Sommario

[Introduzione – Fair Cost Allocation 2](#_Toc135465955)

[Capitolo 1 – Le 3 alternative esistenti 3](#_Toc135465956)

[Caso applicativo 4](#_Toc135465957)

[A) Stand Alone (SA) 5](#_Toc135465958)

[B) Neighbors Savings (NS) 7](#_Toc135465959)

[C) Extreme Neighbors Savings (ENS) 12](#_Toc135465960)

[Analisi della soluzione proposta 14](#_Toc135465961)

[Considerazioni su altri esempi 15](#_Toc135465962)

[Capitolo 2 – Lo Shapley Value 19](#_Toc135465963)

[Storia dello Shapley value 19](#_Toc135465964)

[La forma dello Shapley value 21](#_Toc135465965)

[Appendice sulla complessità 1 23](#_Toc135465966)

[A) ApproShapley 24](#_Toc135465967)

[Applicare ApproShapley ad un problema di TSP 27](#_Toc135465968)

[B) ApproShapley O(1) – prima approssimazione 30](#_Toc135465969)

[Airport problem – caso monodimensionale 30](#_Toc135465970)

[Appendice sulla complessità 2 34](#_Toc135465971)

[La forma di Appro O(1) nel caso multidimensionale 35](#_Toc135465972)

[C) ApproShapley O(2) – seconda approssimazione 40](#_Toc135465973)

[Capitolo 3 – Il programma Python 45](#_Toc135465974)

[Capitolo 4 – ApproShapley, O(1) e O(2) nel caso di studio 46](#_Toc135465975)

[Conclusioni 48](#_Toc135465976)

[Bibliografia 49](#_Toc135465977)

# Introduzione – Fair Cost Allocation

Testo

# Capitolo 1 – Le 3 alternative esistenti

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 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.

Questo lavoro di ricerca propone una strategia di allocazione dei costi al di fuori del paradigma dello Shapley value. Infatti è stato necessario trovare una strategia che si applicasse ad ogni singolo problema di TSP, la cui la totalità delle soluzioni viene generata dal singolo problema di VRP iniziale. Nel singolo TSP ogni cliente è composto da un punto di ritiro Pi e da uno di consegna Di. Inoltre come fattore ulteriore da considerare la quantità di clienti da servire essendo elevata rende necessaria una semplificazione.

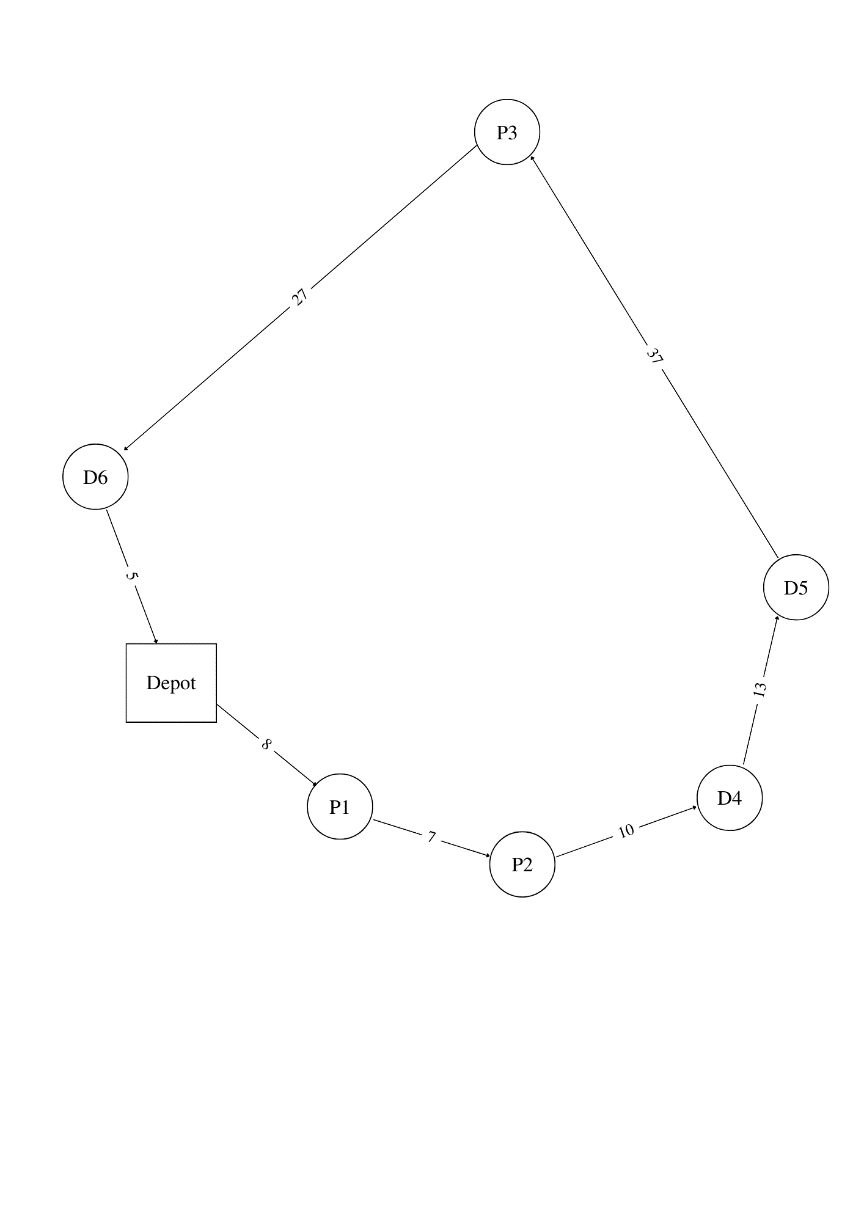
Per l’allocazione dei costi sono state individuate 3 regole distinte, ciascuna con aspetti positivi e negativi, in grado di compensarsi tra di loro. È stata individuata come soluzione finale una media ponderata dei risultati:

* Regola *Stand Alone*
* Regola *Neighbors Savings*
* Regola *Extreme Neighbors Savings*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Dep (o/d)** | **P1** | **P2** | **P3** | **D4** | **D5** | **D6** |
| **Dep**  **(o/d)** | / | 8 | 13 | 31 | 19 | 28 | 5 |
| **P1** | 8 | / | 7 | 22 | 14 | 15 | 11 |
| **P2** | 13 | 7 | / | 25 | 10 | 27 | 14 |
| **P3** | 31 | 22 | 25 | / | 41 | 37 | 27 |
| **D4** | 19 | 14 | 10 | 41 | / | 13 | 28 |
| **D5** | 28 | 15 | 27 | 37 | 13 | / | 19 |
| **D6** | 5 | 11 | 14 | 27 | 28 | 19 | / |

## 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 Pi (i = 1, 2, 3), nodi delivery Di (i = 4, 5, 6) e una matrice delle distanze che esprime il costo come somma di distanza e durata tra i nodi.

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, con un costo totale per servire tutti i clienti i, pari a:

* : uguale a 1 se il veicolo k attraversa arco (i, j) e 0 altrimenti.
* : costo percorrenza arco (i, j)

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:

* n = numero di clienti da servire
* P = {1,...,n} : set di nodi pickup
* D = {n+1 ,..., 2n} : set di nodi delivery
* o = deposito come partenza
* d = deposito come arrivo
* Ci,n+i = costo dell’arco tra il nodo Pi e il nodo Dn+i

