MCAA : Mini-project
Solving the N-queens problems via the Metropolis algorithm

Jean-Baptiste Michel, Orfeas Liossatos, Gaiëtan Renault

Nov. 2022

1 Introduction

The N-Queens problem consists of placing N queens on an $N \times N$ chessboard such that no two queens threaten each other. In this mini-project, we adapt the Metropolis algorithm to find a solution for $N = 1000$, and apply an extension to the algorithm to estimate the number of solutions for $N = 100$ to 1000.
Firstly, our results show that the Metropolis algorithm presents no particular advantage over a fully greedy algorithm in the number of iterations before convergence. Secondly, we show that the estimator for the number of solutions is biased and has a large variance, suggesting absurdly that the number of solutions decreases absolutely for N large enough. We end each section with a discussion of our results.

2 Finding a solution

We adapt the Metropolis algorithm to find a solution to the N-Queens problem. First of all we show a possible solution to N queens problem for $N = 10$ in Fig. 1

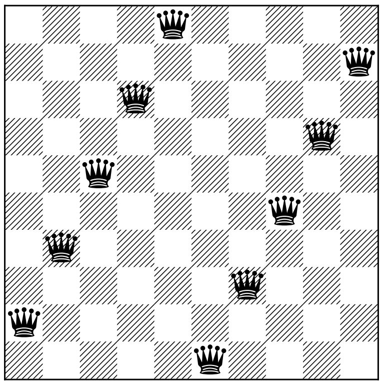

Figure 1: A possible solution to N-queens problem for N = 10

2.1 State space S

Let P denote the set of all permutations of the set $\{1, ..., N\}$.

We thus have for any $p \in P$:

$$\begin{aligned} p : \{1, ..., N\} &\to \{1, ..., N\} \\ n &\mapsto p(n) \end{aligned}$$

We denote by S the state space representing the configurations of queens with only diagonal threats, if they exist. That means the state space contains all the board configurations with only one queen per column and per line. We observe that there exists a bijection between the state space of diagonally-conflicting queens S and the set of permutations of N integers. Thus $S = P$ and $|S| = N!$.

2.2 Base chain

Let Ψ, represented in Fig. 2, be the base chain on S. Call the exchange of two rows of the chessboard a *swap*, or *2-cycle* in the permutation representation. Let these be the legal moves in the chain (see cycle notation in Fig. 2).

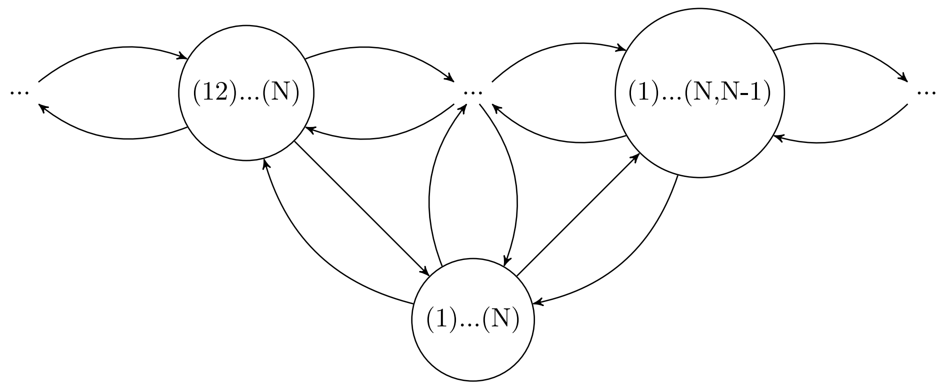

Figure 2: Base chain Ψ

Since there are $\binom{n}{2}$ possible swaps from any configuration, the transition probabilities are the following:

$$\psi_{xy} = \begin{cases} \binom{n}{2}^{-1} & \text{if } d(x,y) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where $d(x,y)$ is the minimum number of swaps required to configure state i into state j. In other words, when $d(x,y) = 1$, the two permutations differ in only 2 places. This chain is

- irreducible, as all configurations are reachable by row-swaps;
- positive-recurrent, as $|S| < \infty$; and
- 2-periodic, as there are no self-loops.

Also, Ψ is such that $\psi_{xy} > 0 \Leftrightarrow \psi_{yx} > 0$ because the chain is symmetric.

2.3 Energy function to optimize

We aim to minimize the number of threats on the chessboard. The permutation representation of a queen configuration $(z_i)_{1 \le i \le N} = p \in S$ is the following:

1. Each queen is numbered according to her column, so $\text{col}(i) = i$.
2. Seen as a permutation, z maps $\text{col}(i)$ to $\text{row}(i)$, so $\text{row}(i) = z_i$.

