

A Report on Stochastic Graph Exploration with Limited Resources

Artúr Bánlaki, Ábel Nagy, Dániel Kelemen

December 2025

1 Introduction

The report was based on the article "***Stochastic Graph Exploration with Limited Resources***" written by Ilan Reuven Cohen [1]. In short the paper analyzed a problem for stochastic graph exploration where we aim to maximize the reward on the reached vertices, but staying under the budget. The rewards are known for the vertices, but unknown for the edges. While an *adaptive* strategy might yield the optimal reward, the authors argued that it is impractical in parallel settings and for some large graphs. They examined 2 families of graphs and provided non-adaptive algorithms to solve it with similar performance.

We implemented a simulation framework in **Python** and used large datasets to validate the claims of the paper.

2 The Stochastic Graph Exploration Model

2.1 Problem Setting

An instance of the stochastic exploration problem consists of:

- A directed tree $G = (V, E)$ rooted at r .
- Each vertex $v \in V$ has a deterministic reward $R(v)$.
- Each edge $e \in E$ has a random cost drawn from a known distribution $\pi(e)$.
- A total budget B limits how large the costs can be.

A strategy (adaptive or non-adaptive) constructs a connected subtree step by step $F \subset V$ containing the root by probing edges one at a time. When the cost of a newly probed edge pushes the total above B , the process stops immediately, not counting the critical vertex's reward.

The goal is to **maximize expected total reward** from all vertices successfully added before halting.

2.2 Adaptive vs. Non-Adaptive Strategies

- **Adaptive strategy:** chooses the next edge based on outcomes seen so far.
- **Non-adaptive strategy:** Probes edges on a fixed order, we call this a list strategy.

The author's work shows that some **non-adaptive strategies require augmentation**. Otherwise if the budget remains only budget B , the adaptivity gap can be $\Omega(n)$. But with a constant budget augmentation constant reward approximation is possible for the 2 graph families:

1. **Spider graphs** (section 4): starting from a central root, much like a star graph, but with the arms having multiple vertices in succession. These graphs usually appear in scheduling.
2. **Bounded depth trees** (section 5): directed tree graph where every vertex is reachable from the root. These graphs are common in social networks (6 handshake theory).

2.3 Performance Guarantees

In the paper **Lemma 2** uses linear programming to provide an upper bound for *any* adaptive strategy. The author's prove that their algorithm can perform on average at least $\alpha \cdot R$ with $\beta \cdot B$ budget where R is the maximal expected reward from **Lemma 2**. Specifically the authors claim the following:

1. Spider graphs: $O(\frac{24}{\epsilon}, 1 + \epsilon)$ algorithm for $\epsilon < 1$
2. Bounded: $\alpha = \frac{16(B+1)}{\epsilon^2}$ for B budget.

3 Code Overview

We made an implementation of the algorithms described in the paper. This included classes that allowed the construction and handling of graphs of the given properties with randomized edge weights and rewards. Linear programming tasks are calculated with PuLP [3].

In addition, we made tools to aid our work with the algorithm: a way to generate, save, and load datasets for testing purposes, and a way to visualize how the algorithm works on graphs.

We also used Matplotlib [2] to visualize the testing results.

4 Spider Graphs

4.1 Structure and Importance

A spider graph (or spider tree) consists of a root and multiple linear legs, each a simple path. Spider graphs are used to depict many practical scheduling and exploration scenarios where tasks or nodes must be visited in a fixed order.

4.2 Algorithmic Approach

The algorithm divides vertices into:

- **Risky vertices:** Nodes whose expected path cost from root is higher than $\frac{B}{2}$. Only one such leg can be probed with high success probability; thus the algorithm picks a leg with a set chance based on LP-derived values.
- **Non-risky vertices:** Nodes with safer paths, where the expected cost of exploration to the node is less than $0.5B$. These are handled using a set strategy derived from the structure of the LP relaxation.

The algorithm outputs the better expected performer of:

- a list strategy for risky vertices (budget $(1 + \epsilon)B$) for some $0 < \epsilon < 1$
- a set strategy for non-risky vertices.

4.3 Experimental Dataset

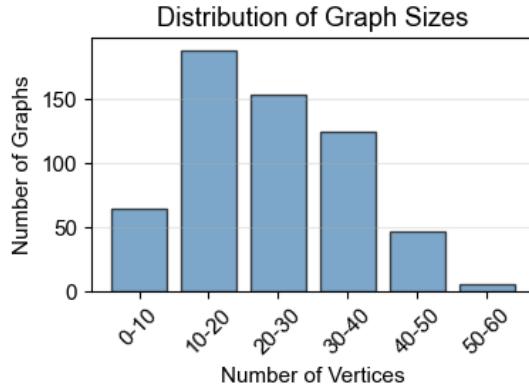


Figure 1: Spider graph experimental dataset

We randomly generated roughly 600 graphs of varying sizes to run the implemented algorithm with. The distribution of the graph sizes is shown in fig. 1. We first generate uniformly the number of the legs 2 – 10 and each with the length from 1 – 7.

