Calculating the 3-D Kings Multiplicity Constant: Configurations of Non-Attacking Kings in 3-D

Neil Calkin, Nick Cohen

Clemson University, University of California, Irvine

July 23, 2021

Thanks to...

- Rob Corless
- Neil Calkin
- NSF

How to count configurations of non-attacking kings on a chess board

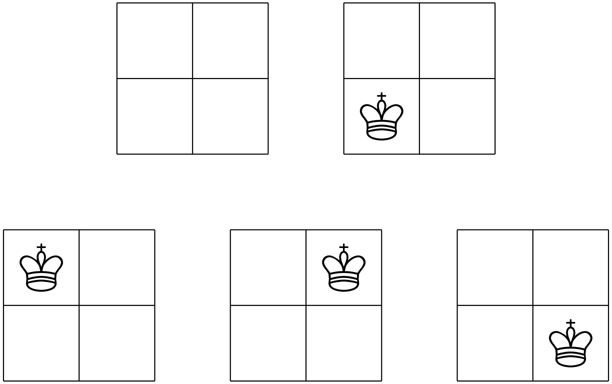


Figure: The five possible 2×2 boards.

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards?

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.
- 2×2 boards?

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.
- 2×2 boards? 5.

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.
- 2×2 boards? 5.
- $2 \times 2 \times 2$ boards?

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.
- 2×2 boards? 5.
- $2 \times 2 \times 2$ boards? 9.

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.
- 2×2 boards? 5.
- $2 \times 2 \times 2$ boards? 9.
- 3×3 boards?

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.
- 2×2 boards? 5.
- $2 \times 2 \times 2$ boards? 9.
- 3×3 boards? 35.

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.
- 2×2 boards? 5.
- $2 \times 2 \times 2$ boards? 9.
- 3×3 boards? 35.
- $3 \times 3 \times 3$ boards?

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.
- 2×2 boards? 5.
- $2 \times 2 \times 2$ boards? 9.
- 3×3 boards? 35.
- $3 \times 3 \times 3$ boards? 2,089.

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.
- 2×2 boards? 5.
- $2 \times 2 \times 2$ boards? 9.
- 3×3 boards? 35.
- $3 \times 3 \times 3$ boards? 2,089.
- $4 \times 4 \times 4$ boards?

How many ways can you place non-attacking kings on a chessboard?

- How many configurations for 1×1 chess boards? 2.
- 2×2 boards? 5.
- $2 \times 2 \times 2$ boards? 9.
- 3×3 boards? 35.
- $3 \times 3 \times 3$ boards? 2,089.
- $4 \times 4 \times 4$ boards? 3,144,692.

In the limit

- One question we can ask is: How does the number of boards increase as we increase the number of squares in the board?

$$\text{multiplicity}_{2D} = \lim_{m,n\to\infty} F(m,n)^{1/mn}$$

In the limit

- One question we can ask is: How does the number of boards increase as we increase the number of squares in the board?

$$\text{multiplicity}_{2D} = \lim_{m,n\to\infty} F(m,n)^{1/mn} \approx 1.3426\ldots$$

In the limit

- One question we can ask is: How does the number of boards increase as we increase the number of squares in the board?

$$\text{multiplicity}_{2D} = \lim_{m,n\to\infty} F(m,n)^{1/mn} \approx 1.3426\ldots$$

- How much information can be stored per square?

$$\text{capacity}_{2D} = \log_2 \text{multiplicity}_{2D} \approx 0.42507\ldots$$

In the limit

- One question we can ask is: How does the number of boards increase as we increase the number of squares in the board?

$$\text{multiplicity}_{2D} = \lim_{m,n\to\infty} F(m,n)^{1/mn} \approx 1.3426\ldots$$

- How much information can be stored per square?

$$\text{capacity}_{2D} = \log_2 \text{multiplicity}_{2D} \approx 0.42507\ldots$$

- What about for three dimensions?

