Randomised Algorithms

Team: Zopu De

Introduction

Overview:

Randomised algorithms play a crucial role in modern computing, offering powerful alternatives to their deterministic counterparts, particularly in situation involving high-dimensionality,uncertainty or adverse inputs. Each algorithm we've used leverages randomness to improve efficiency,optimisation and avoid worst-case scenarios. Through analysis we've tried to highlight why non-determinism isn't a workaround but a strategic advantage in algorithm design.

Algorithms Analysed:

1. Basin-Hopping (Aaryan Shah - 2024113014)
2. Miller-Rabin (Ishaan Shiv Kumar - 2024101098)
3. Freivald's Algorithm (Sahishnu Pawan Kumar - 2024113016)
4. Simultaneous Perturbance Stochastic Algorithm (Soham Acharya - 2024113012)'
5. Randomised Routing (Arjun Dingankar - 2024113024)
6. Randomised Quicksort (Collective)

Bonus:

Used Monte Carlo Simulations to predict the table of a football league

Github Repo: https://github.com/shahiam/ZopuDe_AAD_FinalProject

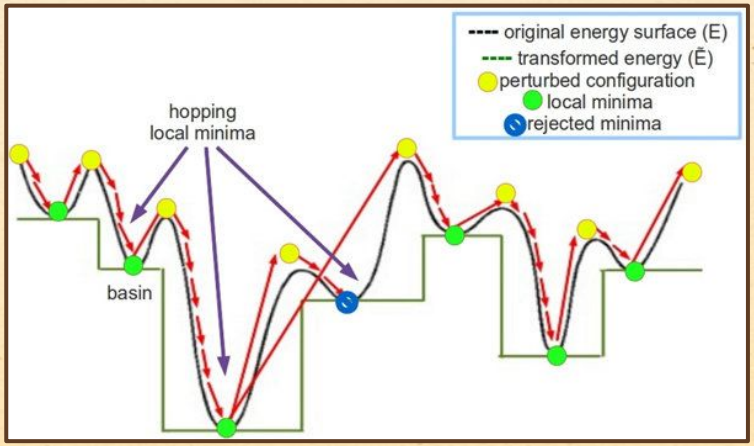
Basin-Hopping Algorithm

Motivation:

→ Several optimization problems are non-convex ⇒ multiple local minima

→ Gradient based methods(BFGS,CG,etc.) get stuck on the local minima

→ Global exploration required, while being efficient locally



Algorithm Overview:

1. Guess Solution(x)
2. Randomised Perturbation(δ): x' → x + δ
3. Local Minimisation(Gradient Descent): x* = local_min(x')
4. If f(x*) is better OR passes Metropolis Criterion: x → x*
5. Repeat

→ Metropolis Criterion:

- Allows occasional uphill moves to escape local minima

$$P(\text{uphill accept}) = e^{(-\Delta E/T)}$$

- If new energy is lower → accept
- If higher accept with above defined probability

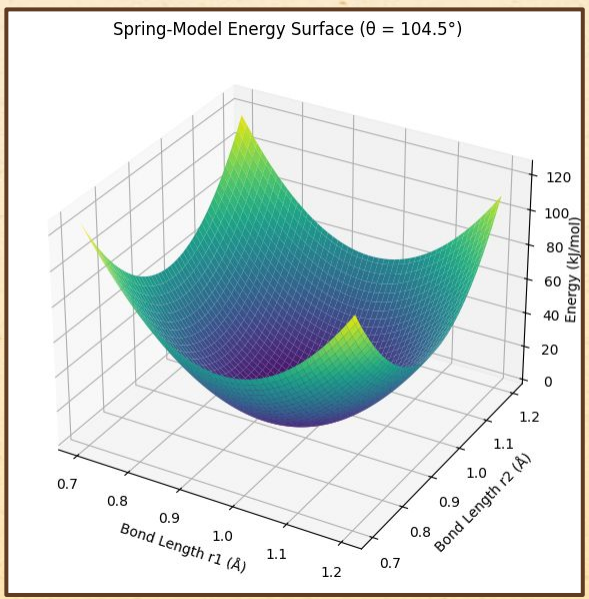
Pic Credits: Hashmi, I., & Shehu, A. (2013). *HopDock: A probabilistic search algorithm for decoy sampling in protein-protein docking* [Figure showing Basin-Hopping framework]. In *HopDock: A probabilistic search algorithm for decoy sampling in protein-protein docking*. ResearchGate.https://www.researchgate.net/publication/260377431/figure/fig2/AS:216443955224580@1428615668331/Overview-of-Basin-Hopping-Framework-Under-the-BH-framework-the-energy-surface-is_W640.jpg

