Supporting Information

for

ArbAlign: A Tool for Optimal Alignment of Arbitrarily Ordered Isomers Using the Kuhn-Munkres Algorithm

Berhane Temelso,[1,4] Joel M. Mabey,[1] Toshiro Kubota,[2] Nana Appiah-Padi,[1,3] George C. Shields[1, 4*]*

[1]Dean's Office, College of Arts and Sciences, and Department of Chemistry, Bucknell University, Lewisburg, PA 17837, USA

[2]Department of Mathematical Sciences, Susquehanna University, Selinsgrove, PA, 17870, USA

[3]Lewisburg Area High School, Lewisburg, PA, 17837, USA

[4]Current Address: Provost's Office and Department of Chemistry, Furman University, Greenville, SC 29613, USA

[*] Corresponding authors: berhane.temelso@furman.edu, george.shields@furman.edu

Table of Contents

1. README – Included tools and their usage

This folder contains the following items:

1.1. Code - a folder containing the following items:

ArbAlign-driver.py - a Python source code for the munkresRMSD script,

ArbAlign.py - a Python source code for the munkresRMSD script,

PrinCoords.py - a Python source code for calculating principal coordinates

genTypes.csh – a shell script to use OpenBabel to convert a Cartesian (XYZ) file to a SYBYL Mol2 file formatted as a conventional XYZ file

genConn.csh – a shell script to use OpenBabel to convert a Cartesian (XYZ) file to a NMA connectivity file formatted as a conventional XYZ file

RMSD-Kabsch.py - a Python script for calculating RMSDs by Jimmy Charnley Kromann (jimmy@charnley.dk) and Lars Bratholm's script from https://github.com/charnley/rmsd - Accessed on Nov 10, 2015. Please see rmsd-script/LICENSE for usage rights.

1.2. **Usage of the Driver Script (ArbAlign-driver.py):**

ArbAlign-driver.py is a Python driver script to run Kuhn-Munkres optimal RMSD matching

ArbAlign can be used either as a command line or web tool. The command line tool has a driver script that can take in many options or resort to sensible defaults when necessary.

Usage:
```
ArbAlign-driver.py
          [-b/--by {l, t, c}]
          [-n/--noHydrogens]
          [-s/--simple]
          filename_1.xyz filename_2.xyz
```

`-b {l,t,c}`, `--by {l,t,c}`	Match atoms by l-label, SYBYL t-type, or NMA connectivity (-c). The default is by atom label (-l)
`-s`, `--simple`	Perform Kuhn-Munkres assignment reordering without axes swaps and reflections. The default is to perform axes swaps and reflections
`-n`, `--noHydrogens`	Ignore hydrogens. The default is to include all atoms

If the pairs of structures pass a sanity test, the tool will align them optimally and provide the following information.

- The initial Kabsch RMSD,

- The Kuhn-Munkres reorderings for each atom and the corresponding RMSDs,
- The final Kabsch RMSD after the application of the Kuhn-Munkres algorithm, and
- The coordinates corresponding to the best alignment of the second structure with the first.

====

1.3. Coordinates - a folder containing Cartesian coordinates of the following systems

Waterclusters - a folder containing Cartesian Coordinates of water clusters of size 2-100

Neon-clusters - a folder containing Cartesian Coordinates of neon clusters of size 10-1000

Peptides - a folder containing Cartesian Coordinates of many conformers of five di- and tri-peptides

S1-MA-W1 - a folder containing Cartesian Coordinates of isomers of $(H_2SO_4)(NH_2CH_3)(H_2O)$

====

1.4. To run a successful alignment using these tools, one would need:

A web version is available at http://www.arbalign.org/. This implementation requires the following scripts and tools.

1. A Python script to convert molecules from arbitrary to principal coordinate system is included in the Supporting Information.
2. In cases where one wants to use atom types including connectivity and hybridization information, it is necessary to use OpenBabel[1] to convert the Cartesian coordinates to SYBYL Mol2 (sy2) and MNA (mna) formats.
3. A fast C++ implementation (`hungarian`)[2] of the Kuhn-Munkres algorithm is needed to solve the assignment problem. As described in https://github.com/hrldcpr/hungarian, one would need to download, build and install the module.

