Task Assignment Problem in Camera Networks

Federico Cerruti Mirko Fabbro Chiara Masiero

Corso di laurea in Ingegneria dell'Automazione
Università degli Studi di Padova

Progettazione di sistemi di controllo

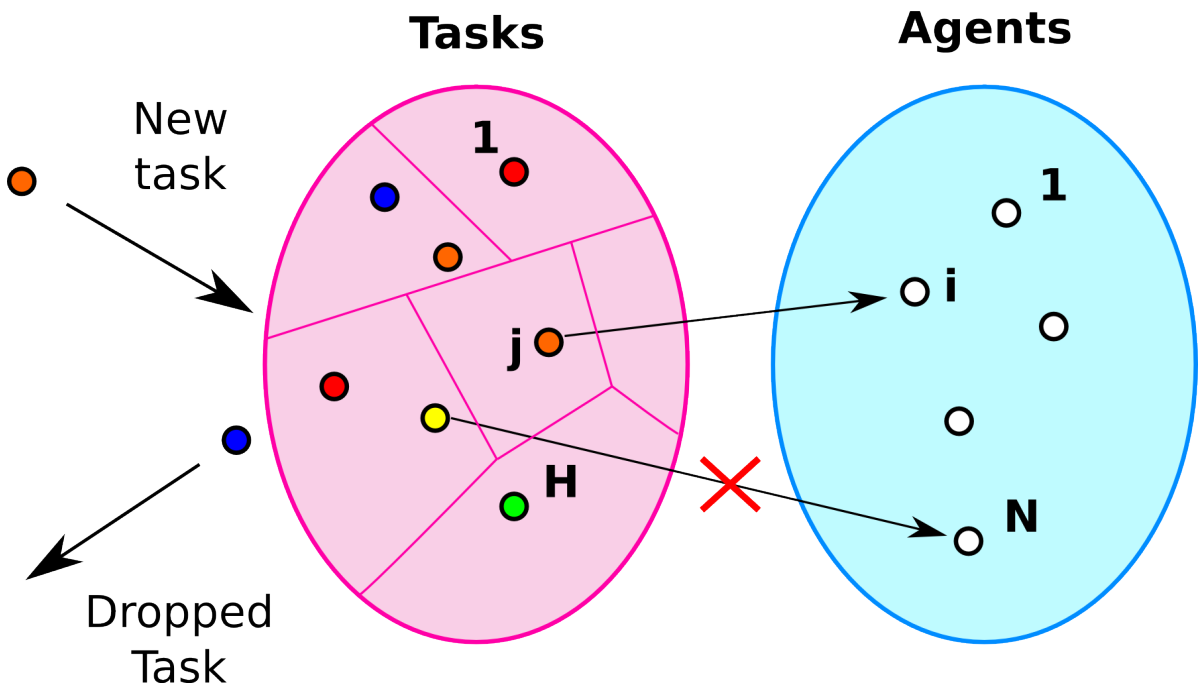
Prof. Schenato Luca

18 Febbraio 2010

Summary

Task Assignment Problem: Definition

We are given:



- a set $\mathcal{A}$ of agents a_i distributed over a certain area to monitor, such that
  - They have to obey to network constraints;
  - They have limited resources.
- a time-variant set $\mathcal{T}$ of tasks to be executed

Objectives

- We aim to attain a good trade-off between optimality and continuity of task execution.

Standing assumption

We will focus mainly on the assignment problem $\Rightarrow$ simplification

State-of-the-art Approaches

From the view-point of theory, many solutions are available. They are mainly based on:

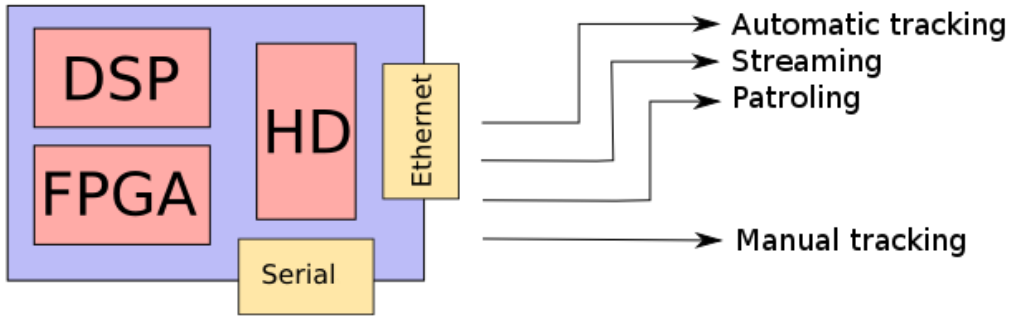
- Dynamic systems identification
- Auctions
- Game theory

Real camera networks

Assignment is performed by nearly Brute-force algorithms

Description of Cameras

Cameras a_i are agents with limited resources:



- The monitored area can be divided into subzones, intersecting the visual ranges of the cameras.

a_i covers one or more subzones

$\Downarrow$

a_i can execute only a subset of all possible tasks

Description of Tasks

Tasks can be:

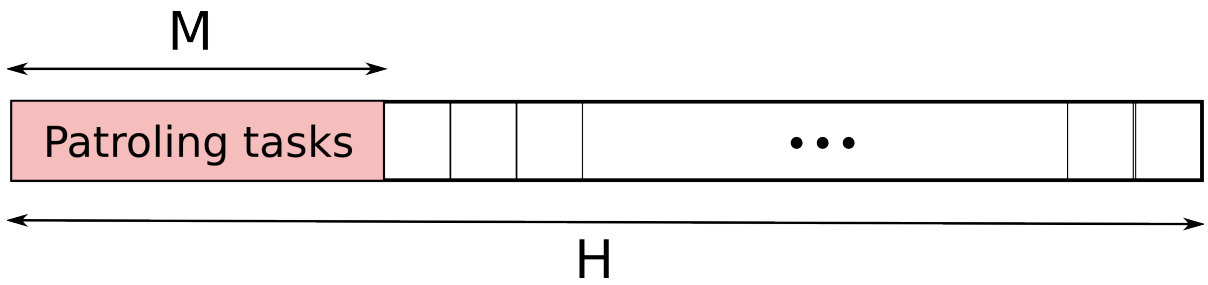
- Synchronous: *snapshots* and *heartbeat*
- Asynchronous:
  1. Manual tracking “Op”
  2. Automatic tracking “Trk”
  3. Streaming “Str”
  4. Patroling “Pat”

Our aim

We want to solve the assignment problem for asynchronous tasks, that can be considered the out-and-out ones.

The Pool

- In order to assure homogeneousness we model patroling activity as a task
- When a task occurs, it is added to the pool



- A task is removed when it is completed or gets obsolete $\Rightarrow$ dropping
- The pool is a global structure

A CLP Model for Task Assignment 1

- Given an istantaneous pool configuration, with H tasks and N agents, it is easy to lay out a CLP model:

CLP Model

$$x_{ij} = \begin{cases} 1 & \text{if agent } a_i \text{ executes task } t_j \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{j=1}^{H} x_{ij} \leq 1 \quad \forall i \quad \text{each agent } a_i \text{ can execute at most one task}$$

$$\sum_{i=1}^{N} x_{ij} \leq 1 \quad \forall j \quad \text{each task } t_j \text{ is assigned at most to one agent}$$

$$x_{ij} \in \{0, 1\} \, \forall i, j$$

- We still haven't got it over, however...

A CLP Model for Task Assignment 2

- Covering constraints should be considered, but...

Covering constraints

...modeling them as constraints lead to a non-TUM formulation, in general!

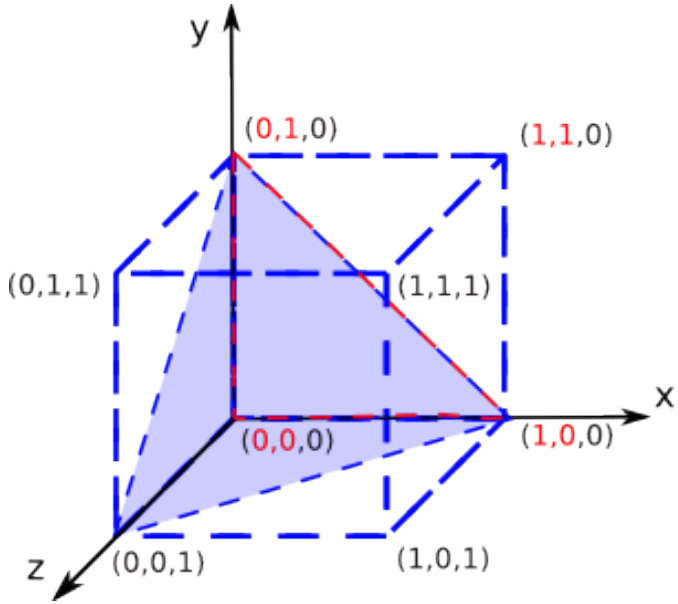
- We skip them out by choosing a clever utility function $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$:

$$c_{ij} = \begin{cases} score(j) & \text{if } a_i \text{ can monitor } t_j.loc \\ -\infty & \text{otherwise} \end{cases}$$

- We now obtain a TUM constraint matrix $\Rightarrow$ Integer constraints are redundant!

