Un algoritmo di Min Cost Flow per l'assegnazione di risorse umane a veicoli GSE in un apron aeroportuale

A cura di

Fabio Bredo

Tesi di laurea in

Statistica e tecnologie informatiche

Università degli studi di Padova

Facoltà di Scienze Statistiche

A.A. 2008/2009

Università degli studi di Padova

Facoltà di Scienze Statistiche

SUNTO

Un algoritmo di Min Cost Flow per l'assegnazione di risorse umane a veicoli GSE in un apron aeroportuale

A cura di **Fabio Bredo**

Relatore **Professor Giovanni Andreatta**

Dipartimento: Matematica Pura e Applicata

La tesi è relativa all'applicazione di un algoritmo di Min Cost Flow per risolvere un problema di assegnazione ottimale di veicoli GSE a risorse umane in un apron aeroportuale.

E' divisa in tre parti:

1. una presentazione del contesto del problema,
2. descrizione dell'algoritmo di Min Cost Flow
3. una implementazione in C dell'algoritmo per trovare la soluzione ottimale del problema.

Sommario

Capitolo 1: progetto AAS

Il problema dell'assegnazione di veicoli GSE (Ground Service Equipment) a risorse umane nasce nell'ambito di un progetto europeo chiamato "Integrated Airport Apron Safety Fleet Management" (acronimo AAS)

Capitolo 1.1: obiettivo del progetto AAS

La vita in un apron aeroportuale è molto movimentata, ci sono molte squadre, spesso di società diverse, che lavorano nelle operazioni di gestione degli aerei, da quando atterrano a quando partono, queste lavorano in aree condivise dell'aeroporto e sono responsabili dell'uso dei veicoli ed equipaggiamenti di quella zona. Attualmente le squadre sono isolate fra loro, non c'è passaggio di informazioni, come ad esempio la localizzazione di un particolare GSE appena usato da un'altra squadra, costringendo così a ricerche di mezzi adatti all'operazione in corso da parte di una task e nel caso di incidente, è difficile anche solo individuare il responsabile (o la società). Gli incidenti e i tempi morti di ricerche di attrezzature nell'apron per l'attività rappresentano poi un costo aggiuntivo per gli aeroporti.

L'obiettivo primario del progetto è migliorare la sicurezza dell'apron aeroportuale con una più efficiente gestione delle squadre e dei veicoli GSE.

Il progetto AAS integrerà le migliori soluzioni per la distribuzione dei compiti a tutti gli utenti e per la gestione dei veicoli GSE, con un approccio efficiente e sicuro. Per esempio si cercherà di evitare possibili tempi morti, come cercare un veicolo GSE libero, con l'implementazione un sistema di monitoraggio basato su Wi-Fi e GPRS per monitorare costantemente le posizioni dei vari veicoli GSE, in modo da offrire alla squadra una scelta ottimale delle attrezzature necessarie.

Capitolo 1.2: obiettivi scientifici e tecnologici del progetto AAS

Il progetto AAS si focalizza su obiettivi scientifici e tecnologici :

- la comprensione delle interdipendenze tra tutte le organizzazioni che sono presenti nell'apron
- la riduzione dei rischi con lo sviluppo di prodotti per la gestione della sicurezza aeroportuale
- l'ottimizzazione delle comunicazioni fra le squadre
- la massimizzazione dell'uso di GSE con le informazioni in modo telematico (stato e posizione)
- l'ottimizzazione di percorsi dei veicoli e staff per incrementare la sicurezza
- la riduzione dei costi di riparazione
- l'acquisizione delle informazioni per pianificare in modo ottimale la gestione e gli investimenti

- l'incremento dei ricavi facendo girare le informazioni in modo automatico da un singolo veicolo GSE all'unità centrale al sistema di gestione.

La varietà delle persone e delle società che utilizzano l'apron, o che sono collegate alle attività di rampa, sono in aumento in numero e densità. Le compagnie sono isolate fra di loro, e questa cattiva gestione può portare alla possibilità di congestione dell'apron stesso e all'aumento del rischio di incidenti. Questi incidenti sono visti dalla compagnia aerea come dei costi aggiuntivi, con un'analisi di questi costi e delle scelte opportune, il progetto punta quindi a ridurre questi rischi di incidenti (e quindi i costi), migliorando allo stesso tempo la sicurezza all'interno dell'apron.

Capitolo 1.3: attività progetto AAS

Nella figura 1 sono illustrate tutte le principali attività del progetto AAS.

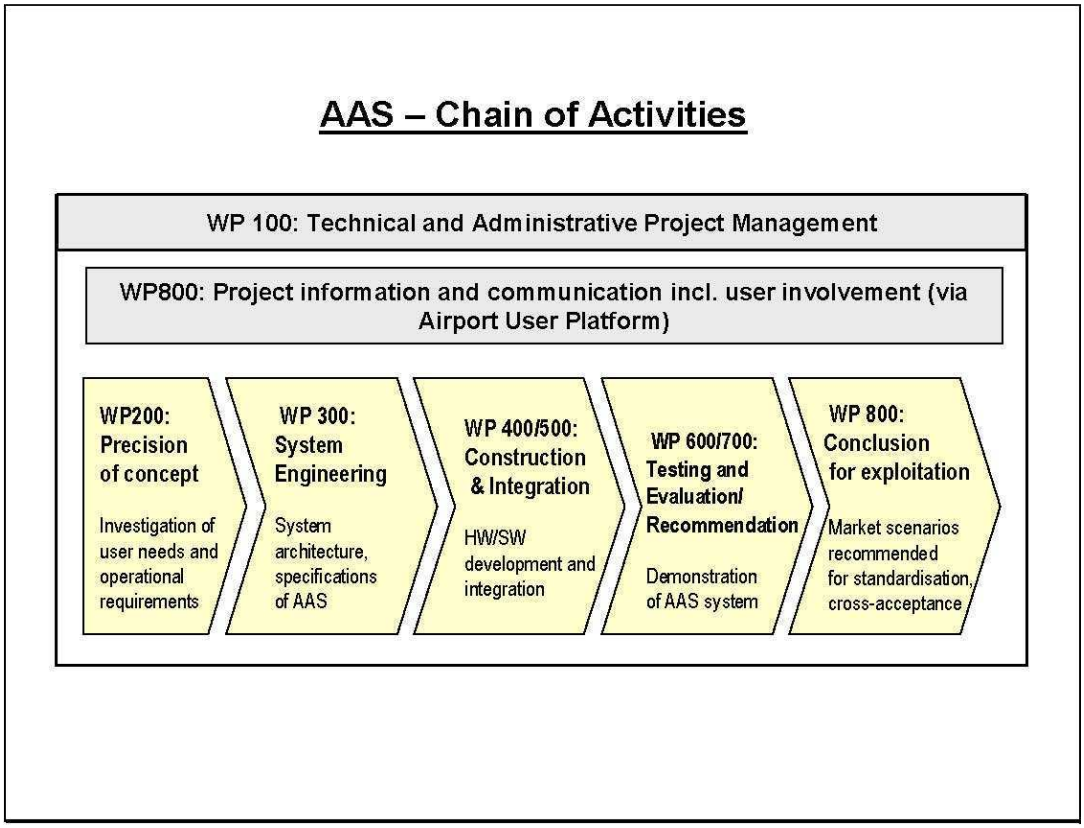

Figura 1

WP 100 - Gestione del progetto: riguarda la gestione tecnica, amministrativa e il coordinamento del progetto, nonché la gestione delle interfacce tra gli altri pacchetti di lavoro incluso la gestione del rischio e dei piani di emergenza.

WP 200 - Istruttoria delle esigenze: esigenze operative e degli utenti con incluso l'identificazione dei soggetti interessati.

WP 300 - Specifiche dell'architettura del sistema per garantire la conformità della soluzione AAS con gli standard aeroportuali.

WP 400 - Sviluppo di hard-/software riguarda la ricerca e lo sviluppo delle componenti AAS

WP 500 - Integrazione di hard-/software consiste di sperimentazione dei componenti e del sistema AAS in un test .

WP 600 - prova del sistema AAS in aeroporti "pilota"

WP 700 - Valutazione della prova

WP 800 - Progetto di informazione e di comunicazione si concentra sulla strategia per la comunicazione interna ed esterna, per l'implementazione del sistema AAS in altri aeroporti

La figura 2 mostra l'architettura del sistema AAS

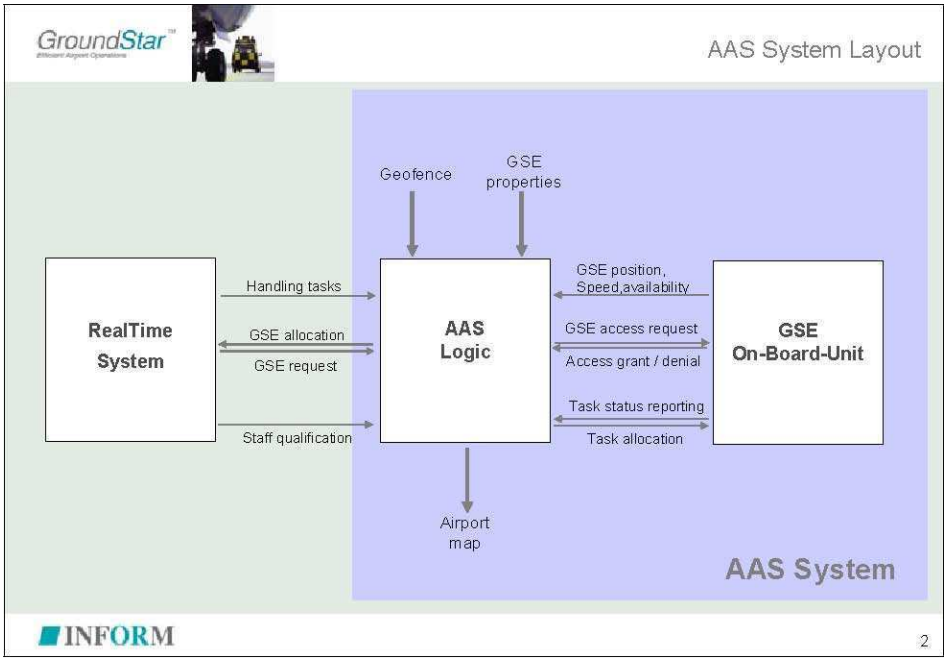

Figura 2

L'unità logica, fra le altre cose, si avvale di un software che risolve un modello matematico per l'ottimizzazione delle richieste dei GSE disponibili per le varie task.

Questa tesi si occupa di un problema che sta all'interno della AAS Logic e quindi è nel cuore del progetto AAS. Più precisamente la tesi sviluppa un algoritmo per l'assegnazione ottimale dei veicoli GSE a risorse umane in un apron aeroportuale. L'assegnazione ottima e' ottenuta tramite l'implementazione di un modello di Min Cost Flow (MCF).

