The Linear Sum Assigment Problem

Scientific Computing Class

Group 14:

Melisa Akdemir
Ludwig Austermann
Tabea Both
Matthias Personnaz

TU Berlin

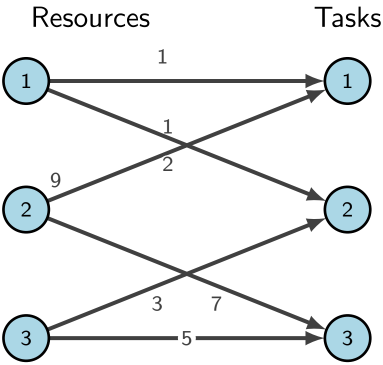
17.2 2023

The Linear Sum Assignment

Problem: Minimize the cost of an Allocation between resources and tasks.
Definitions: The costmatrix $C^{n,m}$, the entry $c_{i,j}$ defines the cost for assigning resource i to task j
Applications: job problem, special assignment problem, minimum weight bipartite matching.

The Linear Sum Assignment Problem

Graph representation



The Linear Sum Assignment Problem
Mathematical Introduction

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} x_{i,j} & & \text{(total cost)} & (1)\\
\text{subject to} \quad & \sum_{j=1}^{n} x_{i,j} = 1 \quad \forall i \in 1, \ldots, n & & & (2)\\
& \sum_{i=1}^{n} x_{i,j} = 1 \quad \forall j \in 1, \ldots, n & & & (3)\\
& x_{i,j} \in 0, 1 \quad \forall i, j \in 1, \ldots, n & & & (4)
\end{aligned}
$$

A greedy, parallelized approach

Motivations

- Test every possible permutation amongst $\mathfrak{S}_n$ and keep the one

$$\arg\min_{\sigma} \sum_{i=1}^{n} C_{i,\sigma(i)} \tag{5}$$

- Stupid? Some problems might need to solve multiple small-scale assignment problems on local data ($n \sim 10^1$), or as proxy heuristics.
- Examples: Task Scheduling, iterative problems with incomplete information, TSP on clustered graphs, etc.
- Doesn't compete with polynomial algorithms for large instances. Use cases are different.

A greedy, parallelized approach
Strategy

- Proposition: make use of GPUs with great parallel capabilities.
- $\Rightarrow$ Simpler set of arithmetic capabilities.
- $\Rightarrow$ Need a way to efficiently enumerate & distribute permutations AND minimize memory transfers.
- *Encode* the permutations with *numbers*.
- Requires *decoding* them with simple computational objects (a.k.a. numbers, arrays + arithmetic operations. No sets, nor fancy data structures in C language).

A greedy, parallelized approach
Encoding

- Lehmer encoding

$$L : \mathfrak{S}_n \longrightarrow \{0\} \times [\![0, 1]\!] \times \cdots \times [\![0, n-1]\!] \tag{6}$$

$$\sigma \longmapsto L(\sigma) = [\#\{j < i, \sigma(i) > \sigma(j)\}]_{1 \leq i \leq n} \tag{7}$$

- Factorial number system

Radix/Base	8	7	6	5	4	3	2	1
Place value	7!	6!	5!	4!	3!	2!	1!	0!
Place value in decimal	5040	720	120	24	6	2	1	1
Highest digit allowed	7	6	5	4	3	2	1	0

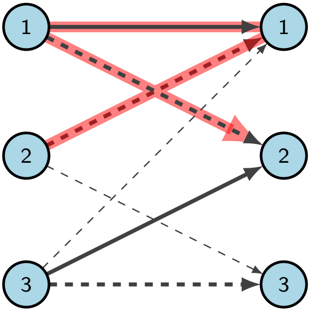
A greedy, parallelized approach

Scopes & limitations

- On modern GPU systems, scales well *in terms of parallelization*.
- Memory complexity lies within $\Theta(k \cdot n^2 \log n)$ with k: number of concurrently running threads.
- Practically, single precision will limit applicable range to $n < 13$ and double integer precision to $n < 21$.
- Easily reach 10 to 1000 billion permutations checked per second depending on hardware.