## Stand Alone (SA)

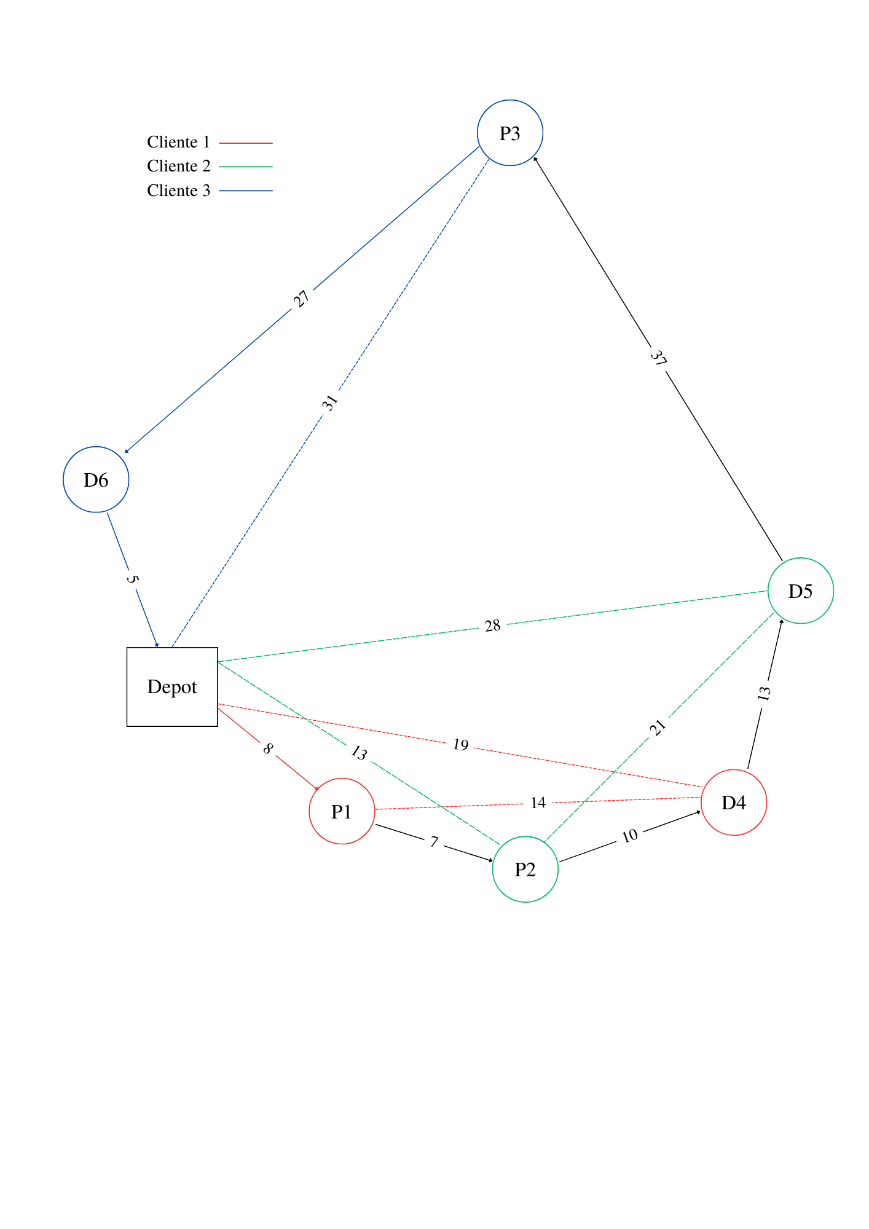
Immagine che contiene diagramma

Descrizione generata automaticamenteCome 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 *Stand Alone* assegna i costi agli n clienti in proporzione al percorso che origina dal deposito verso il nodo di prelievo Pi, consegna al nodo Di, quindi rientra al deposito. Si è optato per uno scenario non cooperativo, nel quale ogni singolo tour venisse considerato a sé stante, 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.

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

Per ogni cliente i, la porzione di costo del costo totale, secondo la regola *Stand alone* risulta:

* **Esempio:**

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

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

Mentre il costo imputabile a ciascun cliente risulta essere

|  |  |  |
| --- | --- | --- |
| **Cliente 1** | **Cliente 2** | **Cliente 3** |
|  |  |  |
|  |  |  |
|  |  |  |

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, spostando 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, che è la maggiore tra le tre, ma è vero anche che questo si trovi a metà strada tra P1 e D4. Non viene percepito il beneficio derivante dal fatto di essere vicini ad altri clienti da servire.

Come si evince, la regola *Stand Alone* ha il grande vantaggio della semplicità di utilizzo e della bassa complessità computazionale, ma questo rappresenta anche il suo principale svantaggio, data la troppa semplicità nel non considerare la riduzione dei costi che si potrebbe ottenere per i clienti vicini tra loro. Questo aspetto viene considerato nelle altre due regole proposte in seguito.

## Neighbors Savings (NS)

Per compensare la mancanza della regola *Stand Alone*, che non considera 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 Pi e Di del tour considerato, e aggiunti tutti gli archi che permettono al tour 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 Pi e Di.

* : rappresenta la distanza totale percorsa da un veicolo k che visita il cliente i nel suo tour.
* : rappresenta la somma di tutte le distanze degli archi percorsi per visitare sia Pi che Di, 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 Pi o Di come nodo in partenza o in arrivo.

Vengono presi, per ogni cliente i, , per calcolare per ogni cliente i. Si faccia riferimento alla variabile booleana del generico arco

Nota: consente di risolvere il caso in cui si abbia il nodo pickup e delivery dello stesso cliente, uno di seguito all’altro in soluzione.