Without loss of generality, set $1 \le i < j \le N$. If two queens i, j are threatening each other on the chessboard, then the following identity holds :

$$\text{col}(j) - \text{col}(i) = |\text{row}(j) - \text{row}(i)|$$

By counting all unordered pairs of queens for which the identity holds, we can know the number of threats. Therefore, the loss of a configuration z is the sum of indicator functions over all unordered pairs of queens

$$\mathcal{L}(z) = \sum_{1 \le i < j \le N} 1\{j - i = z_j - z_i\}.$$

The loss function can be computed in $\mathcal{O}\binom{N}{2}$ operations, which is costly. Thankfully, in practice we need only to compute the difference any given swap makes in the loss function, and then update the current loss with this result. Swapping two queens i, j will only affect the threats they are involved in, which results in the following $\mathcal{O}(N)$ loss update rule

$$\mathcal{L}(z_{t+1}) := \mathcal{L}(z_t) + \Delta\mathcal{L}(z_t),$$

where we define the quantity

$$\Delta\mathcal{L}(z_t) = \#\mathit{Threats}(z_{t+1}, i, j) - \#\mathit{Threats}(z_t, i, j).$$

2.4 Acceptance probabilities and Metropolis chain

In order to apply the Markov chain Monte-Carlo method to our base chain Ψ, we need to define the acceptance probabilities for the Metropolis chain.
Let A be the matrix of acceptance probabilities

$$a_{xy} = \min\{1, \frac{\pi_\beta(y)\psi_{yx}}{\pi_\beta(x)\psi_{xy}}\} = \min\{1, e^{-\beta(\mathcal{L}(y)-\mathcal{L}(x))}\}.$$

From this, we can now compute the transition probabilities P of this new chain.

$$p_{xy} = \begin{cases} \binom{n}{2}^{-1} e^{-\beta(\mathcal{L}(y)-\mathcal{L}(x))} & \text{when } \mathcal{L}(y) > \mathcal{L}(x) \\ \binom{n}{2}^{-1} & \text{when } \mathcal{L}(y) \le \mathcal{L}(x) \end{cases},$$

and for $x = y$,

$$p_{xx} = 1 - \sum_{u:d(x,u)=1} p_{x,u},$$

and 0 otherwise. In practice, however, P is never computed at all, and y is only ever one swap away from x, so a_{xy} is precisely

$$min\{1, e^{-\beta\Delta\mathcal{L}(x)}\}$$

2.5 Inverse temperature β and simulated annealing

There are two ways to pick β. We can use a fixed β, or we can use simulated annealing, making β grow with t, where t is the current iteration of the Metropolis algorithm. The intuition is that the closer β is to 0, the closer the Metropolis chain will be to the uniform base chain, exploring the state space S a lot. As the inverse temperature grows, the Metropolis chain will become pickier, accepting only swaps that decrease the loss function.

2.5.1 Fixed β

We run the algorithm with several fixed values of β and plot its efficiency below. For small values of β ($\beta = 1, 2, 3, 4, 6$), the algorithm did not converge to a solution in under 10 million iterations. This is expected, as a small β means almost uniform random queen swaps. For larger values of β, a solution was found in a desirable time as seen in Table 1 and Fig. 3.

β	Avg. Iterations	
	N=100	N=1000
7	14910	245100
8	23440	291290
9	21760	309240
10	28090	111170
100	28070	144530
1000	26780	111090

Table 1: Average iterations until convergence for fixed β over 30 trials. Snapshots of the loss were taken every 10 iterations, thus the unit digit is always 0.

2.5.2 Simulated annealing

Now in Table 2 we let β vary as a function of time (in iterations). Our intuition originally led us to believe that a slow-cooling scheme would be most efficient, where $\beta(t)$ stays low for a while, allowing the Metropolis chain to explore the state space S, then grows to find states that will minimize $\mathcal{L}(z)$. In fact, the opposite is true. When β grows quickly, the average number of iterations before convergence drops rapidly. This is consistent with the results from Table 1.

$\beta(t)$	Avg. Iterations	
	N=100	N=1000
$\exp(t)$	10810	-
$\log(t)$	13160	230340
t	13000	157250
t^2	17250	167020
$\log(t^2)$	17910	210060

Table 2: Average iterations until convergence for N=1000 and β as a function of iterations t over 30 trials. We could not evaluate $\exp(t)$ for very large values of t. Snapshots of the loss were taken every 10 iterations, thus the unit digit is always 0.

2.6 Results

Our C implementation with a fixed random number generator seed yields the following time-loss plots.

