Parallel Implementation of Random Walk Simulations with Different Movement Algorithms

Conference Paper · November 2021

DOI: 10.1109/TELFOR52709.2021.9653438

5 authors, including:

Andrej Nasteski
Ss. Cyril and Methodius University in Skopje
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Miroslav Mirchev
Ss. Cyril and Methodius University in Skopje
37 PUBLICATIONS **172** CITATIONS
SEE PROFILE

Marjan Gusev
Ss. Cyril and Methodius University in Skopje
487 PUBLICATIONS **2,016** CITATIONS
SEE PROFILE

Lasko Basnarkov
Ss. Cyril and Methodius University in Skopje
56 PUBLICATIONS **318** CITATIONS
SEE PROFILE

Parallel Implementation of Random Walk Simulations with Different Movement Algorithms

Andrej Nasteski Lasko Basnarkov Marjan Gusev Miroslav Mirchev Vladimir Zdravevski
Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University in Skopje, North Macedonia

***Abstract*—This article contains a detailed explanation of the research, methodology and results of different searching strategies when traversing through an unknown area. We have been challenged by the ways to simulate and evaluate the effectiveness of various approaches in order to speed up the simulation using parallel computing. The goal is to compare the results from each combination of algorithms. The two categories of algorithms considered are direction based and step size based algorithms. In summary, the combination of exponential step size with backtracking and forward check direction algorithm produced the best results. We also concluded that using a parallel implementation resulted with a substantial speed up when compared to a sequential approach.**

***Index Terms*—random walk, search, parallel simulation**

I. Introduction and Problem Statement

The main task in this research is simulating the effectiveness of different searching strategies when traversing through an unknown area. The problem is represented with a searching agent roaming in a closed area with fixed dimensions – a 200x200 grid, searching for randomly located food nodes in the area. The goal of the agent is to find as many food nodes as possible using the trial-and-error method (keep moving until you find a food node) and moving to the neighboring cell in a 2D dimensional grid in one of four possible directions - north, east, south and west. Each move is specified by the direction and step size. [1] [2] [3] The results of each combination of search algorithm are compared by a score measured by the amount of food nodes collected in 50, 100, 200 and 500 steps, in addition to the number of steps required for the agent to collect a single food node (also known as first passage time [4] [5] [6]).

For simplicity, only one agent is placed in the middle of the grid, and 100 food nodes are placed randomly. For each epoch the start position of the agent is identical and position of the food nodes is determined randomly. The two main aspects considered are the agent's step size [1] [2] [3] and step direction. The results of each combination of algorithms are composed from statistical measurements taken from each simulation. Therefore, in order for the data to be representative, it is essential that the measurements are taken from a relatively large population size. For each algorithm combination the simulation is repeated 1000 times. Hence a necessity for a parallel approach when computing the simulations arises. Since each of the runs of 500 steps is independent of each other, a parallel approach for computing the simulation may be more suitable for 1000 runs (or more). Note that we can use parallel computing to speed up the calculation of a larger number of iterations of each random walk (run multiple simulations at once) however we cannot divide the task any further by computing each of the 500 steps of an epoch in parallel. The main reason for this is the sequential nature of the problem, meaning the computation of the agent's state in step *n* is dependent of the agent's state in step *n-1*.

The paper is organized according to the following structure. Section II presents the related work and Section III methods used in this paper. The architecture and experiments is presented in Section IV. The results from the simulation are shown in Section V, which are then discussed in Section VI. The conclusion about this simulation is shown in Section VII.