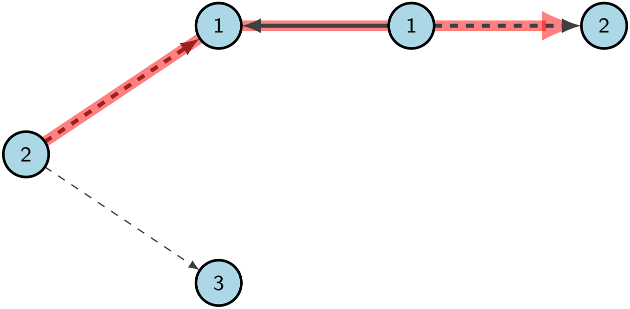
Hungarian Method

- is a primal-dual algorithm
- use a submatching to find a higher order submatching
- feasible dual solution & partial primal solution
- each iteration find optimal assignement on reduced cost subgraph
- we use therefore augmenting paths



Hungarian Method
simple Version

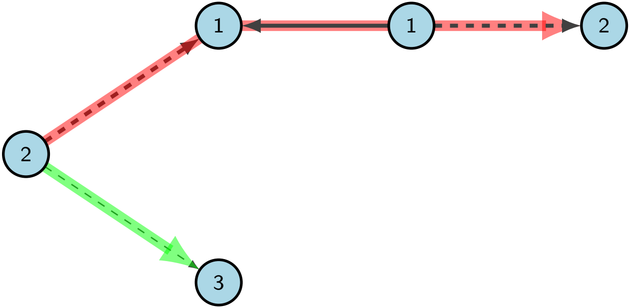
Idea: take one vertex k and build an alternating tree rooted at k.



Hungarian Method

Shortest Augmenting Path Version

Idea: take one vertex k and find the shortest augmenting path. We do not need the whole incremental graph during computation, therefore the cost has to be carried over.



Signature Method

Signature Method is an implementation of the Dual Pivoting Algorithm for the Assignment Problem which has $\mathcal{O}(n^3)$ time complexity where n is the number of tasks.
The algorithm starts by constructing a feasible solution to the problem in the form of a tree, called the dual feasible tree. It then iterates over the tree, pivoting on an edge in the tree, to obtain a new tree. This process is repeated until no more improvements can be made to the solution.

Signature Method

comment: initialization (dual feasible tree of level $n-1$);
$T := ([1, j] : j = 1, 2, ..., n), u_1 := 0$;
for $j := 1$ **to** n **do** $v_j := c_{1j}$;
for $i := 2$ **to** n **do**
  $j^* := argmin(c_{ij} - v_j : j \in V), u_i := c_{ij^*} - v_{j*}, T := T \cup ([i, j^*])$;
endfor
for $k := n-1$ **down to** 2 **do** [**comment**: k = level of T]
comment: decrease by 1 the level of the current tree;
  select a row vertex $t \in U$ with $d_t(T) = 1$ as the target, and set $s := 1$;
  repeat
   comment: dual pivoting;
   let l be the first column vertex in the path connecting s to t in T;
   $T := T \setminus ([s, l])$, and let $T^s, T^l (s \in T^s, l \in T^l)$ be the resulting subtrees;
   $\delta := \min(c_{ij} - u_i - v_j : i \in U(T^l), j \in V(T^s))$;
   let $[s^*, l^*]$ be an edge for which δ is achieved, and set $T := T \cup ([s^*, l^*])$;
   for each $i \in U(T^l)$ do $u_i := u_i + \delta$;
   for each $j \in V(T^l)$ do $v_j := v_j - \delta$;
   $s := s^*$
  until $d_s(T) = 2$
endfor
comment: determine the optimal primal solution;
let r be the unique row vertex having $d_r(T) = 1$;
for each odd edge $[i, j] \in T$ **do** $x_{ij} := 1$ [**comment**: $x_{ij} = 0$ for all other edges]

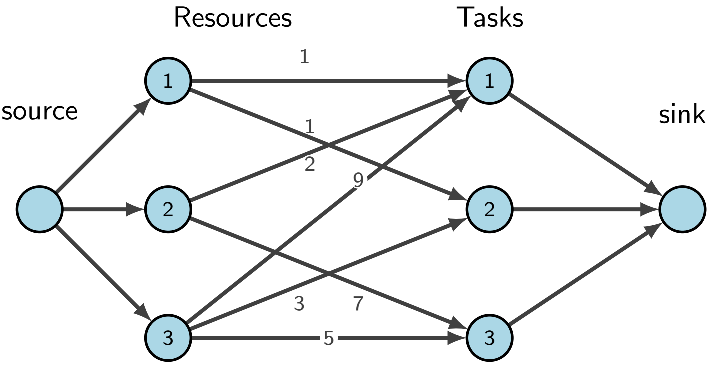
Signature Method

The algorithm works as follows:

