Zero Path Reachability

Jae Tak Kim

1. It's clear that every Z-path must be of the form k number of -1 followed by k number of $+1$, where $k \in \mathbb{N}$. Thus, every Z-path of length $2k$ contains within it Z-paths of length $2, \ldots, 2(k-1)$. My initial idea was to start with Z-paths of length 2 and then build them up as large as you can in the graph, sort of like Prim's algorithm for MST's. However, I couldn't get an algorithm that runs in $O(EV)$-time.

   Another idea in a similar vein is to consecutively shrink the Z-paths from smallest to the largest. That is, to get every pair of consecutive -1 and $+1$ weight and combine them into one edge. If you have nodes a, b, c, d, e and edges $(a, b), (b, c), (c, d), (d, e)$ with $w(a, b) = w(b, c) = -1$ and $w(c, d) = w(d, e) = +1$, you would first combine edges (b, c) and (c, d) into one edge (b, d). However, you have to be sure not to remove the combined edges because there could be another Z-path if edge (f, c) was in the graph with $w(f, c) = -1$, and if you removed (c, d), the algorithm would incorrectly miss the Z-path from f to d. The natural weight for the newly created edge is 0 since the weight of the path $p = b, c, d$ has weight 0. We could merely record this new edge in a separate list or adjacency matrix but since we know that this Z-path is a part of a longer Z-path, we want to use this information somehow to create a new edge (a, e) of weight 0. Since we added the weights of the two edges to create (b, d), we can do the same with the other edges. We can add the weights of edges (a, b) and (b, d) to create an edge (a, d) of weight $-1 + 0 = -1$, which would then be combined with the last remaining edge (d, e).

CFGtoCNF($G = (N, \Sigma, R, S)$)

```
// Step 1: Create dummy non-terminals
for (X → Y) ∈ R where len(Y) ≥ 2:
    if ∃V ∈ Y where V ∈ N:
        // RHS should have at least one variable
        for σ ∈ Y where σ ∈ Σ:
            // loop through terminals
            // create dummy variables
            add new variable V′ to N
            add rule (V′ → σ) to R
// Step 2: Convert unit productions
while ∃(X → Y) ∈ R where len(Y) == 1 and Y[0] ∈ N:
    for such (X → Y):
        // The single RHS must be a variable
        newY = []
        for (X′, Y′) ∈ R where X == Y[0]:
            add Y′ to newY
// Step 3: Make all rules binary
while ∃(X → Y) where len(Y) > 2:
    for such (X → Y):
        // Arbitrarily replace the first two variables
        Replace Y[0 : 2] with new variable V′
        Add rule (V′ → Y[0 : 2]) to R
```