Models and Algorithms for

Matching and Assignment Problems

Silvano Martello
DEI "Guglielmo Marconi", Università di Bologna, Italy

7. Other linear assignment problems

Ranking AP Solutions

- When AP models real-life problems, it can be useful to also determine the second, third, ..., K-th best solution;
- applications in multiobjective programming (Pedersen, Nielsen and Andersen (2005), Przybylski, Gandibleu and Ehrgott (2005));
- military applications (radar tracking problems).
- Algorithms:
  - Murty (1968): truncated branch-decision tree, time complexity $O(Kn^4)$;
  - Chegireddy and Hamacher (1987) different branching rule, time complexity $O(Kn^3)$;
  - Pascoal, Captivo and Clímaco (2003): modified Murty's algorithm, time $O(Kn^3)$.

K-Cardinality Assignment Problem

- Assign exactly K rows to K columns so that the sum of the corresponding costs is a minimum.
- Mathematical model:

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} \leq 1 \qquad (i = 1, 2, \ldots, n),$$

$$\sum_{i=1}^{n} x_{ij} \leq 1 \qquad (j = 1, 2, \ldots, n),$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} = K,$$

$$x_{ij} \in \{0, 1\} \qquad (i, j = 1, 2, \ldots, n).$$

- Generalization of the assignment problem:

  AP is the special case arising when $K = n$.

- Dell'Amico and Martello (1997)
  - the constraint matrix of the problem is totally unimodular;
  - primal algorithm, time complexity $O(n^3)$.

Linear Bottleneck Assignment Problem

- Given an $n \times n$ cost matrix $C = (c_{ij})$, find a permutation φ of $\{1, 2, \ldots, n\}$ that **minimizes the maximum cost** used:

$$\min_{\varphi} \max_{1 \le i \le n} c_{i\varphi(i)}.$$

- Mathematical model: as AP, with objective function

$$\min \max c_{ij} x_{ij}.$$

- **Applications:**

  scheduling: given n machines and n jobs, each of which has to be executed on one machine, given a matrix $T = (t_{ij})$ with t_{ij} = processing time if job i is executed on machine j, assign each job to a different machine s.t. the latest completion time is minimized;

  military: track missiles in space given their locations at two different times;

  transportation: bus driver rostering.

- **Observation 1.** The optimal solution φ^* depends only on the relative order of the cost coefficients, not on their numerical value.
  - **Example:**
$$C = \begin{pmatrix} \pi & \sqrt{3} & 54392 \\ 0 & e^3 & e^3 \\ -2 & \sin \pi/8 & 4 \end{pmatrix};$$
  - ordering: $-2 < 0 < \sin \pi/8 < \sqrt{3} < \pi < 4 < e^3 = e^3 < 54392$;
  - the smallest element -2 can be modeled by 0; the second smallest element 0 can be modeled by 1, and so on. New cost matrix:
$$C = \begin{pmatrix} 4 & 3 & 7 \\ 1 & 6 & 6 \\ 0 & 2 & 5 \end{pmatrix};$$
  - optimal solution: $\varphi^* = (2, 1, 3)$ (same for the two matrices).
- **Observation 2.** Suppose a value c^* is selected as **threshold**.
  Bipartite graph $G = (U, V; E)$ with $|U| = |V| = n$ and edges $[i, j] \in E$ iff $c_{ij} \leq c^*$.
  If G contains a perfect matching then c_{ij} is a feasible solution value.
  Optimal solution value = smallest c_{kl} such that, for $c^* = c_{kl}$, G contains a perfect matching.
- **Trivial algorithm:** start with $c^* = \max\{c_{ij}\}$ and decrease it to second maximum, third maximum, ... until the first value is found for which G does not contain a perfect matching.
  Time complexity $O(T(n)n^2)$ ($T(n)$ = time for testing the existence of a perfect matching).

A better algorithm:

Algorithm Threshold (**comment:** basic scheme)

select a c_{ij} value as *threshold* value c^*;

repeat

define a bipartite graph $G[c^*]$ having edges $[i, j]$ if and only if $c_{ij} \leq c^*$;

if $G[c^*]$ has a perfect matching **then** appropriately decrease c^* to a smaller c_{ij} value

else appropriately increase c^* to a larger c_{ij} value

until two thresholds $c_{i_1 j_1} < c_{i_2 j_2}$ (consecutive in value) are found such that

$G[c_{i_2 j_2}]$ has a perfect matching and $G[c_{i_1 j_1}]$ does not;

the optimal solution is produced by $c_{i_2 j_2}$.

- Different reasonable ways to increase/decrease c^*, avoiding to test the same value twice;
- Good choice: start with the **median**: $c^* := \min\{c_{kl} : |\{c_{ij} : c_{ij} \leq c_{kl}\}| \geq n^2/2\}$;

  at each iteration, take the median of one of the two halves;

  Time complexity: $O(T(n) \log n)$;

  Using the Alt et al. $O(n^{1.5}\sqrt{m/\log n})$ matching algorithm, $O(n^{2.5}\sqrt{\log n})$ (dense graphs).
- Best (very complicated) implementation: **time complexity:** $O(n^{2.5}/\sqrt{\log n})$ (dense graphs).
- **Other approaches: Augmenting path algorithms**, **Dual algorithms**.

