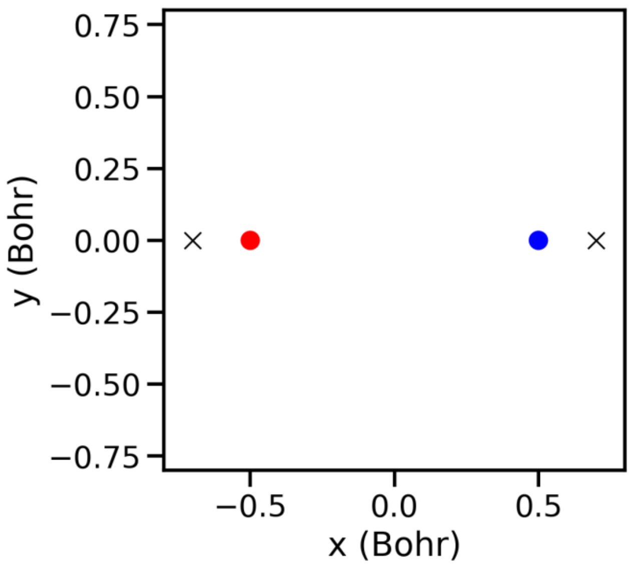
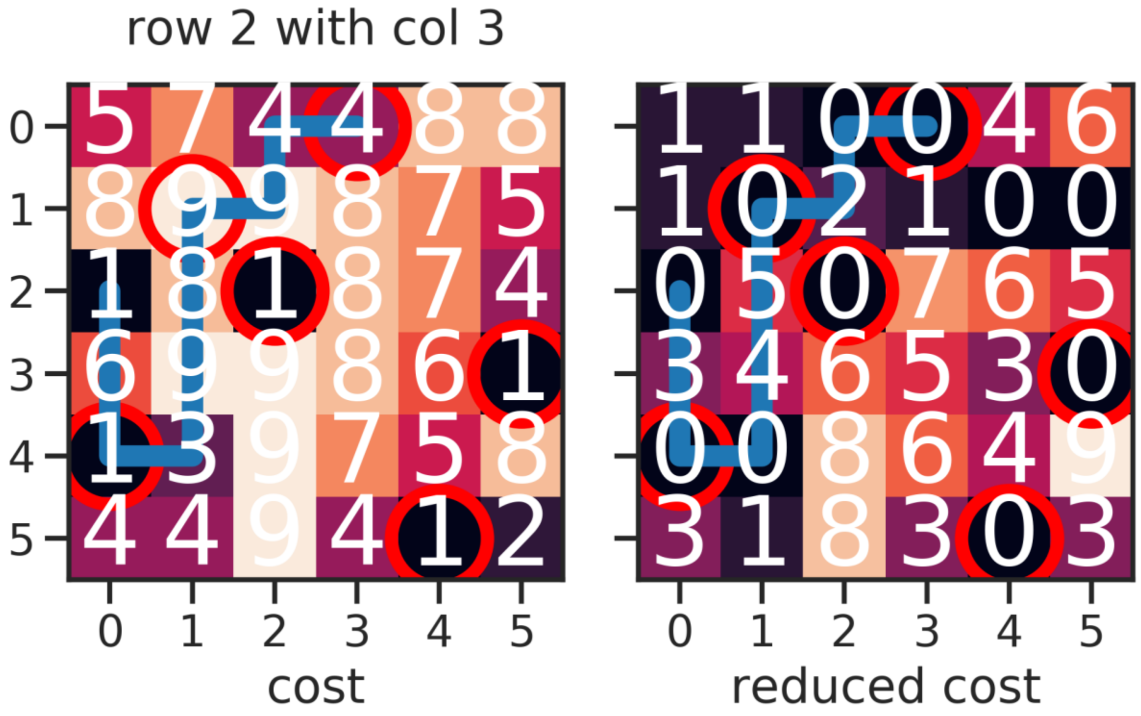
Polynomial-Scaling Algorithm for the Linear Sum Assignment Problem

Yubo “Paul” Yang, 2020-10-18, Algorithm Interest Group (http://algorithm-interest-group.me)





What is the (balanced) linear sum assignment problem (LSAP)?