Capitolo 2: problemi di flusso a costo minimo

Il problema di Flusso a Costo Minimo consiste nel determinare la via più economica per trasportare una determinata quantità di un bene da uno o più punti di produzione ad uno o più punti di consumo, attraverso una rete di trasporto data. Visto che il modello matematico che risolve un problema a costo minimo si adatta a realtà diverse, si preferisce la nozione astratta di flusso mentre la rete può essere rappresentata mediante un grafo orientato.

Capitolo 2.1: problemi di flusso a costo minimo Vs progetto AAS

La figura 3 mostra una rappresentazione di un possibile scenario in cui l'unità logica del progetto AAS deve assegnare nel modo più opportuno le risorse (GSE) alle varie attività (task) presenti in quel momento nell'apron.

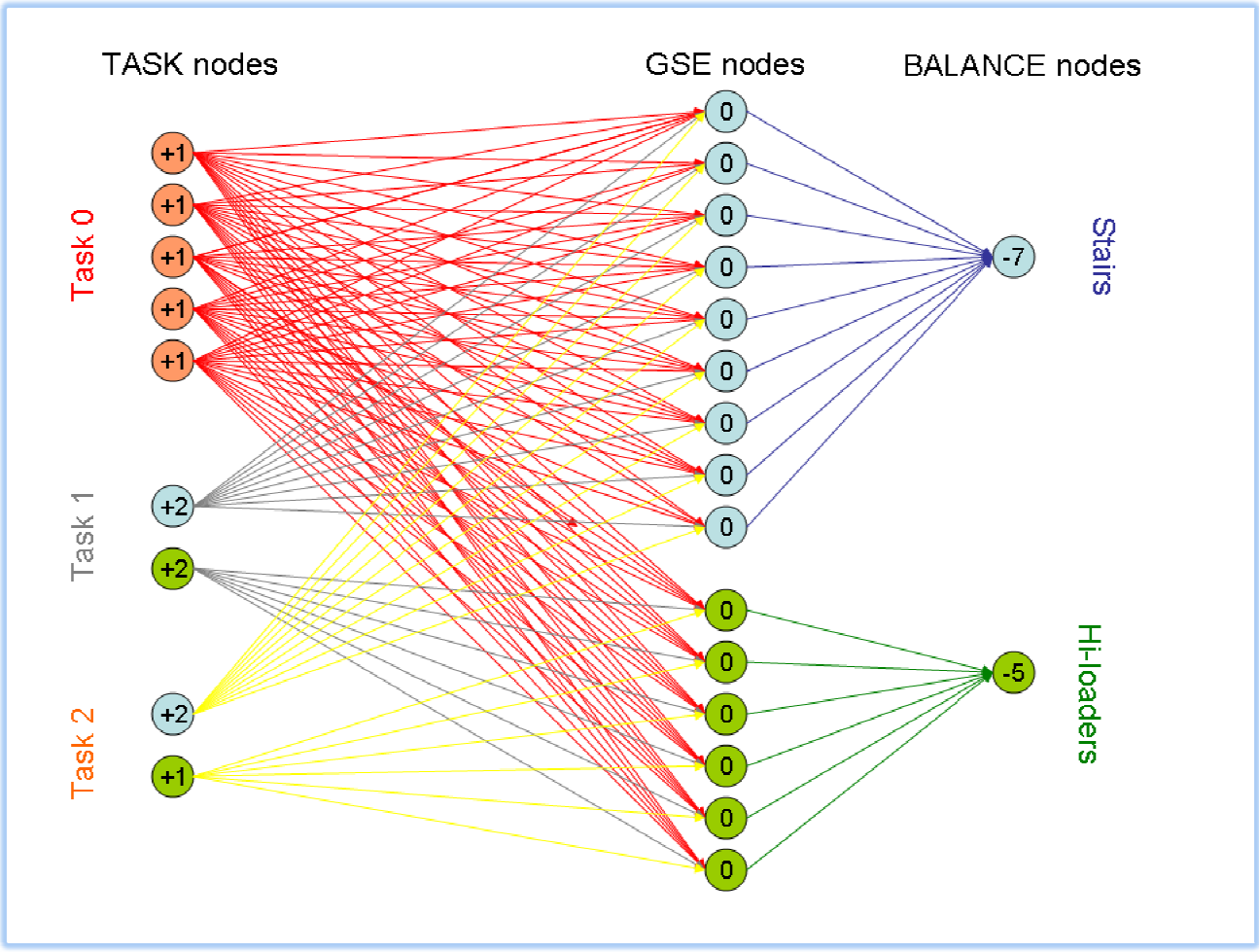

Figura 3

Ad ogni attività sono già state assegnate le squadre precedentemente, possiamo valutare la "task 0" come attività di riferimento, mentre le altre sono in qualche modo sono in competizione per le risorse (GSE) disponibili. La figura 3 mostra la prima iterazione, poi, in modo sequenziale la task 1 diventerà l'attività di riferimento per l'iterazione successiva, continuando così fino all'assegnazione di tutte le attività nell'arco temporale designato. In questo esempio suppongo tre

attività e due tipi di GSE (Stairs e Hi-loaders), gli archi orientati che uniscono queste due tipologie di nodi hanno un costo (non negativo). Da notare che i numeri all'interno dei nodi indicano una quantità di un bene (astratto), si può dire che i nodi di tipo Task hanno un eccesso, quelli di tipo GSE sono di transito mentre quelli finali di 'balance' sono in difetto, si dice anche che questi nodi generano uno sbilanciamento del flusso. Quando il grafo sarà bilanciato avremo risolto il problema di flusso a costo minimo.

Capitolo 2.2: flussi a costo minimo

In termini matematici, si parla di flussi a costo minimo quando G= (N,A) è un grafo orientato con associati ad ogni arco (i, j) ϵ A due numeri k_{ij} e c_{ij} che rappresentano rispettivamente la capacità superiore e il costo unitario di attraversamento dell'arco (i , j). Siano inoltre fissati, per ogni nodo i ϵ N, dei numeri b(i) che rappresentano la richiesta (quando $b(i) < 0$) o la disponibilità (quando $b(i) > 0$) del nodo i.

Il problema del flusso a costo minimo può essere così formulato:

$$Min \sum_{(i,j)\in A} c_{ij} \bullet k_{ij}$$

Soggetto a:

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b(i) \qquad \forall\, i \in N$$

$$0 \le x_{ij} \le k_{ij} \qquad \forall\, (i,j) \in A$$

Nel seguito si assume che tutti i dati siano interi, che tutti i costi e le capacità superiori siano positivi, che $\sum_{i\in N} b(i) = 0$ ed infine che esista una soluzione ammissibile per il problema (e quindi anche una soluzione ottima).

Concetto di pseudoflusso

Si introduce inoltre il concetto di *pseudoflusso*: una funzione $A \to R$ si dice pseudo flusso quando soddisfa i vincoli $0 \le x_{ij} \le k_{ij} \quad \forall\, (i,j) \in A$

Dato un pseudoflusso x, si definisca $e(i)$, per ogni nodo i, nel seguente modo:

$$e(i) = b(i) - \sum_{j:(i,j)\in A} x_{ij} + \sum_{j:(j,i)\in A} x_{ji}$$

Diciamo quindi, che il nodo i presenta un eccesso se $e(i) > 0$ o un deficit se $e(i) < 0$.

Concetto di rete residua

Data una rete, come sopra specificato, ed un pseudo flusso x, la rete residua $G(x) = (N, A(x))$ si ottiene da $G = (N, A)$ sostituendo ad ogni arco (i, j) di A una coppia di archi (i, j) e (j, i): l'arco (i, j) ha costo c_{ij} e capacità residua $r_{ij} = k_{ij} - x_{ij}$, mentre l'arco (j, i) ha costo $c_{ji} = -c_{ij}$ e capacità residua $r_{ji} = x_{ij}$. Infine si eliminano gli archi con capacità residua nulla. Intuitivamente, se nella rete originaria il nodo u ha un eccesso ed il nodo v un deficit, allora l'eccesso di u e il deficit di v possono essere ridotti se nella rete residua vi è un cammino da u a v. Poiché voglio minimizzare i costi, fra tutti i cammini viene scelto quello di costo minimo.

Capitolo 2.3: problemi di cammino a costo minimo

Si consideri un grafo orientato e valutato G = (N, A) e si denoti con $l(\alpha)$ o con $d(i, j)$ la valutazione associata al generico arco $\alpha = (i, j)$. Tale valutazione, detta anche lunghezza dell'arco (i, j), può essere positiva, negativa o nulla.

Dato un cammino $P = (\alpha_1, \alpha_2, \dots, \alpha_n)$ si definisce come lunghezza del cammino la somma delle valutazioni (lunghezze) degli archi che compongono il cammino:

$$l(P) = \sum_{i=1}^{k} l(\alpha_i)$$

La nostra funzione obiettivo sarà trovare una soluzione al problema

$\underset{P}{Min}\, l(P)$ dove P è un qualunque cammino nella rete residua che collega un nodo in eccesso ad uno con difetto

Molti algoritmi per la risoluzione del problema del cammino a costo minimo, associano ad ogni nodo v un'etichetta (in inglese "label"). Tale etichetta, durante l'esecuzione dell'algoritmo, rappresenta una stima per eccesso della lunghezza del cammino più breve da un certo nodo fissato s al suddetto nodo v. Quando questa etichetta diventa definitiva, vuol dire che essa rappresenta l'esatta lunghezza (e non solo un confine superiore) del cammino più breve dal nodo s al nodo v a cui si riferisce.

Algoritmo di Dijkstra

L'algoritmo di Dijkstra viene usato per trovare un cammino più breve in un grafo. Il metodo si basa sull'assegnazione ai nodi di etichette (labels) provvisorie che sono dei confini superiori per la lunghezza del cammino più breve dal nodo s (detto sorgente) al nodo generico etichettato. Le etichette vengono continuamente aggiornate con un processo iterativo e, in ciascuna iterazione,

esattamente una delle etichette diventa permanente (definitiva), indicando l'esatta lunghezza del cammino più breve da s al nodo in questione.
Per la validità dell'algoritmo di Dijkstra e' necessario che la lunghezza di ciascun arco sia non negativa: $d(i,j) \geq 0 \qquad \forall\,(i,j) \in A$

Se l'arco *(i, j)* non esiste si pone per convenienza $d(i,j) = \infty$

Per applicare l'algoritmo si usa:

$l(v_i)$ l'etichetta del generico nodo v_i che rappresenta la lunghezza del miglior cammino da s a v_i correttamente trovato

$p(v_i)$ il nodo che nel cammino correttamente migliore da s fino a v_i, precede immediatamente il nodo v_i.

Descrizione dell'algoritmo di Dijkstra

PASSO 1: *inizializzazione*

Si ponga $l(s) \coloneqq 0$ e si dichiari tale etichetta definitiva;

si ponga per tutti i nodi $v \neq s$ (etichette provvisorie);

si ponga $p(s) \coloneqq s$ e $p(v) \coloneqq 0 \qquad \forall\, v \neq s$;

si ponga $u{:=}s$

PASSO 2: *revisione delle etichette*

Per tutti i nodi v immediatamente successori di u che hanno etichetta provvisoria, si revisioni l'etichetta in base al seguente confronto:

Se $l(v) > l(u) + d(u,v)$ allora si ponga: $l(v) \coloneqq l(u) + d(u,v), p(v) \coloneqq u$, altrimenti le etichette $l(v)$ e $p(v)$ rimangono invariate.