A CLP Model for Task Assignment 3

- We can now obtain optimum instantaneous solution, but what about problem dynamics?



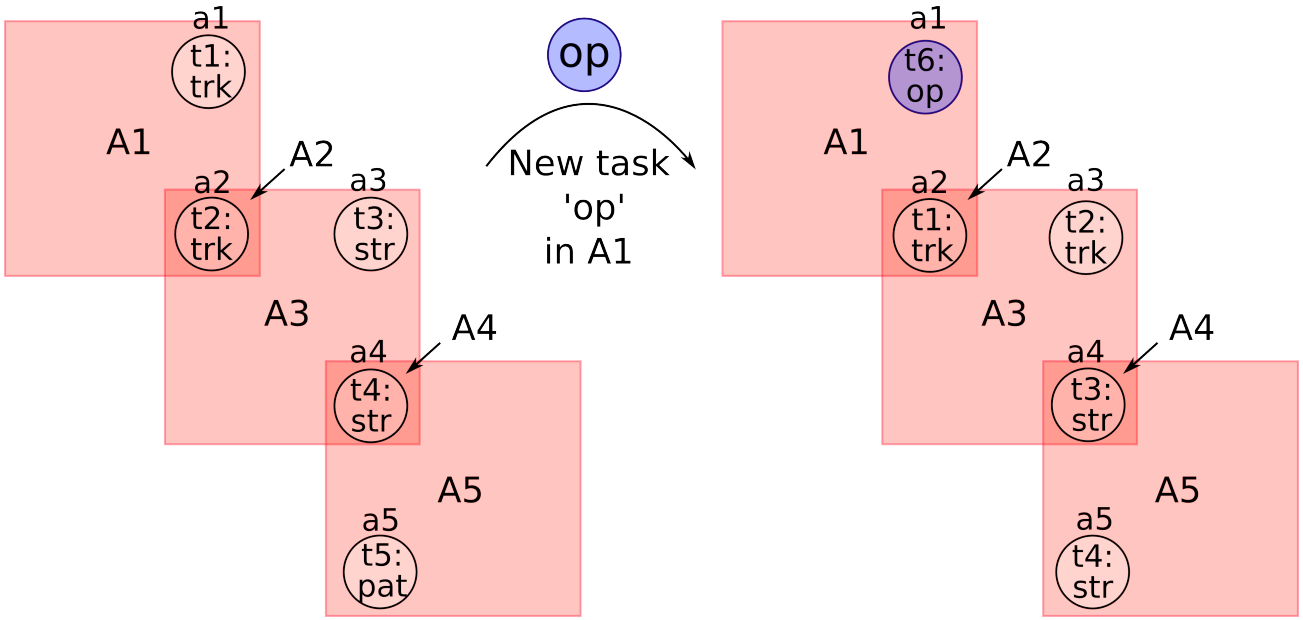
- Modifications in the pool imply variations in the polyhedron of decision variables x_{ij}

A CLP Model for Task Assignment 4

- The number of feasible solutions is

$$|\mathcal{X}| = \sum_{k=0}^{N} \binom{N}{k} \frac{H!}{(H-k)!}$$

- New or completed tasks: it is extremely difficult to understand how the optimum solution changes!



Performance Metrics 1

- A measure of optimality (in the sense of task's type priority) is given by

$$P(t) = \frac{1}{t} \sum_{\tau=0}^{t} \sum_{n=1}^{N} p_n(\tau)$$

- Average discontinuity rate: cases of agents that leave tasks before completing them.

$$D(t) = \frac{1}{t} \sum_{\tau=0}^{t} \sum_{n=1}^{N} d_n(\tau)$$

- Let be the IDLE occurrence rate

$$I(t) = \frac{1}{t} \sum_{\tau=0}^{t} \sum_{n=1}^{N} \mathrm{Idle}_n(\tau)$$

Performance Metrics 2

- The Average waiting time is:

$$W = \frac{1}{|\mathcal{H}_C|} \sum_{h \in \mathcal{H}_C} T_{\text{end}}(h) - T_{\text{occ}}(h) - T_{\text{serv}}(h)$$

- Complexity: expressed in term of $O(\cdot)$.
- Dropping rate:

$$F = \frac{|\mathcal{H}_{\text{drop}}|}{|\mathcal{H}|}$$

with $\mathcal{H}_{\text{drop}}$ set of dropped tasks and $\mathcal{H}$ set of generated tasks.

Instance Parameters and Constraints

- In case of uniform topology, a general load factor can be estimated.

$$C = \frac{1}{N}\left(\frac{\overline{T}^{OP}_{serv}}{\overline{T}^{OP}_{occ}} + \frac{\overline{T}^{TRK}_{serv}}{\overline{T}^{TRK}_{occ}} + \frac{\overline{T}^{STR}_{serv}}{\overline{T}^{STR}_{occ}}\right)$$

- If there is no redundancy in covering areas, i.e.:

$$rank(\mathbf{V}) = N$$

$\Rightarrow$ in optimal solution all agents are busy

The SMP Problem

We are given:

- a set of n men and n women
- a preference list for each person, where all the persons of the opposite sex appear

Objective

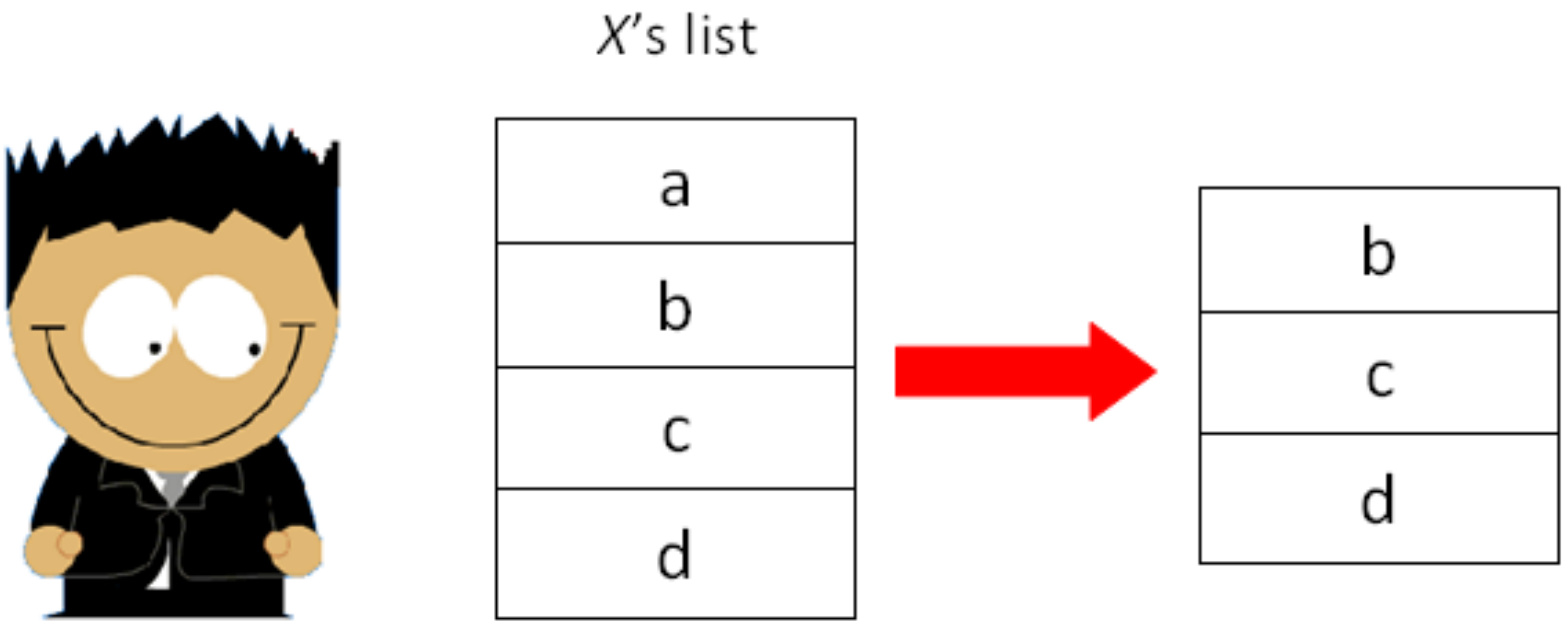
Finding a set of stable marriages between the men and the women.

