



# Progetto Esame

## Algoritmi di Ottimizzazione

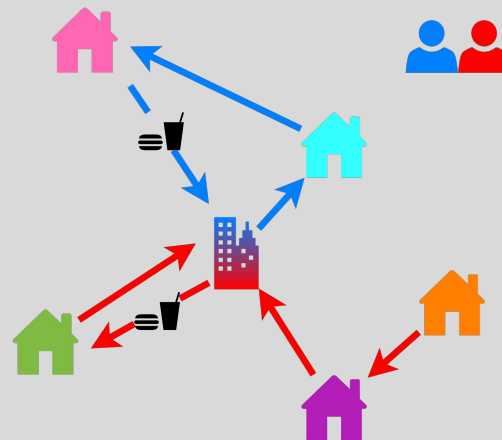
Francesco Ghinelli (Matr. 150117)  
Gabriele Felici (Matr. 150400)

# Problema

Il problema che abbiamo voluto risolvere con i nostri modelli è una rivisitazione del problema di Traveling Salesman Problem with Time Windows.

Il problema infatti definisce la necessità di determinare, dato un set di agenti, i percorsi che faranno per servire i diversi clienti, definendo però alcuni ulteriori vincoli:

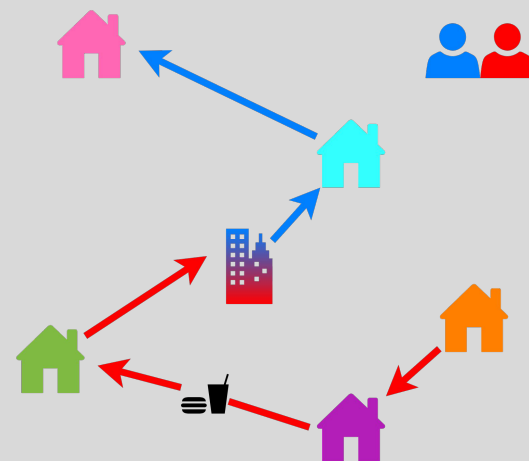
- 30 min per il pranzo tra le 12:00 e le 14:00,
- Tempo lavorativo limitato a 8 ore,
- Tempo attesa e spostamento limitato,
- Tempo minimo da passare in ufficio di 1 ora.



# Problema

Inoltre il problema richiede di:

- Massimizzare attività produttive,
- Prioritizzare i meeting,
- Minimizzare gli spostamenti,
- Favorire grossi blocchi di lavoro uguale (minimizzare i cambiamenti tra lavoro di ufficio e meeting).





# Problema

Tale problema è descritto in modo più dettagliato nel seguente paper:


<https://www.sciencedirect.com/science/article/pii/S157106531000106X>



# Modelli risolutivi

Per risolvere il problema abbiamo deciso di sviluppare due modelli diversi:

- **Modello di risoluzione basato su Programmazione Intera Quadratica**
- **Modello di risoluzione Set cover euristico**




# Modello di risoluzione basato su Programmazione Intermedia Quadratica

Questo modello vuole ottenere la soluzione basandosi unicamente sulla programmazione intera. Per ottenere tale scopo va a definire **6** variabili e **25** vincoli.

## Variabili

<b>x</b>	Modella i percorsi attraversati dai diversi agenti	(booleana $NP \times NP \times NA$ )
<b>s</b>	Modella l'istante nel quale un agente serve un cliente	(intera $NP \times NA$ )
<b>c</b>	Modella la durata del servizio	(intera $NP \times NA$ )
<b>w</b>	Modella il tempo di attesa di un agente	(intera $NP \times NA$ )
<b>l</b>	Modella il punto del percorso nel quale l'agente pranza	(binaria $NP \times NP \times NA$ )
<b>t</b>	Mantiene il più alto tempo di servizio dell'agente, necessario per valutare se l'agente necessita di pranzare	(intera $NA$ )

NA : Numero Agenti  
NC : Numero Clienti  
 $NP = (NC+1)*2$




# Modello di risoluzione basato su Programmazione Intermedia Quadratica

Com'è possibile osservare dalla precedente tabella la dimensione delle matrici usate per creare il modello, presentano quasi sempre il **doppio dei clienti  $((NC+1)*2)$** , tale caratteristica è stata necessaria per garantire la possibilità che un agente possa tornare in ufficio dopo aver visitato un cliente, mantenendo un percorso privo di incroci e facilmente ripercorribile.

Quello che viene di fatto modellato dalle celle con **id > NC** è:

**id = NC+1** Indirizzo di partenza fittizio a distanza 0 da tutti gli altri nodi;

**id > NC+1** Percorso dal cliente id-NC all'ufficio.



# Modello di risoluzione basato su Programmazione Intera Quadratica

## Vincoli

I 25 vincoli sono raggruppabili in 7 categorie:

- **Vincoli di percorso**

- Garantiscono che tutti i clienti siano serviti, che tutti gli agenti inizino e terminino il loro percorso dalla posizione fittizia, che i percorsi siano continui, che non vi siano loop su stesso cliente, che non vi siano percorsi insensati.