- **Example:**

$$C = \begin{pmatrix} 8 & 2 & 3 & 3 \\ 2 & 7 & 5 & 8 \\ 0 & 9 & 8 & 4 \\ 2 & 5 & 6 & 3 \end{pmatrix}; \quad \textbf{median} = 4.$$

$G[4]$ has the edges corresponding to red entries (maximum matching in bold, underlined):

$$C[4] = \begin{pmatrix} 8 & \underline{\mathbf{2}} & 3 & 3 \\ \underline{\mathbf{2}} & 7 & 5 & 8 \\ 0 & 9 & 8 & \underline{4} \\ 2 & 5 & 6 & 3 \end{pmatrix}; \; \boldsymbol{c^*} \textbf{ is increased: new median} = 7.$$

$$C[7] = \begin{pmatrix} 8 & \underline{\mathbf{2}} & 3 & 3 \\ \underline{\mathbf{2}} & 7 & 5 & 8 \\ 0 & 9 & 8 & \underline{\mathbf{4}} \\ 2 & 5 & \underline{\mathbf{6}} & 3 \end{pmatrix}; \; \boldsymbol{c^*} \textbf{ is decreased: new median} = 5.$$

$$C[5] = \begin{pmatrix} 8 & \underline{\mathbf{2}} & 3 & 3 \\ 2 & 7 & \underline{\mathbf{5}} & 8 \\ \underline{\mathbf{0}} & 9 & 8 & 4 \\ 2 & 5 & 6 & \underline{\mathbf{3}} \end{pmatrix}; \; \textbf{optimal solution } c^* = 5.$$

Other algorithms for the Bottleneck Assignment Problem

- Dual/Augmenting path algorithms
  - start with a lower bound as threshold value c^*
    e.g., $\max\left(\max_i\{\text{minimum cost in row } i\}, \max_j\{\text{minimum cost in column } j\}\right)$;
  - find a maximum matching in the bipartite graph having edges $[i, j]$ if and only if $c_{ij} \leq c^*$;
  - **while** the matching is not perfect,
    - Augmenting path algorithms (Derigs and Zimmermann, 1978):
    - find a special augmenting path (*b-shortest augmenting path*, modified Dijkstra) which increases the value of c^*, and iterate.
    - Dual algorithms (Carpaneto and Toth, 1981):
    - take the minimum cost not covered by the matching as new c^*, and iterate.
- Implementations with best time complexities:
  - $O(m\sqrt{n \log n})$, Gabow and Tarjan (1988);
  - $O(n\sqrt{mn})$, Punnen and Nair (1994).

Balanced Assignment Problem

- **Balanced optimization problems**:

  Given a set S, a family $\mathcal{F}$ of feasible subsets of S and a cost $c(s)$ associated with every $s \in S$

  find $F \in \mathcal{F}$ for which $\max\{c(s) - c(s') \ : \ s, s' \in F\}$ is minimized

  (i.e., minimize the difference between the largest and the smallest value used)

  introduced by Martello, Pulleyblank, Toth, and de Werra, 1984:

- **General result proved:** if a solution can be tested for feasibility in polynomial time,

  then any balanced optimization problem can be solved in polynomial time.

- **Balanced Assignment Problem:** Given an $n \times n$ cost matrix $C = (c_{ij})$, find a permutation φ of $\{1, 2, \ldots, n\}$ that minimizes the difference between maximum and minimum cost in the assignment:

$$\min_{\varphi}(\max_{i}\{c_{i\varphi(i)}\} - \min_{i}\{c_{i\varphi(i)}\})$$

- **Application:** c_{ij} = expected life of a component j produced by company i:

  choose the components so that all components will have to be replaced at around the same time

- **Observation:** Let c^* be the solution value of the **Bottleneck AP** on the same matrix:

  any feasible AP solution must use elements of **value at least equal to** c^*.

Algorithm (outline)

- solve the corresponding Bottleneck AP, and let φ be the optimal solution; let

  $$lower := \min_i\{c_{i\varphi(i)}\},\ upper := \max_i\{c_{i\varphi(i)}\};$$

- define a bipartite graph $G[lower,upper]$ having edges $[i,j]$ if and only if $lower \leq c_{ij} \leq upper$;
- $G[lower,upper]$ does have a perfect matching (corresponding to the bottleneck AP solution);
- alternate between two phases:
  1. increase $lower$ to higher c_{ij} values **until** no perfect matching exists on $G[lower,upper]$;
  2. increase $upper$ to the smallest value of an uncovered element c_{ij}

     until $G[lower,upper]$ has a perfect matching
- terminating when $upper = \max\{c_{ij}\}$ and $G[lower,upper]$ does not have a perfect matching:
- the optimal solution is given by the minimum $(upper - lower)$ value encountered for which $G[lower,upper]$ has a perfect matching.
- $O(n^2)$ $(lower,upper)$ pairs are checked:

  using an $O(n^{2.5})$ algorithm for matching, **Time complexity** $O(n^{4.5})$;
- using augmenting paths to augment the current matching, **Time complexity** $O(n^4)$.