1. Initialize the dual feasible tree (T) of level $n - 1$ and compute the dual variables ($uandv$) associated with the tree.
2. Select two nodes s and t in the tree and obtain the path between them.
3. Perform the dual pivot operation on the selected edge (s, l), where l is the neighbor of s in the path between s and t. This operation involves removing the edge (s, l) from the tree, splitting the tree into two subtrees (T^s and T^l), adding the edge $[s^*, l^*]$ to T, where s^* and l^* are the vertices that achieve delta, and updating the cost matrix and the dual variables.
4. Repeat the steps 2 and 3 until no more improvements can be made to the solution, meaning until degree of s is 2.

Minimum Cost Flow

Reformulation



Minimum Cost Flow

Reformulation

Mathematical formulation:

$$\text{minimize } \textstyle\sum_{i,j \in V} \underbrace{f(i,j) * c(i,j)}_{\text{total cost}}$$

Constraints:

$$\begin{array}{cr} 0 \leq f(i,j) \leq 1 & \rightarrow \text{one-to-one matching} \\ f(i,j) = -f(j,i) \quad \forall j, i \neq \{s,t\} & \rightarrow \text{matching property} \\ \sum_{j \in V} f(s,j) = n = \sum_{j \in V} f(j,t) & \rightarrow \text{complete matching} \end{array}$$

Optimal solution is stored in binary flow variable $f(i,j)$ on edge.

Minimum Cost Flow

Successive Shortest Path Algorithm

Input: Flownetwork $G = (V, E, s, t)$; n number of flowamount;
Output: Minimal cost flow;

```
Initialize e.flow is 0 on all edges;
Flow amount is 0;

while Flow amount smaller than n do
    Find augmenting path s − t Path P in residual Graph G_f;
    Find minimal residual capacity δ in Path P;
    Push flow amount δ on path P to augment flow;
    Increase flowamount with δ;
    Update G_f;
```

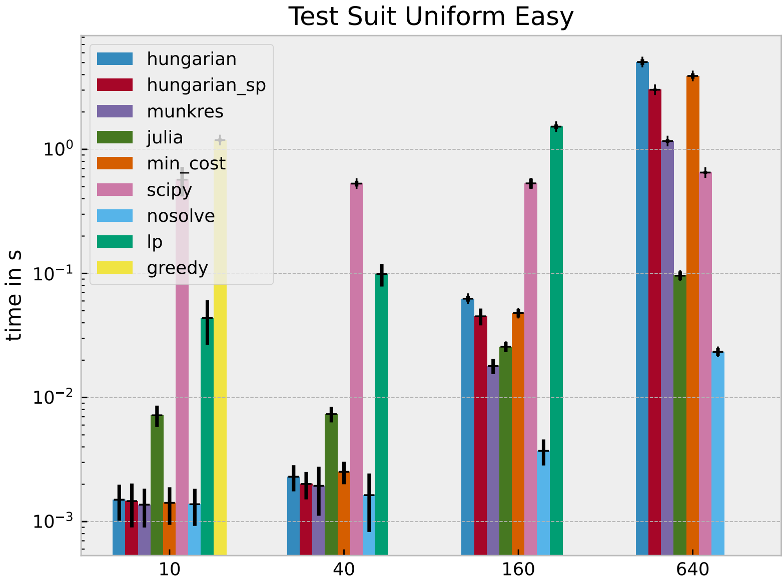
The Linear Sum Assigment Problem

Successive Shortest Path Algorithm

Input: Costmatrix $C^{n,m}$;
Output: Optimal cost solution;