4.4 Testing the theorem

We first measure how the $0 < \epsilon < 1$ parameter affects the output of the algorithm by running each graph through it with 6 distinct

$$\epsilon \in \{0.01, 0.1, 0.25, 0.5, 0.75, 0.99\}$$

After the non-adaptive strategy is calculated, we run the actual exploration 10 times and use the obtained reward values to calculate how they compares to the theoretical minimum bound of the non-adaptive algorithm accounting for the number of vertices in the graph. This theoretical minimum is calculated as $\frac{\Phi_{I,1}(t)}{\alpha}$ due to the definition given in (α, β) -approximations.

The author defines α for his non-adaptive spider graph algorithm as $\frac{24}{\epsilon}$ in his *Theorem 11*[1].

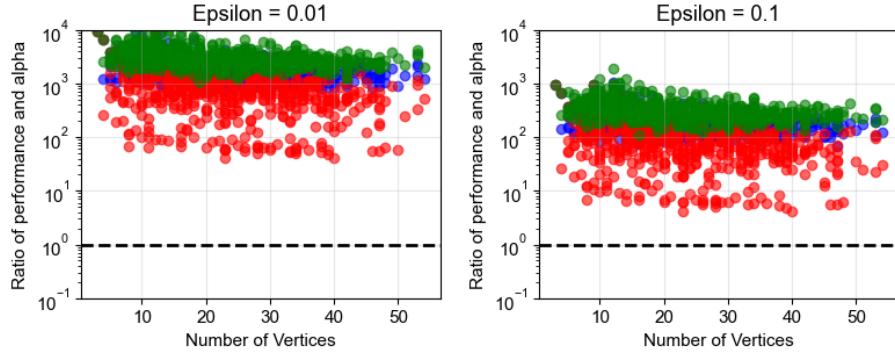


Figure 2: Proposed minimal vs actual minimum, mean and maximum gain for low ϵ values

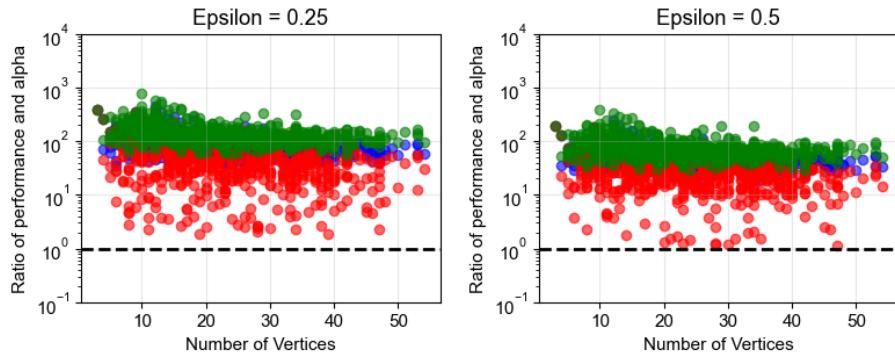


Figure 3: Proposed minimal vs actual minimum, mean and maximum gain for medium ϵ values

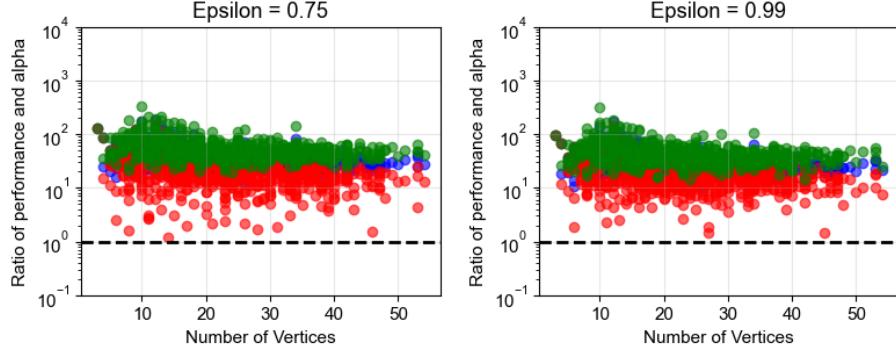


Figure 4: Proposed minimal vs actual minimum, mean and maximum gain for high ϵ values

The figures fig. 2,fig. 3,fig. 4 show a logarithmic scale on the y axis, which is the actual obtained reward divided by the theoretical minimum. This shows that for lower ϵ values, a larger multiplicate of the minimal fraction is obtained, which is important, but since the theoretical minimum contains ϵ as a multiplier, this does not show a direct improvement for actual rewards obtained in case of lower ϵ values.