- **Vincoli relativi al tempo di servizio**

- Garantiscono che la somma dei tempi di servizio a un cliente equivalga al tempo richiesto, che la somma dei tempi di servizio in ufficio equivalga al tempo minimo da spendere in ufficio, che non si spenda tempo in posizioni non nel percorso dell'agente.

- **Vincoli di ufficio**

- Garantiscono che ogni agente vada almeno una volta in ufficio, che un agente non vada in ufficio dall'ufficio, che un agente visiti la posizione relativa all'ufficio di partenza solo se "parte" da lì, che un agente visiti l'ufficio fittizio solo se ha visitato il corrispettivo cliente e che lo faccia appena dopo averlo visitato.





# Modello di risoluzione basato su Programmazione Intera Quadratica

- **Vincoli legati alle finestre temporali**
  - Garantiscono che i clienti siano serviti dopo che le loro finestre temporali si sono aperte e prima che si siano chiuse.
- **Vincoli legati al pranzo**
  - Garantiscono che l'agente pranzi dopo la finestra relativa al pranzo si apre e prima della fine di tale finestra, che se l'agente pranza nel percorso tra due posizioni, effettivamente lui percorra tale percorso, che l'agente abbia il pranzo solo se la sua giornata lavorativa è sufficientemente lunga da richiederlo.
- **Vincolo di durata del servizio**
  - Calcola la durata del tempo di servizio come somma del tempo di servizio alla posizione precedente, più la durata di tale servizio, più eventuale pranzo, più eventuale attesa.
- **Vincolo di giornata lavorativa**
  - Garantisce che la durata della giornata lavorativa non superi la durata massima, tale giornata include la somma del tempo speso viaggiando, servendo i clienti, aspettando, mangiando.



# Modello di risoluzione basato su Programmazione Intermedia Quadratica

## Funzione obiettivo

La funzione obiettivo va a minimizzare i percorsi fatti dagli agenti, i tempi di attesa e i tempi di servizio. Il minimizzare i tempi di servizio garantisce di avere le attese adiacenti al cliente che le causa.

La funzione obiettivo fa uso di due moltiplicatori che consentono di favorire maggiormente le attese o i viaggi.

$$\min \text{travel\_cost} * \sum_{i \in \text{ALL}} \sum_{j \in \text{ALL}} \sum_{a \in A} x_{i,j,a} * \text{distance}_{i,j} + \text{wait\_cost} * \sum_{i \in C} \sum_{a \in A} w_{i,a} + \sum_{i \in \text{RBLE}} \sum_{a \in A} s_{i,a}$$

- 1)  $\sum_{i \in \text{ALL}} \sum_{a \in A} x_{i,j,a} = 1, \quad j \in C$
- 2)  $\sum_{i \in \text{ALL}} \sum_{a \in A} x_{i,j,a} \leq 1, \quad j \in \text{FO}$
- 3)  $\sum_{j \in \text{REAL}} x_{\text{start},j,a} = 1, \quad a \in A$
- 4)  $x_{\text{start},j,a} = 0, \quad j \in \text{FO}, a \in A$
- 5)  $\sum_{i \in \text{RBLE}} x_{i,s,a} = 1, \quad a \in A$
- 6)  $\sum_{i \in \text{ALL}} x_{i,h,a} - \sum_{j \in \text{ALL}} x_{h,j,a} = 0, \quad h \in \text{ALL}, a \in A$
- 7)  $x_{i,i,a} = 0, \quad i \in \text{ALL}, a \in A$
- 8)  $c_{j,a} = \text{service}_j * \sum_{i \in \text{ALL}} x_{i,j,a}, \quad i \in C, a \in A$
- 9)  $\sum_{i \in \text{ALL}} \sum_{j \in o+FO} c_{j,a} * x_{i,j,a} \geq \text{service}_o, \quad a \in A$
- 10)  $c_{j,a} \geq x_{i,j,a}, \quad i \in \text{ALL}, j \in \text{RBLE}, a \in A$
- 11)  $\sum_{i \in \text{ALL}} \sum_{i \in o+FO} x_{i,j,a} \geq 1, \quad a \in A$
- 12)  $x_{i,j,a} = 0, \quad i \in o+FO, j \in o+FO, a \in A$
- 13)  $x_{i,o,a} = 0, \quad i \in C, a \in A$
- 14)  $x_{j,j+\text{clients},a} \leq \sum_{i \in \text{ALL}} x_{i,j,a}, \quad j \in C, a \in A$
- 15)  $\{x_{i,j,a} = 0 \text{ if } i \neq j - \text{clients}\}, \quad i \in \text{REAL}, j \in \text{FO}, a \in A$
- 16)  $\text{window\_start}_i * \sum_{h \in \text{ALL}} x_{h,i,a} \leq s_{i,a}, \quad i \in C, a \in A$
- 17)  $\text{window\_end}_i * \sum_{h \in \text{ALL}} x_{h,i,a} \geq s_{i,a}, \quad i \in C, a \in A$
- 18)  $x_{i,j,a} * (s_{i,a} + c_{i,a} + \text{distance}_{i,j} + w_{j,a}) \geq l_{i,j,a} * \text{lunch\_start},$   
 $i \in \text{RBLE}, j \in \text{RBLE}, a \in A$
- 19)  $l_{i,j,a} * (s_{i,a} + c_{i,a} + \text{distance}_{i,j} + w_{j,a}) \leq x_{i,j,a} * \text{lunch\_end},$   
 $i \in \text{RBLE}, j \in \text{RBLE}, a \in A$
- 20)  $l_{i,j,a} \leq x_{i,j,a} * \text{lunch\_end}, \quad i \in \text{RBLE}, j \in \text{RBLE}, a \in A$
- 21)  $t_a = \max(s_{i,a}, i \in \text{RBLE}), \quad a \in A$
- 22)  $\text{work\_end} * \sum_{i \in \text{RBLE}} \sum_{j \in \text{RBLE}} l_{i,j,a} \geq t_a - \text{lunch\_start}, \quad a \in A$
- 23)  $\text{work\_end} * \sum_{i \in \text{RBLE}} \sum_{j \in \text{RBLE}} (1 - l_{i,j,a}) \geq \text{lunch\_start} - t_a, \quad a \in A$
- 24)  $s_{j,a} = \sum_{i \in \text{RBLE}, i \neq j} x_{i,j,a} * (s_{i,a} + c_{i,a} +$   
 $\text{distance}_{i,j} + l_{i,j,a} * \text{lunch\_len}) + w_{j,a}, \quad j \in \text{RBLE}, a \in A$
- 25)  $\sum_{i \in \text{RBLE}} \sum_{j \in \text{RBLE}} x_{i,j,a} * \text{distance}_{i,j} +$   
 $\sum_{i \in \text{ALL}} \sum_{j \in \text{RBLE}} x_{i,j,a} * \text{service}_j +$   
 $\sum_{i \in \text{RBLE}} w_{i,a} +$   
 $\sum_{i \in \text{RBLE}} \sum_{j \in \text{RBLE}} l_{i,j,a} * \text{lunch\_len} +$   
 $\text{work\_start} \leq \text{work\_end}, \quad a \in A$

