ATLAS Graph: Programming Task Report

Sushant Kumar (2024121004)
IIIT Hyderabad

February 10, 2025

Contents

1 Introduction **3**

2 Dataset Collection **3**
2.1 Sources . . . 3
2.2 Graph Construction and Visualization . . . 3

3 Task 1 - Graph Analysis **6**
3.1 Assumptions and Conventions . . . 6
3.2 First steps: Analyzing simple networks . . . 6
3.3 Finding Paths: What worked and what didn't... . . . 7
3.3.1 A reasoning thread . . . 7
3.3.2 Terminals that are not dead-ends . . . 9
3.4 Graph Reduction . . . 9
3.4.1 Countries . . . 9
3.4.2 Cities . . . 11
3.4.3 Combined . . . 11
3.5 Finding terminals . . . 11
3.5.1 Countries . . . 11
3.6 Graph Abstraction . . . 13
3.6.1 Countries . . . 14
3.6.2 Cities . . . 18
3.6.3 Combined . . . 18
3.7 A probabilistic approach . . . 18
3.7.1 Countries . . . 18

4 Task 2 - Community Detection **20**
4.1 Selection of Algorithms . . . 20
4.2 Lieden Community Detection . . . 21
4.2.1 Communities returned by Lieden . . . 22
4.2.2 Analysis of results . . . 23
4.3 Infomap . . . 23
4.3.1 Communities returned by Infomap . . . 24
4.3.2 Analysis of results . . . 25
4.4 Assessment metrics . . . 25

5 Conclusion **25**

Abstract

This report presents the analysis of the ATLAS Graph game as part of the Precog programming task. The objective was to construct, explore, and analyze graph-based properties and infer qualitative implications while identifying strategies to optimize player performance. Various graph-theoretic structures and properties were utilized to investigate effective moves.

1 Introduction

Attempted: Dataset Collection, Task 1, Task 2 completed in full. The bonus task was left due to time constraints.

Graph theory plays a crucial role in the analysis of complex networks. Two tasks were performed. The first task involves exploring a directed graph where nodes represent countries or cities, and edges correspond to country name transitions in the ATLAS game. Our goal was to:

- Construct the ATLAS graph based on the country names.
- Compute key graph properties such as degree distribution, centrality measures, and shortest paths.
- Implement game strategies to determine optimal moves.
- Simulate and analyze results over multiple game instances.

2 Dataset Collection

2.1 Sources

Countries: This wikipedia page consisting of a list of officially recognized countries was used. By "officially recognized" we mean that it is a member of the United Nations (like India) or has an observer status (like Palestine). We scraped the data (using `countries.py` program) from that page instead of the official UN website so that the names of the countries match with the common lingo like we don't want to be saying "Republic of India" instead of "India" while playing ATLAS.

Cities: No new, reliable, or complete data was available for the world's 500 most *densely* populated cities. We instead used this world population review page to download the data for the 500 most populated cities in the world (after getting permission from the task in charge).

2.2 Graph Construction and Visualization

The nodes correspond to the country (or city) names and there is a directed edge from X to Y if the last letter of X is the same as the first letter of Y.
The directed graph for countries with 195 nodes and 2033 arcs was constructed using Python's **NetworkX** library. We used the **spring_layout** function for visualization as

the graph was large and dense. We controlled the spacing and randomness by appropriately tuning k, iteration, and seed. We distributed our nodes over a coloring gradient based on degree centrality. The nodes with higher centrality appear darker. This was done for better visualization and to point out well-connected nodes as would be significant later in our analysis. Figure 1 illustrates the countries' ATLAS graph.

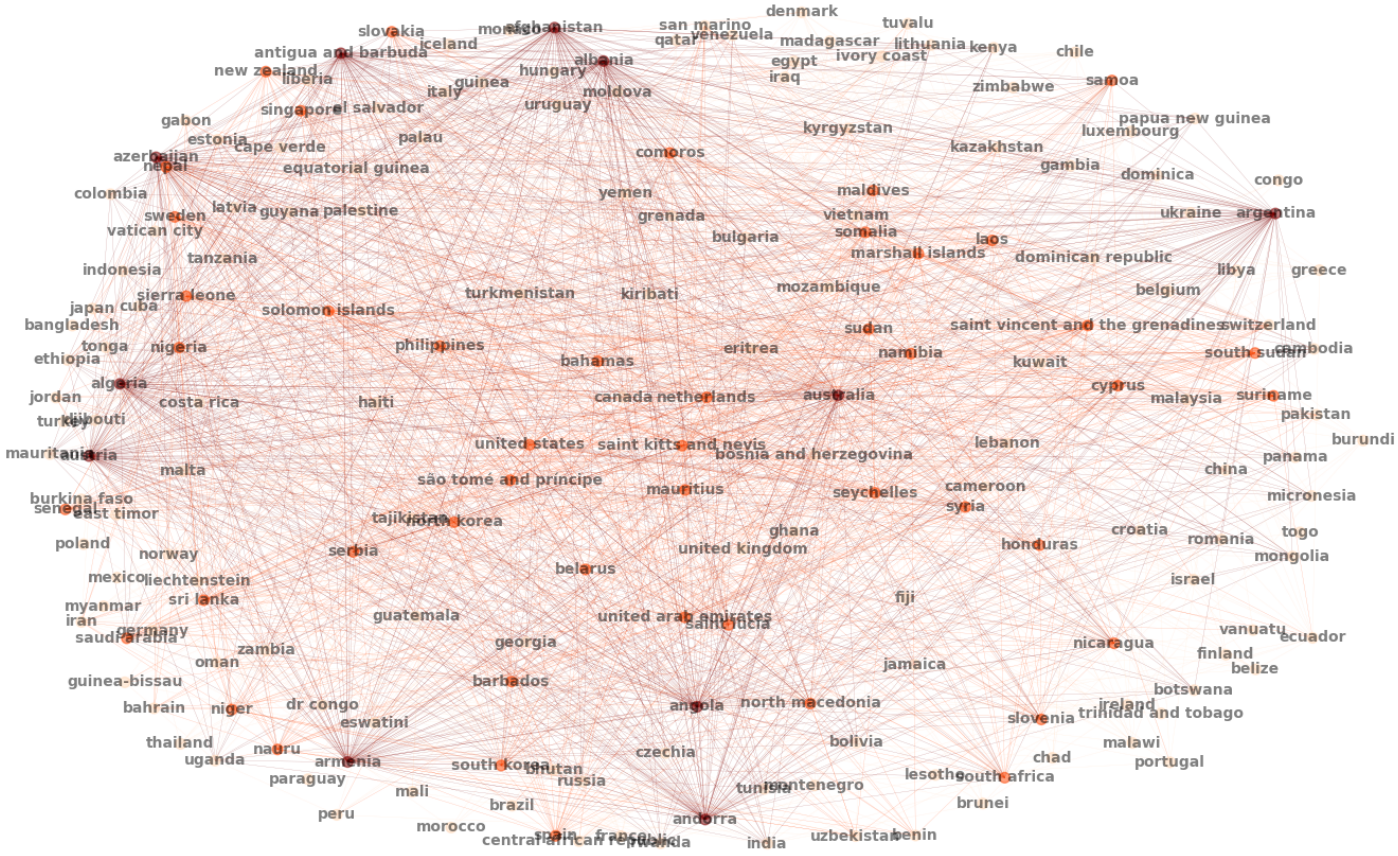

Figure 1: Countries ATLAS graph (check GitHub repository for the image file)

The directed graph for cities with 498 nodes and 9201 arcs was constructed in the same manner as the countries graph (498 instead of 500 because there's another Hyderabad in Pakistan and two Suzhou-named cities in China) Figure 2 illustrates the cities' ATLAS graph.

The directed graph for cities and countries combined was created with 692 nodes and 20410 edges with a similar methodology. Figure 3 illustrates the combined ATLAS graph.