What these graphs show, is that for even the worst actual performance of the strategies obtained from the algorithm - which are shown as red dots - the fraction does not tend below 1 - which is shown as a dotted line. This proves that for all epsilon values the theorem holds, and the theoretical minimum stays an actual minimum to the performance.

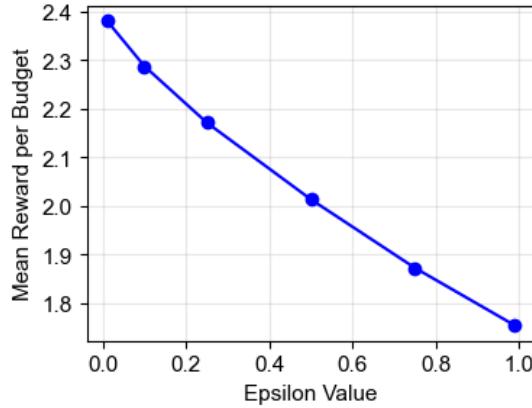


Figure 5: Rewards obtained adjusted for ϵ

As for the β value of the constant-constant approximation, the author defines

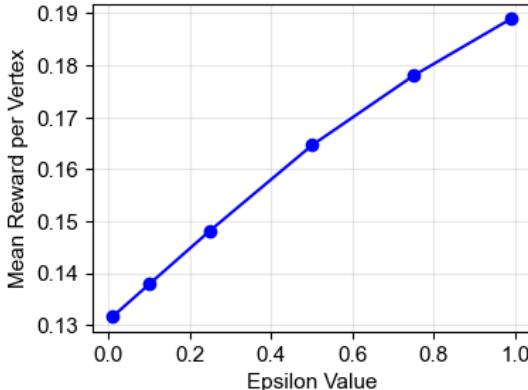


Figure 6: Rewards obtained per vertex count adjusted for ϵ

it as $1 + \epsilon$, meaning the budget needs augmentation, and this the resulting rewards obtained need to be adjusted for it.

Figure fig. 5 shows that for the experimental graphs used, we obtained less reward per budget used for higher values of ϵ .

Figure fig. 6 shows us that for each vertex a graph contains, more of the rewards could be obtained for higher values of ϵ .

In conclusion, the reward per unit of budget decreased by 25% and reward per vertex increased by 32% linearly over the possible values of ϵ . This means, that for non-edge-case scenarios, all ϵ values work similarly with budget being traded for rewards. **This is in line with what we can infer from *Theorem 11*[1].**

4.5 Result Analysis

We also tried to analyze how the graphs layout influences the obtained rewards, regardless of the values of ϵ . Figure fig. 7 shows

1. that the number of legs does not have a major influence on the total reward gained per graph. The increase in mean rewards obtained is minimal, while there are outcrops along the data.
2. that the depth of graphs does correlate to the total reward gained per graph. In our instances we see a 89% increase in mean rewards obtained while going from 1 to 7 vertices of max depth. This suggests a trend, but needs to be studied further to gain appropriate conclusions. We think this is a result of being presented with more options throughout the legs, since more often than not, only a single leg may be explored using the non-adaptive algorithm.

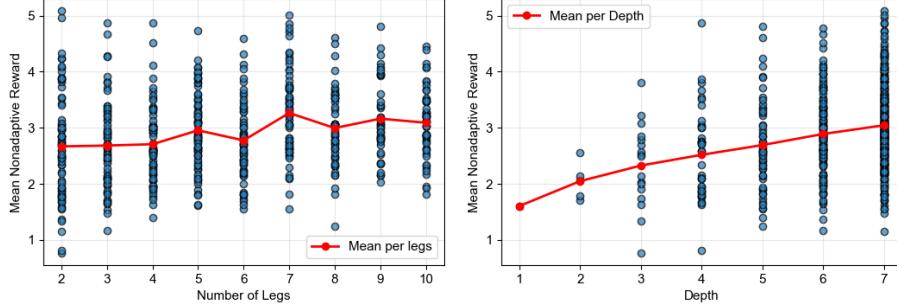


Figure 7: Rewards obtained per different graph features

5 Bounded Weighted Depth Trees

5.1 Structure and Importance

A bounded weighted depth tree (BWDT) is a tree that is bound in the cost of the path from the root to any given node in it. The formal definition is that a tree is (β, B) -bounded weighted depth if

$$\forall v \in V : \mu_B(D(v)) \leq \beta$$

In many real networks (e.g., social graphs), the expected sum of edge costs along any root-to-node path is naturally bounded and as such can be described as a BWDT.