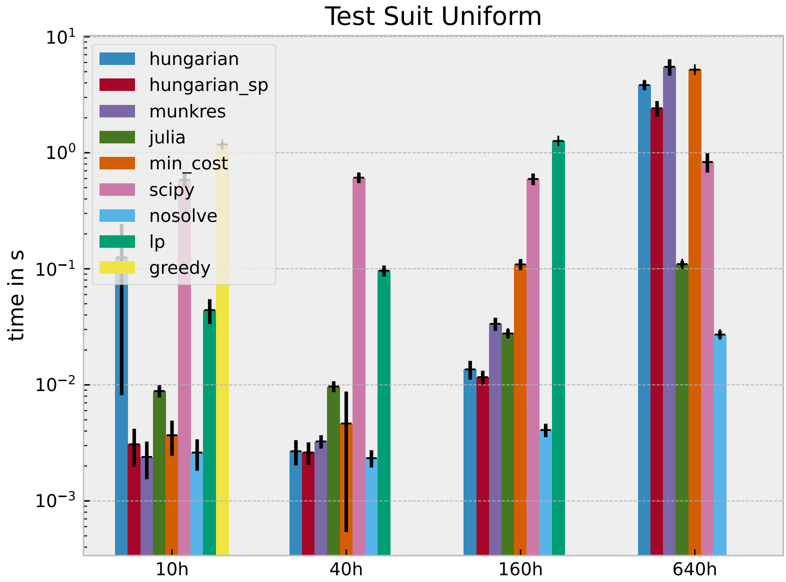
```
Initialize Flownetwork;
Initialize feasible flow 0; amnt =0;

while flow amount smaller than n do
    Find augmenting path s − t Path P in residual Graph G_f;
    Find minimal residual capacity δ in Path P;
    Push flow amount δ on path P to augment flow;
    Increase flowamount with δ;
    Update G_f;
Optimal solution stored in flow;
```

Comparison

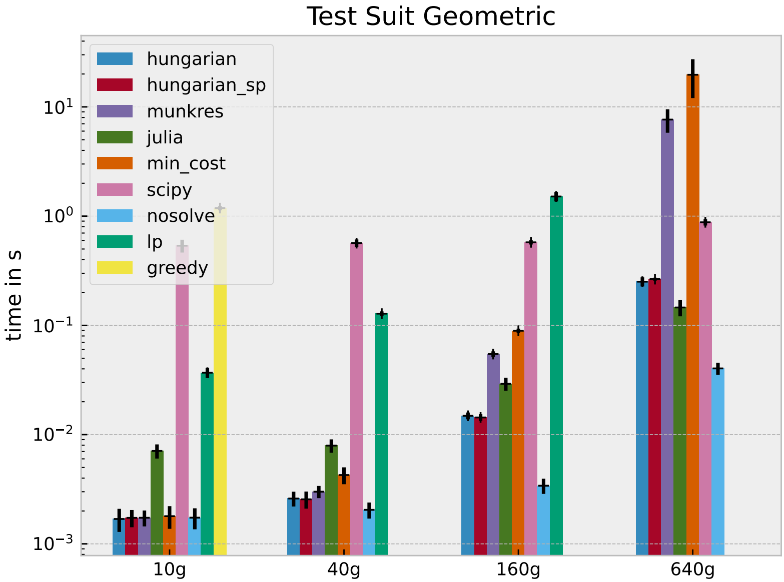


Test Suit Uniform Easy
hungarian
hungarian_sp
munkres
julia
min_cost
scipy
nosolve
lp
greedy
time in s
10^0
10^{-1}
10^{-2}
10^{-3}
10
40
160
640

Comparison

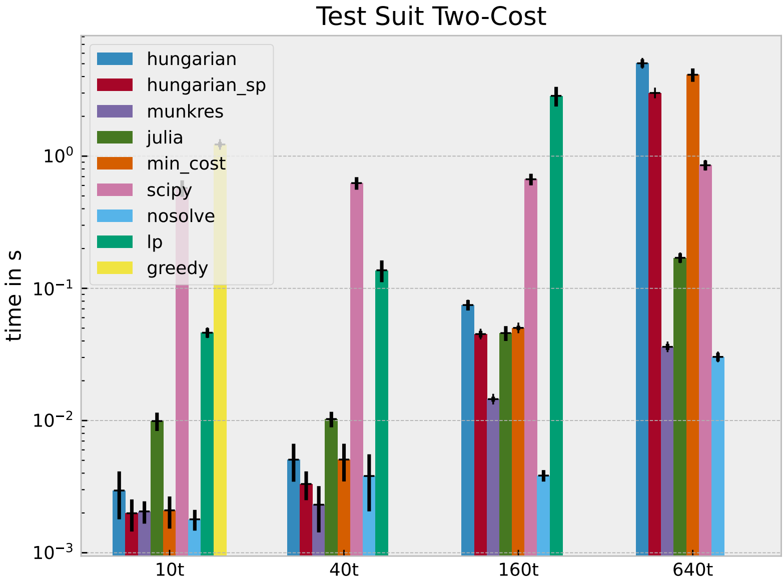

Test Suit Uniform
hungarian
hungarian_sp
munkres
julia
min_cost
scipy
nosolve
lp
greedy
time in s
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10h
40h
160h
640h

