out+0.0156	N/A+**17**	N/A+**17**
14	7		✓		2.1800	9219.5312+0.0312	+**0.0156**	46,633,413+185	+**5**
14	7	✓	✓		2.1800	+0.0312	+**0.0156**	+185	+**5**
14	7		✓	✓	2.2500	+0.0156	+**0.0000**	+60	+**53**
14	7	✓	✓	✓	2.2500	+0.0156	+**0.0000**	+60	+**53**
15	8		✓		28.1500	45,053.5487+0.1718	+**0.0937**	92,243,508+2,919	+**468**
15	8	✓	✓		28.1500	+0.1718	+**0.0937**	+2,919	+**468**
15	8		✓	✓	3.5000	+ **0.0156**	+**0.0156**	+124	**+121**
15	8	✓	✓	✓	3.5000	+ **0.0156**	+**0.0156**	+124	**+121**