Goal find minimum-cost assignment of n *"agents"* to n *"tasks"*.

Problem defined by a *cost matrix.* c_{ij} is the cost to assign agent i to task j.

Mathematically a linear programming (LP) problem:

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

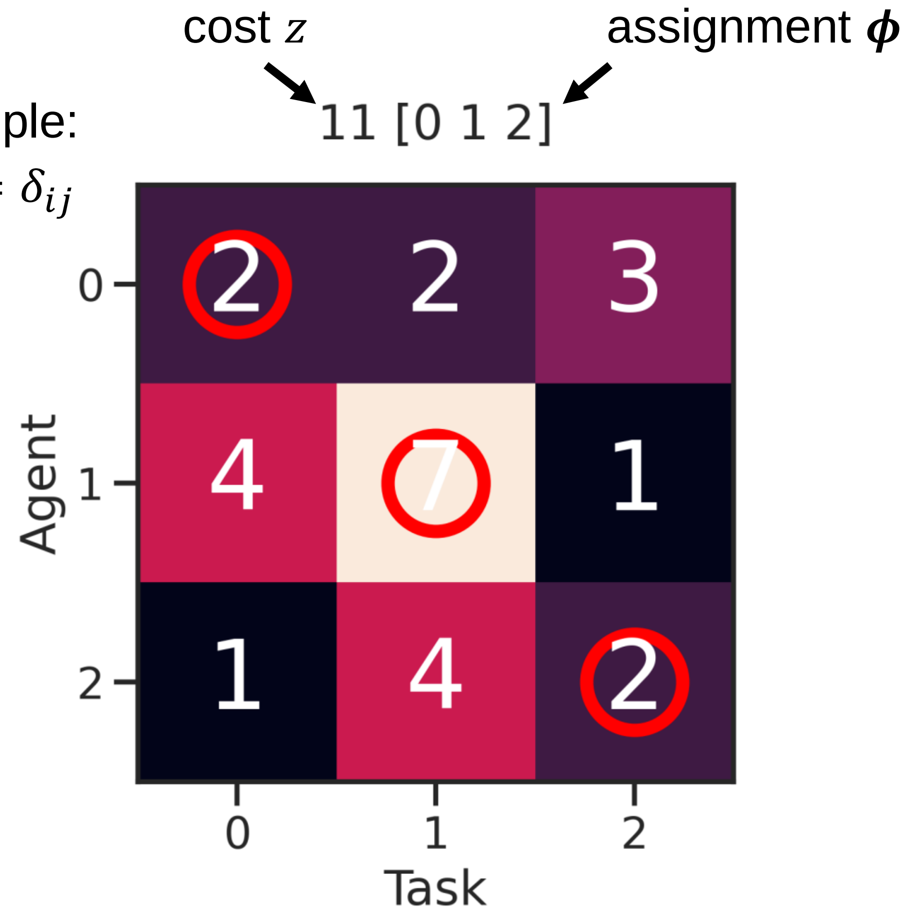
with constraints $$\sum_{i=1}^{n} x_{ij} = 1$$

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

$$x_{ij} \geq 0, \forall i, j$$

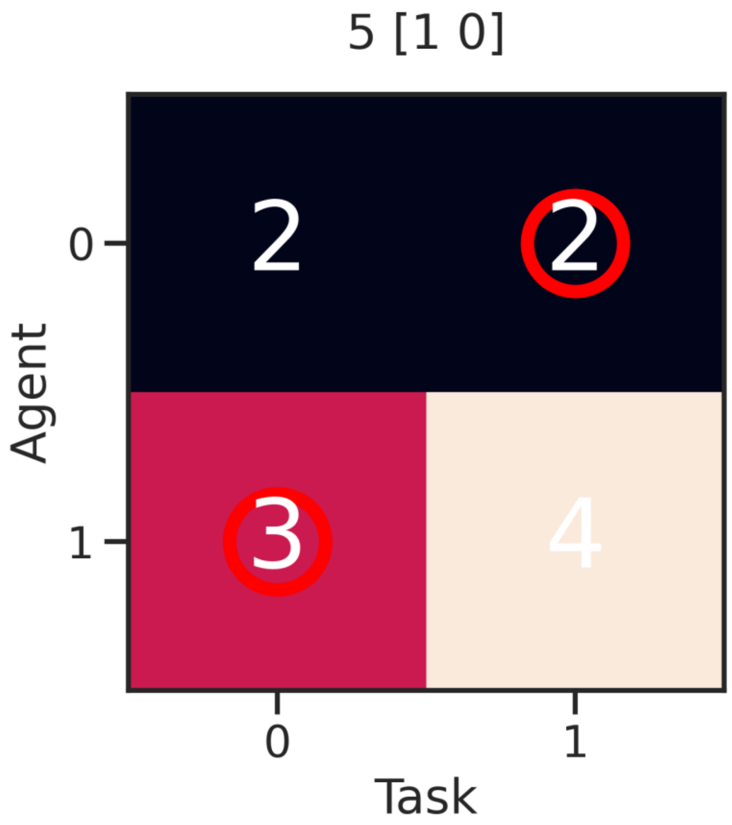
Example:

$x_{ij} = \delta_{ij}$



What is the (balanced) linear sum assignment problem (LSAP)?

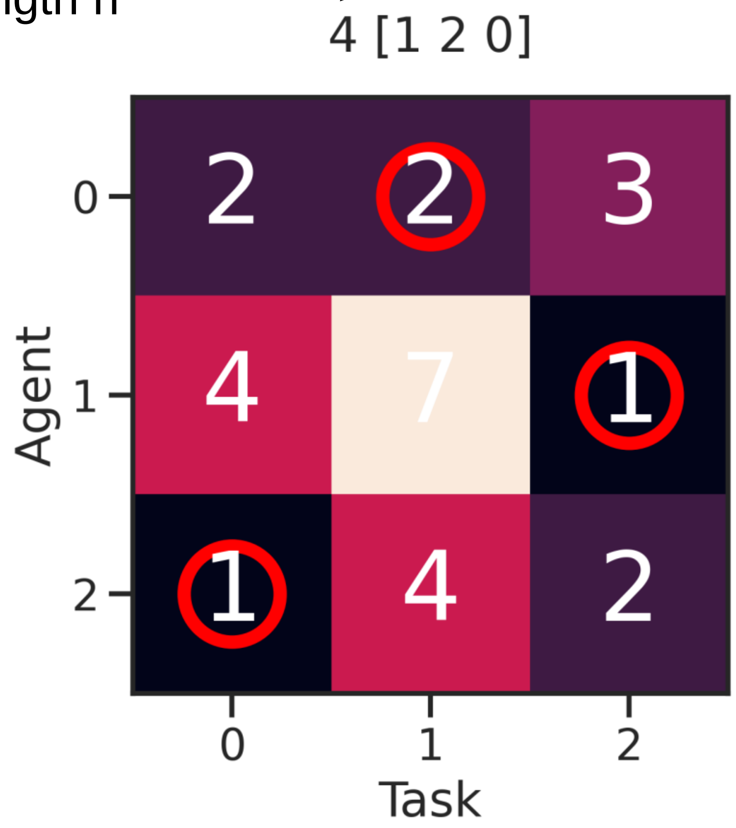
Example 2x2



assignment ϕ

is a *permutation* of length n
$\Rightarrow n!$ possibilities!

Example 3x3



Why solve LSAP? Many practical (real-world!) applications

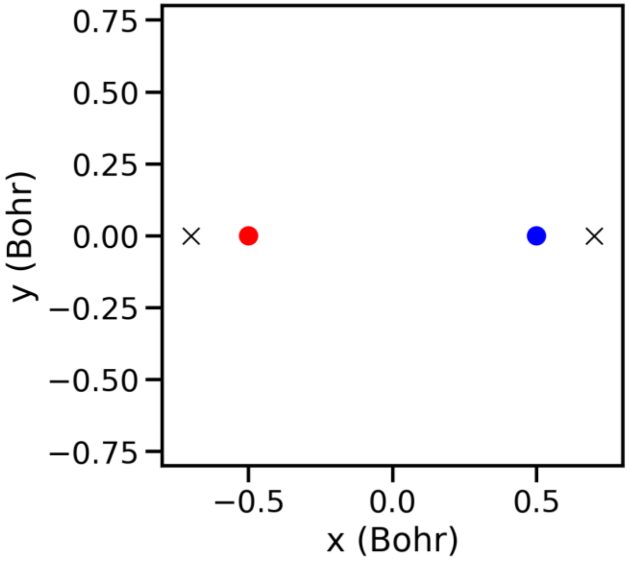
Choose 4/n swimmers for relay team (back, breast, butterfly, free style) [1]