5.2 Tree Decomposition Algorithm

The key difficulty is that the LP's support tree (T , where all vertices have a chance of being probed) may have total expected cost much larger than B . The author resolves this via a tree decomposition algorithm that cuts this tree into multiple subtrees, all of which have a total expected cost of exploration that does not exceed an arbitrary number α (not to be confused with the α of (α, β) -approximation). Upon successful tree decomposition we receive the result of the non-adaptive strategy as a set of set strategies, where any one of the subtrees can be probed and explored.

5.3 Experimental Dataset

In *Theorem 13*[1] the author makes a constraint for the BWTDs that his algorithm can guarantee the constant-constant approximation for. This constraint is dependent on the value of ϵ , and says that only $(B(1 - \epsilon), B)$ -bound weighted depth trees can be used.

Thus we randomly generated 100 graphs for each value of

$$\epsilon \in \{0.05, 0.1, 0.3, 0.5, 0.7\}$$

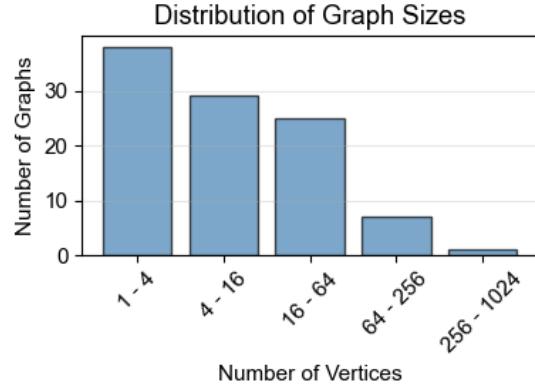


Figure 8: BWDT experimental dataset

with depths from $1 - 7$ and varying number of vertices to run the implemented algorithm on. The distribution of the graph sizes is shown in fig. 8.

5.4 Testing the theorem

We already established that one of the constraints of the theorem is in the BWTDs structure. *Theorem 13*[1] promises a $(\frac{16(B+1)}{\epsilon^2}, B)$ -approximation of the adaptive algorithms for these instances of BWDT, provided the tree decomposition algorithm we use:

$$\alpha = 1 - \epsilon/2$$

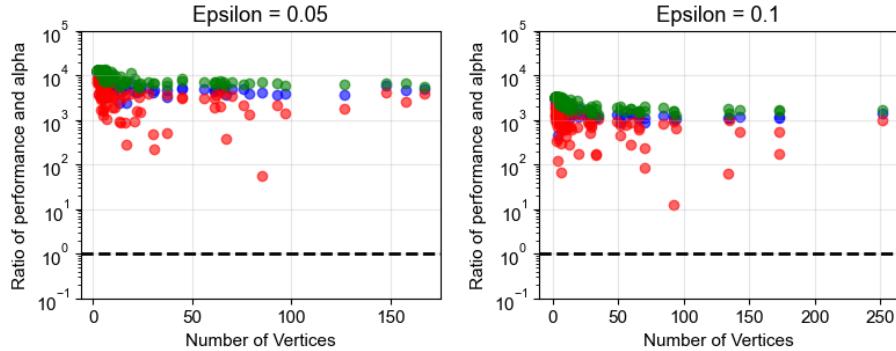


Figure 9: Proposed minimal vs actual minimum, mean and maximum gain for low ϵ values

Figures fig. 9, fig. 10, fig. 11 show that just like with the spider graphs section 4 a lower multiplicate of the minimum theoretical limit of the obtainable rewards is actually obtained with higher ϵ values. At 0.7 the mean obtained values are