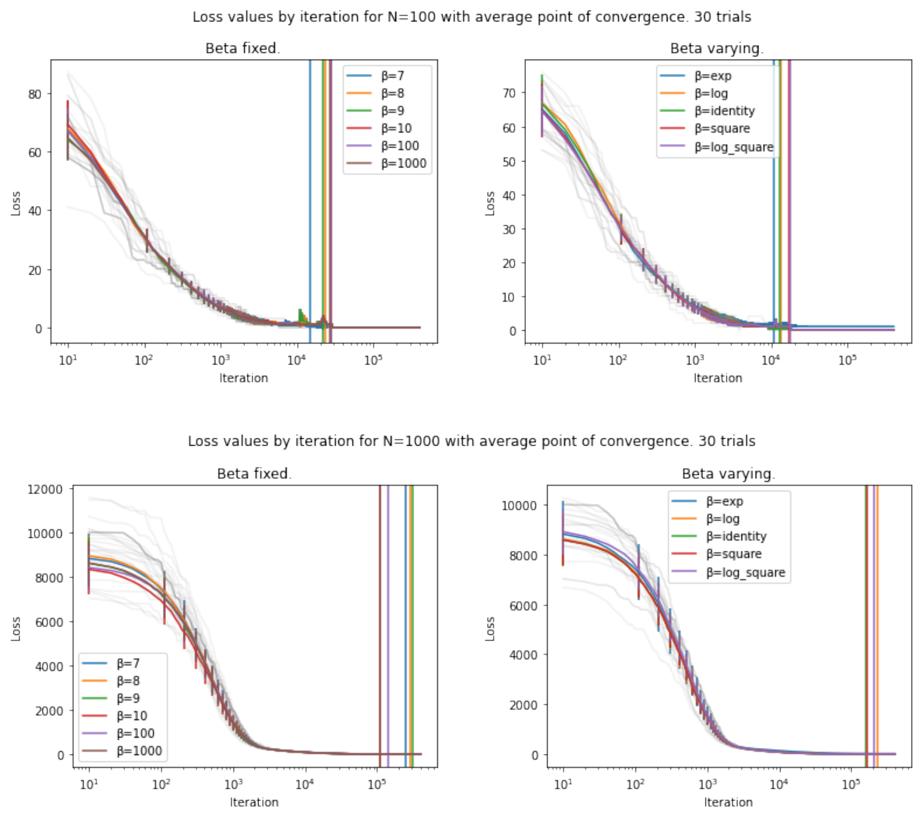

Figure 3: Loss evolutions for N=100 and N=1000.

When β is taken to be $\exp(t)$, we find a solution for $N = 100$ in 10810 iterations on average. As for $N = 1000$, $\beta = 1000$ allows for 111090 iterations on average.

2.7 Discussion

It appears to be that the algorithm is more efficient when using very large β. In opposition to our intuition it looks like the greedier algorithm is more efficient than the one that explores the solution space. We hypothesize that every configuration has a minimal sequence of less than $\binom{n}{2}$ swaps that directly lead to a solution. If this is true, a completely naive greedy algorithm could run faster than the Metropolis algorithm.

3 Estimating the number of solutions

3.1 Choosing the parameters

Let $Z_\beta = \sum_{x \in S} e^{-\beta \mathcal{L}(x)}$ be the partition function. Then Z_∞ denotes the number of solutions to the N-queens problem, and $Z_0 = N!$ is the number of chessboard configurations for a given value of N. Our goal is to find an efficient estimator of Z_∞ / Z_0, which we can multiply by $N!$ to estimate Z_∞. Simply drawing random samples and computing the proportion of valid configurations results in an unbiased, but very inefficient estimator of Z_∞ / Z_0. However, if for a given β^* we expand the fraction

$$\log(Z_{\beta^*}/Z_0) = \sum_{t=0}^{T-1} \log(Z_{\beta_{t+1}}/Z_{\beta_t})$$

Then we notice that for β_t and β_{t+1} close enough, each term of the sum is a reasonably small negative number (in absolute value), and is therefore manageable by a computer. To estimate the ratio $Z_{\beta_{t+1}}/Z_{\beta_t}$, we draw M i.i.d samples $X_1 ... X_M$ according to π_{β_t} and compute

$$\frac{1}{M} \sum_{k=1}^{M} exp(-(\beta_{t+1} - \beta_t) f(X_k))$$

for all values of $t \in \{0, 1, ..., T-1\}$.