Stable matching

A marriage is stable if there are no dissatisfied pairs: $\nexists (m, w)$ s.t. m prefers w to his current wife and w prefers m to her current husband

Gale-Shapley Algorithm 1

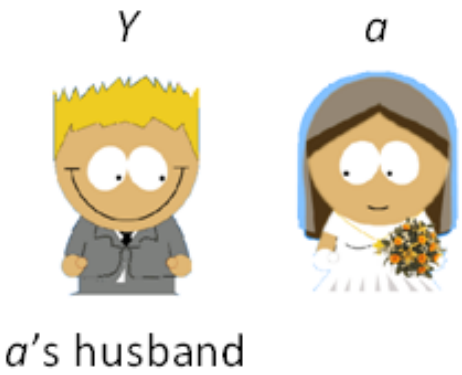
- an unpaired man *X* considers the first woman on his list *a* and removes her from it

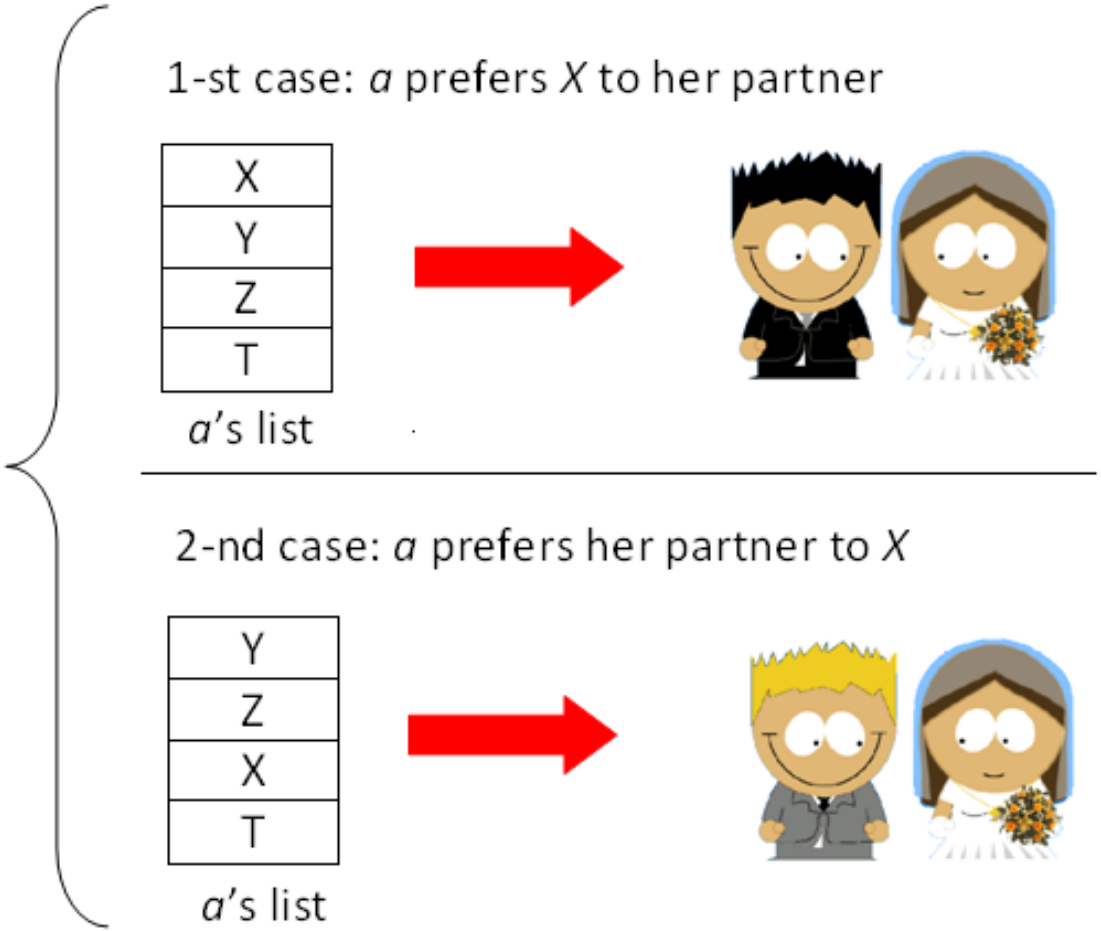


- two situations are possible, depending on *a* 's status

Gale-Shapley Algorithm 2

1. a is not engaged $\Rightarrow$ a accepts X's proposal
2. a is engaged $\Rightarrow$ a can change her husband, depending on her list





Gale-Shapley Algorithm 3

Observations

- Once a woman becomes attached, she remains married, altough she can change her partner
- The termination of the algorithm is assured
- This marriage is stable

Question

Is it possible a direct application of this algorithm to our case study problem?

The SMP Problem: Variants 1

Stable marriage problem with incomplete lists (SMI)

- The number of men and women may be not the same
- Each person's preference list consists of a subset of the members of the opposite sex in strict order

Stable marriage problem with ties (SMT)

- Lists can have ties i.e. no strict order in preferences

Stable solution

- There is always at least one stable matching for an instance of both SMI and SMT

The SMP Problem: Variants 2

Our case:

Stable marriage problem with ties and incomplete lists (SMTI)

- A stable matching can be found by breaking all the ties...
- ... but the ways in which ties are broken affect the solution!
- Finding a solution of maximum cardinality is a NP-hard problem