* : 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 Pi e Di 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:

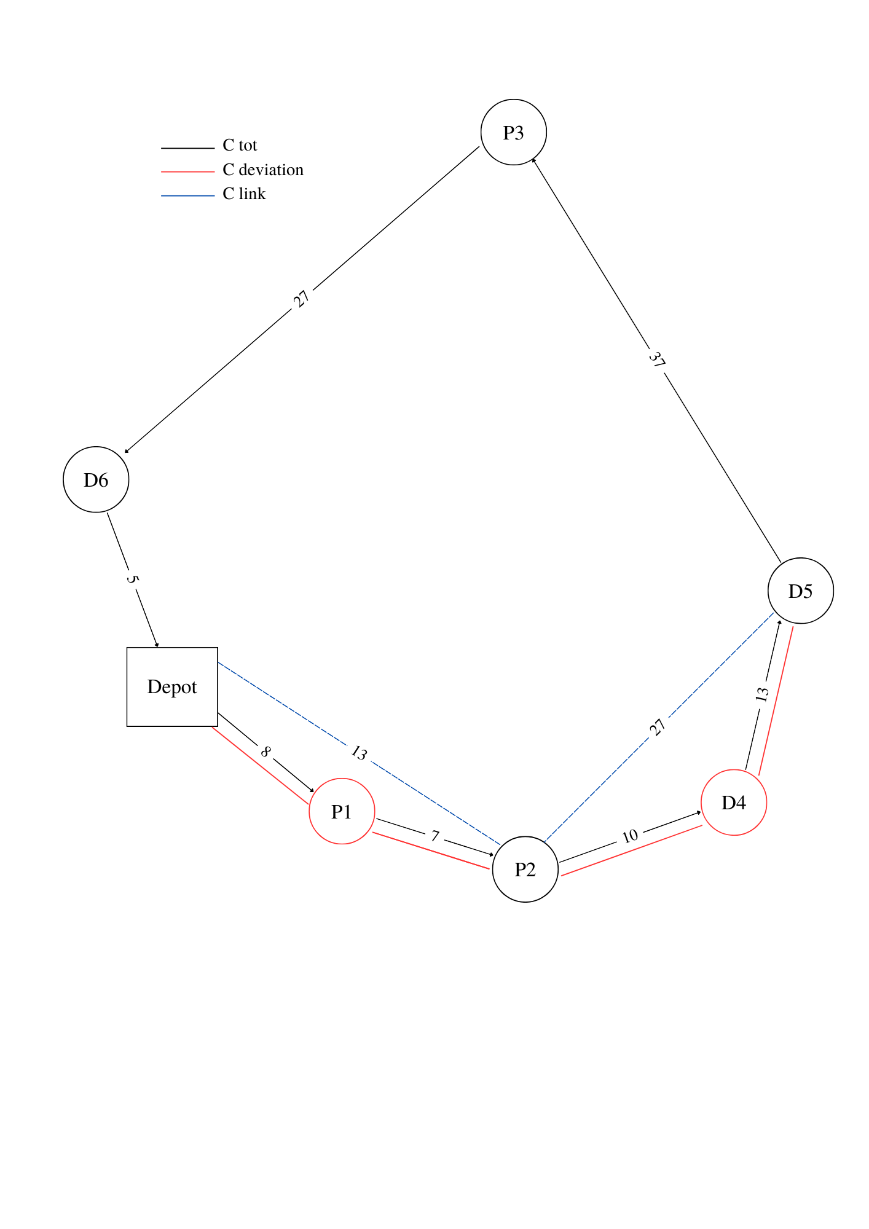
Nota: e consentono di risolvere il caso in cui si abbia il nodo pickup e delivery dello stesso cliente, uno di seguito all’altro in soluzione.

* : rappresenta il contributo della distanza marginale di i, essendo .

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 *Stand Alone* e sono applicate le formule descritte in precedenza per ognuno dei 3 clienti.

**Cliente 1**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Dep (o/d)** | **P1** | **P2** | **P3** | **D4** | **D5** | **D6** |
| **Dep**  **(o/d)** | / | 8 | 13 | 31 | 19 | 28 | 5 |
| **P1** | 8 | / | 7 | 22 | 14 | 15 | 11 |
| **P2** | 13 | 7 | / | 25 | 10 | 27 | 14 |
| **P3** | 31 | 22 | 25 | / | 41 | 37 | 27 |
| **D4** | 19 | 14 | 10 | 41 | / | 13 | 28 |
| **D5** | 28 | 15 | 27 | 37 | 13 | / | 19 |
| **D6** | 5 | 11 | 14 | 27 | 28 | 19 | / |