PASSO 3: *determinazione di un'etichetta permanente*

Tra tutti i vertici con etichetta provvisoria si scelga un vertice v^* tale che $l(v^*)$ sia minima; si dichiari permanente tale etichetta e si ponga $u \coloneqq v^*$

PASSO 4: *regola d'arresto*

Se $u = t$ si vada al passo 5.
Se $u \neq t$ si torni al passo 2.

PASSO 5: *ricostruzione del cammino*

Se $l(t) = \infty$ allora non esiste alcun cammino da s a t.

Se $l(t) < \infty$ allora si può ricostruire a ritroso il cammino ottimo da s a t nel seguente modo:

$$t \leftarrow p(t) \leftarrow p[p(t)] \leftarrow \cdots \leftarrow p\{\dots p[p(t)]\} = s$$

STOP

Cenni sulla complessità computazionale dell'algoritmo di Dijkstra

Nell'algoritmo di Dijkstra ogni etichetta di un nodo comporta un numero di operazioni che cresce al più linearmente con n. Poiché ci sono n nodi da etichettare, l'algoritmo di Dijkstra ha complessità $O(n^2)$.

Capitolo 2.4: problemi di flusso a costo minimo

Il problema di flusso a costo minimo può essere ricondotto alla risoluzione iterativa di problemi di Cammini Minimi da un nodo con eccesso e un nodo con deficit. Nella generica iterazione bisogna trovare un cammino di costo minimo da un nodo con eccesso s e un nodo con deficit u nella rete residua (vedi concetto di rete residua).

Ricordo che nella rete residua ci sono i costi

- $c'_{ij} = -c_{ij}$ se l'arco (i, j) fa parte di un cammino minimo trovato in una qualsiasi iterazione precedente
- $c'_{ij} = c_{ij}$ se l'arco (i, j) non fa parte di nessun cammino minimo

Quindi, ad eccezione della prima iterazione in cui i costi sono tutti $c_{ij} \geq 0$, in quelle successive non posso applicare l'algoritmo di Dijkstra perché questo richiede sempre che tutte le $c'_{ij} \geq 0$.

In questo caso si modificano i coefficienti di costo ridotto c'_{ij} in c''_{ij} in modo che $c''_{ij} \geq 0 \; \forall \; (i,j) \in A$ ed il cammino minimo ottenuto usando quest'ultimi sia minimo anche per i coefficienti di costo ridotto c'_{ij}.

Per ottenere ciò, introduciamo una funzione *potenziale* π, definita sui nodi nel modo seguente:

- Inizialmente $\pi(i) \coloneqq 0 \; \forall i$
- Successivamente, indicando con *d(i)* la distanza ottima del nodo i al nodo s, trovata dalla più recente applicazione dell'algoritmo di Dijkstra, aggiorno i potenziali con:

$$\pi(i) \coloneqq \begin{cases} \pi(i) - d(i) + d(u) \; se \; il \; nodo \; i \text{ è } etichettato \; in \; modo \; permanente \\ \pi(i) \qquad\qquad se \; il \; nodo \; i \; NON \text{ è } etichettato \; in \; modo \; permanente \end{cases}$$

I nuovi coefficienti di costo ridotto da usare per la successiva applicazione dell'algoritmo di Dijkstra sono:

$$c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$$

Si può dimostrare che $c_{ij}^{\pi} \geq 0 \; \forall (i,j) \in A$ quindi si può applicate l'algoritmo di Dijkstra.

Ogni cammino minimo trovato abbassa lo sbilanciamento (nel nostro caso esattamente di 1 unità) e quindi dopo un numero di iterazioni per annullare lo sbilanciamento complessivo iniziale si arriva ad un flusso che è di costo minimo.

Capitolo 3: soluzione del problema

In questo capitolo illustro una possibile algoritmo per risolvere il problema dell'assegnazione di veicoli GSE a risorse umane del progetto AAS avendolo modellato come un problema di flusso a costo minimo.

Capitolo 3.1: considerazioni iniziali

Per semplicità, iniziamo col considerare la prima iterazione del problema, rappresentato nel grafo della figura 3 (pag. 4), esso rappresenta una finestra temporale all'interno di una giornata di lavoro nell'apron aeroportuale, la lunghezza di questa finestra potrebbe coincidere col tempo che impiega un aereo ad atterrare, eseguire le operazioni da parte delle squadre assegnate e ripartire. Si è adottata questa semplificazione perché riduce notevolmente il carico computazionale dell'algoritmo e quindi la sua complessità, ricordo inoltre che il problema dovrebbe essere risolto in tempo reale.

I nodi che riguardano le attività (task) sono suddivisi in 2 gruppi, il primo è la task di riferimento composta da cinque elementi, nel secondo gruppo ci sono tutte le task che vanno in competizione per le risorse GSE disponibili. Da notare che dal secondo gruppo in poi gli elementi sono accorpati in sottogruppi, tanti quanti le tipologie GSE. Questo approccio ha consentito di ridefinire la complessità del problema ad una complessità polinomiale.

Nell'esempio ci sono 2 tipi di nodi GSE, per convenienza li chiamo gruppo α e gruppo β, ogni membro della task di riferimento ha accesso a tutti i nodi GSE, mentre nelle task successive solo un sottogruppo ha accesso al tipo di GSE di riferimento.

I nodi con eccesso sono i nodi delle task, nello specifico: i nodi appartenenti alla task di riferimento hanno un eccesso di 1 unità, mentre i sottogruppi delle altre task hanno un valore compreso fra 1 e 3; i nodi con difetto sono i 'balance' e la loro somma rappresenta il totale di risorse GSE di tipo α e β richieste in quella finestra temporale, i nodi GSE sono solo di transito (ossia hanno b(i) = 0).

Ogni arco che unisce un nodo di una task a un nodo GSE ha un peso non negativo, ogni arco che unisce un nodo GSE al suo nodo 'deficit' rispettivo ha costo zero. Tutti gli archi hanno una capacità unitaria, quindi se uso l'arco (i, j) il costo totale è esattamente.

Per le considerazioni fatte prima sul problema di flusso aggiungo ad ogni nodo i anche un suo potenziale ($\pi\ (i)$), e per ogni arco ho il costo originario c_{ij} e il costo corrente c_{ij}^{π}.

L'algoritmo fa alcune assunzioni di base come, per esempio, che ci siano abbastanza nodi GSE per soddisfare le richieste e che tutti i costi iniziali siano tutti positivi. Se nel caso reale mancano archi possiamo inserirli lo stesso con costo pari a infinito.

Capitolo 3.2: descrizione della generica iterazione dell'algoritmo

Questo è un possibile algoritmo per una soluzione ottimale del problema scritto in pseudo-linguaggio.

PASSO 1: inizializzazione

Controllo che i costi siano non negativi
Pongo $\pi(i) := 0 \quad \forall i$
Controllo che $\sum_{i \in N} b(i) = 0$
Pongo $c_{ij}^{\pi} := c_{ij} \quad \forall (i,j) \in A$

PASSO 2: Inizio

$s \leftarrow un\ nodo\ con\ b(i) > 0.$

PASSO 3: algoritmo di Dijkstra

Eseguo l'algoritmo di Dijkstra descritto nel precedente capitolo usando però i costi ridotti c_{ij}^{π} per il confronto e il calcolo delle etichette provvisorie.

PASSO 4: fine algoritmo di Dijkstra

Appena etichetto in modo definitivo un nodo con $b(i) < 0$ mi fermo (esempio nodo u).
Assesto le richieste del nodo s (-1) e del nodo u (+1)
Sia *d(l)* la distanza dal nodo s al nodo u sul cammino di Dijkstra
Sia *d(i)* la distanza dal nodo s al nodo i sul cammino di Dijkstra

PASSO 5: aggiorno i potenziali

Aggiorno tutti i $\pi(i)$ per tutti i nodi analizzati nell'algoritmo di Dijkstra nel seguente modo:

$$\pi(i) := \begin{cases} \pi(i) - d(i) + d(l)\ se\ il\ nodo\ i\ \text{è}\ etichettato\ in\ modo\ permanente \\ \pi(i) \qquad\qquad se\ il\ nodo\ i\ NON\ \text{è}\ etichettato\ in\ modo\ permanente \end{cases}$$

PASSO 6: aggiorno i costi ridotti

Aggiorno il grafo con i costi ridotti per tutti gli archi entrati ed uscenti da un nodo con potenziale $\pi(i)$ modificato:

$$c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$$

PASSO 7: aggiorno il grafo residuo

Aggiorno il grafo residuo, cambiando gli archi coinvolti nel cammino minimo di Dijkstra

PASSO 8: fine

Se ci sono ancora nodi con eccesso torno al PASSO 2
Altrimenti ho finito e costruisco la soluzione partendo dagli archi invertiti a costo non nullo.

Capitolo 3.3: applicazione dell'algoritmo

Prendendo spunto dal grafo della figura 3, ridisegno nella figura 4 il problema dando un nome ai nodi:

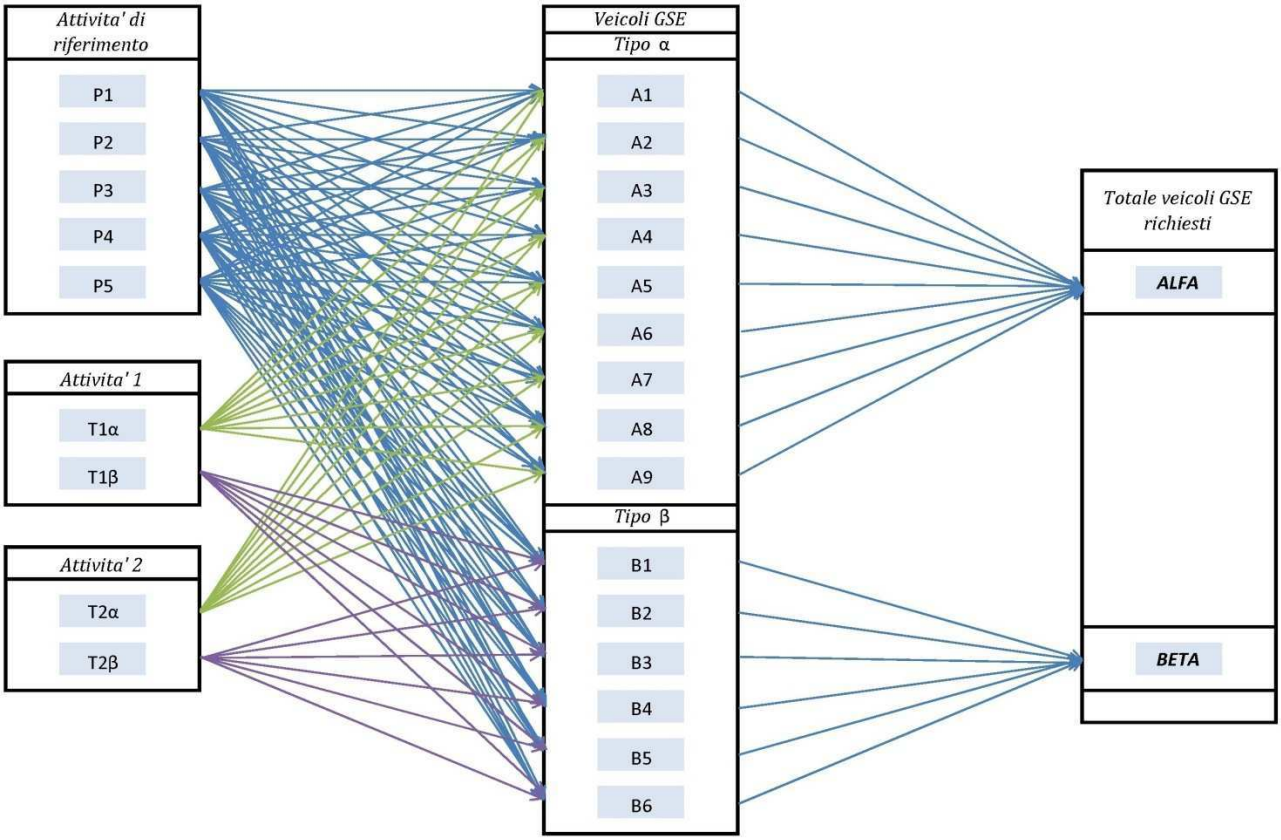

Figura 4

La tabella 1 mostra l'esempio della figura 4 in forma matriciale, sulle righe ci sono i componenti delle attività, sulle colonne i nodi GSE e ogni cella *(i, j)* della matrice rappresenta il costo per assegnare la risorsa j all'unità i.

	GSE	Tipo α									Tipo β						
		a1	**a2**	**a3**	**a4**	**a5**	**a6**	**a7**	**a8**	**a9**	**b1**	**b2**	**b3**	**b4**	**b5**	**b6**	**b(i)**
Task 0	**P1**	1	5	7	4	3	5	10	11	1	107	109	107	106	105	104	**1**
	P2	3	4	5	6	3	4	5	7	10	5	6	7	8	9	10	**1**
	P3	10	11	3	4	3	11	21	10	11	7	6	7	9	10	1	**1**
	P4	7	8	9	10	9	8	7	6	5	3	4	5	6	7	3	**1**
	P5	5	4	3	2	1	10	7	7	9	7	3	2	1	10	11	**1**
Task 1	**T1- α**	1	2	3	4	1	2	3	5	7	-	-	-	-	-	-	**2**
	T1- β	-	-	-	-	-	-	-	-	-	10	11	9	7	6	5	**2**
Task 2	**T2- α**	5	6	8	9	10	11	3	6	8	-	-	-	-	-	-	**2**
	T2- β	-	-	-	-	-	-	-	-	-	12	30	5	7	1	3	**1**
Richieste	**ALFA**	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	**-7**
	BETA	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	**-5**

Tabella 1

Inizio ad applicare l'algoritmo

Riporto solo le fasi più esplicative dell'algoritmo

PASSO 2:

Nodo sorgente : T2β (attività 2) con eccesso : 1

PASSO 4:

Trovato nodo destinatario : BETA con difetto : -5
Percorso a ritroso nodo(costo): BETA (1)<-B5 (1)<- T2β (0)
Nodi etichettati in modo definitivo : [T2-β] [B5] [BETA]

PASSO 5:

Aggiornamento pgreco (potenziali)
pgreco nodo[T2β]= 1 [vecchio pgreco= 0]
pgreco nodo[B5]= 0 [vecchio pgreco= 0]
pgreco nodo[BETA]= 0 [vecchio pgreco= 0]

PASSO 6:

Aggiornamento costi ridotti
$c^{\pi}_{T2\beta;B6} = c_{T2\beta;B6} - \pi(T2\beta) + \pi(B6)$
$c^{\pi}_{T2\beta;B6} = 3 - 1 + 0 = 2$
Analogamente:
$c^{\pi}_{T2\beta;B5} = 0$
$c^{\pi}_{T2\beta;B4} = 6$
$c^{\pi}_{T2\beta;B3} = 4$
$c^{\pi}_{T2\beta;B2} = 29$
$c^{\pi}_{T2\beta;B1} = 11$

PASSO 7:

Aggiornamento del grafo
Elimino arco nodo[B5] -> [BETA]
Aggiungo arco nodo[BETA] -> [B5]
Elimino arco nodo[Bbeta] -> [B5]
Aggiungo arco nodo[B5] -> [Bbeta]

PASSO 8:

Fine iterazione 1
Ritorno al PASSO 2

Da questa iterazione ottengo un grafo residuo rappresentato nella figura 5:

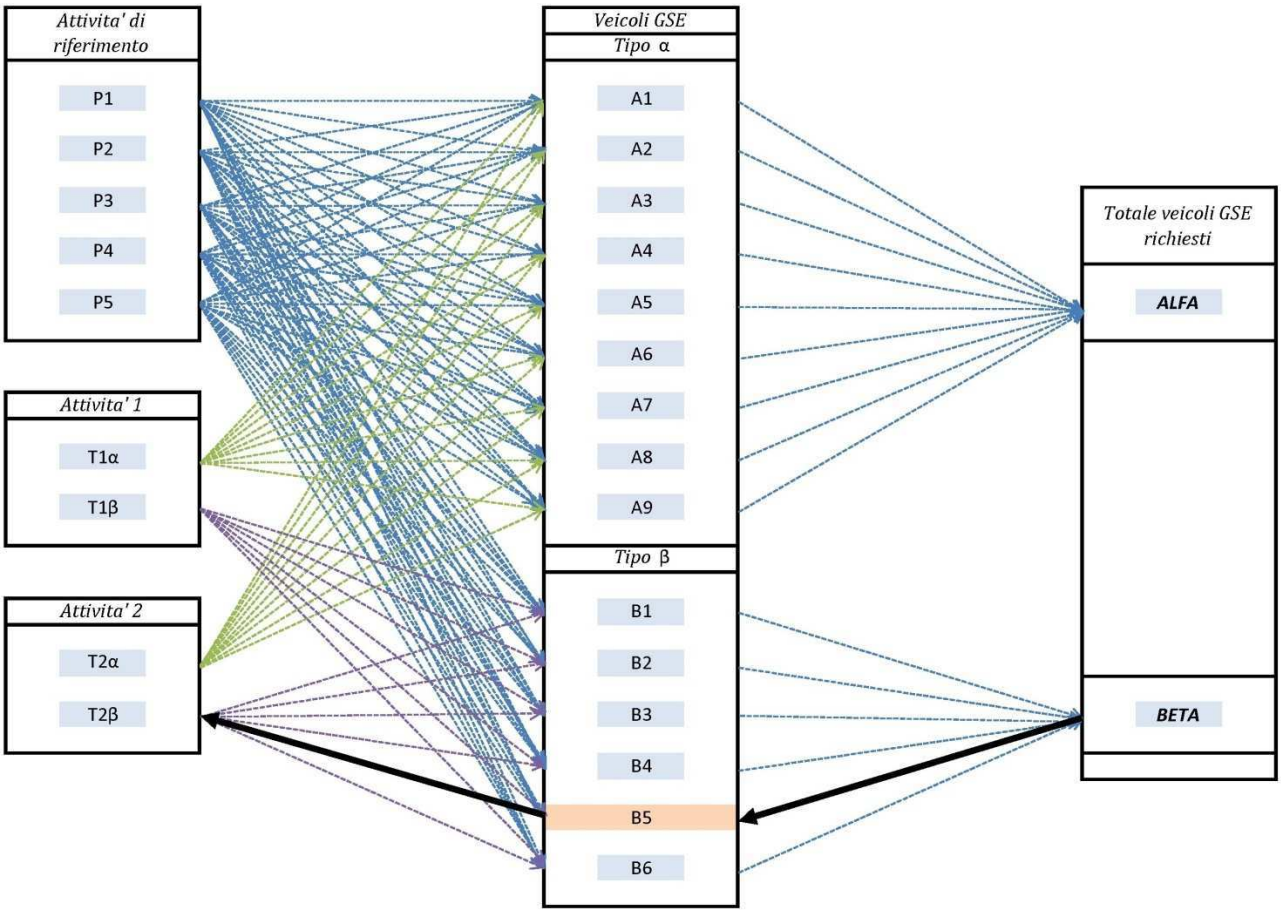

Figura 5

NB: Gli archi invertiti sono di colore nero

Nella prossima iterazione riparto da questo grafo residuo, al posto di quello originario, ottengo quindi:

PASSO 2:

Nodo sorgente : T2α (attività 2) con eccesso : 2

PASSO 4:

Trovato nodo destinatario : ALFA con difetto : -7
Percorso a ritroso nodo(costo): ALFA (3)<-A7 (3)<- T2α (0)
Nodi etichettati in modo definitivo :
[T2α] [A7] [ALFA]

PASSO 5:

Aggiornamento pgreco
pgreco nodo[Balfa]= 3 [vecchio pgreco= 0]
pgreco nodo[A7]= 0 [vecchio pgreco= 0]
pgreco nodo[ALFA]= 0 [vecchio pgreco= 0]

PASSO 6:

Aggiornamento costi ridotti

(analogamente ricalcolo i costi ridotti come nello step precedente)

$c^{\pi}_{T2\alpha;A9} = 5$

$c^{\pi}_{T2\alpha;A8} = 3$

$c^{\pi}_{T2\alpha;A7} = 0$

$c^{\pi}_{T2\alpha;A6} = 8$

$c^{\pi}_{T2\alpha;A5} = 7$

$c^{\pi}_{T2\alpha;A4} = 6$

$c^{\pi}_{T2\alpha;A3} = 5$

$c^{\pi}_{T2\alpha;A2} = 3$

$c^{\pi}_{T2\alpha;A1} = 2$

PASSO 7:

Aggiornamento del grafo

Elimino arco nodo[A7] -> [ALFA]

Aggiungo arco nodo[ALFA] -> [A7]

Elimino arco nodo[T2α] -> [A7]

Aggiungo arco nodo[A7] -> [T2α]

PASSO 8:

Fine iterazione 2

Ritorno al PASSO 2

Da questa iterazione ottengo un grafo residuo descritto nella figura 6:

