



TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA DELLA PRODUZIONE

TESI DI LAUREA

**Analisi dei costi e fairness in problemi
di Vehicle Routing**

RELATORE:

Giuseppe Stecca

CORRELATORE:

Diego Maria Pinto

CANDIDATO:

Lorenzo Pucci

MATR. 0286098

Anno Accademico 2021/2022

Introduzione.....	1
Capitolo 1 – Waste management e il caso di studio	3
Capitolo 2 – Metodi di allocazione costi nel vehicle routing	7
<i>Caso applicativo</i>	8
A) <i>Isolated Cost Allocation (ICA).....</i>	9
B) <i>Neighbors Savings (NS)</i>	11
C) <i>Normalized Marginal Allocation (NMA).....</i>	16
<i>Analisi della soluzione proposta.....</i>	17
Considerazioni su altri esempi	19
Capitolo 3 – Applicazione dello Shapley value all’allocazione fair dei costi	23
<i>Storia dello Shapley value</i>	23
<i>La forma della Shapley value</i>	25
Nota sulla complessità 1	27
A) <i>ApproShapley</i>	28
Applicare ApproShapley ad un problema di TSP	31
B) <i>ApproShapley $O(1)$ – prima approssimazione</i>	34
Airport problem – caso monodimensionale	34
Nota sulla complessità 2	38
La forma di Appro $O(1)$ nel caso multidimensionale.....	39
C) <i>ApproShapley $O(2)$ – seconda approssimazione.....</i>	44
Capitolo 4 – Algoritmo di calcolo fairness per problemi di routing.....	56
<i>Programma – VRP_cost_SOLVER.....</i>	57
<i>Programma – VRP_v_kk</i>	59
<i>Programma – VRP_cost_allocation_SHAPLEY.....</i>	60
<i>Programma – VRP_s_kk.....</i>	62
<i>Programma – VRP_cost_allocation_appro_SHAPLEY_O1</i>	63
<i>Programma – VRP_cost_allocation_appro_SHAPLEY_O2</i>	65
Capitolo 5 – Confronto tra Shapley value e metodi “average”	69
<i>Analisi dei risultati</i>	71
Appro $O(1)$ vs Appro $O(2)$	73
<i>Limiti Shapley e plot dei tempi</i>	73
Nota sul numero di operazioni a cliente.....	74
<i>Allocazione costi e strategie di prezzo.....</i>	74
Conclusioni.....	77
Bibliografia.....	80
Ringraziamenti	82

Introduzione

Il presente elaborato si sviluppa all'interno del progetto PIPER - Piattaforma intelligente per l'ottimizzazione di operazioni di riciclo, supportato dal programma EU POR FESR di Regione LAZIO su "Gruppi di Ricerca 2020", approfondendo il concetto di *fair cost allocation*, con lo scopo di analizzare e comprendere come uno strumento come lo Shapley value possa essere sfruttato in questo contesto.

Con *fair cost allocation* si fa riferimento ad un metodo di distribuzione dei costi coinvolti nella fornitura di un servizio (o nella produzione di un prodotto) tra le parti interessate in modo equo e ragionevole. L'obiettivo del *fair cost allocation* è distribuire i costi, come le spese generali, la manodopera e i materiali, così da riflettere accuratamente il modo in cui i costi sono stati sostenuti e garantire che ciascun partecipante o stakeholder paghi una quota equa e ragionevole dei costi totali. Il metodo di equa allocazione dei costi può variare a seconda della natura dei costi stessi, dei servizi e/o dei prodotti interessati e può utilizzare vari fattori/metodi di allocazione per la loro distribuzione. L'allocazione equa dei costi viene spesso utilizzata in contesti in cui più parti contribuiscono ad un unico obiettivo, come nei servizi di ride sharing o nella progettazione di reti.

Il caso di studio preso in esame si colloca nell'ambito della logistica e dei trasporti. Nell'elaborato di partenza: "*Customer cost forecasting through patterns learning of optimal capacity allocation*" by (Diego Maria Pinto, Marco Boresta, e Giuseppe Stecca), l'obiettivo era individuare un nuovo modello di programmazione lineare per l'ottimizzazione dei percorsi di pianificazione per i servizi di ritiro e consegna dei rifiuti. Lo studio consiste nella gestione della catena di approvvigionamento e in letteratura è noto come Vehicle Routing Problem (VRP), ovvero un problema di ottimizzazione combinatoria NP-Hard. È stata presentata una formulazione, che non viene approfondita in questo lavoro, che consente di individuare l'insieme ottimale di percorsi per una singola flotta di camion che, partendo da un deposito, prelevano prodotti da smaltire da un dato insieme di clienti e li consegnano ad un altro dato insieme di nodi delivery, per poi fare ritorno al deposito di partenza. La soluzione soddisfa contemporaneamente i vincoli del sistema di rete, i vincoli della capacità di trasporto e quelli associati a ciascun cliente e luogo di consegna. Può essere vista come una variante del VRP con ritiri e consegne (VRPPD). In breve, se si considera una richiesta di servizio logistico da un cliente i , questa è identificata da due nodi, i e $n + i$, rispettivamente le fermate di ritiro (Pickup) e consegna (Delivery) della richiesta. È possibile che diversi nodi possano rappresentare la stessa posizione geografica. Quindi sono stati indicati l'insieme dei nodi di prelievo con $P = \{1, \dots, n\}$, l'insieme dei nodi di consegna con $D = \{n + 1, \dots, 2n\}$ e si consideri K come l'insieme dei veicoli.

Questa tesi, in particolare, tende a dimostrare come l'applicazione di approssimazioni significative dello Shapley Value alla *fair cost allocation* in un problema di VRPPD nell'ambito del Waste Management, possa contribuire a determinare una più precisa individuazione dei costi variabili per servizi di ritiro a clienti, se preordinati per aree geografiche di appartenenza.

La struttura del lavoro si suddivide in cinque capitoli che possiamo riassumere come segue.

1. Il primo capitolo "Waste management e il caso di studio" inquadra il contesto di riferimento e il problema VRPPD applicato ad un caso di studio reale, la Innocenti S.r.L., precedentemente analizzato nel progetto PIPER.
2. Il secondo capitolo "Metodi di allocazione costi nel vehicle routing" analizza la proposta di Cost Allocation fornita da Diego Maria Pinto, Marco Boresta, e Giuseppe Stecca nell'elaborato "*Customer cost forecasting through patterns learning of optimal capacity allocation*".
3. Il terzo capitolo "Applicazione dello Shapley value all'allocazione fair dei costi" approfondisce l'analisi applicata dello Shapley Value puro nonché le approssimazioni proposte da Dan C. Popescu e Philip Kilby al problema di VRPPD
4. Il successivo capitolo "Algoritmo di calcolo fairness per problemi di routing" effettua una trasposizione dei risultati emersi dall'attività illustrata nel capitolo precedente in un algoritmo di calcolo finalizzato allo studio della fairness in problemi di routing.
5. Infine, il quinto capitolo "Confronto tra Shapley value e metodi average" espone e illustra la comparazione dei risultati a conferma della tesi avanzata, ai fini dell'utilizzo dello Shapley Value come valida alternativa nella politica di cost allocation.

I risultati del presente lavoro testimoniano in conclusione come sia possibile utilizzare le approssimazioni dello Shapley Value nel VRPPD riducendo i tempi computazionali e mantenendo al tempo stesso pressoché inalterata la precisione di output. Viene inoltre fornito un possibile spunto per utilizzare gli stessi dati nella tariffazione dei servizi di pick up per clienti in base al loro posizionamento geografico.

Capitolo 1 – Waste management e il caso di studio

Il Waste Management è il processo di raccolta di trasporto di lavorazione e di smaltimento dei materiali di scarto. Il suo obiettivo è ridurre l'impatto negativo dei rifiuti sull'ambiente e sulla salute del pianeta. Ampilissimo è lo spettro di generatori di rifiuti. Si passa dagli individui alle famiglie, nonché alle industrie, alle istituzioni pubbliche e private e a organizzazioni di ogni genere.

Una delle principali sfide nella gestione dei rifiuti è rappresentata dall'esigenza di efficientare la loro raccolta e il loro trasporto dai punti di prelievo a quelli di consegna. I mezzi principalmente utilizzati sono camion (truck), che passano attraverso diversi punti prestabiliti lungo un itinerario da determinarsi di volta in volta. Tale processo, tuttavia, può risultare piuttosto inefficiente e costoso se non pianificato ed eseguito correttamente. È qui che entra in gioco il problema del percorso del Vehicle Routing Problem con nodi di ritiro (Pickup) e consegna (Delivery), abbreviato: VRPPD.

Il VRPPD è un noto problema di ottimizzazione complesso che implica la ricerca dei percorsi ottimali per una flotta di veicoli per il ritiro e la consegna degli articoli in posizioni specificate. Nel corso degli ultimi anni questo processo ha guadagnato attenzione da parte di diversi ricercatori e molti sono stati gli algoritmi proposti per risolverlo, tra cui algoritmi esatti, euristiche e metaeuristiche. Tuttavia, trovare la soluzione ottimale per istanze su larga scala del VRPPD risulta ancora oggi un compito impegnativo. In caso di Waste Management, gli articoli da raccogliere e consegnare sono materiali di scarto e i luoghi sono abitazioni o altre fonti di produzione di rifiuti. L'obiettivo del VRP con nodi di Pickup e Delivery è ridurre al minimo la distanza totale percorsa dai truck, garantendo al contempo che tutte le richieste di ritiro e consegna siano soddisfatte.

Il problema di ottimizzazione presenta diverse criticità di costo nella gestione dei rifiuti. Uno di quelli più critici è il costo operativo del veicolo, che include il carburante, la manutenzione e la manodopera. Instradamento e programmazione inefficienti dei veicoli possono aumentare significativamente i costi operativi. Un altro fattore di costo è rappresentato dal processo di smaltimento, che prevede anche il trasporto dei materiali di scarto alle discariche o ai centri di riciclaggio. Se il processo di raccolta dei rifiuti non è ottimizzato correttamente, il costo di smaltimento può essere elevato quasi quanto il costo operativo.

L'impatto ambientale della gestione dei rifiuti può essere anch'esso un fattore significativo nell'equazione dei costi. Se i materiali di scarto non vengono raccolti e smaltiti correttamente, si rischia una notevole ricaduta sull'ambiente, portando a maggiori rischi per la salute delle comunità circostanti. Un processo di raccolta dei rifiuti mal pianificato può anche comportare un aumento della

congestione del traffico, dell'inquinamento acustico e dell'inquinamento atmosferico, con conseguenti costi economici e impatti ambientali significativi.

Sulla base di vari risultati di ricerca, tra i punti di forza del VRPPD (Vehicle Routing Problem with Pickups and Deliverys) troviamo la capacità di ottimizzare i percorsi di una flotta di veicoli per il ritiro e la consegna. Ciò comporta un potenziale vantaggio economico per i mercati più aperti. Di contro, i punti deboli del VRPPD risiedono nella sua stessa natura; infatti, spesso risultano essere processi più complessi rispetto ad altre varianti del Vehicle Routing Problem e possono creare delle limitazioni nella selezione del percorso. Per risolverli in modo efficiente potrebbero addirittura rendersi necessari algoritmi e software di ottimizzazione avanzati.

È chiaro come il VRPPD sia un'area di ricerca critica nella gestione dei rifiuti, con notevoli problemi di costo. Ottimizzando il percorso dei veicoli e il processo di programmazione, le organizzazioni di gestione dei rifiuti possono ridurre significativamente i costi operativi e di smaltimento. Si ridurrebbero così al minimo i rischi di impatto ambientale. In sintesi, il VRP con nodi di ritiro e consegna è un potente strumento che può aiutare a mitigare i rischi intrinseci ad una materia estremamente attuale come la problematica della gestione dei rifiuti.

Il presente lavoro di tesi si inserisce nell'elaborato di ricerca "*Customer cost forecasting through patterns learning of optimal capacity allocation*" by (Diego Maria Pinto, Marco Boresta, e Giuseppe Stecca) che analizza il problema del VRPPD in un caso di studio reale, in appoggio all'azienda Innocenti S.r.L. di Roma. L'azienda faceva parte di un precedente progetto di ricerca denominato "*Remind*" e le sue coordinate geografiche [41.96007582971499, 12.764351670262428] sono da considerare il punto di partenza e di rientro della flotta di truck disponibili per la soluzione ottimale del problema.

L'elaborato sopra citato ha fornito una possibile soluzione pratica al problema di VRPPD, generando come output una proposta di allocazione delle risorse (in questo caso sette trucks), come una serie di istanze di TSP. Ai fini della presente tesi, dall'elaborato è stato estratto un esempio basato sulle richieste di servizio da parte di sette clienti il giorno 2020/03/13 ciascuno con il proprio nodo di Pickup e Delivery.

Importante notare come in questo caso i nodi di consegna spesso coincidano tra di loro, in quanto la consegna viene effettuata presso uno stesso punto di raccolta. Altro dato interessante risulta essere che il deposito della Innocenti S.r.L. si trova nei pressi dei pochi nodi di Delivery previsti dall'iter.

Per risolvere il problema di VRPPD associato alla data presa in esame, l'algoritmo proposto lo scompone in quattro sottoinsiemi di TSP per giungere ad una delle soluzioni ottimali possibili.

Cliente	Pickup name	lat_P	long_P	Delivery name	lat_D	long_D
1	2	42.1320716	12.5839994	226	41.9555557	12.7643387
2	14	41.9687393	12.686	226	41.9555557	12.7643387
3	94	44.5252388	11.1757875	226	41.9555557	12.7643387
4	91	42.3369153	13.4628064	226	41.9555557	12.7643387
5	238	40.9601508	14.488986	226	41.9555557	12.7643387
6	9	41.0128754	14.3201006	226	41.9555557	12.7643387
7	223	41.0152357	14.2977433	226	41.9555557	12.7643387

Tabella 1.1 – Clienti da servire il 2020/03/13 con coordinate

Ogni problema di TSP risolto corrisponde all'utilizzo di un truck k per visitare tutti i nodi della soluzione. Il risultato che l'algoritmo genera sarà:

- Truck 1: serve i clienti 1 e 2 con percorso (D – 14 – 2 – 226 – D) e km tot: 84.86 km
- Truck 2: serve i clienti 3 e 4 con percorso (D – 94 – 91 – 226 – D) e km tot: 899.06 km
- Truck 3: serve solo il cliente 5 con percorso (D – 238 – 226 – D) e km tot: 412.35 km
- Truck 4: serve i clienti 6 e 7 con percorso (D – 223 – 9 – 226 – D) e km tot: 380.53

Si riporti in figura [1.1] la soluzione su mappa del problema di VRPPD sotto forma di quattro problemi di TSP.

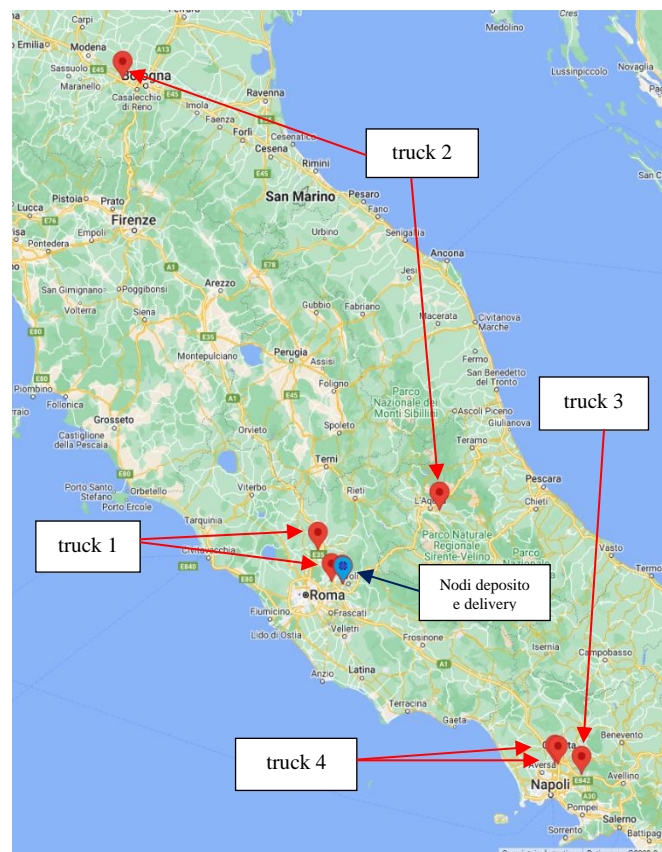


Figura 1.1 – Localizzazione clienti da servire con assegnazione dei truck per il ritiro e la consegna

La somma dei km percorsi dai vari truck corrisponde alla soluzione ottimale del problema di VRPPD iniziale, per la data del 2020/03/13.

Una volta risolto il problema di ricerca operativa (OR), viene proposta una Customer Cost Allocation Strategy finalizzata ad imputare a ciascun cliente servito la propria porzione di costo equa, al fine di stabilire il prezzo del relativo pagamento. In termini di equità e prestazioni di regressione attese, la strategia di allocazione dei costi è una delle parti più importanti del lavoro. Affinché il modello di previsione possa essere “addestrato” per fornire stime accurate dei costi dei clienti, i costi devono essere allocati in modo equo a ciascun cliente sia negli scenari di produzione che di servizio. Uno degli obiettivi di risulta è poter formulare offerte commerciali competitive ai propri clienti.

È in questo ambito che il presente lavoro di tesi si sviluppa, adottando metodi di Fair Cost Allocation che possano essere il più possibile equi. Viene riportata la figura [1.2] ripresa dal progetto “*Customer cost forecasting through patterns learning of optimal capacity allocation*” per illustrare il processo iterativo e il punto di inserimento di questo elaborato.

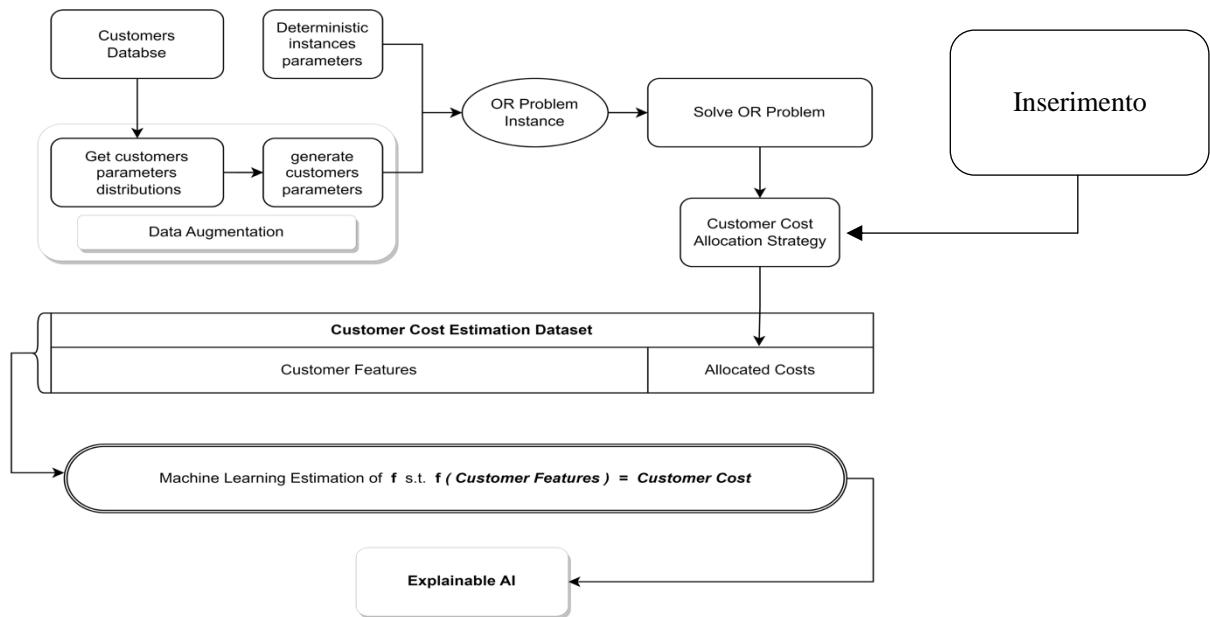


Figura 1.2 – Quadro di apprendimento sui modelli di costo del cliente

Capitolo 2 – Metodi di allocazione costi nel vehicle routing

Una volta risolto il problema di allocazione della capacità, è necessario definire una regola che possa ripartire il costo per il servizio (in questo caso di trasporto) tra tutti i clienti in modo equo. Una delle principali problematiche nella logistica dei trasporti è definire quanto del costo totale del percorso complessivo di un determinato veicolo è giusto imputare ad ogni cliente da servire.

Trovare una soluzione che avalli tutte le caratteristiche che possano influenzare la ripartizione del costo non è di facile risoluzione. Lo stato dell'arte, se così si può intendere, in letteratura consiste nell'applicazione dello Shapley value, teorizzato dall'omonimo nel 1953. Lo Shapley value $\phi_i(v)$ si presenta come il *contributo marginale* medio di un giocatore i in una specifica coalizione, rispetto le possibili permutazioni dei giocatori e permette di calcolare la convenienza che questi può avere nell'entrare in un "gioco". Nell'ambito del commesso viaggiatore (TSP), lo Shapley value permette di calcolare una giusta ripartizione dei costi, per ogni giocatore i , basandosi appunto sul loro *contributo marginale*. Nonostante questa sia la soluzione migliore dal punto di vista della qualità, è invece la peggiore per quanto riguarda la complessità. Lo Shapley value, infatti, opera su una complessità computazionale dell'ordine di $O(n!)$, il che vuol dire che se già il numero di clienti da servire è composto da dieci o più elementi, questo rende possibile calcolarlo in tempo polinomiale, ma solo nei casi in cui il numero di giocatori non è elevato. Sono numerosi gli autori che hanno elaborato soluzioni alternative che si propongono di cercare un'approssimazione che sia un buon compromesso tra qualità e complessità, ma non saranno oggetto di trattazione in questo articolo.

L'elaborato di ricerca originale ("*Customer cost forecasting through patterns learning of optimal capacity allocation*") ha proposto una strategia di allocazione dei costi al di fuori del paradigma dello Shapley value, che è stata applicata ad ogni singolo problema di TSP derivante dal problema di VRPPD iniziale. Nel singolo TSP ogni cliente è composto da un punto di ritiro P_i e da uno di consegna D_i . È stata inoltre considerata la quantità di clienti da servire che è stata semplificata in quanto giudicata troppo elevata.

Per l'allocazione dei costi sono state individuate tre regole distinte, ciascuna con aspetti positivi e negativi, in grado di compensarsi tra di loro. È stata individuata come soluzione finale una media ponderata dei rispettivi risultati denominata *Average method*:

- Regola *Isolated Cost Allocation*
- Regola *Neighbors Savings*
- Regola *Normalized Marginal Allocation*

Caso applicativo

Nei paragrafi successivi, una volta argomentate le tre regole di allocazione dei costi, queste verranno applicate al seguente esempio. Il problema del VRP ha generato un sotto problema di TSP, definendo il percorso minimo per visitare tutti i clienti. Si considerino quindi n clienti ($n = 1, 2, 3$), un deposito, nodi pickup P_i ($i = 1, 2, 3$), nodi delivery D_i ($i = 4, 5, 6$) e una matrice delle distanze che esprime il costo come somma di distanza e durata tra i nodi.

	Dep (o/d)	P ₁	P ₂	P ₃	D ₄	D ₅	D ₆
Dep (o/d)	/	8	13	31	19	28	5
P ₁	8	/	7	22	14	15	11
P ₂	13	7	/	25	10	27	14
P ₃	31	22	25	/	41	37	27
D ₄	19	14	10	41	/	13	28
D ₅	28	15	27	37	13	/	19
D ₆	5	11	14	27	28	19	/

Tabella 2.1 – Matrice costo (distanza/durata)

La soluzione che l'algoritmo descritto nell'appendice A propone di servire ogni cliente i dell'istanza con un unico veicolo k , e risulta essere quella rappresentata in figura. L'obiettivo è quindi ripartire in maniera equa il costo totale tra i 3 clienti i . Per farlo useremo le 3 regole di allocazione, ma prima definiamo:

- $x_{i,j} \in \{0,1\}$: uguale a 1 se il veicolo k attraversa arco (i, j) e 0 altrimenti.
- $C_{i,j}$: costo percorrenza arco (i, j)
- n = numero di clienti da servire
- $P = \{1, \dots, n\}$: set di nodi pickup
- $D = \{n+1, \dots, 2n\}$: set di nodi delivery
- o = deposito come partenza
- d = deposito come arrivo
- $C_{i,n+i}$ = costo dell'arco tra il nodo P_i e il nodo D_{n+i}

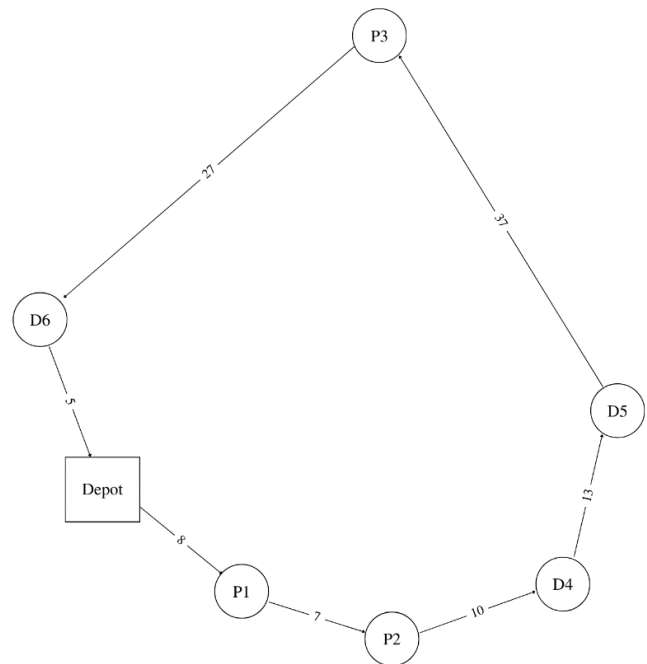


Figura 2.1 – Grafo percorso minimo soluzione del TSP

Il costo totale per servire tutti i clienti i , è pari a:

$$C_{\text{tot}} = \sum_{i \in \text{Nod}} \sum_{j \in \text{Nod}} C_{i,j} x_{i,j} = 107\text{€}$$

A) Isolated Cost Allocation (ICA)

Come punto di partenza è stato preso come benchmark di riferimento una regola che potesse garantire una prima soluzione ammissibile, da migliorare in seguito. La regola *Isolated Cost Allocation* assegna i costi agli n clienti in proporzione al percorso che si origina dal deposito al nodo di prelievo P_i con successiva consegna al nodo D e rientro finale al deposito. Si è optato per uno scenario non cooperativo, nel quale ogni singolo tour venisse considerato isolatamente, senza considerare eventuali riduzioni di costi imputabili alla vicinanza tra i clienti da servire. L'ipotesi alla base consiste nella normalizzazione di tutte le distanze percorse, alle quali è associato un costo proporzionale sia alla distanza che alla durata.