Why is this problem important?

Why is this problem important?

- Statistical mechanics (Entropy)

Why is this problem important?

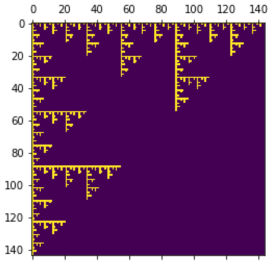
- Statistical mechanics (Entropy)
- Information Theory (Channel Capacity)

Why is this problem important?

- Statistical mechanics (Entropy)
- Information Theory (Channel Capacity)
- Dynamical Systems (Subshifts of Finite Type)

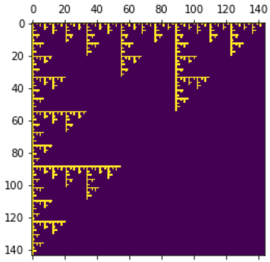
Why is this problem interesting?

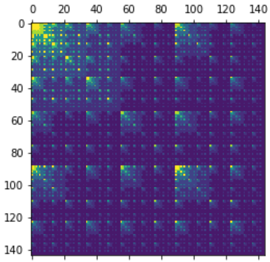
- Look at these beautiful matrices!
- We can solve many types of recurrence relations exactly.
- It feels like an exact solution should be right around the corner.



Why is this problem interesting?

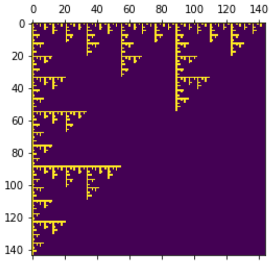
- Look at these beautiful matrices!
- We can solve many types of recurrence relations exactly.
- It feels like an exact solution should be right around the corner.

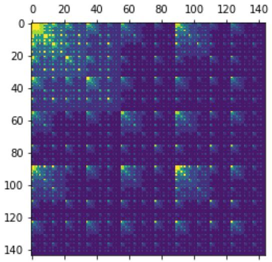


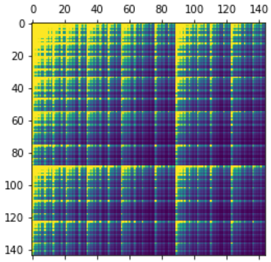


Why is this problem interesting?

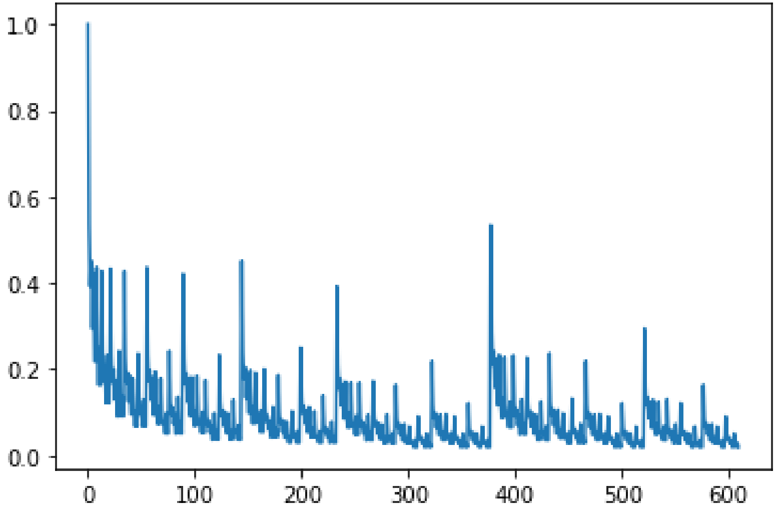
- Look at these beautiful matrices!
- We can solve many types of recurrence relations exactly.
- It feels like an exact solution should be right around the corner.







Look at these beautiful eigenvectors!