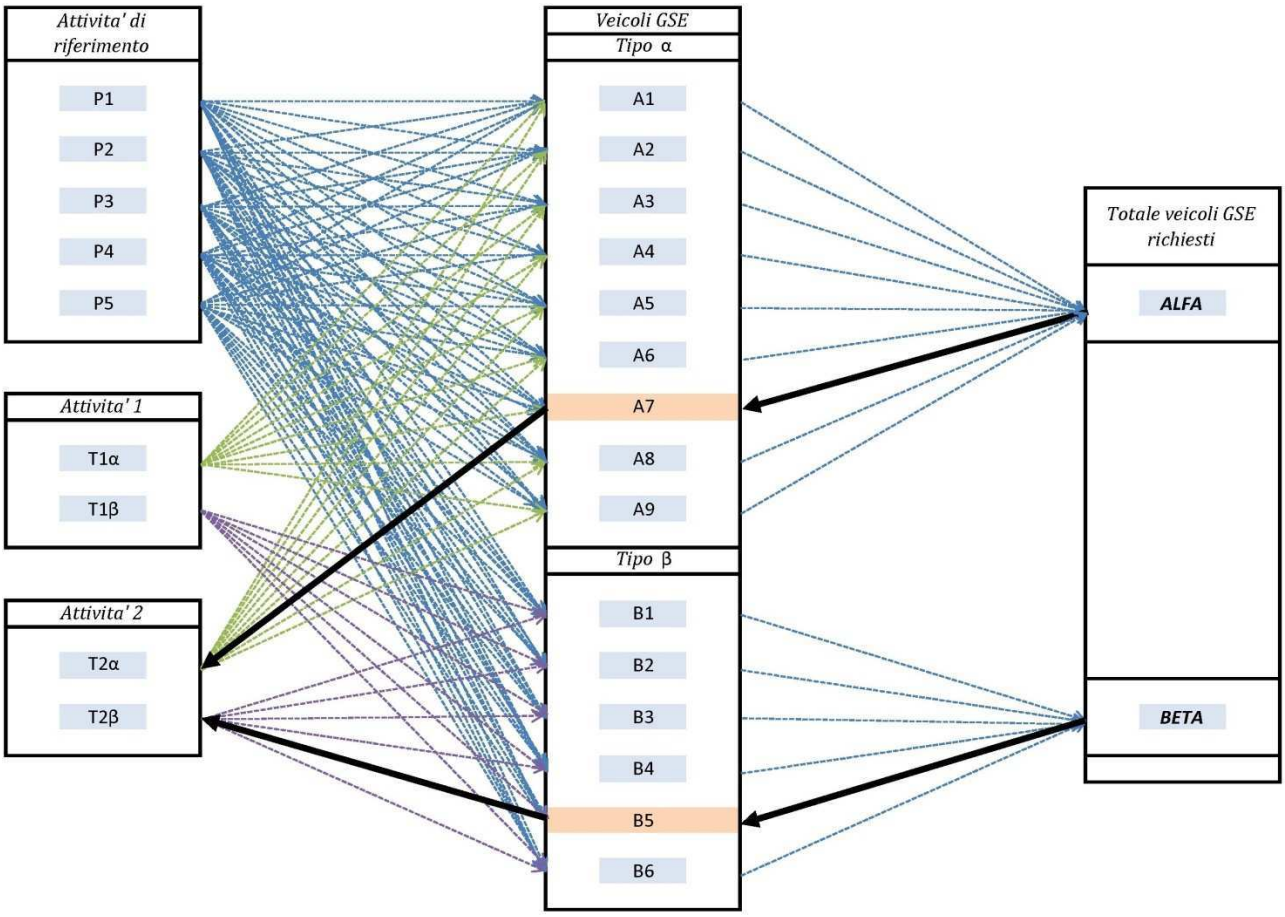

Figura 6

L'algoritmo continua assegnando nell'ordine:

$$\boldsymbol{T2\alpha \rightarrow A1 \rightarrow ALFA}$$

Task 1

$$\boldsymbol{T1\beta \rightarrow B6 \rightarrow BETA}$$

$$\boldsymbol{T1\beta \rightarrow B4 \rightarrow BETA}$$

$$\boldsymbol{T1\alpha \rightarrow A5 \rightarrow ALFA}$$

$$\boldsymbol{T1\alpha \rightarrow A2 \rightarrow ALFA}$$

Task di riferimento

$$\boldsymbol{P5 \rightarrow A4 \rightarrow ALFA}$$

$$\boldsymbol{P4 \rightarrow B1 \rightarrow BETA}$$

$$\boldsymbol{P3 \rightarrow A3 \rightarrow ALFA}$$

$$\boldsymbol{P2 \rightarrow A6 \rightarrow ALFA}$$

Di particolare interesse è l'ultima iterazione dell'algoritmo, il grafo della figura 7 riporta lo stato delle assegnazioni correnti:

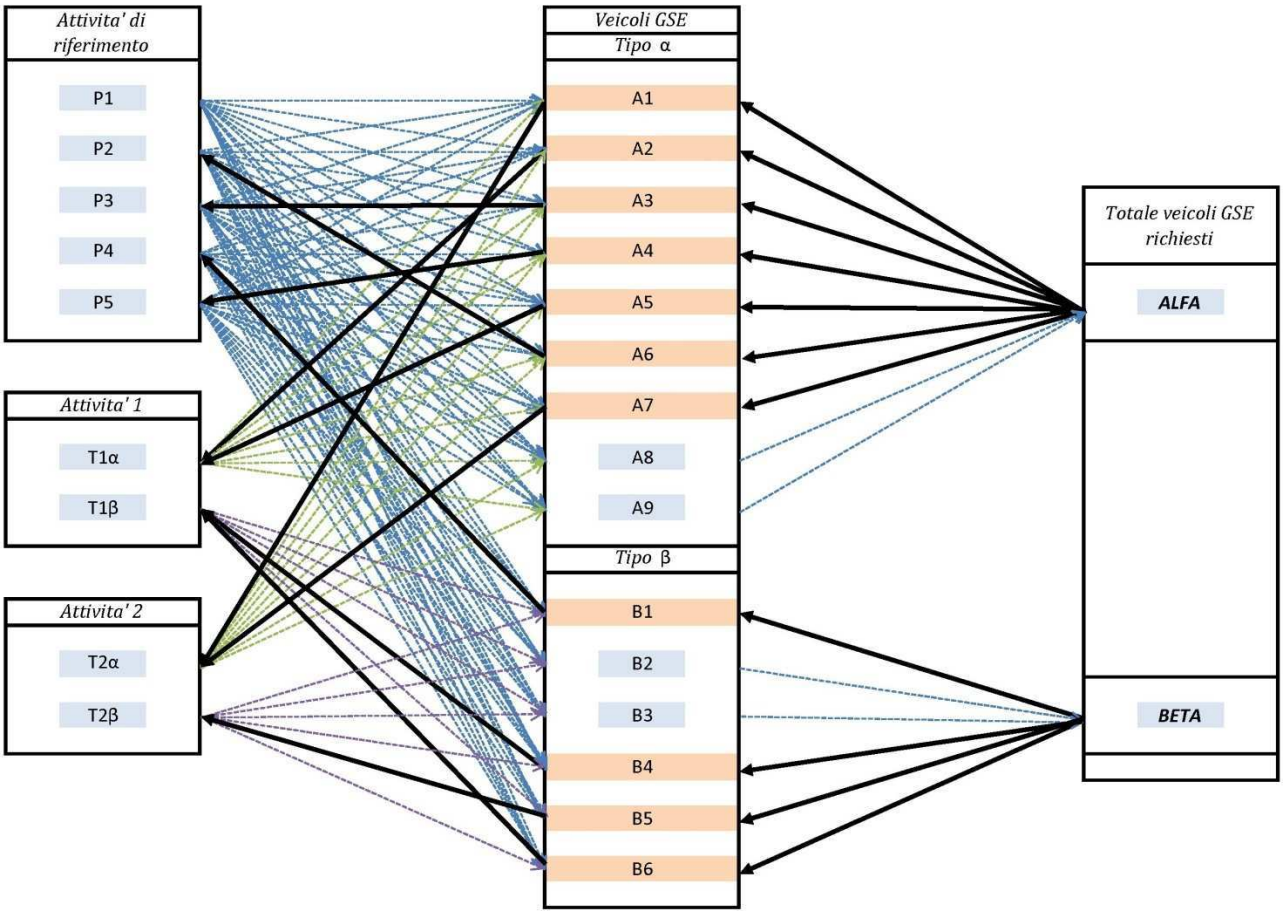

Figura 7

L'ultima iterazione produce questo cammino minimo per P1:

$$\boldsymbol{P1 \rightarrow A9 \rightarrow ALFA \rightarrow A4 \rightarrow P5 \rightarrow B3 \rightarrow BETA}$$

Il significato di questo assegnamento e' riassegnare P5 alla risorsa B3, e assegnare la risorsa A9 a P1.

Si nota questo cambio nel grafo della figura 8 (archi in nero rappresentano il risultato delle precedenti iterazioni, la nuova è in di colore rosso)

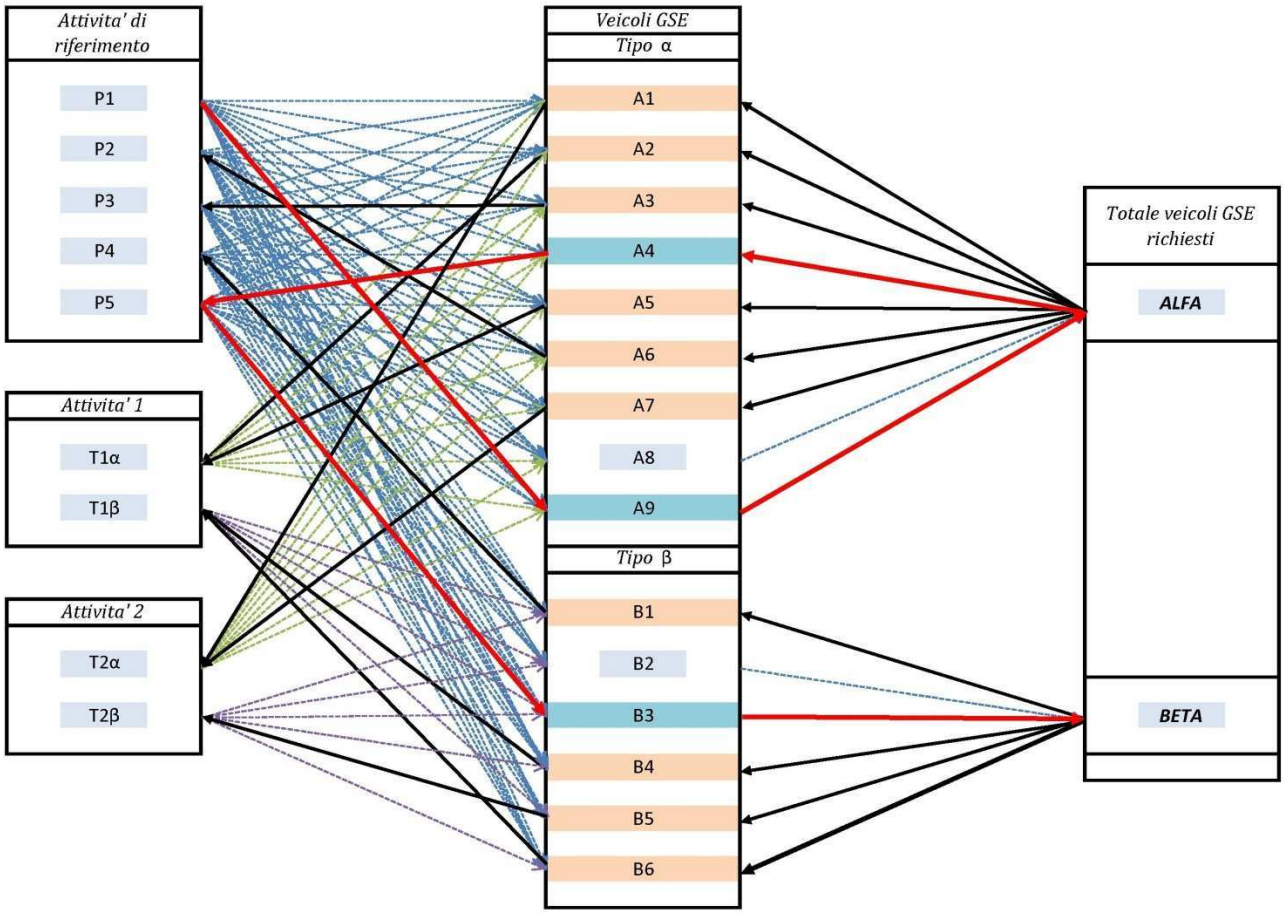

Figura 8

Dopo l'algoritmo ottengo questa assegnazione delle risorse GSE alle varie attività:

Task	Risorse GSE Assegnate
Attività di riferimento	
P1	**A9**
P2	**A6**
P3	**A3**
P4	**B1**
P5	**B3**
Attività 1	
T1α	**A1 – A5**
T1β	**B4 – B6**
Attività 2	
T2α	**A1 – A7**
T2β	**B5**
Costo Totale	**37**

Tabella 2

Da notare che l'assegnamento potrebbe avvenire anche in modo diverso, ma il costo totale, essendo la funzione obiettivo da minimizzare resta sempre 37.

Capitolo 3.4: programma in C dell'algoritmo

Il C è rinomato per la sua efficienza, e si è imposto come linguaggio di riferimento per la realizzazione di software di sistema su gran parte delle piattaforme hardware moderne. La standardizzazione del linguaggio (da parte dell'ANSI prima e dell'ISO poi) garantisce la portabilità dei programmi scritti in C (standard, spesso detto ANSI C) su qualsiasi piattaforma.

Il programma in C dell'algoritmo prevede la gestione dei dati del grafo non in forma matriciale ma in forma di lista di adiacenza. È probabilmente la rappresentazione più immediata a cui è possibile pensare e la più semplice da implementare, l'idea della rappresentazione è semplicemente che ad ogni vertice V viene associata una lista contenente tutti e soli i vertici W tali che esista l'arco da V a W. Per quel che riguarda l'efficienza in termini di tempo, la rappresentazione per liste di adiacenza si comporta piuttosto bene sia nell'accesso che nell'inserimento, effettuando le operazioni principali in tempo O(n) e nel nostro caso ha anche un buon compromesso di efficienza in memoria.

Per provare più volte l'algoritmo genero con una funzione una lista di adiacenza sempre diversa, cosi facendo non solo valuto la robustezza dell'algoritmo ma anche la sua velocità di esecuzione.

Capitolo 3.5: test dell'applicazione in C dell'algoritmo

Per dare un'idea di quanto l'algoritmo sia veloce, ho fatto un test usando il grafo di esempio in Excel e risolvendolo col risolutore, ebbene, a parità di hardware, il programma in C impiega pochi centesimi di secondo mentre il risolutore impiega più di un secondo (eccedendo anche i suoi default). Di seguito illustro una serie di prove del programma C su varie piattaforme e sistemi operativi, il programma genera 50 task, con una richiesta per sottogruppo che varia da 1 a 3 e ci sono 3 tipologie di GSE.
I test consistono in 5 prove consecutive registrando il tempo di impiego della CPU dal programma, per ogni set di prove, riporto nella tabella 2 il tempo minimo, medio e massimo espresso in secondi.

Test	Cpu	S.O.	Min	Medio	Max
1	Intel Core 2 E6600 2.4GHz	WinXP SP3	0,062	0,084	0,094
2	Pentium III 1.0 GHz	Linux RH 7.3	1,100	1,416	2,270
3	Intel(R) Xeon(TM) CPU 2.40GHz	Linux SL 305	0,132	0,148	0,170
4	Dual Core AMD Opteron(tm) 270 2.0GHz	Linux SLC 47	0,100	0,138	0,190
5	Intel(R) Xeon(R) E5420 @ 2.50GHz	Linux SLC 46 x64	0,050	0,066	0,100
6	Intel(R) Xeon(TM) CPU 3.00GHz	Linux SLC 46	0,120	0,146	0,170
7	Quad-Core AMD Opteron(tm) 2360 2.5GHz	Linux SL 45	0,080	0,092	0,130

Tabella 3

Nella figura 9 riporto i risultati in forma grafica :

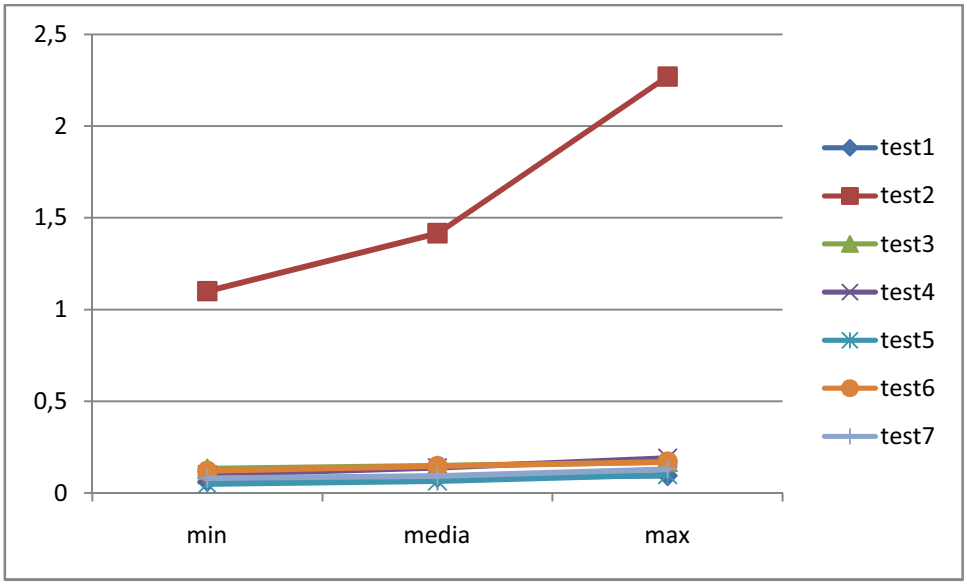

Figura 9

Nella figura 9 si nota il netto divario delle prestazioni del test 2 che è una macchina ormai obsoleta rispetto alle altre, mentre un elemento positivo va al primo test, eseguito sotto windows con un porting del compilatore.

Nel grafico della figura 10 rappresento solo le macchine di ultima generazione con stesso sistema operativo cosi da poter valutare meglio le prestazioni:

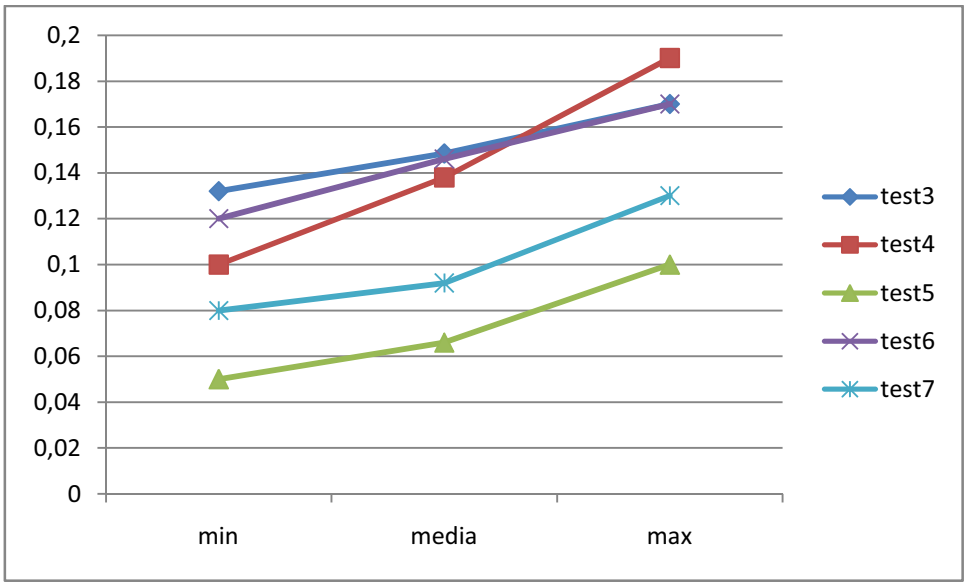

Figura 10

Concludo che il programma ottiene la massima performance da una CPU con molta cache (memoria interna alla CPU), come i processori Intel di ultima generazione.

