Contents

Supplementary Materials for "Tentative Agreements Unique Offers Protocol" (Submitted to AAAI 2023)

This folder contains all the code implementing the algorithms described in the paper and all datasets used.

Reproducibility and Public Availability

- This guides shows how to reproduce every result in the paper.
- The data is already publicly available in the GENIUS platform.
- All the content of our supplementary materials will be made publicly available on Github in case of acceptance. Moreover, we will contribute our algorithms back the negotiation platform we use (NegMAS) as a pull request and if accepted will be available directly from the platform (which is community driven) in the future.

What is included?

- **`code`** Implementation of algorithms, negative cases, and experiments.
- **`data`** All the negotiation scenarios used in the experiments.

Code

- **`algorithms`** Contains implementations of the TAU protocol and SCS protocol. Implementations are modular and use a generalized version of the protocol description method introduced in the paper (Subsections Negotiation Protocol and Negotiation Strategy).
- **`experiments`** All code needed to run the experiments reported in the paper and generate the figures and tables in it.
  - **`experiment1.sh`** Runs experiment 1 and generates `e1.csv` with all raw results of the experiment.
  - **`experiment2.sh`** Runs experiment 2 and generates `e2.csv` with all raw results of the experiment.
  - **`analyze1.sh`** Runs analysis of the first experiment and generates Table 1 of the paper in latex format. It uses `e1.csv` generated by `experiment1.sh`. To just run the analysis on the data already in the data folder, change `e1.csv` in this script to `e1.csv`.
  - **`analyze2.sh`** Runs analysis of the second experiment and generates Fig. 1 of the paper in latex format. It uses `e2.csv` generated by `experiment2.sh`. To just run the analysis on the data already in the data folder, change `e2.csv` in this script to `e2.csv`.
  - **`experiment.py`** The actual code used to run all experiments (Called by the experiment shell scripts above). It is a command line utility that can be called with different arguments to run different experiments.
  - **`eval1.py`** Used to analyze experiment 1 (flexible and can be adjusted to do other evaluations as well not reported in the paper).
  - **`eval2.py`** Used to analyze experiment 2 (flexible and can be adjusted to do other evaluations as well not reported in the paper).
  - **`utils.py`** Utility methods used by other python scripts
  - **`helpers`** Helper scripts containing `clean.py` which is used in the second experiment to clean out extra runs if any and runs without controlled reserved value. In principle, it should not be needed.
  - **`tests`** Contains scripts for testing AOP and TAU strategies on specific conditions.
    - **`testaop.py`** Tests an AOP strategy (default is `MiCRO`). You can run three simple experiments with it:

    - **e1** (`incomplete`) shows that `MiCRO` and `Atlas3` are not exactly complete using the tiny `NiceOrDie` scenario (other strategies may vary).
    - **e2** (`all-rational`) shows success on the same domain when all outcomes are rational.
    - **e3** (`none-rational`) shows expected disagreement on the same domain when no outcomes are rational.
    - **e4** (`small`) runs the negotiation a relatively small domain (`Outfit`)
    - **e5** (`large`) runs the negotiation on a relatively large domain (`Kitchen`)
    - **e6** (`difficult`) runs the negotiation on a relatively large domain (`Outfit`) with larger reserved values (more difficult agreement)
    - **`free`** can be used to run on any domain.
  * **`testtau.py`** Tests the proposed SCS strategy on few example cases.
    - **e1** (`complete`) shows that TAU(∞, ∞) is not exactly complete using the tiny `NiceOrDie` scenario (TAU(∞, 0) does get an agreement).
    - **e2** (`all-rational`) shows success on the same domain when all outcomes are rational.
    - **e3** (`none-rational`) shows expected disagreement on the same domain when no outcomes are rational.
    - **e4** (`small`) runs the negotiation a relatively small domain (`Outfit`)
    - **e5** (`large`) runs the negotiation on a relatively large domain (`Kitchen`)
    - **e6** (`difficult`) runs the negotiation on a relatively large domain (`Outfit`) with larger reserved values (more difficult agreement)
    - **`free`** can be used to run on any domain.