SMTI Revised Algorithm 1

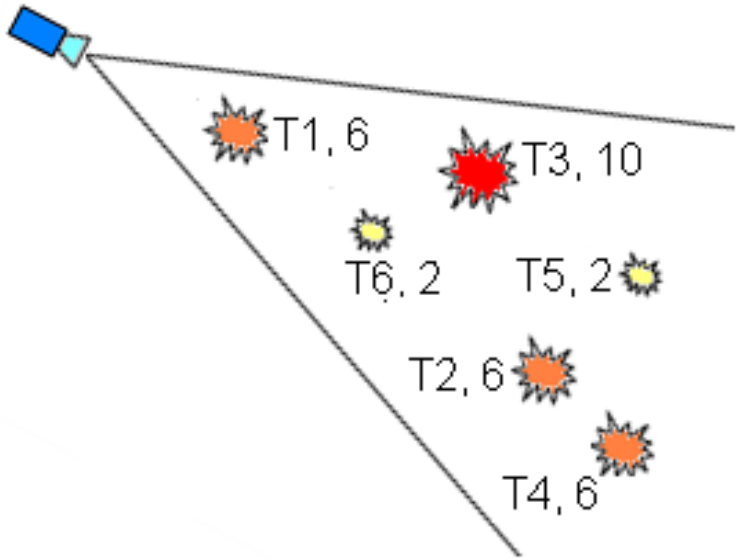
a.k.a. The Wedding Planner Algorithm

Building lists

- agent $\Rightarrow$ list containing all tasks it can execute, sorted by

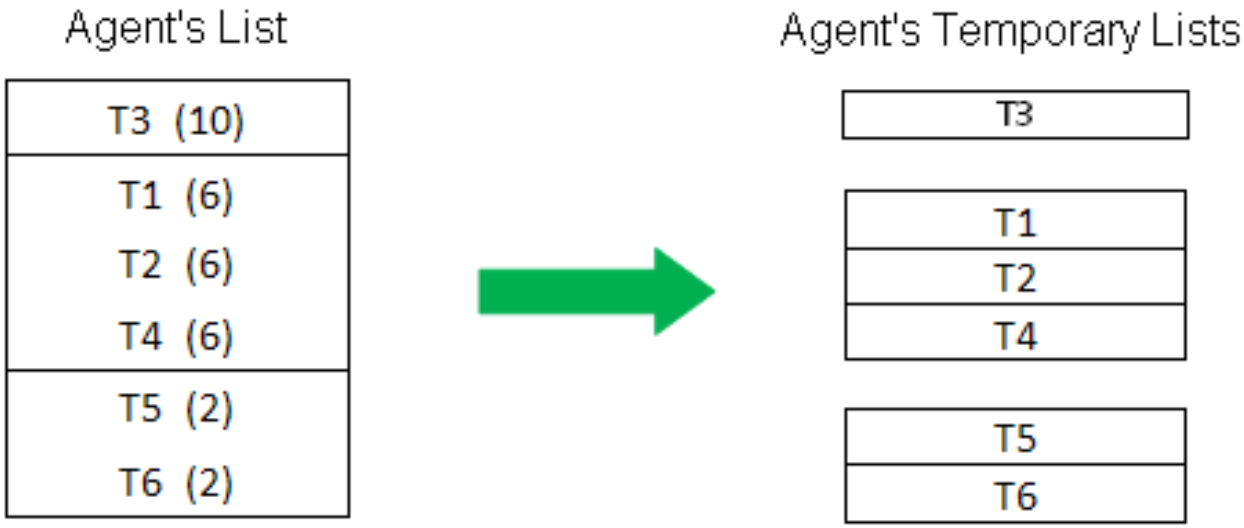
$$score = \alpha \cdot pr_{tsk} + \gamma \cdot \frac{lft_{tsk}}{drop_{tsk}}$$

- $pr_{tsk} \Rightarrow$ intrinsic priority of the task *tsk*
- $lft_{tsk} \Rightarrow$ lifetime of the task *tsk*
- $drop_{tsk} \Rightarrow$ drop time of the task *tsk*
- $\gamma = 1 - \alpha$



SMTI Revised Algorithm 2

- Ties $\Rightarrow$ Temporary lists of tasks t_j



- Current agent a_i proposes to tasks of the current Temporary List (*currTempList*)

Three situations are possible, depending on status of the tasks

SMTI Revised Algorithm 3

1. a_i already executing its favourite task $\Rightarrow$ no change
2. $\exists\, t_j \in currTempList$ unmatched $\Rightarrow (a_i, t_j)$ now matched
3. $\nexists\, t_j \in currTempList$ unmatched $\Rightarrow$ there may be a swap

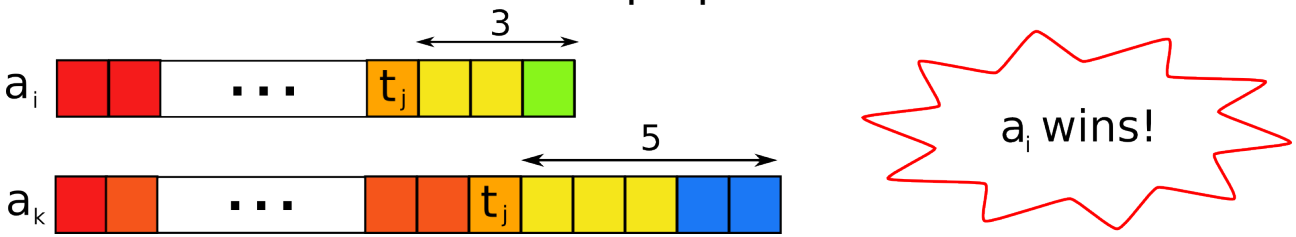
- if a_i still unmatched, a new temporary list (lower score) is built
- if a_i's list has come to an end, another agent a_k is considered

Termination

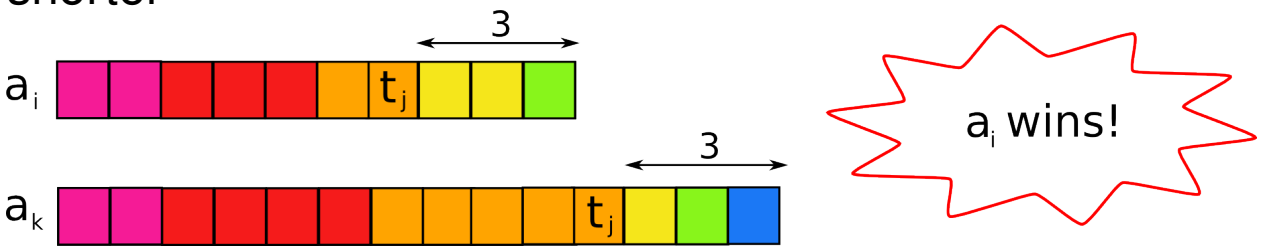
The algorithm is repeated until it attains a stable matching.

Swap Policy

- a_i asks for a task t_j being executing by a_k. There is a swap if:
  1. a_i has fewer left tasks it can propose to



  2. a_i and a_k have the same number of residual tasks, but a_i 's global list is shorter



Motivation

We want to prevent the tasks from being idle, possibly

Proposed Algorithms

SMTI Revised algorithm will be compared with:

- Randomized SMTI Revised
- Centralized Assignment
- Nearly Brute-force (Purely Random) Assignment
- Greedy Assignment

SMTI Revised

To begin with, let's analyse our approach...

- Computational complexity $O(N^2)$.

Pros

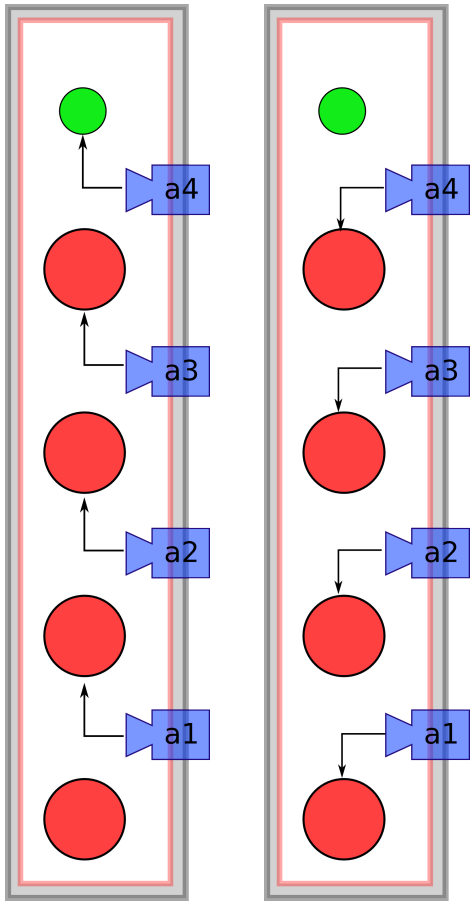
1. No deadlocks (Swap rule)
2. Scalable

Cons

1. Continuity is not assured
2. Assignment is stable but it could be not optimal
3. There could be IDLE agents

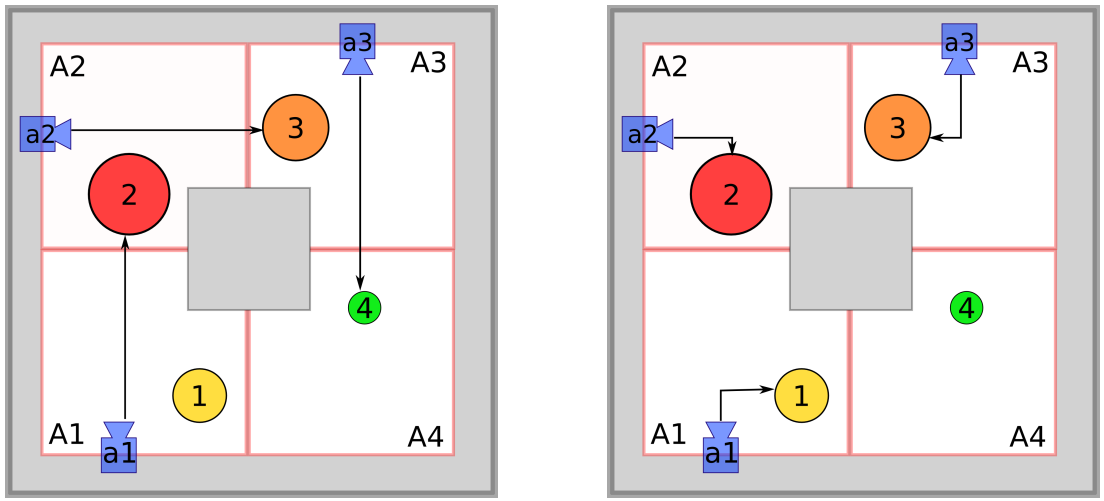
SMTI Revised: Critical Situations

- Non-optimality from the view-point of the utility function:



- The matching depends on the first considered agent!

