Summary of Prolog Code for Finding Simple Cycles in Directed and Undirected Graphs

A Summary

October 26, 2023

1 Introduction

This document summarizes Prolog code designed to find simple cycles in directed and undirected graphs. The directed graph implementation is in `simpleCycle.pl`, and the undirected graph variant is in `simpleCycleUndirected.pl`. A cycle is considered simple if, for any two distinct nodes within the cycle, the shortest path between them in the entire graph is the path that follows the cycle's edges.

2 Directed Graph Implementation (simpleCycle.pl)

2.1 Graph Representation and Setup

The graph is represented using dynamically asserted Prolog facts:

- `node(NodeID, Type)`: Declares a node with a unique ID and an associated type.
- `arc(ArcID, Type, SourceNode, TargetNode)`: Declares an arc with a unique ID, type, source, and target node.
- `edge(Source, TargetNode)`: Helper predicate generated dynamically for traversal efficiency.

The graph setup is handled by `setup_test_graph/1`, which retracts existing `arc/4` facts and asserts new ones based on a chosen test case. `generate_edges_from_arcs/0` creates `edge/2` facts from `arc/4` facts for faster lookup during graph traversal.

2.2 Finding Elementary Cycles

Elementary cycles are found using Depth-First Search (DFS):

- `find_all_elementary_cycles/1`: Collects all nodes from `node/2` facts and initiates DFS from each.

- `find_cycles_from_potential_starts/2`: Iterates through nodes and calls `dfs_for_cycle/4` to find cycles.
- `dfs_for_cycle/4`: Performs the recursive DFS. It uses `edge/2` to find the next node to visit, ensures elementarity by checking that the node hasn't been visited in the current path, and returns the cycle in reverse traversal order.

2.3 Filtering for Simple Cycles

The algorithm filters elementary cycles based on the shortest path criterion:

- `filter_to_simple_cycles/2`: Takes a list of elementary cycles and returns a list of normalized simple cycles.
- `is_simple_cycle_candidate/1`: Checks if an elementary cycle is simple. This involves reversing the cycle, extracting unique nodes, and then checking all node pairs for chords.
- `check_all_node_pairs_for_chords/2`: Iterates through all ordered pairs of nodes in the cycle.
- `check_one_node_against_all_others/3`: For a given node, iterates through other nodes in the cycle and checks if the shortest path between them is shorter than the path along the cycle. The function `get_distance_along_cycle/4` and `find_shortest_path_length/3` are used to get the respective shortest paths.

2.4 Cycle Normalization

Normalization ensures unique representation of identical cycles starting at different nodes:

- `normalize_cycle_representation/2`: Converts a raw DFS cycle to a normalized node list. It finds the lexicographically smallest node and rotates the cycle to start with that node.

The final list of simple cycles is produced using `setof/3` on the normalized cycles to ensure uniqueness and canonical order.

3 Undirected Graph Implementation (simpleCycleUndirected.pl)

3.1 Edge Generation

In the undirected version, each arc generates two facts: `edge(Source, Destination)` and `edge(Destination, Source)`. This is handled by `generate_edges_from_arcs/0`, while ensuring to avoid duplicates.

3.2 Finding Elementary Cycles

The DFS procedure is modified to avoid spurious cycles that can arise from immediately returning to the previous node:

- `dfs_for_cycle/4`: The predicate keeps track of the immediate predecessor in the path and ensures that the next node to visit is not this predecessor.

3.3 Chord Detection

The version for undirected graphs handles chord checking in a particular way to avoid false positives:

- `check_one_node_against_all_others/3`: The check skips pairs of adjacent nodes in the cycle (in both directions) to avoid confusing the cycle's own edges with possible chords.

3.4 Cycle Normalization

A cycle in an undirected graph can be traversed in both directions. The undirected version normalizes cycles by considering both possible directions:

- `normalize_cycle_representation/2`: Generates two canonical representations (one for each traversal direction) and chooses the lexicographically smaller one as the canonical representation.

4 Conclusion

The Prolog program provides an effective solution for finding simple cycles in both directed and undirected graphs. It leverages DFS, BFS, dynamic facts, and list manipulation to achieve its goal. The cycle normalization ensures that equivalent cycles are represented consistently, and `setof/3` provides a final, ordered list of these unique simple cycles.