Exercise 5

Find a solution to the bottleneck assignment problem defined by the matrix in the next slide using the following implementation of Algorithm Threshold, and solving the matching subproblems through the Greedy Initialization Algorithm.

Algorithm Threshold:

$z^* := +\infty$; first threshold $c^* := \min\{c_{kl} : |\{c_{ij} : c_{ij} \leq c_{kl}\}| \geq n^2/2\}$ (median);

repeat

define a bipartite graph $G[c^*]$ having edges $[i, j]$ if and only if $c_{ij} \leq c^*$;

if $G[c^*]$ has a perfect matching **then** $z^* := \min(z^*, c^*)$, $c^* := \max\{c_{ij} : c_{ij} < c^*\}$

else $c^* := \{\min c_{ij} : c_{ij} > c^*\}$

until the current c^* was already used as the threshold in a previous iteration.

Solution:

1. Give the 16 sorted values of the matrix to find the first threshold:

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ .

2. At each iteration,

– give the value of c^* and underline the entries of C that produce $G[c^*]$;

– draw the edges of $G[c^*]$;

– highlight the edges found by the Greedy Initialization Algorithm and give the resulting value of z^*.

Iteration 1:

$c^* =$ ____ ;

$$C = \begin{pmatrix} 1 & 4 & 7 & 6 \\ 1 & 8 & 9 & 8 \\ 9 & 7 & 6 & 4 \\ 6 & 8 & 4 & 4 \end{pmatrix};$$

U	V
①	①
②	②
③	③
④	④

$z^* =$ ____ .

Iteration 2:

$c^* =$ ____ ;

$$C = \begin{pmatrix} 1 & 4 & 7 & 6 \\ 1 & 8 & 9 & 8 \\ 9 & 7 & 6 & 4 \\ 6 & 8 & 4 & 4 \end{pmatrix};$$

U	V
①	①
②	②
③	③
④	④

$z^* =$ ____ .

Iteration 3:

$c^* =$ ____ ;

$$C = \begin{pmatrix} 1 & 4 & 7 & 6 \\ 1 & 8 & 9 & 8 \\ 9 & 7 & 6 & 4 \\ 6 & 8 & 4 & 4 \end{pmatrix};$$

U	V
①	①
②	②
③	③
④	④

$z^* =$ ____ .

Exercise 6

Starting from the solution to the bottleneck assignment problem of Exercise 5, determine a solution to the balanced assignment problem on the same matrix (see the next slides).

Solve the matching subproblems through the Greedy Initialization Algorithm.

Implement the lower/upper bound increase as follows:

- alternate between two phases:
  1. increase $lower$ to the next (higher) c_{ij} value **until** no perfect matching is found on $G[lower, upper]$;
  2. increase $upper$ to the next (higher) c_{ij} **until** a perfect matching is found on $G[lower, upper]$;

At each iteration,

– give the values $lower$ and $upper$ and underline the entries of C that produce $G[lower, upper]$;

– draw the edges of $G[lower, upper]$;

– highlight the edges found by the Greedy Initialization Algorithm and give the current value of

$$z^* = \text{minimum } (upper - lower) \text{ value for which a perfect matching has been found.}$$

Terminate as soon as a solution of value $z^* = (upper - lower) = 2$ is found.

Iteration 1:

$lower = \_\_\_$ $upper = \_\_\_$;

$$C = \begin{pmatrix} 1 & 4 & 7 & 6 \\ 1 & 8 & 9 & 8 \\ 9 & 7 & 6 & 4 \\ 6 & 8 & 4 & 4 \end{pmatrix};$$

U	V
①	①
②	②
③	③
④	④

$z^* = \_\_\_\_$.

Iteration 2:

$lower = \_\_\_$ $upper = \_\_\_$;

$$C = \begin{pmatrix} 1 & 4 & 7 & 6 \\ 1 & 8 & 9 & 8 \\ 9 & 7 & 6 & 4 \\ 6 & 8 & 4 & 4 \end{pmatrix};$$

U	V
①	①
②	②
③	③
④	④

$z^* = \_\_\_\_$.

Iteration 3:

$lower = \_\_\_$ $upper = \_\_\_$;

$$C = \begin{pmatrix} 1 & 4 & 7 & 6 \\ 1 & 8 & 9 & 8 \\ 9 & 7 & 6 & 4 \\ 6 & 8 & 4 & 4 \end{pmatrix};$$

U	V
①	①
②	②
③	③
④	④

$z^* = \_\_\_\_$.

Iteration 4:

$lower = \_\_\_$ $upper = \_\_\_$;

$$C = \begin{pmatrix} 1 & 4 & 7 & 6 \\ 1 & 8 & 9 & 8 \\ 9 & 7 & 6 & 4 \\ 6 & 8 & 4 & 4 \end{pmatrix};$$

U	V
①	①
②	②
③	③
④	④

$z^* = \_\_\_\_$.

