Linear assignment

The assignment problem is a classical problem in graph theory and combinatorial optimization[?]. Informally, it deals with finding the most optimal (least-cost) way of assigning tasks to workers. It also has an equivalent graph theory formulation. We call such an assignment problem problem *linear* if there are equal numbers of tasks and workers and the total cost for all tasks is the sum of the individual costs incurred. This is a quite generalized problem that has many applications in resource allocation, including originally in transportation theory.

Formal specification

Let X and Y be two sets of equal size. Let C be a cost function

$$C : X \times Y \to \mathbb{R}$$

The problem is to find a bijection $f : A \to T$ such that the cost function is minimized:

$$\sum_{x \in X} C(x, f(x))$$

The linear assignment problem is often given as a cost matrix

$$C \mid c_{ij} = \text{cost of moving element } i \text{ to } j$$

Equivalently, in graph theoretic terms: given a weighted bipartite graph, find the minimum weight perfect matching.

Motivations

This problem first arose in transportation theory, where the goal is to assign n mines to n factories which consume the ore that the mines produce. However, the problem arises whenever there is a need to assign tasks to workers with potentially different costs for each potential task-worker pair.

We investigate this problem in order to align the discrete probability distributions of two shapes and then measure the distance between them. This is explained in the **sliding wavelets** section.

Algorithms and runtime

Because this problem can be formulated as a linear program, it can be solved with general linear program solving algorithms (e.g. simplex and variants). However, we can solve this more efficiently by exploiting the structure of the problem.

One famous algorithm is the famous Hungarian algorithm, which was proposed by Kuhn in the 1950s and originally had time complexity of $O(n^4)$, although this was improved by Lawler in 1976 to $O(n^3)$.

We use the Jonker-Volgenant algorithm[**?**][**?**], which is notably faster in a practical sense.