Vehicle routing problems (eg. Taxi, flights) [2]

Reconstruct mass distribution in the early universe [3]

Shoot down Soviet nukes [4]

Quantum exchange [5]



[1] R.E. Machol, “An application of the assignment problem,” Oper. Res. **18**, 745 (1970)
[2] M. Fischetti, A. Lodi, S. Martello, and P. Toth, “A polyhedral approach to simplified crew scheduling and vehicle scheduling problems,” Management Sci. **47**, 833 (2001).
[3] U. Frisch and A. Sobolevskii, “Application of optimal transport theory to reconstruction of the early universe,” J. Math. Sci. **133**, 1539 (2006).
[4] B.L. Schwartz, “A computational analysis of the auction algorithm,” Euro. J. Oper. Res. **74**, 161 (1994).
[5] D.M. Ceperley, G. Jacucci, “Calculation of Exchange Frequencies in bcc 3He with the Path-Integral Monte Carlo Method,” Phys. Rev. Lett. 58, 1648 (1987).

How to solve LSAP?

Short answer: read a book [1]

I will walk through the internals of the "Hungarian algorithm"

Dénes Kőnig

Jenő Egerváry

[1] Rainer Burkard, Mauro Dell'Amico, Silvano Martello, "Assignment Problems : Revised Reprint," SIAM (2009).

[2] T. Bonniger, G. Katzakidis, R.E. Burkard, U. Derigs, "Solution Methods with FORTRAN-Programs", Springer-Verlag Berlin (1980).

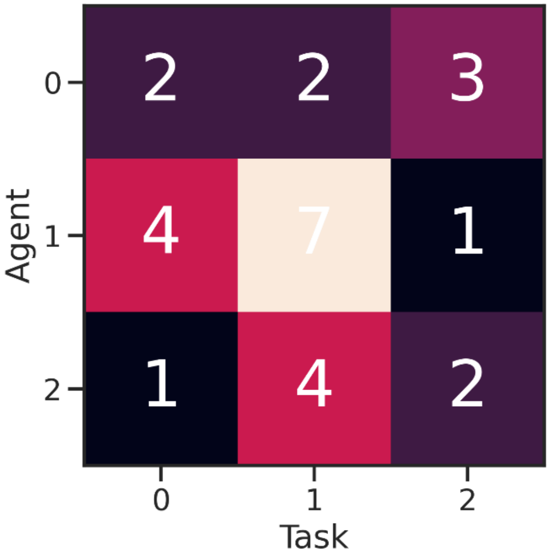
Table 4.2. *Evolution of algorithms for LSAP.*

Year	Reference	Time complexity	Category
1946	Easterfield [255]	exponential	Combinatorial
1955	Kuhn [457]	$O(n^4)$	Primal-dual
1957	Munkres [524]	$O(n^4)$	Primal-dual
1964	Balinski and Gomory [65]	$O(n^4)$	Primal
1969	Dinic and Kronrod [243]	$O(n^3)$	Dual
1971	Tomizawa [670]	$O(n^3)$	Shortest path
1972	Edmonds and Karp [260]	$O(n^3)$	Shortest path
1976	Lawler [467]	$O(n^3)$	Primal-dual
1976	Cunningham [211]	exponential	Primal simplex
1977	Barr, Glover, and Klingman [69]	exponential	Primal simplex
1978	Cunningham and Marsh [214]	$O(n^3)$	Primal
1980	Hung and Rom [395]	$O(n^3)$	Dual
1981	Bertsekas [88]	$O(n^3 + n^2\mathcal{C})$	Auction
1985	Balinski [63], Goldfarb [346]	$O(n^3)$	Dual simplex
1985	Gabow [307]	$O(n^{3/4} m \log \mathcal{C})$	Cost scaling
1988	Bertsekas and Eckstein [94]	$O(nm \log(n\mathcal{C}))$	Auction + ε-rel.
1989	Gabow and Tarjan [309]	$O(\sqrt{n}\, m \log(n\mathcal{C}))$	Cost scaling
1993	Orlin and Ahuja [538]	$O(\sqrt{n}\, m \log(n\mathcal{C}))$	Auction + ε-rel.
1993	Akgül [20]	$O(n^3)$	Primal simplex
1995	Goldberg and Kennedy [340]	$O(\sqrt{n}\, m \log(n\mathcal{C}))$	Pseudoflow
2001	Kao, Lam, Sung, and Ting [419]	$O(\sqrt{n}\, \mathcal{W} \log(\frac{n^2}{\mathcal{W}/\mathcal{C}}) / \log n)$	Decomposition