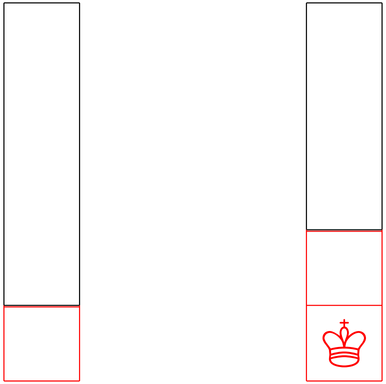
The 2-D problem

The 2-D problem is relatively easy.

1. Identify the 1-D slices
2. Build an adjacency matrix A_k of all possible slices of height k with 1s identifying slices that are allowed to be placed next to each other.
3. Compute $\mathbf{e}_1^T A_k^{n+1} \mathbf{e}_1$ to calculate the number of configurations of kings on a board of dimension $k \times n$.

The 2-D problem: identifying the 1-D slices

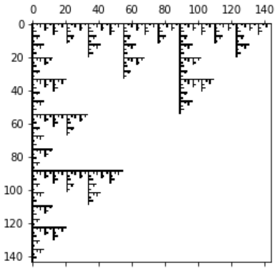
The 1-D slices can be generated using the simple recursive relationship shown below. It is easy to see that the number of 1-D boards are Fibonacci numbers.



The 2-D problem: building an adjacency matrix

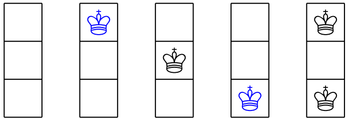
- An adjacency matrix is indexed by the previously shown ordering of 1-D slices and shows us which slices may be adjacent.
- If we index as indicated in the previous slides, it is easy to see that A_{k-1} appears in the top left corner and copies of A_{k-2} appear in the bottom left and top right corners.

$$\begin{bmatrix} A_{k-1} & A_{k-2} \\ A_{k-2} & \mathbf{0} \end{bmatrix}$$



The 2-D problem: building an adjacency matrix

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & \color{blue}{1} & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & \color{blue}{1} & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$



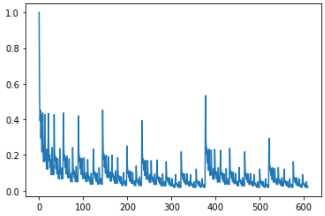
The 2-D Problem: Interpreting the Adjacency Matrix

- It is easy to see (and well known) that the (i, j) entry of A_k^{n+1} counts the number of configurations of n slices sandwiched between slice i and slice j.
- To simplify computation, we can sum the entries of A_k^{n-1} to count the number of $k \times n$ configurations of kings like so:

```
$\mathbf{v} \leftarrow (1, 1, \ldots, 1)$
for $i \leftarrow 1$ to $n - 1$:
        $\mathbf{v} \leftarrow A_k \mathbf{v}$
return $\sum v_i$
```

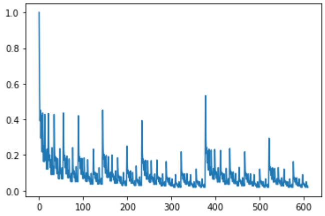
The 2-D Problem: Interpreting the Adjacency Matrix

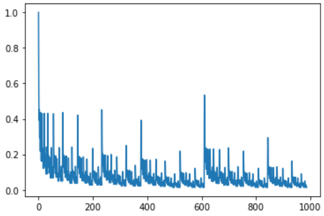
$A^n\mathbf{1}$ approaches a multiple of the Perron eigenvector, and its growth rate, the Perron eigenvalue, tells us what the asymptotic growth rate of the number of boards is as we increase the thickness.



The 2-D Problem: Interpreting the Adjacency Matrix

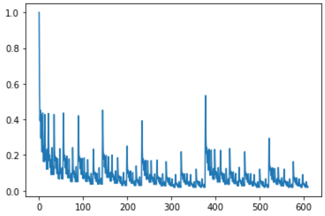
$A^n\mathbf{1}$ approaches a multiple of the Perron eigenvector, and its growth rate, the Perron eigenvalue, tells us what the asymptotic growth rate of the number of boards is as we increase the thickness.

