# Munkre’s Assignment Algorithm

The goal of Munkre’s Assignment Algorithm is to find the optimal assignment of an assignment problem. For example, let us consider the case where workers can be chosen to perform tasks represented in the table below, with each worker asking for a different price to perform each task. The resultant matrix formed from the cost of hiring each worker to do each task is known as the cost matrix.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Worker 1** | **Worker 2** | **Worker 3** |
| **Task 1** | $3 | $4 | $3 |
| **Task 2** | $3 | $3 | $4 |
| **Task 3** | $4 | $5 | $4 |

In order to find the optimal assignment, we could first consider the case where worker 1 is assigned to task 1, worker 2 to task 2 and worker 3 is assigned to task 3. This results in a total cost of . To identify the optimal assignment which generates the total cost, we would need to permutate through each combination of worker and task. This means that we would next need to test the combination of worker 1 with task 1, worker 2 with task 3 and worker 3 with task 3, followed by worker 1 with task 2, worker 2 with task 1, worker 3 with task 3 and so on, for a total of 9 iterations through the matrix. This leads to a complexity of order .

The Munkre’s Assignment Algorithm is simply an algorithm that is able to find the optimal assignment of the same cost matrix with a lower complexity of order .

To avoid misinterpretation, below is a step by step description of the algorithm from <http://csclab.murraystate.edu/~bob.pilgrim/445/munkres.html> to aid programmers in the implementation of the algorithm.

***Step 0:****Create a matrix called the cost matrix in which each element represents the cost of assigning one of workers to one of jobs.  Rotate the matrix so that there are at least as many columns as rows and let .*

***Step 1:****For each row of the matrix, find the smallest element and subtract it from every element in its row.  Go to Step 2.*

***Step 2:****Find a zero (Z) in the resulting matrix.  If there is no starred zero in its row or column, star Z. Repeat for each element in the matrix. Go to Step 3.*

***Step 3:****Cover each column containing a starred zero.  If K columns are covered, the starred zeros describe a complete set of unique assignments.  In this case, Go to DONE, otherwise, Go to Step 4.*

***Step 4:****Find a noncovered zero and prime it.  If there is no starred zero in the row containing this primed zero, Go to Step 5.  Otherwise, cover this row and uncover the column containing the starred zero. Continue in this manner until there are no uncovered zeros left. Save the smallest uncovered value and Go to Step 6.*

***Step 5:****Construct a series of alternating primed and starred zeros as follows.  Let Z0 represent the uncovered primed zero found in Step 4.  Let Z1 denote the starred zero in the column of Z0 (if any). Let Z2 denote the primed zero in the row of Z1 (there will always be one).  Continue until the series terminates at a primed zero that has no starred zero in its column.  Unstar each starred zero of the series, star each primed zero of the series, erase all primes and uncover every line in the matrix.  Return to Step 3.*

***Step 6:****Add the value found in Step 4 to every element of each covered row, and subtract it from every element of each uncovered column.  Return to Step 4 without altering any stars, primes, or covered lines.*