$$x_{i,j,a} \in \{0,1\}, i \in \text{ALL}, j \in \text{ALL}, a \in A$$

$$y_{i,a} \in N, i \in \text{ALL}, a \in A$$

$$c_{i,a} \in N, i \in \text{ALL}, a \in A$$

$$w_{i,a} \in N, i \in \text{ALL}, a \in A$$

$$l_{i,j,a} \in \{0,1\}, i \in \text{ALL}, j \in \text{ALL}, a \in A$$

$$t_a \in N, a \in A$$

$$\text{start} = \text{clients} + 1$$

$$o = 0$$

$$C = \{1, \dots, \text{clients}\}$$

$$\text{ALL} = \{0, \dots, 2\text{clients} + 1\}$$

$$\text{REAL} = \{0, \dots, \text{clients}\}$$

$$\text{RBLE} = \text{ALL} - \{s\}$$

$$\text{FO} = \{P + 2, \dots, 2\text{clients} + 1\}$$



# Modello di risoluzione basato su Set-cover Euristico

In questo modello viene definito un primo modello di programmazione lineare che risolve un problema di set cover, associando una variabile intera a un sottoinsieme di possibili schedule, e risolve il problema rispettando i vincoli precedentemente descritti.

Data una qualsiasi soluzione si cercano possibili nuove schedule cercandole nel vicinato  $N(s)$ , effettuando lo swap dei nodi della prima soluzione. Le schedule vengono generate a partire dalla sequenza di visite di un agente, e vengono poi passate al modello di programmazione lineare per una nuova soluzione.

Si itera fino a convergenza.



# Modello di risoluzione basato su Set-cover Euristico

**Modello di programmazione lineare intera per la risoluzione del problema di set-cover**

## Variabili

Si usa una sola variabile binaria  $x_s \in \{0, 1\}$  associata a ogni possibile schedula ammissibile.

## Vincoli

- Ogni agente  $a \in A$  ha una in una sola schedula associata tra quelle selezionate,
- Ogni meet  $m \in M$  deve essere presente in una e una sola schedula tra quelle selezionate.

## Obiettivo

Minimizzare i costi di tutti gli agenti. Il costo è dato dai tempi di viaggio e dai tempi di attesa.

# Modello di risoluzione basato su Set-cover Euristico

$$\min \sum_{a \in A} \sum_{s_a \in S_a} c_{s_a} x_{s_a}$$

$$\text{s. t. } \sum_{s_a \in S_a} x_{s_a} = 1 \quad \forall a \in A$$

$$\sum_{m \in s_a} x_{s_a} = 1 \quad \forall m \in M$$

$$x_{s_a} \in \{0, 1\} \quad \forall a \in A, s_a \in S_a$$



# Modello di risoluzione basato su Set-cover Euristico

## Generazione delle schedule

Il numero di variabili  $x_s$  cresce esponenzialmente all'aumentare dei meet, rendendo così computazionalmente impensabile valutare tutte le possibilità. Si genera un sottoinsieme di schedule a partire da una data configurazione che verrà passato al modello per la risoluzione del set covering.

Data una soluzione che associa a ogni agente un percorso come sequenza di meet, vengono generati possibili percorsi scambiando due nodi tra loro, per tutti i nodi di tutti gli agenti. Ogni agente ha quindi associata una lista di valori interi e, per avere più schedule tra cui scegliere e variare la dimensione delle liste, vengono generate anche schedule di dimensione  **$len(agente) + 1$**  e  **$len(agente) - 1$** , semplicemente rimuovendo un nodo dalla lista o aggiungendo un nodo della lista dell'agente successivo.

