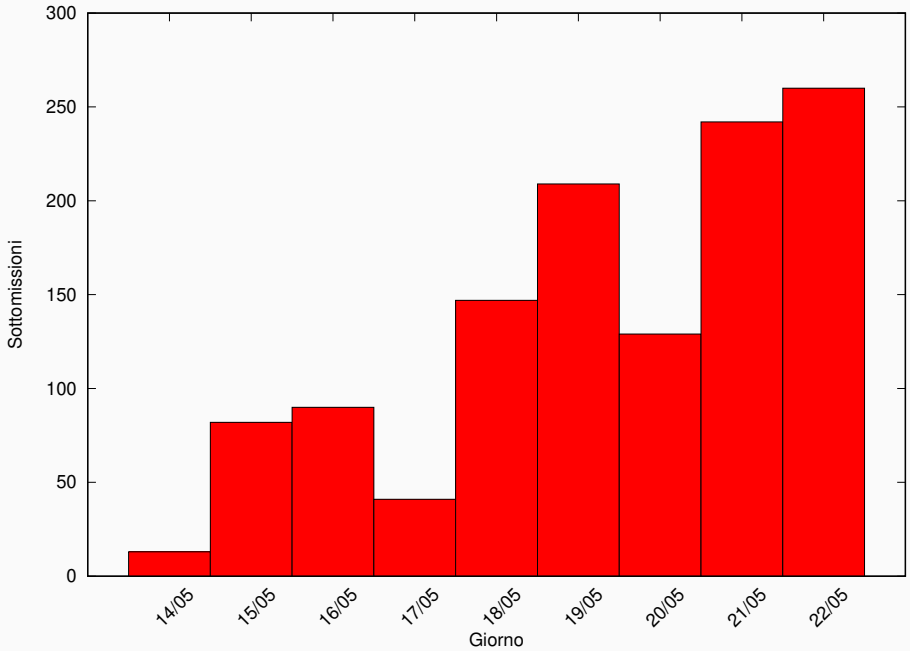
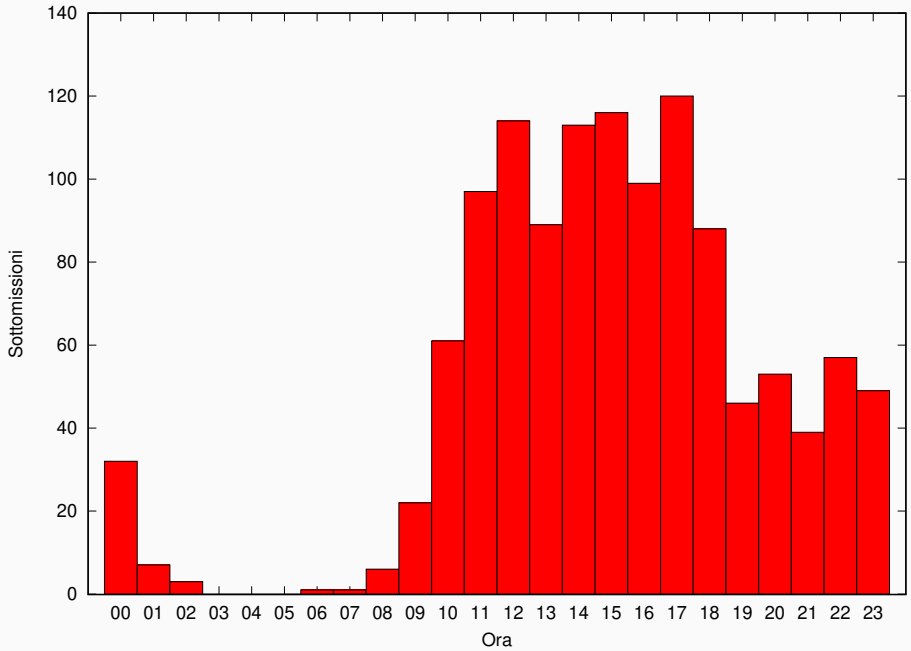
# Secondo Progetto ASD 2025/2026

# Secondo Progetto ASD 2025/2026

# Statistiche

Numero sottoposizioni: 1204.