Comparison

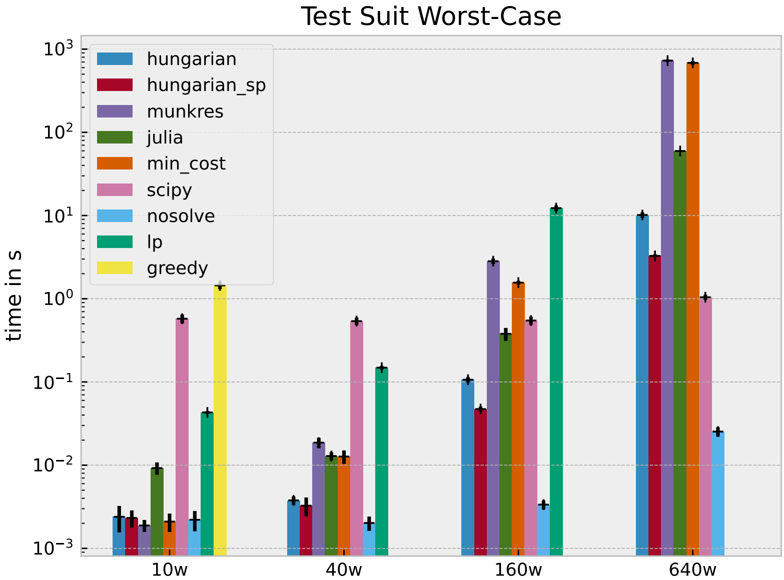


Test Suit Geometric
hungarian
hungarian_sp
munkres
julia
min_cost
scipy
nosolve
lp
greedy
time in s
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10g
40g
160g
640g

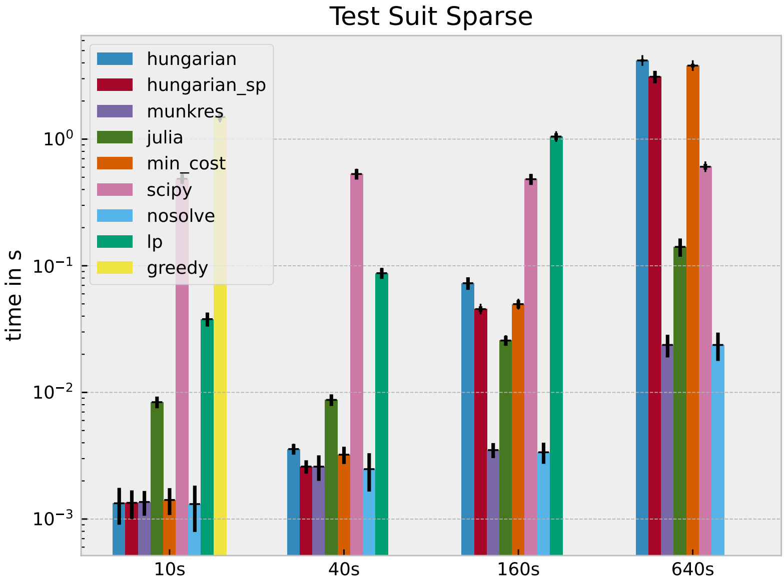
Comparison


Test Suit Two-Cost
hungarian
hungarian_sp
munkres
julia
min_cost
scipy
nosolve
lp
greedy
time in s
10^0
10^{-1}
10^{-2}
10^{-3}
10t
40t
160t
640t

Comparison


Test Suit Worst-Case
hungarian
hungarian_sp
munkres
julia
min_cost
scipy
nosolve
lp
greedy
time in s
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
10w
40w
160w
640w

Comparison



Test Suit Sparse
hungarian
hungarian_sp
munkres
julia
min_cost
scipy
nosolve
lp
greedy
time in s
10^0
10^{-1}
10^{-2}
10^{-3}
10s
40s
160s
640s