   A. First, download or clone the module:

   ```
   git clone https://github.com/hrldcpr/hungarian
   ```

   B. Then go into the folder and build the module by executing

   ```
   python setup.py build
   ```

C. If this step goes smoothly, you can either put the file `'build/lib-*/hungarian.so'` in the directory where the code using it resides or you can install it in a central location where all of your python programs can see it by entering

```
python setup.py install
```

4. Please make sure you have the `Numpy` module[3] which is needed on top of other standard packages in Python.

2. Cost and Efficiency

A brute force approach to the atom assignment problem is very limited because it scales factorially with the largest element of a molecule's automorphism class, whether it is based on atom name, type or connectivity. Therefore, our current approach needs to have substantially lower scaling to be useful. In general, the most expensive step in our method is the atom re-ordering using the Kuhn-Munkres algorithm. This step typically scales as $O(N^3)$ where N is the dimension of the cost matrix to be solved for making an assignment. N is not the total number of atoms in a molecule, but rather the number of atoms of a given name, type or connectivity. The leading term in the cost of this algorithm is the number of atoms of most common atom name, type or connectivity in the structures being aligned. For example, when aligning two water decamer, $(H_2O)_{10}$, structures, the most expensive step would be optimally assigning the 20 hydrogen atoms by solving a 20x20 distance matrix using the Kuhn-Munkres algorithm. Next, we would assign the 10 oxygen atoms by solving the 10x10 distance matrix. And we need to perform these operations forty-eight times for every atom name, type or connectivity. In the end, we would have done 96 Kuhn-Munkres assignments and compiled 96 RMSD values that would be used to determine the best assignments.

To analyze the cost of the Kuhn-Munkres algorithm in the absence of other steps necessary for its application to molecular systems, we applied it to randomly generated cost matrices and plotted the timing in **Figure S1**. The cost of the Python (Py)[4] and C++[2] implementations of the Kuhn-Munkres algorithm differ substantially. In **Figure S2**, we show that the application of the algorithm for assignment of water clusters, $(H_2O)_n$ of size n=10-100, leads to similar conclusions about the cost as in the generic cost matrix case shown in **Figure S1**. The C++ implementation takes two to four orders of magnitude less time than the Python analog. Thus, it can be applied to optimally align very large molecules or perform a high throughout study of small molecules efficiently.

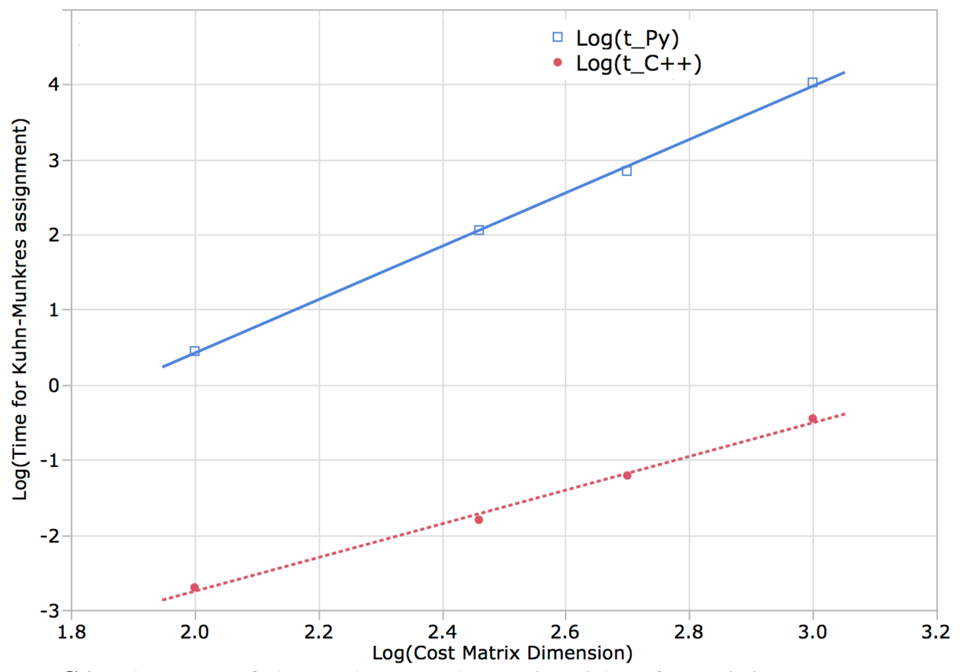

Figure S1. The cost of the Kuhn-Munkres algorithm for solving a square matrix of size 100 to 1000. The Python (Py) and C++ implementations show very different cost. The C++ implementations of the Kuhn-Munkres algorithm is significantly faster than the Python implementation.