Appendice A: codice C

```
/*
**  Fabio Bredo
**
*/

#include <stdlib.h>
#include <stdio.h>
#include <limits.h>
#include <conio.h>
#include <sys/types.h>
#include <time.h>
#include <string.h>

#define MAX 50
#define INFINITO 999
/*#define INFINITO INT_MAX*/

#define NTASK    50
#define PPLTASK  3
#define PPLTASK1 5
#define TIPORES  3
#define MAXBI    3
#define TIPOA    (5+NTASK*MAXBI)
#define TIPOB    (5+NTASK*MAXBI)
#define TIPOC    (5+NTASK*MAXBI)
#define MAXCOSTO 100

/* TALFA + TBETA + TGAMMA == 5 !!! */

#define TALFA     2
#define TBETA     2
#define TGAMMA    1

#define NNODI    (5+(NTASK-1)*PPLTASK+TIPOA+TIPOB+TIPOC+TIPORES)
#define STARTGSE (5+(NTASK-1)*PPLTASK)

struct nodo {
  int v;
  int costo;
  int c_orig;
  struct nodo *next;
};

struct link_solve{
  int orig;
  int dest;
  int costo;
  struct link_solve *next;
};


 void leggi_grafo_dinamico(struct nodo *G[],int *bi) {
     int i,n,j,k, ca,cb,cc;
     struct nodo *p, *p1;
```

```
   n=NNODI - TIPORES;
   srand(time(0));

ca=TALFA;
cb=TBETA;
cc=TGAMMA;

 for (i=0;i<NNODI;i++){
   G[i] = NULL;
   bi[i]=0;
   }

 /* Prima task da 5 elementi*/
 for (i=0;i< PPLTASK1;i++){
       bi[i]=1;
       //printf ("\n Genero il nodo %3d (task1) ", i);
       for (j=STARTGSE ; j< n; j++){
       //printf (" -> nodo %3d  ", j);
         p = malloc(sizeof(struct nodo));
         if (p==NULL){
             printf ("\n Errore grave!");
             fflush(NULL);
             system("PAUSE");
             exit(5);
         }
         p->v=j;
         p->costo=1 + rand() % MAXCOSTO;
         p->c_orig=p->costo;
         p->next = G[i];
         G[i] = p;
       }
 } //fine for
 /* Le altre Task NTASK-1*/
   k=TIPORES;

   /* archi costo zero to alfa/beta/gamma*/
   if (k>0 ){
 for (i=PPLTASK1; i< PPLTASK1 + ((NTASK-1) *PPLTASK); i+=PPLTASK){

  // tipo alfa
        bi[i]=1 + rand() % MAXBI;
        ca=ca+bi[i];
        //printf ("\n Genero il nodo %3d (Talfa) %d (%d)", i,  bi[i],ca);
        for (j=STARTGSE ; j< STARTGSE + TIPOA; j++){
            //printf (" -> nodo %3d  ", j);
         p = malloc(sizeof(struct nodo));
         if (p==NULL){
             printf ("\n Errore grave!");
             fflush(NULL);
             system("PAUSE");
             exit(5);
         }
         p->v=j;
         p->costo=1 + rand() % MAXCOSTO;
         p->c_orig=p->costo;
         p->next = G[i];
         G[i] = p;
       }
 }
```

```
   k--;
   }
   if (k>0 ){
   for (i=PPLTASK1+1; i< PPLTASK1 + ((NTASK-1) *PPLTASK); i+=PPLTASK){
       // tipo beta

          bi[i]=1 + rand() % MAXBI;
          cb=cb+bi[i];
          //printf ("\n Genero il nodo %3d (Tbeta) %d (%d)", i, bi[i],cb);
          for (j=STARTGSE+TIPOA ; j< STARTGSE + TIPOA + TIPOB; j++){
              //printf (" -> nodo %3d  ", j);
           p = malloc(sizeof(struct nodo));
           if (p==NULL){
               printf ("\n Errore grave!");
               fflush(NULL);
               system("PAUSE");
               exit(5);
           }
           p->v=j;
           p->costo=1 + rand() % MAXCOSTO;
           p->c_orig=p->costo;
           p->next = G[i];
           G[i] = p;
         }
   }
   k--;
   }
   if (k>0 ){

   for (i=PPLTASK1+2; i< PPLTASK1 + ((NTASK-1) *PPLTASK); i+=PPLTASK){
         // tipo gamma

         bi[i]=1 + rand() % MAXBI;
         cc=cc+bi[i];
          //printf ("\n Genero il nodo %3d (Tgamma) %d (%d)", i, bi[i],cc);
          for (j=STARTGSE+TIPOA+TIPOB ; j< STARTGSE + TIPOA + TIPOB + TIPOC;
j++){
           p = malloc(sizeof(struct nodo));
           if (p==NULL){
               printf ("\n Errore grave!");
               fflush(NULL);
               system("PAUSE");
               exit(5);
           }
           p->v=j;
           p->costo=1 + rand() % MAXCOSTO;
           p->c_orig=p->costo;
           p->next = G[i];
           G[i] = p;
         }
       }//fine for tutte le task gamma
   k--;
   }



       k=TIPORES;

     /* archi costo zero to alfa/beta/gamma*/
```

```
     if (k>0 ){
         bi[NNODI-k]=( -1)* ca;
         for (j=STARTGSE ; j< STARTGSE + TIPOA; j++){
           p = malloc(sizeof(struct nodo));
           if (p==NULL){
               printf ("\n Errore grave!");
               fflush(NULL);
               system("PAUSE");
               exit(5);
           }
           p->v=NNODI-k;
           p->costo=0;
           p->c_orig=p->costo;
           p->next = G[j];
           G[j] = p;
         }
       k--;
       }
       if (k>0 ){
         bi[NNODI-k]=( -1)* cb;
         for (j=STARTGSE +TIPOA ; j< STARTGSE + TIPOA + TIPOB; j++){
           p = malloc(sizeof(struct nodo));
           if (p==NULL){
               printf ("\n Errore grave!");
               fflush(NULL);
               system("PAUSE");
               exit(5);
           }
           p->v=NNODI-k;
           p->costo=0;
           p->c_orig=p->costo;
           p->next = G[j];
           G[j] = p;
         }
       k--;
       }
       if (k>0 ){
         bi[NNODI-k]=( -1)* cc;
         for (j=STARTGSE +TIPOA +TIPOB; j< STARTGSE + TIPOA + TIPOB +TIPOC;
j++){
           p = malloc(sizeof(struct nodo));
           if (p==NULL){
               printf ("\n Errore grave!");
               fflush(NULL);
               system("PAUSE");
               exit(5);
           }
           p->v=NNODI-k;
           p->costo=0;
           p->c_orig=p->costo;
           p->next = G[j];
           G[j] = p;
         }
       k--;
       }
```

```
 }//fine funz


/*
 *  Visualizza la lista di adiacenza a video
 */

void stampa_lista(struct nodo *n) {
    int i=1;
     while (n != NULL) {
        if (i % 4 == 0) printf ("\n");
           printf(" [%3d / %3d] -> ", n->v, n->costo);
           n = n->next;
           i++;
     }
     printf(" NULL\n");
     return;
}
/*
**  Algoritmo: bubble_sort
**
**  Accetta come parametro il puntatore al primo elemento
**  della lista della soluzione;
**  la ordina utilizzando l'algoritmo "bubble sort"
**  e restituisce il puntatore al primo elemento
**  della lista.
*/

struct link_solve *bubble_sort(struct link_solve *s) {
  struct link_solve *p, *last;
  int flag, orig, dest, costo;

  last = NULL;

  flag = 1;
  while (flag == 1) {
    p = s;
    flag = 0;
    while (p->next != last) {
      if (p->orig > (p->next)->orig) {
        orig = p->orig;
        dest = p->dest;
        costo= p->costo;
        p->orig = (p->next)->orig;
        p->dest = (p->next)->dest;
        p->costo = (p->next)->costo;
        (p->next)->orig = orig;
        (p->next)->dest = dest;
        (p->next)->costo = costo;
        flag = 1;
      }
      p = p->next;
    }
    last = p;
  }

  return(s);
}
```

```
/*
 *  Visualizza la soluzione a  video
 */
void stampa_soluzione(struct nodo *G[], int *tipo_nodi, int num_nodi ) {
     int i,c_tot,k,temp;
     struct nodo *p;
     struct link_solve *S=NULL,*p1;
     c_tot=0;
     char c;
     for (i=0;i<num_nodi;i++){
            if (tipo_nodi[i]== 0 ) {
               p=G[i];
               while (p != NULL) {
                  if (tipo_nodi[p->v] > 0 ){
          //            printf ("\n Alla risorsa [%3d] assegno [%3d]  con
costo: %3d", i, p->v, p->c_orig);
                     c_tot=c_tot+p->c_orig;
                     p1 = malloc(sizeof(struct link_solve));
                     if (p1==NULL){
                        printf ("\n Errore grave!");
                        fflush(NULL);
                        system("PAUSE");
                     exit(5);
                     }
                     p1->orig=p->v;
                     p1->dest=i;
                     p1->costo=p->c_orig;
                     p1->next = S;
                     S = p1;
                  }
                  p = p->next;
               }
            }
     }

     S=bubble_sort(S);
     p1=S;
     i=0;
     c='A';
     k=1;
     temp=0;
//     temp=p1->orig;
     while ( p1 != NULL) {
        if (i <5 ){
             printf("\n P%d  ->  GSE ID: [%3d]          Costo %3d ] ", p1-
>orig + 1, p1->dest, p1->costo);
               temp=p1->orig;
        }else{

             if (k>PPLTASK) {
                c='A';
                k=1;
             }
             if ( (i-PPLTASK1) % PPLTASK == 0)
                printf("\n\n Task [%2d]", ((i-PPLTASK1)/PPLTASK)+1);
             printf("\n   [%3d] SubGrp %d[%c]  ->  GSE ID: [%3d] Type [%c]
Costo %3d ] ",p1->orig, k,c, p1->dest, c, p1->costo);
             while ( p1->next != NULL  && (p1->next)->orig == p1->orig ){
```

```
               p1 = p1->next;
               printf("\n   [%3d]              ->  GSE ID: [%3d]
Costo %3d ] ",p1->orig, p1->dest, p1->costo);
               //p1 = p1->next;
              }
              k++;
              c++;
        }
        temp=p1->orig;
           p1 = p1->next;
           i++;
      }
        printf ("\n\n Costo totale: %4d \n\n",c_tot);
    /*
    p1=S;
      while (p1 != NULL) {
            printf(" [%3d / %3d] -> ", p1->orig, p1->dest);
            p1 = p1->next;
      }
    */

     return;
}

/*
 *  Stampa Vettori l, pi, label (x debug)
 */

void stampa_vettori(int *l, int *pi,int *label, int n ) {
    int i;
    printf ("\n               [i]  -> ");
    for (i=0; i<n; i++)
         printf(" %3d ",i);

    printf ("\n Distanze     l[i]  -> ");
    for (i=0; i<n; i++)
         printf(" %3d ",l[i]);
    printf ("\n Predecessori pi[i] -> ");
    for (i=0; i<n; i++)
         printf(" %3d ",pi[i]);
    printf ("\n Etichette label[i] -> ");
    for (i=0; i<n; i++)
         printf(" %3d ",label[i]);
    printf("\n");
      return;
}

void aggiorna_costi(struct nodo *G[], int v, int *pg,int num_nodi) {
    struct nodo *p,*p1, *prec;
    int w,i;

    /* 1) aggiorno i costi degli archi uscenti dal nodo v (con pgreco
modificato)*/
    p=G[v];