# Modello di risoluzione basato su Set-cover Euristico

## Generazione delle schedule

Agente 1 = [1, 3, 4]

Agente 2 = [2, 5, 7]

Agente 3 = [ ]



[1, 3, 4, 2, 5, 7]

[3, 1, 4, 2, 5, 7]

[4, 3, 1, 2, 5, 7]

[2, 3, 4, 1, 5, 7]

[5, 3, 4, 2, 1, 7]

[7, 3, 4, 2, 5, 1]

[1, 4, 3, 2, 5, 7]

[1, 2, 4, 3, 5, 7]

...

Si mettono le liste in  
sequenza e si procede  
allo swap.

Le liste degli agenti vengono ricostruite riprendendo  $L(a)$  nodi dalla lista, e vengono ricostruite anche le varianti  $L(a)+1$  e  $L(a)-1$ . In tutto vengono generate  $3N^2/2$  schedule, con  $N$  = numero meet.





# Modello di risoluzione basato su Set-cover Euristico

## Generazione delle schedule

Ad esempio, con la sequenza generata [3, 1, 4, 2, 5, 7] generiamo le seguenti varianti per gli agenti:

**L(a)**

Agente 1 = [3, 1, 4]

Agente 2 = [2, 5, 7]

Agente 3 = [ ]

**L(a) - 1**

Agente 1 = [3, 1]

Agente 2 = [4, 2]

Agente 3 = [5, 7]

**L(a) +1**

Agente 1 = [3, 1, 4, 2]

Agente 2 = [5, 7]

Agente 3 = [ ]



# Modello di risoluzione basato su Set-cover Euristico

## Generazione delle schedule

Per ogni sequenza di meet generata viene generata una schedula corrispondente.

Data una sequenza dei meet che un agente deve visitare, si “incastrano” i ritorni in ufficio e i pranzi, verificando che vengano rispettate le seguenti condizioni:

- Mezz'ora di pranzo in un dato range di tempo
- C'è abbastanza tempo tra un meet e l'altro per tornare in ufficio e lavorare almeno un'ora.

Per la seconda condizione si considerano anche i tempi di viaggio per tornare in ufficio da un meet e il tempo del viaggio per andare dall'ufficio al prossimo meet.



# Modello di risoluzione basato su Set-cover Euristico

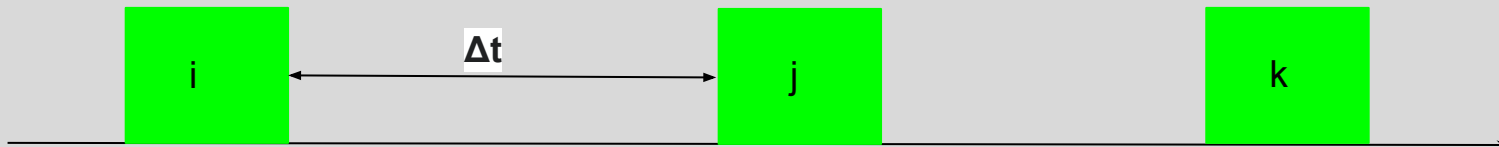
## **Generazione delle schedule**

Vengono scartate le schedule non ammissibili. Una schedula non è ammissibile se due meet hanno orari di visita tali per cui il meet che viene prima (in sequenza) ha un orario di visita antecedente al successivo.

Per queste schedule vengono ridotti i tempi di wait, spostando i meet lungo le loro finestre temporali e cercando di sfruttare questo tempo per il lavoro in ufficio.

# Generazione di schedule data la sequenza di meet

Per una generica sequenza di nodi di un agente

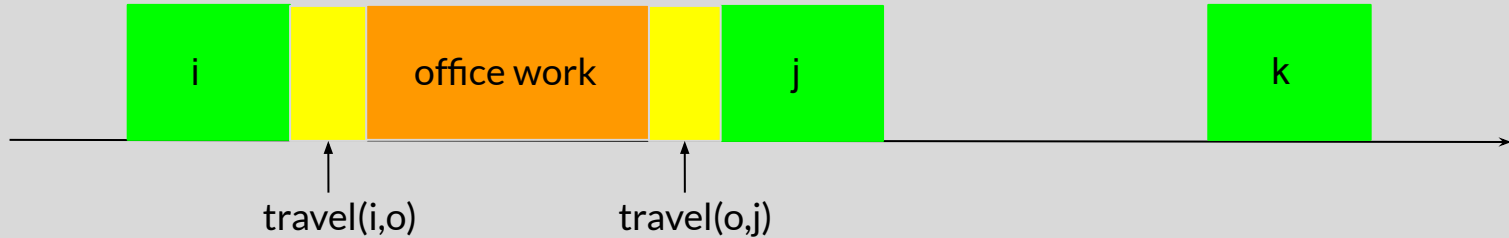


Tra i generici nodi  $i$  e  $j$ , ognuno con il proprio tempo di arrivo, devono essere piazzate alcune attività tra travel, office, wait, e lunch.