Primal-dual formulation of the LSAP

Cost Matrix

c_{ij}



set dual variable ⇔

set primal constraint

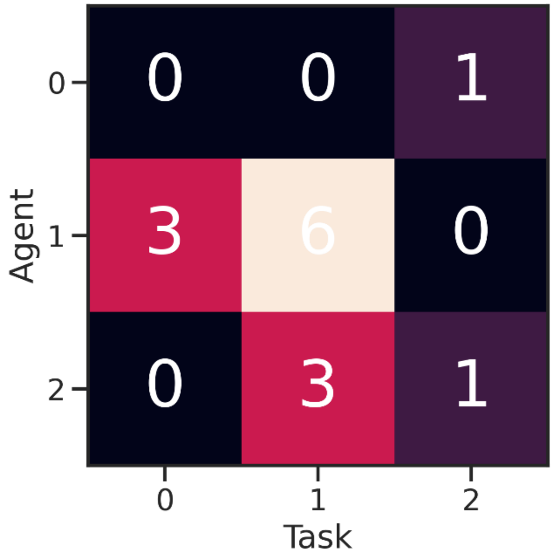
$\boldsymbol{u} = (2, 1, 1) \qquad \boldsymbol{v} = (0, 0, 0)$

admissibility transform

$$\bar{c}_{ij} = c_{ij} - u_i - v_j$$

Reduced Cost Matrix

$\bar{c}_{ij}$



Primal problem

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

with constraints $$\sum_{j=1}^{n} x_{ij} = 1$$

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

variables become constraints

constraints become variables

Dual problem

Maximize $$\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$$

with constraints $$c_{ij} - u_i - v_j \geq 0, \qquad \forall i, j$$

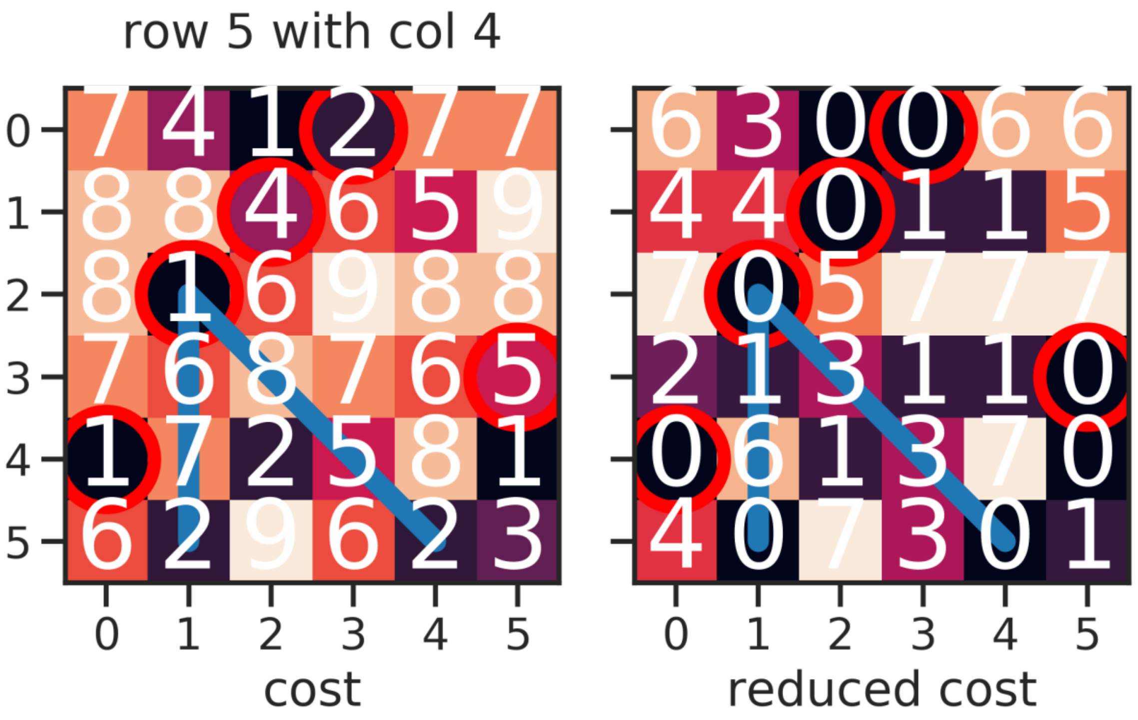
Take home: each 0 in $\bar{c}$ is *admissible* in an optimal primal assignment.