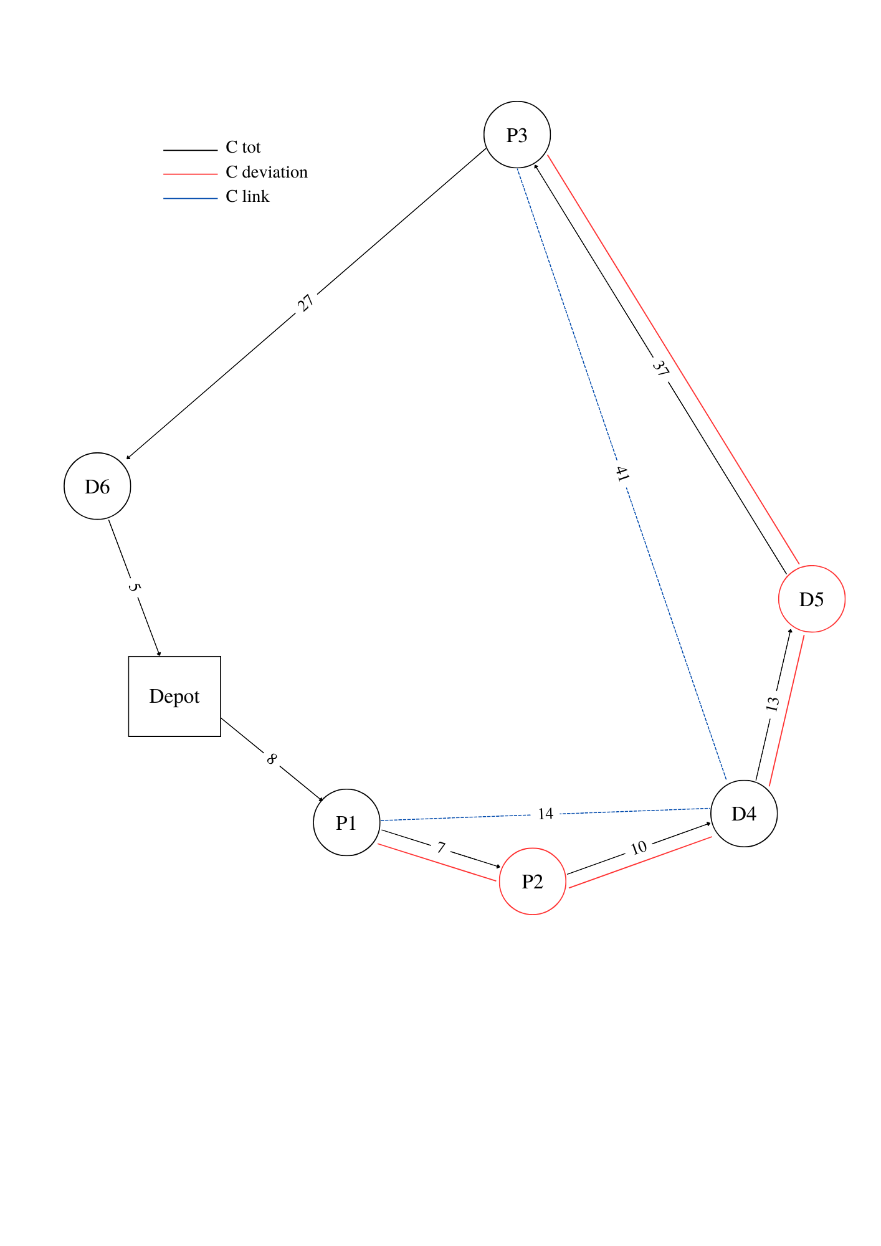
Prendiamo il cliente i = 1 ( e

*Costo totale (uguale per tutti)*

*Costo deviation*

*Costo link*

*Costo marginale*

**Cliente 2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Dep (o/d)** | **P1** | **P2** | **P3** | **D4** | **D5** | **D6** |
| **Dep**  **(o/d)** | / | 8 | 13 | 31 | 19 | 28 | 5 |
| **P1** | 8 | / | 7 | 22 | 14 | 15 | 11 |
| **P2** | 13 | 7 | / | 25 | 10 | 27 | 14 |
| **P3** | 31 | 22 | 25 | / | 41 | 37 | 27 |
| **D4** | 19 | 14 | 10 | 41 | / | 13 | 28 |
| **D5** | 28 | 15 | 27 | 37 | 13 | / | 19 |
| **D6** | 5 | 11 | 14 | 27 | 28 | 19 | / |