- 58 gruppi iscritti, di cui 57 hanno fatto almeno una sottoposizione e 55 hanno raggiunto la sufficienza;
- 138 studenti iscritti, di cui 137 appartenenti a gruppi che hanno fatto almeno una sottoposizione.

# Risultati

## Punteggi

- $P < 30 \longrightarrow$ progetto non passato
- $30 \leq P < 58 \longrightarrow$ 1 punto bonus (8 gruppi)
- $58 \leq P < 91 \longrightarrow$ 2 punti bonus (12 gruppi)
- $91 \leq P \leq 100 \longrightarrow$ 3 punti bonus (36 gruppi)

Classifiche e sorgenti sul sito (controllate i numeri di matricola):
`https://asdlab.disi.unitn.it/asd25/classifica_prog2.pdf`

# Problema (I)

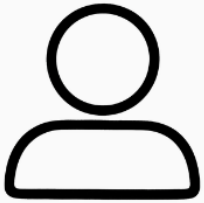
Dati:

- un insieme di $S$ skill;
- un insieme di $P$ persone: ogni persona $p$ ha una capacità $cap_p$ (ore settimanali), un insieme di skill $skills(p)$ e un valore di coesione $w_{pq}$ con ogni altra persona $q$;
- un insieme di $J$ progetti: ogni progetto $j$ richiede un numero di ore $dem_j$ e un insieme di skill $req(j)$;
- un intero $K$: numero massimo di progetti a cui una persona può partecipare.

# Problema (I) - esempio

**P = 3, J = 3, S = 3, K = 2**

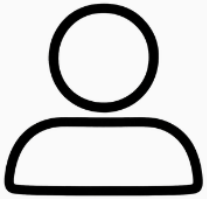
**Persona 0**



40 ore

skills: 0 (python), 2 (c++)
coesioni: 0, 3, -1

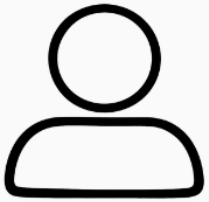
**Persona 1**



24 ore

skills: 1 (git), 2 (c++)
coesioni: -2, 0, 2

**Persona 2**



32 ore

skills: 0 (python)
coesioni: -1, 5, 0

**Progetto 0**

42 ore

skills richieste:
0, 1, 2

**Progetto 1**

20 ore

skills richieste:
1, 2

**Progetto 2**

30 ore

skills richieste:
0, 2

# Problema (II)

Dette $x_{p,j} \geq 0$ le ore di $p$ allocate sul progetto $j$, con $x_{p,j} = 0$ se $p$ non partecipa a $j$, desideriamo trovare un assegnamento tale che:

- ogni persona partecipi ad al più $K$ progetti:

$$|\{j : x_{p,j} > 0\}| \leq K \quad \forall\, p;$$

- ogni persona non superi la propria capacità:

$$\sum_{j=0}^{J-1} x_{p,j} \leq cap_p \quad \forall\, p;$$

- il tempo allocato su ogni progetto non superi la domanda:

$$\sum_{p=0}^{P-1} x_{p,j} \leq dem_j \quad \forall\, j;$$

- ogni progetto abbia tutte le skill richieste coperte:

$$\forall\, j,\ \forall\, s \in req(j),\ \exists\, p : x_{p,j} > 0 \text{ e } s \in skills(p);$$

# Problema (III)

E, soddisfatti i requisiti precedenti, desideriamo trovare l'assegnamento che minimizzi:

$$\texttt{score} = \alpha \cdot \texttt{total\_unmet} - \beta \cdot \texttt{total\_cohes}.$$

**Pesi:** $\alpha = 7$, $\beta = 3$.

**Tempo non coperto:**

$$\texttt{total\_unmet} = \sum_{j=0}^{J-1} u_j \qquad u_j = \begin{cases} dem_j & \text{se le skill del progetto } j \text{ non sono tutte coperte} \\ \text{tempo non coperto nel progetto } j & \text{altrimenti} \end{cases}$$