II. Related Works

Regarding other research articles related to the topic, the majority focus on the step size of the search agent. [1] [2] [3] Mainly using Lévy flights in contrast to Brownian motion (In our case Brownian motion is Uniform strategy). Lévy flights [1] are defined as randomly reoriented ballistic excursions whose l length is drawn from a power law distribution 1. From research articles that discuss first-passage time in complex systems and cover time in random searches ([4] [5] [6]) we saw fit to adopt first-passage time to be used as a statistical measurement for our search algorithms. The simulation we are modeling is based on the assumption that the agent has some kind of a memory and orientation of the field. Meaning the agent has memory of which direction he came from (the agent's last step) and a sense of orientation in order to determine with a certain probability, which direction the agent's next step should be.

$$P(l) \underset{l\to\infty}{\propto} \frac{1}{l^{1+\mu}}, \quad 0 < \mu \leq 2 \tag{1}$$

III. Methodology

A. Strategies to solve the problem

The first problem with this simplistic task description is that the agent can walk past a food node that is really close to the

978-1-6654-2585-8/21/ $31.00 ©2021 IEEE

agent's position. With this kind of an approach the number of simulations needed to get a sufficiently representative result increases exponentially. Instead, each agent has a 'sight area', meaning if a food node is spotted in that agent's sight area, the agent stops moving randomly and starts converging to the closest food node position one step at a time. The sight area spans 5 coordinate points in each direction of the agent's current position.

Four different strategies for deciding which direction the agent is going to move next are elaborated as:

1) *Uniform (no) strategy* where all directions for the next action are equally probable.
2) *Backtracking strategy* based on calculation of the probability for an agent to go back to a spot from which he just came from is lowered (If an agent's previous action was going east the probabilities for the next action for going north, east, south and west are 0.3, 0.3, 0.3, 0.1 respectively).
3) *Forward Check and Backtracking strategy* where the probability for an agent to go back is lowered also, the probability for the agent to move to his relative 'forwards' direction is increased. If an agent's previous action was going east the probabilities for the next action for going north, east, south and west are 0.25, 0.4, 0.25, 0.1 respectively.
4) *Forward Check and Backtracking with Right side skew - strategy* same as the previous Forward Check and Backtracking strategy, only difference is that the agent has an increased probability to move to its relative right position rather than to its left position. If an agent's previous action was going east the probabilities for the next action for going north, east, south and west are 0.15, 0.4, 0.35, 0.1 respectively. The goal here is to stimulate a spiral - like movement.

The probability for each action for each algorithm is calculated from the previous action. Only one step is considered. The other aspect of the searching algorithm is the step size of the agent for each individual step. [1] [2] [3] The step size is decided with one of three different methods:

1) *Uniform step size* – step size is fixed (the default is one coordinate length at a time)
2) *Gaussian step size* – determining the step size by a sample from a Gaussian distribution with mean = 2, and standard deviation = 1, and
3) *Exponential step size* – determining the step size by a sample from an Exponential distribution with parameters: $\lambda = 0.5$ (the function takes a beta parameter, $\beta = 1/\lambda$)

The value of the step is round to an integer to reduce model complexity.

B. Evaluation methodology

In order to compare each individual combination of algorithms, statistical measurements were added as follows:

- *Collections for fixed number of steps*, by checking how many food nodes an agent has collected at a given milestone of steps. In our experiments we specify measurements for 50, 100, 200 and 500 steps.
- *Average number of steps per collection*, determining how fast can an agent find any food node, calculating the number of steps needed to find a single node. [7] [4] [5] [6]

In our experiments, one epoch is one random walk simulation of a given number of steps for a specified starting position of the agent, and locations of the food nodes. The total amount of food nodes collected are reset for each epoch.

The experiments were evaluated for each of the 1000 epochs. The agent in each epoch moves 500 steps. The mean of all 1000 epochs is calculated in a particular evaluation for each combination of algorithms. The final score is calculated by (2), where N_x is number of food nodes collected at step x, and N^* are steps taken to find the first food node.

$$score = \frac{N_{50} \cdot N_{100} \cdot N_{200} \cdot N_{500}}{N^*} \tag{2}$$

IV. Experiments

The model's architecture is described starting with description of the *search function*. For each step, check if the agent's current position is near any food node (5 coordinate points in each direction). If a food node is present in that range, the agent starts to move to the closest food node one step at a time. If there is no food node near the agent at the current position, then the implemented algorithm calculates the probability to move in each direction. The next step is to calculate how big of a step the agent should make. Again, depending on the step size algorithm, different probabilities are assigned to different step sizes. When the search function finishes the calculation of the probabilities, the agent's location is moved to the location with the highest probability.