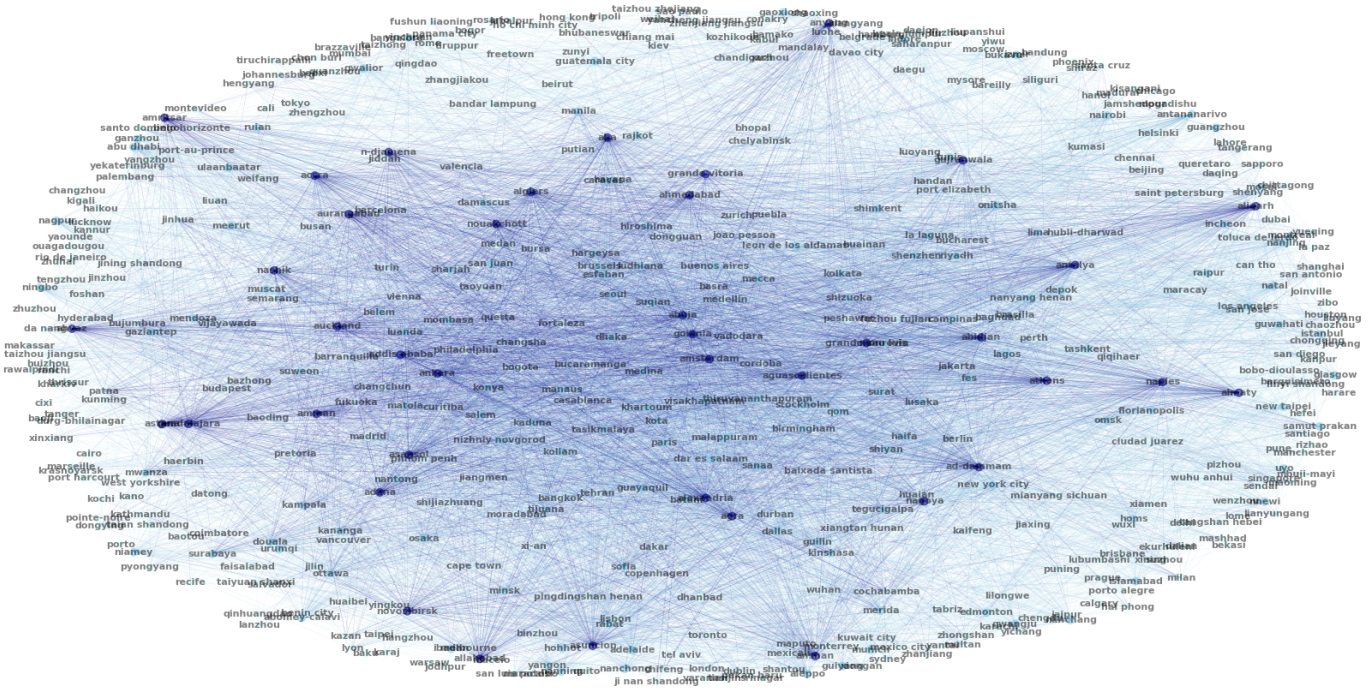

Figure 2: Cities ATLAS graph (check GitHub repository for the image file)

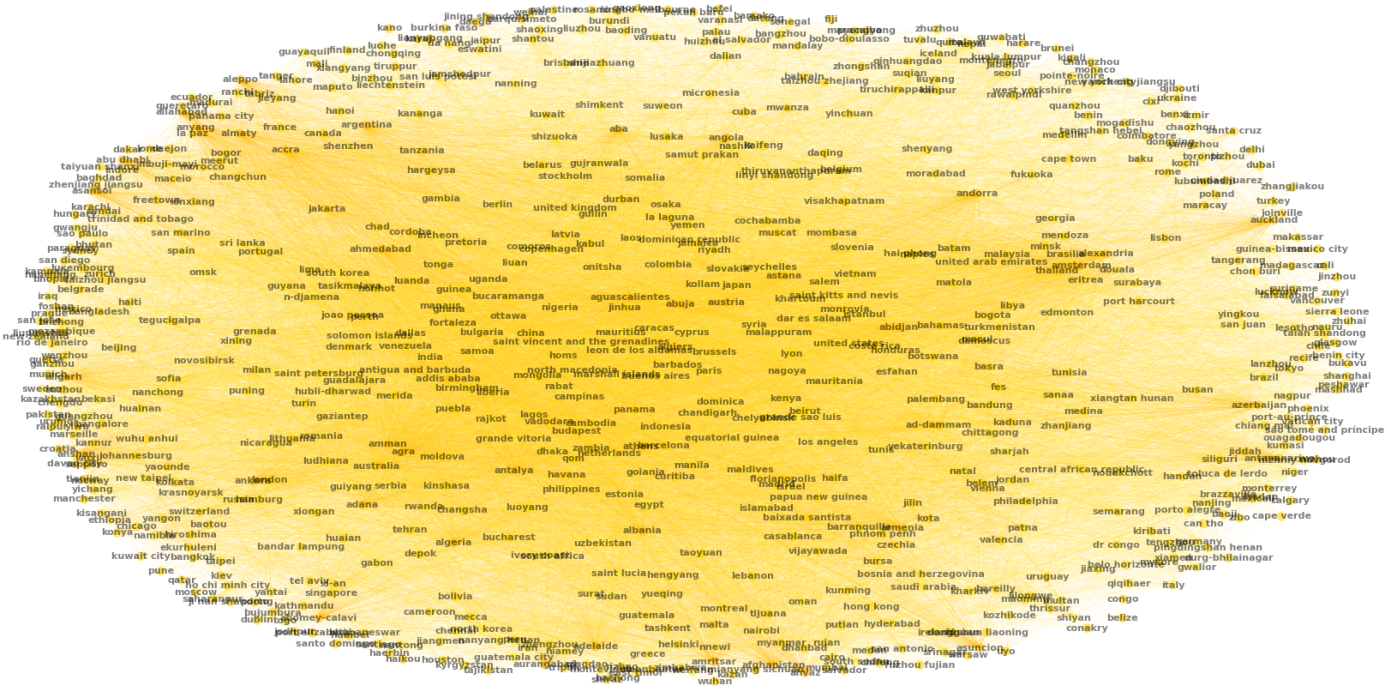

Figure 3: Combined ATLAS graph (check GitHub repository for the image file)

3 Task 1 - Graph Analysis

This section documents the data exploration process for the three graphs: countries, cities, and combined. Our analysis revolves around the central problem of finding strategies to win the ATLAS game in a two-player scenario. Inferences relevant to this goal were made and have been documented. Qualitative and quantitative measures have been used to support the findings. Some other insights not directly related to the problem have also been listed. The tools and structures utilized are fundamentally graph-theoretic.

3.1 Assumptions and Conventions

- The ATLAS game begins with a place starting with the letter A. This assumption has been made to ease the game simulation process and to be consistent with how the game is played in practice.
- The player who starts the game, that is, guesses the first place name is designated as *Player 1* and the opponent as *Player 2*. Further, *Player i* and *Player j* have been used while performing analysis agnostic to game initialization.
- A player cannot say the same name twice even if it belongs to two different places. For example, *Hyderabad* can be said only once even if it can refer to cities in two different countries.

3.2 First steps: Analyzing simple networks

Unlike some other abstract structures like vector spaces, graphs can be built inductively by adding nodes and edges to a smaller graph. Instead of directly attacking the problem on the given graph, we take a look at the simple networks to mathematically formulate the *winning criteria*.

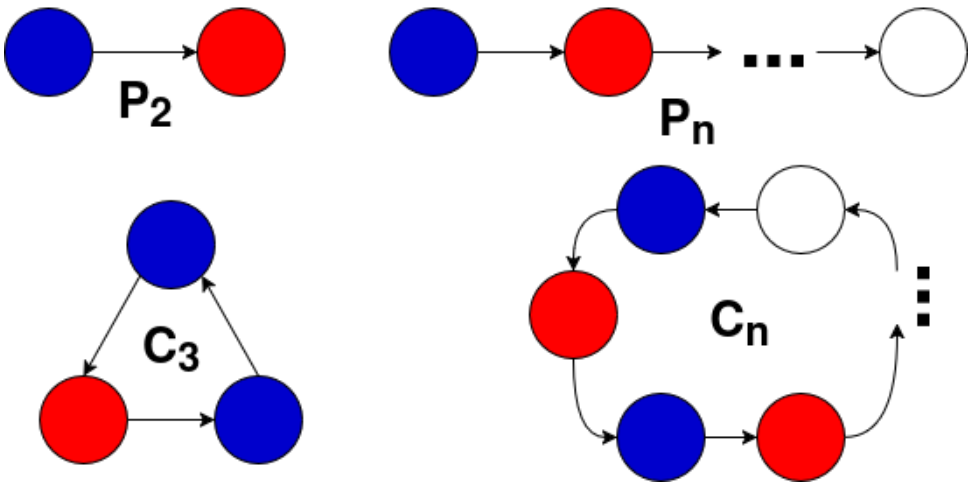

Figure 4: Blue corresponds to Player 1 and red to Player 2

Consider the graph as obtained by the traversal of Player 1 and Player 2 from some starting node to the node at which the ATLAS game decisively ends. Let's call this graph a *run*.
Observation 1: Every *run* is a simple path as any node is not allowed to repeat. If the path length (number of nodes) is odd then Player 1 wins else Player 2 wins. Further, the path should be maximal concerning the final node.