Se  $\Delta t \geq \text{travel}(i,o) + \text{min\_office\_time\_work} + \text{travel}(o,j)$  l'agente torna in ufficio, altrimenti va direttamente dal meet  $i$  al meet  $j$  e attende l'inizio dell'incontro.

# Generazione di schedule data la sequenza di meet

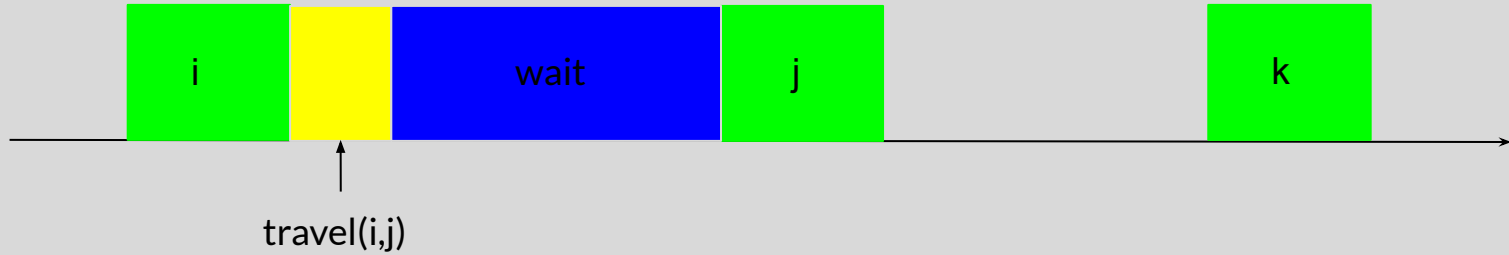
Caso in cui c'è abbastanza tempo per tornare in ufficio



$$\text{office\_work} = \Delta t - \text{travel}(i,o) - \text{travel}(o,j) \geq \text{min\_office\_time\_work}$$

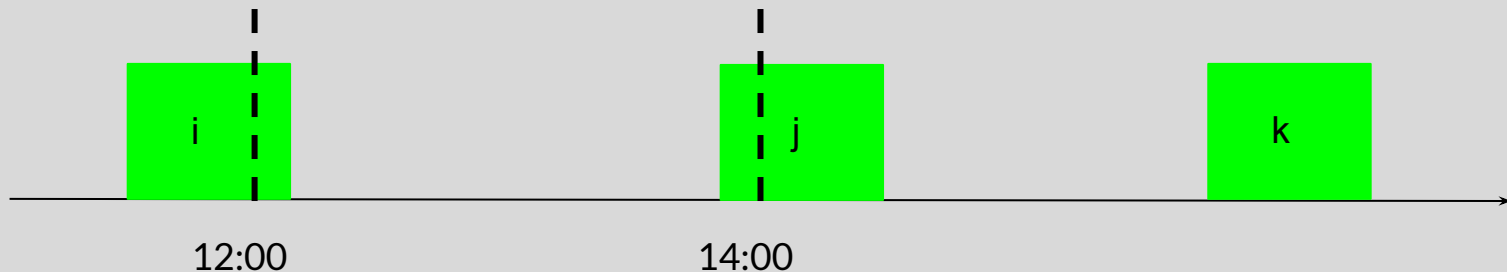
# Generazione di schedule data la sequenza di meet

Caso in cui non c'è abbastanza tempo per tornare in ufficio



# Generazione di schedule data la sequenza di meet

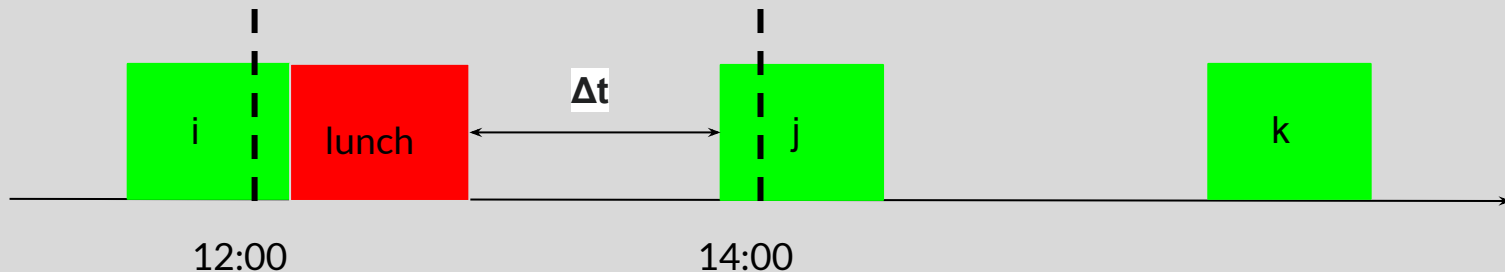
## Gestione del pranzo



Tra le 12:00 e le 14:00 deve esserci un periodo di mezz'ora per il pranzo. Se si cade in questo periodo viene aggiunto il pranzo di mezz'ora verificando che non sia già stato fatto e ci sia abbastanza tempo per raggiungere in tempo il meet successivo, eventualmente tornando anche in ufficio se le condizioni lo permettono.