Our experiments are based on calculating this sequence of search and move functions for each step for at least 500 times to produce one random walk or epoch out of one thousand.

The sequential nature of the problem makes it harder to fully utilize the parallel architecture of a GPU, since at each step there are different actions to choose from, depending on the previous state. Even though we cannot fully take advantage of the GPU, we can still make use of it for simulating many epochs of each 500 step random walk.

Since the initial code was written in Python, to utilize parallel computation without changing the programming language, we have used the library Numba. A number of simulations were done using the same machine and two (almost) identical functions, one sequential, and the other parallel. In order to compare the performance between them, a series of simulations with different number of epochs were computed. Each epoch of random walk was modeled with uniform step size and uniform step direction algorithms since our goal for this second task is to compare performance using different methods, therefore the code was simplified and unnecessary parts were removed for the sake of coherence.

V. Results

A. Sequential Model Results

The results were gathered from simulations with seemingly corresponding parameters. Note that the fixed steps size algorithm has the same value as the mean step size from the Gaussian step size algorithm. As a consequence, certain results may appear to be better only because the parameters favor one algorithm instead of another. We observe that some algorithm may have undesirable result with one set of parameters, but perform better with different parameters. The results for each combination of algorithms are shown in Table I.

The initial averaged results show that Forward Check and Backtracking gives the best results. As for the step size algorithms, Exponential step size gave the best results. In an attempt to further improve the algorithm with the best results, additional simulations were done with the only difference being that the agent now has different probabilities for sideways movement. Meaning the agent's probability for going right is bigger than the probability for going left. The reasoning for this kind of change is to observe if the agent performs better when moving in a spiral - like pattern. This raises the question: How big of a difference should there be between the probabilities for moving right and moving left. Additional tests were made to assess which right to left probability ratio performed the best. Considering that the main idea of the algorithm is to keep moving forward, the probability to move right should not exceed the probability to move forward. A series of simulations were done with different probabilities for right and left movement, the range of these values varies from: P(right) = 0.25, P(left) = 0.25; to P(right) = 0.35, P(left) = 0.15; incremented (decremented) by 0.01.

Regarding the parameters for the algorithms, particularly the step size algorithms, it is necessary to evaluate and compare how the results change when adjusting the parameters for each algorithm. For that purpose, a parameter analysis was conducted for the best (Exponential) and worst (Uniform) step size algorithms. The parameter for the Uniform step size is simply the fixed steps size (number of coordinate points), while for the Exponential step size the parameter is β ($\beta = 1/\lambda$). The evaluation results of each algorithm are shown in Figure 1.

TABLE I
Score for each Algorithm combination

Direction	Step Size	First found	Collected food score	Score
Uniform	Uniform	103.44	1.76	**0.017**
Uniform	Gaussian	48.63	47.56	**0.977**
Uniform	Exponential	45.34	73.81	**1.628**
Backtrack	Uniform	91.76	3.61	**0.039**
Backtrack	Gaussian	44.95	77.06	**1.714**
Backtrack	Exponential	38.10	120.55	**3.163**
FC+BT	Uniform	92.12	4.38	**0.047**
FC+BT	Gaussian	40.38	107.18	**2.654**
FC+BT	Exponential	35.57	141.84	**3.987**
FC+BT+RS	Uniform	87.89	4.04	**0.046**
FC+BT+RS	Gaussian	42.47	81.90	**1.928**
FC+BT+RS	Exponential	38.05	121.63	**3.196**

