The Journal of Open Source Software

Matching: A Python library for solving matching games

Henry Wilde[1], **Vincent Knight**[1], **and Jonathan Gillard**[1]

1 School of Mathematics, Cardiff University

DOI: 10.21105/joss.02169

Software

- Review
- Repository
- Archive

Editor: Viviane Pons

Reviewers:

- @igarizio
- @mbdemoraes

Submitted: 28 February 2020
Published: 17 April 2020

Summary

Matching games allow for the allocation of resources and partnerships in a fair way. Typically, a matching game is defined by two sets of players that each have preferences over at least some of the elements of the other set. The objective of the game is then to find a mapping between the sets of players in which everyone is *happy enough* with their match.

One of the most ubiquitous matching games is the Stable Marriage Problem (SM). In SM, there are two distinct player sets of size N: a set of suitors S and a set of reviewers R. Each suitor must strictly rank all of the reviewers, and vice versa. This arrangement of suitors, reviewers, and their preferences is called a game of size N (Gale & Shapley, 1962).

In SM, a matching is any bijection M between S and R, and it is considered to be stable (i.e. no one has a reason to modify their current match) if it contains no blocking pairs. A blocking pair is defined as any pair $(s, r) \in S \times R$ that would rather be matched to one another than their current match. This definition differs between matching games but the spirit is the same in that a pair blocks a matching if their envy is mutually rational. Irrational envy would be where one player wishes to be matched to another over their current match but the other player does not (or cannot) reciprocate.

Consider the game of size three shown in Figure 1 as an edgeless graph with suitors on the left and reviewers on the right. Beside each vertex is the name of the player and their associated ranking of the complementary set's elements.

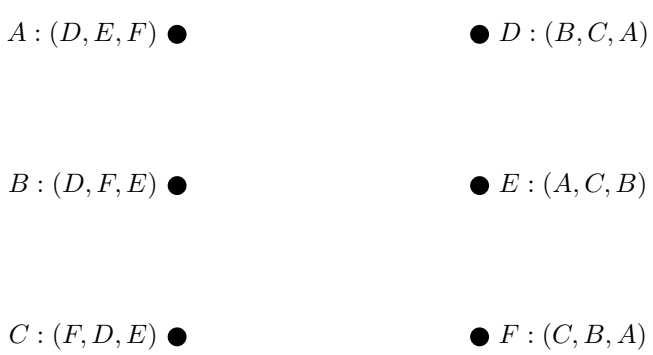

Figure 1: A game of size three.

Gale & Shapley (1962) presented an algorithm for finding a unique, stable and suitor-optimal matching to any instance of SM. The matching this algorithm produces is shown in Figure 2.

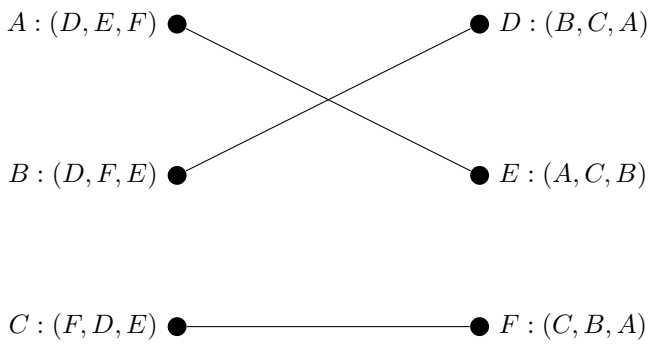

Figure 2: A stable, suitor-optimal solution.

Using Matching, this matching can be computed as follows:

```
>>> from matching.games import StableMarriage
>>> suitor_preferences = {
...     "A": ["D", "E", "F"], "B": ["D", "F", "E"], "C": ["F", "D", "E"]
... }
>>> reviewer_preferences = {
...     "D": ["B", "C", "A"], "E": ["A", "C", "B"], "F": ["C", "B", "A"]
... }
>>> game = StableMarriage.create_from_dictionaries(
...     suitor_preferences, reviewer_preferences
... )
>>> game.solve()
{A: E, B: D, C: F}
```

While it is relatively easy to find solutions to games like this with pen and paper, instances of other matching games tend to have more players than this and require the use of software to be solved in reasonable time.