Basin-Hopping Algorithm

Performance and Behaviour: (Convex Functions)

- Spring Model of H_2O: $E = \Sigma\ 0.5*k_b*(r_i - r_0) + 0.5*k_a*(\theta - \theta_0)$

Method	Time Taken	Accuracy	Notes
Basin-Hopping	~1.5-3.0 s	High	Slow,as it performs many Gradient Descent minimisations
BFGS	~0.005-0.02 s	Precise	Fast,ideal for smooth landscapes
Newton's Method	~0.001–0.01 s	Precise	Fast due to quadratic convergence near optimum
Conjugate Gradient	~0.01–0.03 s	Precise	Slower, due to no Hessian approximation



- Robust but slow as each Basin-Hopping iteration → Multiple Gradient Descent Iterations

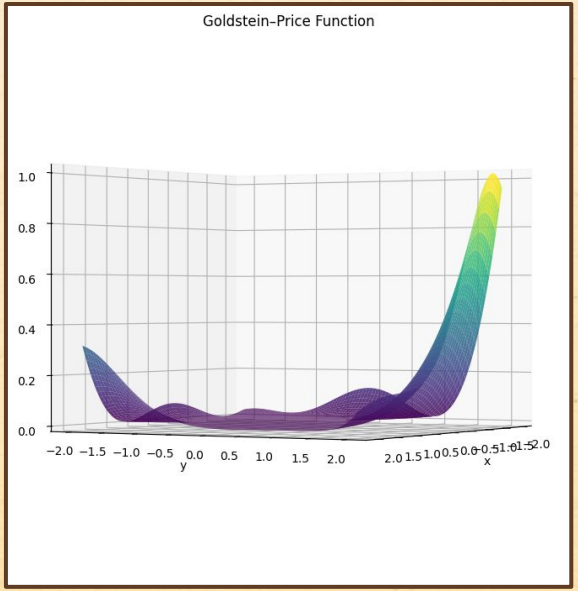
- Algorithm designed for non-convex → gives accurate answers but slower

Basin-Hopping Algorithm

Performance and Behaviour: (Non-Convex Functions)

- Goldstein-Price Function: $z=[1+(x+y+1)^2(19-14x+3x^2-14y+6xy+3y^2)][30+(2x-3y)^2(18-32x+12x^2+48y-36xy+27y^2)]$

Method	Time Taken	Accuracy	Notes
Basin-Hopping	~2.0-2.5 s	High	Slowest,best Global Performance
BFGS	~0.02 s	Low	Fast but answer sensitive to initial guess
Newton's Method	~0.001–0.02 s	Low	Fastest but again sensitive to initial guess
Conjugate Gradient	~0.03–0.06 s	Low	Slowest, step size decreases so more iterations



- Slowest but only one that gives correct as it doesn't get trapped in a basin

- Thus, non-determinism helpful in case of non-convex functions

Basin-Hopping Algorithm

Key Takeaways:

→ Essential for optimisation in multimodal/non-convex functions

→ Stopping criteria important as that governs the time complexity of the algorithm

Interesting Finds:

→ Basin-Hopping reached the global minima from the worst of the initial guesses

→ Performance heavily dependent on the local minimiser used

→ T(n) = O(h*t*n)

- h = number of Basin-Hopping iterations
- t = number of Gradient Descent iterations
- n = dimension of the objective function

Challenges:

→ Tuning of parameters is hard(step size,temperature,etc.) as it is problem dependent

→ Computational cost depends on the convergence criterion thus, an optimal condition must be found