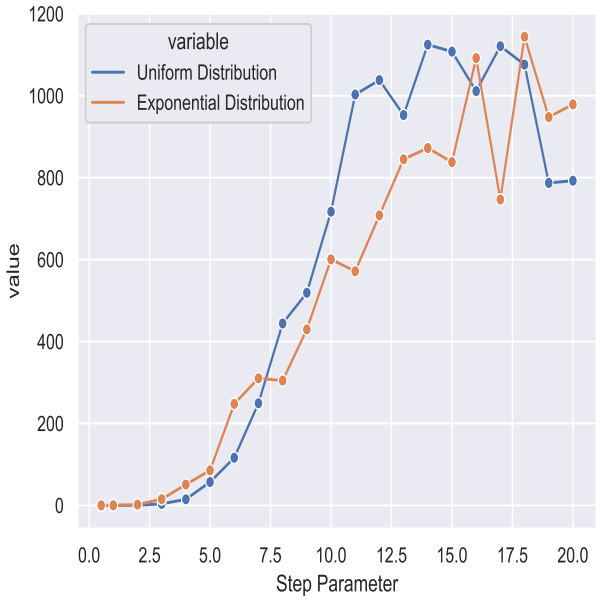

Fig. 1. Graphical representation of the uniform and exponential step size scores

B. Parallel Approach

The results from the parallel implementation of the uniform step size and uniform direction simulation are presented in Table II. A certain speed up in favour of the parallel approach was expected when reviewing completion time of simulations with larger amount of epochs compared to the sequential approach. Surprisingly the parallel function performed better regardless of the number of epochs. Considering that both the sequential function and parallel function were written in pure Python code and the fact that the parallel function was implemented with Numba, which in addition of enabling CUDA functionality in Python code, it also automatically (when using CUDA functionality) uses a compiler that translates Python and NumPy code into machine code very efficiently and quickly. Although the performance gap between the sequential and parallel model may not be entirely and only due to the parallel implementation, the computation speed up is hard to ignore when considering calculations performed on large amounts of data. The computation time for the sequential and parallel approach, and the appropriate speedup are shown in Table II.

FT + BT: Forward Check + Backtrack algorithm
FT + BT + RS: Forward Check + Backtrack algorithm + Right skew

TABLE II
SEQUENTIAL AND PARALLEL COMPUTATION TIME

Number of epochs	Sequential time (seconds)	Parallel time (seconds)	Computational Speedup
1	1.938	0.983	1.971
2	3.857	1.124	3.432
4	7.871	1.020	7.719
6	11.576	1.052	11.008
8	15.365	1.154	13.310
10	19.316	1.005	19.217
20	38.155	1.131	33.726
30	57.169	1.035	55.251
40	76.250	1.105	68.984
50	95.115	1.171	81.200

TABLE III
SCORE FOR DIFFERENT RIGHT AND LEFT MOVEMENT PROBABILITIES

Probability		Food collected in (steps)					
P(left)	P(right)	50	100	200	500	First found	Score
0.25	0.25	1.39	2.49	4.48	9.10	35.013	**4.037**
0.24	0.26	1.40	2.47	4.37	8.90	37.924	**3.546**
0.23	0.27	1.44	2.51	4.29	8.94	36.790	**3.772**
0.22	0.28	1.40	2.43	4.39	9.05	37.509	**3.594**
0.21	0.29	1.41	2.43	4.29	8.95	37.519	**3.491**
0.2	0.3	1.39	2.46	4.37	8.83	36.285	**3.625**
0.19	0.31	1.38	2.38	4.20	8.87	36.664	**3.343**
0.18	0.32	1.41	2.44	4.25	8.74	39.502	**3.230**
0.17	0.33	1.39	2.45	4.29	8.81	36.742	**3.498**
0.16	0.34	1.42	2.44	4.31	8.73	37.216	**3.502**
0.15	0.35	1.42	2.48	4.26	8.73	35.109	**3.710**

VI. DISCUSSION

Regarding the different step size algorithms, when analysing each algorithm's score and actual step size, it is evident that simulations with statistically bigger step size performed better. The initial simulations were completed with uniform step size of 1, step size sampled from a Gaussian distribution with $\mu = 2$ and standard deviation $\sigma = 1$ and step size sampled from a Exponential distribution with a parameter $\beta = 2$. Concerning the exponential step size algorithm, the parameter β does not give an intuitive meaning to how big the actual sample is. With a further review of the exponential function with a parameter $\beta = 2$, it is concluded that the samples taken from this distribution have a mean of 2 and standard deviation of 2. Considering that the model is configured to only select non-negative step size samples from each distribution function, and round them to an integer, the differences between the Gaussian and Exponential algorithm (mainly the difference in standard deviation) are more noticeable when sampling numbers bigger than the mean.