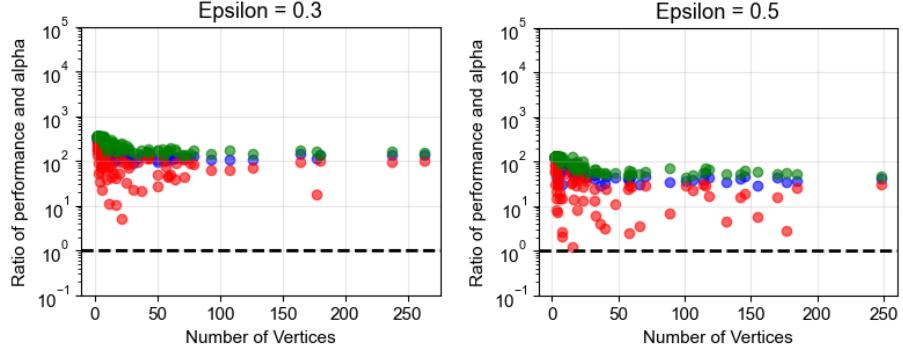


Figure 10: Proposed minimal vs actual minimum, mean and maximum gain for medium ϵ values

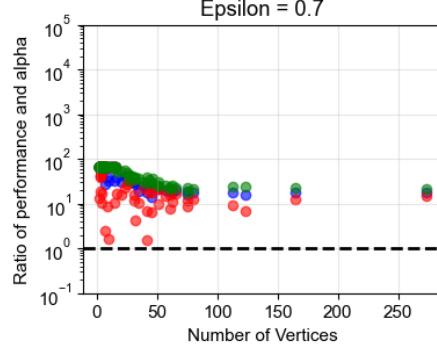


Figure 11: Proposed minimal vs actual minimum, mean and maximum gain for high ϵ values

close to only 50 times the minimum, and the minimal values are reaching the 1 limit, which the theorem guarantees. This is however not a bad thing, since the theoretical minimal limit of the non-adaptive algorithm is proportional to ϵ^2 so a lower multiplicate still means a higher total, as is shown in fig. 12. Here we can observe an almost linear correlation between higher ϵ values and obtained mean rewards, which go up by 62% over the observed range. **This is in line with what we can infer from the definition of BWDTs section 5.1 and the article's Theorem 13[1], where we trade off less constrained BWDT instances for more obtainable rewards at lower ϵ values, and the opposite at high values.**

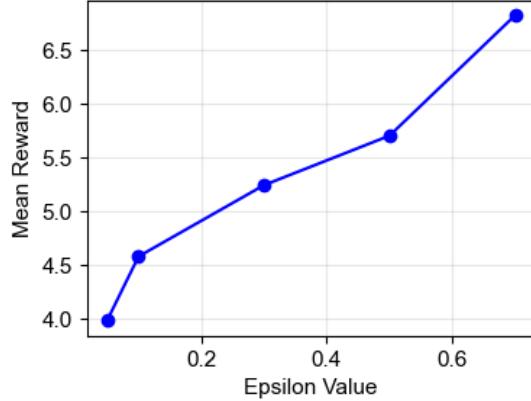


Figure 12: Rewards obtained for ϵ values

5.5 Result Analysis

We analyzed the results from other, independent factors from ϵ to measure how the algorithm operates on these.

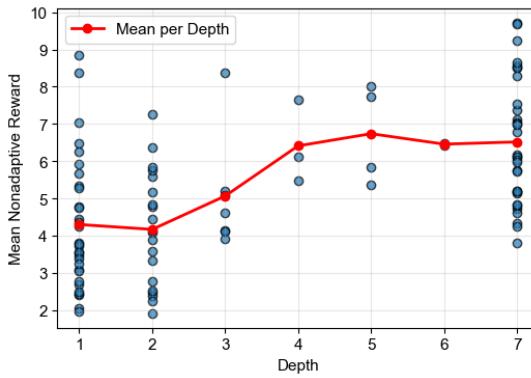


Figure 13: Rewards obtained per depth

In figure fig. 13 we see how depth and the obtained mean rewards correlate, which seems incremental, but a correct line is hard to determine due to the sparse data from the depth values in the middle. However, since the 1, 2 and 7 depth graphs are numerous, we can take an educated guess and say that the algorithm works better for deeper graphs, likely due to them being more numerous in vertices and thus having more options to choose from.

6 Conclusion

This report has combined the author's theoretical framework with real-life results from our program implementation. The main points of the article have been determined to hold true, with extensive documentation on how the different values of ϵ affect each algorithm and graph type.

Further studies could include larger generated sample sizes and also graphs with more vertices to confirm the theories for higher n values as well, but inferring from our sample size, it can still be said, that the results were constant in all sizes of graphs.

References

- [1] COHEN, I. R. Stochastic graph exploration with limited resources. In *International Workshop on Approximation and Online Algorithms* (2022), Springer, pp. 172–189.
- [2] DEVELOPMENT TEAM, T. M. Matplotlib, 2021. Accessed: October 2025.
- [3] FOUNDATION, P. S. Pulp, 2025.