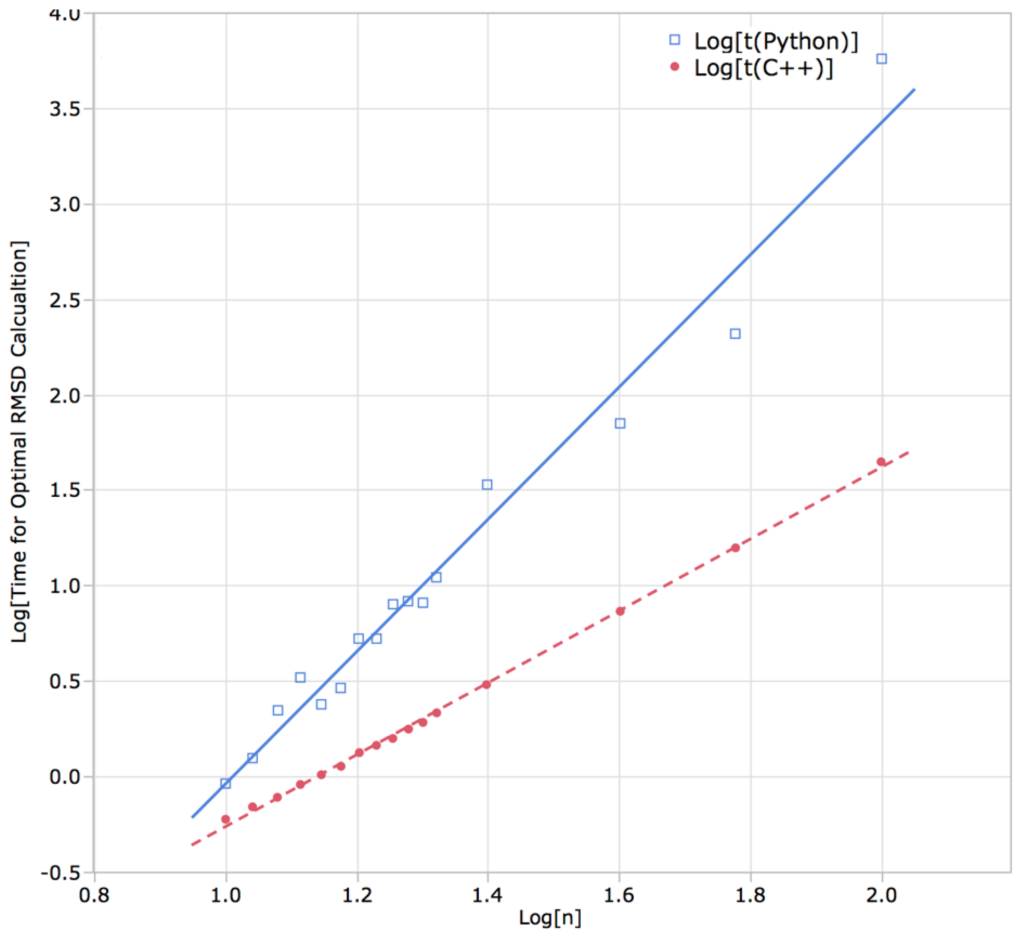

Figure S2. The cost of matching two arbitrary water clusters, $(H_2O)_n$, n=10-100 using our method. The C++ implementations of the Kuhn-Munkres algorithm is significantly faster than the Python implementation.

3. Examples

Conventional (Kabsch) RMSD calculations are compared with those using the Kuhn-Munkres (KM) algorithm based on atom name, atom type or atom connectivity for a series of systems next.