È possibile definire il costo associato ad ogni cliente i (con i che vada 1 a n), come distanza di ogni cliente (composto da un nodo pickup e un nodo delivery) dal deposito.

$$C_i = C_{o,P_i} + C_{P_i,D_{n+i}} + C_{D_{(n+i)},d}$$

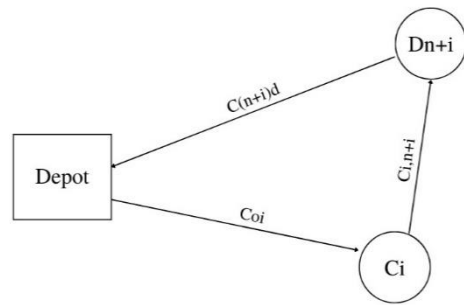


Figura 2.2 – Generico percorso per servire cliente i

Una volta calcolati tutti i costi associati alle distanze per ogni cliente i , vanno normalizzati al fine di individuare la porzione del costo totale da imputare a ciascuno di loro.

$$\%C_{ISA} = \frac{C_i}{\sum_{i=1}^n C_i} = \frac{C_{o,P_i} + C_{P_i,D_{n+i}} + C_{D_{(n+i)},d}}{\sum_{i=1}^n (C_{o,P_i} + C_{P_i,D_{n+i}} + C_{D_{(n+i)},d})}$$

Per ogni cliente i , la porzione di costo del costo totale, secondo la regola *Isolated Cost Allocation* risulta:

$$C_{ISA} = \%C_{ISA} * C_{tot}$$

➤ **Esempio:**

Si riprenda l'esempio riportato nel “*Caso applicativo*”, nel quale l'obiettivo è ripartire il costo totale per servire tutti i clienti i:

$$C_{tot} = \sum_{i \in Nod} \sum_{j \in Nod} C_{i,j} * x_{i,j} = 107€$$

Si utilizza la regola *Isolated Cost Allocation* per imputare a ciascun cliente i, una quota del costo totale, in funzione della distanza di ogni cliente dal deposito.

$$\%C_{ISA} = \frac{C_{o,P_i} + C_{P_i,D_{n+i}} + C_{D_{(n+i),d}}}{\sum_{i=1}^n (C_{o,P_i} + C_{P_i,D_{n+i}} + C_{D_{(n+i),d}})}$$

$$\sum_{i=1}^n (C_{o,P_i} + C_{P_i,D_{n+i}} + C_{D_{(n+i),d}})$$

$$(C_{o,P_1} + C_{P_1,D_4} + C_{D_4,d}) + (C_{o,P_2} + C_{P_2,D_5} + C_{D_5,d}) + (C_{o,P_3} + C_{P_3,D_6} + C_{D_6,d})$$

$$(41) + (68) + (63) = 172$$

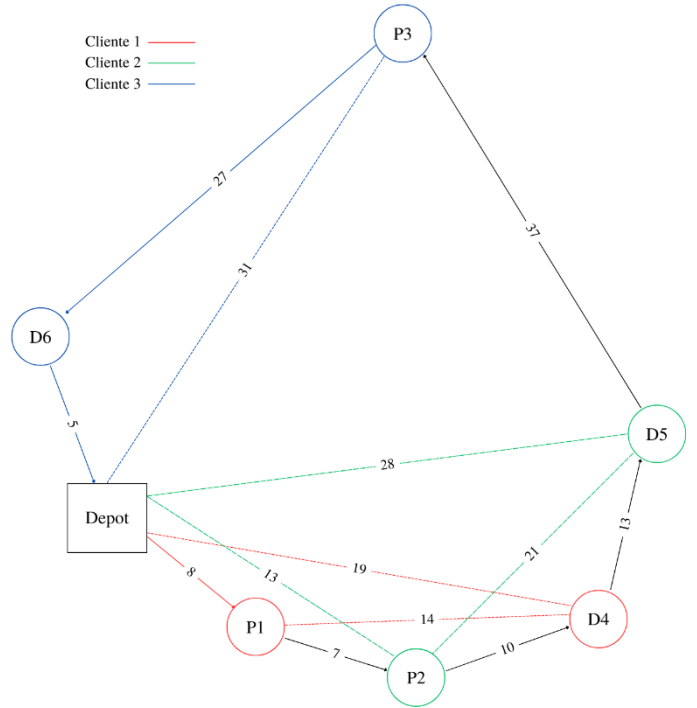


Figura 2.3 – Grafo soluzione ICA

Mentre il costo imputabile a ciascun cliente risulta essere

Cliente 1	Cliente 2	Cliente 3
$C_1 = C_{o,P_1} + C_{P_1,D_4} + C_{D_4,d} = 41$	$C_2 = C_{o,P_2} + C_{P_2,D_5} + C_{D_5,d} = 68$	$C_3 = C_{o,P_3} + C_{P_3,D_6} + C_{D_6,d} = 63$
$\%C_{1SA} = 41/172 = 24\%$	$\%C_{2SA} = 68/172 = 40\%$	$\%C_{3SA} = 63/172 = 37\%$
$C_{1SA} = \%C_{1SA} * C_{tot} = 26 €$	$C_{2SA} = \%C_{2SA} * C_{tot} = 42 €$	$C_{3SA} = \%C_{3SA} * C_{tot} = 39 €$

Tabella 2.2 – Risultati ICA

In questo scenario, basato unicamente sulle distanze tra il deposito e i punti di prelievo e consegna di ciascun cliente, sono i clienti 2 e 3 quelli che si vedono imputare un costo maggiore. In particolare, se spostiamo il focus sul cliente 2, si evince come a questo venga imputato il costo maggiore, senza considerare la sua distanza tra i nodi prelievo e consegna del cliente 1. Per cui è pure vero che la sua

distanza totale dal deposito (espressa in termini di costo) è 68, ovvero la maggiore tra le tre, ma è anche vero che questo si trova a metà strada tra P_1 e D_4 . Non viene sostanzialmente percepito il beneficio derivante dal fatto di essere vicini ad altri clienti da servire.

Come si evince, la regola *Isolated Cost Allocation* ha il grande vantaggio della semplicità di utilizzo e della bassa complessità computazionale, ma questo rappresenta anche il suo principale elemento critico. E' infatti evidente l'eccessiva semplicità del modello che di fatto impedisce di considerare la riduzione dei costi che si potrebbe ottenere per clienti vicini tra loro. Questo aspetto di contro, viene considerato nelle altre due regole proposte in seguito.

B) Neighbors Savings (NS)

Per compensare il difetto della regola *Isolated Cost Allocation*, ovvero il non considerare la vicinanza tra i clienti, l'idea alla base è caratterizzata, partendo da un cliente i , dal calcolo del suo contributo rispetto al costo totale, escludendolo dalla distanza totale percorsa “tagliando” e “ricucendo” il passaggio. Dato il cliente i , vengono rimossi tutti gli archi adiacenti ai nodi P_i e D_i del tour considerato, e aggiunti tutti gli archi che permettono al tour stesso di bypassare questi nodi. Così facendo, maggiore sarà la differenza tra la vicinanza dei nodi e l'effettiva distanza dal deposito, minore risulterà il costo marginale nel visitare P_i e D_i .

- C_{tot} : rappresenta la distanza totale percorsa da un veicolo k che visita il cliente i nel suo tour.

$$C_{tot} = \sum_{i \in Nod} \sum_{j \in Nod} C_{i,j} x_{i,j} = 107\text{€}$$

- $C_{deviation}$: rappresenta la somma di tutte le distanze degli archi percorsi per visitare sia P_i che D_i , come deviazione dal tour del veicolo k , senza passare per il cliente i . In sintesi, è la somma di tutte le distanze associate agli spigoli del grafo, attraversati da k , che hanno P_i o D_i come nodo in partenza o in arrivo.

Vengono presi, per ogni cliente i , P_i, D_{n+i} , per calcolare C_{dev} per ogni cliente i . Si faccia riferimento alla variabile booleana $X_{i,j}$ del generico arco

$$C_{dev} = \sum_{l=0}^{2n+2} C_{l,P_i} X_{l,P_i} + \frac{\sum_{l=0}^{2n+2} C_{P_i,l} X_{P_i,l} + \sum_{l=0}^{2n+2} C_{l,D_{n+i}} X_{l,D_{n+i}}}{1 + X_{P_i,D_{n+i}}} + \sum_{l=0}^{2n+2} C_{D_{n+i},l} X_{D_{n+i},l} \quad \forall i \in n$$

Nota: $1 + X_{P_i,D_{n+i}}$ consente di risolvere il caso in cui si abbia il nodo pickup e delivery dello stesso cliente, uno di seguito all'altro in soluzione.

- C_{link} : rappresenta la somma di tutte le distanze associate ai bordi del grafo che il veicolo k dovrebbe attraversare per completare il suo tour senza visitare P_i e D_i e rispettando l'ordine di visita dei nodi della soluzione originale. Riprendendo la notazione precedente, ma con l che rappresenta i nodi antecessori, mentre h i successori:

$$C_{link} = \left[\sum_{l=0}^{2n+2} \sum_{h=0}^{2n+2} C_{l,h} (X_{l,P_i} * X_{P_i,h}) + \sum_{l=0}^{2n+2} \sum_{h=0}^{2n+2} C_{l,h} (X_{l,D_{n+i}} * X_{D_{n+i},h}) \right] (1 - X_{P_i,D_{n+i}}) \\ + \left[\sum_{l=0}^{2n+2} \sum_{h=0}^{2n+2} C_{l,h} (X_{l,P_i} * X_{D_{n+i},h}) \right] (X_{P_i,D_{n+i}}) \quad \forall i \in n$$

Nota: $(1 - X_{P_i,D_{n+i}})$ e $(X_{P_i,D_{n+i}})$ consentono di risolvere il caso in cui si abbia il nodo pickup e delivery dello stesso cliente, uno di seguito all'altro in soluzione.

- \hat{C} : rappresenta il contributo della distanza marginale di i , essendo $\hat{C} = C_{tot} - C_{dev} + C_{link}$.

Una volta calcolati i contributi della distanza marginale di tutti i clienti, questi devono essere normalizzati e in base a questa normalizzazione, viene assegnato a ciascuno una porzione di costo.

➤ Esempio:

Viene ripreso lo stesso esempio utilizzato per spiegare la regola Isolated Cost Allocation e sono applicate le formule descritte in precedenza per ognuno dei 3 clienti, utilizzando come riferimento la matrice delle distanze riportata nella tabella [1.3].

	Dep (o/d)	P ₁	P ₂	P ₃	D ₄	D ₅	D ₆
Dep (o/d)	/	8	13	31	19	28	5
P ₁	8	/	7	22	14	15	11
P ₂	13	7	/	25	10	27	14
P ₃	31	22	25	/	41	37	27
D ₄	19	14	10	41	/	13	28
D ₅	28	15	27	37	13	/	19
D ₆	5	11	14	27	28	19	/

Tabella 2.3 – Matrice distanze

Cliente 1

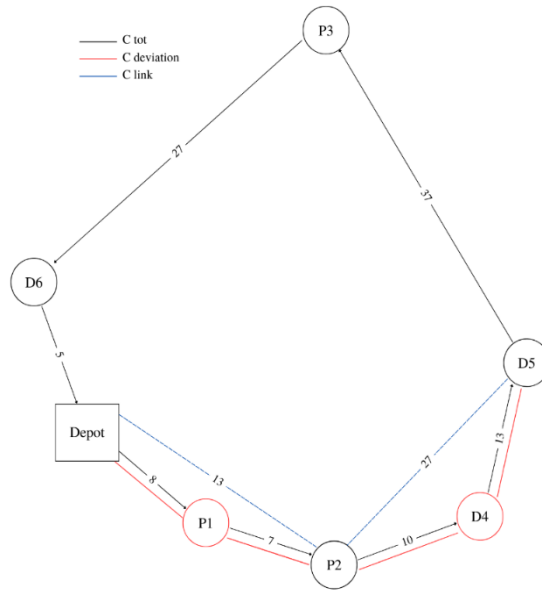


Figura 2.4 – Grafo con focus sui nodi del cliente 1

Prendiamo il cliente $i = 1$ (P_1, D_4) e $X_{P_1, D_4} = 0$

Costo totale (uguale per tutti)

$$C_{tot} = \sum_{i \in Nod} \sum_{j \in Nod} C_{i,j} x_{i,j} = 107€$$

Costo deviation

$$C_{dev} = \sum_{l=0}^{2n+2} C_{l,P_1} X_{l,P_1} + \frac{\sum_{l=0}^{2n+2} C_{P_1,l} X_{P_1,l} + \sum_{l=0}^{2n+2} C_{l,D_4} X_{l,D_4}}{1 + X_{P_1,D_4}} + \sum_{l=0}^{2n+2} C_{D_4,l} X_{D_4,l}$$

$$C_{dev} = C_{0,P_1} + \frac{C_{P_1,P_2} + C_{P_2,D_4}}{1} + C_{D_4,D_5}$$

$$C_{dev} = 8 + 7 + 10 + 13 = 38€$$

Costo link

$$C_{link} = \sum_{l=0}^{2n+2} \sum_{h=0}^{2n+2} C_{l,h} (X_{l,P_1} * X_{P_1,h}) + \sum_{l=0}^{2n+2} \sum_{h=0}^{2n+2} C_{l,h} (X_{l,D_4} * X_{D_4,h})$$

$$C_{link} = C_{0,P_2} + C_{P_2,D_5}$$

$$C_{link} = 13 + 27 = 40€$$

Costo marginale

$$\widehat{C}_1 = C_{tot} - C_{dev} + C_{link} = 107 - 38 + 40 = 109€$$

Cliente 2

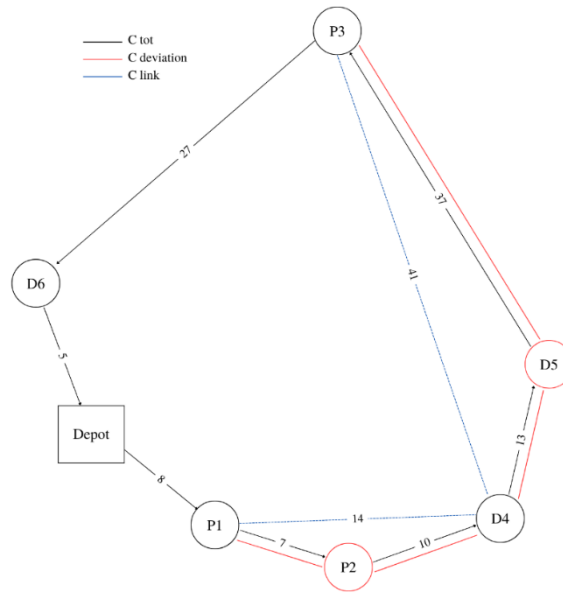


Figura 2.5 – Grafo con focus sui nodi del cliente 2

Prendiamo il cliente $i = 2$ (P_2, D_5) e $X_{P_2, D_5} = 0$

Costo totale (uguale per tutti)

$$C_{tot} = \sum_{i \in Nod} \sum_{j \in Nod} C_{i,j} x_{i,j} = 107€$$

Costo deviation

$$C_{dev} = \sum_{l=0}^{2n+2} C_{l,P_2} X_{l,P_2} + \frac{\sum_{l=0}^{2n+2} C_{P_2,l} X_{P_2,l} + \sum_{l=0}^{2n+2} C_{l,D_5} X_{l,D_5}}{1 + X_{P_2,D_5}} + \sum_{l=0}^{2n+2} C_{D_5,l} X_{D_5,l}$$

$$C_{dev} = C_{P_1,P_2} + \frac{C_{P_2,D_4} + C_{D_4,D_5}}{1} + C_{D_5,P_3}$$

$$C_{dev} = 7 + 10 + 13 + 37 = 67€$$

Costo link

$$C_{link} = \sum_{l=0}^{2n+2} \sum_{h=0}^{2n+2} C_{l,h} (X_{l,P_2} * X_{P_2,h}) + \sum_{l=0}^{2n+2} \sum_{h=0}^{2n+2} C_{l,h} (X_{l,D_5} * X_{D_5,h})$$

$$C_{link} = C_{P_1,D_4} + C_{D_4,P_3}$$

$$C_{link} = 14 + 41 = 55€$$

Costo marginale

$$\widehat{C}_1 = C_{tot} - C_{dev} + C_{link} = 107 - 67 + 55 = 95€$$

Cliente 3

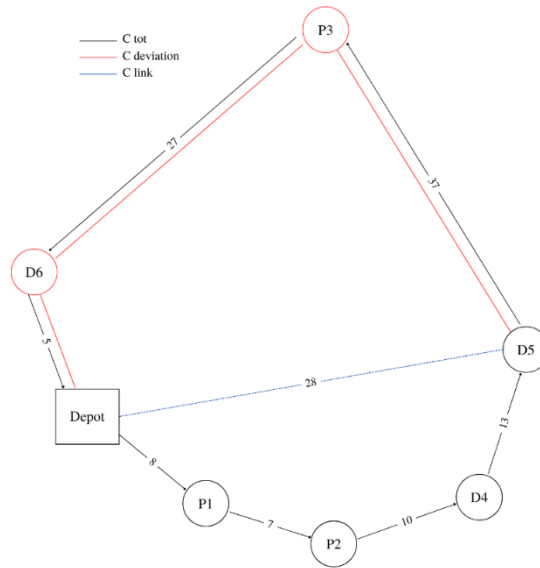


Figura 2.6 – Grafo con focus sui nodi del cliente 3

Prendiamo il cliente $i = 3$ (P_3, D_6) e $X_{P_3, D_6} = 1$

Costo totale (uguale per tutti)

$$C_{tot} = \sum_{i \in Nod} \sum_{j \in Nod} C_{i,j} x_{i,j} = 107€$$

Costo deviation

$$C_{dev} = \sum_{l=0}^{2n+2} C_{l, P_3} X_{l, P_3} + \frac{\sum_{l=0}^{2n+2} C_{P_3, l} X_{P_3, l} + \sum_{l=0}^{2n+2} C_{l, D_6} X_{l, D_6}}{1 + X_{P_3, D_6}} + \sum_{l=0}^{2n+2} C_{D_6, l} X_{D_6, l}$$

$$C_{dev} = C_{D_5, P_3} + \frac{C_{P_3, D_6} + C_{P_3, D_6}}{2} + C_{D_6, 0}$$

$$C_{dev} = 37 + \frac{27 + 27}{2} + 5 = 69€$$

Costo link

$$C_{link} = \sum_{l=0}^{2n+2} \sum_{h=0}^{2n+2} C_{l,h} (X_{l, P_3} * X_{D_6, h})$$

$$C_{link} = C_{0, D_5} = 28$$

$$C_{link} = 28€$$

Costo marginale

$$\widehat{C}_1 = C_{tot} - C_{dev} + C_{link} = 107 - 69 + 28 = 66€$$

Una volta calcolati tutti i contributi della distanza marginale per ogni cliente i , questi vanno normalizzati al fine di individuare i costi da imputare a ciascuno di loro.

$$\%C_{iNS} = \frac{\hat{C}_i}{\sum_{i=1}^n \hat{C}_i} \text{ per poi calcolare il vero e proprio } C_{iNS} = \%C_{iNS} * C_{tot}$$

Cliente 1	Cliente 2	Cliente 3
$\%C_{1NS} = 109/265 = 40\%$	$\%C_{2NS} = 95/265 = 35\%$	$\%C_{3NS} = 61/265 = 24\%$
$C_{1NS} = \%C_{1NS} * C_{tot} = 43 \text{ €}$	$C_{2NS} = \%C_{2NS} * C_{tot} = 38 \text{ €}$	$C_{3NS} = \%C_{3NS} * C_{tot} = 26 \text{ €}$

Tabella 2.4 – Risultati NS

In questo scenario, sono i clienti 1 e 2 quelli che si vedono imputare un costo maggiore, mentre a differenza del caso precedente, il cliente più lontano risente di una diminuzione di costo, in funzione della sua vicinanza relativa agli altri clienti da servire.

La problematica maggiore nella regola *Neighbors savings*, come si evince anche dall'esempio, consiste nel rischio di gravare troppo su quei clienti che si trovano più vicini al deposito. Questi, infatti, più si troveranno ad essere serviti all'interno di un percorso con una durata di percorrenza maggiore, più verranno svantaggiati in termini di allocazione di costi. Per bilanciare questo svantaggio, viene introdotta una terza regola. Il *Normalized Marginal Allocation*.

C) Normalized Marginal Allocation (NMA)

Come visto in precedenza, maggiore sarà il percorso che il veicolo k dovrà affrontare, maggiore sarà la quota di costo imputabile ai nodi più vicini. Per bilanciare l'aspetto negativo in questione, dato che il percorso effettuato da k corrisponde proprio a C_{tot} , questo viene tolto dall'equazione. Potrebbe verificarsi il caso in cui la somma delle distanze degli archi percorsi per visitare sia P_i che D_i come deviazioni dal tour del veicolo k sia inferiore alla somma di tutte le distanze associate ai bordi del grafico. In altre parole, onde evitare che vi sia un costo del *contributo marginale* negativo e quindi impossibile, questo diventa:

$$\hat{C} = \max(0, C_{dev} - C_{link})$$

Esempio

Viene ripreso lo stesso esempio precedente. Viene applicata la formula descritta per tutti e 3 i clienti.

Cliente 1

$$\widehat{C}_1 = \max(0, C_{dev} - C_{link}) = \max(0, 38 - 40) = 0$$

Cliente 2

$$\widehat{C}_2 = \max(0, C_{dev} - C_{link}) = \max(0, 67 - 55) = 12$$

Cliente 3

$$\widehat{C}_3 = \max(0, C_{dev} - C_{link}) = \max(0, 69 - 28) = 41$$

Una volta calcolati tutti i contributi della distanza marginale per ogni cliente i , questi vanno normalizzati al fine di individuare i costi da imputare a ciascuno di loro.

$$\%C_{iENS} = \frac{\widehat{C}_i}{\sum_{i=1}^n \widehat{C}_i} \text{ per poi calcolare il vero e proprio } C_{iENS} = \%C_{iENS} * C_{tot}$$

Cliente 1	Cliente 2	Cliente 3
$\%C_{1ENS} = 0/58 = 0\%$	$\%C_{2ENS} = 12/58 = 23\%$	$\%C_{3ENS} = 46/58 = 77\%$
$C_{1ENS} = \%C_{1NS} * C_{tot} = 0 \text{ €}$	$C_{2ENS} = \%C_{2NS} * C_{tot} = 24 \text{ €}$	$C_{3ENS} = \%C_{3NS} * C_{tot} = 83 \text{ €}$

Tabella 2.5 – Risultati NMA

Come è facile notare, il punto debole di questo metodo consiste nel considerare nulli tutti quei risultati che potrebbero far uscire un valore negativo. Secondo questa logica, infatti, il cliente 1 non dovrebbe pagare nulla per il servizio di prelievo e consegna, ma come già anticipato in precedenza, questa regola ha lo scopo di controbilanciare il difetto della *Neighbors savings* e l'effetto è la compensazione reciproca.

Analisi della soluzione proposta

Ognuna delle soluzioni proposte, come quasi sempre accade, ha sia punti di forza che aspetti critici. L'obiettivo prefissato è stato quello di cercare un set di regole la cui unione potesse generare una soluzione ammissibile, di complessità computazionale non elevata. Infatti, questa soluzione risiede al di fuori del paradigma di Shapley che sarebbe stato uno strumento molto più preciso, ma ad un

costo computazionale maggiore. Per giungere ad una conclusione unica, l'output della strategia proposta consiste in una media ponderata dei risultati delle tre regole.

Riprendendo i tre diversi risultati dell'esempio e riassumendoli in una tabella, è possibile trovare facilmente una media delle alternative proposte:

RULE	Ciente 1	Ciente 2	Ciente 3
ICA	26	42	39
NS	43	38	26
NMA	0	24	83
Media	23	35	49

Tabella 2.6 – Comparazione dei risultati

Per una maggiore comprensione, si riportano di seguito i risultati ottenuti sotto forma di grafico:

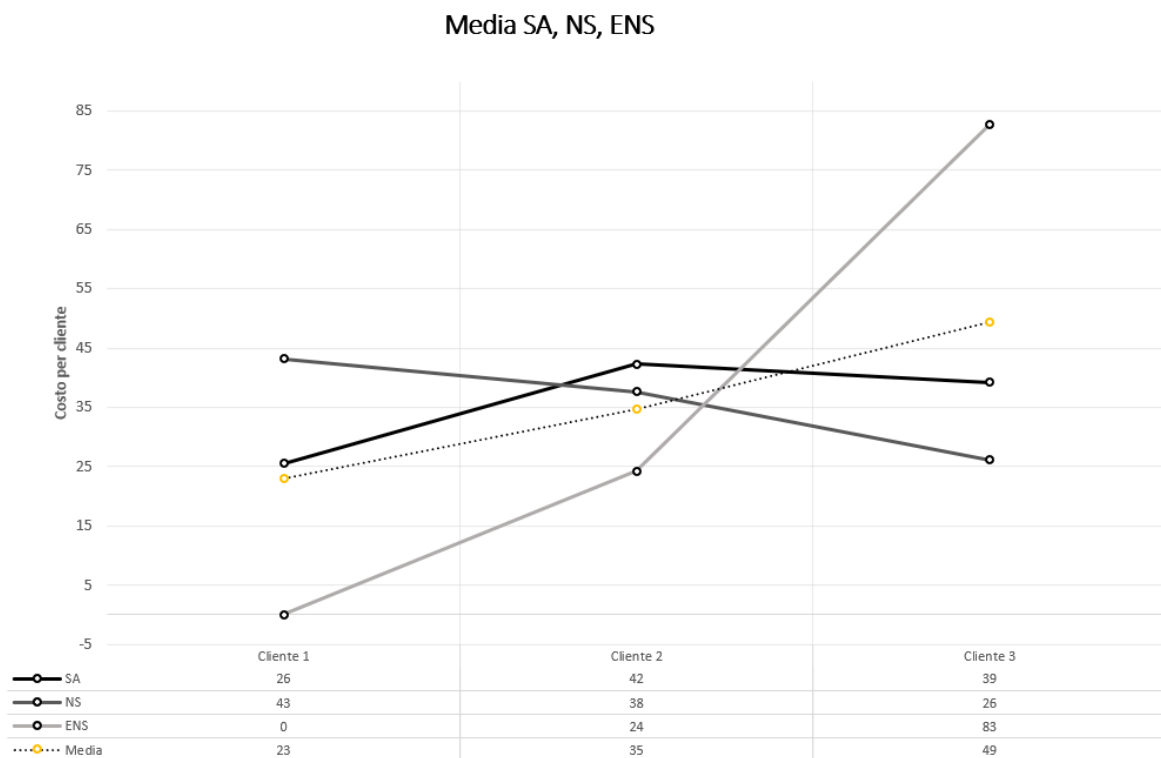


Grafico 2.1 – Andamento Average

Considerazioni su altri esempi

Per valutare la bontà dell'algoritmo, si prenda un caso simile al precedente, ma con il cliente 3 (P_3, D_6) posto ad una distanza di un ordine di grandezza maggiore rispetto alle altre.