Observation 2: A *run* terminates at either (1) dead-end, i.e., a node with out-degree equal to zero, or (2) node that ONLY has out-edges looping back into the traversal.

We thus identify 2 defining elements of winning criteria - *run* length parity and termination condition. This motivates us to investigate our graphs for these two types of terminal nodes and the paths leading to them.

3.3 Finding Paths: What worked and what didn't...

3.3.1 A reasoning thread

Say we are *Player i* and want to defeat *Player j*

Q: What will finding terminal nodes accomplish?
A: It will give us an idea of where to lead the traversal. If we jump into a dead-end then there's no coming back for *Player j*. (S)he's out.

Q: You are at the initial state of the game with zero, one or two moves played. Assume you know all the dead-ends. How exactly would you "lead the traversal" to *j's* demise?
A: So there are multiple things to consider here...there can be many dead-ends and our preference for them can change dynamically depending on how *Player j* moves. We can find all odd-length paths from every node to all the dead-ends. The ends that have a higher number of shorter odd-length paths leading to them would have a higher preference on our list. Additionally, we can delete the paths whose nodes have been traversed by *j*.

Q: You can't do that. This amounts to brute forcing every single *run* which has O(n!) complexity. How do you "lead the traversal" to dead-ends again?
A: Yes. So we cannot compute all paths but we can certainly find all-pairs shortest paths in polynomial time. Aiming for "shortest" paths makes sense as it 1) decreases *j's* chance of deviating to other paths and 2) that is the only path we can compute efficiently (we can parameterize the distance by some k and find all k-length paths but that's also expensive)

Q: Though shortest paths can be found efficiently, it does not necessarily mean fewer deviations can be caused by *Player j*. Consider the scenario shown in Figure 5:
A: Yes though the shortest path reduces the number of nodes, it assumes that all nodes have an equal likelihood of letting j escape which is not true as evident from the example. We can refine our approach as follows: The goal is to find an odd path to the dead-end such that the nodes in the path do not let *player j* escape. This can be accomplished by checking if the even-numbered nodes have outdegree = 1 (no escape) or have a successor that precedes another even-numbered node in the path to force j back in the path. This

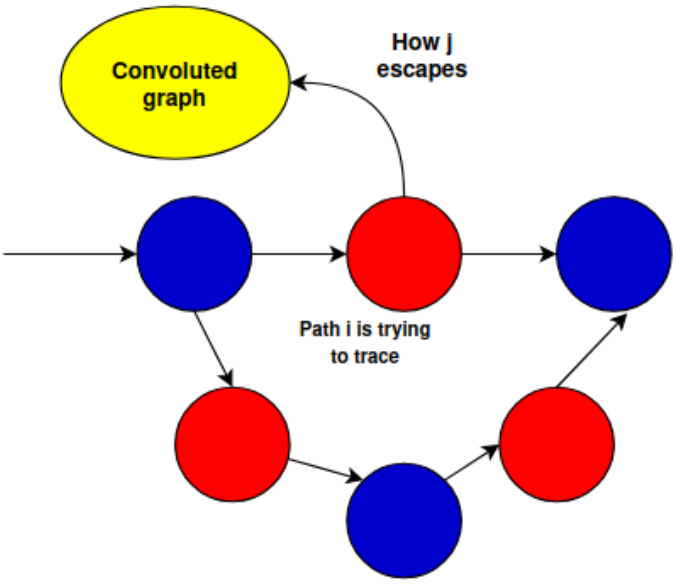

Figure 5: Blue corresponds to Player i and red to Player j

can also be done if there are an odd number of nodes before looping back into the path but that will allow for more degrees of freedom for *Player j* so we limit our evaluation to just 1 node. This is illustrated in Figure 6.

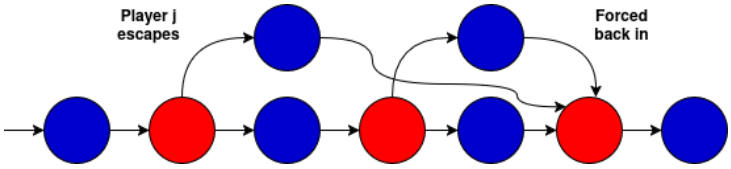

Figure 6: Blue corresponds to Player i and red to Player j

Q: How can such paths with minimal deviations and "loop-back" characteristics be found?
A: We can only find an approximation of such paths. We can run *breadth first search* on the reverse graph with dead-end as the root node and check for reachable nodes till some k^{th} level. Then we can examine each node in a particular path for the given characteristics. Accordingly, we select or reject that path and make the choices for a blue node which is in one of our accepted paths.

Q: Such an analysis is possible only if k is not comparable to n. Otherwise, it becomes as bad as enumerating over all paths.
A: This is obvious from the BFS tree where k is asymptotically $log(n)$ as we do not expect the BFS tree for a convoluted graph to be skewed. Further, we can limit k to the length of the **diameter**.

This pre exploratory reasoning leads us to the following:
Conclusion: Global assessment of paths leading to a terminal is infeasible. We can instead utilize **local** structures centered around terminals and extend our evaluation of its neighborhood using traversal techniques like the *breadth first search*.

3.3.2 Terminals that are not dead-ends

We now know that we aim to perform a local search around terminals to acquire some knowledge of paths for trapping our opponent. However, for this, we need to know our terminals in the first place!
Dead-ends are easy to find. They are just nodes with no outgoing edges. However, our analysis in section 3.2 also suggests another kind of stopping point - nodes that "loopback" into the traversal like in Figure 8

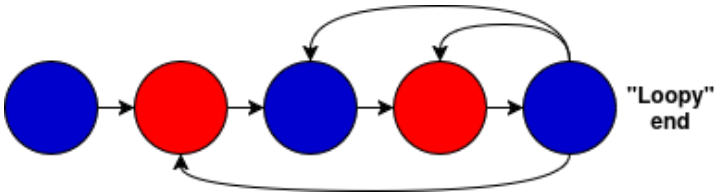

Figure 7: Blue corresponds to Player i and red to Player j

These *loopy ends* cannot be categorized by a single property like dead-ends. The goal for the upcoming sections would be to learn local features and structures (by graph theory) that help us find these terminals and a way to trap the opponent in them.

3.4 Graph Reduction

Observation 1: Nodes that (1) are **orphans** which means that they have no incoming edges and (2) have only orphans as their predecessors are never traversed unless they are the starting point.

Observation 2: All nodes can be reduced to a two-letter representation - $< firstletter >< lastletter >$, for example, India can be written as 'ia'. This follows from the fact that all connections in the graph are only determined by the first and last letters and the 'internal' letters only help in distinguishing nodes with the same start and end letters.

Our game of ATLAS starts with A. Thus all orphans not starting with A along with the type 2 nodes pointed out in observation 1 shall be removed as they are irrelevant.
Let G be the graph thus obtained. We further reduce G to G' as per observation 2. Note that every node of G' is a *clique* in G if the node is of type xx or an *independent set* in G if the node is of type xy where x and y are distinct letters.

3.4.1 Countries

The original graph had 195 nodes and 2033 edges. By our `scc_countries.py` program we make the following **inferences:**
(1) All countries starting with b, f, j, p, v, and z are orphans.
(2) All countries starting with 'h' have only 'bangladesh' as their predecessor which is an orphan.
(3) No orphan starts with 'a'.
(4) Removing all orphans and countries starting with the letter 'h' (list stored in `countries_inPlay.txt` leaves us with a graph that is **strongly connected**.