VII. CONCLUSION

It is apparent from the simulation outcome that when considering a step size algorithm, the best results are produced from algorithms that generate relatively bigger step size samples, which in this case is the Exponential step size algorithm. Regarding the direction algorithms, the Backtrack algorithm 'guides' the agent by avoiding the recently visited places in order to avoid redundancy in the search. The Forward Check and Backtrack algorithm give the searching agent a sense of direction when traversing the field in order to avoid a situation where the agent's movement is indecisive, consequently producing the best results. Evidently adding a right skew to the Forward Check and Backtrack algorithm resulted in worse performance.

Another way of interpreting the results is the trade-off between the agent's score (how well the agent performed) and the complexity of the agent. For example, the backtracking algorithm performed better than the uniform algorithm at the expense of complexity (storing the agent's previous position in memory). In order to maximise the area covered, the agent must keep track of the locations that are already visited. A perfect searching agent keeps all visited locations in memory and avoids visiting them again, but this solution is too complex when considering a relatively large domain that needs to be searched. It is interesting to note that the FT + BT and backtrack algorithm are not exceedingly dissimilar in terms of the agent's stored memory. Both algorithms only store the agent's last position in memory, the only difference between the algorithms is the different method of calculating the direction probabilities.

The results in this article provide us with an general idea of which combination of algorithms perform better, however, further investigation is desirable when considering each algorithms probability ratios and parameters.

Regarding the parallel simulation, the results clearly show that this kind of an approach has potential for increasing the number of epochs in the simulation without sacrificing time spent on computation. Given the sequential nature of the problem it is hard to fully utilize the GPU architecture to our advantage, nevertheless it is definitely beneficial in terms of computation time of the simulation.

REFERENCES

[1] V. Tejedor, R. Voituriez, and O. Bénichou, "Optimizing persistent random searches," *Physical review letters*, vol. 108, no. 8, p. 088103, 2012.

[2] G. Viswanathan, V. Afanasyev, S. V. Buldyrev, S. Havlin, M. da Luz, E. Raposo, and H. Stanley, "Lévy flights in random searches," *Physica A: Statistical Mechanics and its Applications*, vol. 282, no. 1, pp. 1–12, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437100000716

[3] G. Viswanathan, F. Bartumeus, S. V. Buldyrev, J. Catalan, U. Fulco, S. Havlin, M. da Luz, M. Lyra, E. Raposo, and H. Eugene Stanley, "Lévy flight random searches in biological phenomena," *Physica A: Statistical Mechanics and its Applications*, vol. 314, no. 1, pp. 208–213, 2002, horizons in Complex Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437102011573

[4] S. Redner, *A Guide to First-Passage Processes*. Cambridge University Press, 2001.

[5] S. Condamin, O. Bénichou, V. Tejedor, R. Voituriez, and J. Klafter, "First-passage times in complex scale-invariant media," *Nature*, vol. 450, no. 7166, pp. 77–80, Nov 2007. [Online]. Available: https://doi.org/10.1038/nature06201

[6] A. J. Bray, S. N. Majumdar, and G. Schehr, "Persistence and first-passage properties in nonequilibrium systems," *Advances in Physics*, vol. 62, no. 3, pp. 225–361, 2013. [Online]. Available: https://doi.org/10.1080/00018732.2013.803819

[7] M. Chupeau, O. Bénichou, and R. Voituriez, "Cover times of random searches," *Nature Physics*, vol. 11, no. 10, pp. 844–847, Oct 2015. [Online]. Available: https://doi.org/10.1038/nphys3413