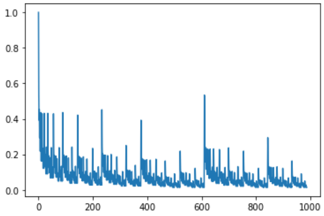


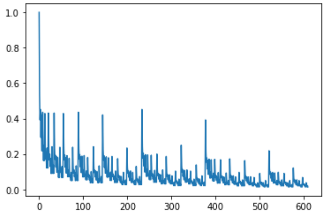


The 2-D Problem: Interpreting the Adjacency Matrix

$A^n\mathbf{1}$ approaches a multiple of the Perron eigenvector, and its growth rate, the Perron eigenvalue, tells us what the asymptotic growth rate of the number of boards is as we increase the thickness.







The 2-D Problem: Interpreting the Adjacency Matrix

- The dimension of A_k grows like $Fib(k) \times Fib(k)$. E.g. A_{30} is $2,178,309 \times 2,178,309$.
- The number of non-zero entries in these matrices grow like 2^k (A_{30} has $1,431,655,765$ nonzero entries), but it turns out, you only need to store the vector, since the matrix operation on the vector can be coded without the need to hold the matrix in computer memory, so the memory requirements only grow like ϕ^k, where $\phi \approx 1.618$ is the golden ratio.

The 3-D Problem is Hard

- Just as we create an adjacency matrix of 1-D slices for the 2-D problem, we can also create an adjacency matrix of 2-D slices for the 3-D problem.
- But there is no nice way of generating these 2-D slices that yields easy to store matrices.
- Unlike for the 2-D problem, we need to store the entire matrix in memory.

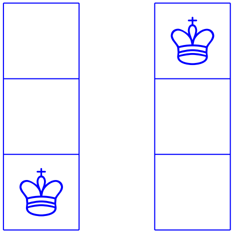
The 3-D Problem is Hard

- Just as we create an adjacency matrix of 1-D slices for the 2-D problem, we can also create an adjacency matrix of 2-D slices for the 3-D problem.
- But there is no nice way of generating these 2-D slices that yields easy to store matrices.
- Unlike for the 2-D problem, we need to store the entire matrix in memory... sort of.

Matrix Compression

- Studying the Perron eigenvector, we noticed that many entries were repeated.
- These repeated entries usually corresponded with symmetries of slices such as flips or rotations.

$$\underline{x} = \begin{bmatrix} 1.0 \\ 0.5513875 \\ 0.3554157 \\ 0.5513875 \\ 0.3554157 \end{bmatrix}$$



Matrix Compression

- We then realized that we could use these repeated eigenvector entries to reduce the dimension of the matrices used in our calculations by summing the rows corresponding to identical eigenvector entries and eliminating the redundant column index.

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Matrix Compression

- In fact, we can do all the compression in one process as follows:

$$A_C = L\,A\,R$$

- A_c is the compressed matrix
- A is a $d \times d$ matrix while A_c is a $d' \times d'$ matrix with $d' < d$.
- R is a $d \times d'$ matrix that has one column for every unique eigenvector entry, and the column is the indicator function for that class of slices.
- L is a $d' \times d$ matrix that has one row for every unique eigenvector entry. For each row and its corresponding class, L has a 1 at the very first instance of an index corresponding to that class and 0s elsewhere.