This reduces the number of nodes in the graph G to 153.

Conclusion: The reduced graph G has *no dead-ends*.
Justification: Note that since G is strongly connected there exists a path from every node u to every node v. A path cannot originate from a node with out-degree 0. Thus, there are no dead-ends.

Now we transform our graph into G'. As noted, xx in G' would mean a clique in G, and xy would mean an independent set. We obtain a graph with 75 nodes and 313 edges, a large reduction from 195 nodes and 2033 edges!

The clique trap
We observe that there is only one node aa in G' that corresponds to a clique in G. This means that there does not exist any country that starts and ends with the same letter except a. This clique structure can be exploited to develop a deterministic strategy as follows:

Claim: Player 1 has a winning guarantee if (s)he starts with a country beginning (and ending) with the letter a by forcing Player 2 to remain in the clique.

Proof: Player 1 starts with a node in the $aa - clique$.

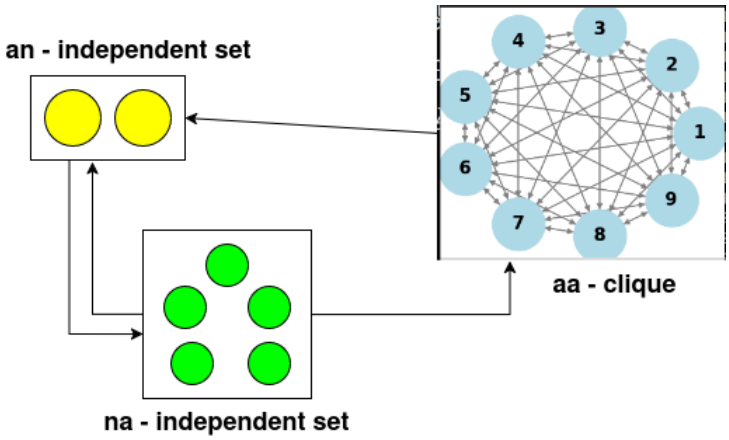

Figure 8: Trapping Player 2 in the AA-clique

Note that since the clique size is odd if the 2 players continue to guess country names belonging to the clique, the last remaining node will be said by player 1. Thus player 2 will have to escape from the clique to the an-independent set to say Afghanistan or Azerbaijan. Note that since the cardinality of the na-independent set is more than that of the an-set, player 1 can force player 2 back into the an-set or the clique. In the former case, player 2 again leads player 1 to the na-set. Having exhausted the an-set, player 2 has no option but to backtrack to the clique (if it's not already exhausted). Note that the number of nodes left in the clique is now even as the last node in it was guessed by player 1. Thus an even length traversal in the remaining clique starting from player 2 exhaust all countries starting with the letter 'a' and player 1 wins the game. ■

Strategy: Player 1 ALWAYS wins the game of ATLAS on countries if (s)he starts by guessing the name of a country that both starts and ends with *a* by trapping player 2 in the odd-sized aa-clique or the an-independent set.

3.4.2 Cities

3.4.3 Combined

3.5 Finding terminals

In our analysis of paths, we noted that there are 2 kinds of terminals - *dead ends* and nodes that *loopback* into the path. Since *dead ends* do not exist in any of our graphs, we search for the *loopy ends*. As concluded in our search for paths section, we will try to do a local evaluation of nodes.

3.5.1 Countries

The *bfs-tree* of the countries graph (without edges) is shown in Figure 9.

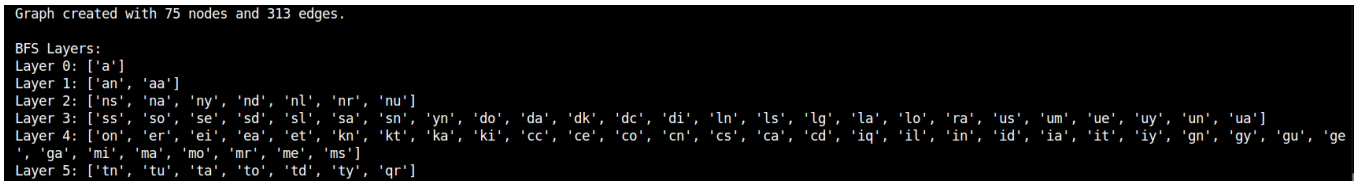

Figure 9: Auxiliary node 'a' added as initialization can happen from either 'aa' or 'an'

Inference: Any country can be reached in less than 5 turns. Thus, a 3-step local evaluation (node_level $\pm$ 3) is as bad as traversing the entire tree. We thus limit our evaluation to the immediate neighborhood and nodes at a distance of 2.

Now, let us try to characterize the nodes that can show such *loopback* behavior.

Independent Set and Cycles

Looking at Figure 8, we see that such nodes have outgoing edges to their 'ancestors' in a typical traversal. An immediate idea would be to enumerate the nodes based on the number of 'back edges' in the *bfs-tree*. However, this approach doesn't work since we can be at any node in the graph in less than 5 turns, any node can serve as the root of a new *bfs-tree* with 70 nodes. So, the number of back edges for a particular tree is not very insightful. Further and more importantly, it is a property dependent on the path and not relevant to our local analysis.
Instead of directly searching for such probable 'loopy ends' we focus on the 'stopping point' that the loopy end leads to, that is, the node that ends up getting repeated.
If such a node is an independent set (in the original non-reduced graph) of size >= 2 then the player has a way to escape. Thus we desire to find nodes that are an independent set of size 1 in the original graph. In other words, letters that have just one country starting with them. Further, such nodes should have a high in-degree to allow many nodes to *loop* into them.

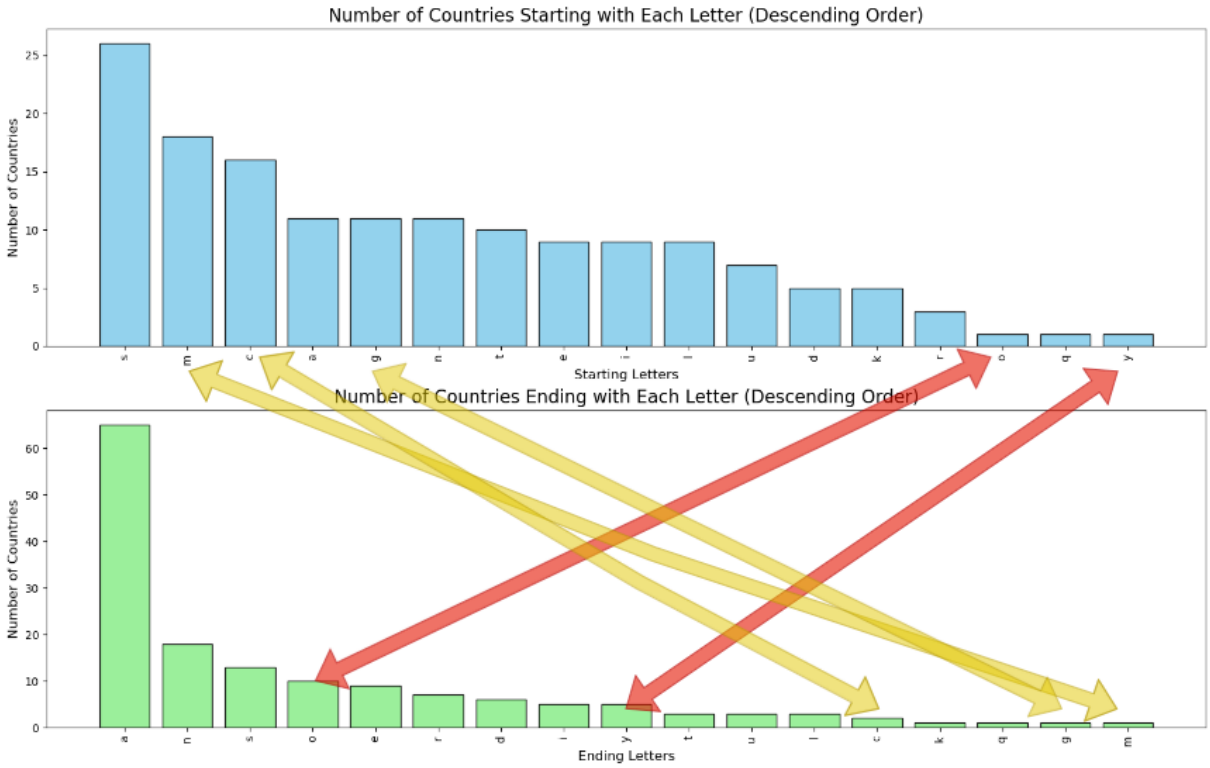

Figure 10: Red arrows - Letters that are first letter of just 1 country but last letter of many, Yellow arrows - Letters that are last letter of very few countries but first letter of many

Observation 1: Only one country starts with o (oman) and y(yemen) but many end with those letters.