Randomized SMTI Revised



- Can randomized swaps solve the problem?
- P [*equivalent agents do swap*] $= p$
  In this case: a_1 and a_2 should swap in the 1^{st} iteration, then they should not: $P = p(1 - p)$

$$\Downarrow$$

There are not sufficient guarantees of improvement!

Centralized Assignment

- It finds the optimum solution of the CLP problem (by means of Simplex, for instance).

Pros

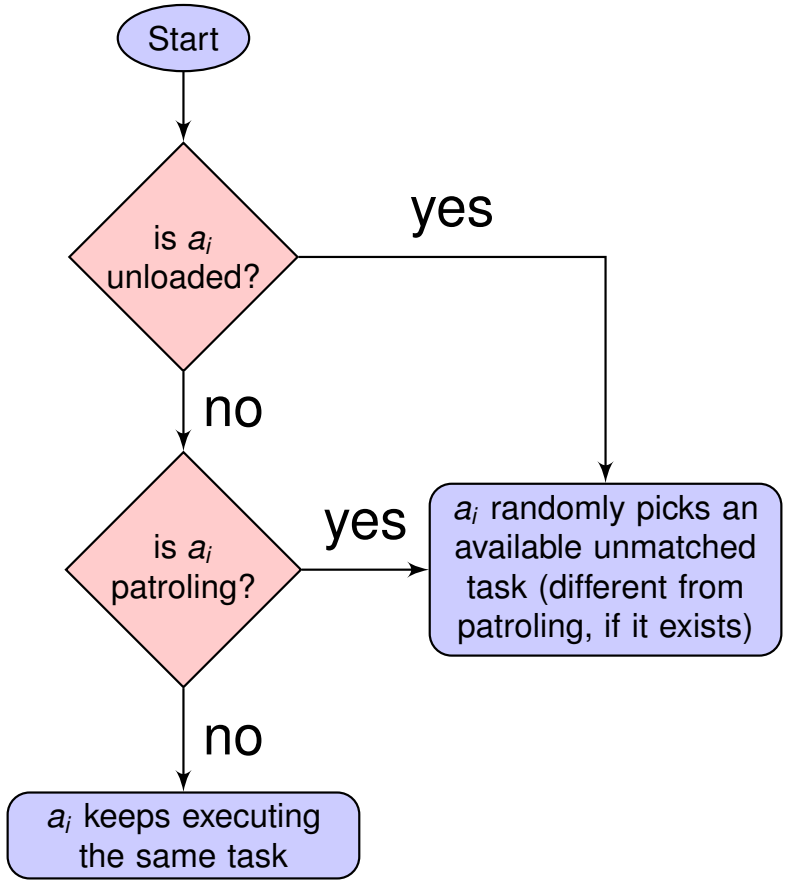
1. Optimality

Cons

1. Discontinuity
2. Worst case complexity: $O(2^{HN})$
3. Not scalable

Nearly Brute-force

- Each agent performs the following sequence of operations:



Nearly Brute-force 2

- Its computational complexity in the worst case is $O(NH)$

Pros

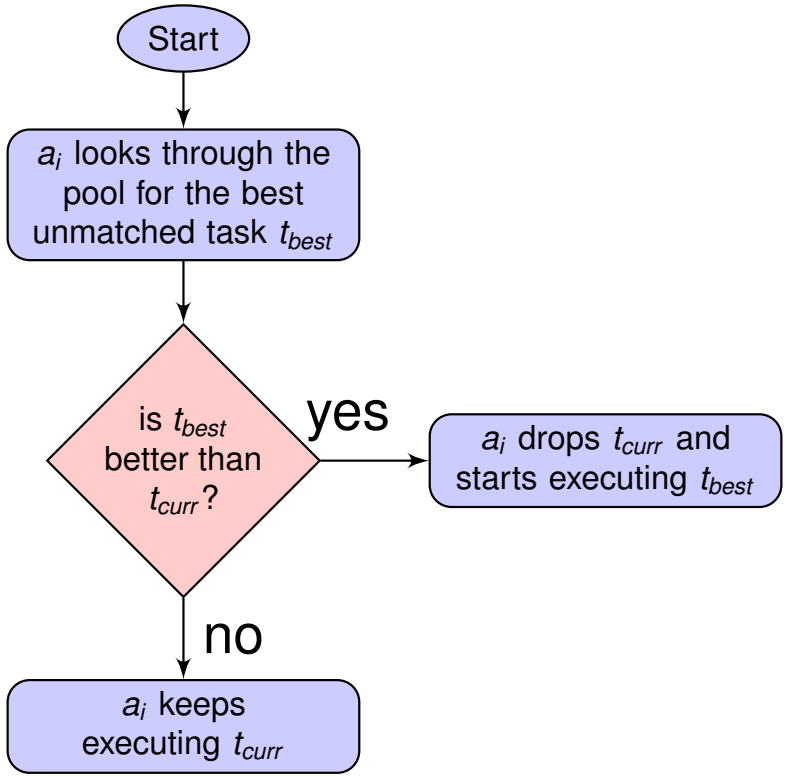
1. Maximum continuity
2. Scalable

Cons

1. No optimality criterion
2. Idle agents

Greedy Assignment

- Each agent performs the following sequence of operations:



Greedy Assignment 2

- The computational complexity is always $O(NH)$

Pros

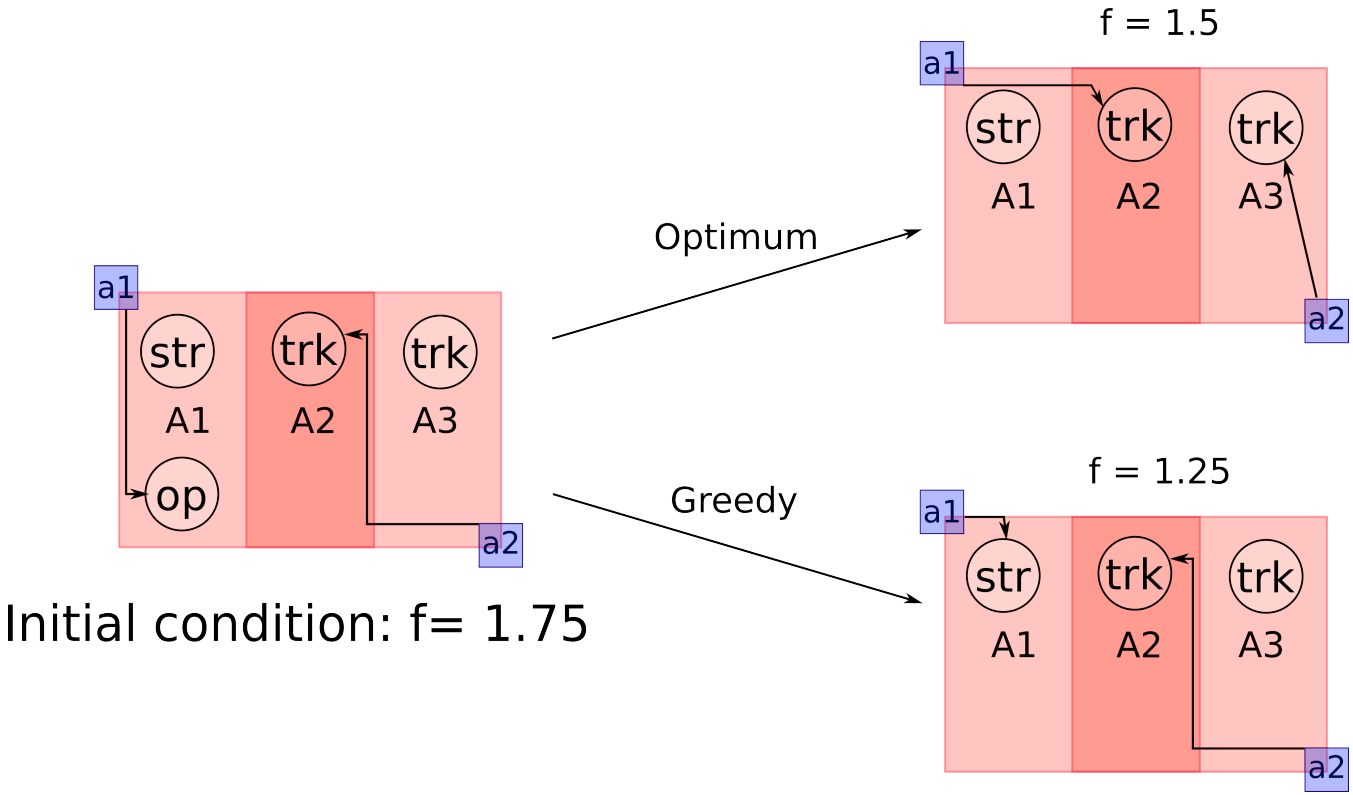
1. Scalable

Cons

1. Continuity is not assured
2. Idle agents

Non optimality of GA and NBf

- GA and NBf do not assure optimality!