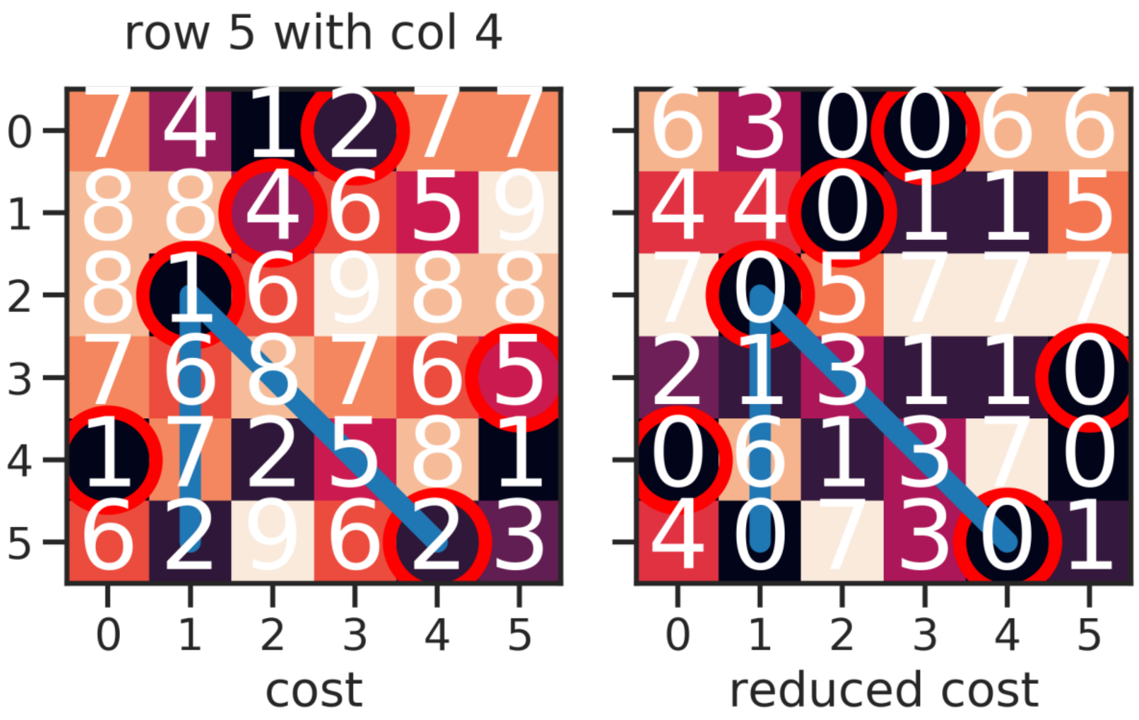
The Hungarian algorithm : simultaneous primal and dual solutions

Big idea: By duality theorem, an optimal solution to both primal and dual problems is THE optimal solution.

Algorithm:

1. Initialize ultra optimal dual variables. Only a partial primal assignment is *admissible*.
2. Search *admissible solutions* for the primal assignment with largest cardinality.
3. If all tasks assigned, then done.
4. Otherwise relax constraints, i.e. decrease dual cost function by *minimum reduced cost*.