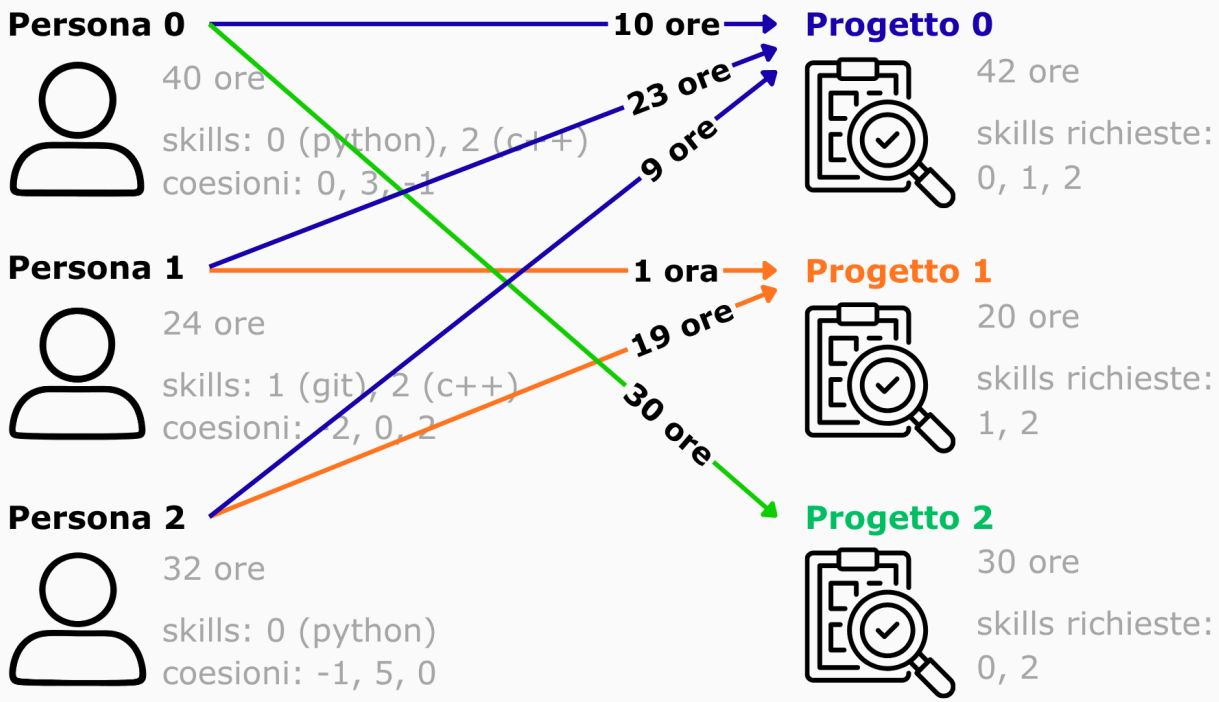
**Coesione totale:**

$$\texttt{total\_cohes} = \left\lfloor \frac{1}{2P} \sum_{j=0}^{J-1} \sum_{p,q=0}^{P-1} y_{j,p} y_{j,q} w^*_{j,pq} \right\rfloor \qquad w^*_{j,pq} = \begin{cases} -|w_{pq}| & \text{se le skill di } j \text{ non sono tutte coperte} \\ w_{pq} & \text{altrimenti} \end{cases}$$

$y_{j,p}$ è 1 se $p$ è assegnato al progetto $j$, 0 altrimenti. Vale sempre che $w_{pp} = 0$.

# Problema (III) - esempio

Nell' esempio precedente, una soluzione valida è la seguente:




In questo caso, i valori $x_{p,j}$ sono:

$$x_{0,0} = 10, x_{0,1} = 0, x_{0,2} = 30$$

$$x_{1,0} = 23, x_{1,1} = 1, x_{1,2} = 0$$

$$x_{2,0} = 9, x_{2,1} = 19, x_{2,2} = 0$$

# Il problema è NP-completo (I)

Set Cover, a sua volta NP-completo, può essere ridotto al nostro problema.

