Dynamic All-Pairs Reachability

Mithilesh Vaidya

1 Introduction

In this work, we implement the dynamic all pairs reachability (DAPR) algorithm for general graphs studied in chapters 2 and 3 of the lecture notes (Key result in [3]). In particular, we study the performance of the algorithm as a function of the size of the graph and compare the experimental results with the theoretical complexity. Since the studied algorithm is correct with a high probability, another interesting direction we have touched upon is the failure probability as a function of the parameters of the algorithm. Moreover, experiments have been carried out on two distinct graph structures, in order to study if graphs with certain properties benefit more from the proposed algorithm.

2 Setup

In this section, we describe the baseline, graph structure and failure probability in more detail.

2.1 Baseline

Floyd-Warshall (FW) [4] algorithm is the baseline algorithm. It has a complexity of $O(n^3)$ where n is the number of nodes in the graph as opposed to the proposed algorithm, which has a complexity of $O(n^2 logn)$. The baseline is implemented for two reasons:

1. As the name suggests, I will be comparing the performance of the studied algorithm with FW. Although the studied algorithm has a superior theoretical complexity than FW, it is possible that its constant factor is large enough for it to perform slower than the naive algorithm for small values of n. Hence, it can help determine the threshold n^*, beyond which the theoretical speedup starts to reap benefits.

2. Since Floyd-Warshall is guaranteed to be correct, we use its results to analyse the failure probability of the proposed algorithm.

2.2 Failure probability

The proposed algorithm holds with a given probability, which is determined by the size of the field p. We have studied in class that failure probability $\frac{2n}{p}$ which can me made arbitrarily small by setting $p \approx n^k$ (for some large k). The goal is verify this result experimentally. We can use the outputs of the FW algorithm for checking the correctness of our algorithm.

2.3 Graph structure

We generate graphs from two distinct families:

- Erdos–Renyi model [2]: In such graphs, each edge has a fixed probability of being present (say p), independently of the other edges. This is a naive procedure for constructing the graph. We vary p and check the failure rate.

- Barabási–Albert (BA) model [1]: In such graphs, new nodes are connected with m existing nodes with probability proportional to the degree of the existing node. These are more representative of several natural and human-made systems, including the Internet, the World Wide Web, citation networks, and some social networks. They contain few nodes (called hubs) with unusually high degree as compared to the other nodes of the network. It would

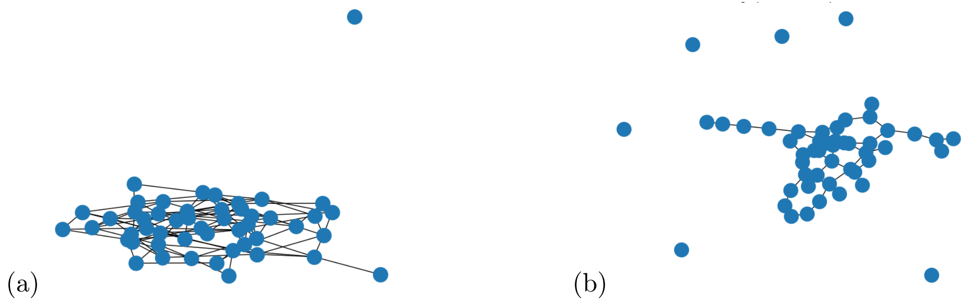

Figure 1: (a) ER graph with $n = 50, p = 0.1$ (b) ER graph with $n = 50, p = 0.05$