Observation 2: Only one country ends with m (united kingdom) and g(luxembourg) and just 2 countries end with c (dominican republic and central african republic) but many start with those letters.

We focus our attention to observation 1 and will return to observation 2 in later sections.
We use our `cycle_countries.py` program to find all 3-cycles that *oman* is a part of.

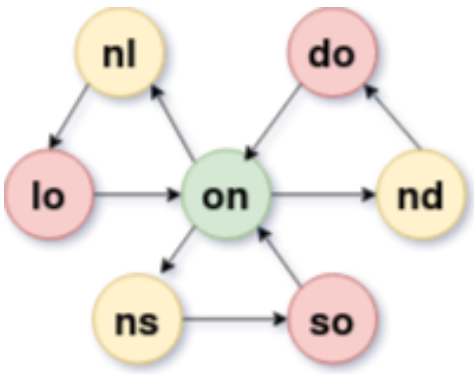

Figure 11: Looping back in oman

Strategic choice: If player j lands at do, lo or so then player i can trap player j by forcing it in *oman* again (provided j guesses nl, nd or ns).

We now look at the other country pointed out in observation 1 - *yemen*.

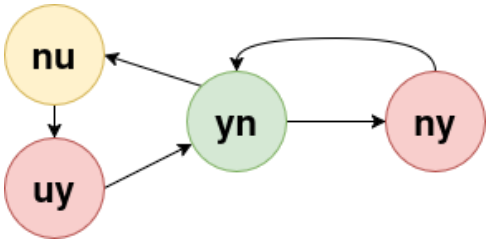

Figure 12: Looping back in yemen

Strategic choice: If player j lands at yn or nu then player i can trap player j by forcing it in *yemen* again (provided j guesses nl, nd or ns).

Limitations: Though this analysis does provides us with strategic choices for certain scenarios, it is limited to a set of nodes that are likely to be reached. It also assumes that the 'loopy-ends' have not already been traversed.

In our next section, we perform reformations that help us get much more comprehensive and deterministic strategies for an extensive set of places.

3.6 Graph Abstraction

Till now we have been looking at the graph purely in terms of its structure - cliques, independent sets, cycles that allow us to loop-back into a particular node that has no options left.
We will add more contextual meaning to our graph now...When does a game of ATLAS decisively ends? Simply put, when all places starting from a particular **letter** are exhausted and a place ending with that letter appears. Thus instead of finding likely terminal points, we shift our focus to exhausting the set of places starting from a particular **letter**. In other words, we abstract the details of the interconnection between places by focusing on the letters reachable by a particular letter.

We perform a transformation of our graph as follows:

1. All letters that are either the first letter or the last letter of a place become the nodes of the graph.
2. All places xy become directed edges from the letter x to letter y.
3. Every edge xy carries a weight equal to the number of places starting from x and ending at y.

Results of transformation:

1. Apart from adding contextual meaning to the graph, *abstraction* leads to further *reduction* in the graph structure. The cardinality of the node set can at most be 26 and that of the edge set equals the cardinality of the node set from the previous reduced graph.

2. While playing the game, if player i goes from x to y then the weight of the edge xy reduces by 1 if it was formerly >= 1. The game ends when a player arrives at a node with no non-zero-weighted outgoing edges.

Goal: To trap the opponent we need to select a node with the *loopback* characteristic. We of course have the option of counting the number of 2-cycles and 3-cycles, but having already explored this, we motivate our choices using deductions from centrality measures.

3.6.1 Countries

The figures depict the transformed graph for countries with just 17 nodes and 75 edges.

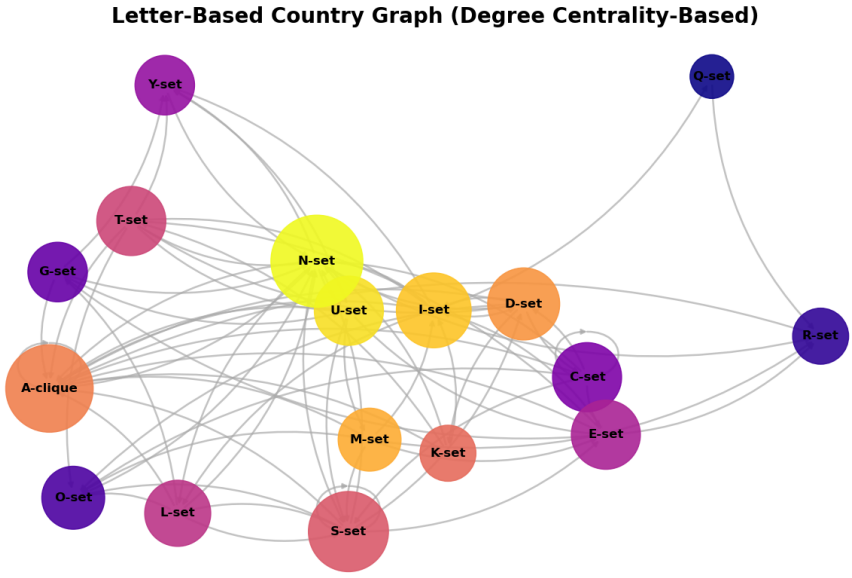

Figure 13: Transformed graph with node sizes proportional to their degree centrality

Degree Centrality
Degree centrality was computed for each node:

$$C_D(v) = \frac{\text{in-degree}(v) + \text{out-degree}(v)}{N - 1} \tag{1}$$

Betweenness Centrality
Betweenness centrality highlights critical nodes:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{2}$$

where σ_{st} represents the number of shortest paths from s to t, and $\sigma_{st}(v)$ denotes the fraction passing through v.
Computational Results:

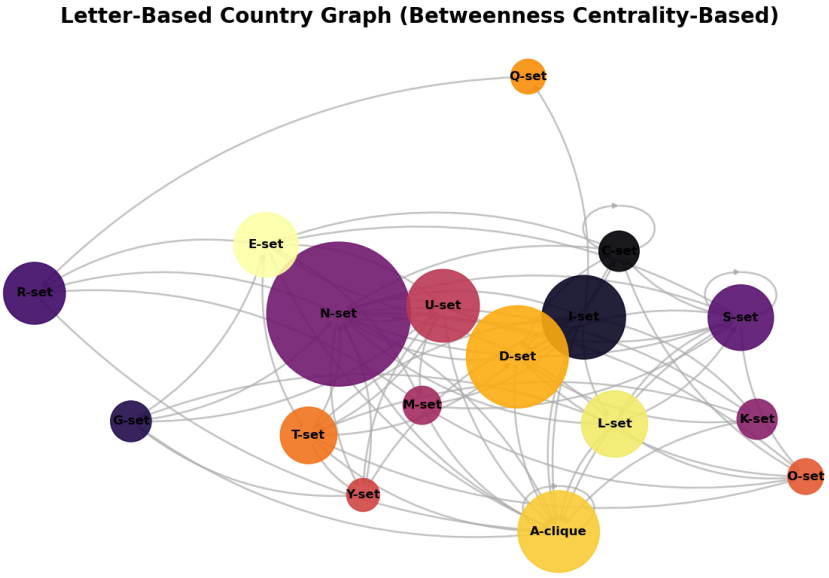

Figure 14: Transformed graph with node sizes proportional to their betweenness centrality