	Dep (o/d)	P ₁	P ₂	P ₃	D ₄	D ₅	D ₆
Dep (o/d)	/	8	13	130	19	28	116
P ₁	8	/	7	22	14	15	11
P ₂	13	7	/	25	10	27	14
P ₃	31	22	25	/	79	88	57
D ₄	19	14	10	41	/	13	28
D ₅	28	15	27	37	13	/	65
D ₆	5	11	14	27	28	19	/

Tabella 2.7 – Matrice distanze aggiornata

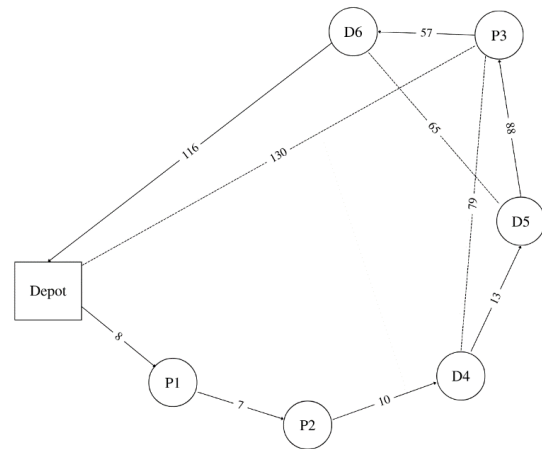


Figura 2.7 – Grafo con distanze aggiornate

$$C_{tot} = 299$$

RULE	Cliente 1	Cliente 2	Cliente 3
% ICA	10%	16%	74%
% NS	47%	43%	10%
% NMA	0%	10%	90%
% Media	19%	23%	58%

Tabella 2.8 – Comparazione nuovi risultati

Media SA, NS, ENS

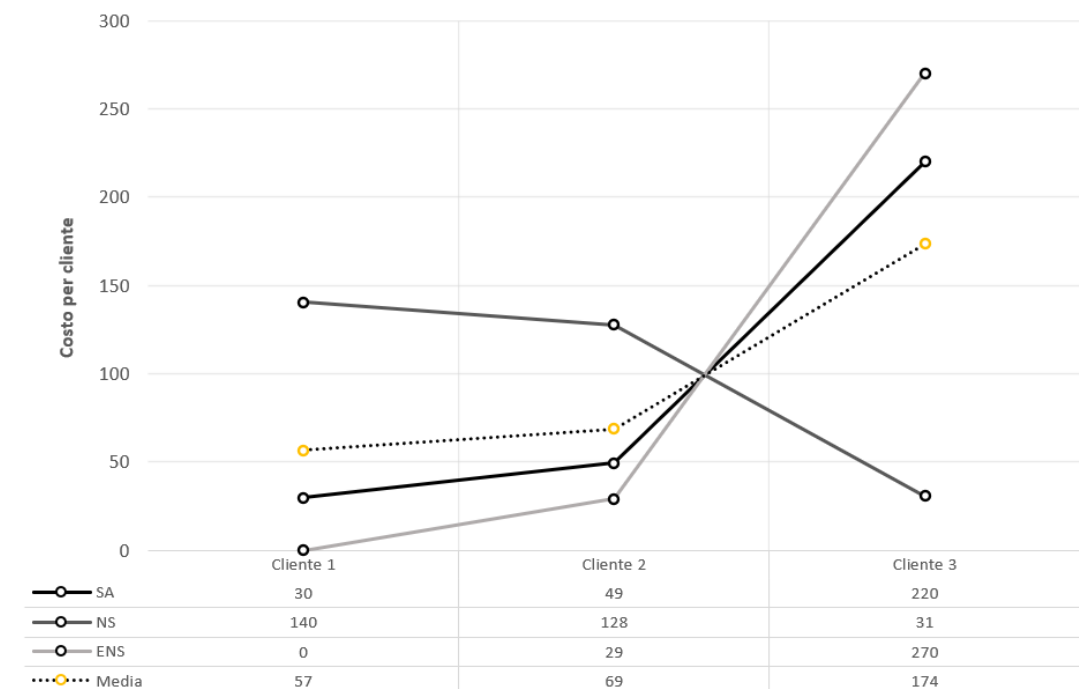


Grafico 2.2 – Andamento Average aggiornato

Partendo dallo stesso esempio iniziale, si consideri il caso in cui il cliente 3 è ancora una volta posto a distanza maggiore dal deposito, ma allo stesso tempo più vicino agli altri clienti.

	Dep (o/d)	P ₁	P ₂	P ₃	D ₄	D ₅	D ₆
Dep (o/d)	/	8	13	130	19	28	116
P ₁	8	/	7	22	14	15	11
P ₂	13	7	/	25	10	27	14
P ₃	31	22	25	/	19	12	8
D ₄	19	14	10	41	/	13	28
D ₅	28	15	27	37	13	/	21
D ₆	5	11	14	27	28	19	/

Tabella 2.9 – Matrice distanze aggiornata

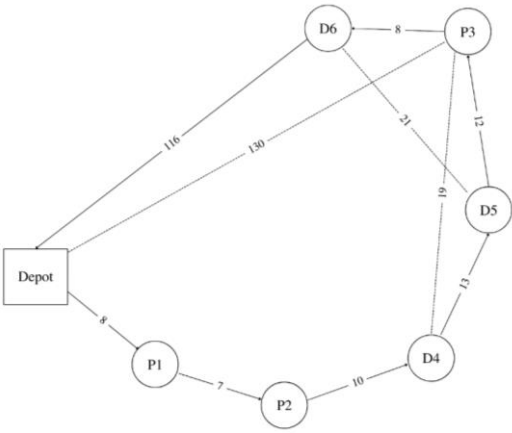


Figura 2.8 – Grafo con distanze aggiornate

$C_{tot} = 174$

RULE	Ciente 1	Ciente 2	Ciente 3
% ICA	11%	19%	70%
% NS	43%	41%	16%
% NMA	0%	8%	92%
% Media	18%	22%	60%

Tabella 2.10 – Comparazione nuovi risultati

Media SA, NS, ENS

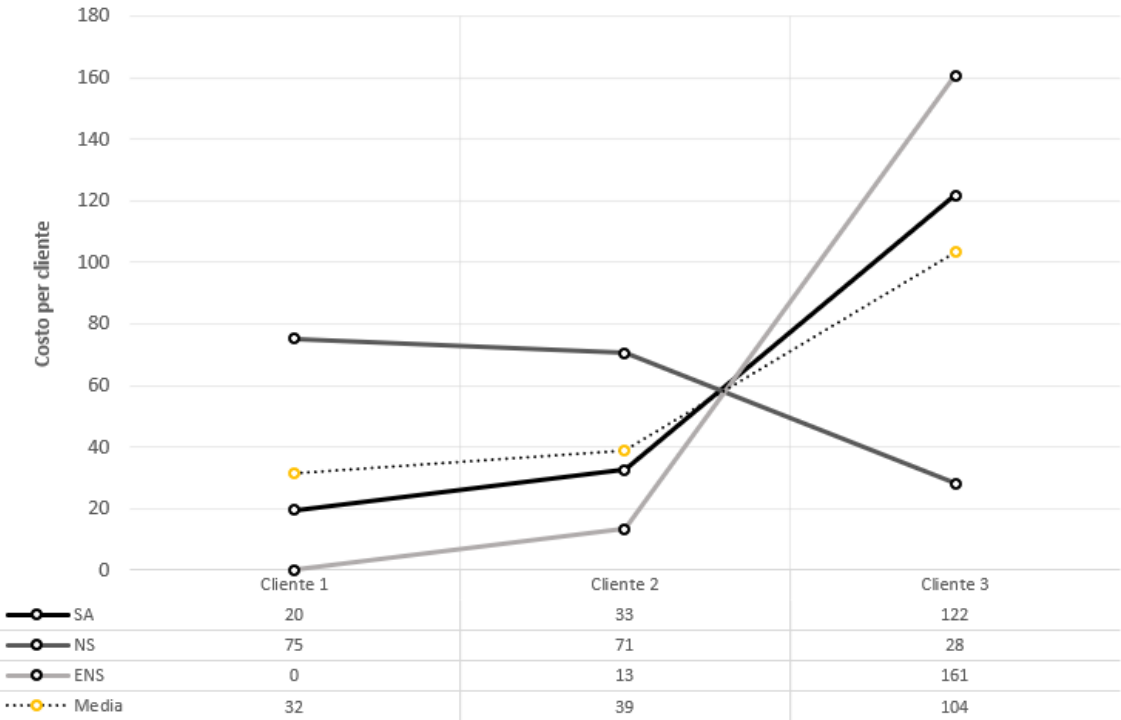
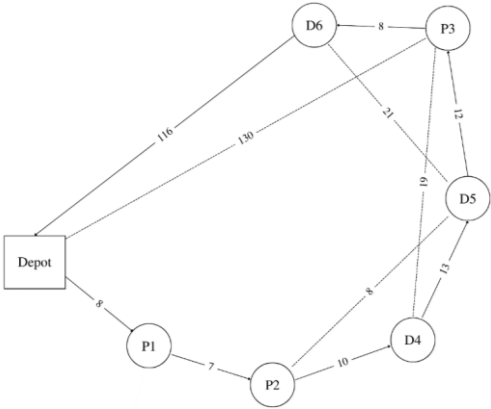


Grafico 2.3 – Andamento Average aggiornato

Da notare come anche nel caso in cui la vicinanza tra il cliente 3 e gli altri sia minore, il risultato rimane quasi costante. La regola *Normalized Marginal Allocation*, “tira” l’aumento dei costi verso il cliente 3, insieme alla regola *Isolated Cost Allocation*. Si provi pertanto a ridistribuire la situazione, facendo in modo che $\widehat{C}_1 = \max(0, C_{dev} - C_{link})$ non vada a 0.

	Dep (o/d)	P ₁	P ₂	P ₃	D ₄	D ₅	D ₆
Dep (o/d)	/	8	13	130	19	28	116
P ₁	8	/	7	22	14	15	11
P ₂	13	7	/	25	10	8	14
P ₃	31	22	25	/	19	12	8
D ₄	19	14	10	41	/	13	28
D ₅	28	15	27	37	13	/	21
D ₆	5	11	14	27	28	19	/

Tabella 2.11 – Matrice distanze aggiornata



Grafo 2.9 – con distanze aggiornate

$$C_{tot} = 174$$

RULE	Cliente 1	Cliente 2	Cliente 3
% ICA	12%	14%	74%
% NS	40%	43%	17%
% NMA	13%	7%	80%
% Media	22%	21%	57%

Tabella 2.12 – Comparazione nuovi risultati

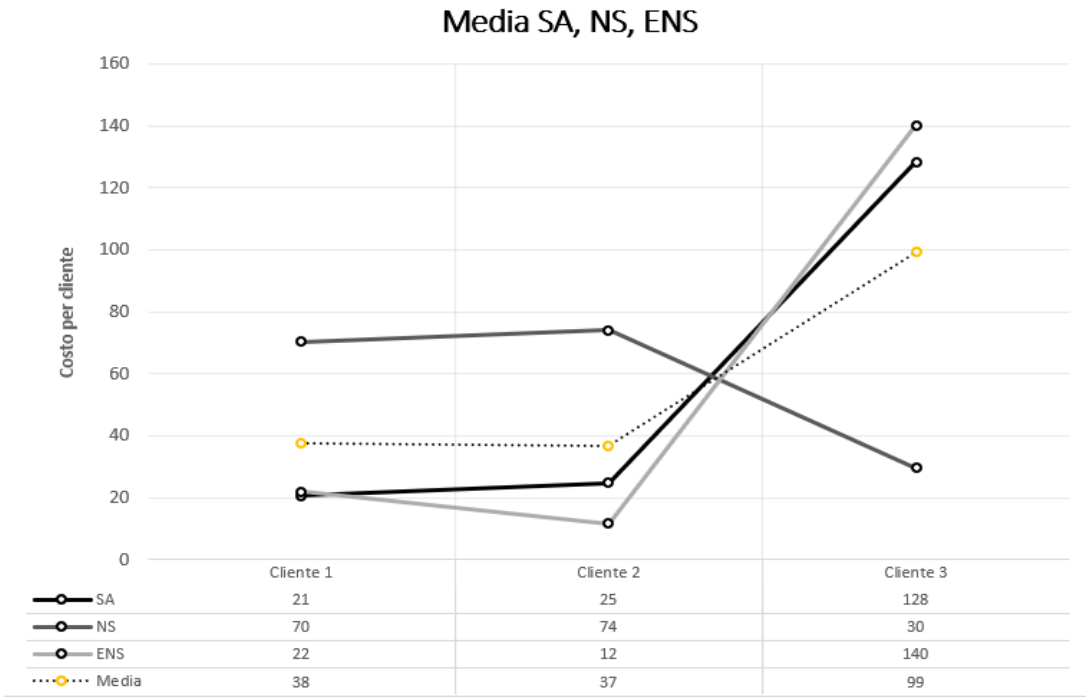


Grafico 2.4 – Andamento Average aggiornato

Ad una prima conclusione si può considerare come anche in quest'ultimo caso non vi sia una variazione troppo significativa della ripartizione del costo. In tutti i casi trattati, la media finale assume un andamento simile alla regola *Isolated Cost Allocation*, indice del fatto la distanza tra il deposito e i clienti da servire è il parametro con peso maggiore. Questo dato viene aggiustato dalle altre due regole, mantenendo il risultato finale sullo stesso ordine di grandezza. Si raggiunge così l'obiettivo di trovare una strategia di allocazione dei costi al di fuori del paradigma dello Shapley value mantenendo allo stesso tempo una consistenza decisamente accettabile.

Capitolo 3 – Applicazione dello Shapley value all'allocazione fair dei costi

Storia dello Shapley value

Lloyd Stowell Shapley è stato un matematico ed economista statunitense. Con l'uscita nel 1953 del suo paper "A value for n-person games" Shapley formula un approccio atto ad aggirare tutta quella gamma di problemi della Teoria dei giochi, che si riassume nell'integrazione strategica complessa. Infatti, egli sosteneva come fosse possibile valutare, in modo numerico il "value", il valore nel partecipare ad un determinato gioco. Da questo studio ha avuto origine lo Shapley value, che fin dalla data della sua prima pubblicazione, continua a suscitare interesse in ricercatori e studiosi ed è stato il principale motivo per cui nel 2012, al suo autore è stato assegnato il premio Nobel per l'economia. Per comprendere a pieno i motivi per cui questo valore sia così importante in ambito accademico, è necessario fare un passo indietro lungo quasi dieci anni ovvero al 1944, anno della pubblicazione da parte dei ricercatori John von Neumann e Oskar Morgenstern del "*Theory of games and economic behavior*".

L'approccio descritto è quello secondo cui bisogna "dividere le difficoltà", al fine di trovare dei modelli più semplici, che si adattino all'ambiente strategico di riferimento. Per farlo è stato ipotizzato di provare a elencare tutte le condizioni che portano a determinate preferenze di un giocatore. Questo, messo davanti ad una serie di opzioni è portato a scegliere quella che massimizza la propria funzione di utilità, soddisfacendo pertanto il valore atteso della sua funzione di valore. L'obiettivo è arrivare a riassumere la distribuzione delle probabilità di scelta tra una serie di alternative in un unico numero definito come l'utilità attesa. La classe di giochi presa in esame è la cosiddetta "transferable utility games" (TU games), che rispetta le seguenti assunzioni:

1. L'utilità deve essere misurabile e quindi viene incorporata nel mezzo "scambio di denaro". Che deve essere completamente trasferibile tra i giocatori.
2. Per mantenere l'obiettività, ogni giocatore all'interno di una coalizione deve poter valutare le proprie opportunità senza fare riferimento a giocatori non inclusi nella coalizione.
3. Per la distribuzione del valore, ogni coalizione è libera di stipulare accordi tramite un procedimento concordato tra tutti i membri, ma senza alcun costo aggiuntivo.

Lo scopo principale dei TU games è individuare una serie nella distribuzione dei payoff di ciascun giocatore con il vincolo che la somma dei payoff di ciascuna coalizione S sia almeno pari a $v(S)$.

Neumann e Morgenstern nel loro studio evidenziano una problematica fondamentale che può essere chiarita meglio attraverso un esempio. Ci troviamo nella situazione in cui vi è un venditore (1) e due compratori (2, 3) entrambi interessati ad un potenziale oggetto in possesso di 1.

Funzione di costo	Valore	
$v(0)$	0	
$v(1)$	10	1 valuta l'oggetto €10
$v(2)$	0	
$v(3)$	0	2 valuta l'oggetto €20
$v(1, 2)$	20	3 valuta l'oggetto €30
$v(1, 3)$	30	
$v(2, 3)$	10	
$v(1, 2, 3)$	30	

Tabella 3.1 – Funzione costo dei giocatori

Se ci troviamo in un “*transferable utility game*” e quindi valgono le assunzioni precedentemente citate, è possibile modellizzare il gioco come segue: $N = (1, 2, 3)$ e funzioni di costo come riportato in tabella. Per il raggiungimento dello scopo di ottimizzare della ricchezza collettiva, solo le coalizioni che contengono il venditore 1 possono essere prese in esame. Il “core” corrisponde a quei payoff in cui il venditore vende al prezzo di riserva più alto. Von Neumann e Morgenstern nel loro paper propongono un tipo di approccio chiamata “*stable set*”, nella quale vi sono infinite soluzioni, ognuna delle quali composta dall’obiettivo (core) più una serie infinita di possibilità, distribuite su una curva continua, che permettono di dividere la ricchezza tra i due acquirenti per ogni prezzo possibile che sia inferiore ai €20. Tali comportamenti sono chiamati “standard of behavior”.

Il problema risiede proprio nella molteplicità delle soluzioni che purtroppo non derivano dalla funzione caratteristica, bensì da tutta una serie di comportamenti che ogni giocatore può esibire, dovuti principalmente alle caratteristiche dell’ambiente. Il loro scopo era quello di portare alla luce la mancanza nel modello di tutta quella serie di interazioni strategiche tra giocatori, che però sono fondamentali in un qualunque modello di teoria dei giochi. Per questo motivo studi successivi si sono focalizzati solo sul “core” del gioco, tralasciando il resto. Anche in questo caso, tuttavia, vi possono essere casi in cui il “core” sia vuoto oppure vi siano presenti un numero infinito di soluzioni.

Da questo punto di partenza, la teoria dei giochi ha cercato di lavorare verso l’inserimento di più dettagli istituzionali possibili, allo scopo di descrivere al meglio queste complessità, che sono di ampio spettro, un riflesso della complessità dell’interazione strategica tra giocatori. In questo modo aumentano le necessità di trovare un metodo più semplice per effettuare una valutazione preliminare dei giochi. È esattamente in questo contesto che si inserisce la Shapley value.

La forma della Shapley value

Shapley riprende i concetti base analizzati da Neumann e Morgenstern e li porta ad un livello superiore. Nel paper del 1944 era nata l'idea di ridurre ogni singola alternativa ad un numero che esprimesse l'utilità attesa del giocatore e la convenienza delle varie coalizioni all'interno di un gioco. Shapley fa un passo avanti analizzando il gioco da un gradino più alto, con l'obiettivo di trovare una singola funzione caratteristica, composta da un unico numero, rappresentante il valore ("value") nel partecipare ad un dato gioco.

Indichiamo con U l'universo di tutti i possibili giocatori. Da questo punto di partenza, Shapley aveva ipotizzato di considerare tutti i giochi v ai quali avrebbero potuto partecipare tutti i giocatori rientranti in U . Viene così definita una funzione che assegna ad ogni gioco v , un numero $\phi_i(v)$ per ogni giocatore $i \in U$. $\phi_i(v)$. Questa viene chiamata "Shapley value" ed è il valore, espresso sotto forma di numero, associato al gioco v . per ogni giocatore i .

Prima di andare a definire la funzione caratteristica, è necessario introdurre i tre assiomi ai quali la funzione fa riferimento. Per farlo, si prendano due sottoinsiemi di U : S e N , tali che $v(S) = v(S \cap N)$:

1. **Symmetry axiom**: tradotto in italiano come assioma dell'anonimità, stabilisce che il contributo dato da un giocatore non può essere condizionato da "chi" è il giocatore, ma solo e soltanto da quanto quest'ultimo sia in grado di ottenere.

Gioco v		Gioco w	
$v(1) = 0$	$v(1,2) = 3$	$w(1) = 0$	$w(1,2) = 7$
$v(2) = 0$	$v(1,3) = 3$	$w(2) = 0$	$w(1,3) = 3$
$v(3) = 0$	$v(2,3) = 7$	$w(3) = 0$	$w(2,3) = 3$
$v(1,2,3) = 15$		$w(1,2,3) = 15$	

Tabelle 3.2 – Esempio Symmetry axiom

Nell'esempio il giocatore 3 nel gioco w , indipendentemente da chi sia, si trova nella stessa identica situazione del giocatore 1 nel gioco v , per cui il contributo dato ad entrambi deve essere identico. Giocatori trattati in modo identico dalla funzione caratteristica sono trattati in modo identico dallo Shapley value.

Lemma: questo discorso può essere espanso prendendo un gioco v e una sua permutazione $\sigma : N \rightarrow N$, allora $\phi_i(v) = \phi_{\sigma(i)}(\sigma v)$

2. **Carrier axiom:** questo assioma viene spesso suddiviso in due assiomi distinti: efficiency axiom & dummy player property axiom:

- a. **Efficiency axiom:** preso ogni gioco v e $\phi(v)$ il valore dato dal gioco, questo deve essere una ripartizione di quello che il “tutto” riesce ad ottenere. In altre parole, la somma dei $\phi_i(v)$ su tutti i giocatori i di un qualsiasi sottoinsieme $N \subset U$ deve essere pari a $v(N)$.

$$\sum_{i \in N} \phi_i = v(N)$$

- b. **Dummy player property axiom:** assegniamo il valore di $\phi_i(v) = 0$ a quel giocatore i che è considerato un “dummy player” o “null” nel gioco v , ovvero un giocatore i che non fa parte della coalizione di riferimento. Definiamo (ed analizzeremo meglio in seguito) come *contributo marginale* di un giocatore i alla coalizione S ($i \notin S$), il numero reale: $v(S \cup i) - v(i)$. Se prendiamo $i \in Z \ \forall \ S \subseteq Z$ tale che $v(S \cup i) = v(i)$, con $i \notin S$, allora i è un dummy player e quindi $\phi_i(v) = 0$.

3. **Additivity axiom:** concetto secondo il quale, presi due giochi v e w abbiamo che

$$\phi_i(v + w) = \phi_i(v) + \phi_i(w) \text{ per ogni } i \in N$$

Shapley dimostra che esiste una e una sola funzione che soddisfa contemporaneamente tutti e tre gli assiomi, ed è $\phi_i(v)$:

$$\phi_i(v) = \sum_{S \subset N} \frac{(s-1)!(n-s)!}{n!} [v(S) - v(S-i)]$$

Formula 3.1

Lo Shapley value $\phi_i(v)$ si presenta come il *contributo marginale* medio di un giocatore i in una specifica coalizione, rispetto le possibili permutazioni dei giocatori. In un gioco cooperativo, un giocatore può avere un peso molto piccolo, ma a determinate condizioni, il suo effetto può essere determinante per l’esito di un risultato. Questo effetto è determinato da uno shapley value elevato.

Quello che bisogna capire è il funzionamento del così detto “*contributo marginale*”. Per meglio definire il problema, immaginiamo di porci in una stanza vuota con tre giocatori in attesa. Fissiamo la permutazione 1, 2, 3 e vediamo quello che succede spostandoci in avanti con l’indice i .

- Entra il giocatore 1 con una sua *funzione di costo* $v(1)$. Gli viene assegnato come valore marginale esattamente la sua *funzione di costo*, dato che non vi era nessuno prima nella stanza.
- Entra il giocatore 2, con una sua *funzione di costo* $v(2)$, a questo viene assegnato un valore marginale pari a $v(1, 2) - v(1)$, che corrisponde all’aumento di costo imputabile a 2.
- Entra il giocatore 3 con *funzione di costo* $v(3)$ e gli viene assegnato un valore marginale pari a $v(1, 2, 3) - v(1, 2)$.

$\phi_i(v)$ è il *contributo marginale* atteso dato dal giocatore i quando entra nella stanza. Non bisogna fare distinzioni secondo l'ordine di arrivo, quello che bisogna considerare è una media dei contributi marginali che ogni giocatore, entrando nella stanza, porta con sé, in ogni ordine possibile. Questo porta ad un numero di operazioni dipendente dal numero di permutazioni, ovvero $n!$ [Appendice sulla complessità 1]. Secondo Shapley quindi, questa operazione va eseguita tenendo fissato i e scorrendo tutte le permutazioni degli n elementi dei nostri giocatori, sommando tutti i valori marginali del giocatore i per poter trovare il suo *contributo marginale*.

Nota sulla complessità 1

Dalla formula è possibile quindi evincere che il numero di operazioni necessarie per calcolare esattamente lo Shapley value segue il numero di permutazioni su un insieme di n elementi ($n!$). La complessità dell'algoritmo si stanza come $O(n!)$, che a differenza degli altri modelli come si può evincere dalla figura x, a parità di elementi, è quella che necessita un maggior numero di operazioni per essere portata a termine. Cresce molto più velocemente rispetto alle altre.

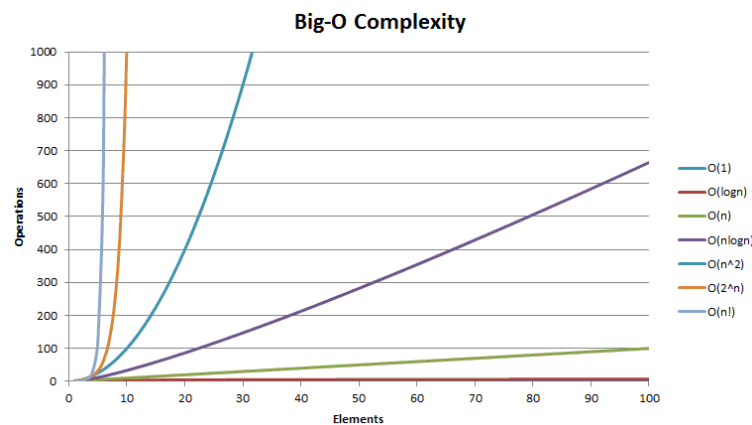


Grafico 3.1 – Andamento $O()$

È possibile, pertanto, calcolare lo Shapley value in tempo polinomiale, ma solo nei casi in cui il numero di giocatori non sia elevato. Se infatti il numero di giocatori fosse per esempio 10, il numero di operazioni richieste ad un calcolatore sarebbe di $10!$, ovvero 3.628.800 operazioni. In questa sezione ci siamo limitati a descrivere ed analizzare l'algoritmo, ma questo problema sarà ripreso più avanti, introducendo delle possibili soluzioni euristiche e non, per permettere ad un calcolatore di poter estrapolare una soluzione accettabile in un tempo pseudo polinomiale.

A) ApproShapley

Una semplificazione originalmente suggerita nel Mann and Shapley (1960) e chiamata ApproShapley si pone lo scopo di riformulare lo Shapley value nel seguente modo:

$$\phi_i(v) = \frac{1}{n!} \sum_{\pi \in S_n} (v(\pi_i \cup \{i\}) - v(\pi_i))$$

Formula 3.2

π_i = Insieme di tutti i giocatori che precedono i nella permutazione S_n

S_n = Insieme di tutti gli ordinamenti possibili dati dalle n permutazioni

$v(\pi_i \cup \{i\}) - v(\pi_i)$ = Contributo marginale del giocatore i secondo funzione di costo v

Si definisce π_i come l'insieme di tutti i giocatori che precedono il giocatore i nella permutazione S_n (insieme di tutti gli ordinamenti possibili dati dalle n permutazioni), pertanto il *contributo marginale* $v(\pi_i \cup \{i\}) - v(\pi_i)$ è il costo di tutti i predecessori di i, con i compreso al netto del costo dei predecessori di i stesso.

Per una maggiore comprensione dell'algoritmo vengono presentati di seguito 2 esempi:

Esempio 1:

Nel primo esempio riprendiamo il caso precedente, ovvero presi 3 giocatori, $N = (1, 2, 3)$, nella situazione in cui vi è un venditore (1) e due compratori (2, 3) interessati ad un potenziale oggetto in possesso di 1.

- 1 valuta l'oggetto €10
- 2 valuta l'oggetto €20
- 3 valuta l'oggetto €30

Con delle funzioni di costo come segue (ricordando che $v(0)$ sia l'insieme vuoto): Si applica l'algoritmo di Shapley

$$\phi_i(v) = \frac{1}{n!} \sum_{\pi \in S_n} (v(\pi_i \cup \{i\}) - v(\pi_i))$$

Funzione di costo	Valore
$v(0)$	0
$v(1)$	10
$v(2)$	0
$v(3)$	0
$v(1, 2)$	20
$v(1, 3)$	30
$v(2, 3)$	10
$v(1, 2, 3)$	30

Tabella 3.3 – Funzione di costo

E si estrapolano i seguenti calcoli:

	i = 1		i = 2		i = 3	
Permutazioni	Contributo marginale	Calcolo	Contributo marginale	Calcolo	Contributo marginale	Calcolo
1, 2, 3	$v(1) - v(0)$	10	$v(1, 2) - v(1)$	10	$v(1, 2, 3) - v(1, 2)$	10
1, 3, 2	$v(1) - v(0)$	10	$v(1, 2, 3) - v(1, 3)$	0	$v(1, 3) - v(1)$	20
2, 1, 3	$v(1, 2) - v(2)$	20	$v(2) - v(0)$	0	$v(1, 2, 3) - v(1, 2)$	10
2, 3, 1	$v(1, 2, 3) - v(2, 3)$	20	$v(2) - v(0)$	0	$v(2, 3) - v(2)$	10
3, 1, 2	$v(1, 3) - v(3)$	30	$v(1, 2, 3) - v(1, 3)$	0	$v(3) - v(0)$	0
3, 2, 1	$v(1, 2, 3) - v(2, 3)$	20	$v(2, 3) - v(3)$	10	$v(3) - v(0)$	0
	Somma:	110	Somma:	20	Somma:	50

Tabella 3.4 – Risultati

Per ogni giocatore i devono essere eseguiti $n!$ operazioni (in questo caso 6). Ne segue che per i e giocatori coinvolti lo shapley value è il seguente:

- $\phi_1(v) = \frac{110}{6} = 18.33$
- $\phi_2(v) = \frac{20}{6} = 3.33$
- $\phi_3(v) = \frac{50}{6} = 8.33$

$$\sum_{i \in N} \phi_i = v(N) = 18.33 + 3.33 + 8.33 = 30$$

Si nota che l'assioma 2.a (efficiency axiom) è rispettato e si deduce quindi che indipendentemente da chi sia il venditore 1, questo, avendo uno Shapley value di gran lunga superiore ai due clienti, valuta la possibilità di inserirsi nel gioco v , estremamente più vantaggiosa rispetto agli altri.