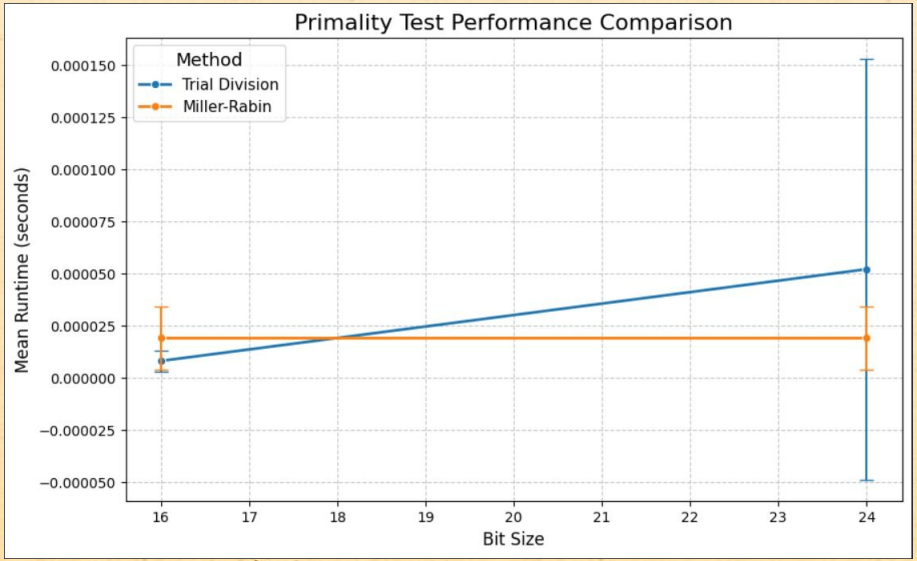
Miller-Rabin Algorithm

High-Level Overview of Algorithm:

- Consists of Fermat's Little Theorem ($a^{p-1} = 1 \pmod p$), then squaring $x = a^d \pmod n$; $\{n - 1 = 2^s \cdot d\}$, and squaring x repeatedly.
- If $x = (n - 1) \pmod n$ at any point during the squaring, the round of testing is passed, BUT if it fails, it is *definitely* composite [no false negatives] – the probability of error for a positive output is 0.25^k, for k trials
- Time complexity: $O(n \cdot (\log n)^3)$

Results (graph and charts):

Here, we can see that Miller-Rabin performs better than the deterministic trial division at a bit size of over 18 (exponential), as it stays effectively constant in runtime.



Miller-Rabin Algorithm

Key takeaways, interesting findings, and challenges [during implementation]:

Miller-Rabin is a probabilistic primality test that offers a big advantage over deterministic algorithms. Miller-Rabin offers a much faster primality test, and the difference is considerable for larger primes with an interestingly *near zero error probability for even just 5 iterations* (probability around 9.1×10^{-4}).

Because of this, the Miller-Rabin algorithm can be used in several real life applications, such as RSA encryption which uses very large primes.

One challenge was deciding what k (iteration number) to use to balance between accuracy and speed.

Freivalds' Algorithm

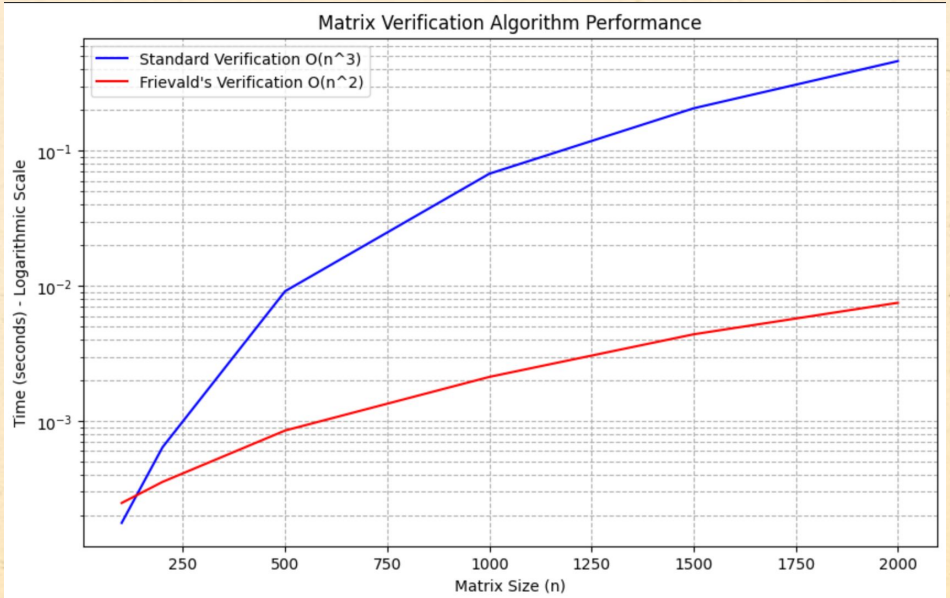
High-Level Overview of Algorithm:

- Frievald's Algorithm is a randomised matrix verification algorithm
- Performs the operation in $O(k \cdot n^2)$, for k trials
- Chooses a random 1xn vector ***r*** that is composed completely of 0s and 1s
- Then multiplying it with one of the input matrices B, *then* multiplying the result by the other input matrix A; result = K
- Finally, it subtracts the product of C [target matrix, presumed to be A•B] and ***r*** from K.
- If the result is NOT 0, then matrix C is definitely not a product of A and B, as Freivald's cannot output false negatives.
- If the result is 0, it is very likely that C *is* the product of A and B, with a small error probability of $2^{-k} \leqslant 0.5$.