Iteration 5:

$lower = \_\_\_$ $upper = \_\_\_$;

$$C = \begin{pmatrix} 1 & 4 & 7 & 6 \\ 1 & 8 & 9 & 8 \\ 9 & 7 & 6 & 4 \\ 6 & 8 & 4 & 4 \end{pmatrix};$$

U	V
①	①
②	②
③	③
④	④

$z^* = \_\_\_\_$.

Iteration 6:

$lower = \_\_\_$ $upper = \_\_\_$;

$$C = \begin{pmatrix} 1 & 4 & 7 & 6 \\ 1 & 8 & 9 & 8 \\ 9 & 7 & 6 & 4 \\ 6 & 8 & 4 & 4 \end{pmatrix};$$

U	V
①	①
②	②
③	③
④	④

$z^* = \_\_\_\_$.

8. Quadratic assignment problem

Quadratic Assignment Problem (QAP)

- Koopmans and Beckmann (1957) for the location of indivisible economical activities:

  Assign n facilities to n locations, with the cost being proportional to

  the flow between the facilities multiplied by the distances between the locations.

- Two $n \times n$ matrices:
  - $A = (a_{ik})$: a_{ik} = flow from facility i to facility k;
  - $B = (b_{jl})$: b_{jl} = distance from location j to location l;
  - $\varphi(i)$ = location assigned to facility i;
  - $QAP(A, B)$: find a permutation φ of $\{1, 2, \ldots, n\}$ (to be applied to the rows and column of B) that minimizes

    $$\sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\varphi(i)\varphi(k)}.$$

    i.e., such that the **inner product** of A and B (permuted) is a minimum.
- More complete (but equivalent) formulation:
  - $C = (c_{ij})$: c_{ij} = cost of placing facility i at location j;
  - $QAP(A, B, C)$: $\min \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\varphi(i)\varphi(k)} + \sum_{i=1}^{n} c_{i\varphi(i)}$.
  - $QAP(A, B, C)$ can be transformed into an equivalent $QAP(A, B)$

Meaning of the objective function

$$\sum_{i=1}^{n}\sum_{k=1}^{n} a_{ik}b_{\varphi(i)\varphi(k)} + \sum_{i=1}^{n} c_{i\varphi(i)} = \sum_{i=1}^{n}\left(c_{i\varphi(i)} + \sum_{k=1}^{n} a_{ik}b_{\varphi(i)\varphi(k)}\right)$$

- Each individual product

$$a_{ik}b_{\varphi(i)\varphi(k)}$$

  is the transportation cost caused by assigning
  facility i to location $\varphi(i)$ and facility k to location $\varphi(k)$;

- Hence each term $c_{i\varphi(i)} + \sum_{k=1}^{n} a_{ik}b_{\varphi(i)\varphi(k)}$ is the total cost given, for facility i, by
  - cost for installing it at location $\varphi(i)$, plus
  - transportation costs to all facilities k, if installed at locations $\varphi(1), \varphi(2), \ldots, \varphi(n)$.

Numerical example

- Consider a $QAP(A, B, C)$ with $n = 3$ and input matrices

$$A = \begin{pmatrix} 1 & 2 & 4 \\ 3 & 4 & 5 \\ 5 & 6 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 3 & 6 \\ 1 & 4 & 7 \\ 5 & 6 & 2 \end{pmatrix} \quad \text{and} \quad C = \begin{pmatrix} 9 & 7 & 9 \\ 6 & 5 & 7 \\ 8 & 9 & 8 \end{pmatrix}.$$

- Given a permutation, say $\varphi = (2, 1, 3)$, we compute the objective function value by

(i) first permuting the rows of B according to φ : $\tilde{B} = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 3 & 6 \\ 5 & 6 & 2 \end{pmatrix}$

then the columns according to φ,

(ii) then the columns according to φ : $B_\varphi = (b_{\varphi(i)\varphi(k)}) = \begin{pmatrix} 4 & 1 & 7 \\ 3 & 2 & 6 \\ 6 & 5 & 2 \end{pmatrix}$,

- and deriving from $\sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\varphi(i)\varphi(k)} + \sum_{i=1}^{n} c_{i\varphi(i)}$,

$$z = (4 + 2 + 28) + (9 + 8 + 30) + (30 + 30 + 2) + (7 + 6 + 8) = 164.$$

The QAP has very many real life applications

- Location theory: campus planning, backboard wiring, forest parks.
- Ergonomics: typewriter keyboard design, eye fatigue minimization in control boards.
- Turbine balancing: turbine runner problem.
- Ranking problems: ranking of archeological data, ranking of a team in a relay race.
- Scheduling of parallel production lines, analysis of chemical reactions, combinatorial data analysis, ...
- Very important special cases, e.g., Traveling Salesman Problem.

Complexity of the QAP

- The QAP is Strongly $\mathcal{NP}$-hard
- Even finding an approximate solution within some constant factor from the optimum value cannot be done in polynomial time, unless $\mathcal{P} = \mathcal{NP}$ (Sahni and Gonzalez, 1976).
- Even if the coefficient matrices fulfil the triangle inequality (Queyranne, 1986).
- Vary many techniques explored for its solution.