# Generazione di schedule data la sequenza di meet

## Gestione del pranzo



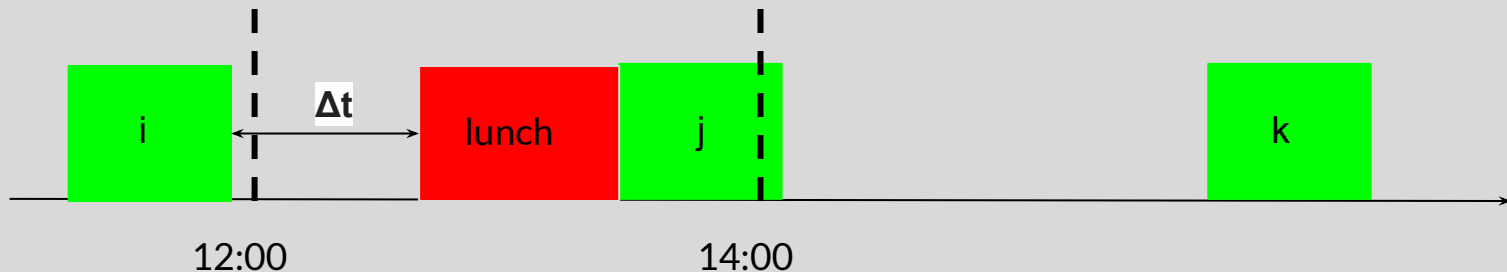
Se  $\Delta t \geq \text{travel}(i,o) + \text{min\_office\_time\_work} + \text{travel}(o,j)$  l'agente torna in ufficio, altrimenti va direttamente dal meet i al meet j e attende l'inizio dell'incontro.

Se non dovesse esserci abbastanza tempo per raggiungere il prossimo meet, la schedula viene scartata poiché non è possibile inserire il pranzo.



# Generazione di schedule data la sequenza di meet

## Gestione del pranzo

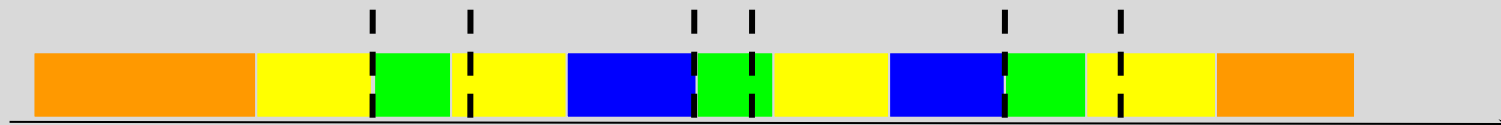


Se  $\Delta t \geq \text{travel}(i,o) + \text{min\_office\_time\_work} + \text{travel}(o,j)$  l'agente torna in ufficio, altrimenti va direttamente dal meet i al meet j e attende l'inizio dell'incontro.

Se non dovesse esserci abbastanza tempo per raggiungere il prossimo meet, la schedula viene scartata poiché non è possibile inserire il pranzo.

# Generazione di schedule data la sequenza di meet

## Gestione delle time window



● Office

● Travel

● Wait

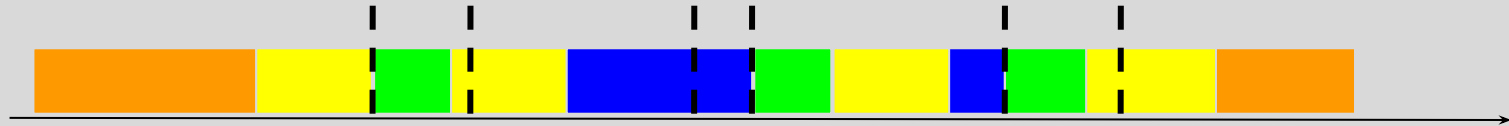
● Meet

Ogni meet  $i$  è stato piazzato allo  $\text{start\_time}_i$  della sua finestra temporale. Dopo la creazione di una schedula si cerca di minimizzare i  $\text{wait\_time}$  sfruttando le finestre temporali.

Poiché il tempo di ogni meet e travel è fisso, il tempo guadagnato deve essere speso in ufficio.

# Generazione di schedule data la sequenza di meet

## Gestione delle time window



● Office

● Travel

● Wait

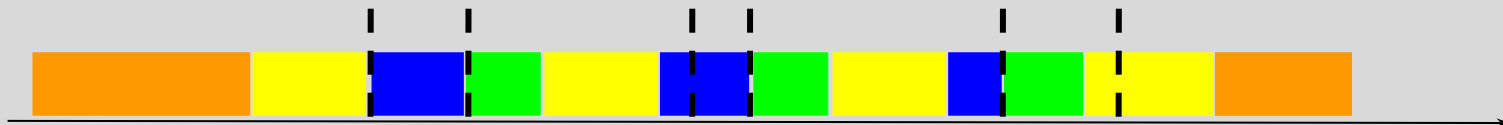
● Meet

Per ogni schedula vengono prese le porzioni di azioni comprese tra due office.

Partendo poi dall'ultimo wait si trasla il meet precedente di  $\min(time\_win_i, wait\_time_j)$ .