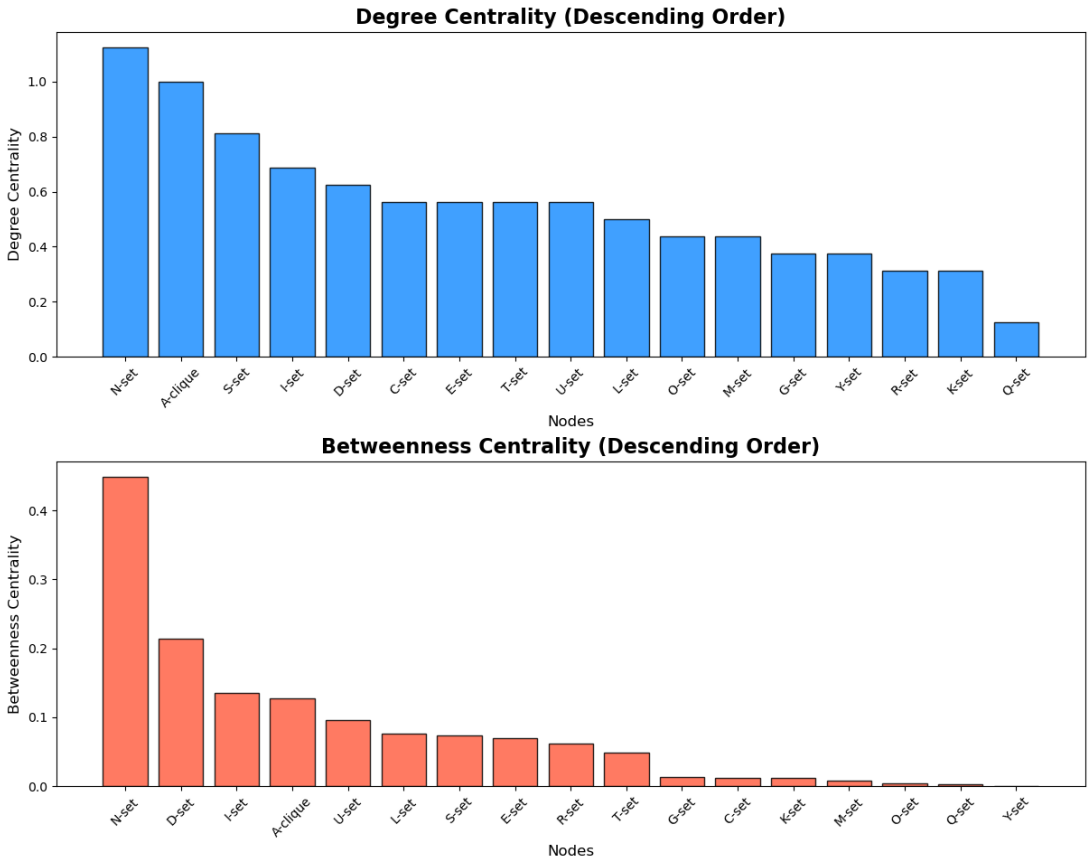

Figure 15: Computed centrality measures for each node

Interpretation of results:

Degree centrality measures "how well connected" a node is to other nodes in the graph. Betweenness centrality measures "how pivotal" a node is, that is, how well it acts like a bridge.
What we want is a node that is (1) easily reachable and (2) participates in many short cycles.

Inference: The $N - set$, $D - set$ and the $I - set$ have both very high betweenness and degree centrality values. This means that they are well connected to other sets in the graph and also control the 'flow' of the game.

The N-flower trap

Note that the $N - set$ has degree centrality > 1. This suggests that the sum of its in-degree and out-degree is $>$ N-1 and hence the $N - set$ has both outgoing edges and incoming edges to and from the same nodes. This combined with the fact that it has a very high betweenness centrality suggests that there are many three and two cycles like $X - set \rightarrow N - set \rightarrow X - set$.
On investigation, we get this flower sub-graph illustrated in figure 16.

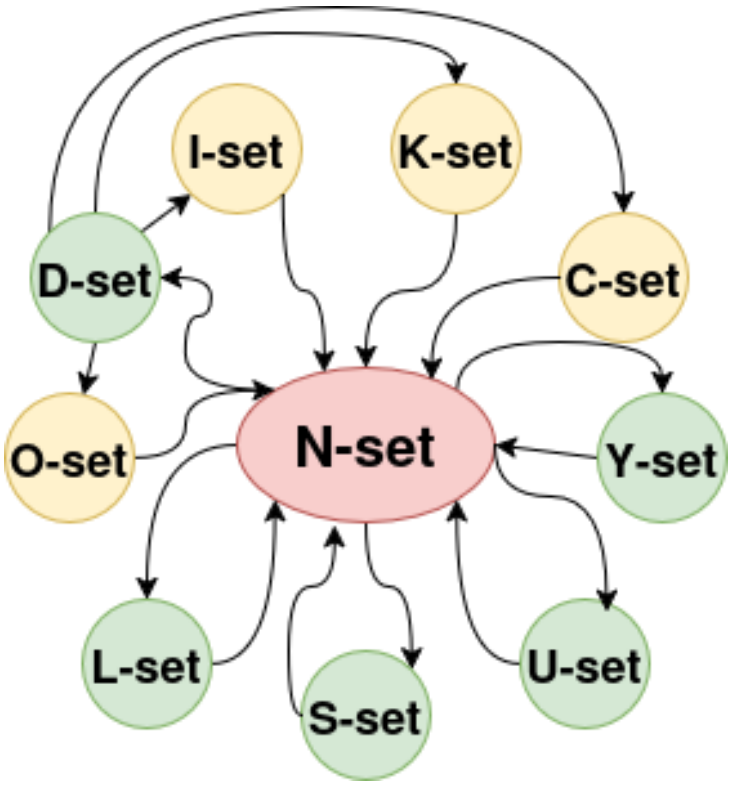

Figure 16: The N-flower

Note: Since the letter A (both aa and an) have been thoroughly explored earlier and a winning strategy has been developed for it, we omit it from our current analysis.

Claim: *Player i* has a winning guarantee if (s)he lands at the $N-set$ first by trapping *player j* inside it.

Proof: *Player i* arrives at the $N-set$ first and says 'nd' to transition to the $D-set$. Now *player j* can go to either O, I, K or the C-set from D. All of these have edges to the N-set thus putting *player j* inside the N-set. Note that the weight of all edges leaving the N-set is 1 and thus no edge can be traversed more than once. Since nd has already been utilized by *player i*, *player j* can escape only to the $L-set$, $S-set$, $U-set$ or the $Y-set$ but all of these nodes loopback into the N-set by a single transition thus exhausting the N-set with *player j* inside it. ■

Strategy: Player i ALWAYS wins the game of ATLAS on countries if (s)he is the first one to guess a country starting with the letter N from the A-set (via Afghanistan or Azerbaijan).

The D-flower trap

By performing similar inferences from the centrality measures on the D-set we get the following structure in Figure 17.

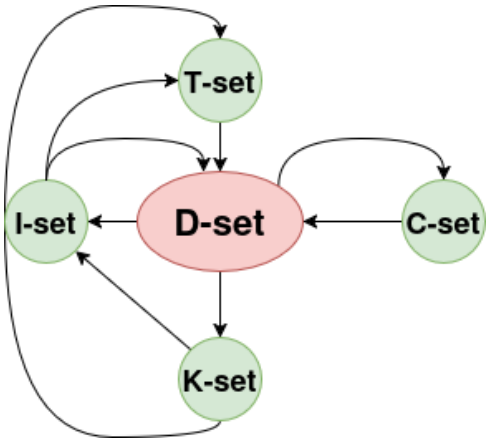

Figure 17: The D-flower (well, not exactly a flower!)

Note: Having analyzed the N-set and the A-set, these letters have been omitted from the current evaluation. We further assume that the player arrives at the D-set by a node in the same level of A-rooted bfs or higher (N-set or S-set as parent).

Claim: *Player i* has a winning guarantee if (s)he lands at the $D-set$ first by trapping *player j* inside it.

Proof: *Player i* arrives at the D-set and transitions to the K-set. *Player j* can escape to either the I-set or the T-set. Both of these have edges coming back into the D-set and thus *player j* gets inside the D-set. Now (s)he can escape to either the C-set or the I-set but both loopback into the D-set thus trapping *player j*. ■

Strategy: Player i ALWAYS wins the game of ATLAS on countries if (s)he is the first one to guess a country starting with the letter D arriving from the N-set or the S-set in a monotonic descent of bfs rooted at the letter A.