**Set Cover**

Dato un universo $U$ e una famiglia $\mathcal{S} = \{S_1, \ldots, S_m\}$ di sottoinsiemi di $U$, un *set cover* è una sottofamiglia $C \subseteq \mathcal{S}$ tale che $\bigcup_{S \in C} S = U$.

Dati $U$, $\mathcal{S}$ e un intero $k$, esiste un set cover $C \subseteq \mathcal{S}$ con $|C| \leq k$?

**Riduzione:**

- $m$ persone $p_1, \ldots, p_m$, una per ogni $S_i \in \mathcal{S}$:
  - $skills(p_i) = S_i$
  - $cap_i = 1$
- un progetto $j^*$:
  - $req(j^*) = U$
  - $dem_{j^*} = 1$
- $m - k$ progetti "dummy", t.c. per ogni progetto $j$:
  - $\text{req}(j) = \emptyset$
  - $dem_j = 1$
- $K = 1$, $w_{pq} = 0$ sempre.

# Il problema è NP-completo (II)

Se trovo la soluzione in tempo polinomiale al nostro problema, riesco a trovarla anche all'altro.

**Idea:** essendo $K = 1$, ogni persona potrà lavorare su al più un progetto. Delle $m$ persone, $m - k$ dovranno essere assegnate ai progetti "dummy", lasciando al più $k$ persone disponibili per $j^*$. L'unione delle skills di queste $k$ persone dovrà essere l'insieme universo $U$. Quindi, se esiste una soluzione polinomiale al nostro problema, esiste una soluzione polinomiale anche a Set Cover.

La riduzione è **polinomiale**, quindi il nostro problema è **NP-completo**.

... E quindi necessita di soluzioni approssimate!

# Caso base

Consideriamo il caso in cui $K = J$, $w_{pq} = 0$ per ogni $p$ e $q$, e tutte le persone possiedono la *stessa* skill e tutti i progetti richiedono *solo* quella skill. Significa che non dobbiamo preoccuparci né della copertura delle skills, né della coesione delle persone, né del vincolo $K$. Ovvero: ci interessa solo distribuire bene i tempi.

# Soluzione caso base (I)

**Una possibile soluzione:** ordinare persone e progetti per ore disponibili/richieste in modo decrescente, poi assegnare in modo greedy.

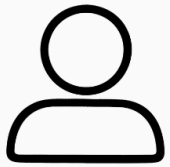
**Algoritmo:**

- ordina i progetti per $dem_j$ decrescente;
- ordina le persone per $cap_p$ decrescente;
- per ogni progetto $j$ (in ordine):
  - scorre le persone in ordine e assegna $x_{p,j} = \min(cap_p^{\text{res}}, dem_j^{\text{res}})$;
  - aggiorna $cap_p^{\text{res}}$ e $dem_j^{\text{res}}$ (valori residui);
  - si ferma quando $dem_j^{\text{res}} = 0$ o le persone sono esaurite.

# Soluzione caso base (II)

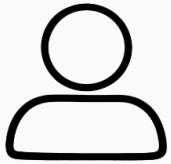
Consideriamo il nostro esempio, ma teniamo conto solo dei tempi:

**Persona 0**



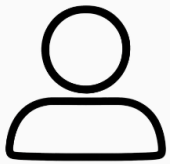
40 ore

**Persona 2**



32 ore

**Persona 1**



24 ore

**Progetto 0**

42 ore

**Progetto 2**

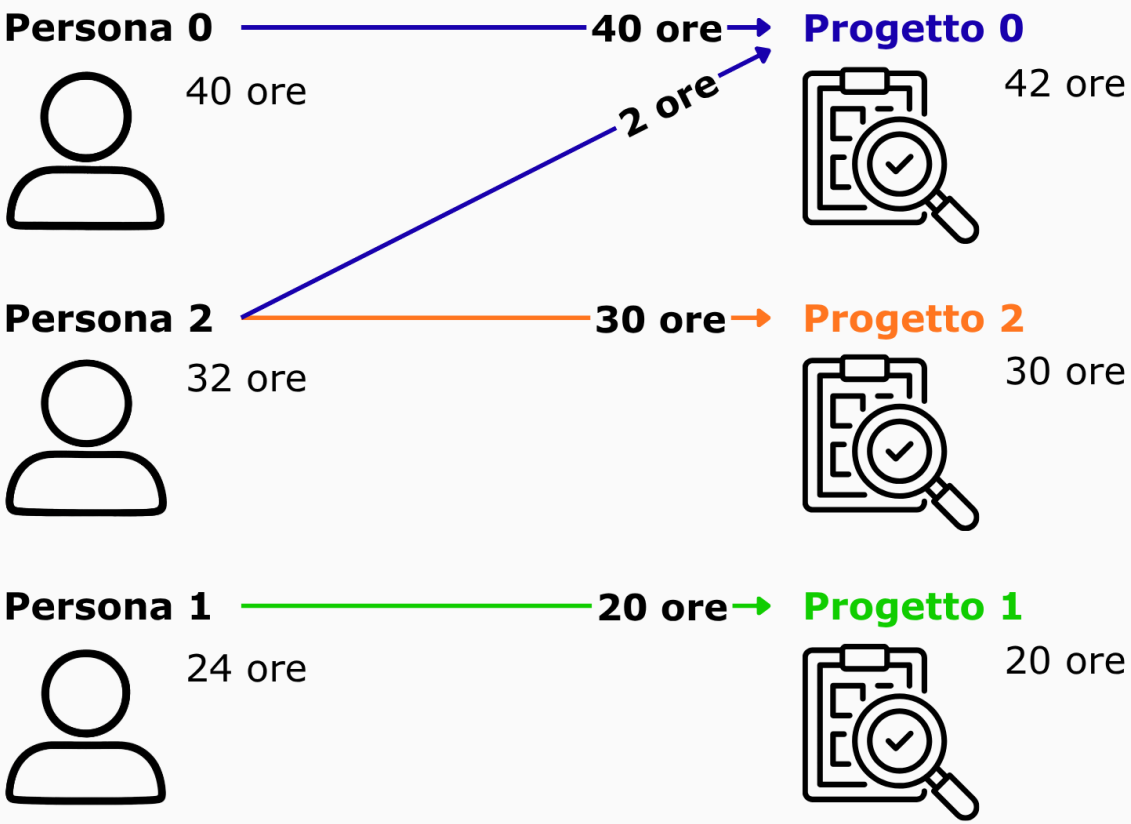
30 ore

**Progetto 1**

20 ore

# Soluzione caso base (III)

La soluzione base ci dà questo risultato:



# Gestione progetti non completabili

**Idea fondamentale:** se un progetto non riesce a coprire *tutte* le skill richieste, **conviene non assegnare alcuna persona** (assegnazione nulla).

**Confronto punteggi:**

- **Team parziale** (skill non coperte):
  - penalità domanda: $unmet_j = demand_j$ (massima, uguale al caso vuoto);
  - penalità coesione: ogni coppia $(p, q)$ nel team contribuisce con un valore **negativo** alla coesione totale.
- **Team vuoto** (0 persone):
  - penalità domanda: $unmet_j = demand_j$ (identica);
  - penalità coesione: 0 (nessuna coppia $\Rightarrow$ nessun contributo negativo).

**Conseguenza :**

- assegnare persone senza coprire tutte le skill **peggiora sempre** il punteggio finale.

# Soluzione intermedia 76 punti

**Strategia generale:** algoritmo greedy in due fasi con ordinamento per difficoltà.

**Idea chiave:**

- i progetti con skill *rare* sono il collo di bottiglia $\rightarrow$ gestiscili prima;
- prima soddisfa i vincoli **hard** (skill richieste), poi ottimizza i vincoli **soft** (coesione);
- se un progetto non può essere completato: *rollback* totale (nessuna assegnazione parziale).