# Generazione di schedule data la sequenza di meet

## Gestione delle time window



● Office

● Travel

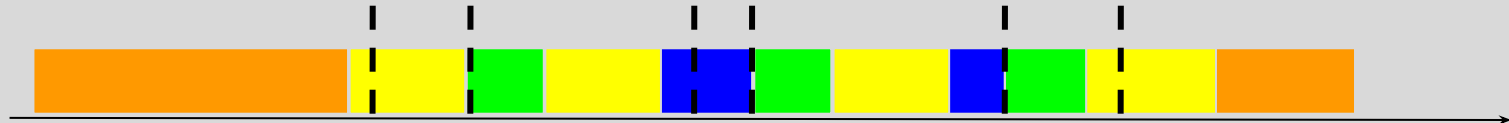
● Wait

● Meet

Per ogni schedula viene presa la porzione di azioni compresa tra due office.  
L'operazione viene fatta per tutti i wait a partire dalla fine all'inizio, spostando i  
meet di  $\min(time\_win_i, wait\_time_i)$ .

# Generazione di schedule data la sequenza di meet

Gestione delle time window



● Office

● Travel

● Wait

● Meet

I tempi di wait vengono ridotti e si sfrutta il tempo guadagnato in ufficio.



# Modello di risoluzione basato su Set-cover Euristico

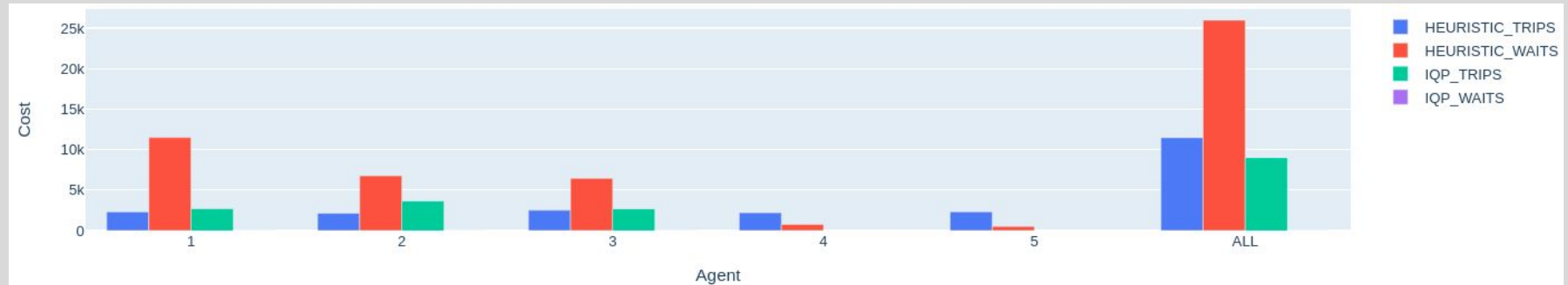
## Modello finale

- Tutte le schedule generate vengono passate al modello di programmazione lineare che sceglierà un sottoinsieme di schedule valido e per le quali è dato un costo. Si minimizzano i tempi di attesa e i tempi di viaggio.
- Dalla soluzione ottenuta si ottengono nuove schedule da valutare e si continua finché si riesce a minimizzare il costo.
- La prima soluzione da cui si ottengono le schedule è data da un algoritmo greedy che, per ogni agente, assegna a quest'ultimo il maggior numero di meet per la sua giornata.

# Confronto tra approccio di Programmazione Intera Quadratica e Set-cover euristico

## Costi

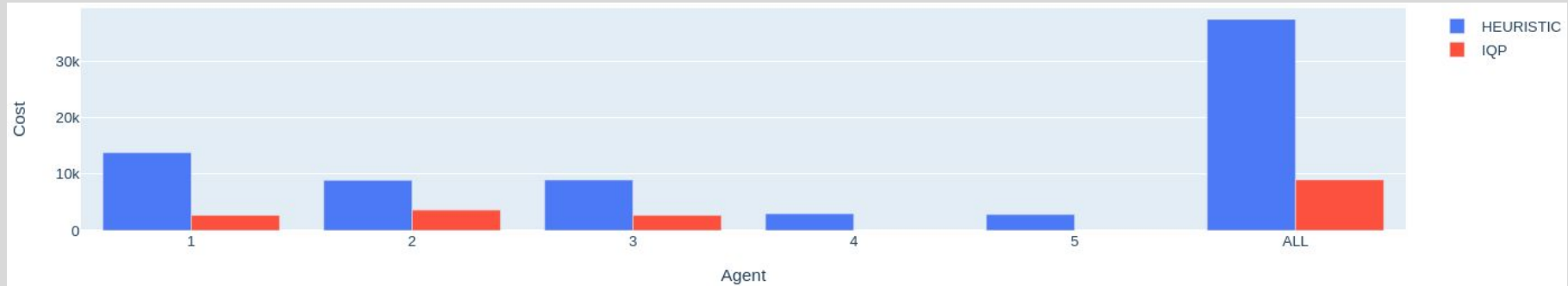
Confrontando i risultati dei due approcci è possibile osservare come l'approccio basato sulla programmazione intera dia dei risultati più precisi, riducendo al minimo i costi dei percorsi e ove possibile azzerando le attese.



Dati: Test\_2.txt

# Confronto tra approccio di Programmazione Intera Quadratica e Set-cover euristico

Di conseguenza i costi ridotti risultanti sono notevolmente più bassi.