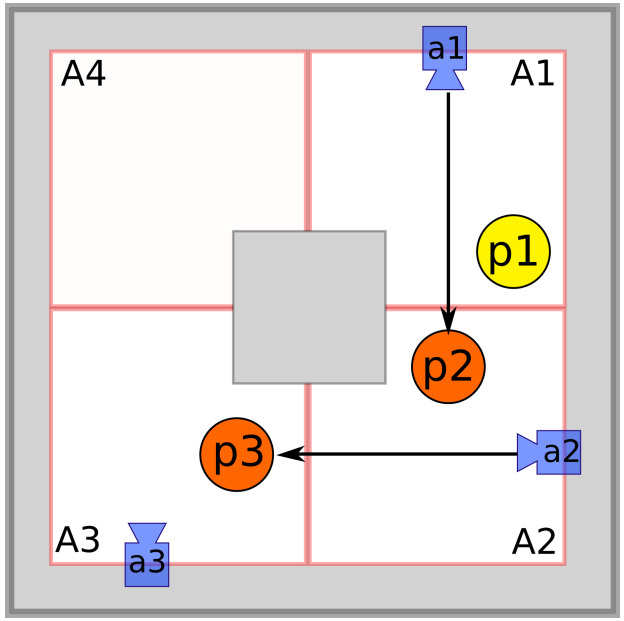


Initial condition: f= 1.75

$$f_{OP} = 1 \qquad f_{TRK} = 0.75 \qquad f_{STR} = 0.5$$

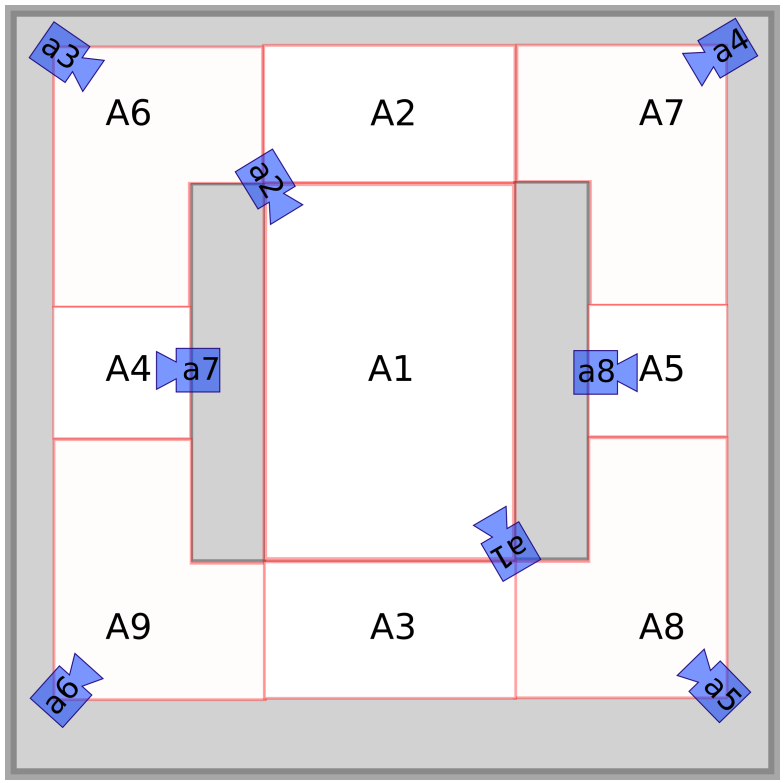
Idle agents occurrence

- With NB-f and GA there may be idle agents!



Simulations

Our framework:



$N = 8 \qquad M = 9$

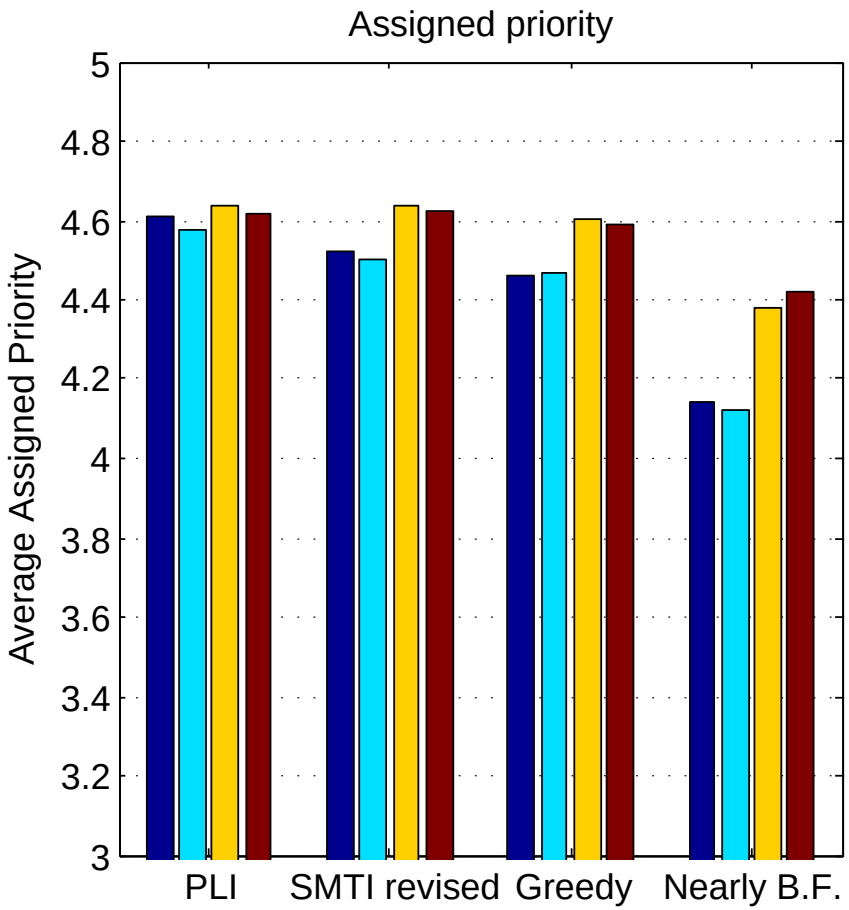
- Covering matrix:

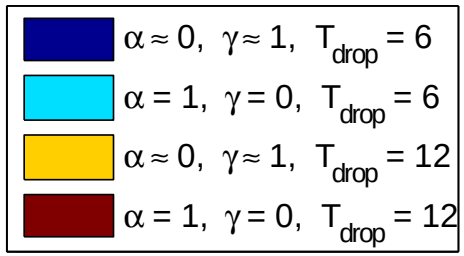
$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

- Interarrival time: $T_{\text{occ}} = 2s$
- Service time: $T_{\text{serv}} = 3s$
- Simulation time: $T = 500s$.
- Load factor: $C \approx 0.48$.