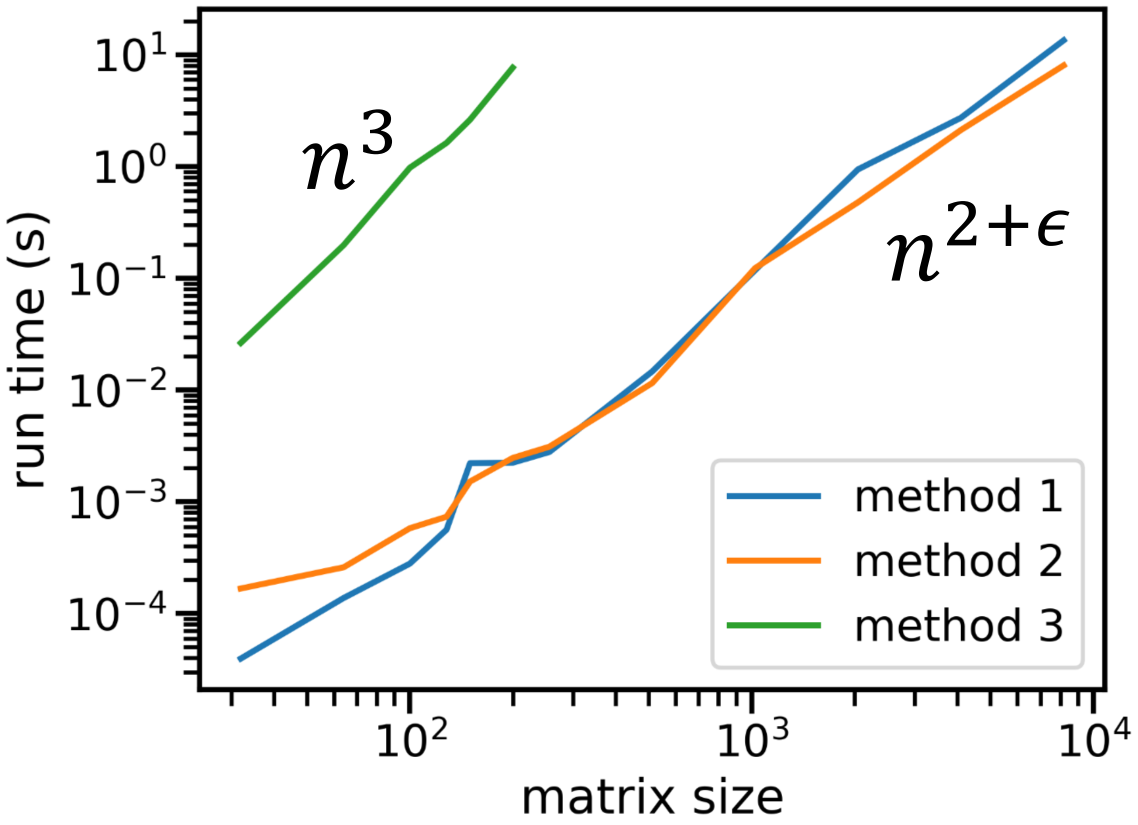
Timings

Three implementations:

- Yang (Python)
- Ceperley (FORTRAN)
- Scipy

Which one is which?

- Method 1 = Ceperley (FORTRAN)
- Method 2 = SciPy
- Method 3 = Yang (Python)