***DONE:****Assignment pairs are indicated by the positions of the starred zeros in the cost matrix.  If C(i,j) is a starred zero, then the element associated with row i is assigned to the element associated with column j.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The function linear\_sum\_assignment located in scipy’s opitimize module implements the algorithm to perform the assignment in several steps which will be outlined later. However, before that, it is worth nothing that the function accepts rectangular matrixes, it simply transposes the matrix if the number of rows is larger than the number of columns to ensure that the number of columns, is greater than the number of rows. It then returns assignments, one for each column.  The following steps have been inferred from the source code at <https://github.com/scipy/scipy/blob/v0.18.1/scipy/optimize/_hungarian.py#L13-L107> and thus may have been subject to misinterpretation.   |  |  |  |  | | --- | --- | --- | --- | | 2 | 3 | 4 | 3 | | 4 | 3 | 4 | 5 | | 2 | 4 | 3 | 4 | | 4 | 5 | 4 | 3 |   Step 1:  First, we loop through the matrix row by row, subtracting the lowest element from each element in the row, resulting in at least one zero (from subtracting the element with itself) in the row. Each zero located in represents a task which is done most efficiently by worker , or in terms of the original cost matrix, at the lowest cost. At this point it is possible that an optimal solution can be found. To check for it, we proceed to the next step.  For illustration, we shall follow an example from this step on to better understand the steps. Furthermore, we will hide all non-zero values not in step 4 or 6 as required.   |  |  |  |  | | --- | --- | --- | --- | | 0 | 1 | 2 | 1 | | 1 | 0 | 1 | 2 | | 0 | 2 | 1 | 2 | | 1 | 2 | 1 | 0 |   Step 2:  We star each zero in the matrix at with no other starred zeros in row and column to denote that it an optimal worker-task pair.   |  |  |  |  | | --- | --- | --- | --- | | 0 |  |  |  | |  | 0\* |  |  | | 0 |  |  |  | |  |  |  | 0\* |   Step 3:  We cover each column containing a starred zero. If columns are covered, the state represents an optimal assignment and we can return the position of marked zeros as the result. Otherwise, we continue to the next step.   |  |  |  |  | | --- | --- | --- | --- | | 0 |  |  |  | |  | 0\* |  |  | | 0 |  |  |  | |  |  |  | 0\* |   Step 4:  Find an uncovered zero and prime it. If there is no starred zero in the row containing the primed zero, then we continue to the next step. If not, cover this row and uncover the column containing the starred zero. Continue doing so until there are no more uncovered zeros. Take note of the value of the smallest uncovered value and move on to Step 6.   |  |  |  |  | | --- | --- | --- | --- | | 0’ |  |  |  | |  | 0\* |  |  | | 0 |  |  |  | |  |  |  | 0\* |   Step 5:  Construct a series of alternating primed and starred zeros as follows. Denote the primed zero without starred zeros in its row identified in step 4 as . Denote the starred zero in its column, if any, as . Denote the starred zero in the row of as . Continue until the series terminates at a primed zero with no starred zero in its column, i.e. unable to find .   |  |  |  |  | | --- | --- | --- | --- | | 0’ |  |  |  | |  | 0\* |  |  | | 0 |  |  |  | |  |  |  | 0\* |   Then unstar each starred zero in the series, star each primed zero in the series and erase all primes and uncover every line in the matrix before returning to step 3.   |  |  |  |  | | --- | --- | --- | --- | | 0\* |  |  |  | |  | 0\* |  |  | | 0 |  |  |  | |  |  |  | 0\* |   At this point we should follow the example to step 3 and then 4 and so on. However before that, let me describe the last step.  Step 6:  Add the value found in step 4 to every element of each covered row and subtract it from every element of each uncovered column. Return to step 4 without altering any stars, primes or covered lines. |
| Now back to the example.  Step 3:   |  |  |  |  | | --- | --- | --- | --- | | 0\* |  |  |  | |  | 0\* |  |  | | 0 |  |  |  | |  |  |  | 0\* |   Step 4:   |  |  |  |  | | --- | --- | --- | --- | | 0\* | 1 | 2 | 1 | | 1 | 0\* | 1 | 2 | | 0 | 2 | 1 | 2 | | 1 | 2 | 1 | 0\* |   Smallest uncovered value: 1  Step 6:   |  |  |  |  | | --- | --- | --- | --- | | 0\* | 1 | 1 | 1 | | 1 | 0\* | 0 | 2 | | 0 | 2 | 0 | 2 | | 1 | 2 | 0 | 0\* |   Step 4:   |  |  |  |  | | --- | --- | --- | --- | | 0\* |  |  |  | |  | 0\* | 0’ |  | | 0 |  | 0’ |  | |  |  | 0 | 0\* |   Step 5:   |  |  |  |  | | --- | --- | --- | --- | | 0\* |  |  |  | |  | 0\* | 0’ |  | | 0 |  | 0’ |  | |  |  | 0 | 0\* |  |  |  |  |  | | --- | --- | --- | --- | | 0\* |  |  |  | |  | 0\* | 0 |  | | 0 |  | 0\* |  | |  |  | 0 | 0\* |   Step 3:   |  |  |  |  | | --- | --- | --- | --- | | 0\* |  |  |  | |  | 0\* | 0 |  | | 0 |  | 0\* |  | |  |  | 0 | 0\* |   Final assignments: (1, 1), (2, 2), (3, 3), (4, 4)   |  |  |  |  | | --- | --- | --- | --- | | 2 | 3 | 4 | 3 | | 4 | 3 | 4 | 5 | | 2 | 4 | 3 | 4 | | 4 | 5 | 4 | 3 |   Total cost = 2+3+3+3 = 11 |