Formulations: Integer Quadratic Program

The $QAP(A, B, C)$,

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\varphi(i)\varphi(k)} + \sum_{i=1}^{n} c_{i\varphi(i)}$$

can be formulated as an **Integer Quadratic Program** by defining

$$x_{ij} = \begin{cases} 1 & \text{if facility } i \text{ is assigned to location } j; \\ 0 & \text{otherwise.} \end{cases}$$

Then:

$$\min \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} a_{ik} b_{jl} x_{ij} x_{kl} + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad (j = 1, 2, \ldots, n),$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1, 2, \ldots, n),$$

$$x_{ij} \in \{0, 1\} \qquad (i, j = 1, 2, \ldots, n).$$

Formulations: Inner Product formulation

- A permutation φ can be represented by a permutation matrix $X_\varphi = (x_{ij})$ with

$$x_{ij} = \begin{cases} 1 & \text{if } j = \varphi(i); \\ 0 & \text{otherwise.} \end{cases}$$

- Given an $n \times n$ matrix B and a permutation φ of $\{1, 2, \ldots, n\}$, with permutation matrix X_φ, we have

$$X_\varphi B X'_\varphi = (b_{\varphi(i)\varphi(k)}) \tag{14}$$

(the first product permutes the rows, the second product permutes the columns)

- **Inner product** of two $n \times n$ matrices A and B:

$$\langle A, B \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{ij}.$$

- $\mathbf{X}_n$ = set of all $n \times n$ permutation matrices
  = set of all $n \times n$ matrices produced by $\sum_{i=1}^{n} x_{ij} = 1 \ \forall j$, $\sum_{j=1}^{n} x_{ij} = 1 \ \forall i$.
- QAP(A,B,C):

$$\begin{aligned} \text{min} \quad & \langle A, XBX^T \rangle + \langle C, X \rangle \\ \text{s.t.} \quad & X \in \mathbf{X}_n. \end{aligned}$$

Numerical example

- $QAP(A, B, C)$ with $n = 3$, input matrices

$$A = \begin{pmatrix} 1 & 2 & 4 \\ 3 & 4 & 5 \\ 5 & 6 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 3 & 6 \\ 1 & 4 & 7 \\ 5 & 6 & 2 \end{pmatrix} \quad \text{and} \quad C = \begin{pmatrix} 9 & 7 & 9 \\ 6 & 5 & 7 \\ 8 & 9 & 8 \end{pmatrix}.$$

  - Permutation matrix corresponding to $\varphi = (2, 1, 3)$: $X_\varphi = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$,

  - $X_\varphi B X'_\varphi = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 6 \\ 1 & 4 & 7 \\ 5 & 6 & 2 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 1 & 7 \\ 3 & 2 & 6 \\ 6 & 5 & 2 \end{pmatrix}$.

  - $z = \langle A, X_\varphi B X_\varphi^T \rangle + \langle C, X_\varphi \rangle$:

$$z = \left\langle \begin{pmatrix} 1 & 2 & 4 \\ 3 & 4 & 5 \\ 5 & 6 & 1 \end{pmatrix}, \begin{pmatrix} 4 & 1 & 7 \\ 3 & 2 & 6 \\ 6 & 5 & 2 \end{pmatrix} \right\rangle + \left\langle \begin{pmatrix} 9 & 7 & 9 \\ 6 & 5 & 7 \\ 8 & 9 & 8 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right\rangle$$

$$= (4 + 2 + 28 + 9 + 8 + 30 + 30 + 30 + 2) + (7 + 6 + 8) = 164.$$

Formulations: Trace formulation

- The **trace** of an $n \times n$ matrix $A = (a_{ik})$ is the sum of its diagonal elements:

$$\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii}.$$

- Some simple properties: $\mathrm{tr}(A) = \mathrm{tr}(A^T)$ (A^T = transpose of A);

$$\mathrm{tr}(A + B) = \mathrm{tr}(A) + \mathrm{tr}(B);$$

$$\mathrm{tr}(AB) = \mathrm{tr}(A^T B^T).$$

- It is easily seen that the inner product $\sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{ik}$ can be written as the trace of the product AB^T of the two matrices $A = (a_{ik})$ and $B = (b_{ik})$.
- Moreover, the matrix $(b_{\varphi(i)\varphi(k)})$ can be written as $X_\varphi B X_\varphi^T$ (see (14)).
  Since $\mathrm{tr}(CX^T) = \sum_{i=1}^{n} c_{i\varphi(i)}$, it can be shown that the QAP can be formulated as

$$\begin{array}{ll} \min & \mathrm{tr}((AXB^T + C)X^T) \\ \text{s.t.} & X \in \mathbf{X}_n. \end{array}$$

  The trace formulation of the QAP has been used to obtain eigenvalue bounds for QAPs.