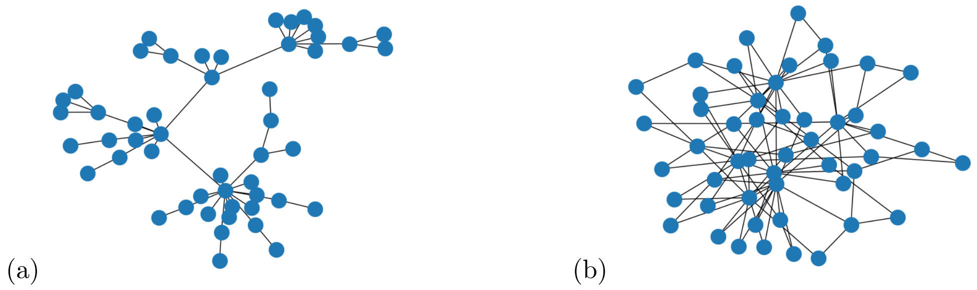

Figure 2: (a) BA graph with $n = 50, m = 1$ (b) BA graph with $n = 50, m = 2$

be interesting to analyse the failure probability, especially when an edge shared with such hubs are removed.

Some observations about ER and BA graphs:

- For ER graphs, as p increases, the graph quickly becomes one connected component. Hence, for such graphs, removing one edge will not change reachability for most nodes.
- For BA graphs, it becomes fully connected for very small values of m such as $m = 2$. Hence, the reachability will change only when m is small ($m = 1$) and the edge to one of the 'hub' is removed.

3 Experiments

In this section, we describe the parameters of the experiments and the pseudo-code.

3.1 Parameters

We try out the following values for n (number of nodes in graph):
$S_n = \{20, 50, 100, 200, 500, 1000, 5000, 10000\}$.
For ER graphs, the parameter p is picked from the set $S_p = \{0.005, 0.01, 0.05, 0.1\}$. The reason for the above choice of p is that as p increases, the graph quickly becomes fully connected as n increases.
For a given n and p, we generate a graph and then carry out 1 random change (insertion or deletions), followed by 1 random query. The source and target edges for both insertion/deletion and queries are picked randomly.

Each experiment is run K times and we report all values by averaging across these runs. Standard deviation is also tracked across these K trials.

3.2 Pseudocode

In this way, we compute the time taken for one update in case of both FW and DAPR. We also track the outputs of a random query so as to compute the error probability of DAPR.

Algorithm 1 Comparing FW and DAPR

```
procedure RUN($n, p, seed$)                                   ▷ One iteration of algorithm
    $random.seed \leftarrow seed$                             ▷ Set random seed
    $g = ER(n, p)$                                            ▷ Get ER graph with parameters n and p
    $c_s \leftarrow random(n), c_t \leftarrow random(n)$      ▷ Randomly pick edge to be modified
    $q_s \leftarrow random(n), q_t \leftarrow random(n)$      ▷ Randomly pick query source and target
    $f \leftarrow random\{0, 1\}$                             ▷ Randomly pick whether to insert or delete edge
    procedure FLOYD-WARSHALL($g, c_s, c_t, q_s, q_t, f$)      ▷ Run FW
        $t_s \leftarrow time()$                               ▷ Start counting time
        $g.Adj[c_s, c_t] \leftarrow f$                        ▷ Randomly change edge
        $dist \leftarrow FW(g)$                               ▷ Call Floyd-Warshall
        $t_e \leftarrow time()$                               ▷ End counter
        return $dist[q_s, q_t]! = inf, t_e - t_s$            ▷ Distance = inf denotes not reachable
    end procedure
    procedure DAPR($g, c_s, c_t, q_s, q_t, f$)                ▷ Run DAPR
        $t_s \leftarrow time()$                               ▷ Start counting time
        $DAPR.update(c_s, c_t, f)$                            ▷ Run DAPR algorithm to compute updated inverse
        $t_e \leftarrow time()$                               ▷ End counter
        return $DAPR.inverse[q_s, q_t] > 0, t_e - t_s$        ▷ Check if entry at inverse is non-zero
    end procedure
```