3.1. (H2SO4)2(DMA)(W)

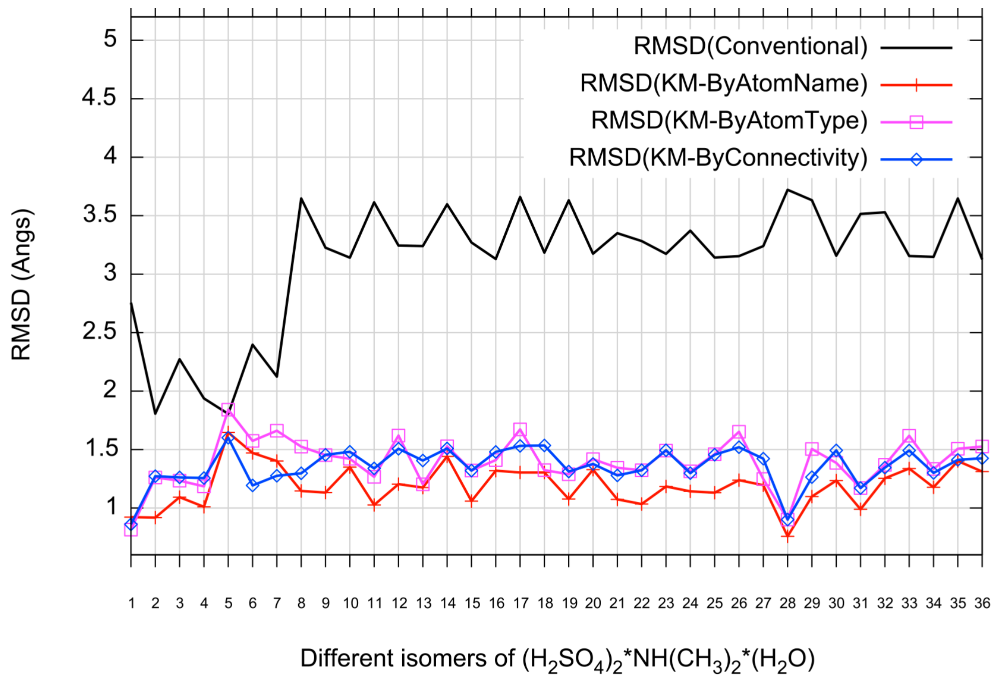

Figure S3. A comparison of conventional and Kuhn-Munkres RMSD matching for isomers of a complex containing two sulfuric acid, one methylamine and one water molecule. The RMSD is between the most stable isomer and the next thirty-six isomers. Assignments based on atom names alone generally yield the lowest RMSDs, but factoring in atom type and connectivity often yields a more meaningful overlay albeit a larger RMSD.

3.2. Peptides – GFA, GGF, WG, WGG

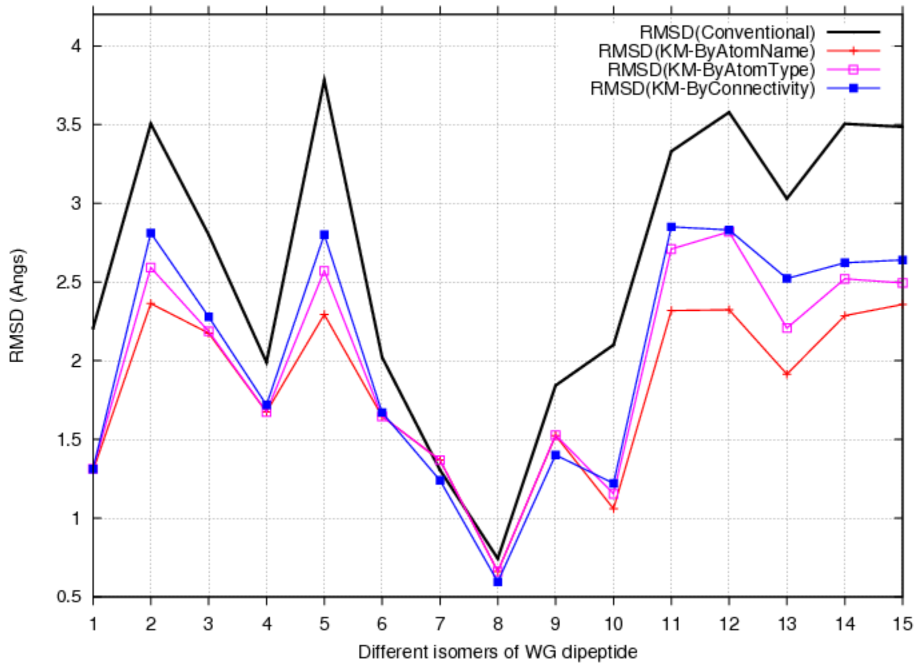

Figure S4. A comparison of conventional and Kuhn-Munkres RMSD matching for isomers of GFA tripeptide. The RMSD is between the most stable isomer and the next fourteen isomers.