Numerical example

- $QAP(A, B, C)$ with $n = 3$, input matrices

$$A = \begin{pmatrix} 1 & 2 & 4 \\ 3 & 4 & 5 \\ 5 & 6 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 3 & 6 \\ 1 & 4 & 7 \\ 5 & 6 & 2 \end{pmatrix} \quad \text{and} \quad C = \begin{pmatrix} 9 & 7 & 9 \\ 6 & 5 & 7 \\ 8 & 9 & 8 \end{pmatrix}.$$

  - Permutation $\varphi = (2, 1, 3)$: $X_\varphi = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$:

- In order to define the trace formulation we need to compute

$$AX_\varphi B^T = \begin{pmatrix} 1 & 2 & 4 \\ 3 & 4 & 5 \\ 5 & 6 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 5 \\ 3 & 4 & 6 \\ 6 & 7 & 2 \end{pmatrix} = \begin{pmatrix} 31 & 34 & 24 \\ 47 & 51 & 48 \\ 33 & 33 & 62 \end{pmatrix}, \Longrightarrow$$

$$(AX_\varphi B^T + C)X_\varphi^T = \left(\begin{pmatrix} 31 & 34 & 24 \\ 47 & 51 & 48 \\ 33 & 33 & 62 \end{pmatrix} + \begin{pmatrix} 9 & 7 & 9 \\ 6 & 5 & 7 \\ 8 & 9 & 8 \end{pmatrix} \right) \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 41 & 40 & 33 \\ 56 & 53 & 55 \\ 42 & 41 & 70 \end{pmatrix}. \Longrightarrow \text{Trace: } z = 41 + 53 + 70 = 164.$$

Exact solution techniques

- Other equivalent mathematical formulations:
  - Kronecker product formulation (Lawler , 1963);
  - Convex and concave integer programs
- Linearizations: the quadratic objective function is linearized by introducing new variables, e.g.,

$$y_{ijkl} = x_{ij}x_{kl}(i, j, k, l = 1, 2, \ldots, n)$$

and additional constraints. Different linearization techniques:
  - Lawler (1963)
  - Kaufman and Broeckx (1978)
  - Balas and Mazzola (1984)
  - Frieze and Yadegar (1983)
  - Adams and Johnson (1994)
  - Adams, Guignard, Hahn and Hightower (2007)

Exact solution techniques

- Lower bounds and reductions:
  - Reduction: move costs to the linear term, decreasing the contribution of the quadratic term.
  - Gilmore-Lawler (1962-1963): $O(n^3)$ time bound (and reduction) by solving an associated AP; probably still the best ratio (quality)/(computational effort).
  - Conrad (1971)
  - Burkard (1973)
  - Roucairol (1979)
  - Edwards (1980)
  - Frieze (1983)
  - Finke, Burkard and Rendl (1987), eigenvalues
  - Hadley, Rendl and Wolkowicz (1992)
  - Chakrapani and Skorin-Kapov (1994)
  - Li, Pardalos, Ramakrishnan and Resende (1994)
  - Hahn and Grant (1998)
  - Anstreicher and Brixius (2001), convex quadratic programming
  - Zhao, Karisch, Rendl and Wolkowicz (1998), Rendl and Sotirov (2007), semidefinite programming

Exact solution techniques

- Algorithms:
  - Benders' decomposition
  - Branch-and-bound, with different branching techniques:
    - *single assignment*
    - *pair assignment*
    - *relative positioning*
    - *polytomic*
    - *polytomic* with *row branching* and *column branching*
  - Branch-and-cut
  - Parallel algorithms
  - Grid computing (massively parallel algorithms)

Heuristics

- Traditional heuristics
  - Greedy algorithms
  - Truncated exact algorithms
- Meta-heuristics
  - Simulated annealing
  - Tabu search (robust TS, reactive TS, parallel reactive TS, ...)
  - Genetic algorithms
  - Hybrid algorithms
  - Greedy Randomized Adaptive Search Procedure (GRASP)
  - Ant colony optimization (hybrid ACO, fast ACO, ...)
  - Scatter search
  - Path relinking
  - Large scale and variable neighborhood search
  - ...

Exact solution: Computational experiments

- Classical benchmarks: *Nugent instances* (Nugent et al. 1968): **Nugxx** ($\leftrightarrow n = \mathrm{xx}$).
- Up to 1994: largest solved instance **Nug16** (sequential computers).
- Starting from 1995: parallel computers (times normalized to single CPU):

- 1995: **Nug20**, 14 CPU hours (Clausen and Perregaard);
- 1997: **Nug21**, 12 CPU days (Clausen and Perregaard);
- 1999: **Nug25**, 30 CPU days (Marzetta and Brüngger);

- 2002: Anstreicher, Brixius, Goux and Linderoth, federation of over 2500 CPUs:
  - **Nug27**, 65 CPU days;
  - **Nug28**, 10 CPU months;
  - **Nug30**, 7 CPU years;

- 2007: Adams, Guignard, Hahn and Hightower:
  - **Nug27**, 20 CPU days;
  - **Nug28**, 5 CPU months;
  - **Nug30**, 2.5 CPU years.

- Other difficult instances ('esc' instances) solved by Fischetti, Monaci and Salvagnin in 2012.