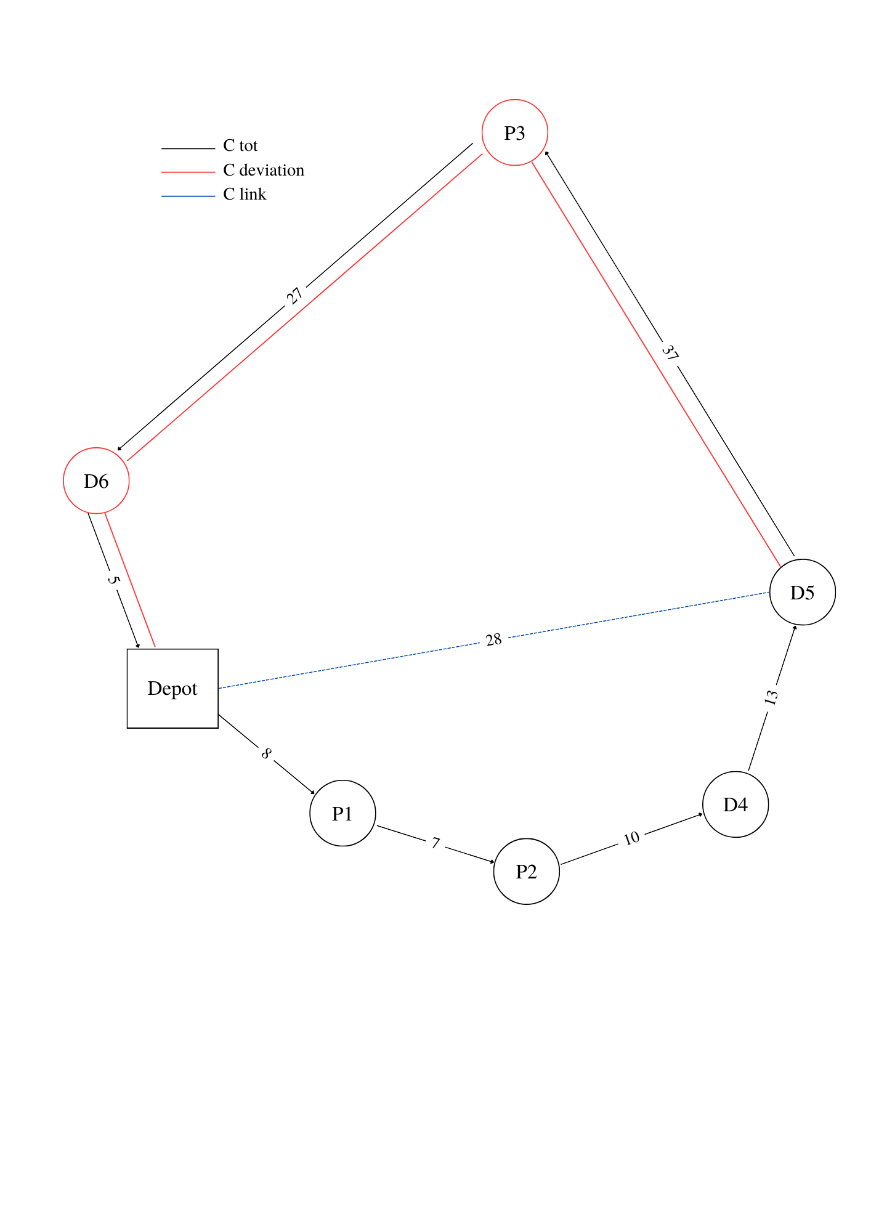
Prendiamo il cliente i = 2 ( e

*Costo totale (uguale per tutti)*

*Costo deviation*

*Costo link*

*Costo marginale*

**Cliente 3**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Dep (o/d)** | **P1** | **P2** | **P3** | **D4** | **D5** | **D6** |
| **Dep**  **(o/d)** | / | 8 | 13 | 31 | 19 | 28 | 5 |
| **P1** | 8 | / | 7 | 22 | 14 | 15 | 11 |
| **P2** | 13 | 7 | / | 25 | 10 | 27 | 14 |
| **P3** | 31 | 22 | 25 | / | 41 | 37 | 27 |
| **D4** | 19 | 14 | 10 | 41 | / | 13 | 28 |
| **D5** | 28 | 15 | 27 | 37 | 13 | / | 19 |
| **D6** | 5 | 11 | 14 | 27 | 28 | 19 | / |

Prendiamo il cliente i = 3 ( e

*Costo totale (uguale per tutti)*

*Costo deviation*

*Costo link*

*Costo marginale*

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.

per poi calcolare il vero e proprio

|  |  |  |
| --- | --- | --- |
| **Cliente 1** | **Cliente 2** | **Cliente 3** |
|  |  |  |
|  |  |  |

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. L’*Extreme Neighbors savings*.

## Extreme Neighbors Savings (ENS)

Come visto in precedenza, maggiore sarà il percorso che il veicolo k dovrà affrontare, maggiore sarà anche 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 , questo viene tolto dall’equazione. Potrebbe verificarsi il caso in cui la somma delle distanze degli archi percorsi per visitare sia Pi che Di 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:

* **Esempio:**

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

**Cliente 1**

**Cliente 2**

**Cliente 3**

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.

per poi calcolare il vero e proprio

|  |  |  |
| --- | --- | --- |
| **Cliente 1** | **Cliente 2** | **Cliente 3** |
|  |  |  |
|  |  |  |

Come è facile notare, l’aspetto negativo 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.