Simulation Environment

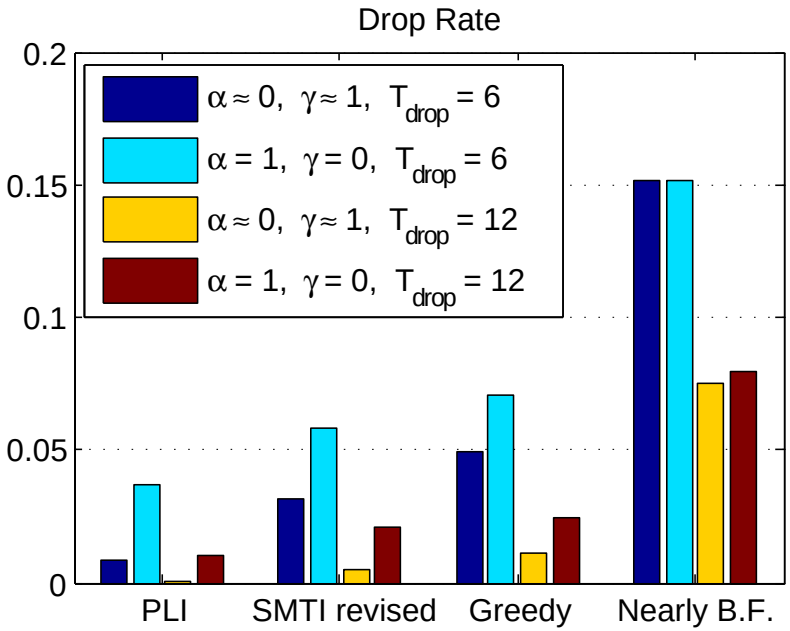
Assigned priority



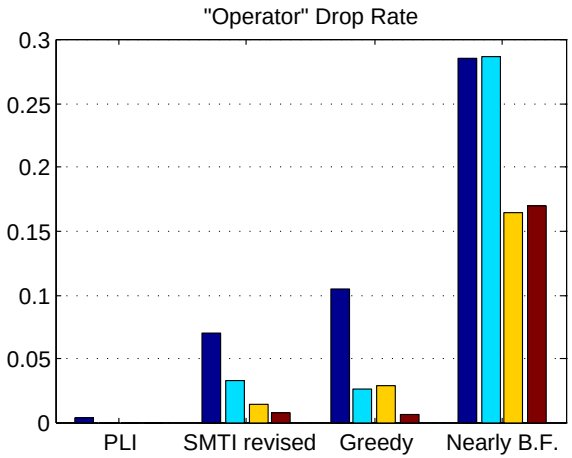


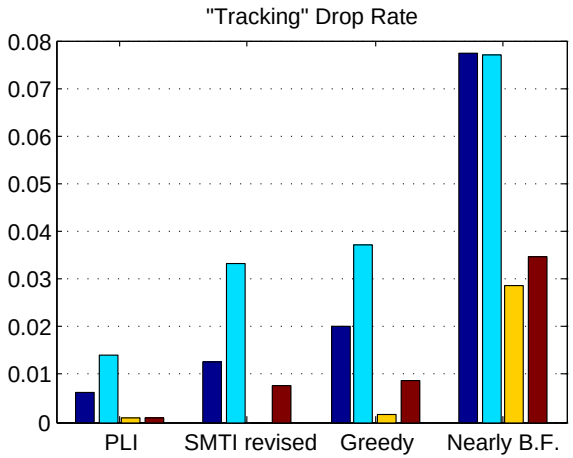
- PLI performs best, as it is designed to maximize instantaneous priorities.
- SMTI Revised does well, and places itself immediately under the top.
- Greedy achieves mean results, especially with longer T_{drop} that provides more feasible tasks in pool.
- NBF does not care about priorities.

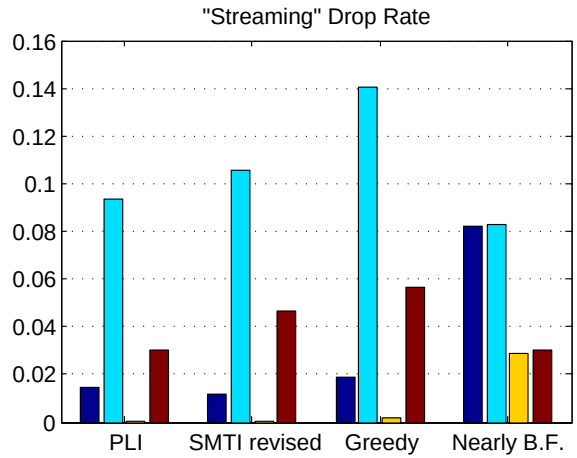
Drop Rate



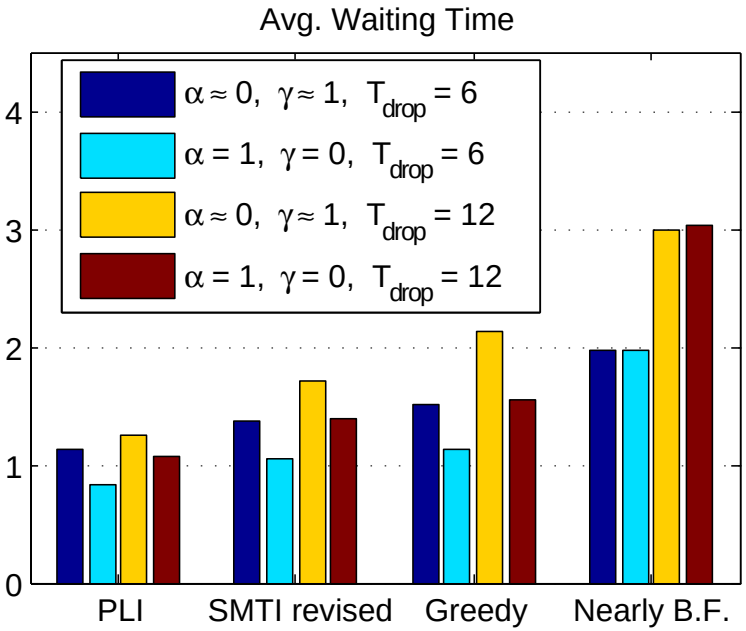
- $T_{\text{drop}} \uparrow \Longrightarrow$ Drop Rate $\downarrow$
  - *Upper bound defined by the practice implementation.*
- Big α favours only the highest priority tasks.



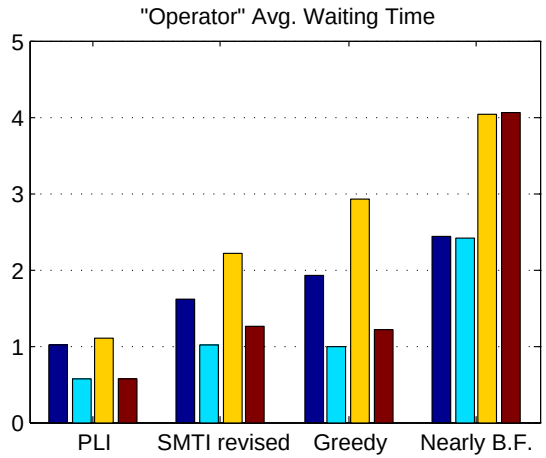


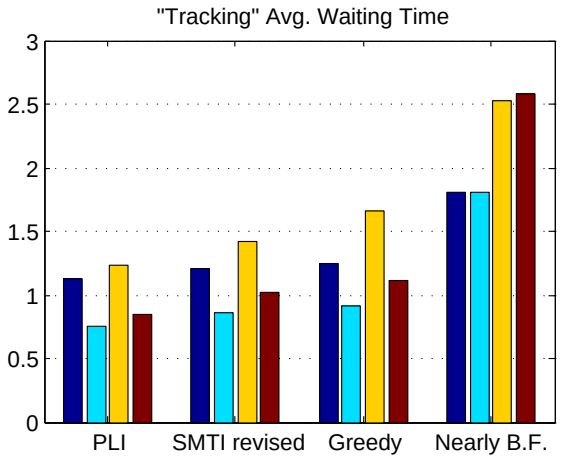


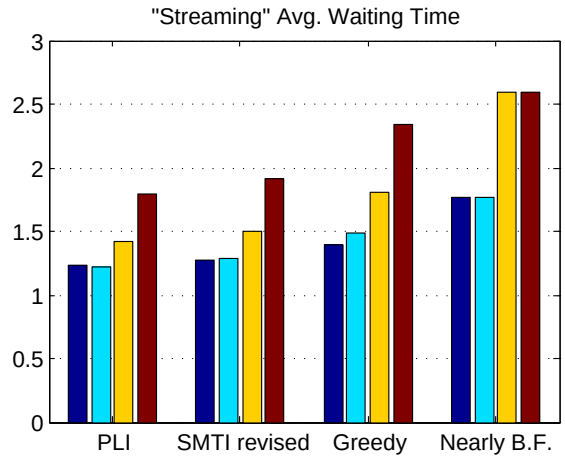
Average Waiting Time 1



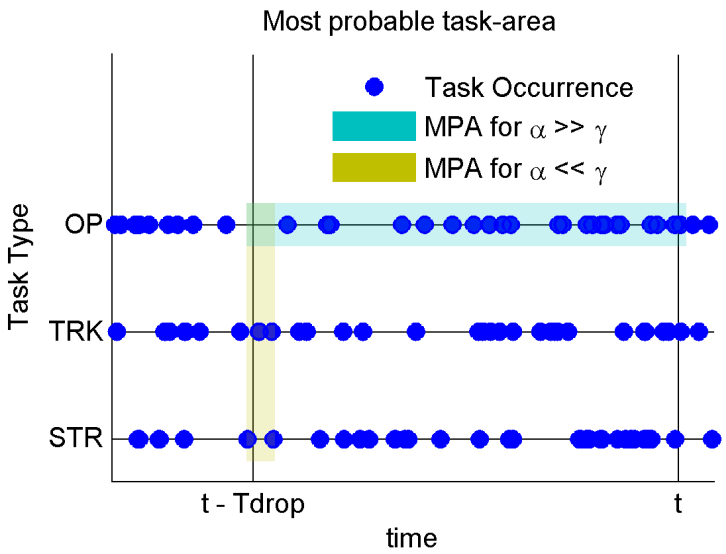
- Longer T_{drop} allows larger queue. Waiting time increases.
- In general big α means short queue time for tasks, but...