data

- **`domains`** All the negotiation domains used for the experiments reported in the papers. These are all the domains of the ANAC 2013 competition and they can also be found on the GENIUS platform. Each domain folder consists of the following files:
  - **`{domain-name}.xml`** describes the outcome space in GENIUS XML format.
  - **`{domain-name}1.xml`** describes the utility function of first negotiator (GENIUS XML format).
  - **`{domain-name}2.xml`** describes the utility function of second negotiator (GENIUS XML format).
  - **`stats.json`** Statistics of the negotiation scenario like opposition, welfare of the Nash Bargaining Solution, etc
  - **`info.json`** More information about the negotiation scenario for different choices of reserved values including the fraction of rational outcomes `f_rational`.

  Note that folder names prefix each domain with the size of its outcome space.
- **`results`** Raw results of all negotiations in experiments (`e1` for experiment 1 and `e2` for experiment 2) as reported in the paper.
  - **`stats`** Contains the results of all statistical tests reported in the paper (as well as others not reported due to lack of space).

Software Requirements

- Python `3.9` or later (tested on MacOS with Python 3.10.5).
- Java `18` (tested on MacOS with OpenJDK 18 2022-03-22 build 18+36-2087).
  - only needed to run state-of-the-art negotiators (and Nice Tit for Tat)
- negmas `0.9.2` (our forked version included in this code repository and installed when following the instructions in the following section).
- genius-bridge `v0.2.4` (installed when following the instructions in the following section).
  - only needed to run state-of-the-art negotiators (and Nice Tit for Tat)

All experiments were done on MacOS 12.4 on a MacBook Pro with Apple M1 Pro processor and 16GB of RAM but should be reproducible on other operating systems as we only use OS-independent libraries in this work.

Installing Requirements

Please use your platform's preferred method to install Python 3.9+ and Java 18+.

To install other requirements run the following command within the `code` folder:

```
pip install -r requirement.txt
negmas genius-setup
```

Note that the later command will download the `negmas-genius` bridge and install in as:

```
$HOME/negmas/files/geniusbridge.jar
```

We assume that this is run within a virtual environment (as always recommended).

Running Experiments

Assumes that you installed requirements

To run the first experiment and generate Table 1, run the following within `code/experiments`:

```
> java -jar $HOME/negmas/files/geniusbridge.jar --silent --no-logs&
> experiment1.sh
> eval1.sh
```

To run the second experiment and generate Fig. 1, run the following within `code/experiments`:

```
> java -jar $HOME/negmas/files/geniusbridge.jar --silent --no-logs&
> experiment2.sh
> eval2.sh
```

- These experiments may take a LONG time mostly because of the relatively slow speed of some state-of-the-art algorithms.
- In case, the previous method did not work, it may be because of an issue with the NegMAS-GENIUS bridge. To investigate the issue you will need two terminals. In the first terminal run:

  ```
  > java -jar $HOME/negmas/files/geniusbridge.jar --verbose --debug --no-logs
  ```

  Now, In the second terminal run the remaining two lines. For example for the second experiment:

  ```
  > experiment2.sh
  > eval2.sh
  ```

Running example negotiations

You can use `code/experiments/tests/test*.py` to run single negotiations and test properties of TAU and AOP on different situations.

To run one of the five predefined examples given in the description above just pass its name. For example, to run the negotiation that shows that MiCRO is not complete, run:

```
> python testaop.py e1
```

or

```
> python testaop.py incomplete
```

To change the strategy pass it using `--strategy`, for example:

```
> python testaop.py e1 --strategy=Atlas3
```

To change the domain and run a free negotiation use the `free option`

```
> python testaop.py free --strategy=micro --domain=<path to the domain folder>
```

You can run similar negotiations using the TAU protocol

```
> python testtau.py e1
```

By default TAU$(\infty, 0)$ will be used, but you can change β by passing `--beta`. For example to show that TAU(∞, ∞) is not complete you can run:

```
> python testtau.py e1 --beta=-1
```

Passing `--beta` less than zero will be interpreted as ∞.