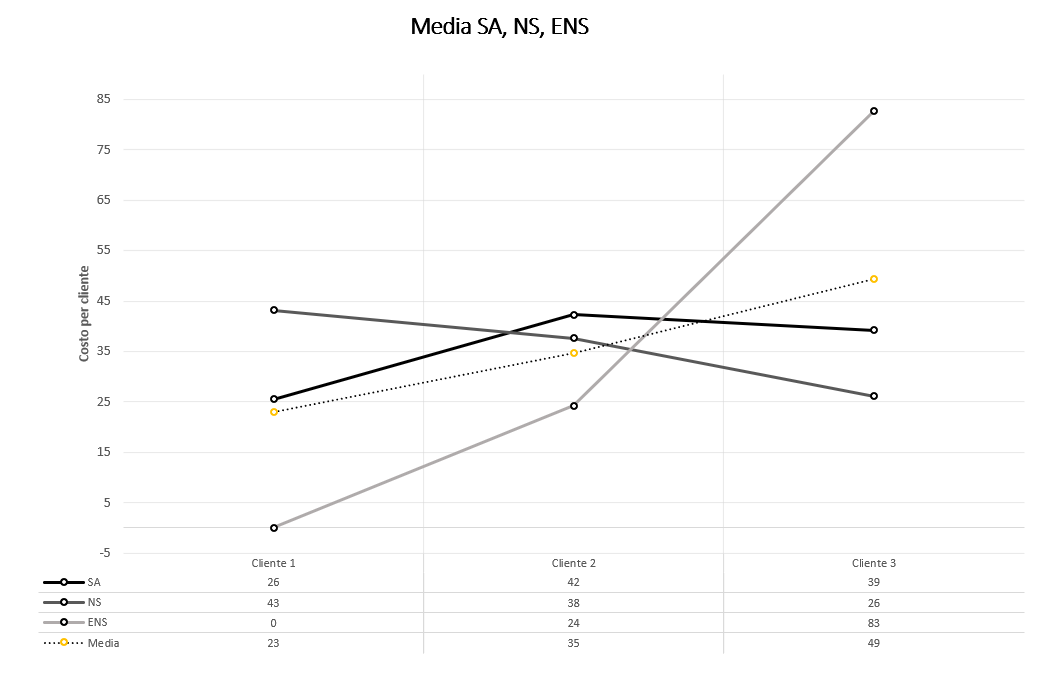
## Analisi della soluzione proposta

Ognuna delle soluzioni proposte ha vantaggi e svantaggi. L’obiettivo prefissato è stato quello 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.