Freivalds' Algorithm

Results (graph and charts):

Frievald's algorithm outperforms the deterministic method of naive verification (matching every element of C with element of A•B) for nearly all input matrix sizes, and does especially well for larger sizes, with much faster runtimes.



Key takeaways, interesting findings, and challenges:

- Although Freivald's algorithm has simple principles, it is exceedingly effective in executing its goal!
- An interesting finding is that it is a lot faster than any known corresponding deterministic algorithm
- Even after over 100 years of computer scientists and mathematicians trying to improve itd efficiency, their *best result is* $O(n^{2.3729})$.
- One challenge during implementation was ensuring there were no random false negatives from the logic of the code.

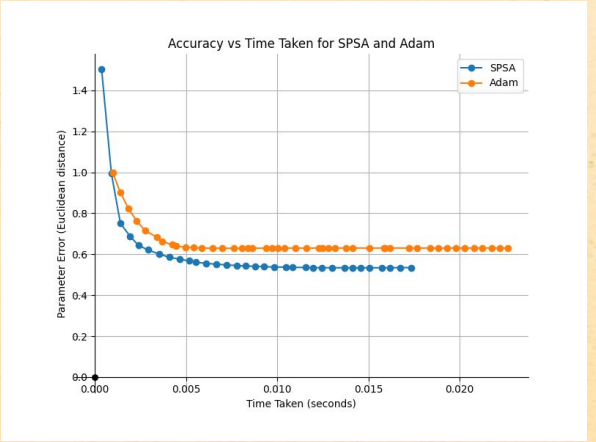
Simultaneous Perturbation Stochastic Algorithm

High-Level Overview of Algorithm:

SPSA (Simultaneous Perturbation Stochastic Approximation) works by: simultaneously perturbing all parameters in random directions, evaluating the objective function at only two perturbed points, estimating the gradient from these two noisy measurements, and updating parameters along the negative gradient direction with decaying step sizes. This makes it highly efficient for high-dimensional noisy optimization since it requires only 2 function evaluations per iteration regardless of dimensionality.

Results (graph and charts):

At high noise levels, SPSA significantly outperformed Adam because its stochastic perturbation-based gradient estimation is inherently robust to noise, while Adam's numerical gradients became unreliable. SPSA converged faster with lower parameter error, demonstrating that algorithms designed for noisy evaluations excel when measurement uncertainty is high.



Simultaneous Perturbation Stochastic Algorithm

Key takeaways, interesting findings, and challenges [during implementation]

Key Takeaways: SPSA estimates gradients using only two function evaluations per iteration regardless of dimensionality, making it highly efficient compared to finite-difference methods. The algorithm's robustness to noise and adversarial inputs (through random perturbations) mirrors randomized Quicksort's advantage over deterministic approaches.

Interesting Findings: SPSA dramatically outperformed Adam at high noise levels with better parameter recovery and lower computational cost, demonstrating that domain-specific algorithms excel where general-purpose methods struggle.

