Dynamic Graph Framework with Surreal Numbers and Visualization

December 16, 2024

Abstract

This paper presents an extended dynamic graph framework that integrates surreal numbers for representing nodes, edges, and flows. By leveraging ordinal arithmetic and surreal constructions, we model hierarchical and infinite systems such as traffic networks, knowledge graphs, and decision processes. The framework includes dynamic updates, bottleneck detection, perturbation mechanisms, and visualization via Graphviz.

1 Introduction

Surreal numbers, introduced by John Horton Conway, provide a universal number system that includes real numbers, infinities, and infinitesimals. These properties make surreal numbers a natural choice for modeling hierarchical and complex systems. This paper demonstrates how surreal numbers can be integrated into a dynamic graph framework, with applications in traffic systems, policy negotiation, and knowledge graphs.

2 Surreal Numbers and Ordinals

2.1 Definition

Surreal numbers are defined recursively as:

$$x = \{L|R\}, \tag{1}$$

where L and R are sets of surreal numbers such that $l < r$ for all $l \in L$ and $r \in R$.

Ordinals are a subset of surreal numbers that represent hierarchical positions. For example:

$$\omega + 1, \quad \omega^2, \quad \omega^\omega. \tag{2}$$

2.2 Operations

Ordinal arithmetic is extended to surreal numbers:

- Addition: $\omega + 1, \quad \omega + \epsilon$.
- Multiplication: $\omega \cdot 2, \quad \omega \cdot \omega = \omega^2$.
- Exponentiation: $\omega^\omega, \quad \omega^{\omega^\omega}$.

2.3 Applications

Surreal numbers model:

- Infinite hierarchies (e.g., ω^2, ω^ω).
- Infinitesimal effects (ϵ).
- Arbitrary scales of influence or cost.

3 Dynamic Graph Framework

3.1 Nodes, Edges, and Weights

A graph G is defined as:

$$G = (V, E, W), \tag{3}$$

where V is the set of nodes (surreal numbers), E is the set of edges, and W assigns weights to edges (ordinals).

3.2 Flow Dynamics

Flow rules govern the graph:

- **Update Flow:** Weights decrease over time to simulate flow dynamics.
- **Detect Bottlenecks:** Nodes with insufficient outgoing flow are identified.
- **Perturbations:** New nodes or edges are added to resolve bottlenecks.

3.3 Visualization

Graph visualization uses Graphviz to display nodes, edges, and weights dynamically.

4 Implementation

4.1 Haskell Code

The implementation integrates surreal numbers into a dynamic graph framework. The key functions include flow updates, bottleneck detection, and perturbations. Below is an excerpt of the Haskell code:

```
data Ordinal = Finite Int | Transfinite Int Int deriving (Eq,
    Show)

data Surreal = Zero
             | Node { left :: [Surreal], right :: [Surreal],
                birthday :: Ordinal }
             deriving (Eq, Show)

type Node = Surreal
type Weight = Ordinal
type Edge = (Node, Weight)
type Graph = Map.Map Node [Edge]
```

Listing 1: Ordinal and Surreal Number Definitions

```
updateFlow :: Graph -> Graph
updateFlow = Map.mapWithKey updateNodeFlow
  where
```

```
    updateNodeFlow _ = map (\(n, w) -> (n, addOrdinals w (
        Finite (-1))))

detectBottlenecks :: Graph -> [Node]
detectBottlenecks graph =
  Map.keys $ Map.filter (\edges -> sumWeights edges < Finite
      1) graph
  where
    sumWeights = foldr (addOrdinals . snd) (Finite 0)
```

Listing 2: Flow Updates and Bottleneck Detection

4.2 Graph Visualization

The graph is visualized using Graphviz. Nodes represent surreal numbers, and edges are annotated with ordinal weights. The following function generates a DOT file:

```
visualizeGraph :: Graph -> FilePath -> IO ()
visualizeGraph graph filePath = do
    let dotGraph = GVM.digraph (GVT.Str "SurrealGraph") $ do
            GVM.graphAttrs [GVA.RankDir GVA.FromLeft]
            forM_ (Map.toList graph) $ \(src, edges) -> do
                forM_ edges $ \(dst, weight) -> do
                    GVM.edge (GVT.Str $ TL.pack $ showNode
                        src)
                            (GVT.Str $ TL.pack $ showNode dst)
                            [GVA.Label $ GVA.StrLabel $ TL.
                                pack $ showWeight weight]

    TLIO.writeFile (filePath ++ ".dot") (GV.printDotGraph
        dotGraph)
    callCommand $ "dot␣-Tpng␣-o␣" ++ filePath ++ ".png␣" ++
        filePath ++ ".dot"
```

Listing 3: Graph Visualization with Graphviz

5 Applications

5.1 Traffic Optimization

Nodes represent intersections, and edges represent roads with weights as congestion levels. Random perturbations, such as opening new lanes, improve flow efficiency.

5.2 Knowledge Graphs

Nodes are concepts, and edges represent influence or association. Perturbations simulate the introduction of new ideas.

5.3 Policy Negotiation

Nodes represent policies, and edges encode dependencies. Flow rules guide consensus-building.

6 Conclusion

This framework unifies surreal numbers and dynamic graph theory, offering new tools for modeling hierarchical and complex systems. Future work will focus on real-time optimization and interactive visualizations.