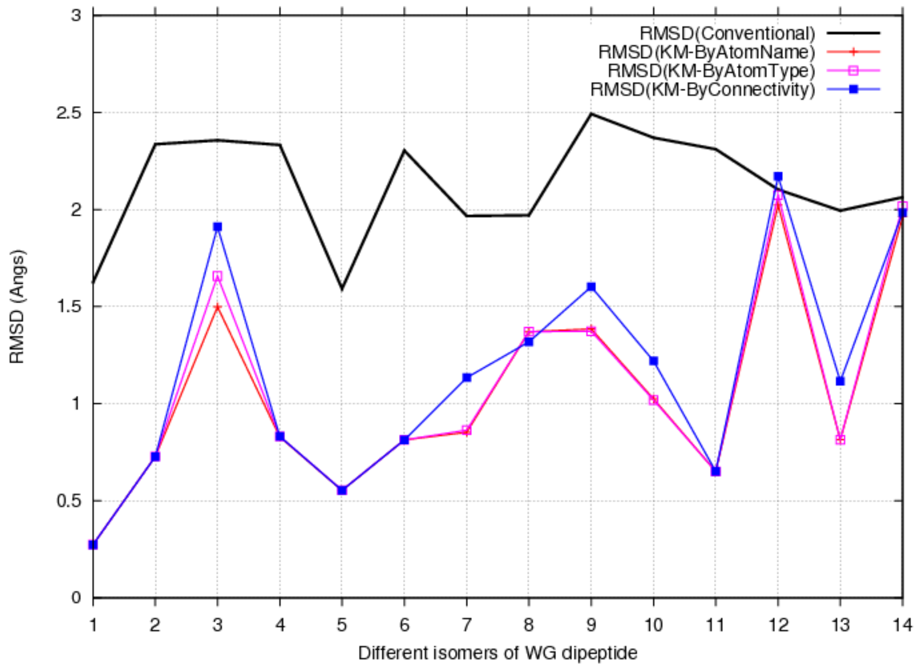

Figure S5. A comparison of conventional and Kuhn-Munkres RMSD matching for isomers of GGF tripeptide. The RMSD is between the most stable isomer and the next fourteen isomers.

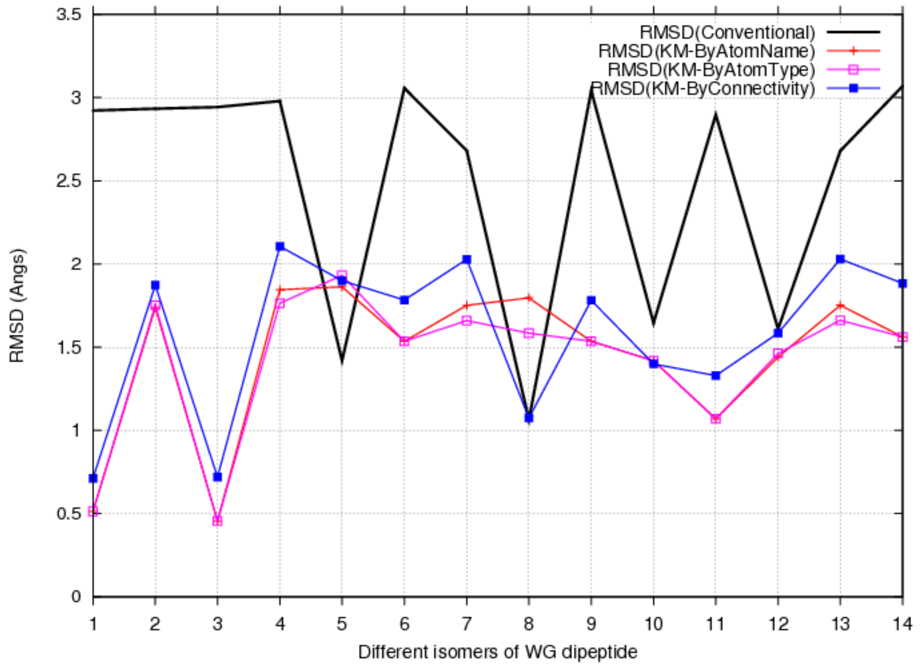

Figure S6. A comparison of conventional and Kuhn-Munkres RMSD matching for isomers of WGG tripeptide. The RMSD is between the most stable isomer and the next fourteen isomers.