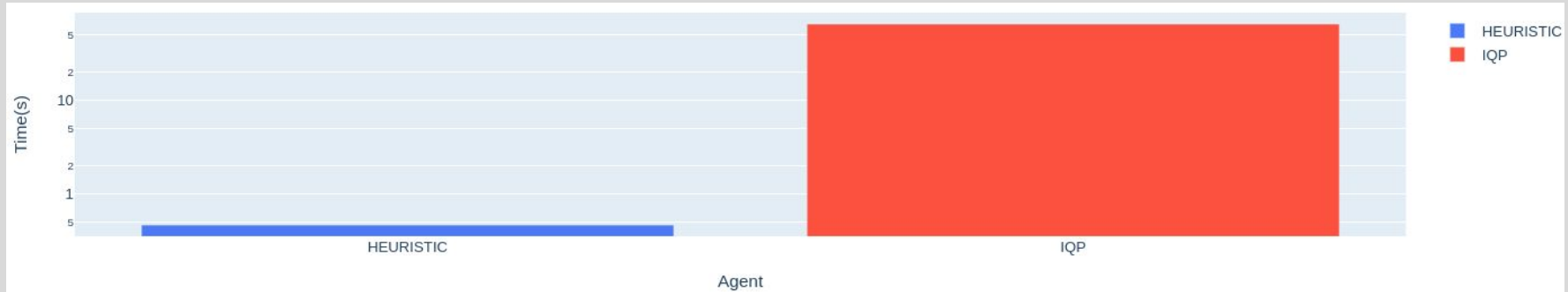
Dati: Test\_2.txt



# Confronto tra approccio di Programmazione Intera Quadratica e Set-cover euristico

## Tempi

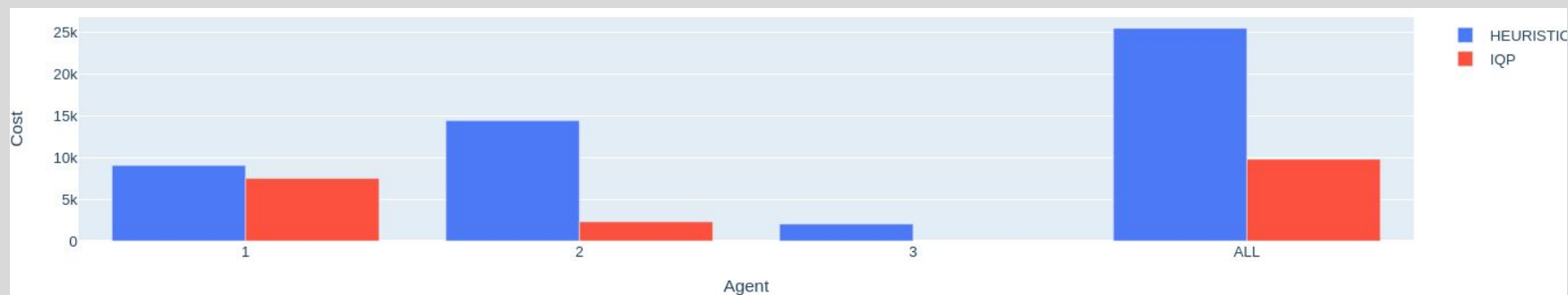
I tempi di attesa al contrario risultano essere **più bassi anche di quattro ordini di grandezza** per il modello euristico.



Dati: Test\_2.txt

# Confronto tra approccio di Programmazione Intera Quadratica e Set-cover euristico

Dati: Test1.txt  
Costi

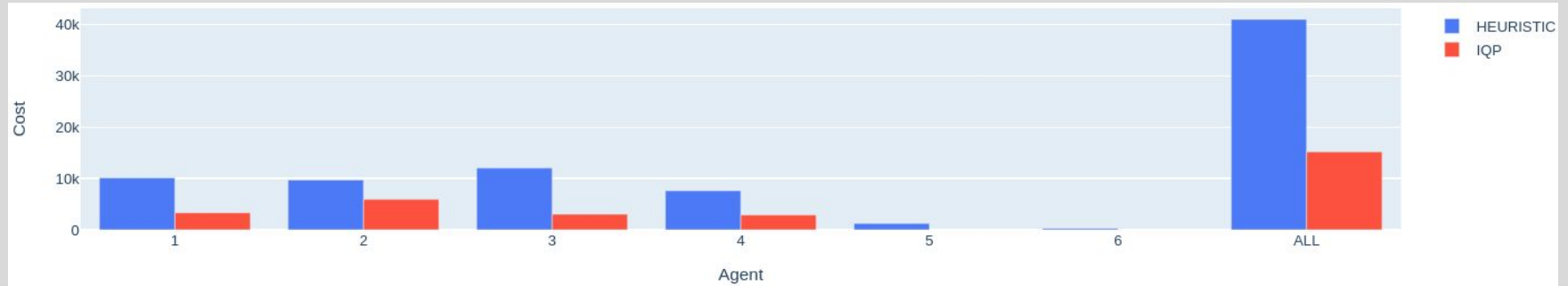


Tempi



# Confronto tra approccio di Programmazione Intera Quadratica e Set-cover euristico

Dati: Test3.txt  
Costi



Tempi