Matrix Compression

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A_c = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

- L, A, and R for 3×1 slices.
- If $\underline{\mathrm{x}}$ is a Perron eigenvector of A and λ is the Perron eigenvalue of A, then $L\,\underline{\mathrm{x}}$ is a Perron eigenvector of $L\,A\,R$ with

$$L\,A\,R\;L\underline{\mathrm{x}} = \lambda L\,\underline{\mathrm{x}}.$$

Matrix Compression

$$L\,\underline{\mathrm{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_a \\ x_b \\ x_c \\ x_b \\ x_c \end{bmatrix} = \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix}$$

$$R(L\,\underline{\mathrm{x}}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} = \begin{bmatrix} x_a \\ x_b \\ x_c \\ x_b \\ x_c \end{bmatrix}$$

$$A_c\,(L\,\underline{\mathrm{x}}) = L\,A\,R\,L\,\underline{\mathrm{x}} = L\,A\underline{\mathrm{x}} = \lambda\,L\,\underline{\mathrm{x}}$$

$$\begin{bmatrix}
0&0&1&0&0&0&0&0&0&0&0&1&0&0&1&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
1&0&0&1&0&0&0&0&0&0&0&0&1&0&0&1&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&1&0&0&1&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1\\
0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1\\
1&0&0&1&0&0&0&0&0&0&0&0&1&0&0&1&0&0&1\\
0&0&1&0&0&0&0&0&0&1&1&1&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
1&0&0&1&1&1&0&0&0&0&0&0&1&0&0&1&1&1&1\\
0&0&1&0&0&1&0&0&1&0&0&1&1&1&1&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&1\\
1&0&0&1&0&0&1&0&0&1&0&0&1&1&1&1&0&0&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{bmatrix} \longrightarrow \begin{bmatrix}
0&0&1\\
0&3&1\\
12&6&1
\end{bmatrix}$$

The adjacency matrix for 3×3 slices with one opposite edge pair glued together: uncompressed and compressed.

Matrix Compression

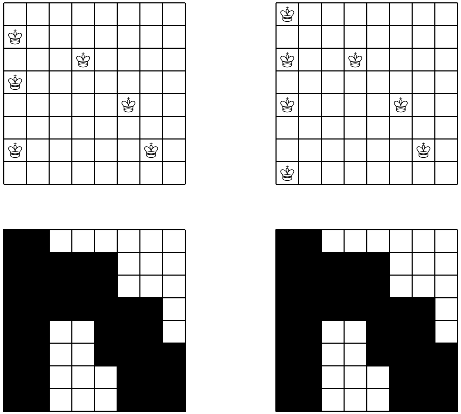
We searched the literature for others using matrix compression and found:

- "Exact and Approximate Compression of Transfer Matrices for Graph Homomorphisms" by Lundow and Markström
- "Compression of Transfer Matrices" by Lundow.

These authors describe matrix compression via graph homomorphisms. In our case, the vertices are slices and edges are possible adjacencies between slices.

The key step in using this matrix compression technique is identifying the graph homomorphisms.

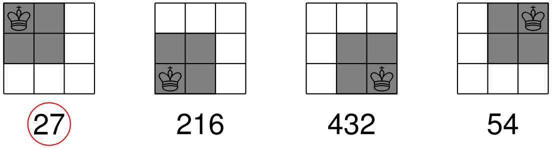
Graph Homomorphisms for Non-Attacking Kings



We find graph homomorphisms by identifying the regions where kings can be placed in adjacent slices.

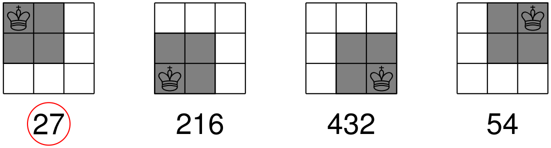
Graph Homomorphisms for Non-Attacking Kings

- We call the adjacent region of possible kings positions to a slice a PKP.
- To check whether two slices were equivalent, we compared their PKP patterns under rotation, reflection, and translation (when applicable).
- We converted each PKP to an integer, and for each class of equivalent PKPs, we determined the least such integer, which we called the min PKP.