(what we mean by monotonic descent is that you start at the letter A and then keep coming down at the lower levels of bfs by not taking a 'backedge' to an upper level)

We thus see that how high centrality measures translate to a *loopback* characteristic and our motivated to make the following hypothesis:

Hypothesis: For small strongly connected graphs, there exists a positive correlation between the number of short cycles a node participates in and its degree and betweenness centrality values.

3.6.2 Cities

3.6.3 Combined

3.7 A probabilistic approach

Till now we have been finding structures in our graph and deriving guaranteed winning strategies. We now attempt a *greedy* approach that does not guarantees winning but does provide a competitive edge.

3.7.1 Countries

We remove the letters A and N from our analysis as deterministic strategies have already been found centered around them and we desire to analyze the remaining structure of the graph.

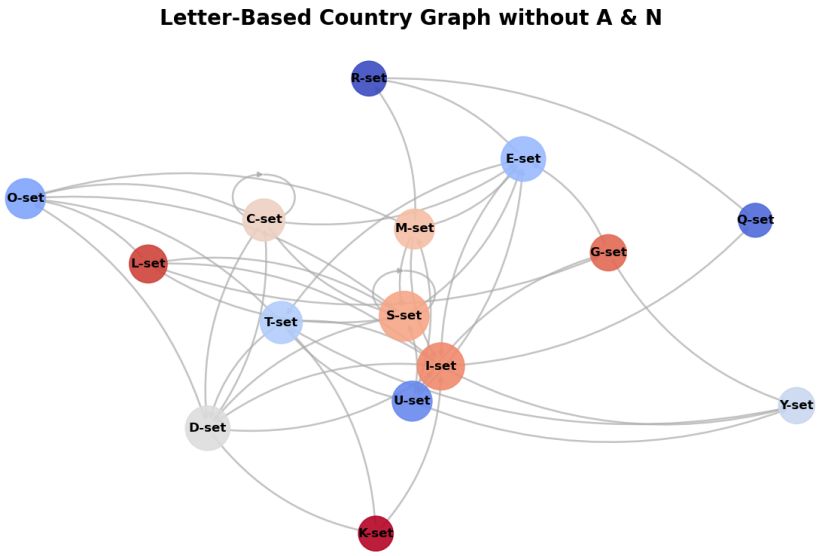

Figure 18: Letter graph with letter A and N removed

We utilize this transformed graph for this purpose.

Game Simulation Rules:

1. Player 1 starts the game with a country with the letter S, U, L or D (These letters are at the topmost level of bfs after the exclusion of A and N).
2. Going from one letter node to another by a non-zero weighted edge is a *legal* move.
3. Every legal move decreases the edge weight of that move by 1.
4. Player 1 makes legal moves as per the greedy approach described below.
5. Player 2 makes random but legal moves.
6. Game ends in favor of player i when player j arrives at a node with no non-zero weighted outgoing edge from that node.

The greedy step:

1. Transition to the letter with the total weight of outgoing edges as zero if possible.
2. Else transition to the letter with the highest difference in sum of weights of incoming edges and the sum of weights of outgoing edges.

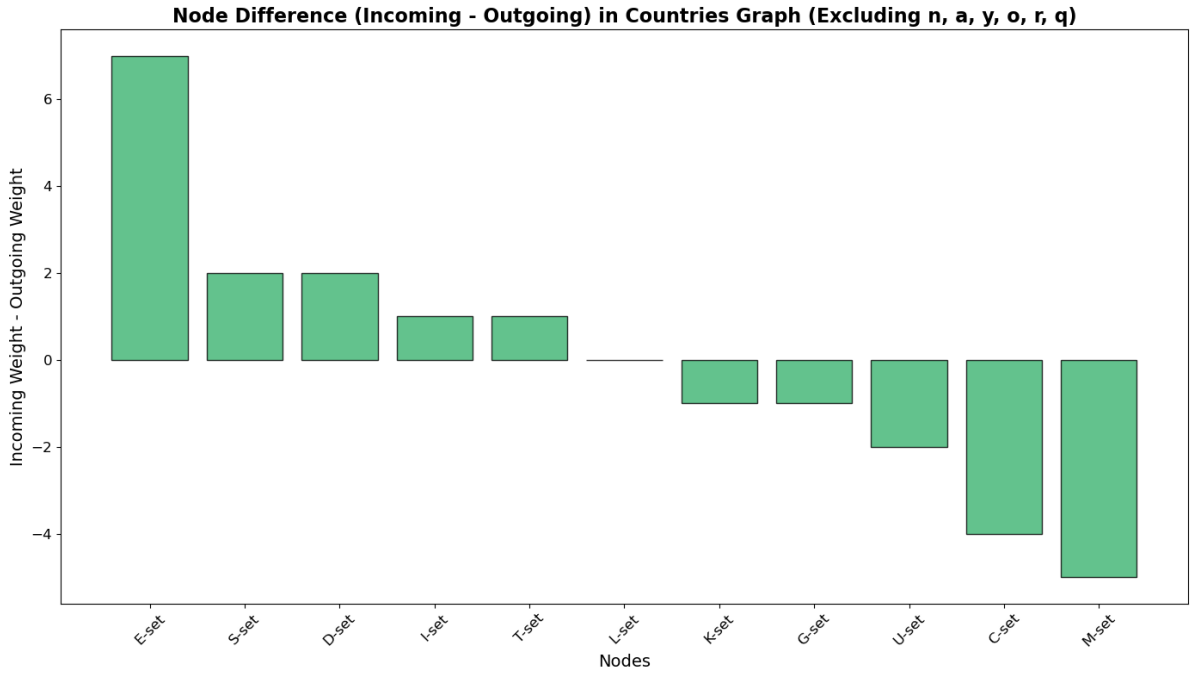

Figure 19: Left to right: decreasing priority order

Explanation: If player 1 has successors with no outgoing edge with a non-zero weight then it makes complete sense for player 1 to force player 2 in that dead-end and win the game there and then. Otherwise, player 1 evaluates two parameters of its successors - total weight of incoming edges and the total weight of outgoing edges. The former is desired to be high as it maintains flexibility for player 1 to keep that option safe for

him/her for later usage. The latter is desired to be low as it limits the escape options for player 2. To assimilate both the factors we compute the total incoming weight - total outgoing weight for every node in the graph and prepare a priority list for player 1 by assigning the highest priority to the node with the highest difference.

Strategy: Player 1 has a competitive edge over player 2 if (s)he makes greedy transitions at his turn as per the above stated rule.

Testing the strategy:

We use our `test_countries.py` program to simulate the games as per the stated rules and developed strategy. Table 1 summarizes the results of simulation.

Number of games simulated	**Player 1 wins**
10	80%
50	64%
100	69%
500	74.4%
1000	75.2%
5000	74.48%
10000	74.66%
100000	75%

Table 1: Simulation results for testing strategy

Since the player 1 win percentage converges around 75% we conclude that our strategy helps us win three-fourth of the games!

4 Task 2 - Community Detection

Now, we perform community detection on our original countries graph with 195 nodes and 2033 arcs.

4.1 Selection of Algorithms

There are various algorithms available for community detection. We analyze the assumptions, advantages and disadvantages of these algorithms in the context of our graph.

1. **Triangle Count and Clustering Coefficient:** This algorithm gives an idea of how 'tightly' a group is clustered compared to how tightly it 'could' be clustered. This algorithm is effective in finding standard structures like cliques. Note that we have already analyzed cliques and cycles in task 1. Further, this algorithm is not much useful for directed graphs, it requires specialized adaptations.

2. **Strongly Connected Components:** We have already found the SCCs in our raw country graph at the early stage of analysis and found that there are 43 SCCs with 153 nodes in one component and others comprising of orphans and their successors.

3. **Label Propagation:** This involves initializing random seeds and propagation of labels to its neighborhood. Conflict management for a node is done through the weight of the relationship. This algorithm is not that useful for our graph as the original graph is unweighted.

4. **Louvian Modularity:** Tries different groupings and compares community density with the goal of reaching a global optimum by quantifying how well a node is assigned to a group by comparing cluster density to the average sample. It first starts with a random baseline then computes modularity. Undergoes multiple iterations to test different groupings and then increases coarseness to get larger communities encompassing sub-communities. Thus it learns a hierarchy of clusters.