Esempio 2:

Qualora invece i giocatori non fossero 3 ma 4, come nell'appendice alla complessità 1, si andrebbe ad aumentare in modo esponenziale la complessità computazionale.

Se prendiamo il caso riportato dalla tabella sulla destra, che riporta i valori della funzione di costo $v()$, i calcoli per lo Shapley value diverranno:

Funzione di costo	Valore
$v(0)$	0
$v(1)$	22
$v(2)$	24
$v(3)$	20
$v(4)$	10
$v(1, 2)$	26
$v(1, 3)$	36
$v(1, 4)$	33
$v(2, 3)$	28
$v(2, 4)$	36
$v(3, 4)$	19
$v(1, 2, 3)$	30
$v(1, 2, 4)$	37
$v(1, 3, 4)$	35
$v(2, 3, 4)$	27
$v(1, 2, 3, 4)$	29

Tabella 3.5 – Nuova funzione di costo

	i = 1		i = 2		i = 3		i = 4	
P₄	Contributo marginale	Calc.	Contributo marginale	Calc.	Contributo marginale	Calc.	Contributo marginale	Calc.
1, 2, 3, 4	v(1) - v(0)	22	v(1, 2) - v(1)	4	v(1, 2, 3) - v(1, 2)	4	v(1, 2, 3, 4) - v(1, 2, 3)	-1
1, 2, 4, 3	v(1) - v(0)	22	v(1, 2) - v(1)	4	v(1, 2, 3, 4) - v(1, 2, 4)	-8	v(1, 2, 4) - v(1, 2)	11
1, 3, 2, 4	v(1) - v(0)	22	v(1, 2, 3) - v(1, 3)	-6	v(1, 3) - v(1)	14	v(1, 2, 3, 4) - v(1, 2, 3)	-1
1, 3, 4, 2	v(1) - v(0)	22	v(1, 2, 3, 4) - v(1, 3, 4)	-6	v(1, 3) - v(1)	14	v(1, 3, 4) - v(1, 3)	-1
1, 4, 2, 3	v(1) - v(0)	22	v(1, 2, 4) - v(1, 4)	4	v(1, 2, 3, 4) - v(1, 2, 4)	-8	v(1, 4) - v(1)	11
1, 4, 3, 2	v(1) - v(0)	22	v(1, 2, 3, 4) - v(1, 3, 4)	-6	v(1, 3, 4) - v(1, 4)	2	v(1, 4) - v(1)	11
2, 1, 3, 4	v(1, 2) - v(2)	2	v(2) - v(0)	24	v(1, 2, 3) - v(1, 2)	4	v(1, 2, 3, 4) - v(1, 2, 3)	-1
2, 1, 4, 3	v(1, 2) - v(2)	2	v(2) - v(0)	24	v(1, 2, 3, 4) - v(1, 2, 4)	-8	v(1, 2, 4) - v(1, 2)	11
2, 3, 1, 4	v(1, 2, 3) - v(2, 3)	2	v(2) - v(0)	24	v(2, 3) - v(2)	4	v(1, 2, 3, 4) - v(1, 2, 3)	-1
2, 3, 4, 1	v(1, 2, 3, 4) - v(2, 3, 4)	2	v(2) - v(0)	24	v(2, 3) - v(2)	4	v(2, 3, 4) - v(2, 3)	-1
2, 4, 1, 3	v(1, 2, 4) - v(2, 4)	1	v(2) - v(0)	24	v(1, 2, 3, 4) - v(1, 2, 4)	-8	v(2, 4) - v(2)	12
2, 4, 3, 1	v(1, 2, 3, 4) - v(2, 3, 4)	2	v(2) - v(0)	24	v(2, 3, 4) - v(2, 4)	-9	v(2, 4) - v(2)	12
3, 1, 2, 4	v(1, 3) - v(3)	16	v(1, 2, 3) - v(1, 3)	-6	v(3) - v(0)	20	v(1, 2, 3, 4) - v(1, 2, 3)	-1
3, 1, 4, 2	v(1, 3) - v(3)	16	v(1, 2, 3, 4) - v(1, 3, 4)	-6	v(3) - v(0)	20	v(1, 3, 4) - v(1, 3)	-1
3, 2, 1, 4	v(1, 2, 3) - v(2, 3)	2	v(2, 3) - v(3)	8	v(3) - v(0)	20	v(1, 2, 3, 4) - v(1, 2, 3)	-1
3, 2, 4, 1	v(1, 2, 3, 4) - v(2, 3, 4)	2	v(2, 3) - v(3)	8	v(3) - v(0)	20	v(2, 3, 4) - v(2, 3)	-1
3, 4, 1, 2	v(1, 3, 4) - v(3, 4)	16	v(1, 2, 3, 4) - v(1, 3, 4)	-6	v(3) - v(0)	20	v(3, 4) - v(3)	-1
3, 4, 2, 1	v(1, 2, 3, 4) - v(2, 3, 4)	2	v(2, 3, 4) - v(3, 4)	8	v(3) - v(0)	20	v(3, 4) - v(3)	-1
4, 1, 2, 3	v(1, 4) - v(4)	23	v(1, 2, 4) - v(1, 4)	4	v(1, 2, 3, 4) - v(1, 2, 4)	-8	v(4) - v(0)	10
4, 1, 3, 2	v(1, 4) - v(4)	23	v(1, 2, 3, 4) - v(1, 3, 4)	-6	v(1, 3, 4) - v(1, 4)	2	v(4) - v(0)	10
4, 2, 1, 3	v(1, 2, 4) - v(2, 4)	1	v(2, 4) - v(4)	26	v(1, 2, 3, 4) - v(1, 2, 4)	-8	v(4) - v(0)	10
4, 2, 3, 1	v(1, 2, 3, 4) - v(2, 3, 4)	2	v(2, 4) - v(4)	26	v(2, 3, 4) - v(2, 4)	-9	v(4) - v(0)	10
4, 3, 1, 2	v(1, 3, 4) - v(3, 4)	16	v(1, 2, 3, 4) - v(1, 3, 4)	-6	v(3, 4) - v(4)	9	v(4) - v(0)	10
4, 3, 2, 1	v(1, 2, 3, 4) - v(2, 3, 4)	2	v(2, 3, 4) - v(3, 4)	8	v(3, 4) - v(4)	9	v(4) - v(0)	10
	Somma:	264	Somma:	196	Somma:	120	Somma:	116

Tabella 3.6 – Nuovi risultati

Per ogni giocatore i devono essere eseguiti $n!$ operazioni (in questo caso 24). Ne segue che per 4 e giocatori coinvolti lo shapley value è il seguente:

- $\phi_1(v) = \frac{264}{24} = 11$
- $\phi_2(v) = \frac{196}{24} = 8.17$
- $\phi_3(v) = \frac{120}{24} = 5$
- $\phi_4(v) = \frac{116}{24} = 4.83$

$$\sum_{i \in N} \phi_i = v(N) = 11 + 8.17 + 5 + 4.83 = 29$$

Applicare ApproShapley ad un problema di TSP

Una volta analizzato il funzionamento dello Shapley value, è necessario descrivere come questo possa essere applicato ad un generico problema di TSP. Il fattore cruciale consiste nel convertire la matrice delle distanze nella *funzione di costo* $V()$. Per comprendere meglio il procedimento, viene proposto un esempio di seguito che ricalca quello precedentemente analizzato con $n = 4$ nodi (più il deposito).

Venga presa la rete in figura con la matrice delle distanze:

Distanze	Valori
d_01	11
d_02	12
d_03	10
d_04	5
d_12	3
d_13	15
d_14	17
d_23	6
d_24	19
d_34	4

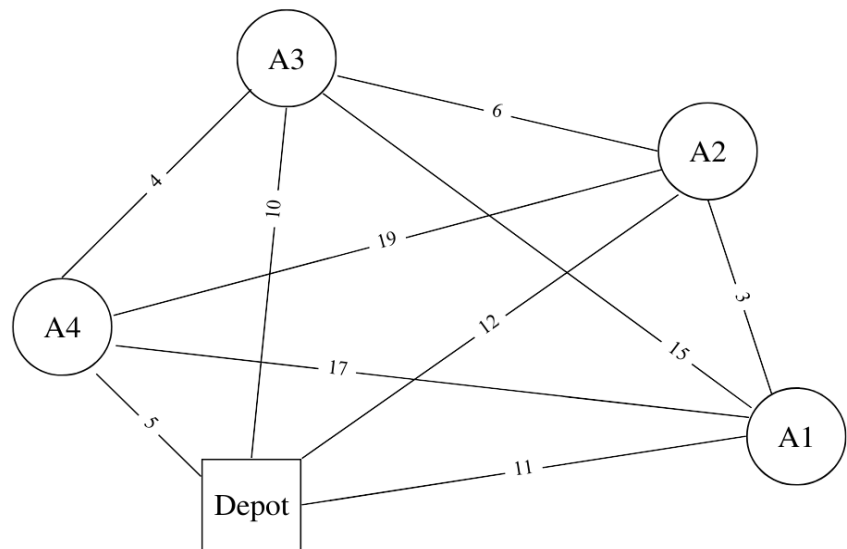


Tabella 3.6 – Nuovi risultati

Figura 3.1 – Grafo completo 6 nodi + deposito

Per creare la matrice *funzione di costo*, bisogna partire dalla definizione: questa rappresenta il costo $v(i, j, k)$ che il costo, quando in soluzione si trovano i nodi i, j, k . Va da se quindi, che il costo in soluzione di $v(1, 2, 3)$ consiste nel costo minore per raggiungere questi 3 nodi. Il tutto quindi si traduce in un problema di TSP per ogni singola $v()$. Come è facile intuire questo aggiunge al raggiungimento della soluzione finale, un calcolo computazionale non indifferente. Analizziamolo punto per punto:

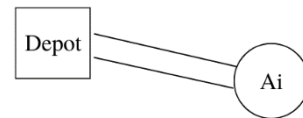
- $v(0)$

La *funzione costo* $v(0)$ è esattamente pari a 0 dato che è l'insieme vuoto

Funzione di costo	Calcoli	Valore
$v(0)$	Insieme vuoto	0

- **v(i)**

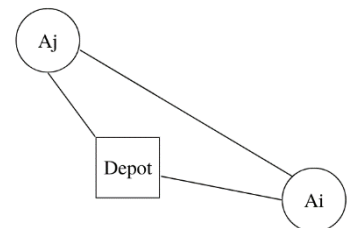
Avendo messo in soluzione un unico nodo, il costo ottimale per partire dal nodo 0 (deposito) e ritornarvi, è esattamente $2 * d_{0i}$



Funzione costo	di	Calcoli	Valore
v(1)		$2 * d_{01}$	22
v(2)		$2 * d_{02}$	24
v(3)		$2 * d_{03}$	20
v(4)		$2 * d_{04}$	10

- **v(i, j)**

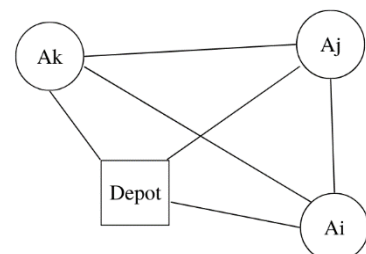
In soluzione adesso si trovano due nodi, ma anche qui il problema di TSP è di facile risoluzione, dato che il risultato non è altro che la somma delle distanze $d_{01} + d_{12} + d_{02}$



Funzione costo	di	Calcoli	Valore
v(1, 2)		$d_{01} + d_{12} + d_{02}$	26
v(1, 3)		$d_{01} + d_{13} + d_{03}$	36
v(1, 4)		$d_{01} + d_{14} + d_{04}$	33
v(2, 3)		$d_{02} + d_{23} + d_{03}$	28
v(2, 4)		$d_{02} + d_{24} + d_{04}$	36
v(3, 4)		$d_{03} + d_{34} + d_{04}$	19

- **v(i, j, k)**

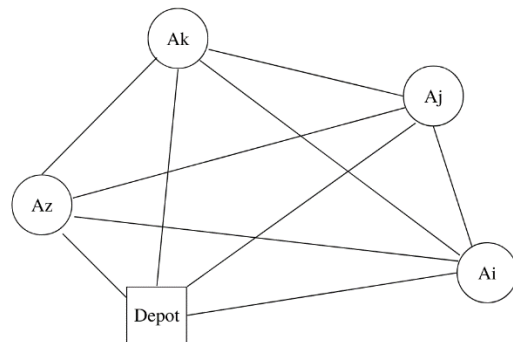
Il problema nasce da $n = 3$ ed andrà a complicarsi in maniera esponenziale. Infatti in questo caso il problema del TSP deve fornire una soluzione andando a paragonare 3 possibili percorsi (p1, p2, p3), trovando il risultato minore.



Funzione costo	di	Calcoli	Valore
$v(1, 2, 3)$		$\min(p1, p2, p3) = d_{01} + d_{12} + d_{13} + d_{03}$	30
$v(1, 2, 4)$		$\min(p1, p2, p3) = d_{02} + d_{12} + d_{14} + d_{04}$	37
$v(1, 3, 4)$		$\min(p1, p2, p3) = d_{01} + d_{13} + d_{34} + d_{04}$	35
$v(2, 3, 4)$		$\min(p1, p2, p3) = d_{02} + d_{23} + d_{34} + d_{04}$	27

- **$v(i, j, k, z)$**

Il meccanismo è lo stesso, per cui per $n = 4$ nodi, sarà necessario confrontare 12 percorsi ($p1, \dots, p12$) per trovare il percorso minimo e risolvere il problema del TSP. Si noti come quello rappresentato sia l'intero grafo.



Funzione costo	di	Calcoli	Valore
$v(1, 2, 3, 4)$		$\min(p1, \dots, p12) = d_{01} + d_{12} + d_{23} + d_{34} + d_{04}$	29

In questo caso, dalla matrice delle distanze si passa alla *funzione di costo*, indispensabile per la risoluzione dello Shapley value. È molto importante notare come, indipendentemente dalla *funzione di costo* che comprende tutti i nodi, la somma dei valori in gioco equivalga esattamente alla soluzione del TSP dell'intero grafo. Riprendendo le soluzioni di questo esempio, il percorso ottimo è:

$[0 - 1 - 2 - 3 - 4 - 0]$ con costo totale $C_{tot} = 29$

Il risultato dello Shapley value per i 4 nodi è: (11, 8.17, 5, 4.83) la cui somma, per l'assioma 2.a (efficiency axiom) è $v(N)$ che corrisponde esattamente alla soluzione del problema di TSP del grafo.

Per cui Shapley non solo fornisce un metodo per allocare in maniera equa il costo tra i nodi, ma riesce anche a fornire una soluzione esatta per ognuno di essi, tale per cui la loro somma restituisce il costo totale.

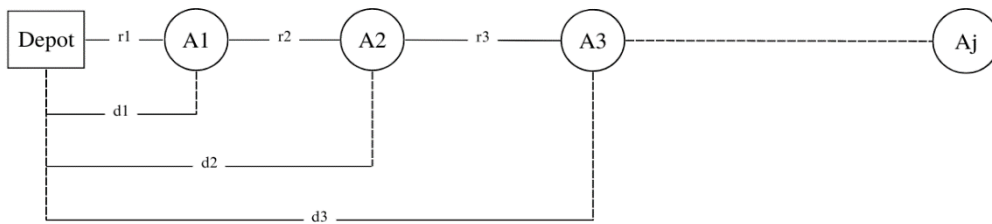
$$11 + 8.17 + 5 + 4.83 = 29 = C_{tot}$$

B) ApproShapley O(1) – prima approssimazione

Come già anticipato, il problema principale, nell'applicare lo Shapley value a questa tipologia di problemi, è la complessità. Questa verrà analizzata meglio nel capitolo 3, dove verranno lanciate delle simulazioni con Python per dimostrare le ipotesi presentate. Nel paper “*Efficient computation of cost allocation of VRPs*” di Dan C. Popescu & Philip Kilby, vengono proposte due soluzioni che approssimano il valore di Shapley con un errore accettabile, ma che mantengono una complessità computazionale relativamente contenuta. Per arrivare a descrivere la prima approssimazione, il paper ipotizza di passare per il caso in cui tutti i nodi si trovino sullo stesso percorso, questo approccio viene definito Airport problem.

Airport problem – caso monodimensionale

L'airport problem, formulato da Littlechild nel 1973 è ampiamente noto in letteratura come un caso particolare dello Shapley value. Come ipotesi di base, dati N giocatori presenti, il giocatore 1 prende ruolo di deposito (0), mentre gli altri N-1 sono i clienti da servire, come un qualunque problema di TSP, con la differenza però che si ha a disposizione una sola “rotta” (un'unica pista). Assegniamo al deposito la lettera D, mentre gli altri N-1 nodi da servire saranno i clienti A_j , con j che va da 1 a n. Per cui, come si può notare in figura, è presente un vettore delle distanze d_j per ogni nodo j e un vettore r_j che corrisponde alla distanza incrementale che separa un cliente dal suo predecessore. Pertanto, r_j non è altro che il *contributo marginale* che il giocatore j apporta alla soluzione finale e possiamo rappresentarlo come:



$$r_1 = d_1$$

Figura 3.2 – Rappresentazione Airport Problem

$$r_j = d_j - d_{j-1} \quad \text{for } (2 \leq j \leq n) \quad \text{Formula 3.3}$$

Questo scenario unidimensionale è noto in letteratura sotto il nome di Airport problem e il valore di Shapley, in questo particolare caso è descritto come:

$$\phi_k = \sum_{j=1}^k \frac{2r_j}{n-j+1}$$

Formula 3.4

Il calcolo dello Shapley value per ogni nodo da servire A_j rappresenta pertanto come sia possibile ripartire equamente i costi di trasporto. Questo perché:

- Tutti gli n clienti, avranno l'onere di pagare $2r_1$, che rappresenta il viaggio di andata e ritorno dal deposito al nodo A_1 e il costo sarà ripartito tra tutti gli n clienti.
- Tutti gli $n - 1$ clienti (tranne A_1), avranno l'onere di pagare $2r_2$ e il costo sarà ripartito tra tutti gli $n - 1$ clienti.
- Tutti gli $n - 2$ clienti (tranne A_1 e A_2), avranno l'onere di pagare $2r_3$ e il costo sarà ripartito tra tutti gli $n - 2$ clienti.

Questo fino a raggiungere il cliente A_n . Come è già stato riportato, il valore r_j rappresenta il *contributo marginale* che il giocatore j apporta alla soluzione finale, ma può anche essere considerato come “*harmonic addition*”, ovvero quel costo naturale aggiunto in proporzione al numero di giocatori, tra i quali è condiviso quel tratto di strada da servire.

Riportiamo in seguito un piccolo esempio per rendere più fruibile il funzionamento dell'algoritmo e che riprenderemo in seguito con le assunzioni che andranno fatte:

Si considerino $N = 4$ nodi, con un deposito D e 3 nodi da servire (A_1, A_2, A_3). La strada che serve tutti gli n clienti è unica condivisa tra tutti e i tratti che separano gli N nodi sono equidistanti ($r_1 = 5, r_2 = 5, r_3 = 5$) generando la seguente matrice delle distanze ($d_1 = 5, d_2 = 10, d_3 = 15$). Ovviamente la condizione del TSP è che si debba partire e rientrare al deposito, per cui la massima distanza percorsa sarà $2d_3 = 30$. Nella forma originale dello Shapley value, quest'ultimo valore non è altro che $v(N)$ e dato il fatto che la validità dei 3 assiomi non viene in alcun modo alterata, allora $\sum_{i \in N} \phi_i = v(N) = 2d_n$.

- $\phi_1 = \sum_{j=1}^1 \frac{2r_j}{n-j+1} = \frac{2r_1}{3-1+1} = \frac{10}{3} = 3.33$
- $\phi_2 = \sum_{j=1}^2 \frac{2r_j}{n-j+1} = \frac{2r_1}{3-1+1} + \frac{2r_2}{3-2+1} = \frac{10}{3} + \frac{10}{2} = 8.33$
- $\phi_3 = \sum_{j=1}^3 \frac{2r_j}{n-j+1} = \frac{2r_1}{3-1+1} + \frac{2r_2}{3-2+1} + \frac{2r_3}{3-3+1} = \frac{10}{3} + \frac{10}{2} + \frac{10}{1} = 18.33$

Si nota subito che l'assioma 3 è rispettato ($\sum_{i \in N} \phi_i = 2d_n = 30$) e dal risultato è evidente che servire il cliente 3, risulta essere, come ci potevamo aspettare, l'operazione che incide maggiormente sul costo totale.

Nello studio di Dan C. Popescu e Philip Kilby “*Approximation of the Shapley value for the Euclidean travelling salesman game*” del 2020 si prende in esame l’“*Airport problem*” e si fornisce un’interpretazione diversa, con lo scopo di utilizzare la stessa intuizione per formulare una possibile soluzione nel caso più complesso multidimensionale, che sarà lo scopo anche di questa ricerca, ma andiamo per gradi.

Per sostituzione diretta dei valori r_j dalla [3.3] nella [3.4], è possibile ottenere lo Shapley value del nodo coda A_n .

$$\phi_n = 2d_n - \sum_{i=1}^{n-1} \frac{2d_{n-i}}{i(i+1)}$$

Formula 3.5

La logica alla base sta nel fatto di prendere in esame la distanza massima, ovvero il costo che A_n pagherebbe naturalmente se fosse da solo e sottrarvici il *contributo condiviso* di tutti gli altri partecipanti, rappresentato dalla somma ponderata sui predecessori.

Riprendendo l’esempio precedente:

$$\bullet \quad \phi_3 = 2d_3 - \sum_{i=1}^2 \frac{2d_{3-i}}{i(i+1)} = 30 - \left(\frac{2d_2}{1(1+1)} + \frac{2d_1}{2(2+1)} \right) = 30 - \left(\frac{2*10}{2} + \frac{2*5}{6} \right) = 18.33$$

Possiamo estrapolare, da questo principio, una funzione di *contributo condiviso* Sc per un determinato numero di fattori $X_i = [x_1, x_2, \dots, x_k]$.

$$Sc(x_1, x_2, \dots, x_k) = \sum_{i=1}^k \frac{y_i}{i(i+1)}$$

Formula 3.6

Partendo dalla [3.6], l’operazione $2d_{n-i}$ “inverte” il vettore preso in esame, per cui è stato scelto il vettore $Y_i = [y_1, y_2, \dots, y_k]$ che rappresenta il vettore $X_i = [x_1, x_2, \dots, x_k]$ ordinato in maniera decrescente. A questo punto possiamo riscrivere la [3.6] come segue:

$$\phi_n = 2d_n - Sc(2d_1, 2d_2, \dots, 2d_{n-1})$$

Formula 3.7

Come fatto in precedenza, viene presentato di seguito lo stesso esempio per comprendere meglio l'operazione conseguita.

- $\phi_3 = 2d_3 - Sc(2d_1, 2d_2)$
- $Sc(x_1, x_2) = Sc(2d_1, 2d_2) = \frac{y_2}{1(1+1)} + \frac{y_1}{2(2+1)} = \frac{2d_2}{1(1+1)} + \frac{2d_1}{2(2+1)}$
- $\phi_3 = 2d_3 - \left(\frac{2d_2}{2} + \frac{2d_1}{6}\right) = 30 - \left(\frac{20}{2} + \frac{10}{6}\right) = 18.33$

È stato estrapolato il potenziale del *contributo condiviso*, per cui si estrapola e lo si utilizza per calcolare lo Shapley value non solo per l'ultimo valore n, ma anche per il generico valore k, come

$$\phi_k = 2d_k - Sc(2d_1, 2d_2, \dots, 2d_{k-1}, 2d_k, \dots, 2d_k)$$

Formula 3.8

La logica alla base è sempre la stessa: preso un generico cliente k, si estrae la distanza massima che bisognerebbe percorrere nel caso in cui k fosse da solo, e si sottrae il *contributo condiviso* di tutti i partecipanti che lo hanno preceduto. Da notare il fattore $2d_k$ appare $n - k$ volte nella funzione Sc come il contributo che va da $k + 1$ ad n.

Per completare l'esempio:

- $\phi_1 = 2d_1 - Sc(2d_1, 2d_1)$ Nota: $2d_1$ appare 2 volte dato che $n - k = 2$
- $\phi_1 = 2d_1 - \left(\frac{2d_1}{2} + \frac{2d_1}{6}\right) = 10 - \left(\frac{10}{2} + \frac{10}{6}\right) = 3.33$
- $\phi_2 = 2d_2 - Sc(2d_1, 2d_2)$ Nota: $2d_2$ appare 1 volta dato che $n - k = 1$
- $\phi_2 = 2d_2 - \left(\frac{2d_2}{2} + \frac{2d_1}{6}\right) = 20 - \left(\frac{20}{2} + \frac{10}{6}\right) = 3.33$

Sono stati forniti quindi due metodi equivalenti, [3.5] e [3.8], per il calcolo dello Shapley value nel caso monodimensionale. Ultimo step da considerare è capire cosa succederebbe se si dovesse andare a collegare un altro problema, di forma e definizione identica, dalla parte opposta del deposito D.

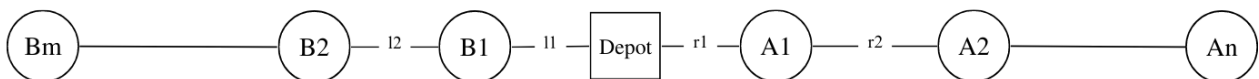


Figura 3.3 – Rappresentazione Airport Problem con precorso opposto all'originale

Come si nota in figura, a sinistra del deposito esiste un vettore di clienti B_j , con j che va da 1 a m e con un vettore l_j che corrisponde alla distanza incrementale che separa un cliente dal suo predecessore.

Se volessimo calcolare il valore di Shapley per un generico cliente k del problema A , allora l'equazione prenderebbe forma:

$$\phi_{A_k} = \sum_{j=1}^n \lambda_{kj} r_j + \sum_{j=1}^m \mu_{kj} l_j$$

Formula 3.9

I coefficienti $\lambda_{kj} \geq 0$ e $\mu_{kj} \geq 0$ dipenderanno solo e soltanto della posizione relativa del generico k preso in esame e dal loro numero di predecessori. La prima sommatoria, infatti, identifica la parte sinistra del deposito nel grafico, la seconda invece la parte destra del grafico. Nel caso specifico della [3.9] si sta cercando ϕ di un generico k -esimo elemento di A e si può dimostrare che tutti i coefficienti λ debbano essere esattamente equivalenti della [3.5] e di conseguenza tutti i coefficienti μ dovranno essere pari 0. Questo ragionamento vale anche per un k -esimo elemento di B , ma il fulcro della questione, è che anche se vi sono due coalizioni diverse di clienti da servire in due casi mododimensionali distinti, il loro Shapley value non dipenderà dai valori assunti dalla controparte.