Average Waiting Time 2

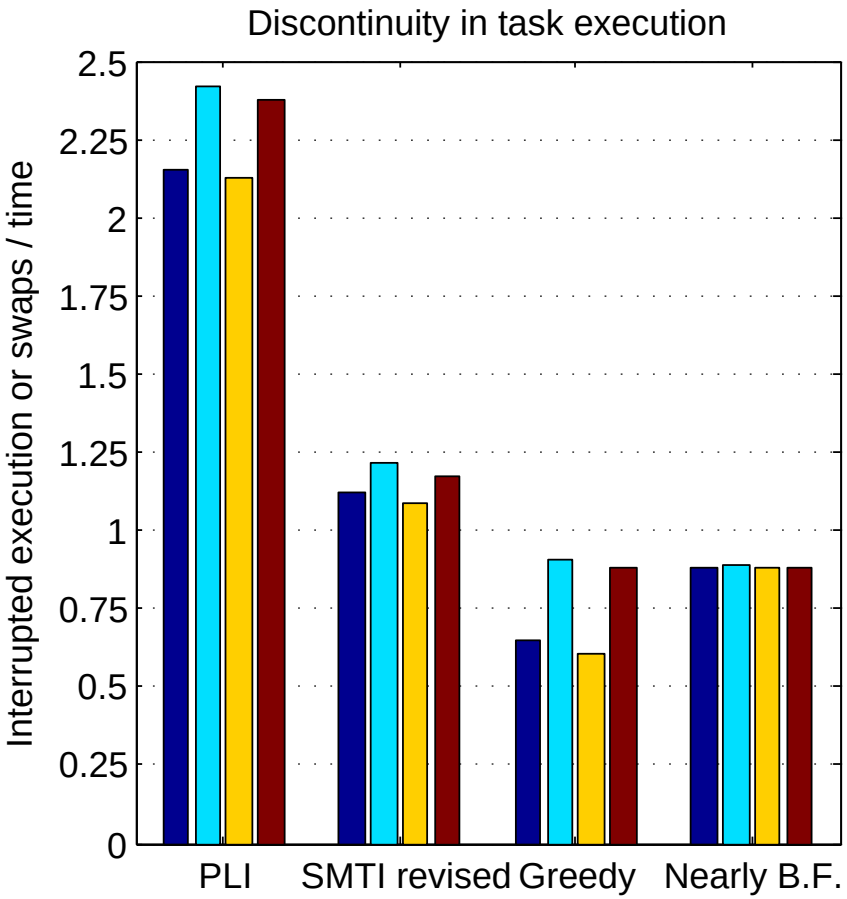


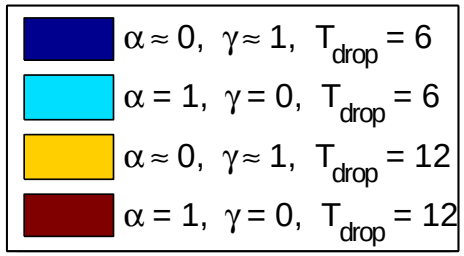
Most probable area of task execution:

- Big α
  Uniformely distributed over time, but shrinked around high priority tasks.
- Small α
  Uniformely distributed over priorities, but closer to the drop limit.

. . . the lowest priority tasks (streaming) do not profit by α, because they are neglected in favour of higher priority tasks!

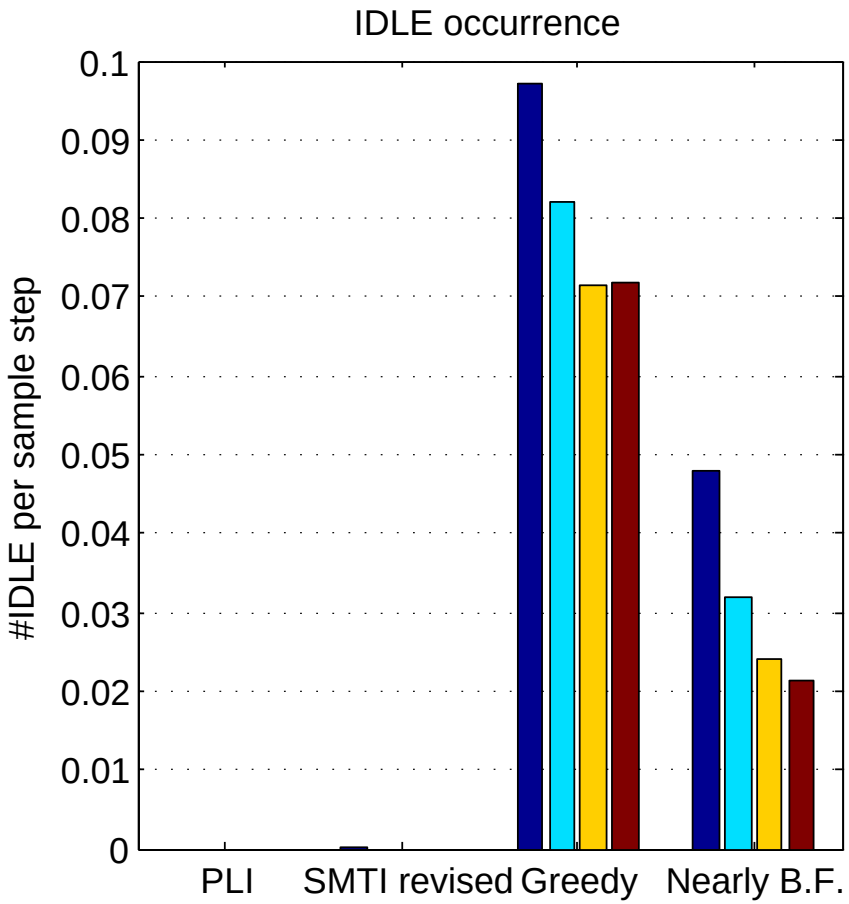
Discontinuity in task execution

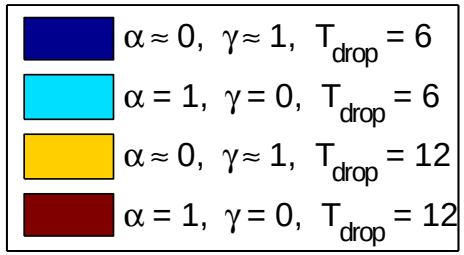




- PLI doesn't care about continuity.
- Greedy and Nearly Brute Force are extremely conservative in assignment.
- SMTI Revised is a good trade off. It leaves past configurations if there are chances to improve assigned priority.

IDLE occurrence





- Under previously stated conditions, PLI never shows IDLE agents.
- SMTI Revised achives good performances, as it swaps agents trying to obtain a better balance.
- In Greedy each agent considers only the best solution for itself.
- Nearly Brute Force performances are random. In average it is placed between Greedy and SMTI Revised.

Conclusions

Centralized assignment

- Instantaneously optimum assignment
- Bad scalability and continuity
- No distributed implementation

SMTI Revised

- Good trade-off between optimality and continuity
- Scalable
- Distributed implementation available

Greedy and NBF

- Strongly dependent on agents sequence (no swap)
- Easy to implement: scalable, potentially distributed and cheap
- Poor performances (especially for NBF)

Future Work

- Communication costs
- Distributed version
  - Consistent pool
  - Synchronization and conflicts management (consensus or leader agent)
- Market-based approach
  - Auction mechanism
  - Swap policy
- Extension to PTZ cams

The End

Grazie per l'attenzione!