3.1.1 Choosing β^*, T

The proposed approach is to fix $M = 1000$ (the number of i.i.d samples we draw) but not T (the number of β_t's with which we'll estimate the ratio) and choose

$$0 < \beta_1 < \beta_2 < ... < \beta_T = \beta^*$$

such that they are separated by a constant $\delta = 0.001$ and to take T large enough to result in estimates for known solution counts (N=4 to N=27) that do not differ more than 0.1% from the ground truth. Then we take $\beta^* = \delta T$ for each value of N for which the *N-Queens Problem* has a known number of solutions and observe their distribution in Fig. 4.

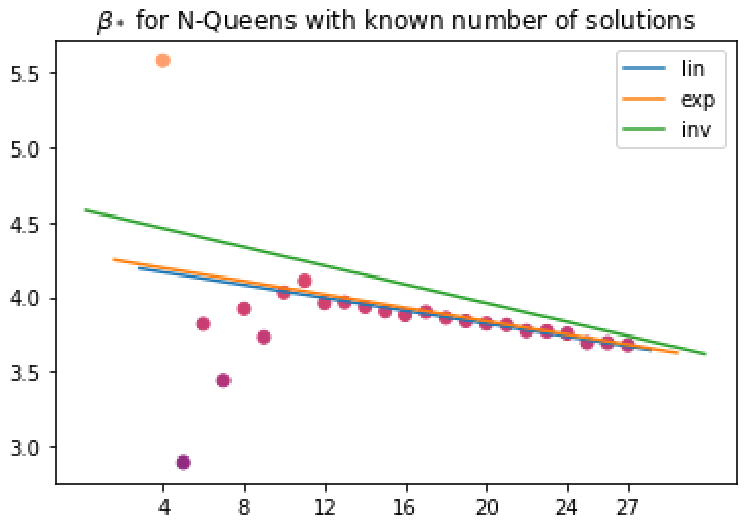

Figure 4: Fitting a linear, exponential, and inverse curve to known β^*. In actuality, we only take the values of β^* for $N \geq 10$ into consideration, since they appear more regular.

We fit various curves to the β^*, and predicted values of β^* for $N = 100$ to $N = 1000$ in increments of 100. Amongst the linear, exponential, and inverse curves, only the exponential curve predicted positive values for all β^* (See Table 3), so we used those values for estimator.

	β^*		
N	$-0.02N + 4.24$	$2.60e^{-0.01N} + 1.69$	$\frac{-0.21}{N^{-0.47}} - 0.47$
100	2.095	2.648	2.838
200	-0.063	2.044	2.131
300	-2.221	1.822	1.596
400	-4.379	1.740	1.151
500	-6.537	1.710	0.761
600	-8.695	1.699	0.411
700	-10.852	1.695	0.090
800	-13.010	1.694	-0.207
900	-15.168	1.693	-0.485
1000	-17.326	1.693	-0.746

Table 3: Predicted values of β^* for large values of N.

3.2 Graph of $log(Z_{\beta_t})$ as a function of β_t

We plot in Fig.5 the function f, for different values of N, where

$$f : t \to \log(Z_{\beta_t}).$$

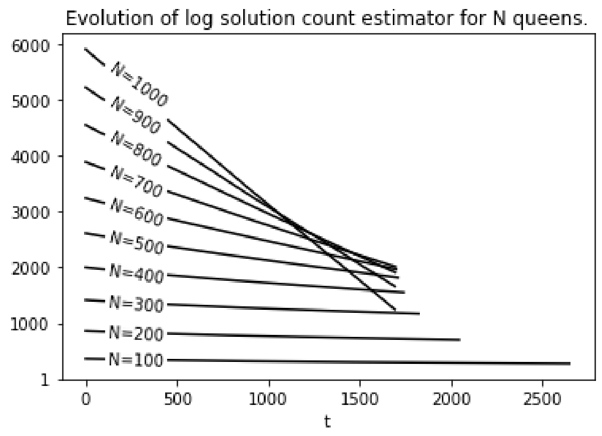

Figure 5: Evolution of a single run the estimator of the log solution count to the N-Queens problem. The value $\log(Z_{\beta_t})$ was computed from the decomposition

$$\log(Z_{\beta_t}) = \log(N!) + \sum_{\tau=1}^{t} \log(\frac{Z_{\beta_\tau}}{Z_{\beta_{\tau-1}}}).$$

This plot reveals how the estimator works. Each term $Z_{\beta_\tau}/Z_{\beta_{\tau-1}}$ is a negative number slightly smaller than 1, and they are almost all the same thanks to δ small. In selecting the best T given a fixed δ for N small, $Z_{\beta_\tau}/Z_{\beta_{\tau-1}}$ correctly counts the proportion of valid solutions in the time granted by M. However, as N grows larger, $Z_{\beta_\tau}/Z_{\beta_{\tau-1}}$ consistently underestimates the proportion of valid solutions. So much so that when t is too large, we reach the absurd estimate that there are less solutions for $N = 1000$ than for $N = 400$ on average (Table 4, column 2).