Si è arrivati quindi alla conclusione che il caso di studio dell'“airport problem”, e quindi il caso monodimensionale del problema del TSP, si può risolvere scomponendolo in due “giochi” distinti. Da un lato avremo A_i clienti situati alla destra del deposito, dall'altro avremo B_i clienti situati a sinistra del deposito. Per il calcolo dello Shapley value di un generico A_k , è possibile utilizzare o la [3.5] o la [3.8], prendendo in considerazione tutti i clienti A_j , con j che va da 1 a n e tralasciando i clienti B_j . Allo stesso modo, per il calcolo dello Shapley value di un generico B_k , è possibile utilizzare o la [3.5] o la [3.8], prendendo in considerazione tutti i clienti B_j , con j che va da 1 a m e tralasciando i clienti A_j . Questa indipendenza sarà fondamentale nel caso multidimensionale.

Nota sulla complessità 2

In questo caso mono-dimensionale abbiamo trovato algoritmi a minor complessità per il calcolo dello Shapley value. La complessità computazionale oscilla tra $O(n)$, nel caso in cui i clienti siano ordinati in funzione della loro distanza dal deposito, e $O(n \log n)$, nel caso in cui vi sia la necessità a monte di un ordinamento. Vi è una riduzione computazionale notevole rispetto al caso di partenza, nonostante ovviamente ci si sia posti in una condizione monodimensionale che se da un lato ci permette di calcolare ϕ in tempi polinomiali, dall'altro non risulta essere un'approssimazione accettabile.

La forma di Appro O(1) nel caso multidimensionale

Prima di proseguire con il caso multidimensionale, è necessario spostare l'attenzione sul fatto che tutti gli esperimenti da qui in poi sono incentrati sul caso in due dimensioni. Ci si affida pertanto solo sulle distanze reciproche tra le località e sull'ipotesi che la matrice delle distanze sia identica e che soddisfi la disuguaglianza triangolare. La scelta consiste nell'analizzare il caso non euclideo delle reti stradali (TSP).

Si assuma quindi di partire da un deposito D marcato come 0 e dover servire n clienti A_i ($1 \leq i \leq n$), con una matrice delle distanze simmetrica $(n + 1) * (n + 1)$.

Per arrivare alla prima approssimazione, si consideri il caso in cui si hanno due clienti A_1 e A_2 e applichiamo per la prima volta lo Shapley value ad un problema di TSP non monodimensionale.

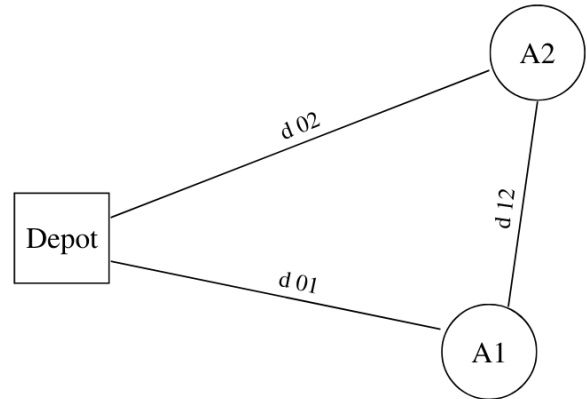


Figura 3.4 – Generico grafo 2 clienti

$$\phi_i(v) = \frac{1}{n!} \sum_{\pi \in S_n} (v(\pi_i \cup \{i\}) - v(\pi_i))$$

È necessario tenere in considerazione il principio del TSP, secondo il quale la partenza e l'arrivo nel tour devono essere effettuati nel medesimo nodo, nel nostro caso il deposito 0. Partendo dalla matrice delle distanze, pertanto, ci costruiamo la *funzione di costo* così come spiegato in precedenza.

Il principio è reso più comprensibile con il prosieguo dell'esempio:

Funzione di costo	Valore
$v(0)$	0
$v(1)$	$2d_{01}$
$v(2)$	$2d_{02}$
$v(1, 2)$	$d_{01} + d_{12} + d_{02}$

Tabella 3.7 – Valore della FdC

	i = 1		i = 2	
Permutazioni	Contributo marginale	Calcolo	Contributo marginale	Calcolo
1, 2	$v(1) - v(0)$	$2d_{01}$	$v(1,2) - v(1)$	$d_{01} + d_{12} + d_{02} - 2d_{01}$
2, 1	$v(1,2) - v(2)$	$d_{01} + d_{12} + d_{02} - 2d_{02}$	$v(2) - v(0)$	$2d_{02}$
	Somma:	$3d_{01} - d_{02} + d_{12}$	Somma:	$3d_{02} - d_{01} + d_{12}$

Tabella 3.8 – Risultati espressi in funzione delle distanze

I valori dello Shapley value dei nodi 1 e 2 sono i seguenti:

- $\phi_1(v) = \frac{1}{2}(3d_{01} - d_{02} + d_{12})$
- $\phi_2(v) = \frac{1}{2}(3d_{02} - d_{01} + d_{12})$

Che possono essere riscritti come:

- $\phi_1(v) = 2d_{01} - \frac{1}{2}(d_{01} + d_{02} - d_{12})$
- $\phi_2(v) = 2d_{02} - \frac{1}{2}(d_{01} + d_{02} - d_{12})$

A questo punto è possibile definire la matrice simmetrica $n * n$ che esprime la *distanza condivisa* tra i clienti i e j .

$$s_{ij} = d_{0i} + d_{0j} - d_{ij}$$

La *distanza condivisa* che è stata trovata riscontra delle analogie con il *contributo condiviso* Sc del paragrafo precedente. Riconoscendo il potenziale di quest'ultimo, è possibile trovare un'approssimazione dello Shapley value che sfrutti questo principio. Tornando nel caso monodimensionale, infatti, è possibile notare come la *distanza condivisa* nel caso i clienti siano in fila sia il doppio della distanza massima, mentre nel caso in cui i clienti si trovino agli estremi del deposito, questa *distanza condivisa* sia 0.

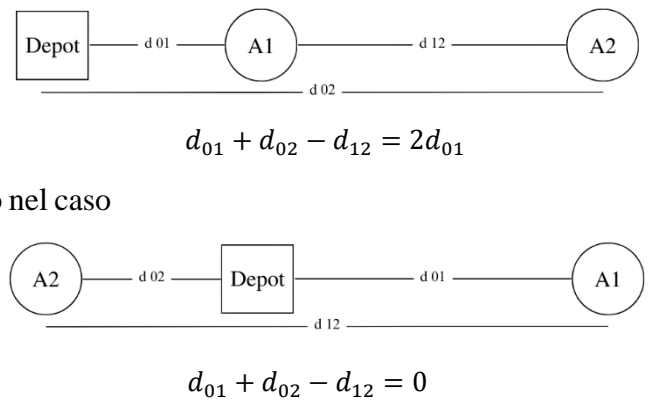


Figure 3.5 – Distanza condivisa coppie opposte

Se a questo ci aggiungiamo che la *distanza condivisa* di un cliente con sé stesso, è esattamente il doppio della propria ($s_{ii} = d_{0i} + d_{0i} - d_{ii} = 2d_{0i}$), allora è possibile riscrivere il risultato dello Shapley value per i clienti 1 e 2 come:

- $\phi_1(v) = s_{11} - \frac{1}{2} s_{12}$
- $\phi_2(v) = s_{22} - \frac{1}{2} s_{12}$

Riprendendo ora il *contributo marginale* dal caso monodimensionale $Sc(x_1, x_2, \dots, x_k) = \sum_{i=1}^k \frac{y_i}{i(i+1)}$ e applicandolo al caso specifico, questo diventa $Sc(s_{12}) = \frac{s_{12}}{2}$.

- $\phi_1(v) = s_{11} - Sc(s_{12})$
- $\phi_2(v) = s_{22} - Sc(s_{12})$

È chiaro che questo funzioni nel caso in cui si abbiano 2 clienti, ma se volessimo estenderlo ad un caso generale e quindi rifacendosi alla [1.8], lo Shapley value prenderebbe forma come:

$$\phi_k^{(1)} = s_{kk} - Sc(s_{1,k}, s_{2,k}, \dots, s_{k-1,k}, s_{k+1,k}, \dots, s_{n,k})$$

Formula 3.10

$\phi_k^{(1)}$ è quindi l'approssimazione di ordine 1 dello Shapley value, anche detta Appro (1). Essenzialmente viene fornito un valore approssimato $\phi_k^{(1)}$ del cliente k partendo dalla *distanza condivisa* della posizione A_k con se stesso ($s_{kk} = 2d_{0k}$) a meno del *contributo condiviso* di tutte le *distanze condivise* tra A_k e tutti gli altri A_i ($i \neq k$) clienti.

Si può notare come questo caso copra interamente il caso monodimensionale; questo accade perché se i clienti si trovano ai lati del deposito e quindi non devono percorrere tratti di strada in comune, la *distanza condivisa* s_{ij} è pari a zero. Altro elemento da considerare è che nel caso in cui i clienti siano 2, allora $\phi_k^{(1)}$ è esattamente pari allo Shapley value nella sua forma originale (risulta essere anche abbastanza ovvio, dato che questo è stato il caso di partenza nella nostra trattazione), mentre per $n > 2$, il valore approssimato si allontana in funzione del denominatore della formula del *contributo condiviso* $Sc(x_1, x_2, \dots, x_k) = \sum_{i=1}^k \frac{y_i}{i(i+1)}$.

Nel momento in cui si voglia utilizzare Appro (1) per una corretta ripartizione dei costi, questa ha la proprietà di non generare discrepanza significative per clienti che siano vicini tra loro, trattandoli in maniera simile.

Esempio

Viene presentato l'esempio con $n = 4$ nodi, riportando la stessa matrice delle distanze utilizzata per l'esempio nell'applicazione di Shapley classico. Per il momento non ci si focalizzerà sui risultati che verranno generati, dato che saranno maggiormente approfonditi nel Capitolo 4, nel quale saranno iterate molteplici applicazioni pratiche. In questa sede per una maggiore chiarezza, ci limitiamo a ricordare come tali risultati fossero:

- $\phi_1(v) = \frac{264}{24} = 11$
- $\phi_2(v) = \frac{196}{24} = 8.17$
- $\phi_3(v) = \frac{120}{24} = 5$
- $\phi_4(v) = \frac{116}{24} = 4.83$

Distanze	Valori
d_01	11
d_02	12
d_03	10
d_04	5
d_12	3
d_13	15
d_14	17
d_23	6
d_24	19
d_34	4

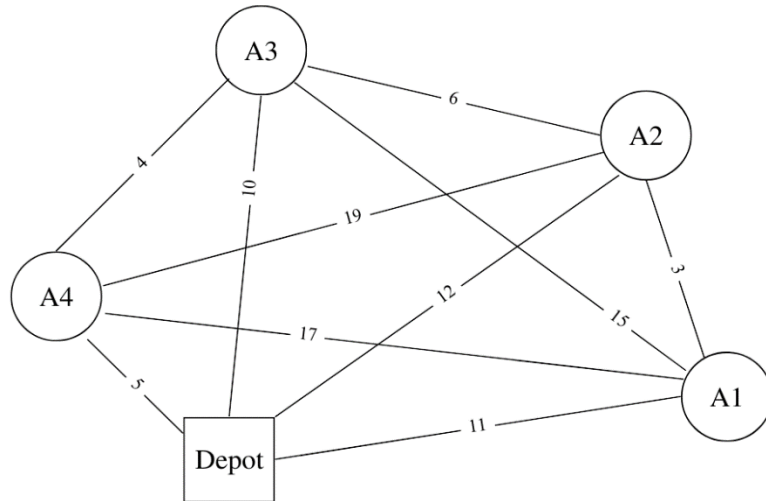


Tabella 3.9 – Matrice distanze

Figura 3.6 – Grafo completo

Il costo ottimo della soluzione del TSP per visitare tutti i nodi è $[0 - 1 - 2 - 3 - 4 - 0]$ con costo totale $C_{tot} = 29$. Per applicare Appro (1): $\phi_k^{(1)} = s_{kk} - Sc(s_{1,k}, s_{2,k}, \dots, s_{k-1,k}, s_{k+1,k}, \dots, s_{n,k})$ la prima iterazione consiste nel calcolare la matrice delle distanze condivise $s_{ij} = d_{0i} + d_{0j} - d_{ij}$

Distanze condivise	$d_{0i} + d_{0j} - d_{ij}$	Valore
s_11	$2d_{01}$	22
s_22	$2d_{02}$	24
s_33	$2d_{03}$	20
s_44	$2d_{04}$	10
s_12	$d_{01} + d_{02} - d_{12}$	20
s_13	$d_{01} + d_{03} - d_{13}$	6
s_14	$d_{01} + d_{04} - d_{14}$	-1
s_23	$d_{02} + d_{03} - d_{23}$	16
s_24	$d_{02} + d_{04} - d_{24}$	-2
s_34	$d_{03} + d_{04} - d_{34}$	11

Tabella 3.10 – Calcolo delle distanze condivise

Vengono scritti di seguito i quattro Appro (1) per gli $n = 4$ nodi:

- $\phi_1^{(1)} = s_{11} - Sc(s_{21}, s_{31}, s_{41}) = s_{11} - Sc(s_{12}, s_{13}, s_{14})$
- $\phi_2^{(1)} = s_{22} - Sc(s_{12}, s_{32}, s_{42}) = s_{22} - Sc(s_{12}, s_{23}, s_{24})$
- $\phi_3^{(1)} = s_{33} - Sc(s_{13}, s_{23}, s_{43}) = s_{33} - Sc(s_{13}, s_{23}, s_{34})$
- $\phi_4^{(1)} = s_{44} - Sc(s_{14}, s_{24}, s_{34}) = s_{44} - Sc(s_{14}, s_{24}, s_{34})$

Applicando $Sc(x_1, x_2, \dots, x_k) = \sum_{i=1}^k \frac{y_i}{i(i+1)}$ con $Y_i = [y_1, y_2, \dots, y_k]$ che rappresenta il vettore $X_i = [x_1, x_2, \dots, x_k]$ ordinato in maniera decrescente, riscriviamo:

- $\phi_1^{(1)} = 2d_{01} - \left(\frac{s_{12}}{1(1+1)} + \frac{s_{13}}{2(2+1)} + \frac{s_{14}}{3(3+1)} \right) = 2d_{01} - \left(\frac{s_{12}}{2} + \frac{s_{13}}{6} + \frac{s_{14}}{12} \right)$ con $(s_{12} > s_{13} > s_{14})$
- $\phi_2^{(1)} = 2d_{02} - \left(\frac{s_{12}}{1(1+1)} + \frac{s_{23}}{2(2+1)} + \frac{s_{24}}{3(3+1)} \right) = 2d_{02} - \left(\frac{s_{12}}{2} + \frac{s_{23}}{6} + \frac{s_{24}}{12} \right)$ con $(s_{12} > s_{23} > s_{24})$
- $\phi_3^{(1)} = 2d_{03} - \left(\frac{s_{23}}{1(1+1)} + \frac{s_{34}}{2(2+1)} + \frac{s_{13}}{3(3+1)} \right) = 2d_{02} - \left(\frac{s_{23}}{2} + \frac{s_{34}}{6} + \frac{s_{13}}{12} \right)$ con $(s_{23} > s_{34} > s_{13})$
- $\phi_4^{(1)} = 2d_{04} - \left(\frac{s_{34}}{1(1+1)} + \frac{s_{14}}{2(2+1)} + \frac{s_{24}}{3(3+1)} \right) = 2d_{02} - \left(\frac{s_{34}}{2} + \frac{s_{14}}{6} + \frac{s_{24}}{12} \right)$ con $(s_{34} > s_{14} > s_{24})$

Sostituendo i rispettivi valori numerici si ottiene:

- $\phi_1^{(1)} = 22 - \left(\frac{20}{2} + \frac{6}{6} + \frac{-1}{12} \right) = 11.08$
- $\phi_2^{(1)} = 24 - \left(\frac{20}{2} + \frac{16}{6} + \frac{-2}{12} \right) = 11.5$
- $\phi_3^{(1)} = 20 - \left(\frac{16}{2} + \frac{11}{6} + \frac{6}{12} \right) = 9.66$
- $\phi_4^{(1)} = 10 - \left(\frac{11}{2} + \frac{-1}{6} + \frac{-1}{12} \right) = 4.83$

Avendo utilizzato una forma approssimata, viene meno l'assioma 2.a (efficiency axiom). Infatti, la somma dei 4 risultati ottenuti non estrapola il Ctot, ma un valore che si avvicina a Ctot, con un certo margine di errore. Pertanto, tali risultati andranno normalizzati sul valore totale del costo di percorrenza. Come già precedentemente detto, questo aspetto verrà approfondito nei successivi capitoli relativi ai calcoli.

C) ApproShapley O(2) – seconda approssimazione

Come discusso nel paragrafo “*applicare ApproShapley ad un problema di TSP*”, nel momento in cui si applichi Shapley ad un problema di TSP, uno dei fattori incisivi sulla complessità computazionale nella risoluzione, consiste nel ricavare la *funzione di costo* a partire dalla matrice delle distanze. Appro (1) fornisce un primo grado di approssimazione introducendo il concetto di *distanza condivisa*, la quale, solo sottospecifiche condizioni, risulta combaciare con lo Shapley value. Al fine di trovare un’approssimazione di ordine maggiore, è stato necessario trovare un parallelismo tra il *contributo marginale* $v(\pi_i \cup \{i\}) - v(\pi_i)$, generato dalla *funzione di costo* $v()$ e la *distanza condivisa* $s_{ij} = d_{0i}$. Per spiegare come questo possa essere possibile, si prenda in considerazione il seguente esempio:

Esempio:

Prendiamo ora in esame il caso che prevede $n = 3$ clienti con un deposito di partenza e una matrice delle distanze dell’ordine $(n + 1) * (n + 1)$. Supponiamo che da questa matrice delle distanze si ricavi la funzione di costo $v()$ per ogni nodo. È possibile notare come, per ogni permutazione $n!$ delle soluzioni, si possano dividere le $v()$ in n coalizioni. Queste rispecchiano il numero di nodi preso in esame.

Si prenda sempre il caso con tre clienti e si analizzi solo A1, secondo la regola dello Shapley value, esistono 3 coalizioni di cui 1 non fa parte:

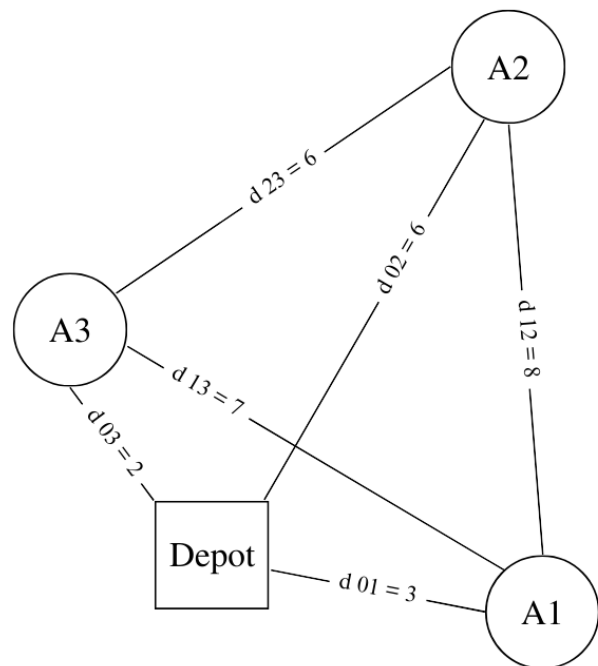
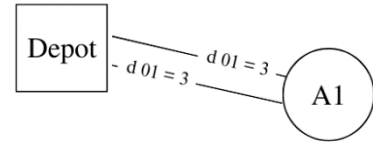


Figura 3.7 – Grafo completo 3 clienti

- **Coalizione 1:** il cliente 1 è preso da solo e quindi la S è vuoto ($S = \{\emptyset\}$) e quindi la *funzione di costo* è pari alla distanza percorsa in andata e in ritorno

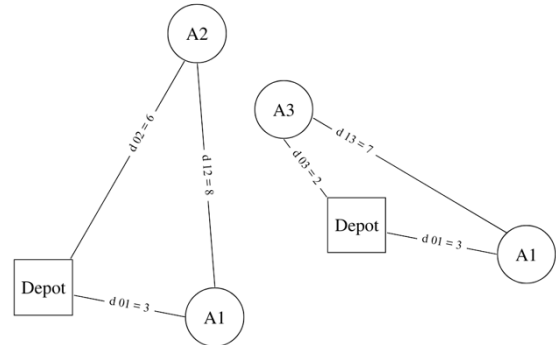
$$v(1) = 2d_{01}$$



- **Coalizione 2:** il cliente 1 è preso in sequenza con uno degli altri clienti e quindi l'insieme S avrà cardinalità pari a 2 ($S = \{2; 3\}$). Avremo pertanto due *funzioni costo*:

$$v(1, 2) = d_{01} + d_{12} + d_{02}$$

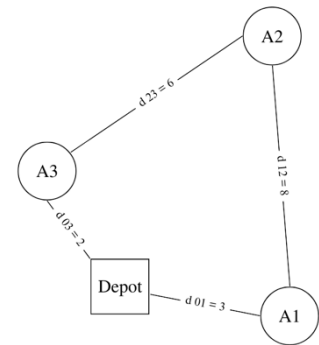
$$v(1, 3) = d_{01} + d_{13} + d_{03}$$



- **Coalizione 3:** il cliente 1 è preso in sequenza con gli altri 2 clienti e quindi l'insieme S avrà cardinalità pari a 1 ($S = \{23\}$). A differenza della coalizione 2 però, non vi è un unico percorso che permetta di andare a “toccare” tutti e tre i clienti, ma è necessario analizzarne 3. Essendo nel caso del TSP è necessario trovare il percorso minore per servire tutti e tre i clienti.

$$v(1, 2, 3) = \min\{p_1, p_2, p_3\} = p_1 \text{ nel nostro caso.}$$

- $p_1 = d_{01} + d_{12} + d_{23} + d_{03}$
- $p_2 = d_{01} + d_{13} + d_{23} + d_{02}$
- $p_3 = d_{02} + d_{12} + d_{13} + d_{03}$



NOTA: ricordiamo che $v(1, 2, 3)$ corrisponde alla soluzione del problema del TSP. Più in generale, nel caso di n clienti da servire, la *funzione costo* $v(N)$ corrisponde alla soluzione del problema di TSP.

Viene calcolato lo Shapley value, con la seguente matrice delle distanze:

	0	A ₁	A ₂	A ₃
0	/	3	6	2
A ₁	3	/	8	7
A ₂	6	8	/	6
A ₃	2	7	6	/

Tabelle 3.11 – Matrice distanze e calcolo della funzione di costo

Distanze	Valori
d ₀₁	3
d ₀₂	6
d ₀₃	2
d ₁₂	8
d ₁₃	7
d ₂₃	6

Funzione di costo	Calcolo in funzione delle distanze	Valore
v(0)	0	0
v(1)	$2d_{01}$	6
v(2)	$2d_{02}$	12
v(3)	$2d_{03}$	4
v(1, 2)	$d_{01} + d_{12} + d_{02}$	17
v(1, 3)	$d_{01} + d_{13} + d_{03}$	12
v(2, 3)	$d_{02} + d_{23} + d_{03}$	14
v(1, 2, 3)	$d_{01} + d_{12} + d_{23} + d_{03}$	19

Si passi al calcolo per ogni i:

	i = 1		i = 2		i = 3	
Permutazioni	Contributo marginale	Calcolo	Contributo marginale	Calcolo	Contributo marginale	Calcolo
1, 2, 3	$v(1) - v(0)$	6	$v(1, 2) - v(1)$	11	$v(1, 2, 3) - v(1, 2)$	2
1, 3, 2	$v(1) - v(0)$	6	$v(1, 2, 3) - v(1, 3)$	7	$v(1, 3) - v(1)$	6
2, 1, 3	$v(1, 2) - v(2)$	5	$v(2) - v(0)$	12	$v(1, 2, 3) - v(1, 2)$	2
2, 3, 1	$v(1, 2, 3) - v(2, 3)$	3	$v(2) - v(0)$	12	$v(2, 3) - v(2)$	2
3, 1, 2	$v(1, 3) - v(3)$	8	$v(1, 2, 3) - v(1, 3)$	7	$v(3) - v(0)$	4
3, 2, 1	$v(1, 2, 3) - v(2, 3)$	5	$v(2, 3) - v(3)$	10	$v(3) - v(0)$	4
	Somma:	35	Somma:	59	Somma:	20

Tabella 3.12 – Calcoli generici per lo Shapley value

- $\phi_1(v) = \frac{35}{6} = 5.83$
- $\phi_2(v) = \frac{59}{6} = 9.83$
- $\phi_3(v) = \frac{20}{6} = 3.33$

Il cliente 3 è sì quello più vicino al deposito, ma anche quello più vicino agli altri clienti in proporzione, per cui lo Shapley value lo premia con un'allocazione di costo inferiore. La valutazione opposta va invece fatta per il cliente 2, che si vede allocare circa il 50% del costo totale.