The criticality of edge directions

Note that the traditional Louvian modularity is a fast and reliable algorithm but for finding communities in undirected graphs. In our game of ALTAS and the countries graph, the edge directions have a great significance. These edge directions introduce an asymmetrical relationship between nodes that should be accounted for in community detection. Simulating a game on the country graph shows a 'flow pattern' and can lead to flow based communities. Thus just using traditional Louvian may undermine the importance of directional properties embedded in our country graph.

4.2 Lieden Community Detection

The Lieden algorithm is an improvement over Louvian. It naturally extends to directed graphs by:

1. **Respecting Edge Directionality:** When optimizing modularity, the algorithm considers the direction of edges to ensure that communities reflect the flow of influence or information. It does so by computing the **directed modularity**.

2. **Well-Connected Communities:** Leiden ensures that communities are not internally disconnected which happens in Louvian.

The **directed modularity** Q measures the quality of a community partition in a directed graph. It is defined as:

$$Q = \frac{1}{m} \sum_{ij} \left[A_{ij} - \frac{k_i^{\text{out}} k_j^{\text{in}}}{m} \right] \delta(c_i, c_j)$$

Where:

- Q: Modularity score, which quantifies the strength of the community structure.
- m: Total weight of all edges in the graph, calculated as $m = \sum_{ij} A_{ij}$.
- A_{ij}: Weight of the edge from node i to node j. For unweighted graphs, $A_{ij} = 1$ if an edge exists and 0 otherwise.
- k_i^{out}: Out-degree of node i, calculated as $k_i^{\text{out}} = \sum_j A_{ij}$.
- k_j^{in}: In-degree of node j, calculated as $k_j^{\text{in}} = \sum_i A_{ij}$.

- $\delta(c_i, c_j)$: Kronecker delta function, which is 1 if nodes i and j belong to the same community ($c_i = c_j$) and 0 otherwise.

The term $\frac{k_i^{\text{out}} k_j^{\text{in}}}{m}$ represents the expected probability of an edge between nodes i and j in a random graph with the same degree distribution. The modularity Q compares the actual edge weight A_{ij} to this expected value, rewarding communities with more internal connections than expected by chance.

Lieden is also more scalable and faster than Louvian.

4.2.1 Communities returned by Lieden

The table below lists the detected communities, with each community containing a set of countries:

Community	Countries
0	afghanistan, albania, algeria, andorra, angola, antigua and barbuda, argentina, armenia, australia, austria, azerbaijan, bolivia, bosnia and herzegovina, botswana, bulgaria, cambodia, canada, china, colombia, costa rica, croatia, cuba, czechia, gambia, georgia, ghana, grenada, guatemala, guinea, guyana, jamaica, kenya, latvia, liberia, libya, malta, mauritania, micronesia, moldova, mongolia, panama, papua new guinea, romania, russia, rwanda, tanzania, tonga, tunisia, venezuela, zambia
1	bahamas, barbados, belarus, comoros, cyprus, honduras, laos, maldives, marshall islands, mauritius, netherlands, philippines, saint kitts and nevis, saint lucia, saint vincent and the grenadines, samoa, san marino, saudi arabia, senegal, serbia, seychelles, sierra leone, singapore, slovakia, slovenia, solomon islands, somalia, south africa, south korea, south sudan, spain, sri lanka, sudan, suriname, sweden, switzerland, syria, são tomé and príncipe, united arab emirates, united states
2	bahrain, benin, bhutan, cameroon, gabon, japan, jordan, kazakhstan, kyrgyzstan, lebanon, liechtenstein, namibia, nauru, nepal, new zealand, nicaragua, niger, nigeria, north korea, north macedonia, norway, oman, pakistan, tajikistan, turkmenistan, uzbekistan, yemen
3	belize, cape verde, chile, east timor, ecuador, egypt, el salvador, equatorial guinea, eritrea, estonia, eswatini, ethiopia, france, greece, mozambique, palestine, ukraine, zimbabwe
4	brunei, burundi, djibouti, fiji, haiti, iceland, india, indonesia, iran, iraq, ireland, israel, italy, ivory coast, kiribati, malawi, mali
5	belgium, madagascar, malaysia, mexico, monaco, montenegro, morocco, myanmar, united kingdom, vietnam
6	chad, denmark, dominica, dominican republic, dr congo, finland, poland, thailand
7	guinea-bissau, palau, peru, tuvalu, uganda, uruguay, vanuatu
8	brazil, lesotho, lithuania, luxembourg, portugal

Community	Countries
9	kuwait, togo, trinidad and tobago, turkey
10	bangladesh, hungary
11	central african republic, congo
12	burkina faso
13	germany
14	qatar
15	vatican city
16	paraguay

4.2.2 Analysis of results

4.3 Infomap

Unlike Lieden which utilizes directed modularity and is a density based community detection algorithm, Infomap presented by Rosvall and Bergstrom is based on the concept of patterns of movement among the nodes of a directed network. The community detection method presented in that work is based on random walks and the main intuition is that a community can be defined as a group of nodes where the random surfer is more likely

to be trapped in. This concept can be treated as an increased flow circulation pattern between the nodes of the same community. The idea behind this algorithm seems to resonate better with our theme of ATLAS game which also a "walk".

4.3.1 Communities returned by Infomap

The following table presents the detected communities, listing the countries in each.

Community	Countries
1	afghanistan, albania, algeria, andorra, angola, antigua and barbuda, argentina, armenia, australia, austria, azerbaijan, bolivia, bosnia and herzegovina, botswana, bulgaria, equatorial guinea, eritrea, estonia, ethiopia, jamaica, namibia, nicaragua, nigeria, north korea, north macedonia, panama, papua new guinea, venezuela, zambia
2	bahamas, barbados, belarus, comoros, cyprus, honduras, laos, maldives, marshall islands, mauritius, netherlands, philippines, saint kitts and nevis, saint lucia, saint vincent and the grenadines, samoa, saudi arabia, senegal, serbia, seychelles, sierra leone, singapore, slovakia, slovenia, solomon islands, somalia, south africa, south korea, south sudan, spain, sri lanka, sudan, suriname, sweden, switzerland, syria, são tomé and príncipe, united arab emirates, united states
3	burkina faso, denmark, dominica, dr congo, finland, kazakhstan, kenya, kyrgyzstan, lesotho, new zealand, oman, poland, san marino
4	belize, east timor, ecuador, el salvador, france, madagascar, myanmar, niger, palestine, romania, russia, rwanda, zimbabwe
5	hungary, norway, paraguay, vatican city, yemen
6	brazil, latvia, lebanon, liberia, libya, liechtenstein, lithuania, nepal, portugal
7	guinea-bissau, nauru, palau, peru, uganda, ukraine, uruguay, uzbekistan, vanuatu
8	bangladesh, brunei, burundi, djibouti, eswatini, fiji, haiti, iceland, india, indonesia, iran, iraq, ireland, israel, italy, kiribati, malawi, mali, qatar
9	egypt, ivory coast, kuwait, tajikistan, tanzania, thailand, togo, tonga, trinidad and tobago, tunisia, turkey, turkmenistan, tuvalu
10	cambodia, cameroon, canada, cape verde, central african republic, chad, chile, china, colombia, congo, costa rica, croatia, cuba, czechia, dominican republic
11	belgium, malaysia, malta, mauritania, mexico, micronesia, moldova, monaco, mongolia, montenegro, morocco, mozambique, united kingdom, vietnam
12	gabon, gambia, georgia, germany, ghana, greece, grenada, guatemala, guinea, guyana, luxembourg
13	benin
14	bhutan
15	bahrain

Community	Countries
16	japan
17	jordan
18	pakistan

4.3.2 Analysis of results

4.4 Assessment metrics

5 Conclusion

This report detailed the exploration of the ATLAS game using graph theory. Key findings include:

- Nodes with higher betweenness centrality influence game outcomes significantly.
- Dead-end strategies give a significant advantage to Player 1.
- Graph-based heuristics can optimize gameplay strategies in turn-based decision environments.

References

[1] M. Newman, *Networks: An Introduction*, Oxford University Press, 2010.

[2] A.-L. Barabási, *Network Science*, Cambridge University Press, 2016.