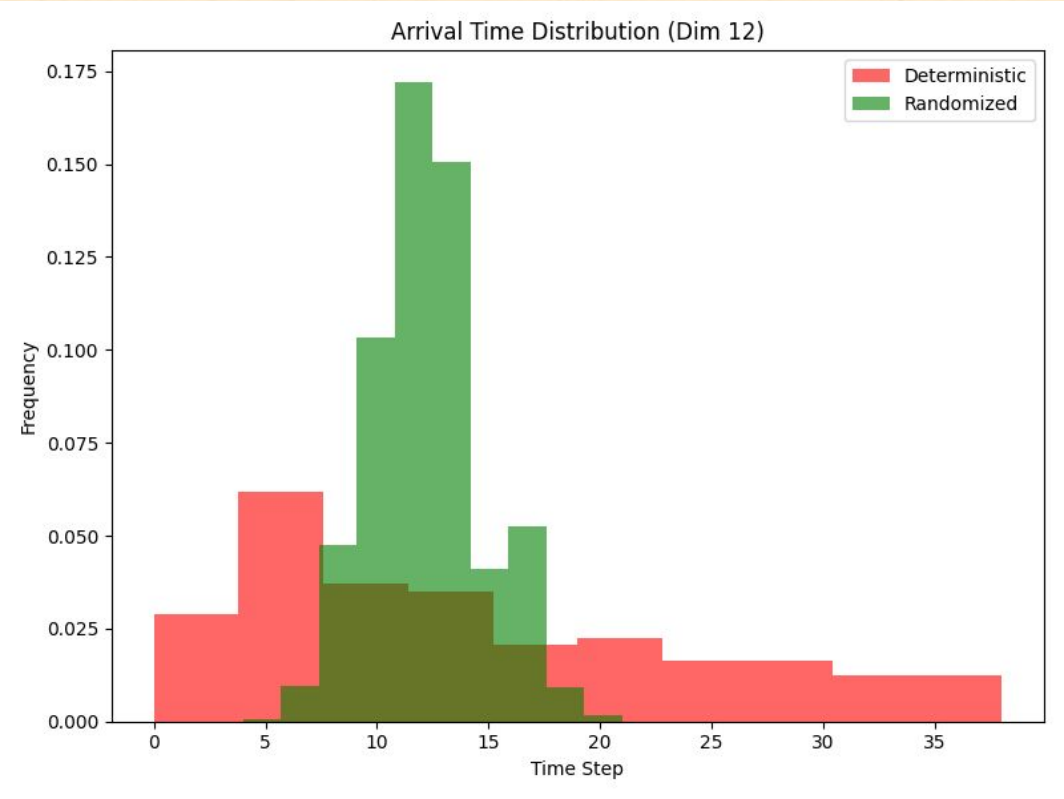
Implementation Challenges: Tuning the two decay schedules (learning rate and perturbation magnitude) is critical but difficult; high recursion depth can exceed stack limits; handling degenerate cases with repeated elements becomes problematic; and expensive loss function evaluations limit practical applicability despite theoretical elegance.

Randomised Routing

High-Level Overview of Algorithm:

- When trying to route messages through a network, any deterministic method has the vulnerability that the worst case scenario scales exponentially with the dimension.
- This is because an antagonistic set of messages has the potential to overload a single edge, which has a limited capacity.
- Randomised routing, instead of routing from source to destination directly, instead routes from a source to a random intermediate and then to the destination.
- This distributes the workload evenly, and prevents worst-case runtimes.
- This effect is seen at higher dimensions, where the higher path length caused by the intermediate matters less than the effect of the overloading.

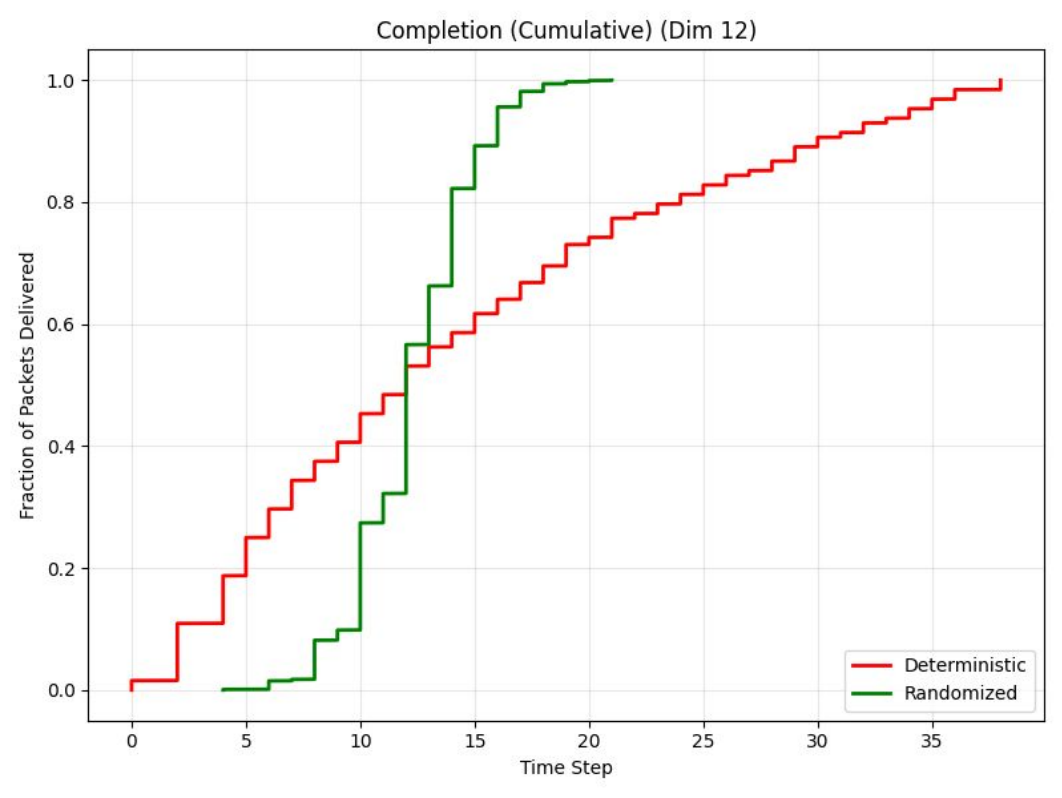
Randomised Routing



For the worst case scenario, the deterministic routing is highly unpredictable, with some packets taking much longer to arrive, which is bad for any implementation.