**Struttura dell'algoritmo:**

1. Preprocessing: calcolo rarità skill e difficoltà progetti
2. Ordinamento progetti per difficoltà decrescente
3. Per ogni progetto: fase di copertura skill + fase di raffinamento coesione
4. Calcolo punteggio finale e output

# Preprocessing e Ordinamento

**Calcolo della rarità delle skill:**

- per ogni skill $s$: conta quante persone la possiedono $\rightarrow$ $rarity[s]$;
- skill più rara = più preziosa (meno persone disponibili).

**Difficoltà di un progetto $j$:**

$$diff_j = demand_j \times \left(1 + \sum_{s \in reqSkills_j} \frac{1}{rarity[s]}\right)$$

*(Se una skill non è posseduta da nessuno: peso $= 10$ per evitare divisioni per zero)*

**Ordinamento greedy:**

- ordina i progetti per $diff_j$ **decrescente**;

# Esempio: calcolo difficoltà e ordinamento

**Persona 0**

40 ore

skills: **0 (python)**, **2 (c++)**
coesioni: 0, 3, -1

**Persona 1**

24 ore

skills: **1 (git)**, **2 (c++)**
coesioni: -2, 0, 2

**Persona 2**

32 ore

skills: **0 (python)**
coesioni: -1, 5, 0

**Progetto 0**

42 ore

skills richieste:
0, 1, 2

**Progetto 1**

20 ore

skills richieste:
1, 2

**Progetto 2**

30 ore

skills richieste:
0, 2

**Rarità delle skills**

**Git: rarità 1**

**Python: rarità 2**

**C++: rarità 2**

**Difficoltà dei progetti**

**D(0) = 42 * (0.5+1+0.5) = 84**

**D(1) = 20 * (1+0.5) = 30**

**D(2) = 30 * (0.5+0.5) = 30**

# Esempio: calcolo difficoltà e ordinamento

**Rarità delle skill (quante persone le hanno):**

- $rarity[0]$ (python) $= 2$ (Persone 0 e 2)
- $rarity[1]$ (git) $= 1$ (Persona 1)
- $rarity[2]$ (c++) $= 2$ (Persone 0 e 1)

**Calcolo difficoltà** $diff_j = demand_j \times \left(1 + \sum_s \frac{1}{rarity[s]}\right)$**:**

- $diff_0 = 42 \times \left(\frac{1}{2} + \frac{1}{1} + \frac{1}{2}\right) = 42 \times 2 = \mathbf{84}$
- $diff_1 = 20 \times \left(\frac{1}{1} + \frac{1}{2}\right) = 20 \times 1.5 = \mathbf{30}$
- $diff_2 = 30 \times \left(\frac{1}{2} + \frac{1}{2}\right) = 30 \times 1 = \mathbf{30}$

**Ordinamento finale (decrescente per difficoltà):**

- Progetto 0 (84) $\rightarrow$ Progetto 1 (30) $\rightarrow$ Progetto 2 (30)

# Step 1: Copertura Skill Obbligatorie

**Obiettivo:** garantire che ogni skill richiesta dal progetto sia presente nel team.

**Per ogni progetto $j$ preso nell'ordine:**

- per ogni skill richiesta $s$ non ancora coperta:
  - cerca persona $p$ con: $s \in skills_p$, $cap_p^{res} > 0$, e $projCount_p < K$;
  - tra le candidate, scegli quella con **massima capacità residua**;
  - aggiungi $p$ al team, marca $s$ come coperta.
- **Validazione:** se alla fine qualche skill non è coperta $\rightarrow$ rollback (rimuovi tutte le assegnazioni temporanee per $j$).

# Step 2: Raffinamento e Coesione

**Obiettivo:** saturare la domanda residua migliorando la coesione del team.

**Algoritmo di raffinamento:**

- mentre $\sum_{p \in team} cap_p^{res} < demand_j$:
  - cerca persona $p \notin team$ con $cap_p^{res} > 0$ e $projCount_p < K$;
  - calcola il **guadagno di coesione**:

$$gain(p) = \sum_{q \in team} \left( cohesion[p][q] + cohesion[q][p] \right)$$

  - aggiungi al team la persona con $gain(p)$ massimo;
- se nessuna persona è disponibile $\rightarrow$ termina (progetto parzialmente soddisfatto).