3.3. Acenaphthene + Water

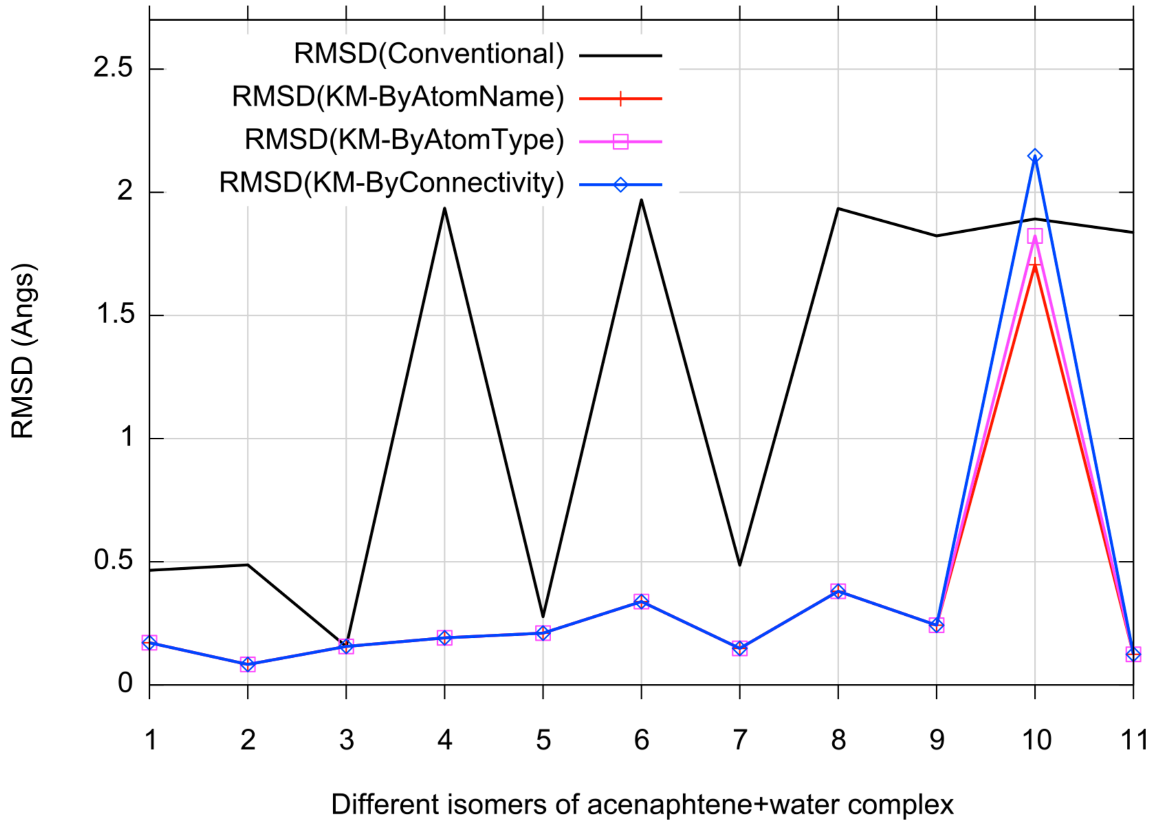

Figure S7. A comparison of conventional and Kuhn-Munkres RMSD matching for isomers of one acenaphtene and one water. The RMSD is between the most stable isomer and the next eleven isomers.

References

(1) OBoyle, N.; Banck, M.; James, C.; Morley, C.; Vandermeersch, T.; Hutchison, G. Open Babel: An Open Chemical Toolbox. *J. Cheminform.* **2011**, *3*, 33.

(2) Cooper, H. *Hungarian: Munkres' Algorithm For The Linear Assignment Problem, In Python*. https://github.com/Hrldcpr/Hungarian; (accessed January 15, 2015).

(3) van der Walt, S.; Colbert, S. C.; Varoquaux, G. The Numpy Array: A Structure For Efficient Numerical Computation. *Computing in Science & Engineering* **2011**, *13*, 22–30.

(4) Clapper, B. M. *Munkres 1.0.8: Munkres Algorithm For The Assignment Problem*. https://pypi.python.org/Pypi/Munkres; (accessed March 15, 2016).