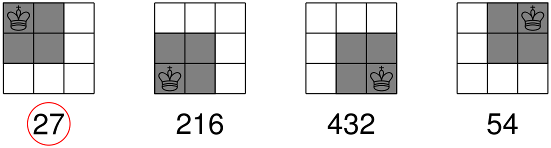
Graph homomorphisms for non-attacking kings

- We partitioned all slices into classes that share the same min PKP.
- This gives us an effective method for computing the graph homomorphisms for our problem.



Graph homomorphisms for non-attacking kings

- We partitioned all slices into classes that share the same min PKP.
- This gives us an effective method for computing the graph homomorphisms for our problem.



All four slices go in bucket labeled 27.

Computation

- The end result compressed matrices we created had entries $c_{i,j}$ whose value is the number of adjacencies between *some* representative of the i^{th} slice class and *all* slices in class j.
- If slice class i has 1000 members, we needed to check roughly 1000 times fewer adjacencies than in a non-compressed adjacency matrix.
- Counting these adjacencies was the primary time constraint we ran into. The largest slice dimensions we were able to work with had roughly 1000 members per class, and so matrix compression sped up our code by roughly a factor of 1000.
- The compressed matrix also decreased the storage requirements by roughly 1000 fold. Both uncompressed and compressed matrices could be stored as sparse matrices.

PSA: GPUs are Your Friend

- Originally we used multiple CPUs to find slice adjacencies with Java.
- Later we used the PyTorch Python library to move this operation to GPUs and achieved roughly another 1000 times speedup.

Results

$$1.1722475193 \leq \text{multiplicity}_{3D} \leq 1.1798420399$$

$$0.2292772260 \leq \text{capacity}_{3D} \leq 0.2385937211,$$

Results

n	$F(n,n,n)$	$F'(n,n,n)$	$F''(n,n,n)$	$F'''(n,n,n)$
1	2	1	1	1
2	9	9	9	9
3	2,089	469	109	28
4	3,144,692	955,597	285,457	86,409
5	2,748, 613,397,101	141,446, 194,951	7,797, 443,501	442, 888,551
6	107,008,949, 868,167,431,857	3,540,028,254, 720,734,235	126,286,208, 383,726,353	
7	13,894,384, 033,156,308,816, 935,906,058,416	73,142,142, 037,998,950, 249,305,520,745	421,725,200, 626,057,564, 456,468,571	

Table: The number of $n \times n \times n$ solids of various sizes with 0, 1, 2, and 3 pairs of attached ends.

Open Questions

- Can we create an indexing scheme for the 2-D slices that allow better storage of uncompressed adjacency matrices (much like what is done for 1-D slices)?

Open Questions

- Can we create an indexing scheme for the 2-D slices that allow better storage of uncompressed adjacency matrices (much like what is done for 1-D slices)?
- Can we bound the relative sizes of the other eigenvalues for an adjacency matrix? Doing so may greatly improve the lower bound on multiplicity$_{3D}$ and capacity$_{3D}$.

Open Questions

- Can we create an indexing scheme for the 2-D slices that allow better storage of uncompressed adjacency matrices (much like what is done for 1-D slices)?
- Can we bound the relative sizes of the other eigenvalues for an adjacency matrix? Doing so may greatly improve the lower bound on multiplicity$_{3D}$ and capacity$_{3D}$.
- Can we develop a technique that scales to 4-D and beyond?

Open Questions

- Can we create an indexing scheme for the 2-D slices that allow better storage of uncompressed adjacency matrices (much like what is done for 1-D slices)?
- Can we bound the relative sizes of the other eigenvalues for an adjacency matrix? Doing so may greatly improve the lower bound on multiplicity$_{3D}$ and capacity$_{3D}$.
- Can we develop a technique that scales to 4-D and beyond?
- Compressed matrices are not always full rank. Why? How much more compression is possible? Is it practical?