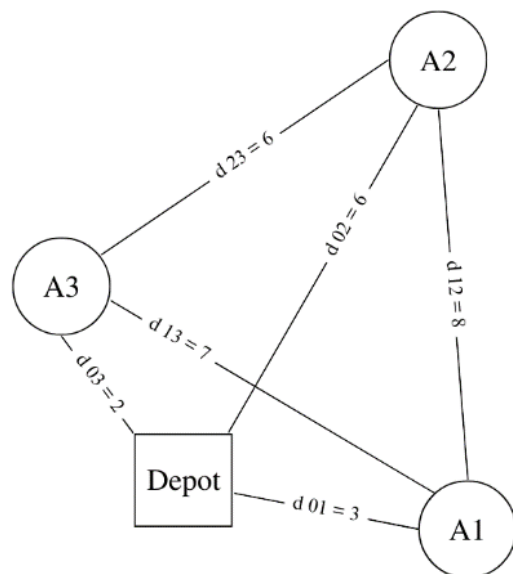


Figura 3.8 – Grafo completo 3 clienti

Tornando all'approssimazione, si riscriva la soluzione di Shapley per il cliente 1 come:

$$\phi_1(v) = \frac{1}{6} * \{[2v(1)] + [(v(1,2) - v(2)) + (v(1,3) - v(3))] + [2(v(1,2,3) - v(2,3))]\}$$

Che è possibile dividere in *funzione del costo* delle 3 coalizioni $C_i^{|S|}$

$$\phi_1(v) = \frac{1}{3} * \left\{ [v(1)] + \frac{1}{2} [(v(1,2) - v(2)) + (v(1,3) - v(3))] + [v(1,2,3) - v(2,3)] \right\}$$

$$\phi_1(v) = \frac{1}{3} * [C_1^0 + C_1^1 + C_1^2]$$

Riscriviamo lo Shapley value utilizzando la *distanza condivisa* tra i clienti i e j introdotto nel paragrafo precedente $s_{ij} = d_{0i} + d_{0j} - d_{ij}$

- **Coalizione 1**

$$C_1^0 = v(1) = 2d_{01} = s_{11}$$

- **Coalizione 2**

$$C_1^1 = \frac{1}{2} [(v(1,2) - v(2)) + (v(1,3) - v(3))] =$$

$$\frac{1}{2} [(d_{01} + d_{12} + d_{02} - 2d_{02}) + (d_{01} + d_{13} + d_{03} - 2d_{03})] =$$

$$\frac{1}{2} [(d_{01} + d_{12} - d_{02}) + (d_{01} + d_{13} - d_{03})] = d_{01} + \frac{1}{2} [d_{12} + d_{13} - d_{02} - d_{03}] =$$

$$2d_{01} - \frac{1}{2} [d_{01} + d_{02} - d_{12} + d_{01} + d_{03} - d_{13}] = s_{11} - \frac{1}{2} (s_{12} + s_{13})$$

- **Coalizione 3**

$$C_1^2 = \min\{p_1, p_2, p_3\} - v(2,3)$$

Poniamoci nel caso in cui il percorso p_1 sia il minore:

$$p_1 \leq p_2 \quad d_{01} + d_{12} + \cancel{d_{23}} + d_{03} \leq d_{01} + d_{13} + \cancel{d_{23}} + d_{02}$$

$$d_{01} + d_{03} - d_{13} \leq d_{01} + d_{02} - d_{12} \quad s_{13} \leq s_{12}$$

$$p_1 \leq p_3 \quad d_{01} + \cancel{d_{12}} + d_{23} + d_{03} \leq d_{02} + \cancel{d_{12}} + d_{13} + d_{03}$$

$$d_{01} + d_{03} - d_{13} \leq d_{02} + d_{03} - d_{23} \quad s_{13} \leq s_{23}$$

Ecco che se il percorso p_1 è il minore, la *distanza condivisa* s_{13} è la più piccola delle tre, mentre se il percorso minore fosse uno degli altri due, risulterebbe:

- $\min\{p_1, p_2, p_3\} = p_1 \rightarrow \min\{s_{12}, s_{13}, s_{23}\} = s_{13}$

- $\min\{p_1, p_2, p_3\} = p_2 \rightarrow \min\{s_{12}, s_{13}, s_{23}\} = s_{12}$

- $\min\{p_1, p_2, p_3\} = p_3 \rightarrow \min\{s_{12}, s_{13}, s_{23}\} = s_{23}$

$$C_1^2 = \min\{p_1, p_2, p_3\} - (d_{02} + d_{23} + d_{03})$$

$$d_{01} + d_{12} + d_{23} + d_{03} - d_{02} - d_{23} - d_{03}$$

$$2d_{01} - d_{01} - d_{02} + d_{12} - d_{01} - d_{03} + d_{13} + d_{01} + d_{03} - d_{13}$$

$$2d_{01} - (d_{01} + d_{02} - d_{12} + d_{01} + d_{03} - d_{13}) + d_{01} + d_{03} - d_{13}$$

$$s_{11} - (s_{12} + s_{13}) + s_{13} \text{ e nel caso generale:}$$

$$s_{11} - (s_{12} + s_{13}) + \min\{s_{12}, s_{13}, s_{23}\}$$

Alla fine, $\phi_1(v) = \frac{1}{3} * [C_1^0 + C_1^1 + C_1^2]$ diventa:

$$\phi_1(v) = \frac{1}{3} [s_{11} + s_{11} - \frac{1}{2} [s_{12} + s_{13}] + s_{11} - (s_{12} + s_{13}) + \min\{s_{12}, s_{13}, s_{23}\}]$$

$$\phi_1(v) = s_{11} - \frac{1}{2} (s_{12} + s_{13}) + \frac{1}{3} \min\{s_{12}, s_{13}, s_{23}\}$$

Data la sua natura, questa riscrittura dello Shapley value coincide con l'Appro (1) , nel caso in cui il valore $+\frac{1}{3} \min\{s_{12}, s_{13}, s_{23}\}$ sia o s_{12} o s_{13} . Ci si ponga nel caso in cui il minore dei tre sia s_{12} :

$$\phi_1^{(1)} = s_{11} - Sc(s_{21}, s_{31}) = 2d_{01} - \frac{s_{13}}{2} + \frac{s_{12}}{6} = s_{11} - \frac{1}{2} (s_{12} + s_{13}) + \frac{1}{3} s_{12} = \phi_1(v)$$

Pertanto, la soluzione Appro (1) avrà un risultato diverso dal valore esatto dello Shapley value, solo nel caso in cui il minore delle distanze marginali sia s_{23} e dipenderà strettamente da questa. Una volta constatato questo, è possibile riscrivere la funzione approssimata Appro (1) nella seguente maniera:

$$\phi_1^{(1)} = s_{11} - \frac{1}{2} (s_{12} + s_{13}) + \frac{1}{3} \min\{s_{12}, s_{13}\} = s_{11} - \frac{1}{2} (s_{12} + s_{13}) + \frac{1}{3} \cap\{s_{12}, s_{13}\}$$

Formula 3.11

Dove $\cap\{s_{12}, s_{13}\}$ corrisponde all'intersezione formale delle due distanze condivise s_{12} e s_{13} . È quindi possibile fornire una generalizzazione del caso di partenza [1.10] come:

$$\phi_k = s_{kk} - \left(\frac{1}{2} \sum_{i \neq k} \{s_{ki}\} - \frac{1}{3} \sum_{i \neq k \neq j} \cap \{s_{ki}, s_{kj}\} + \frac{1}{4} \sum_{i \neq k \neq j \neq l} \cap \{s_{ki}, s_{kj}, s_{kl}\} - \frac{1}{5} [\dots] \right)$$

Formula 3.12

Questa riscrittura è concettualmente identica allo Shapley value, mantenendo la stessa complessità. L'unica differenza è che la si è riscritta sostituendo il *contributo marginale* con la *distanza condivisa*. La complessità si sposta in $\cap \{s_{ki}, s_{kj}, s_{kl}, \dots\}$ il quale non è di facile risoluzione, ecco che Appro O(2) propone la seguente approssimazione:

Per $n = 4$ nodi e preso in esame il nodo 1:

$$\phi_k = s_{kk} - \left(\frac{1}{2} \sum_{i \neq k} \{s_{ki}\} - \frac{1}{3} \sum_{i \neq k \neq j} \cap \{s_{ki}, s_{kj}\} + \frac{1}{4} \sum_{i \neq k \neq j \neq l} \cap \{s_{ki}, s_{kj}, s_{kl}\} \right) =$$

$$\phi_k = s_{kk} - \frac{1}{2} (s_{12} + s_{13} + s_{14}) + \frac{1}{3} [\cap \{s_{12}, s_{13}\} + \cap \{s_{12}, s_{14}\} + \cap \{s_{13}, s_{14}\}] - \frac{1}{4} \cap \{s_{12}, s_{13}, s_{14}\}$$

La complessità si ritrova in $\cap \{s_{12}, s_{13}, s_{14}\}$. Abbiamo visto come sia possibile ottenere un'approssimazione migliore di Appro (1) sfruttando i valori noti delle 2-intersezioni formali. Si approssima ogni k-intersezione formale di distanze condivise con il valore minimo dei suoi costituenti. Possiamo imitare lo stesso processo di approssimazione al livello superiore, passando dal coinvolgere 2-intersezioni formali, ad ordini di 3 o superiore. L'ipotesi migliore è approssimare qualsiasi intersezione formale di ordine superiore a 2 per il valore più basso di tutte le 2 intersezioni di qualsiasi 2 dei suoi costituenti. In altre parole, approssimiamo qualsiasi m-intersezione formale sconosciuta per $m \geq 3$ come segue:

$$\cap \{s_{ki_1}, s_{ki_2}, \dots, s_{ki_m}\} = \min_{1 \leq r, p \leq m} \{s_{ki_1}, s_{ki_p}\}$$

Per realizzare questa idea in modo rigoroso, dovremmo scansionare tutti i sottoinsiemi di distanze condivise di dimensione $m \geq 3$, per identificare il 2-sottoinsieme che fornisce l'intersezione minima. Sfortunatamente, questo porta a una complessità di calcolo esponenziale, che renderebbe il file di approccio poco pratico per scenari di dimensioni maggiori. La domanda chiave è quindi se sia possibile identificare il coefficiente di ogni intersezione formale senza dover fare affidamento su un calcolo per una ricerca impegnativa.

Vengano rinominate come s_1, s_2, \dots, s_{n-1} le $n - 1$ le distanze condivise di un generico s_{ik} con $i \neq k$ senza preoccuparci troppo inizialmente di quale particolare ordine di indicizzazione scegliamo.

- Fissato il nodo k, le *distanze condivise* $s_{k1}, s_{k2}, \dots, s_{ki}$ diventano s_1, s_2, \dots, s_{n-1}

Si indichi con $I^{(2)}$ la matrice simmetrica $(n - 1) \times (n - 1)$ di 2-intersezioni formali, cioè per $(1 \leq i \neq j \leq n - 1)$ la voce $I^{(2)} = \cap \{s_i, s_j\}$

Esempio:

Approfondiamo il concetto della matrice $I^{(2)}$, utilizziamo un numero di nodi $n = 6$, dove, per convenzione, sono stati contrassegnati gli elementi della matrice con numeri nell'ordine crescente dei valori a 2-intersezioni. Fissato un generico nodo k , si indicizzino righe e colonne della matrice $I^{(2)}$ e, scegliendo arbitrariamente i valori, $\cap \{s_1, s_2\}$ è l'elemento più basso di $I^{(2)}$, $\cap \{s_1, s_3\}$ è il successivo più alto e così via.

- $s_{k1} \rightarrow s_1$
- $s_{k2} \rightarrow s_2$
- $s_{k3} \rightarrow s_3$
- $s_{k4} \rightarrow s_4$
- $s_{k5} \rightarrow s_5$

	s_1	s_2	s_3	s_4	s_5
s_1	/	1	2	4	7
s_2	1	/	3	5	8
s_3	2	3	/	6	9
s_4	4	5	6	/	10
s_5	7	8	9	10	/

Tabella 3.13 – Matrice $I^{(2)}$

Con questo ordine dominato dall'indice delle voci in $I^{(2)}$, è facile individuare quali intersezioni di ordine superiore sono approximate da ciascuna particolare 2-intersezione, secondo $\{s_{ki_1}, s_{ki_2}, \dots, s_{ki_m}\} = \min_{1 \leq r, p \leq m} \{s_{ki_1}, s_{ki_p}\}$.

In $\phi_k = s_{kk} - \left(\frac{1}{2} \sum_{k \neq 1} \{s_{k1}\} - \frac{1}{3} \sum_{k \neq 1 \neq 2} \cap \{s_{k1}, s_{k2}\} + \frac{1}{4} \sum_{k \neq 1 \neq 2 \neq 3} \cap \{s_{k1}, s_{k2}, s_{k3}\} - \frac{1}{5} \sum_{k \neq 1 \neq 2 \neq 3 \neq 4} \cap \{s_{k1}, s_{k2}, s_{k3}, s_{k4}\} + \frac{1}{6} \sum_{k \neq 1 \neq 2 \neq 3 \neq 4 \neq 5} \cap \{s_{k1}, s_{k2}, s_{k3}, s_{k4}, s_{k5}\} \right)$, tutti i sottoinsiemi di intersezione in cui appare la coppia (s_1, s_2) saranno minimamente dominati per $\cap \{s_1, s_2\}$:

- La coppia (s_1, s_2) dominerà minimamente, con la sua intersezione, tutti i sottoinsiemi di intersezione in cui compaiono e possono contenere anche s_3, s_4 e s_5
- Le coppie (s_1, s_3) e (s_2, s_3) domineranno minimamente, con le loro 2 intersezioni, tutti i sottoinsiemi di intersezione in cui compaiono e possono contenere anche s_4 e s_5
- Le coppie (s_1, s_4) , (s_2, s_4) e (s_3, s_4) domineranno minimamente, con le loro 3 intersezioni, tutti i sottoinsiemi di intersezione in cui compaiono e possono contenere anche s_5
- Le coppie (s_1, s_5) , (s_2, s_5) , (s_3, s_5) e (s_4, s_5) non domineranno nessun elemento

Per qualsiasi dimensione n e per una qualunque matrice $I^{(2)}$ dominata dall'indice, vi sarà:

- Una coppia (s_1, s_2) corrispondente all'elemento minimo di $I^{(2)}$, la cui intersezione $\cap \{s_1, s_2\}$ dominerà tutti gli insiemi di intersezione risultanti dall'aggregazione con qualsiasi dei restanti $n - 3$ valori di s : s_3, s_4, \dots, s_{n-1} .
 - L'intersezione $\cap \{s_1, s_2\}$ è un 2- intersezione di rango $n - 3$
- Le coppie (s_1, s_3) e (s_2, s_3) le cui intersezioni $\cap \{s_1, s_3\}$ e $\cap \{s_2, s_3\}$ domineranno minimamente tutti gli insiemi di intersezione da aggregazioni con uno qualsiasi dei $n - 4$ valori s : s_4, \dots, s_{n-1} .
 - L'intersezione $\cap \{s_1, s_3\}$ e $\cap \{s_2, s_3\}$ sono un 2- intersezione di rango $n - 4$
- Le coppie (s_1, s_4) , (s_2, s_4) e (s_3, s_4) le cui intersezioni $\cap \{s_1, s_4\}$, $\cap \{s_2, s_4\}$ e $\cap \{s_3, s_4\}$ domineranno minimamente tutti gli insiemi di intersezione da aggregazioni con uno qualsiasi dei $n - 5$ valori s : s_5, \dots, s_{n-1} .
 - L'intersezione $\cap \{s_1, s_4\}$, $\cap \{s_2, s_4\}$ e $\cap \{s_3, s_4\}$ sono un 2- intersezione di rango $n - 5$
- Le coppie (s_1, s_5) , (s_2, s_5) , (s_3, s_5) e (s_4, s_5) le cui intersezioni $\cap \{s_1, s_5\}$, $\cap \{s_2, s_5\}$, $\cap \{s_3, s_5\}$ e $\cap \{s_4, s_5\}$ non domineranno nessun insieme
 - L'intersezione $\cap \{s_1, s_5\}$, $\cap \{s_2, s_5\}$, $\cap \{s_3, s_5\}$ e $\cap \{s_4, s_5\}$ sono un 2- intersezione di rango $n - 6$. Essendo l'ultimo elemento il rango sarà $= 0$.

Il rango, pertanto, è un elemento che va da $n - 3$ a 0 : ($0 \leq rank \cap \{s_{ki}, s_{kj}\} \leq n - 3$). Di tutte le $\binom{n-1}{2}$ 2-intersezioni: una avrà rango $n - 3$, due avranno rango $n - 4$, tre avranno rango $n - 5$ e così via, con l'ultimo $n - 2$ di rango pari a 0 .

Riprendendo la matrice $I^{(2)}$ generata da $n = 6$:

$$\left\{ \begin{array}{l} \cap \{s_1, s_2\} \\ \cap \{s_1, s_3\} \\ \cap \{s_2, s_3\} \end{array} \right\} \rightarrow rank (n - 3) = 3$$

$$\left\{ \begin{array}{l} \cap \{s_1, s_4\} \\ \cap \{s_2, s_4\} \\ \cap \{s_3, s_4\} \end{array} \right\} \rightarrow rank (n - 4) = 2$$

$$\left\{ \begin{array}{l} \cap \{s_1, s_5\} \\ \cap \{s_2, s_5\} \\ \cap \{s_3, s_5\} \\ \cap \{s_4, s_5\} \end{array} \right\} \rightarrow rank (n - 5) = 1$$

$$\left\{ \begin{array}{l} \cap \{s_1, s_5\} \\ \cap \{s_2, s_5\} \\ \cap \{s_3, s_5\} \\ \cap \{s_4, s_5\} \end{array} \right\} \rightarrow rank (n - 6) = 0$$

Per questa particolare configurazione, calcolare il coefficiente di un 2-intersezione di rango r nella sommatoria, è riscrivibile come:

$$\frac{1}{3}\binom{r}{0} - \frac{1}{4}\binom{r}{1} + \frac{1}{5}\binom{r}{2} - \dots = \sum_{i=0}^r \frac{(-1)^i}{i+3} \binom{r}{i} = \frac{2}{(r+1)(r+2)(r+3)}$$

Nella matrice $I^{(2)}$, il rango di una qualsiasi delle sue voci in posizione (s_i, s_j) è facile da calcolare, in quanto il numero di indici l (con $l \neq i, j$) sia tale, che entrambe le seguenti condizioni tengano:

$$I^{(2)}(s_i, s_j) < I^{(2)}(s_i, s_l) \quad \text{oppure} \quad I^{(2)}(s_i, s_j) = I^{(2)}(s_i, s_l) \quad \text{con } j < l$$

$$I^{(2)}(s_i, s_j) < I^{(2)}(s_l, s_j) \quad \text{oppure} \quad I^{(2)}(s_i, s_j) = I^{(2)}(s_l, s_j) \quad \text{con } i < l$$

Sebbene sia improbabile avere un ordine dominato dall'indice $I^{(2)}$ come quello proposto nell'esempio, si scopre che averne una versione simile, sotto qualche permutazione delle voci della matrice, è molto probabile possa verificarsi nella pratica. Inoltre, risulta che scenari che non corrispondono esattamente a questo pattern sono tuttavia ancora molto vicini ad esso, in termini di distribuzione dei ranghi delle voci della matrice $I^{(2)}$. Ciò suggerisce che se siamo disposti ad accettare un altro errore minore nel file approssimazione, potremmo adattare tutto in un quadro uniforme, migliorando la funzione $\text{rank}()$, definita per ogni voce di $I^{(2)}$, come segue.

Per ogni dato $I^{(2)}$ permutiamo prima le sue righe e colonne con la permutazione che corrisponde all'ordine crescente di s_1, s_2, \dots, s_{n-1} . Ciò rende irrilevante l'ordine di indicizzazione originale. Quindi ricalcoliamo i ranghi di tutte le sue voci in base alle condizioni che soddisfano il conteggio e ordiniamo l'elenco di queste voci in ordine decrescente del calcolo dei ranghi e in ordine crescente per i valori di ingresso all'interno dello stesso valore del rango. Infine, si riassegni un rango pari a $n - 3$ al primo elemento in questo elenco ordinato, e un rango pari a $n - 4$ ai due elementi successivi nell'elenco, rango pari a $n - 5$ ai successivi tre elementi e così via. Quindi, al termine di questa procedura, ad ogni elemento in $I^{(2)}$ viene assegnato un rango, $0 \leq \text{rank} \cap \{s_i, s_j\} \leq n - 3$.

Questa procedura non modifica le classifiche calcolate se la matrice $I^{(2)}$ ha il modello di un ordine dominato dall'indice, o una permutazione di tale matrice cosa che accade abbastanza di frequente. In questo caso si forzano solo le poche istanze di $I^{(2)}$ che non sono conformi al modello dominato dall'indice per adattarsi allo stesso modello.

Unendo tutte queste considerazioni, è possibile definire la funzione Appro (2):

$$\phi_k^{(2)} = s_{kk} - \frac{1}{2} \sum_{i \neq k} \{s_{ki}\} + \sum_{i \neq j \neq k} \frac{2}{(r_{ij} + 1) + (r_{ij} + 2) + (r_{ij} + 3)} \cap \{s_{ki}, s_{kj}\}$$

Formula 3.13

L'approssimazione Appro (2) consiste in una approssimazione di “secondo ordine” e tiene conto non solo delle *distanze condivise* s_{ki} , ma anche delle loro 2-intersezioni formali $\cap \{s_{ki}, s_{kj}\}$. L'idea di questa approssimazione origina dalla stessa idea di Appro (1) ovvero ridurre i valori delle intersezioni sconosciute di ordine superiore a valori delle loro più piccole intersezioni note di ordine inferiore. Per Appro (2) tuttavia, piuttosto che implementare rigorosamente una sostituzione ottimale di 2 intersezioni, che coinvolge una complessità computazionale esponenziale, l'approssimazione risulta avere solo una complessità computazionale $O(n^3)$. Questa complessità è essenzialmente dovuta alla procedura di stima dei ranghi delle 2-intersezioni.

Vale la pena notare che forzando i ranghi nello schema univoco dato dall'ordine dominato dall'indice, la somma dei coefficienti di tutte le 2-intersezioni in Appro (2) è esattamente la stessa di quella che risulterebbe da una ricerca esponenziale completa. Ciò significa che Appro (2) mantiene esattamente lo stesso equilibrio complessivo dei coefficienti di 2 intersezioni della sostituzione di ricerca completa, ma potrebbe occasionalmente sostituire alcune intersezioni di ordine superiore con una 2-intersezione subottimale.

Esempio:

Per una maggior comprensione di Appro (2), viene ripreso il solito esempio con $n = 4$ nodi, matrice delle distanze. Da questo si calcolino, come è stato spiegato in precedenza, la matrice delle distanze condivise, come di seguito

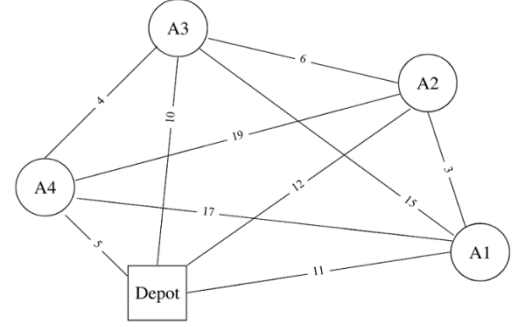


Figura 3.9 – Grafo completo

Distanze	Valori
d_01	11
d_02	12
d_03	10
d_04	5
d_12	3
d_13	15
d_14	17
d_23	6
d_24	19
d_34	4

Distanze condivise	$d_{0i} + d_{0j} - d_{ij}$	Valore
s_11	$2d_{01}$	22
s_22	$2d_{02}$	24
s_33	$2d_{03}$	20
s_44	$2d_{04}$	10
s_12	$d_{01} + d_{02} - d_{12}$	20
s_13	$d_{01} + d_{03} - d_{13}$	6
s_14	$d_{01} + d_{04} - d_{14}$	-1
s_23	$d_{02} + d_{03} - d_{23}$	16
s_24	$d_{02} + d_{04} - d_{24}$	-2
s_34	$d_{03} + d_{04} - d_{34}$	11

Tabelle 3.13 – Matrice delle distanze e distanze condivise

Nodo $k = 1$

$$\phi_1^{(2)} = s_{11} - \frac{1}{2}(s_{12} + s_{13} + s_{14}) + \left(\frac{2}{(r_{23} + 1)(r_{23} + 2)(r_{23} + 3)} \cap \{s_{12}, s_{13}\} + \frac{2}{(r_{24} + 1)(r_{24} + 2)(r_{24} + 3)} \cap \{s_{12}, s_{14}\} + \frac{2}{(r_{34} + 1)(r_{34} + 2)(r_{34} + 3)} \cap \{s_{13}, s_{14}\} \right)$$

Il numero di nodi è $n = 4$ per cui, volendo calcolare lo Shapley value per il cliente 1, il rango $0 \leq rank \cap \{s_{ki}, s_{kj}\} \leq n - 3$ sarà compreso tra 0 e 1. Per assegnare il rango all'intersezione si deve creare la matrice $I^{(2)}$, tenendo in considerazione che $\cap \{s_{ki}, s_{kj}\} = \min \{s_{ki}, s_{kj}, s_{ij}\}$

Per creare la matrice $I^{(2)}$, bisogna riassegnare i valori di s_l

- $s_{12} \rightsquigarrow s_1$
- $s_{13} \rightsquigarrow s_2$
- $s_{14} \rightsquigarrow s_3$

In seguito, inserire il valore corrispondente nell'intersezione degli s.

- $\cap \{s_1, s_2\} = \cap \{s_{12}, s_{13}\} = \min\{s_{12}, s_{13}, s_{23}\} = \min\{20, 6, 16\} = 6$
- $\cap \{s_1, s_3\} = \cap \{s_{12}, s_{14}\} = \min\{s_{12}, s_{14}, s_{24}\} = \min\{20, -1, -2\} = -2$
- $\cap \{s_2, s_3\} = \cap \{s_{13}, s_{14}\} = \min\{s_{13}, s_{14}, s_{34}\} = \min\{5, -1, 11\} = -1$

Costruiamo la matrice $I^{(2)}$:

	s_1	s_2	s_3
s_1	/	6	-2
s_2	6	/	-1
s_3	-2	-1	/

Riscritta per far combaciare le dominanze:

	s_1	s_3	s_2
s_1	/	-2	6
s_3	-2	/	-1
s_2	6	-1	/

Tabelle 3.14 – Matrice $I^{(2)}$ e poi riscritta

- $rank = n - 3 = 1 \rightarrow \cap \{s_1, s_3\} = \cap \{s_{12}, s_{14}\}$
- $rank = n - 3 = 0 \rightarrow \begin{cases} \cap \{s_1, s_2\} = \cap \{s_{12}, s_{13}\} \\ \cap \{s_2, s_3\} = \cap \{s_{13}, s_{14}\} \end{cases}$

$$\phi_1^{(2)} = s_{11} - \frac{1}{2}(s_{12} + s_{13} + s_{14}) + \frac{2}{(1+1)(1+2)(1+3)} \cap \{s_{12}, s_{14}\} + \frac{2}{(0+1)(0+2)(0+3)} \cap \{s_{12}, s_{13}\} + \frac{2}{(0+1)(0+2)(0+3)} \cap \{s_{13}, s_{14}\}$$

$$\phi_1^{(2)} = s_{11} - \frac{1}{2}(s_{12} + s_{13} + s_{14}) + \frac{1}{12} \cap \{s_{12}, s_{14}\} + \frac{1}{3} \cap \{s_{12}, s_{13}\} + \frac{1}{3} \cap \{s_{13}, s_{14}\}$$

$$\phi_1^{(2)} = 22 - \frac{1}{2}(25) + \frac{1}{12}(-2) + \frac{1}{3}(6) + \frac{1}{3}(-1) = 11.84$$

Capitolo 4 – Algoritmo di calcolo fairness per problemi di routing

Al fine di valutare il rapporto tra efficacia ed efficienza computazionale degli Appro per la ripartizione efficiente dei costi, è stato scelto di unirsi ad un programma già esistente, sviluppato dai dottorandi Diego Maria Pinto e Maro Boresta in appoggio il loro gruppo di ricerca presso il Consiglio Nazionale delle Ricerche – IASI. La scelta ha permesso di individuare la soluzione ottimale ad un problema di VRP nello smaltimento di rifiuti per ogni cliente. Come anticipato nel capitolo introduttivo, ogni cliente è composto da due nodi: un nodo Pickup e un nodo Delivery. OR2ML prende in considerazione tutti i percorsi possibili per servire n cliente con k veicoli e propone in output diverse matrici che descrivono il percorso ottimale come un problema di TSP. È qui che si interviene: l'obiettivo è prendere in input, l'output di questo algoritmo e generare un secondo output che assunte le soluzioni finali, possa estrapolare la percentuale di costo da allocare ad ogni singolo cliente.

Dato che, ogni algoritmo lavora sul singolo nodo e non sul singolo cliente, è necessario effettuare delle precisazioni:

1. Ogni cliente è composto da due nodi distinti, pertanto si sfrutta l'assioma 2.a (efficiency axiom) e si è stabilito di considerare separati tutti i nodi, calcolare la ripartizione dei costi e infine unire la ripartizione di Pickup e Delivery per ogni cliente.

Esempio:

Abbiamo 2 clienti, pertanto vi sono 4 nodi (2 Pickup e 2 Delivery), il percorso ottimale individuato da OR2ML è: $(0 - P_1 - P_2 - D_1 - D_2 - 0)$, con un costo totale di 29. Se si applica Shapley ad ogni singolo nodo il risultato sarebbe: $[P_1: 11, D_1: 8.17, P_2: 5, D_2: 4.83]$. Unendo i risultati il cliente 1 dovrà pagare $11 + 8.17 = 19.17$, mentre il cliente 2 dovrà pagare $5 + 4.85 = 9.83$. Il totale è proprio 29, il costo totale.

2. Un dubbio che potrebbe sorgere, consiste nella natura iniziale del problema, ovvero che il nodo P_i dovrà essere visitato per forza prima del nodo D_i . Nella ripartizione equa dei costi non viene considerato questo aspetto, dato il fatto che esiste già una soluzione al problema di VRP. Si potrebbe anche sostenere che non è necessario considerare questo dato dal momento che è già stato considerato a monte. Gli algoritmi lavorano a valle della soluzione e hanno lo scopo di estrapolare una ripartizione di costi e pertanto l'ordine non conta.

Per eseguire questi algoritmi in Python, è possibile utilizzare diverse librerie e funzioni: NumPy può essere utilizzato per gestire matrici e array di grandi dimensioni; Itertools può essere utilizzato per

generare tutte le possibili permutazioni necessarie. Il codice per questi algoritmi può essere scritto come funzioni che accettano un elenco di valori del giocatore come input e restituiscono lo Shapley e gli Appro come output. Una volta che questi algoritmi vengono eseguiti su una determinata soluzione del problema di VRP, le stime del valore di Shapley risultanti possono essere confrontate per determinare quale metodo fornisce i risultati più accurati. L'idea alla base consiste nel dare in input una matrice delle distanze di un problema di TSP di cui si conosca la soluzione ottimale e ottenere in output i valori corrispondenti.