Randomized routing on the other hand has a high peak and low deviation, which makes it predictable and useful.

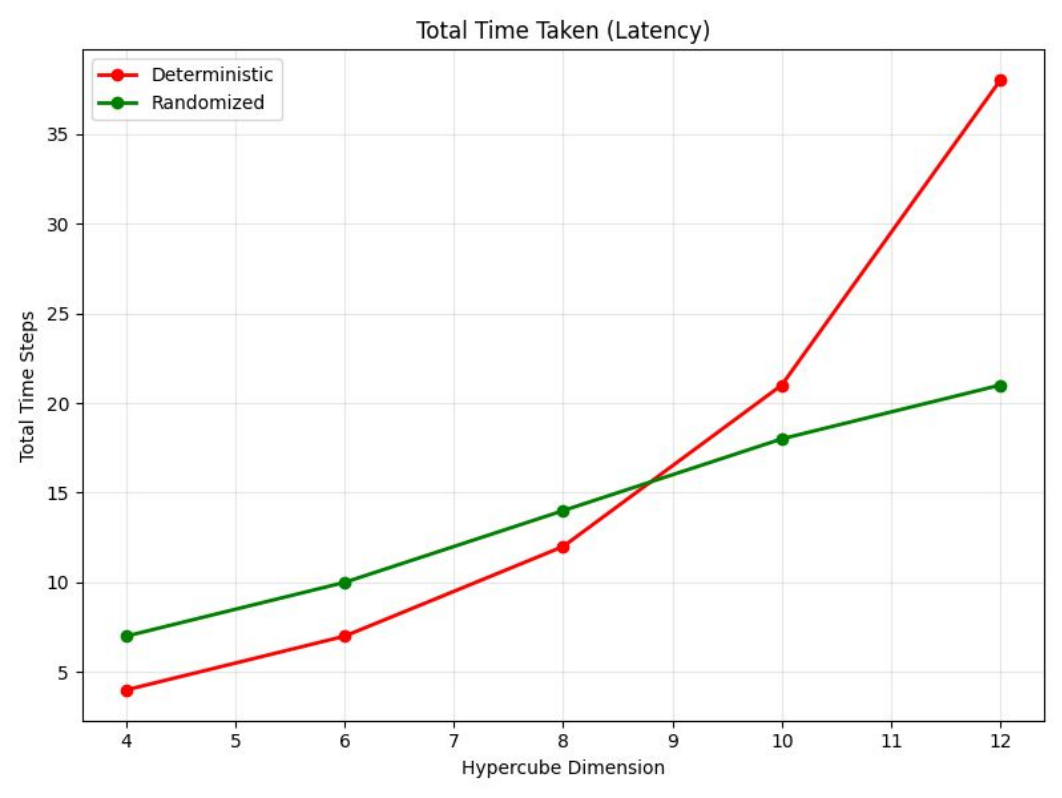
Randomised Routing



While the deterministic method achieves the absolute fastest arrival times (some packets arrive near time 0), it suffers from high variance and a "heavy tail," causing the total completion time to drag out.

The Randomized method has a tightly clustered performance; it processes almost all packets within a predictable window , effectively eliminating extreme delays and finishing the entire workload twice as fast as the deterministic method's worst case.