4 Results

In this section, we discuss the results of the experiments

4.1 Timing analysis

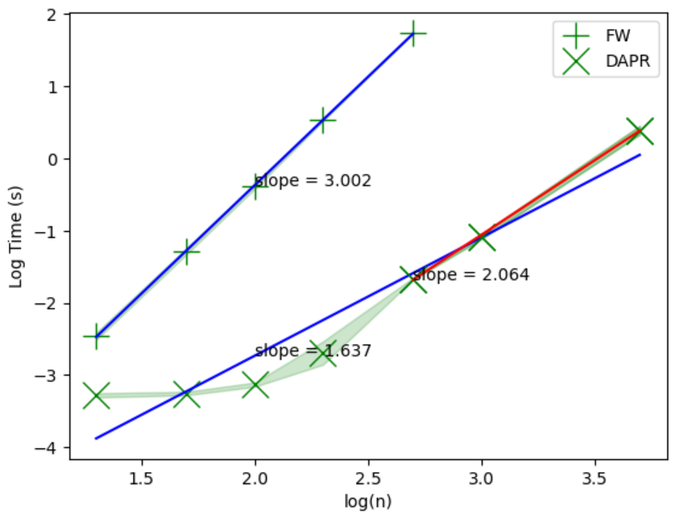

Figure 3: Average time taken for each update as a function of number of nodes n. Note that both axes are in log scale.

We first studied the time taken by both FW and DAPR. Results for $p = 0.1$ and $|F| = 1035301$ are shown in figure 4.1. Along with the exact results for varying n, we also plot the best fit line and annotate it with it's slope. Results are averaged across 100 trials and the mean time is reported. Shaded region denotes one standard deviation. From the figure, we can observe that:

- The slope of FW is approximately 3, which is expected because the algorithm is known to be of complexity $O(n^3)$. Note that running FW for $n > 1000$ took a significant amount of time and hence reliable results for a large number of samples could not be obtained. Hence, we do not report the results for such large values.

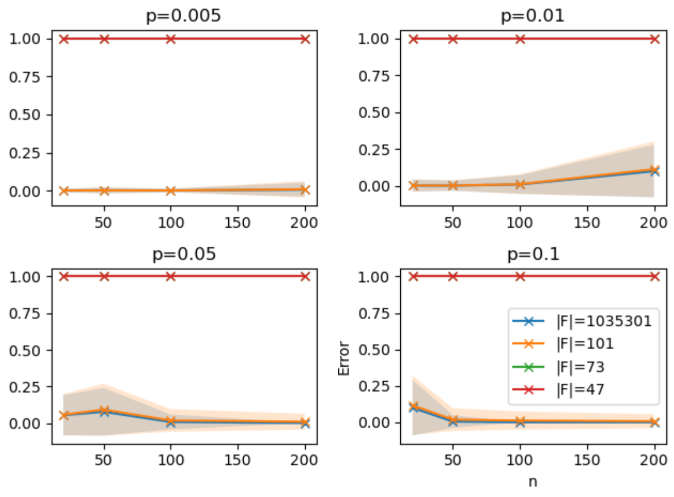

Figure 4: Error probability as a function of n, p for ER graphs. We have one plot for each value of p.

- The slope of DAPR is 1.637 when we consider all 7 points. However, the slope is approximately 2 for higher values of n. This indicates some noise for lower values of n but the slope approaches 2 for higher values of n as expected.

Also, varying the value of p does not given any significant changes in the trend. This is expected since time taken by both algorithms is a function of number of nodes n and not the number of edges (which are directly influenced by the value of p).

4.2 Failure probability

We analyse the failure probability of DAPR with FW serving as the ground truth. The experiments are carried out for four values of the field size F: 1035301, 101, 73 and 47. All four are primes. Values of n are picked from the set $\{20, 50, 100, 200\}$. For each value of n, p, results are averaged over 600 trials. We then report the average failure probability, along with the standard deviation. The primes are picked so as to obtain a smooth increase in the error probability as n increases.