    prec = NULL;
    while (p != NULL) {
      p->costo = p->c_orig -pg[v] + pg[p->v];
      //printf ("\n Nodo [%2d] -> [%3d]: pg[%2d] -> pg[%2d] con nuovo costo:
%3d (%3d)",v, p->v, pg[v] , pg[p->v], p->costo, p->c_orig);
```

```
      p = p->next;
    }
    /* 2) aggiorno i costi degli archi entranti al nodo v (con pgreco
modificato)*/
    for (i=0; i<num_nodi; i++) {
        if (i!=v){
            p=G[i];
            while (p != NULL) {
                  if (p->v == v){
                    p->costo = p->c_orig -pg[i] + pg[p->v];
                    //printf ("\n Nodo [%2d] -> [%3d]: pg[%2d] -> pg[%2d]
con nuovo costo: %3d (%3d)", i, p->v, pg[i] , pg[p->v], p->costo, p-
>c_orig);
                  }
            p = p->next;
            }
        }
    }



}
/* Attenzione: provenendo da Dijkstra, v1 > v2*/
void aggiorna_grafo(struct nodo *G[], int v1, int v2,int *pg) {
    struct nodo *p,*p1, *prec;
    int w=0, w_orig=0;
    int i=0;
    //printf ("\n v1= %d , v2=%d", v1,v2);
    /* Aggiorno arco v2->v1*/
    //printf ("\n Elimino  arco nodo[%2d] -> [%2d]",v2,v1);
    p=G[v2];
    prec = NULL;
    while (p != NULL) {
       if (p->v == v1) {
           if (p == G[v2]) {
           /* salvo il costo ridotto arco v1-v2 */
             w=p->costo;
             w_orig=p->c_orig;
           /* tolgo l'arco */
             G[v2] = G[v2]->next;
             free(p);
             p = G[v2];
          } else {
           /* salvo il costo ridotto arco v1-v2 */
             w=p->costo;
             w_orig=p->c_orig;
           /* tolgo l'arco */
             prec->next = p->next;
             free(p);
             p = prec->next;
          }
      } else {
        prec = p;
        p = p->next;
      }
    }
    /* Accodo il nuovo arco col costo ridotto*/
    //printf ("\n Aggiungo arco nodo[%2d] -> [%2d] con costo: %3d
(%3d)",v1,v2, w, w_orig);
```

```
    p1 = malloc(sizeof(struct nodo));
    p1->v = v2;
    p1->costo=w;
    p1->c_orig=w_orig;
    p1->next = G[v1];
    G[v1] = p1;
    return;
}
/*
 *  Funzione principale che implementa l'algoritmo di Dijkstra con costi
ridotti
 * l[i] -> etichetta del nodo generico i che rappresenta la lunghezza del
 *         miglior cammino minimo da s a i
 * pi[i] -> nodo che prece nel camminimo minimo il nodo i
 * label[i] -> se vale 1 il nodo i e' etichettato in modo permanente
 *
 * bi[i] -> potenziale pos= eccesso, zero=transito neg=deficit
 * nomi_nodi[i] -> vettori di stringhe di appoggio per i nomi dei nodi
 * pg[i] -> vettore con i pgreco dei nodi
 * pg1[i] -> vettore di appaggio per capire se cambiano i pgreco dei nodi
 * tipo_nodi[i] uso il vettore iniziale bi[i] per catalogare i nodi
 *              se <0 deficit se >0eccesso se == 0 transito
 *              serve per l'output
 */


int main(void) {
  int l[NNODI], pi[NNODI], label[NNODI], pg[NNODI], pg1[NNODI],
num_nodi,z,tipo_nodi[NNODI];
  int bi[NNODI];

  int i,s,u, min_dist, label_nodes,capacita,fine_dijk,dl,node_def,k,pg_old;
  struct nodo *p, *G[NNODI], *v, *G1[NNODI];
  time_t t1,t2;
  clock_t ticks1, ticks2;

  printf ("\n Programma in C per la Tesi");
  printf ("\n Autore: Fabio Bredo");
  printf("\n\n\n Dati iniziali \n");
  printf("\n  Num Tot Task   : %3d\n", NTASK);
  printf("\n  Ppl per Task1  : %3d\n", PPLTASK1);
  printf("\n  Ppl per TaskX  : %3d\n", PPLTASK);
  printf("\n  Tipologie GSE  : %3d\n", TIPORES);
  printf("\n  Num TipoA GSE  : %3d\n", TIPOA);
  printf("\n  Num TipoB GSE  : %3d\n", TIPOB);
  printf("\n  Num TipoC GSE  : %3d\n", TIPOC);
  printf("\n  Max Costo      : %3d\n", MAXCOSTO);
  printf("\n  Max Eccesso    : %3d\n", MAXBI);
  printf("\n\n");
  printf("\n  Num Tot Nodi   : %3d\n", NNODI);
  printf("\n  1 nodo GSE     : %3d\n", STARTGSE);
  printf("\n\n");
  printf("\n RICHIESTE per la Task Principale : \n");
  printf("\n  Num Tot TipoA   : %3d\n", TALFA);
  printf("\n  Num Tot TipoB   : %3d\n", TBETA);
  printf("\n  Num Tot TipoC   : %3d\n", TGAMMA);
  printf("\n\n");
  printf("\n\n Premere un tasto per iniziare del programma \n");
```

```
  getch();
  num_nodi = NNODI;
/*
 * Inserisco il grafico
 */


   leggi_grafo_dinamico(G,bi);

/*
 * Stampo il grafico (la lista di adiacenza con i costi)
 */

  printf ("\n Ecco la lista di adiacenza: \n");
  for (i=0; i<num_nodi; i++) {
            printf("\n %d: %d - ", i, bi[i]);
            stampa_lista(G[i]);
      }

  getch();

/*Memorizzo l'ora di inizio */
  t1=time(NULL);

  ticks1=clock();

  /* pgreco */
  for (i=0; i<num_nodi; i++) {
      pg[i]=0;
      pg1[i]=0;
      }
/*
 * Controllo se ci sono nodi con deficit bi<0
 */
  z=0;
  node_def=0;
  for (i=0; i<num_nodi; i++) {
      tipo_nodi[i]=bi[i];
      z=z+bi[i];
      if (bi[i] < 0)
          node_def=node_def + bi[i] ;
  }

  if (z!=0){
       printf ("\n Attenzione sbilanciamente nodi con deficit / eccesso (
Sum[bi_i] != 0 ) \n");
         for (i=0; i<num_nodi; i++)
            printf("\n %d: %d - ", i, bi[i]);
       fflush(NULL);
       system("PAUSE");
       exit(6);
     }

  k=0;
/* ---------------------------------------------------------------------
 * Finche ci sono nodi con bi<0
 * ---------------------------------------------------------------------
 */
```

```
  while (  node_def < 0){
/*
 * Seleziono un nodo sorgente con bi>0
 */

  s=-1;
  for (i=0; i<num_nodi; i++) {
      if (bi[i] > 0)
          s=i;
  }
  //printf ("\n\n Nodo sorgente : %3d con eccesso : %3d  \n", s, bi[s]);


/*
 * Passo 1: Inizializzazione
 */
  /* l distanze*/
  for (i=0; i<num_nodi; i++)
      l[i]=INFINITO;

  l[s]=0;
  /* pi predecessori*/
  for (i=0; i<num_nodi; i++)
      pi[i]=0;


  pi[s]=s;

  /* label: ettichette*/
  for (i=0; i<num_nodi; i++)
      label[i]=0;
  /* Etichetto permanente il nodo u (sorgente) */
  u=s;
  //label[u]=1;


   fine_dijk=0;

   while (  fine_dijk == 0){
/*
 * Passo 2: Revisioni delle etichette
 */
     p = G[u];
     while (p != NULL)  {
          if (  label[p->v]==0 )
                if ( l[p->v] > l[u]+p->costo){
                   pi[p->v] = u;
                   l[p->v] = l[u] + p->costo;
                }
          p = p->next;

     }// fine while
/*
 * Passo 3: Determinazionedi una ettichetta permanente
 */
    min_dist=INFINITO;
    for (i=0; i<num_nodi; i++)
          if (  label[i]==0 && l[i] < min_dist ){
                min_dist =  l[i];
```

```
                u=i;
            }
        /* Etichetto permanente il nodo u*/
        label[u]=1;
        /* Se etichetto un nodo con deficit*/
        if (bi[u] < 0){
            //printf ("\n Trovato nodo destinatario : %3d con difetto : %3d
\n\n", u, bi[u]);
           /* fine dijksta*/
           fine_dijk++;
           /* sistemo i bi(s) e bi(u) */
           bi[s]=bi[s] - 1;
           bi[u]=bi[u] + 1;
           dl=l[u]; /* serve per aggiornare i pgreco */
           node_def++;
        }

 }//fine while   dijkst
 i=u;
  //  printf ("\n\n Fine Dijkstra \n");
 /*
    printf ("\n Percorso a ritroso nodo(costo): \n");
    while (i!=s)  {
           printf ( " %3d (%3d)", i, l[i] );
           i=pi[i];
    }
    printf ( " %3d (%3d)", i, l[i] );

   printf ("\n\n Nodi etichettati in modo definitivo : ");
   for (i=0; i<num_nodi; i++)
        if (label[i] == 1)
           printf("\n label[%d]: %d ", i, label[i]);
   */
  //stampa_vettori(l,pi, label, num_nodi);


    /* Aggiorno i pgreco dei nodi etichettati in modo definitivo */
    i=u;

  /* pgreco pg/pg1 controllo se campio il pgreco dei nodi*/
    for (i=0; i<num_nodi; i++)
      pg1[i]=0;

    //printf ("\n\n Aggiornamento pgreco ");


     for (i=0; i<num_nodi; i++)
        if (label[i] == 1) {
           pg_old=pg[i];
           pg[i]=pg[i]-l[i]+dl;
           //printf ("\n pgreco nodo[%2d]=%2d",i,pg[i]);
           if (pg_old != pg[i])
                pg1[i]=1;
        }


    //printf ("\n\n Aggiornamento costi ridotti ");
    /* Aggiorno il grafo con i costi ridotti di tutti gli archi
       entranti e uscenti da un nodo in cui ho cambiato il pgreco
```

```
    */
    for (i=0; i<num_nodi; i++)
       if (pg1[i]==1) {
          aggiorna_costi (G,i,pg,num_nodi);
       }

    //printf ("\n\n Aggiornamento del grafo ");
    /* Aggiorno il grafico  cambiando gli archi */
    i=u;

    while (i!=s)  {
           aggiorna_grafo(G,i, pi[i], pg);
           i=pi[i];
    }



//    printf("\n\n Fine iterazione %d",k);
//    printf("\n ------------------------------------------------------------
-\n");
      //getch();
      k++;

}//fine while node_def <0

    printf ( "\n\n\n Risultato finale : \n");
      stampa_soluzione(G,tipo_nodi,num_nodi );

    /* Stampo il tempo di esecuzione */
      //(void) time(&t2);
      t2=time(NULL);
      ticks2=clock();
    printf("\n Cicli di Clock CPU %ld [CLOCKS_PER_SEC = %ld].\n",ticks2-
ticks1, CLOCKS_PER_SEC);
      printf("\n Tempo impiegato = %d  Secondi" ,(int)t2-t1);
      getch();
  return(1);
}
```

Glossario

MCF (**Min Cost Flow):** Flusso a costo minimo

GPRS: General Packet Radio Service (GPRS) è una delle tecnologie di telefonia mobile.

Wi-fi: abbreviazione di Wireless Fidelity, è un termine che indica dispositivi che possono collegarsi a reti locali senza fili (WLAN) basate sulle specifiche IEEE 802.11.

GSE: Ground Service Equipment

AAS : Integrated Airport Apron Safety Fleet Management

WP: work package

Cenni Bibliografici

- AHUJA, R.K., MAGNANTI, T.L. e ORLIN, J.B. : "***Network Flows: Theory, Algorithms and Applications***", Presence Hall, Eaglewood Cliffs (NJ, USA), 1993
- DIJKSTRA, E.W.: "***A Note on Two Problems in Connection with Graphs***" Numerical Mathematics, 1959
- G. ANDREATTA, F. MASON, G. ROMANIN JACUR : "***Ottimizzazione su Reti***", Libreria Progetto, Padova, seconda edizione, 1996.
- AAS Project (http://www.aas-project.eu/)