# Basic cost matrix

Table cost matrix of detections and tracks with row and column headers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Detection 1** | **Detection 2** | **Detection 3** | **…** | **Detection n** |
| **Track 1** | Distance 1, 1 | Distance 1, 2 | Distance 1, 3 | Distance 1, j | Distance 1, n |
| **Track 2** | Distance 2, 1 | Distance 2, 2 | Distance 2, 3 | Distance 2, j | Distance 2, n |
| **Track 3** | Distance 3, 1 | Distance 3, 2 | Distance 3, 3 | Distance 3, j | Distance 3, n |
| **…** | Distance i, 1 | Distance i, 2 | Distance i, 3 | Distance i, j | Distance i, n |
| **Track n** | Distance n, 1 | Distance n, 2 | Distance n, 3 | Distance n, j | Distance n, n |

The basic cost matrix has detections as columns and tracks as rows. It is a square matrix which represents detections to be assigned to tracks. represents the cost to assign the detection to the track, calculated based on the Euclidean distance between the detection and the estimated position of the track in the frame.

In the case of uneven detections and tracks, i.e. detections to be assigned to tracks, the cost matrix still has to be a square matrix where . A common strategy to convert the rectangular matrix into a square matrix is to fill the missing rows or columns with zeros, where . An example with , m is shown below. In this way, the Munkre’s assignment algorithm can still produce assignments which will match the best track detection pairs.

Table cost matrix of 4 detections and 2 tracks with row and column headers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Detection 1** | **Detection 2** | **Detection 3** | **Detection 4** |
| **Track 1** | Distance 1, 1 | Distance 1, 2 | Distance 1, 3 | Distance 1, 4 |
| **Track 2** | Distance 2, 1 | Distance 2, 2 | Distance 2, 3 | Distance 2, 4 |
| **-** | 0 | 0 | 0 | 0 |
| **-** | 0 | 0 | 0 | 0 |

The problem with this approach is that the best track detection pairs may not always be optimal for the problem. Consider the case where the best track detection pairs are still a considerable distance from each other due to the tracked object being occluded in the frame resulting in its track being matched to noise. Clearly this is undesirable, especially in the case where the noise was not generated by the object as it would introduce errors into our predictions. In order to account for this deficiency, two changes need to be made. Firstly, additional dummy tracks and detections need to be added in to allow for non-assignment, where assignment of actual tracks or detections to these dummy tracks and detections would represent an unassigned detection and track respectively. Secondly, to avoid all tracks and detections being matched to dummy tracks and detections, a threshold value needs to be introduced, representing the cost of non-assignment as opposed to matching distant tracks and detections. This cost represents the threshold distance and is similarly measured in the same units as the cost of assignment.

# Modified cost matrix

The modified cost matrix is based off the assignDetectionsToTracks function from MATLAB and is where the idea of using a padded cost matrix combined with separate costs for unassigned detections and tracks came from.

To keep things simple, let us consider the case where , m shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Detection 1** | **Dummy Detection** | **Dummy**  **Detection** |
| **Track 1** | Distance 1, 1 | Cost of unassigned track | Cost of unassigned track |
| **Track 2** | Distance 2, 1 | Cost of unassigned track | Cost of unassigned track |
| **Dummy Track** | Cost of unassigned detection | 0 | 0 |

First off, note that the size of the matrix is . The matrix has square to allow the application of Munkre’s assignment algorithm and has maximum dimensions of in order to accommodate the case where all detections and tracks are located outside the threshold distance from each other. In this case, all actual tracks and detections need to be assigned to dummy tracks and detections, requiring dummy tracks and dummy detections. This results in dimensions of which can be rewritten as

Secondly, the modified cost matrix is divided into four sections. Going clockwise, they are the valid assignments, unassigned tracks, dummy pairs and unassigned detections. To understand how these sections work, let us consider the following cases.