Programma – VRP_cost_SOLVER

Questo progetto Python racchiude tutti i successivi codici, raccogliendo la matrice delle distanze e restituendo i valori dei tre algoritmi presi in esame.

```
1.  from Functions_files.VRP_cost_allocation_appro_SHAPLEY_01 import appro_1
2.  from Functions_files.VRP_cost_allocation_appro_SHAPLEY_02 import appro_2
3.  from Functions_files.VRP_cost_allocation_SHAPLEY import shapley
4.
5.  nodi = [1, 2, 3, 4]
6.  n = len(nodi)
7.
8.  matrice_distanze = {
9.      "d_11": 0,
10.     "d_22": 0,
11.     "d_33": 0,
12.     "d_44": 0,
13.     "d_01": 11,
14.     "d_02": 12,
15.     "d_03": 10,
16.     "d_04": 5,
17.     "d_12": 3,
18.     "d_13": 15,
19.     "d_14": 17,
20.     "d_23": 6,
21.     "d_24": 19,
22.     "d_34": 4,
23. }
24.
25. shapley = shapley(matrice_distanze=matrice_distanze, nodi=nodi, n=n)
26. appro1 = appro_1(matrice_distanze=matrice_distanze, nodi=nodi, n=n)
27. appro2 = appro_2(matrice_distanze=matrice_distanze, nodi=nodi, n=n)
```

Codice 4.1 – VRP cost solver

Vengono richiamate le funzioni shapley, appro1 e appro2 che verranno analizzate in seguito. Se viene inizializzato un dict contenente la matrice delle distanze, questo verrà preso e utilizzato come input per il calcolo dei tre algoritmi. La matrice delle distanze riportata è un esempio, nel prossimo capitolo vengono utilizzate tutte le combinazioni che il programma OR2ML genera, al fine di ottenere un consistente numero di dati da analizzare.

Di seguito uno schema di come lavora l'algoritmo (verranno analizzate le sezioni in seguito):

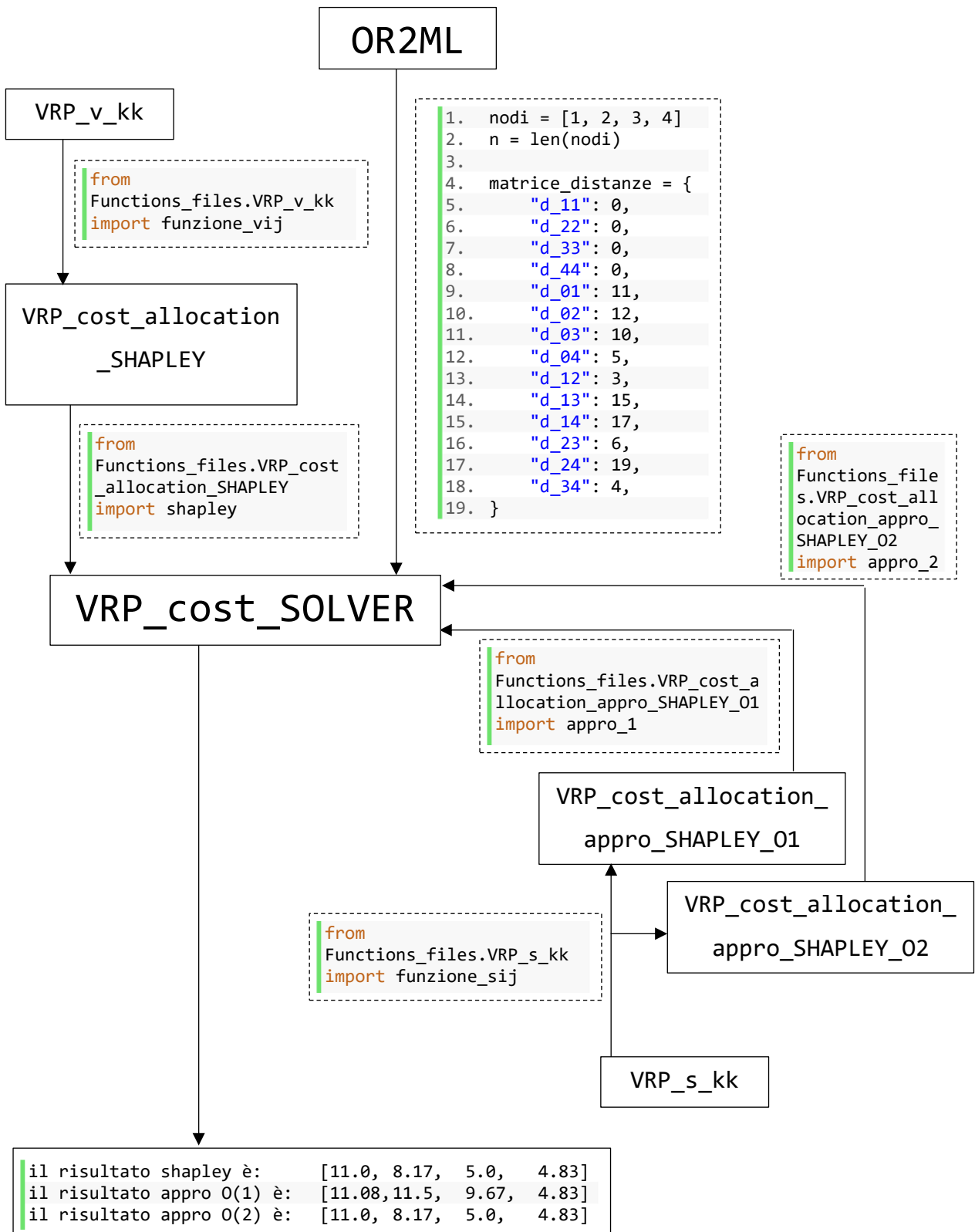


Immagine 4.12 – Schema generale del funzionamento dell'algoritmo di calcolo

Programma – VRP_v_kk

Questa porzione di codice viene richiamata nella funzione `VRP_cost_allocation_SHAPLEY`, per generare la matrice della *funzione di costo*. La principale problematica consiste nel fatto che per ogni singolo valore, è necessario risolvere un problema di TSP.

1) Vengono importati i moduli necessari al funzionamento del programma

```
1. import itertools
```

2) Viene definita la funzione che verrà richiamata nel calcolo dello Shapley value

```
2. def funzione_vij (distances, nodi):
```

3) Viene generato un ciclo for che genera una lista di tutte le combinazioni possibili

```
3.     funzione_costo = {}
4.
5.     all_combinations = []
6.     for length in range(len(nodi) + 1):
7.         for combination in itertools.combinations(nodi, length):
8.             all_combinations.append(list(combination))
```

4) Il ciclo for passa tutti i percorsi possibili che visitano ogni nodo esattamente una volta e calcola la loro distanza totale, controlla se il percorso corrente è più breve del percorso più breve corrente e restituisce il percorso più breve e la sua distanza totale

```
9.     for nodes in all_combinations:
10.
11.         def tsp(start_node, nodes, distances):
12.             all_paths = itertools.permutations(nodes)
13.             shortest_distance = float('inf')
14.             shortest_path = None
15.
16.             for path in all_paths:
17.                 current_distance = 0
18.                 source_node = start_node
19.                 for node in path:
20.                     current_distance += distances[source_node][node]
21.                     source_node = node
22.                 current_distance += distances[source_node][start_node]
23.                 if current_distance < shortest_distance:
24.                     shortest_distance = current_distance
25.                     shortest_path = (start_node,) + path + (start_node,)
26.
27.             return shortest_path, shortest_distance
28.
29.         shortest_path, shortest_distance = tsp(0, nodes, distances)
30.
31.         v = "{}_{}".format("v", nodes)
32.         funzione_costo.update({v: shortest_distance})
```

5) Ritorna la variabile funzione costo che verrà utilizzata da Shapley

```
33.     return funzione_costo
```

Programma – VRP_cost_allocation_SHAPLEY

Di seguito viene riportato il codice che iteri i dati per generare un output che rispetti la formulazione dello Shapley value:

$$\phi_i(v) = \frac{1}{n!} \sum_{\pi \in S_n} (v(\pi_i \cup \{i\}) - v(\pi_i))$$

- 1) Vengono importati i moduli necessari al funzionamento del programma. In più viene importata la funzione `_vij`, che genera tutte le *funzioni di costo* v

```
34.         import math
35.         from itertools import permutations
36.         from Functions_files.VRP_v_kk import funzione_vij
```


- 2) Viene definita la funzione Shapley che verrà richiamata nei fogli di calcolo, prendendo come parametri la matrice delle distanze, i nodi e il loro numero

```
37. def shapley (matrice_distanze, nodi, n):
```

- 3) Affinché la funzione costo funzioni, è necessario riscrivere la matrice delle distanze

```
1.  matrice_distanze = {
2.      "d_11": 0,
3.      "d_22": 0,
4.      "d_33": 0,
5.      "d_44": 0,
6.      "d_01": 11,
7.      "d_02": 12,
8.      "d_03": 10,
9.      "d_04": 5,
10.     "d_12": 3,
11.     "d_13": 15,
12.     "d_14": 17,
13.     "d_23": 6,
14.     "d_24": 19,
15.     "d_34": 4,
16. }
```

```
1.  distances = {
2.      0: {0: 0, 1: 11, 2: 12, 3: 10, 4: 5},
3.      1: {0: 11, 1: 0, 2: 3, 3: 15, 4: 17},
4.      2: {0: 12, 1: 3, 2: 0, 3: 6, 4: 19},
5.      3: {0: 10, 1: 15, 2: 6, 3: 0, 4: 4},
6.      4: {0: 5, 1: 17, 2: 19, 3: 4, 4: 0}
7.  }
```



```
38.     nn = n + 1 # number of nodes
39.     distances = {i: {} for i in range(nn)} # initialize the nested dictionary
40.
41.     for i in range(nn):
42.         for j in range(nn):
43.             if i == j:
44.                 distances[i][j] = 0
45.             else:
46.                 key = "d_{i}_{j}".format(min(i, j), max(i, j))
47.                 distances[i][j] = matrice_distanze[key]
```

- 4) Si richiama la funzione `_vij` per generare il dict la matrice della *funzione di costo* v_{ij} utilizzando però la matrice `distances`, con una serie di TSP

```
48.     funzione_costo = funzione_vij(distances=distances, nodi=nodi)
```


5) Si definiscono le variabili necessarie al completamento del codice

```
49.         numbers = list(range(1, n + 1))
50.         permutation_list = list(permutations(numbers))
51.         n_fattoriale = math.factorial(n)
52.         shapley_value = []
```

6) Il ciclo while itera per ogni nodo x in n. Vengono eseguite operazioni per ogni permutazione

```
53.         x = 1
54.
55.         while x <= n:
56.             i = 0
57.             somma = []
58.
59.             while i < n_fattoriale:
60.                 t = list(permutation_list[i])
61.
62.                 index_t = t.index(int(x))
63.
64.                 # -----
65.                 posizione1 = t[0:index_t + 1]
66.                 posizione1.sort()
67.
68.                 my_variable = 'v'
69.                 result1 = "{}_{}".format(my_variable, posizione1)
70.                 # -----
71.
72.                 # -----
73.                 posizione2 = t[0:index_t]
74.                 posizione2.sort()
75.
76.                 my_variable = 'v'
77.                 result2 = "{}_{}".format(my_variable, posizione2)
78.                 # -----
79.
80.                 z = funzione_costo[result1] - funzione_costo[result2]
81.                 somma.append(z)
82.
83.                 i += 1
84.
85.             s = sum(somma)
86.             shapley = s / n_fattoriale
87.             shapley_value.append(shapley)
88.
89.             x += 1
```

7) Nella variabile shapley_value vengono aggiunte ad ogni giro del primo ciclo while, i valori dello Shapley value risultanti, generando una lista [] con i risultati finali.

```
90.         print("il risultato shapley è:      ", shapley_value)
```

Programma – VRP_s_kk

Viene definita la funzione `s_kk` che verrà richiamata per il calcolo di Appro (1) e Appro (2), prendendo come parametri la matrice delle distanze, i nodi e il loro numero. L'obiettivo è quello di generare la *distanza condivisa*: $s_{ij} = d_{0i} + d_{0j} - d_{ij}$

1) Viene definita la funzione che verrà richiamata nei calcoli

```
1. def funzione_sij (matrice_distanze, nodi, n):
```

2) Si definiscono le variabili

```
2.     values = []
3.     x = 1
4.     y = 1
```

3) Il doppio ciclo while itera per ogni combinazione dei nodi n e restituisce s_{ij} cercando nella matrice delle distanze

```
5.         while x <= n:
6.
7.             while y <= n:
8.                 s_ij = matrice_distanze["d_0" + str(x)] + matrice_distanze["d_0" + str(y)] - m
9.                 atrice_distanze[
10.                     "d_" + str(x) + str(y)]
11.                 values.append(s_ij)
12.                 #print(s_ij)
13.                 y += 1
14.             y = x + 1
15.             x += 1
16.
17.     funzione_costo = {f"s_{i}{j}": values.pop(0) for i in nodi for j in nodi if i <= j}
```

4) Ritorna la variabile funzione costo che verrà utilizzata da Appro (1) e (2)

```
18.     return funzione_costo
```

Programma – VRP_cost_allocation_appro_SHAPLEY_O1

Di seguito viene riportato il codice che iteri i dati per generare un output che rispetti la formulazione di Appro O(1):

$$\phi_k^{(1)} = s_{kk} - Sc(s_{1,k}, s_{2,k}, \dots, s_{k-1,k}, s_{k+1,k}, \dots, s_{n,k})$$

1) Viene importata la funzione_sij, che genera tutte le *distane condivise* s

```
1. from Functions_files.VRP_s_kk import funzione_sij
```

2) Viene definita la funzione appro_1 che verrà richiamata nei fogli di calcolo, prendendo come parametri la matrice delle distanze, i nodi e il loro numero

```
2. def appro_1 (matrice_distanze, nodi, n):
```

3) Si richiama la funzione_sij per generare il dict la matrice delle *distance condivise* s_ij

```
3.         funzione_costo = funzione_sij(matrice_distanze=matrice_distanze, nodi=nodi, n=n)
```

4) Il primo ciclo while permette di estrarre il valore s_{kk}

```
4.     x = 1
5.     appro1 = []
6.
7.     while x <= n:
8.
9.         kk = 's'
10.        result = "{}_{}".format(kk, x, x)
11.        nodi.remove(x)
12.
13.        y = 0
14.        sc = []
```

5) Il secondo ciclo while permette di estrarre il valore $(s_{1,k}, s_{2,k}, \dots, s_{k-1,k}, s_{k+1,k}, \dots, s_{n,k})$

```
15.        while y <= int(n - 2):
16.            k1 = 's'
17.            order1 = [x, nodi[y]]
18.            order1.sort()
19.            order1 = ''.join(str(x) for x in order1)
20.            result1 = "{}_{}".format(k1, order1)
21.
22.            sc.append(result1)
23.
24.            y += 1
25.
26.        nodi.append(x)
```

6) Il terzo ciclo while permette di estrapolare i dati s e estrarre $Sc(x_1, x_2, \dots, x_k) = \sum_{i=1}^k \frac{y_i}{i(i+1)}$

```
27.         y = [funzione_costo[item] for item in sc]
28.         y.sort()
29.         y.reverse()
30.
31.         k = 1
32.         i = 1
33.         total = 0.0
34.
35.         while k <= int(n - 1):
36.             for g in y:
37.                 i = k * (k + 1)
38.                 r = g / i
39.                 total += r
40.                 k += 1
41.
42.         appro = funzione_costo[result] - total
43.
44.         appro1.append(appro)
45.         x += 1
```

7) Nella variabile appro1 vengono aggiunte ad ogni esecuzioni del primo ciclo while, i valori di Appro O(1) risultanti, generando una lista [] con i risultati finali.

```
46.         print("il risultato appro O(1) è: ", appro1)
```

Programma – VRP_cost_allocation_appro_SHAPLEY_O2

Di seguito viene riportato il codice che itera i dati per generare un output che rispetti la formulazione di Appro O(2):

$$\phi_k^{(2)} = s_{kk} - \frac{1}{2} \sum_{i \neq k} \{s_{ki}\} + \sum_{i \neq j \neq k} \frac{2}{(r_{ij} + 1) + (r_{ij} + 2) + (r_{ij} + 3)} \cap \{s_{ki}, s_{kj}\}$$

Per comodità il programma calcola l'output dei tre elementi della formula

- 1) Vengono importati i moduli necessari e la funzione_sij, che genera tutte le *distanze condivise* s

```
1. import itertools
2. from Functions_files.VRP_s_kk import funzione_sij
```

- 2) Viene definita la funzione appro_2 che verrà richiamata nei fogli di calcolo, prendendo come parametri la matrice delle distanze, i nodi e il loro numero

```
3. def appro_2 (matrice_distanze, nodi, n):
```

- 3) Si richiama la funzione_sij per generare il dict la matrice delle *distanze condivise* s_ij

```
4.     funzione_costo = funzione_sij(matrice_distanze=matrice_distanze, nodi=nodi, n=n)
```

- 4) Il ciclo for calcola il primo termine s_kk per ogni k = n – 1 nodi escluso il deposito

```
5.     termine1 = []
6.
7.     for x in nodi:
8.         kk = 's'
9.         result = "{}_{}".format(kk, x, x)
10.
11.         risultato = funzione_costo[result]
12.         termine1.append(risultato)
```

- 5) Il Ciclo while calcola il secondo termine, ovvero la sommatoria per ogni nodo x S_xi cambiando tutti gli i nei restanti nodi

```
13.     x = 1
14.     termine2 = []
15.     y = []
16.
17.     while x <= n:
18.
19.         nodi.remove(x)
20.
21.         for ij in nodi:
22.             k = 's'
23.             order = [x, ij]
24.             order.sort()
25.             order = ''.join(str(x) for x in order)
26.             result1 = "{}_{}".format(k, order)
27.             y.append(funzione_costo[result1])
28.
```

```

29.         nodi.append(x)
30.         total = - sum(y) / 2
31.         termine2.append(total)
32.
33.         y.clear()
34.
35.         x += 1

```

- 6) Nel ciclo while si estrae la sommatoria con il dividendo pari al rango e l'incrocio della matrice $I^{(2)}$. Ci sono due cicli for, il primo crea la lista ordinata s_{ki} e, il secondo crea la lista non ordinata s_{kj} , al fine di raggruppare s_{ij} . Per comprendere meglio: se $x = 2$, s_{12} s_{25} sono in ordine e quindi per avere s_{15} devo lavorare con s_{21} e s_{25} per tirare fuori s_{15} con il secondo ciclo for. Il successivo ciclo while unisce i 2 cicli for in una lista e tira fuori il minimo per inserirlo nella matrice $I^{(2)}$. Alla fine, la variabile terzo è l'elemento in terza posizione s_{ij} nella terna s_{ki} , s_{kj} , s_{ij} . La variabile minimo consiste nei valori da inserire in matrice. Gli ultimi cicli for e while mettono in ordine la matrice $I^{(2)}$ e generano il divisore che corrisponde al rango r_{ij}

```

36.         x = 1
37.         termine3 = []
38.
39.         while x <= n:
40.
41.             nodi.remove(x)
42.
43.             y = []
44.
45.             for ij in nodi:
46.                 k = 's'
47.                 order = [x, ij]
48.                 order.sort()
49.                 order = ''.join(str(x) for x in order)
50.                 result2 = "{}_{}".format(k, order)
51.
52.                 y.append(result2)
53.
54.                 combinations_in_order = list(itertools.combinations(y, 2))
55.
56.
57.                 asso = []
58.
59.                 for ij in nodi:
60.                     k = 's'
61.                     order1 = [x, ij]
62.                     order1 = ''.join(str(x) for x in order1)
63.                     result3 = "{}_{}".format(k, order1)
64.
65.                     asso.append(result3)
66.
67.                     combinations_NOT_in_order = list(itertools.combinations(asso, 2))
68.
69.                     z = 0
70.                     t = []
71.                     matrice_I = []
72.
73.
74.                     while z < len(combinations_in_order):
75.                         s = list(combinations_NOT_in_order[z])

```

```

76.
77.         beta = 's_' + str(x)
78.
79.         new_str1 = s[0].replace(beta, "")
80.         primo1 = int(new_str1)
81.
82.         t.append(primo1)
83.
84.         new_str3 = s[1].replace(beta, "")
85.         primo2 = int(new_str3)
86.
87.         t.append(primo2)
88.
89.         t.sort()
90.         t.clear()
91.
92.         terzo = 's_' + str(new_str1) + str(new_str3)
93.
94.
95.         total = list(combinations_in_order[z])
96.
97.         total.append(terzo)
98.
99.         minimo = min(funzione_costo[total[0]], funzione_costo[total[1]], funzione_costo[total[2]])
100.
101.         matrice_I.append(minimo)
102.         matrice_I.sort()
103.
104.         z += 1
105.
106.
107.
108.
109.
110.
111.         r_ij = []
112.
113.         for i in range(n - 2):
114.             r_ij.append(i)
115.
116.
117.         matrice_I.sort()
118.
119.         divisore = []
120.
121.         for num in r_ij:
122.             divisore.append(2 / ((num + 1) * (num + 2) * (num + 3)))
123.
124.         divisore.sort()
125.
126.         l_matrice = len(matrice_I)
127.         l_divisore = len(divisore)
128.
129.         z = 0
130.
131.         result = []
132.
133.
134.
135.
136.
137.
138.
139.
140.         i = 1
141.

```

```

142.         while len(matrice_I) > 0:
143.             sublist = matrice_I[:i]
144.             result.append(sublist)
145.             matrice_I = matrice_I[i:]
146.             i += 1
147.
148.             risultato = [num * divisore[i] for i in range(len(divisore)) for num in result[i]]
149.
150.             total = sum(risultato)
151.
152.             termine3.append(total)
153.
154.             nodi.append(x)
155.             nodi.sort()
156.
157.             x += 1

```

8) Nella variabile `appro2` vengono aggiunte ad ogni giro del primo ciclo `while`, i valori del primo, secondo e terzo termine, generando una lista `[]` con i risultati finali.

```

158.     appro2 = [sum(x) for x in zip(termine1, termine2, termine3)]
159.
160.     print('il risultato appro 0(2) è: ', appro2)

```


Capitolo 5 – Confronto tra Shapley value e metodi “average”

Per la validazione degli algoritmi proposti, rispetto alla media delle tre regole introdotte nel capitolo 2, sono stati utilizzati i dati raccolti durante tutto l’anno 2020. Questi dati si compongono di una raccolta delle varie domande di prelievo e consegna che ogni generico cliente effettua in qualsiasi giorno dell’anno. La descrizione del caso di studio è riportata nell’omonimo paragrafo dell’introduzione.

L’output della porzione di codice di Python, che estrae la soluzione del problema di VRP e la matrice delle distanze tra i vari nodi da servire, sono stati utilizzati come input per il calcolo simultaneo delle varie sotto-ripartizioni della ripartizione di costo.

Due sono gli aspetti rilevanti:

1. Vengono estratti sia i Km percorsi, sia le ore impiegate per servire tutti i clienti. Pertanto, al fine di poter paragonare due set di dati distinti, il programma itera tutte le operazioni due volte, fornendo una soluzione sia per i Km che per le ore.
2. La matrice delle distanze tra i nodi non è simmetrica: lo spazio o il tempo per andare dal nodo i al nodo j non sono gli stessi, a causa di sensi unici o deviazioni lungo il percorso. Il problema consiste nel fatto che Shapley e le sue approssimazioni lavorano con l’ipotesi di matrici simmetriche. Per il calcolo dei metodi invece, è stata utilizzata una matrice delle distanze simmetrica, dove la distanza da i a j e viceversa risulta la media tra la distanza da i a j e da j a i . Questa soluzione in realtà non influisce sul risultato finale, dato che, come concluso nel capitolo 2, i calcoli per ogni nodo sono effettuati considerando due volte la distanza tra i nodi (andata e ritorno). Pertanto:

$$2d_{ij} = d_{ij} + d_{ji} = 2 \left(\frac{d_{ij} + d_{ji}}{2} \right)$$

Anche se questo aspetto non modifica la soluzione, si è ritenuto corretto riportarlo, nel caso questo algoritmo venga riutilizzato in futuro in altri lavori simili.

Sono state estratte 193 date distinte, a partire dal 2020/01/07 fino ad arrivare al 2020/11/26, tenendo conto della settimana lavorativa composta da 5 giorni e delle giornate di ferie. In ognuna delle giornate lavorative vi sono più clienti che richiedono il servizio di Pickup e Delivery, per un totale di 1228 clienti serviti. Per ogni cliente, all’interno del TSP soluzione del VRP, sono stati estratti:

- La media delle tre alternative esistenti (*Isolated Cost Allocation*, *Neighbors Savings* e *Normalized Marginal Allocation*), che da adesso chiameremo Average
- Il valore di Shapley
- Il valore di Appro (1)
- Il valore di Appro (2)

Esempio su una soluzione calcolata sui Km:

Qui di seguito la soluzione del 2020/10/12, nella cui giornata sono stati serviti 10 clienti distinti. Per risolvere questo problema di VRP il programma ha generato 3 sotto-problemi di TSP:

1. Servire clienti 2, 4, 7, 9 e 10 con una distanza totale minima pari a 34.9

<i>Client</i>	Pickup name	delivery name	Km tot	Average	Shapley	Appro O(1)	Appro O(2)
2	186	226	34.9	9.548037391	6.419359647	6.286776979	6.541774002
4	224	226	34.9	3.725854792	2.604161941	2.550375826	2.699351402
7	14	226	34.9	10.72684086	15.42012814	15.67363209	14.74744624
9	264	226	34.9	4.438296551	1.352914465	1.324971497	1.922865557
10	221	226	34.9	6.463070404	9.105535801	9.066343611	8.990662795

Tabella 5.1 – Confronto risultati primo TSP

2. Servire clienti 1, 3 e 8 con una distanza totale minima pari a 101.6

<i>Client</i>	Pickup name	delivery name	Km tot	Average	Shapley	Appro O(1)	Appro O(2)
1	11	226	101.61	27.15555836	27.94371601	27.55367324	27.94371601
3	225	226	101.61	13.09541963	2.472198154	2.437690824	2.472198154
8	2	226	101.61	61.354622	71.18968583	71.61423593	71.18968583

Tabella 5.2 – Confronto risultati secondo TSP

3. Servire clienti 5 e 6 con una distanza totale minima pari a 165.78

<i>Client</i>	Pickup name	delivery name	Km tot	Average	Shapley	Appro O(1)	Appro O(2)
5	249	226	165.78	82.128377	82.10825774	82.10825774	82.10825774
6	64	226	165.78	83.652423	83.67254226	83.67254226	83.67254226

Tabella 5.3 – Confronto risultati terzo TSP

Si noti come la somma su ogni colonna dei 4 metodi proposti sia pari al totale dei Km percorsi. Infatti, i dati riportati su queste tabelle corrispondono alla porzione del totale, ripartita secondo le regole citate.

Analisi dei risultati

Come ipotesi di base, lo Shapley value è considerato il parametro di riferimento per valutare le ulteriori alternative, in quanto è noto come sia ritenuto “lo stato dell’arte” nell’associare un peso ad un determinato giocatore (nel nostro caso un cliente). Il risultato di Shapley, di conseguenza, viene considerato come il valore ideale nel valutare quanto i risultati degli altri algoritmi si discostano da quello di partenza. A tale fine si considerino:

- Mean absolute error: $MAE = \frac{\sum_{i=1}^n |y_i - x_i|}{n}$
- Percentuale di scostamento dal valore medio
- Mean square error: $MSE = \frac{\sum_{i=1}^n (y_i - x_i)^2}{n}$

Come anticipato, questi valori vengono confrontati con Shapley sia per i Km percorsi, sia per le ore impiegate dal truck k.