Open Questions

- Can we create an indexing scheme for the 2-D slices that allow better storage of uncompressed adjacency matrices (much like what is done for 1-D slices)?
- Can we bound the relative sizes of the other eigenvalues for an adjacency matrix? Doing so may greatly improve the lower bound on multiplicity$_{3D}$ and capacity$_{3D}$.
- Can we develop a technique that scales to 4-D and beyond?
- Compressed matrices are not always full rank. Why? How much more compression is possible? Is it practical?
- Email any questions or ideas to cohenn1@uci.edu

Open Questions

- Can we create an indexing scheme for the 2-D slices that allow better storage of uncompressed adjacency matrices (much like what is done for 1-D slices)?
- Can we bound the relative sizes of the other eigenvalues for an adjacency matrix? Doing so may greatly improve the lower bound on multiplicity$_{3D}$ and capacity$_{3D}$.
- Can we develop a technique that scales to 4-D and beyond?
- Compressed matrices are not always full rank. Why? How much more compression is possible? Is it practical?
- Email any questions or ideas to cohenn1@uci.edu
- Thanks for your attention!

Lower bound

- Maxminum principle
- Switch indices and apply maximum principle twice each.

$$\text{multiplicity}_{3D} \geq \left[\frac{\left(\frac{\lambda_{p+2q+1,t+2u+1}}{\lambda_{p+2q+1,2u+1}} \right)^{1/t}}{\lambda'_{2s,2q+1}{}^{1/2s}} \right]^{1/p}$$

Upper bound

- The trace of powers of adjacency matrices counts cylindrical boards; i.e. the number of boards that both begin and end with the same slice
- The trace is sums of powers of the eigenvalues.
- Switch indices.

$$\eta_3 \leq {\lambda''_{2p,2q}}^{1/4pq}$$

References

[1] R. J. Baxter, I. G. Enting, and S. K. Tsang, “Hard-square lattice gas,” *J. Statist. Phys.*, vol. 22, no. 4, pp. 465–489, 1980.

[2] N. J. Calkin, K. James, S. Purvis, S. Race, K. Schneider, and M. Yancey, “Counting kings: As easy as $\lambda_1, \lambda_2, \lambda_3 \dots$,” in *Proceedings of the Thirty-Seventh Southeastern International Conference on Combinatorics, Graph Theory and Computing*, vol. 183, 2006, pp. 83–95.

[3] N. J. Calkin and H. S. Wilf, “The number of independent sets in a grid graph,” *SIAM Journal on Discrete Mathematics*, vol. 11, no. 1, pp. 54–60, 1998.

[4] S. Friedland, “On the entropy of $\mathbf{Z}^d$ subshifts of finite type,” *Linear Algebra Appl.*, vol. 252, pp. 199–220, 1997.

[5] S. Friedland, P. H. k. Lundow, and K. Markström, “The 1-vertex transfer matrix and accurate estimation of channel capacity,” *IEEE Trans. Inform. Theory*, vol. 56, no. 8, pp. 3692–3699, 2010.

[6] P. H. k. Lundow, “Compression of transfer matrices,” in 1-3, vol. 231, 17th British Combinatorial Conference (Canterbury, 1999), 2001, pp. 321–329.

[7] P. H. k. Lundow and K. Markström, “Exact and approximate compression of transfer matrices for graph homomorphisms,” *LMS J. Comput. Math.*, vol. 11, pp. 1–14, 2008.

[8] H. C. Marques Fernandes, Y. Levin, and J. J. Arenzon, “Equation of state for hard-square lattice gases,” *Physical Review E*, vol. 75, no. 5, 2007.

[9] Z. Nagy and K. Zeger, “Capacity bounds for the 3-dimensional $(0, 1)$ runlength limited channel,” in *Applied algebra, algebraic algorithms and error-correcting codes (Honolulu, HI, 1999)*, ser. Lecture Notes in Comput. Sci. Vol. 1719, Springer, Berlin, 1999, pp. 245–251.