Randomised Routing



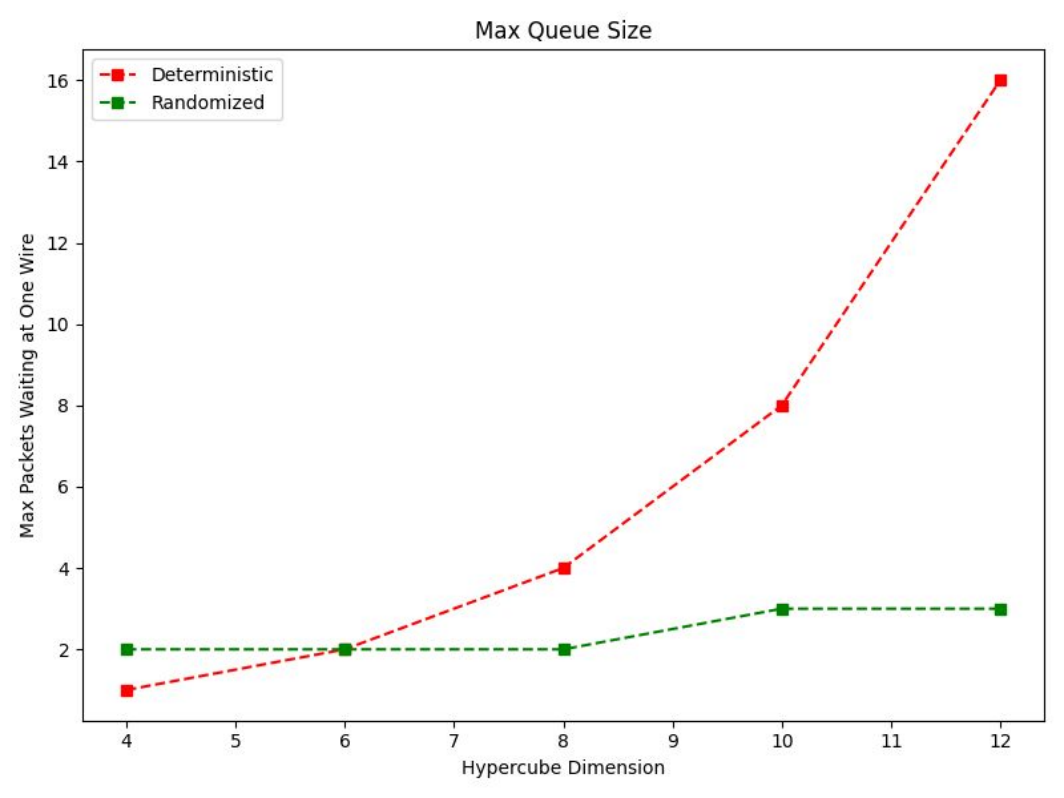
For smaller dimensions, the Deterministic method is more efficient. However, a critical inflection point occurs shortly after dimension 8, where the trends cross.

Beyond this point, the Deterministic algorithm shows a steep, non-linear increase in latency.

In contrast, the Randomized algorithm maintains a much flatter, near-linear growth trajectory.

The Randomized algorithm provides significantly superior scalability and efficiency for higher-dimensional hypercubes.

Randomised Routing



The max queue size grows exponentially with dimension for the deterministic method, while it is linear for randomized routing.
The max queue size is important because in a real network, a node can handle only a certain amount of requests at a time, and there are potential stability issues if this limit gets exceeded.

Randomised Routing

Key takeaways

- Scalability vs. Overhead: The Deterministic algorithm is efficient for small networks because it takes the shortest path. However, it fails to scale, hitting a "scalability wall" around dimension 10. The Randomized algorithm introduces overhead (longer paths), making it slower for small systems, but it the worst case complexity scales linearly, making it better for large-scale systems.
- Predictability: Randomised Routing offers predictable latency regardless of traffic patterns, whereas the Deterministic algorithm is highly sensitive to certain permutations, leading to unpredictablity.

Interesting findings

- The "Worst-Case" Permutation: The exponential explosion in the red curve (Deterministic) likely corresponds to a "Bit-Reversal" or "Transpose" traffic pattern. In these patterns, a deterministic router (like simple Bit-Fixing) forces all packets to pass through a small set of intermediate nodes, creating massive bottlenecks.
- The Paradox of "Longer is Faster": Randomized routing typically sends a packet to a random intermediate node first before sending it to the true destination. Although this doubles the path length, it randomizes the traffic distribution.

Challenges:

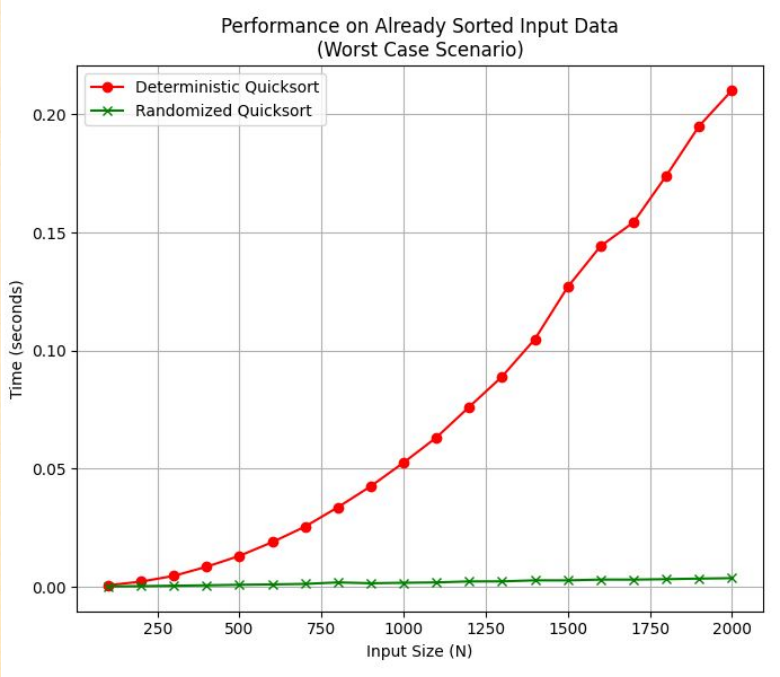
The benefit of randomised routing are visible only at higher dimensions, and in particular scenarios where edges get overloaded.

Randomised QuickSort

High-Level Overview of Algorithm:

- Divide-and-conquer sorting algorithm that selects one of the elements of the array as a pivot, partitions the array into elements less than, equal to and greater than the pivot and then recursively sorts the less than and greater than subarrays by using the same technique.
- In randomised quicksort, the differential step is the pivot selection. A pivot is chosen uniformly at random from the current subarray.
- This randomisation decreases the probability of poor pivot selection.
- Thus it prevents the algorithm from running in its worst-case $O(n^2)$ time complexity

Randomised QuickSort



Results (graph and charts):

You can see in the graph opposite, that the randomised QuickSort does much better than $O(n^2)$, in the worst-case scenario of the elements already being sorted, than the deterministic case - this proves the utility of the former algorithm.

Key takeaways [inferred above], interesting findings, and challenges [during implementation]:

Challenge - in non-worst-case scenarios, the randomised quicksort usually performs worse than the deterministic QuickSort.
Interesting Finding - randomized Quicksort is robust against adversarial inputs designed to exploit weaknesses in pivot selection strategies. The adversary cannot predict or influence the pivot choice, making it difficult to force the algorithm into its worst-case scenario.

BONUS

Football League Simulator

We utilised a Monte Carlo algorithm - with the option to select number of iterations - to create a program that simulates the league standings of European domestic football teams over a period of five years, and it can give you detailed stats of each team in the league, such as goals scored, goal difference - apart from the points of each team, that will determine a winner.

The basis of the simulation is the datasets of previous year league standings, and then uses them to skew a (random) Poisson distribution, in order to then map it to teams for the year of simulation.

BONUS

Football League Simulator

	Team	Avg Points	Avg GD	Avg GF	Avg GA
1	Bayern Munich	76	55.11	99.5	44.38
2	RB Leipzig	65.33	29.49	71.09	41.6
3	Leverkusen	64.74	28.87	72.54	43.66
4	Dortmund	64.69	30.42	81.16	50.74
5	Ein Frankfurt	47.87	0.07	57.29	57.22
6	Stuttgart	46.21	-2.47	56.6	59.07
7	Wolfsburg	45.59	-3.18	51.89	55.07
8	Freiburg	45.43	-3.51	53.14	56.65
9	Union Berlin	44.91	-3.95	47.81	51.76
10	FC St. Pauli 1910	44.69	-5.15	54.81	59.95
11	Holstein Kiel	44.67	-4.83	55.13	59.97
12	M'gladbach	42.7	-9.36	57.69	67.06
13	Hoffenheim	42.48	-9.5	57.15	66.65
14	Heidenheim	41.14	-11.19	51.43	62.62
15	Mainz	40.39	-11.51	47.27	58.77
16	Werder Bremen	35.54	-20.27	46.16	66.42

Conclusion

- Randomised algorithms consistently demonstrate how incorporating controlled randomness enhances performance across a variety of computational tasks.

Key Lessons:

- Randomness helps avoid worst-case behaviour
- Some Probabilistic Algorithms help deliver faster results
- Fine tuning the inherent parameters in each code is a necessity, as in cases where the time complexity depends on stopping criteria sub-optimal parameters lead to non-convergence causing the algorithms to loop in some cases

- Randomness transforms complexity into opportunity, enabling algorithms to solve problems that would otherwise be too slow, too large, or too unpredictable.

Thanks!