A General Overview and Documentation of Heist

MAB-PE-DCOP

We used the pyDCOP framework to construct a new algorithm, heist, to solve DCOPs (Decentralized Constraint Optimization Problems).

Running the Application

To run the program do the following:
1) cd pyDCOP
2) python3 xmlsolve.py xml_solve --algo heisthelper 'file_path'
3) press ctrl + c when done or wait for it to finishing running

Installing

To install, simply clone in the desired folder using the git clone command

Dependencies

It is required to have python 3.6+. To install python 3, please follow the direction on the python website: https://www.python.org

pyDCOP

pyDcop provides a great framework to solve DCOP problems. It orchestrates the message passing mechanism between different computations and enables us to implement our own algorithms. Furthermore contains implementations of multiple DCOP algorithms, which provides a great template for implementing new algorithms.

How Heist was Implemented

There were a few files that had to be implemented to fit our needs:

1) xmlsolve.py
2) xmldcop.py
3) heisthelper.py
4) global_vals.py

The reason we add these files instead of editing existing files is two-fold. First, it provides extra flexibility and allows us to work outside the framework of pyDcop. For example, pyDcop can only take in yaml formatted files, but we need to appropriately interpret xml files and construct a DCOP. Second, in parallel with adding extra flexibility, adding files to pyDcop still allows us to extend a lot of the great features that pyDcop provides, such as agent management.

1) Xmlsolve.py sets up the basic architecture and provides the tools to interpret commands from the command line or terminal. It follows the essential structure of solve.py in pyDcop; however, xmlsolve allows us deconstruct xml formatted files to make a DCOP (Lines 405 to 467), and it organizes the unique heist algorithm structure and computation (Lines 484 to 561). Heist, unlike most algorithms, is structured as a nested algorithm, as it runs a pseudo-maxsum algorithm (implemented in heisthelper.py) and updates the DCOP each timestep. All other lines follow a similar format and often refer to solve.py, but they are slightly modified to provide flexibility in making our own argument parser and interpreting certain arguments.

2) Xmldcop.py is responsible for constructing the DCOP given a xml formatted file. It interprets the file and constructs the domain, variables, constraints, and agents involved in a DCOP. Furthermore, to properly implement heist, we have to construct an unique arm pull constraint table when building the constraints for the DCOP.

3) Heisthelper.py builds the computation behind the pseudo-maxsum algorithm that heist utilizes each timestep. The pseudo-maxsum algorithm utilizes the same message passing mechanism as the standard maxsum algorithm, but the values sent along these messages are different. The UCB (upper confidence bound) function used in multi-armed-bandits defines the values passed in messages. To get the UCB derived value, the utility value (normally provided in traditional maxsum) and the number of arm-pulls associated with that value (unique to the heist implementation) are required. The utility value and the number arm-pulls dictate the trade off between exploration and exploitation. The functions that dictate what value is being passed are called “factor_costs_for_var” and “costs_for_factor”.

4) Global_vals.py simply contains global variables used in the implementation of heist. This includes a dictionary of arm-pulls across the DCOP, the timestep for the DCOP, and a dictionary of normal distributions that we sample from. Each constraint table’s values are sampled from a specific normal distribution.

General Architecture of Heist and its implementation