3.3 Variance of the estimator

In addition, we plot many trials of the estimator in Fig. 6 and compute the standard deviation of f for N = 1000. We observe that as t grows, small errors add up, and estimated solution counts diverge drastically. To make matters worse, the variance of the estimator increases when N grows large, as seen in Fig. 7 and Table 4.

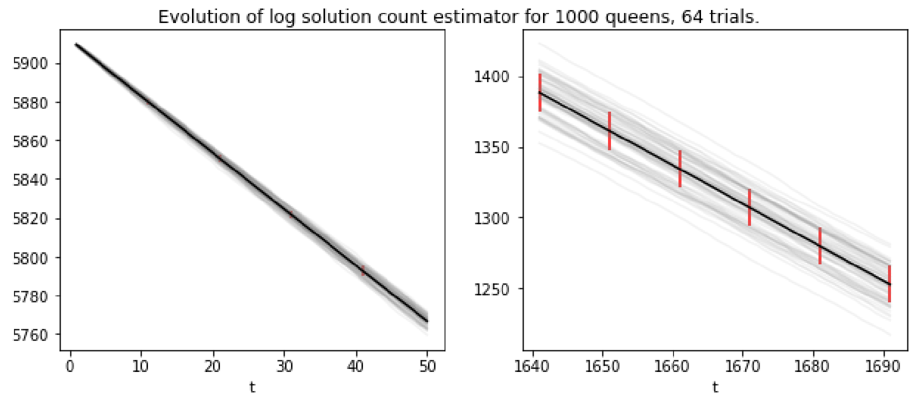

Figure 6: Divergence of solution count for the 1000-Queens problem, over 64 trials.

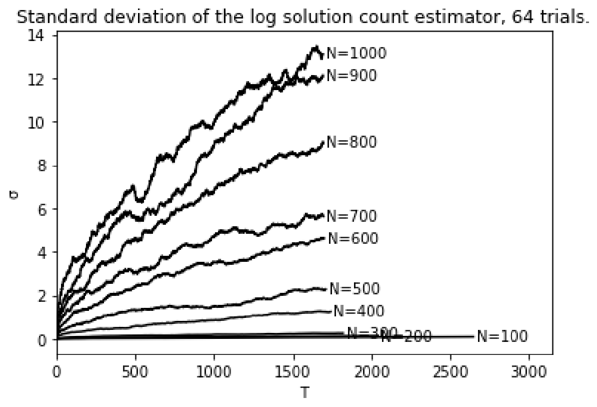

Figure 7: Standard deviation of solution count for the N-Queens problem, over 64 trials.

N	μ	σ
100	276.299	0.097
200	703.048	0.137
300	1170.833	0.258
400	1551.526	1.253
500	1818.873	2.277
600	1970.830	4.631
700	2003.705	5.632
800	1905.808	9.037
900	1664.776	12.117
1000	1252.723	13.110

Table 4: Empirical means μ and standard deviations σ of the estimator of f for large values of N, over 64 trials.

3.4 Result

According to our estimator, for $N = 1000$, we have $\mu = 1252.723$. That would mean that they are $exp(1252.723)$ solutions to the *1000-Queens* problem, which is in the order of 10^{544}. According to most recent research (See [1] or [2]) a good estimator could be $(0.143 * N)^N$, giving an estimation of 10^{2155} solutions for $N = 1000$, thus very far from our estimator.

4 Discussion

Our approach of fixing $M = 1000$ and $\delta = 0.001$, and finding T to compute β^* that fit known solutions well has a drawback in that the resulting β^* for N large (See Table 3) are much smaller than those suggested in the section of quickly finding a single solution to the N-Queens problem. If instead we had fixed $M = 1000$ and $\beta^* = 1000$, and searched for a good-enough δ, perhaps our results for the mean of the estimator would have been more realistic, although this does not address the problem of the variance, which comes from the iterative stacking of errors.

5 Project Github

The code for generating solutions to the N-Queens problem and the graphs can be found in our Github Repository.
`https://github.com/xMissingno/N-Queens-MCMC`

References

[1] M. Simkin, “The number of n-queens configurations,” 2021.

[2] OEIS Foundation Inc., “The On-Line Encyclopedia of Integer Sequences : A000170 - Number of ways of placing n nonattacking queens on an n X n board.,” 2022. Published electronically at `http://oeis.org`.