To study these issues in more depth

History of the Hungarian Algorithm:

- T. Gallai (1986). Dénes Kőnig: A biographical sketch. In *Theory of Finite and Infinite Graphs*, Birkhäuser, Boston (English translation by R. McCoart).
- T. Rapcsák (2010). The life and works of Jenő Egerváry. *Central European Journal of Operations Research* 18, 59-71.
- S. Martello (2010). Jenő Egerváry: from the origins of the Hungarian algorithm to satellite communication. *Central European Journal of Operations Research* 18, 47-58.
- H.W. Kuhn (2012). A tale of three eras: The discovery and rediscovery of the Hungarian method. *European Journal of Operational Research*.

Excerpts (**free download**) from **Assignment Problems** `http://www.assignmentproblems.com`

- **Table of contents:** `http://www.assignmentproblems.com/doc/Contents.pdf`
- **Linear Assignment Problem:**
  `http://www.assignmentproblems.com/doc/LSAPIntroduction.pdf`
- **Bottleneck Assignment Problem:**
  `http://www.assignmentproblems.com/doc/LBAPIntroduction.pdf`
- **Quadratic Assignment Problem:**
  `http://www.assignmentproblems.com/doc/QAPIntroduction.pdf`

Freeware

Usually copyrighted for professional use, but free for research and teaching purposes.

Linear Assignment Problem

- `http://www.assignmentproblems.com/linearAPfreeware.htm`:
  direct downloads and links: Pascal, Fortran and C codes.
- `http://www.assignmentproblems.com/linearAPdidactic.htm`:
  Java applets for the Hungarian algorithm ($O(n^4)$ and $O(n^3)$ implementations).

Bottleneck Assignment Problem

- `http://www.assignmentproblems.com/bottleneckAPfreeware.htm`:
  Pascal codes (direct download).
- `http://www.assignmentproblems.com/linearBNAPdidactic.htm`:
  Java applet for the Threshold Algorithm

Quadratic Assignment Problem

- `http://www.seas.upenn.edu/qaplib/codes.html`:
  links to exact and approximation algorithms;
  various languages: Pascal, Fortran, C.