# Come migliorare?

La soluzione greedy ottiene già un buon punteggio, ma ha dei limiti:

- le decisioni sono prese in modo quasi irreversibile;
- una persona con skill rara può essere usata troppo presto;
- il vincolo $K$ rende difficile correggere un assegnamento sbagliato;
- se un progetto resta senza anche una sola skill, viene considerato completamente non coperto;

Per migliorare la soluzione abbiamo aggiunto [1]:

- **randomized restart**, per provare molte costruzioni greedy diverse;
- **simulated annealing**, per uscire da ottimi locali;
- **repair mirato**, per recuperare progetti non coperti;
- **destroy & rebuild**, per ricostruire team problematici.

[1] tecniche standard in problemi di ottimizzazione

# Randomized restart e simulated annealing

**Randomized restart:**

- ripetiamo molte volte la costruzione greedy;
- aggiungiamo rumore casuale nell'ordinamento dei progetti e nella scelta dei candidati;
- ogni volta manteniamo la migliore soluzione trovata.

**Simulated annealing:**

- partendo da una soluzione greedy, proviamo modifiche locali;
- se una mossa migliora lo score, viene sempre accettata;
- se peggiora lo score, può comunque essere accettata con probabilità decrescente.

$$Pr(\text{accetta}) = \begin{cases} 1 & \text{se } \Delta score \leq 0, \\ e^{-\Delta score/T} & \text{se } \Delta score > 0. \end{cases}$$

$$T \leftarrow \lambda T, \qquad 0 < \lambda < 1.$$

# Mosse locali

Durante il simulated annealing vengono provate modifiche semplici alla soluzione corrente.

Quando aggiungiamo una persona $p$ al team $T_j$ del progetto $j$, stimiamo il guadagno marginale di coesione come:

$$gain(p, j) = \sum_{q \in T_j} (w_{pq} + w_{qp}) .$$

Le mosse locali principali sono:

- aggiungere o aumentare ore su un progetto;
- rimuovere o diminuire ore assegnate;
- spostare ore di una persona da un progetto a un altro;
- sostituire una persona in un team;
- spostare ore tra persone dello stesso progetto.

Ogni mossa deve rispettare la domanda dei progetti e il vincolo $K$.

# Repair dei progetti scoperti

**Osservazione:** se $u_j$ è il contributo del progetto $j$ a `total_unmet`, allora se il progetto non copre tutte le skill richieste si ha $u_j = dem_j$.

Quindi il progetto contribuisce come se fosse completamente non coperto, anche se alcune ore sono state assegnate.

**Aggressive repair:**

1. considera i progetti non coperti, dando priorità a quelli con domanda più alta;
2. calcola le skill mancanti;
3. cerca persone che coprano quelle skill;
4. se una persona è bloccata da capacità o da $K$, prova a liberarla da un altro progetto;
5. se il progetto diventa valido, prova a riempire la domanda residua.

Il repair è utile perché recuperare un progetto intero può ridurre molto `total_unmet`.

# Destroy & rebuild di un progetto

A volte le varie run di soluzioni greedy possono comunque produrre interi team costruiti male, e le mosse locali non sono abbastanza per recuperare.

Per tentare di sopperire a questo problema, posso:

1. scegliere a caso un progetto $j$ e rimuovere tutto il suo team;
2. liberare la capacità e slot $K$ delle persone rimosse;
3. ricostruire il progetto usando le risorse correnti (in maniera greedy);
4. accettare o rifiutare la nuova soluzione tramite simulated annealing.

Questa mossa esplora un intorno molto più grande rispetto alle modifiche locali. Devo trovare il modo di bilanciare (tentativi) le mosse piccole e grandi.

# Ricapitolando

La soluzione finale combina quindi più livelli di euristiche:

- **Greedy**: costruisce una soluzione iniziale;
- **Multi-start**: esplora molte costruzioni diverse;
- **Simulated annealing** / **local search**: migliora la soluzione con mosse locali;
- **Repair**: recupera progetti non coperti;
- **Rebuild**: ricostruisce completamente team casuali.

**Risultato:** circa 94.2 punti.