GiMPy and GrUMPy: Visualization for Optimization in Python

Ted Ralphs\textsuperscript{1} Aykut Bulut\textsuperscript{1} Brady Hunsaker\textsuperscript{3}
Michael O’Sullivan\textsuperscript{2} Osman Ozaltin\textsuperscript{4}

\textsuperscript{1}COR@L Lab, Department of Industrial and Systems Engineering, Lehigh University
\textsuperscript{2}Department of Engineering Science, University of Auckland
\textsuperscript{3}Google
\textsuperscript{4}Department of Management Sciences, University of Walterloo

INFORMS Computing Society Conference, 7 January, 2013
GiMPy Overview

- A graph class written in Python 2.7.
- Depends on Pydot (and hence Graphviz) and Pygame.
- Builds, displays, and saves graphs.
- Focus is on *visualization* of well-known graph algorithms and use in the classroom.
  - Priority in implementation is on *clarity* of the algorithms.
  - Efficiency is *not* the goal (though we try to be as efficient as we can).
GiMPy Data Structure

- Derived from **Dot** class of **Pydot**.
  - **Pydot** is an API for building graphs and printing them in **Dot** language used by **Graphviz**.
  - **Graphviz** provides the methods for layout and visualization.
  - **Graphviz** produces images that can then either be displayed live or saved.

- **GiMPy** extends the API and adds the data structures and methods needed to implement graph algorithms.
- The graph is stored in adjacency list format, but alternative data structures can be easily added.
- Includes a wide range of graph algorithms and more are being added.
- Derived classes implement tree and binary tree.
- Available open source through COIN-OR.
GiMPy Display Capabilities

- Has all the display capabilities of Graphviz.
- Can display graphs on a window or save to the disk in various formats.
from gimpy import Graph
if __name__=='__main__':
    g = Graph(display='pygame')
    g.add_edge(0,1)
    g.add_edge(1,2)
    g.add_edge(3,4)
    g.display()
    g.search(0)

Figure: Resulting window
GIMPy Example

[Images of network diagrams showing nodes labeled 0, 1, 2, 3, and 4, with edges connecting them in different states.]
GiMPy Example
from gimpy import Graph
if __name__=='__main__':
    g = Graph(display='pygame')
    g.add_edge(0,1)
    g.add_edge(1,2)
    g.add_edge(3,4)
    g.search()
GiMPy Algorithm Visualization

The following problem/algorithm pairs with similar visualization options exist.

- **Graph Search**: BFS, DFS, Prim’s, Component Labeling, Dijkstra’s, Topological Sort.
- **Shortest path**: Dijkstra’s, Label Correcting
- **Maximum flow**: Augmenting Path, Preflow Push
- **Minimum spanning tree**: Prim’s Algorithm, Kruskal Algorithm
- **Minimum Cost Flow**: Network Simplex, Cycle Canceling
- **Data structures**: Union-Find (quick union, quick find)
GiMPy Binary Tree

- **Tree** class derive from **Graph** class.
- **BinaryTree** class derive from **Tree** class.
- Has binary tree specific API and attributes.
Visualizes branch and bound process.

Adds dynamic visualization to the Branch and Bound Analysis Kit (BAK).

Reads branch and bound data in a specific format.

Derived from `BinaryTree` of GiMPy.

Builds branch and bound binary tree.

Has all the analysis (search/visualization) capabilities of GiMPy.

Biggest advantage to all its predecessors is the capability to do visualizations on the fly.
There are four visualizations of BB tree provided by GrUMPy.

- BB tree
- Histogram
- Scatter plot
- Incumbent path
from baktree import BAKTree
if __name__ == '__main__':
    bt = BAKTree()
    bt.set_display_mode('pygame')
    line_number = 0
    file_ = open('p0201_GLPK.in', 'r')
    for line in file_:
        bt.ProcessLine(line)
        line_number = line_number+1
        if line_number%100 != 0:
            continue
        if bt.root is not None:
            gnuplot_image = bt.GenerateTreeImage()
            if gnuplot_image is not None:
                bt.display_image(gnuplot_image)
Figure: BB tree generated by GrUMPy.
Figure: BB histogram generated by GrUMPy
Figure: Scatter plot generated by GrUMPy
GrUMPy Incumbent Path

Figure: Incumbent path generated by GrUMPy

Depends on scatterplot
GrUMPpy On the Fly

- GrUMPpy can read input from stdin.
- By configuring the solver output you can make GrUMPpy generate visualizations as your problem is being solved.
- GrUMPpy reads stdin and updates binary tree and visualizations on the fly.
- GrUMPpy can be used on the fly with the following command.

```
 solver problem | python baktree.py
```
from baktree import BAKTree
import sys
if __name__ == '__main__':
    bt = BAKTree()
    bt.set_display_mode('pygame')
    line_number = 0
    file_ = open('p0201_GLPK.in', 'r')
    for line in sys.stdin:
        bt.ProcessLine(line)
        line_number = line_number+1
        if line_number%100 != 0:
            continue
        if bt.root is not None:
            bt.display_all()
Wrap-up

- The software is available and under active development.
- Please try it and help us improve!