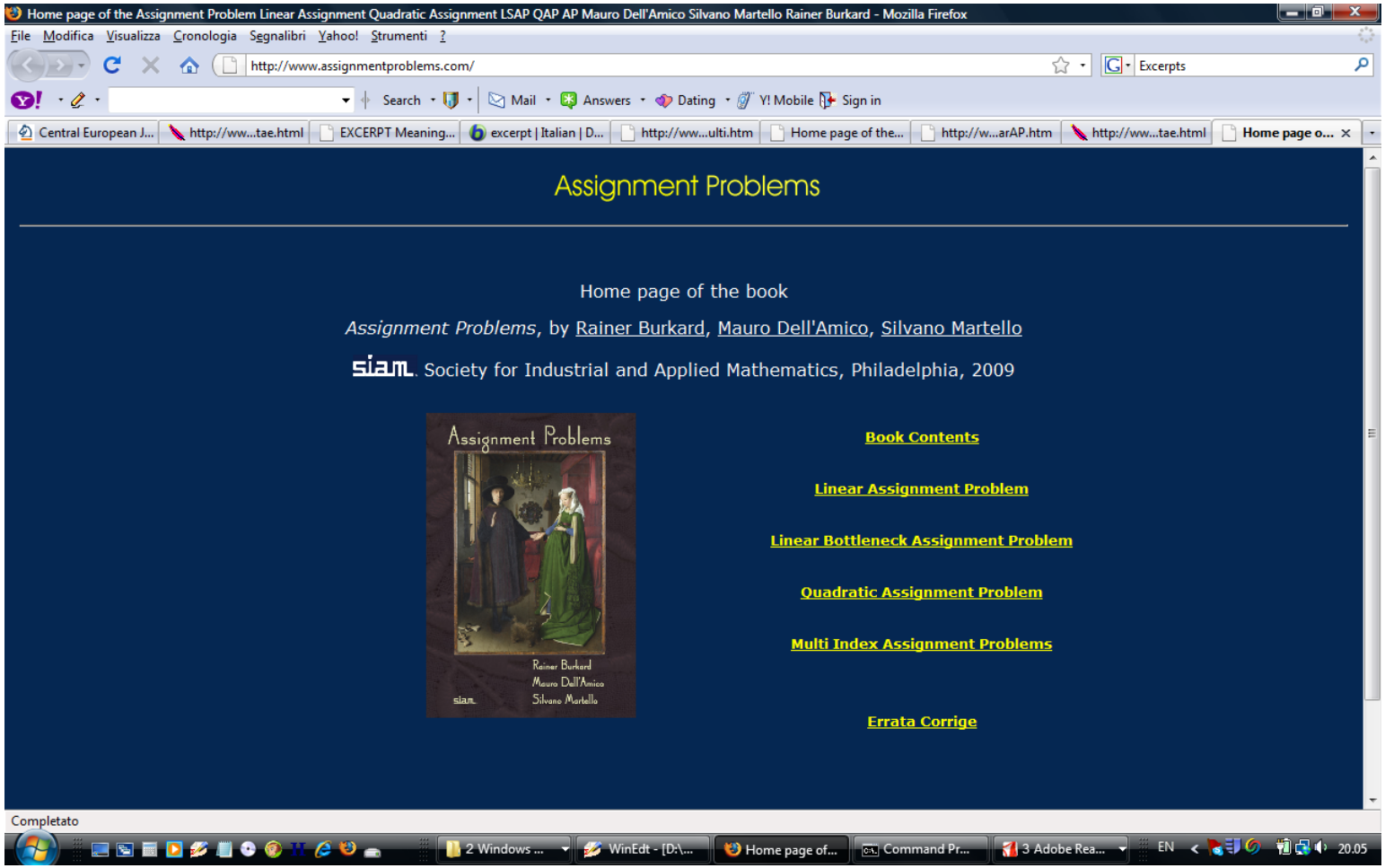

Figure 8: Why this painting? Because of the marriage theorem, of course, but for another reason too. It is known that the dual can reveal something that is "hidden" in the primal, and this painting is famous for its Primal-Dual content. Why?

Figure 9: Jan van Eyck, The Arnolfini portrait: this is the Primal. What about the dual? Look at the mirror on the wall. What will it show? The Arnolfini's back, and **...**

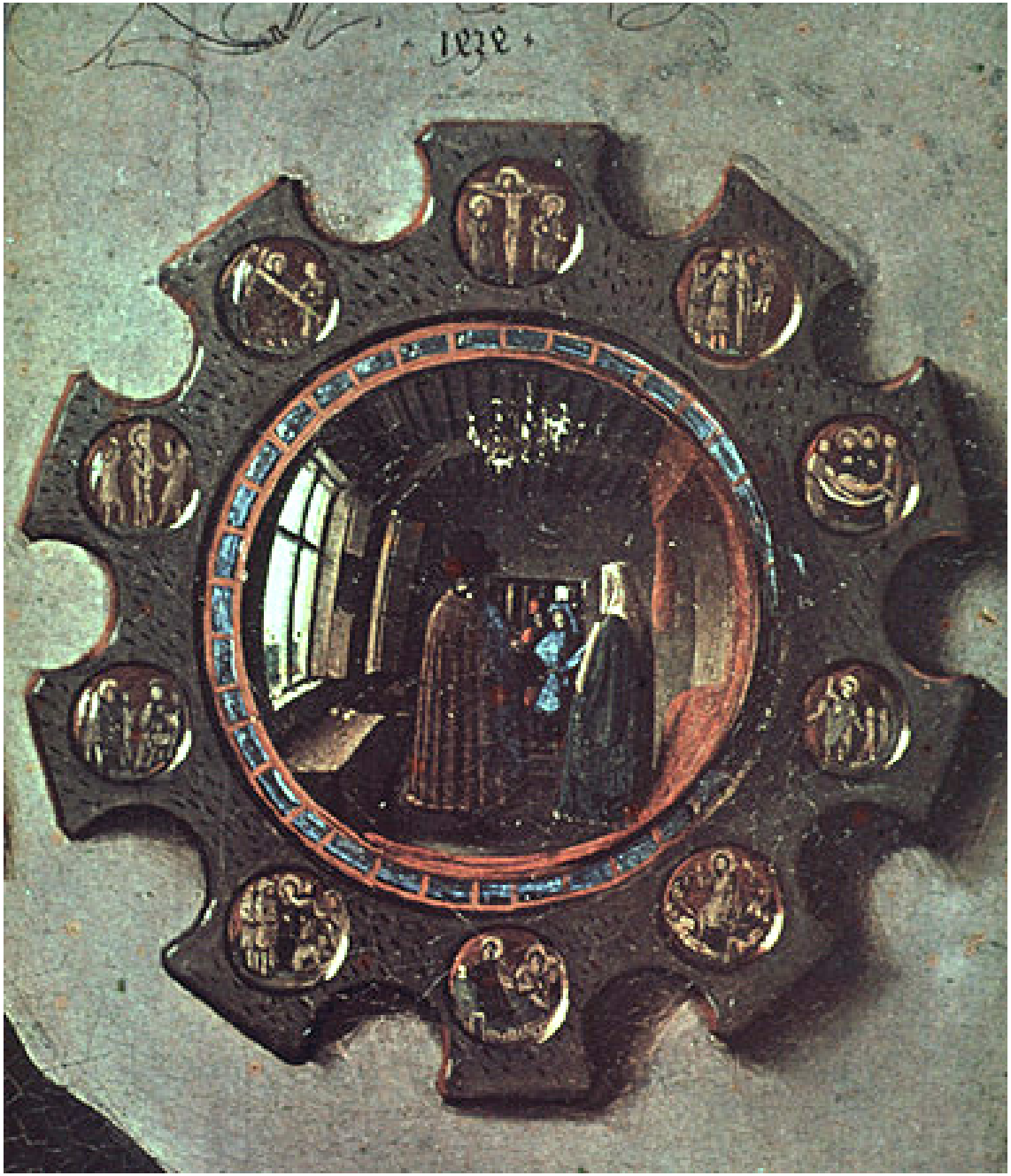

Figure 10: ... the painter and his assistant at work (the Dual).

Thank you for your attention!