4.2.1 ER graphs

Results for ER graphs are given in figure 4.2. We observe that:

- For $|F| = 47$, the error probability is always 1. This is expected since the size of the field is too low. The same holds for $|F| = 73$ (green overlaps with red and is hence not visible in the plot).
- As we increase $|F|$, we see a sudden jump in performance when $|F| = 101$. The performance is similar to $|F| = 1035301$, a much larger prime.
- For $p = 0.05, p = 0.1$, we see a higher error for low values of n while it becomes 0 as n increases. This contradicts the expected result of error being proportional to $\frac{n}{p}$.

4.2.2 BA graphs

Results for ER graphs are given in figure 4.2.2. We observe that:

- For field sizes of 47 and 73, the error is 1 for all n and m. This is not surprising since the values are small.
- The error probability for large field size (1035301) is 0 for all cases. However, for $|F| = 101$, we see an increases in the error as n increases (for m = 1). The increase is almost linear.

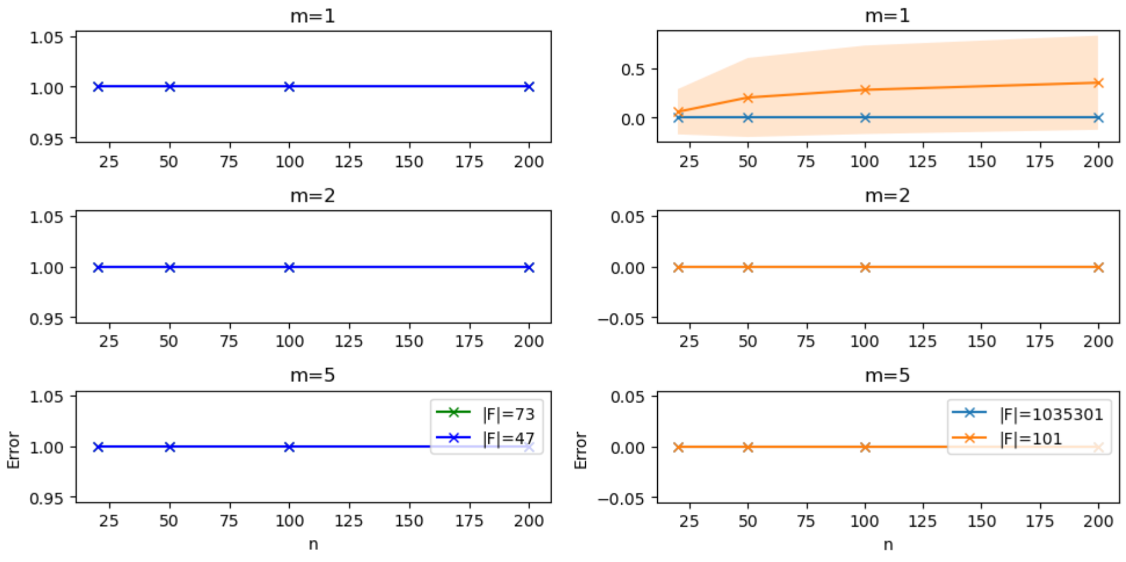

Figure 5: Error probability for BA graphs with varying m and field size $|F|$.

- For higher connectivity ($m = 2, 5$), the error is 0. This could be because removing an edge does not change the s-t reachability of the connected graph. That is the not the case for $m = 1$. This indicates that the DAPR algorithm fails when there does not a path while the algorithm indicates otherwise.

5 Conclusion

- We have experimentally verified the theoretical complexities of DAPR algorithm ($O(n^2)$) and Floyd-Warshall ($O(n^3)$).
- The results indicate that the error probability does not scale linearly with the expected $\frac{n}{p}$. This could be because of insufficient runs and can be examined more carefully in future work. We believe this section needs more careful examination in future work.

References

[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[2] Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[3] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 509–517. IEEE, 2004.

[4] Eric W Weisstein. Floyd-warshall algorithm. *https://mathworld. wolfram. com/*, 2008.