Implementation Breakdown (Python)

```python
def kuhn_hungarian(cmat):
  u, v = dual_variables(cmat)
  cbar = admissible_transform(cmat, u, v)
  # initialize partial solution
  phi = partial_primal_solution(cmat)
  row = transpose_map(phi)
  # main loop
  uset = set(np.where(phi>=0)[0])
  while len(uset) < n:
    k = (full_set(n)-uset).pop()
    # find alternating path
    sink, jpred, su, lv, sv = alternate(k, row, cbar)
    if sink >= 0:  # enlarge primal solution
      uset.add(k)
      j = sink
      i = -1
      while i != k:
        i = jpred[j]
        row[j] = i
        phi[i], j = j, phi[i]
    else:  # update the dual solution
      delta = minimum_reduced_cost(cbar, su, lv)
      for i in su:
        u[i] += delta
      for j in lv:
        v[j] -= delta
      cbar = admissible_transform(cmat, u, v)
    uset = set(np.where(phi>=0)[0])
  return phi
```

```python
def dual_variables(cmat):
  u = np.min(cmat, axis=1)
  v = np.min(cmat-u[:, None], axis=0)
  return u, v
```

```python
def admissible_transform(cmat, u, v)
  m, n = cmat.shape
  assert m == n
  cbar = np.array([
    [cmat[i, j]-u[i]-v[j] for j in range(n)]
  for i in range(n)])
  return cbar
```

```python
def alternate(k, row, cbar):
  su = lv = sv = set()
  jpred = -np.ones(n, dtype=int)
  while (not fail) and (sink < 0):
    su.add(i)
    for j in vset-lv:  # look for admissible column at row i
      if (cbar[i, j] == 0):  # admissible
        lv.add(j)
        jpred[j] = i
    # scan lv by selecting one column at a time
    if len(lv-sv) > 0:  # scan next column in lv
      j = (lv-sv).pop()
      sv.add(j)
      if row[j] < 0:
        sink = j
      else:
        i = row[j]
    else:  # finished scan without finding sink
      fail = True
    iwhile += 1
  return sink, jpred, su, lv, sv
```

```python
def minimum_reduced_cost(cbar
  m, n = cbar.shape
  assert m == n
  vset = full_set(n)
  delta = np.inf
  for i in su:
    for j in vset-lv:
      if cbar[i, j] < delta:
        delta = cbar[i, j]
  return delta
```

[1] Rainer Burkard, Mauro Dell'Amico, Silvano Martello, "Assignment Problems: Revised Reprint," SIAM (2009).

Implementation Breakdown (FORTRAN)

```
      do 110 i=1,n
      if(la(i)) go to 110
      if(dm(i)+eps.ge.d) go to 110
      d=dm(i)
      k=i
110   continue
      if(ize(k).le.0) go to 400
      la(k)=.true.
      iw=ize(k)
      iws=(iw-1)*n
      dp(iw)=d
      do 130 i=1,n
      if(la(i)) go to 130
      vgl=d+c(i,iw)-ys(iw)-yt(i)
      if(dm(i).le.vgl+eps) go to 130
      dm(i)=vgl
      ivor(i)=iw
130   continue
      go to 105
c augmentation
400   iw=ivor(k)
      ize(k)=iw
      ind=iperm(iw)
      iperm(iw)=k
      if(iw.eq.iu) go to 500
      k=ind
      go to 400
c transformation
500   do 510 i=1,n
      if(dp(i).eq.sup) go to 505
      ys(i)=ys(i)+d-dp(i)
505   if(dm(i)+eps.ge.d) go to 510
      yt(i)=yt(i)+dm(i)-d
510   continue
1000  continue
```

```
c make partial primal assignment
      do 2 i=1,n
      do 3 j=1,n
       cc=c(j,i)
       if(j.eq.1) goto 4
       if(cc-ui.ge.eps) goto 3
4      ui=cc
       j0=j
3      continue
       ys(i)=ui
       if(ize(j0).ne.0) go to 2
       ize(j0)=i
       iperm(i)=j0
2     continue
```

```
c initialize dual variabales (ys, yt)
      do j=1,n
       if(ize(j).eq.0) yt(j)=sup
      enddo
      do 6 i=1,n
      ui=ys(i)
      do 7 j=1,n
      vj=yt(j)
      if(vj.le.eps) go to 7
      cc=c(j,i)-ui
      if(cc+eps.ge.vj) go to 7
      yt(j)=cc
      ivor(j)=i
7     continue
6     continue
```

Enlarge primal assignment

Update dual variables

[2] T. Bonniger, G. Katzakidis, R.E. Burkard, U. Derigs, “Solution Methods with FORTRAN-Programs”, Springer-Verlag Berlin (1980).

Your Application!

Assign project teams?

Dating app? : optimal assignment of stable marriages

Your idea goes here!

Conclusion: The Hungarian alg. is a primal-dual poly.-time solution to the LSAP