When the first detections are made, there are no tracks created yet. In the case of , m, all detections can only be made to the unassigned detection section.

|  |  |  |
| --- | --- | --- |
|  | **Detection 1** | **Detection 2** |
| **Dummy Track** | Cost of unassigned detection | Cost of unassigned detection |
| **Dummy Track** | Cost of unassigned detection | Cost of unassigned detection |

When the number of objects in the scene does not change, the ideal case would be that all detections would have come from tracked objects leading to a case where and where for all detections and tracks is lower than the cost of assignment. In the case of , m, assignments will only be made to the valid assignment and dummy pairs sections.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Detection 1** | **Detection 2** | **Dummy Detection** | **Dummy**  **Detection** |
| **Track 1** | Distance 1, 1 | Distance 1, 2 | Cost of unassigned track | Cost of unassigned track |
| **Track 2** | Distance 2, 1 | Distance 2, 2 | Cost of unassigned track | Cost of unassigned track |
| **Dummy Track** | Cost of unassigned detection | Cost of unassigned detection | 0 | 0 |
| **Dummy Track** | Cost of unassigned detection | Cost of unassigned detection | 0 | 0 |

Most likely however, noise and occlusions will result in a mismatch in the number of detections and tracks. Let us first consider the case of noise, when extra detections further than the threshold distance from any track are generated. In the case where , m, assignments will first be made to match the detection generated by the tracked object and the dummy pair. The remaining assignment can thus only be made between the remaining detection and the remaining dummy track.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Detection 1** | **Detection 2** | **Dummy Detection** |
| **Track 1** | Distance 1, 1 | Distance 1, 2 | Cost of unassigned track |
| **Dummy Track** | Cost of unassigned detection | Cost of unassigned detection | 0 |
| **Dummy Track** | Cost of unassigned detection | Cost of unassigned detection | 0 |

In the case where a tracked object has been occluded, it is likely that the track will be located further than the threshold value from the other detections. In the case where , m, assignments will once again be made to the valid assignments and dummy pairs first, before matching the remaining track with the remaining dummy detection.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Detection 1** | **Dummy Detection** | **Dummy Detection** |
| **Track 1** | Distance 1, 1 | Cost of unassigned track | Cost of unassigned track |
| **Track 2** | Distance 2, 1 | Cost of unassigned track | Cost of unassigned track |
| **Dummy Track** | Cost of unassigned detection | 0 | 0 |

New objects appearing in the frame behave similarly to noise in the context of a single frame, in that they result in extra detections far away from existing tracks. Objects leaving the frame behave similarly to occlusions in that they result in extra tracks located far away from existing tracks.

Finally let us consider a case where both noise and occlusions occur simultaneously. In such a case, there will be both tracks located far from detections and detections located far from tracks. While the number of detections could differ from the number of tracks, for simplicity let us consider a case where , m, all detections arise from noise and all tracks are occluded. Assuming the threshold distances are appropriately chosen, they will be lower than the distances within the valid assignments section. In this case, the assignment algorithm is likely to chose to match tracks with dummy detections and detections with dummy tracks, with no assignments made to the valid assignments or dummy pairs sections.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Detection 1** | **Detection 2** | **Dummy Detection** | **Dummy**  **Detection** |
| **Track 1** | Distance 1, 1 | Distance 1, 2 | Cost of unassigned track | Cost of unassigned track |
| **Track 2** | Distance 2, 1 | Distance 2, 2 | Cost of unassigned track | Cost of unassigned track |
| **Dummy Track** | Cost of unassigned detection | Cost of unassigned detection | 0 | 0 |
| **Dummy Track** | Cost of unassigned detection | Cost of unassigned detection | 0 | 0 |

Of course, with a mix of detections arising from tracks, noise and occlusions, it is likely that in practice assignments will be made to all four sections. In fact, this is the desired behaviour, where the assignment algorithm can properly categorise the detections and tracks into these distinct sections. However, depending on the threshold distance chosen as well as how detections are generated, it is entirely possible that incorrect assignments may occur.

Therefore, other filtering techniques such as pre-processing to remove as much noise as possible and making tracked objects more easily detectable as well as time dependent techniques such as monitoring the