Statement of Need

Matching games have applications in many fields where relationships between rational agents must be managed. Some example applications include: being able to inform on healthcare finance policy (Agarwal, 2017); helping to reduce the complexity of automated wireless communication networks (Bayat, Li, Song, & Han, 2016); and education infrastructure (Chiarandini, Fagerberg, & Gualandi, 2019). Thus, having access to software implementations of algorithms that are able to solve such games is essential.

The only current adversary to Matching is MatchingR (Tilly & Janetos, 2018). MatchingR is a package written in C++ with an R interface and its content overlaps well with that of Matching. However, the lack of a Python interface makes it less relevant to researchers and other users as Python's popularity grows both in academia and industry.

Matching is a Python library that relies on one core library from the standard scientific Python stack – NumPy (Oliphant, n.d.) – that currently implements algorithms for four types of matching games:

- the stable marriage problem (SM) (Gale & Shapley, 1962);

- the hospital-resident assignment problem (HR) (Gale & Shapley, 1962; Roth, 1984);
- the student-project allocation problem (SA) (Abraham, Irving, & Manlove, 2007);
- the stable roommates problem (SR) (Irving, 1985).

MatchingR implements all of these except SA but also implements an algorithm for the indivisible goods trading problem.

In addition to this, Matching has been developed to a high degree of best practice in research software development (Jiménez et al., 2017), and is thoroughly documented: matching.readthedocs.io. The documentation has been written to maximise its effect as a resource for learning about matching games as well as for the software itself. Furthermore, the software is automatically tested using example, integration, and property-based unit tests with 100% coverage. The current version of Matching has also been archived on Zenodo (The Matching library developers, 2020); as has all of the data used in the documentation tutorials.

Matching has been designed to be used as a research tool and to aid in the education of game theory. It is currently being used by a number of undergraduate students and postgraduate researchers in universities around the world, and has been used to massively streamline the final year project allocation process for one of the largest schools at Cardiff University (as an instance of SA). Furthermore, Matching proved to be instrumental in the practical implementation of a novel initialisation method for a categoric clustering algorithm (Wilde, Knight, & Gillard, 2020). With Matching being written in Python, this tool is widely accessible by programmers and non-programmers alike as a readable, portable, and reproducible piece of software.

References

Abraham, D. J., Irving, R. W., & Manlove, D. F. (2007). Two algorithms for the student-project allocation problem. *Journal of Discrete Algorithms*, *5*(1), 73–90. doi:10.1016/j.jda.2006.03.006

Agarwal, N. (2017). Policy analysis in matching markets. *American Economic Review*, *107*(5), 246–50. doi:10.1257/aer.p20171112

Bayat, S., Li, Y., Song, L., & Han, Z. (2016). Matching theory: Applications in wireless communications. *IEEE Signal Processing Magazine*, *33*, 103–122. doi:10.1109/MSP.2016.2598848

Chiarandini, M., Fagerberg, R., & Gualandi, S. (2019). Handling preferences in student-project allocation. *Annals of Operations Research*, *275*(1), 39–78. doi:10.1007/s10479-017-2710-1

Gale, D., & Shapley, L. S. (1962). College admissions and the stability of marriage. *The American Mathematical Monthly*, *69*(1), 9–15. doi:10.2307/2312726

Irving, R. W. (1985). An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, *6*(4), 577–595. doi:10.1016/0196-6774(85)90033-1

Jiménez, R. C., Kuzak, M., Alhamdoosh, M., Barker, M., Batut, B., Borg, M., Capella-Gutierrez, S., et al. (2017). Four simple recommendations to encourage best practices in research software. *F1000Research*, *6*, ELIXIR–876. doi:10.12688/f1000research.11407.1

Oliphant, T. (n.d.). NumPy: A guide to NumPy. USA: Trelgol Publishing. Retrieved from http://www.numpy.org/

Roth, A. (1984). The evolution of the labor market for medical interns and residents: A case study in game theory. *Journal of Political Economy*, *92*(6), 991–1016. doi:10.1086/261272

The Matching library developers. (2020). Matching: V1.3. doi:10.5281/zenodo.3751325

Tilly, J., & Janetos, N. (2018). MatchingR: Matching algorithms in R and C++. *GitHub repository*. GitHub. Retrieved from https://github.com/jtilly/matchingR

Wilde, H., Knight, V., & Gillard, J. (2020). A novel initialisation based on hospital-resident assignment for the k-modes algorithm. Retrieved from http://arxiv.org/abs/2002.02701