Km. Il valore medio di distanza percorsa per servire ogni singolo cliente è di 118.4 km e i valori di confronto risultano essere:

Average			Appro O(1)			Appro O(2)		
MAE	% scostamento	MSE	MAE	% scostamento	MSE	MAE	% scostamento	MSE
22.65	19.1%	1360.56	0.33	0.278%	0.74	0.47	0.399%	1

Tabella 5.4 – Confronto risultati delle tre alternative

Ore. Il valore medio di tempo impiegato per servire ogni singolo cliente è di 2.01 h e i valori di confronto risultano essere:

Average			Appro O(1)			Appro O(2)		
MAE	% scostamento	MSE	MAE	% scostamento	MSE	MAE	% scostamento	MSE
0.35	17.5%	0.31	0.0079	0.394%	0.0003	0.012	0.567%	0.0008

Tabella 5.5 – Confronto risultati delle tre alternative

Come è possibile constatare dalle tabelle sopra riportate, vi è uno scostamento considerevole rispetto alla regola Average, mentre per i valori di Appro (1) e (2), lo scarto è estremamente ridotto. Utilizzare il valore di Shapley rispetto alle approssimazioni proposte nel capitolo 2 è praticamente equivalente. Nella prossima sezione vengono considerati e approfonditi i tempi di calcolo.

Riprendendo l'esempio precedente del 2020/10/12, si riportano i tre grafici rispettivamente per le tre soluzioni di TSP, al fine di evidenziare lo scostamento rispetto all'Average.

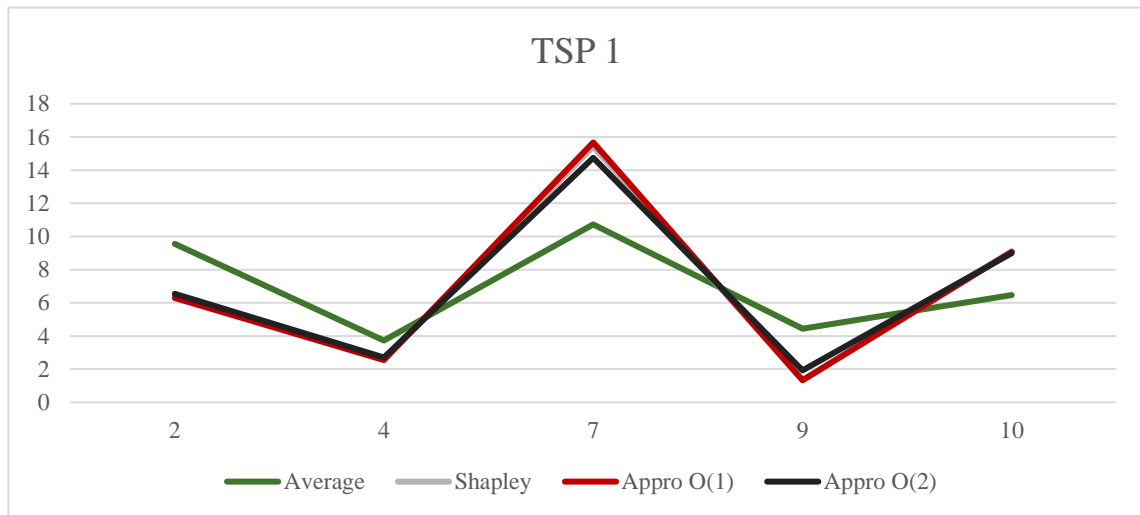
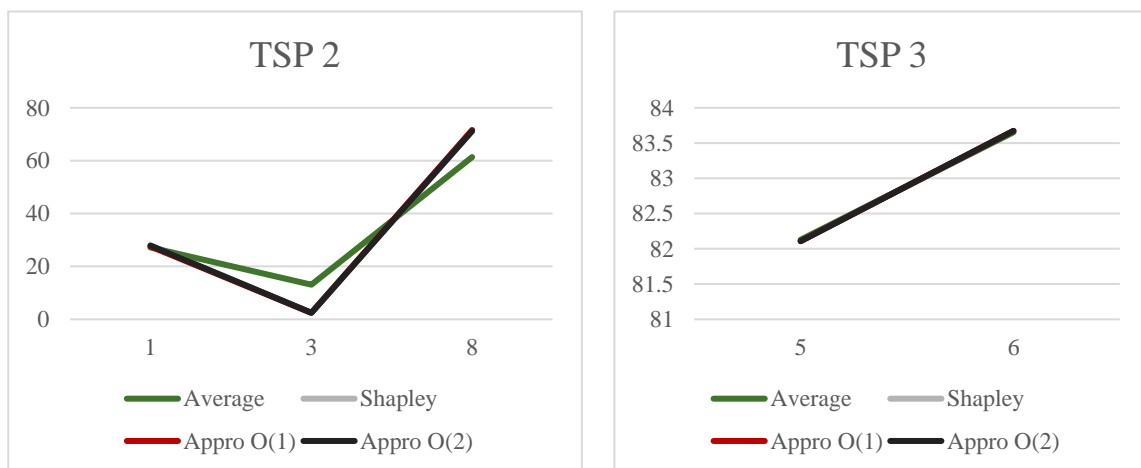


Grafico 5.1 – Soluzioni del primo TSP



Grafici 5.2 – Soluzioni del secondo e terzo TSP

Appro O(1) vs Appro O(2)

Nella letteratura riportata, l'approssimazione di secondo livello dovrebbe in teoria fornire una soluzione che meno si discosti dal valore di partenza di Shapley. Tramite i risultati ottenuti invece, anche se lo scostamento è minimo, è Appro (1) che genera soluzione più “precise”. Dall'analisi dei dati, l'ipotesi più probabile per spiegare questo fenomeno, è relativa al fatto che quasi tutti i clienti, composti dai 2 nodi Pickup e Delivery, condividono lo stesso nodo D. Alla base di Appro (1) infatti c'è l'ipotesi di considerare tutti i nodi, come se condividessero lo stesso percorso (Airport Problem), e dato che nella maggior parte delle soluzioni del TSP l'unico nodo D è posto alla fine del tragitto del truck, l'approssimazione proposta da Appro (1) risulta migliore rispetto ad Appro (2).

Limiti Shapley e plot dei tempi

L'ultimo aspetto rilevante da considerare risiede nella complessità computazionale. Come già esposto, lo Shapley value lavora in tempo esponenziale $O(n!)$, pertanto il suo utilizzo è limitato dal numero di nodi da riportare in soluzione. Lanciando il programma per calcolare la ripartizione dei costi, è stato inserito un timer per calcolare il tempo in secondi impiegato da Python per completare il run.

```
1. start = time.time()
2.
3. [...]
4.
5. end = time.time()
6. elapsed_time = end - start
7. print(elapsed_time)
```

Codice 5.1 – Stringa conteggio del tempo di calcolo

Ne è emerso che per il calcolo di tutte e quattro le alternative, il tempo medio di esecuzione è pari a 73.62 secondi. Mentre, se al codice si dovesse rimuovere solo il calcolo dello Shapley value puro, il tempo scenderebbe a 1.77 secondi. In pratica con un errore di calcolo mai superiore all'1%, il tempo di calcolo si riduce del 97.6%. Ecco il vero punto di forza delle approssimazioni proposte nel capitolo 2: per calcolare il valore esatto della ripartizione dei costi, è possibile utilizzare solo uno dei due Appro, senza ricorrere all'originale.

Un ulteriore aspetto da considerare consiste nel fatto che, durante l'esecuzione del programma, il codice lavora in tempo accettabile in termini computazionali *up to 10 clienti*. Per quelle poche istanze che chiedono di soddisfare la domanda di più di 10 clienti in una singola giornata, il tempo di risoluzione scala l'ordine di grandezza in modo quasi esponenziale, passando dai secondi alle decine di ore.

Si riportano i tempi medi per numero di clienti serviti a singola giornata:

# clienti / giornata	Tempo medio di completamento
1	5.44 sec
2	5.45 sec
3	5.48 sec
4	7.37 sec
5	10.81 sec
6	11.8 sec
7	15.97 sec
8	29.61 sec
9	38.26 sec
10	94.23 sec

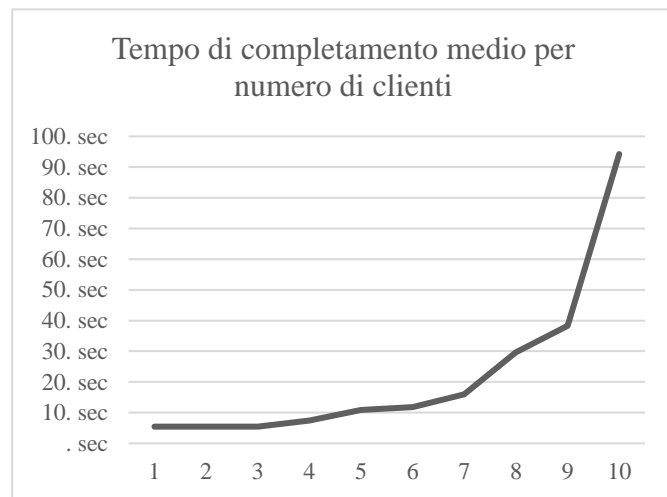


Tabella 5.6 – Tempi di calcolo per clienti serviti

Grafico 5.3 – Tempi di calcolo per clienti serviti

Nota sul numero di operazioni a cliente

Si consideri che, nel momento in cui il programma lavora su una data fissata, che contiene 10 clienti da servire, non è detto che Shapley lavori su tutti i 20 nodi (10 Pickup e 10 Delivery). Spieghiamo meglio questo aspetto.

Nell'ultimo esempio riportato, i clienti erano 10, ma la soluzione del VRP ha generato 3 sottoinsiemi che risolvono rispettivamente 3 problemi di TSP. La complessità è proporzionale al numero di clienti presenti in questi sottoinsiemi. Shapley quindi non lavorerà su 20 nodi, ma lavorerà indipendentemente su 10 nodi (5 clienti) 6 nodi (3 clienti) e 4 nodi (2 clienti), permettendo di abbassare notevolmente il numero di operazioni.

È per questo motivo che i tempi di calcolo tra due date possono risultare molto differenti, pure condividendo lo stesso numero di clienti da servire.

Allocazione costi e strategie di prezzo

Le strategie di prezzo si riferiscono alle metodologie e ai processi che le aziende utilizzano per determinare e stabilire il miglior prezzo per i loro prodotti o servizi. Vengono presi in considerazione vari fattori come le condizioni di mercato, i segmenti, la capacità di credito, le azioni della concorrenza, i margini commerciali e i costi di input. Le aziende devono individuare prezzi che massimizzino la redditività e raggiungano gli obiettivi desiderati soddisfacendo il cliente. Esistono vari tipi di strategie di prezzo, come i prezzi di costo maggiorato, i prezzi competitivi, i prezzi basati sul valore, i prezzi di penetrazione, i prezzi di scrematura e altri ancora, che le aziende possono utilizzare per fissare loro tariffe. Nel caso di studio in esame, una volta identificata la soluzione volta

a minimizzare il costo totale per servire un dato cliente i, è necessario comprendere come questa possa essere utilizzata per generare un tariffario. Le approssimazioni di Shapley possono risultare utili in questo contesto. Restando al nostro esempio, la Innocenti S.r.L. grazie a Shapley, avrà a disposizione un metodo che lavori in tempo polinomiale e che sia equo per calcolare la tariffa da proporre ad un nuovo cliente.

È stato evidenziato come lo Shapley value e le sue due approssimazioni possano essere considerate equivalenti in un problema di VRPPD; pertanto, per la composizione di un tariffario si farà riferimento esclusivamente ad un unico valore di Shapley (mantenendo tutte le assunzioni fatte sulla riduzione del tempo di calcolo) rispetto al metodo Average.

Nel campo della logistica dei trasporti per il Waste management, è possibile separare i costi fissi da quelli variabili. Il confronto tra l'utilizzo di Shapley value e il metodo Average per la ripartizione del costo di trasporto si riferisce alla seconda categoria di costi. Si consideri la figura [5.1] e si noti come il metodo Average fornisca una differenza più marcata su clienti appartenenti alla stessa area geografica



Figura 5.1 – Mappa clienti serviti nel primo TSP nella tabella 5.1

Prendendo invece in esame istanze che servano due clienti molto lontani tra loro nella stessa soluzione di TSP, come per esempio i due clienti serviti il 2020/03/03, l'uno a Bologna e l'altro a Roma, i chilometri complessivi percorsi sono 759 Km. Shapley li ripartisce 749 a carico del cliente bolognese e 10 del cliente romano, mentre il metodo Average li divide rispettivamente in 746 e 13. Si evince pertanto che la differenza di calcolo dei due metodi per la generazione di un tariffario è maggiore se

i punti di ritiro sono relativamente vicini l'uno all'altro. Di contro al crescere della distanza, si riduce il gap tra i due metodi di calcolo che finiscono quasi per coincidere.

Per creare un tariffario che sfrutti appieno le potenzialità di uno dei due metodi, un buon punto di inizio potrebbe essere il dividere i clienti serviti in aree geografiche di appartenenza.

Sempre tornando al nostro esempio, consideriamo come la posizione della Innocenti S.r.L. sia centrale rispetto alle potenziali rotte di ritiro in Italia. Dall'analisi storica dei dati dell'azienda di evince che suoi servizi vengono resi disponibili a clienti situati su tutto il territorio nazionale e che il 22% (21 su 95) di essi è concentrato nell'area urbana e sub urbana di Roma e su di loro è canalizzato il 61% (749 su 1228) di tutti i prelievi effettuati dalla Innocenti S.r.L.. Il restante 39% dei ritiri viene pertanto eseguito sul rimanente 78% dei clienti che si trovano lontani da Roma. Una larga parte dei clienti più remoti viene di fatto servita solo una volta durante tutto l'anno.

Abbiamo visto come l'applicazione dello Shapley value possa risultare più puntuale nel determinare i costi per clienti che operano nella stessa area geografica, mentre si appiattisce sulla Average value per realtà situate in aree diverse. Pertanto, può risultare utile ricorrere allo Shapley value nel determinare i tariffari di pick up all'interno di zone geografiche predeterminate.

In sintesi, per la Innocenti S.r.L. utilizzare lo Shapley value vorrebbe dire riuscire ad individuare con maggior precisione i costi variabili della propria flotta di truck in base alle distanze percorse. Si otterrebbe così una definizione di prezzo più puntuale rispetto agli effettivi costi sostenuti area per area.

È importante evidenziare come tuttavia ciò non rappresenti di per sé una soluzione ideale ma solo uno strumento per avere maggiore consapevolezza nella propria politica di gestione aziendale e una migliore fotografia dei rischi potenziali. La costruzione del prezzo finale al cliente deve infatti tenere anche conto di variabili meno ponderabili quali analisi della concorrenza, appetibilità di un nuovo cliente, settore di appartenenza del cliente ecc.

Se torniamo ancora al nostro esempio, una possibile soluzione alle diverse dinamiche analizzate, potrebbe essere quella di applicare una riduzione di prezzo a clienti di aree geografiche più lontane compensata da un incremento di prezzo diluito sul maggior numero di ritiri effettuati nell'area di Roma.

Conclusioni

La ricerca nella programmazione matematica e nella gestione del Vehicle routing problem in ambito del waste management incorpora studi che richiamano l'attenzione sulle diverse fonti di incertezza che le imprese devono imparare a gestire. Vi sono molti fattori da tenere in considerazione e focalizzarsi solo su un singolo aspetto piuttosto che avere una visione di insieme può risultare spesso fuorviante. Giunti alla fine di questo elaborato, è pertanto possibile effettuare due considerazioni principali:

- la prima è di natura prettamente tecnica e matematica
- la seconda è più generica e sposta il focus su una più ampia visione di insieme allargando il punto di osservazione.

Nell'ambito della prima considerazione il quadro proposto in questo documento indaga come sia possibile, in un contesto di Vehicle Routing Problem, utilizzare l'approccio proposto da Shapley, nelle sue forme approssimate invece che nella sua forma esatta. È stato dimostrato che lo Shapley value può essere implementato a scapito del tempo di calcolo su dei problemi che coinvolgono singoli clienti composti da 2 nodi distinti di Pickup e Delivery. Ma è stato anche evidenziato come il principale punto di debolezza di questo metodo possa essere aggirato tramite l'utilizzo di Appro (1) e Appro (2) proposti da Dan C. Popescu & Philip Kilby nel loro *Approximation of the Shapley value for the Euclidean*", con le opportune modifiche tali da adattarli al problema preso in esame.

Dal punto di vista puramente tecnico e matematico, per questa tipologia di criticità è superfluo impiegare un'enorme quantità di tempo per calcolare lo Shapley value, dato che è possibile ridurre il tempo computazionale del 97.6% accettando un errore nel calcolo dell'1% massimo. Questa applicazione non si limita alla tipologia di problemi presi in esame, ma rende possibile estrarre gli algoritmi di riferimento per applicarli a qualsiasi altro argomento di ottimizzazione lineare, ottenendo un risultato simile.

Questo lavoro di ricerca espone il quadro con un'allocazione dei costi idealmente equa, sfruttando nuove metodologie in grado di produrre approssimazioni accurate del valore di Shapley con un minore sforzo di calcolo. Ciò rende possibile concretizzare la strategia di allocazione dei costi dei valori di Shapley nell'applicazione del caso reale che coinvolge diversi clienti. In questo contesto potrebbe essere rilevante estendere i test sperimentali all'allocazione della capacità nella gestione dei problemi tipici di un ambiente di produzione.

Come seconda considerazione, si è cercato di costruire una visione di insieme derivante dai dati riportati. È necessario, infatti, conciliare la Fairness con il carattere commerciale, ovvero la tariffa che la società di servizio di raccolta e consegna decide di applicare al cliente servito. Se da un lato è vero che Shapley fornisce un approccio preciso per imputare i costi in funzione delle distanze e dei tempi, dall'altro questo costituisce anche il punto più critico. Non bisogna cadere nell'errore di considerare questo metodo applicabile ad ogni situazione e in grado di restituire un valore da seguire universalmente. Le alternative Appro sono da considerarsi come uno strumento, un mezzo per raggiungere uno scopo e come tali, il loro utilizzo può essere corretto o meno.

In effetti si può notare dai dati riportati, come la differenza di costo imputato tra i clienti utilizzando Shapley, oscilla tra range di valori molto ampi. Vi sono clienti in luoghi difficilmente raggiungibili e clienti in luoghi più accessibili, ma questo metodo non deve essere il solo parametro preso in considerazione per l'imputazione dei costi. Abbiamo compreso come nel caso di un'area non facilmente servibile, se utilizzassimo esclusivamente lo Shapley value, o le sue approssimazioni, la conseguenza sarebbe un'imputazione di porzione di costi maggiore senza assolutamente tenere in conto i vari aspetti sociali connessi che risultano essere non meno importanti.

Per essere competitivi in modo corretto su quella determinata quota di mercato diventa pertanto opportuno imputare una parte dei costi su clienti più facilmente raggiungibili e servibili, che secondo la teoria di Shapley avrebbero dovuto pagare di meno.

Questo aspetto viene evidenziato nei seguenti grafici, che riportano alcuni risultati su problemi di TSP. Si evince come il metodo Average generi uno smorzamento con l'effetto di caricare più costi sui clienti più vicini e alleggerire quelli sui clienti più lontani.

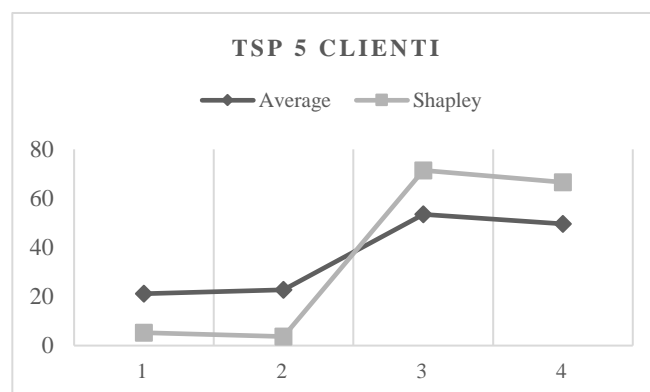
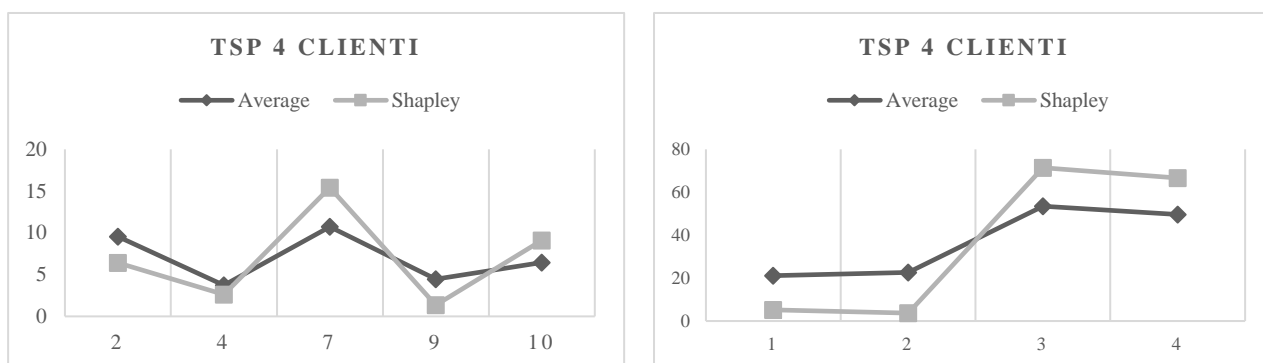


Grafico 6.1 – Confronto risultati TSP con 5 clienti



Grafic1 6.2 – Confronto risultati TSP con 4 clienti

In conclusione, a nostro avviso, è stato raggiunto lo scopo di fornire dei metodi che potessero essere alternativi allo Shapley value, riducendone la complessità computazionale. Non bisogna tuttavia cadere nell'errore di considerare quest'ultimo un metodo quasi infallibile. Deve ritenersi altresì un valido strumento cui fare ricorso in tutte quelle situazioni che ne richiedono l'apporto. In linea generale, occorre sempre considerare che qualunque caso di studio presenta aspetti diversificati a seconda dei dati trattati. È possibile utilizzare le soluzioni proposte in questo elaborato in altri ambiti, tenendo conto delle differenze dovute alle realtà in esame.

Bibliografia

- Progetto PIPER - Piattaforma intelligente per l'ottimizzazione di operazioni di riciclo [Grant No. A0375- 2020-36611, COPPA B85F21001480002].
- “*Customer cost forecasting through patterns learning of optimal capacity allocation*” by (Diego Maria Pinto, Marco Boresta, and Giuseppe Stecca)
- “*Waste Mismanagement in Developing Countries: A Review of Global Issues*” by Okonkwo et al. (2019)
- “*Sustainable Manufacturing and the Role of Environmental Stewardship*” published in the journal Sustainability in 2017.
- “*Open-source VRPLite Package for Vehicle Routing with Pickup and Delivery*” by A. Khademi and M.G.C. Resende
- “*Vehicle Routing Problem with Deliveries and Pickups: Modelling Issues and Heuristic Solutions*” by S. Ropke and D. Pisinger
- “*A Column Generation Algorithm for the Vehicle Routing Problem with Deliveries and Pickups*” by G. Desaulniers, J. Desrosiers, and M. Solomon
- “*A value for n-person games*” by Lloyd S. Shapley
- “*Approximation of the Shapley value for the Euclidean*” by Dan C. Popescu & Philip Kilby
- “*Efficient computation of cost allocation of VRPs*” by Dan C. Popescu & Philip Kilby
- <http://www.library.fa.ru/files/roth2.pdf>
- <https://mate.unipv.it/atorre/borromeo2010/shapley.pdf>
- https://web.mit.edu/16.070/www/lecture/big_o.pdf
- <https://gryhasselbalch.com/>
- <https://github.com/jvkersch/pyconcorde>
- Gentile, C., Pinto, D. M., & Stecca, G. (2022). Price of robustness optimization through demand forecasting with an application to waste management. Soft Computing, 1-12.
- Stecca, G., & Kaihara, T. (2021). Negotiation based approach for collecting and recycling operations in circular economy. Procedia CIRP, 104, 200-205.
- Pinto, D. M., & Stecca, G. (2021). Optimal planning of waste sorting operations through mixed integer linear programming. Graphs and Combinatorial Optimization: from Theory to Applications: CTW2020 Proceedings, 307-320.

Ringraziamenti

È la seconda volta che passo per tutto questo, uno pensa di esserci abituato, ma si sbaglia. Quest'ultimo anno accademico è stato il più strano della mia vita, costellato di tanti cambiamenti, sia belli che brutti. Ho fatto molti errori di cui mi pento e alcune cose di cui vado fiero. Questo mi ha portato a non essere più la stessa persona che ero prima, e con il mio comportamento ho allontanato da me molte persone care e di questo chiedo scusa. I miei ringraziamenti, pertanto, vanno a tutti coloro che nonostante tutto, mi sono rimasti vicino, perché so che potrò sempre contare su di loro.

Alla mia mamma, poco paziente e con un carattere difficile, proprio come me, ma con un grande cuore. Che non si ferma mai e antepone sempre me e Bea a sé stessa.

Al mio papà, il gigante gentile, che da lontano ha sempre cercato di vegliare su di me con occhio vigile, indirizzandomi per farmi trovare la mia strada.

A Bea, mia sorella, la mia migliore amica e confidente, cresciuti allo stesso modo e per questo al tempo stesso specchio e opposto l'uno dell'altro. Abbiamo condiviso segreti e paure, dolore e gioia, e so che saremo sempre pronti a sostenerci a vicenda, qualunque cosa accada, saremo sempre noi due contro il mondo, fino alla fine.

A nonno Cesare e Angelo, che non ci sono più, che sotto la loro divisa, hanno rivelato di essere più umani di quanto non dessero a vedere. A nonna Piera che c'è e ci sarà sempre e a nonna Silvana.

Ringrazio il mio relatore, il professor Giuseppe Stecca che mi ha dato l'opportunità di lavorare su un progetto molto interessante e il mio correlatore Diego Maria Pinto, che mi è stato accanto e mi ha supportato in tante cose, compreso comprendere in modo nuovo, quello che potrebbe attendermi nell'immediato.

Ai 2 boyz, fratelli di una vita, con le nostre vite imperfette, sempre pronti a sostenerci a vicenda, che quando uno di noi prova a dare in massimo in qualcosa, e il dubbio lo assale: "What if my best isn't good enough?" gli altri sono pronti a sostenerlo "It's good enough for me".

A Daniele, incontrati all'asilo, cresciuti insieme per poi essere spinti in direzioni diverse dalla vita. Al grande amico che ho ritrovato in questo strano periodo di passaggio. Alla nostra piccola grande avventura.

Ad Andrea, senza il quale, non avrei trovato la spinta per concludere questo lavoro. La nostra vena fanciullesca non morirà mai e saremo sempre pronti a tornare a ridere insieme.

A Lorenzo G, sempre presente per tutti e sempre pronto a dare una mano. Grazie di esserci stato anche per me, magari solo per un caffè, ma mi ha sempre aiutato.

A Luigi, forti di questo percorso universitario iniziato insieme, scusami ma abbandono un po' prima la nave, ma sono sicuro che mi tu sarai lì, non mollando mai e pronto a seguirmi subito.

A Fiore, che mi asseconda sempre nelle mie follie, a Tommy con la sua allegria contagiosa e ad Ale con il suo amore per il vino e la vita.

A tutta ALITUR, o meglio, a tutte le persone che ne fanno parte, che hanno lasciato un'impronta indelebile dentro di me. Come in tutte le cose ci sono stati momenti, belli e brutti, ma non cambierei nulla e rifarei tutto da capo, perché è anche grazie a loro che sono diventato la persona che sono oggi.

A QBR, che nonostante tutto, sono sempre pronti a brindare insieme.

A tutti coloro che ho dimenticato di citare, perdonatemi e agli amici, familiari e i compagni che ci sono stati, sono passati e che ora non ci sono più, grazie.