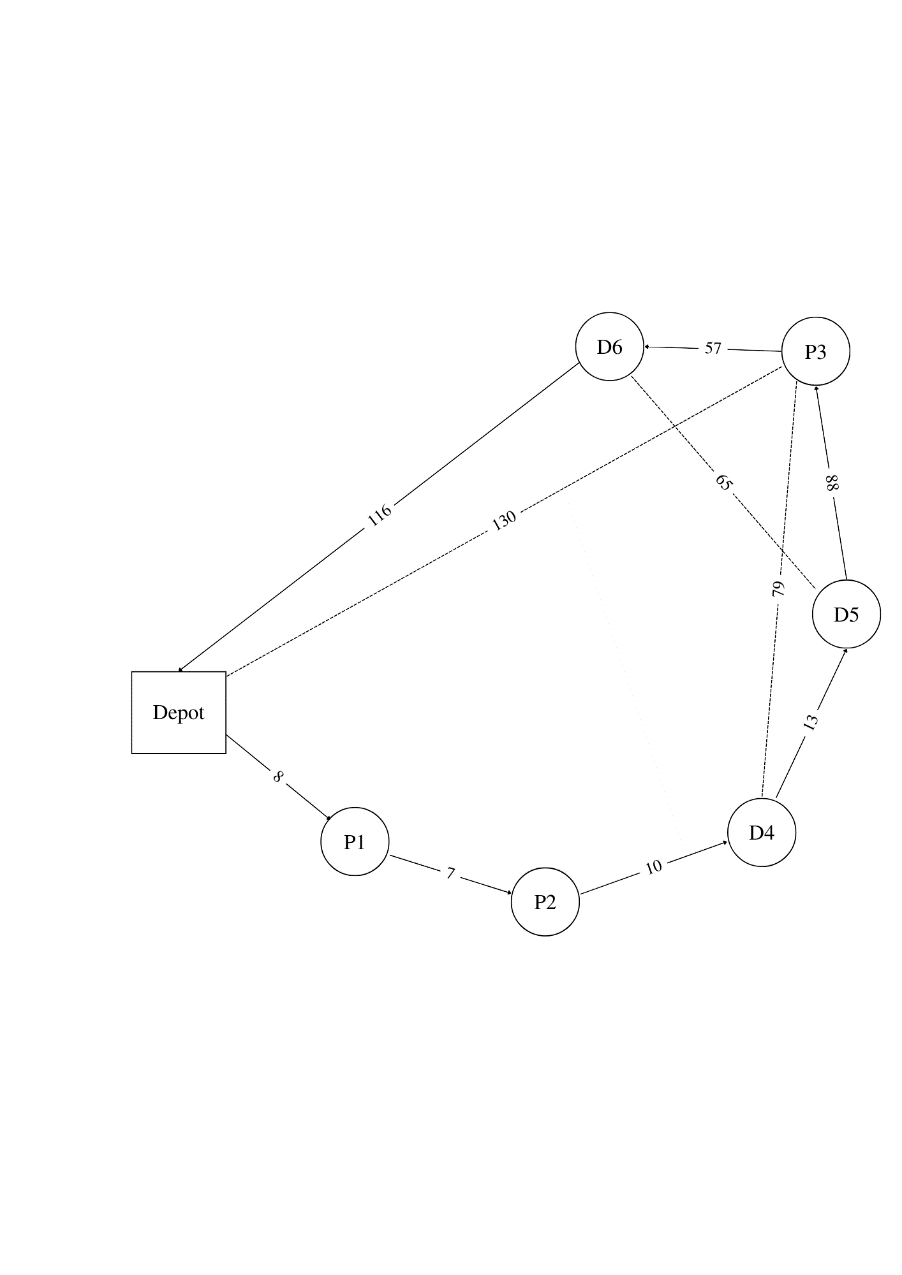
Riprendendo i tre diversi risultati dell’esempio e riassumendoli in una tabella è possibile trovare facilmente una media delle alternative proposte:

|  |  |  |  |
| --- | --- | --- | --- |
| **RULE** | **Cliente 1** | **Cliente 2** | **Cliente 3** |
| SA | 26 | 42 | 39 |
| NS | 43 | 38 | 26 |
| ENS | 0 | 24 | 83 |
| **Media** | **23** | **35** | **49** |

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

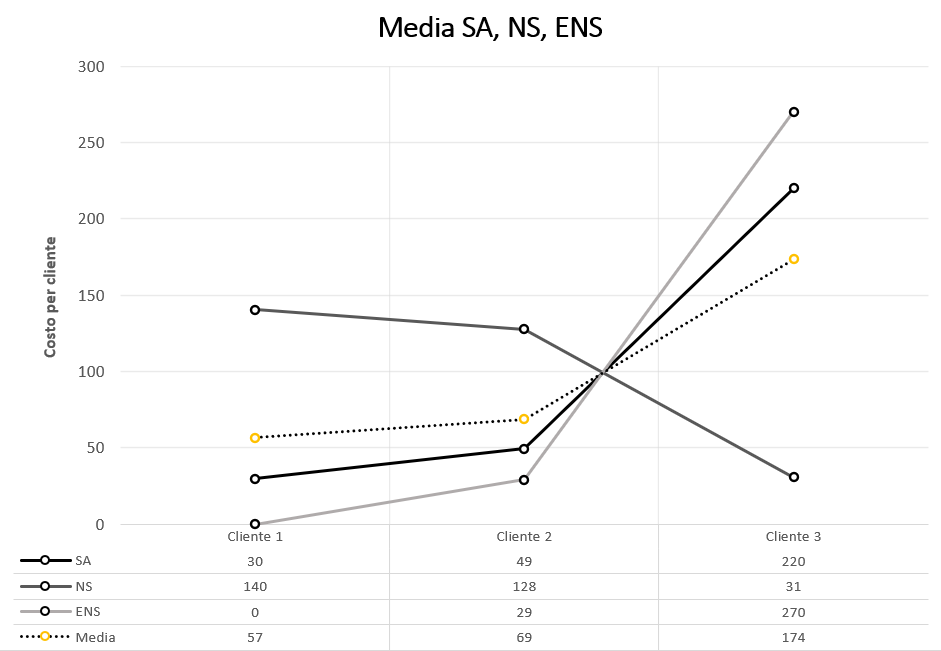


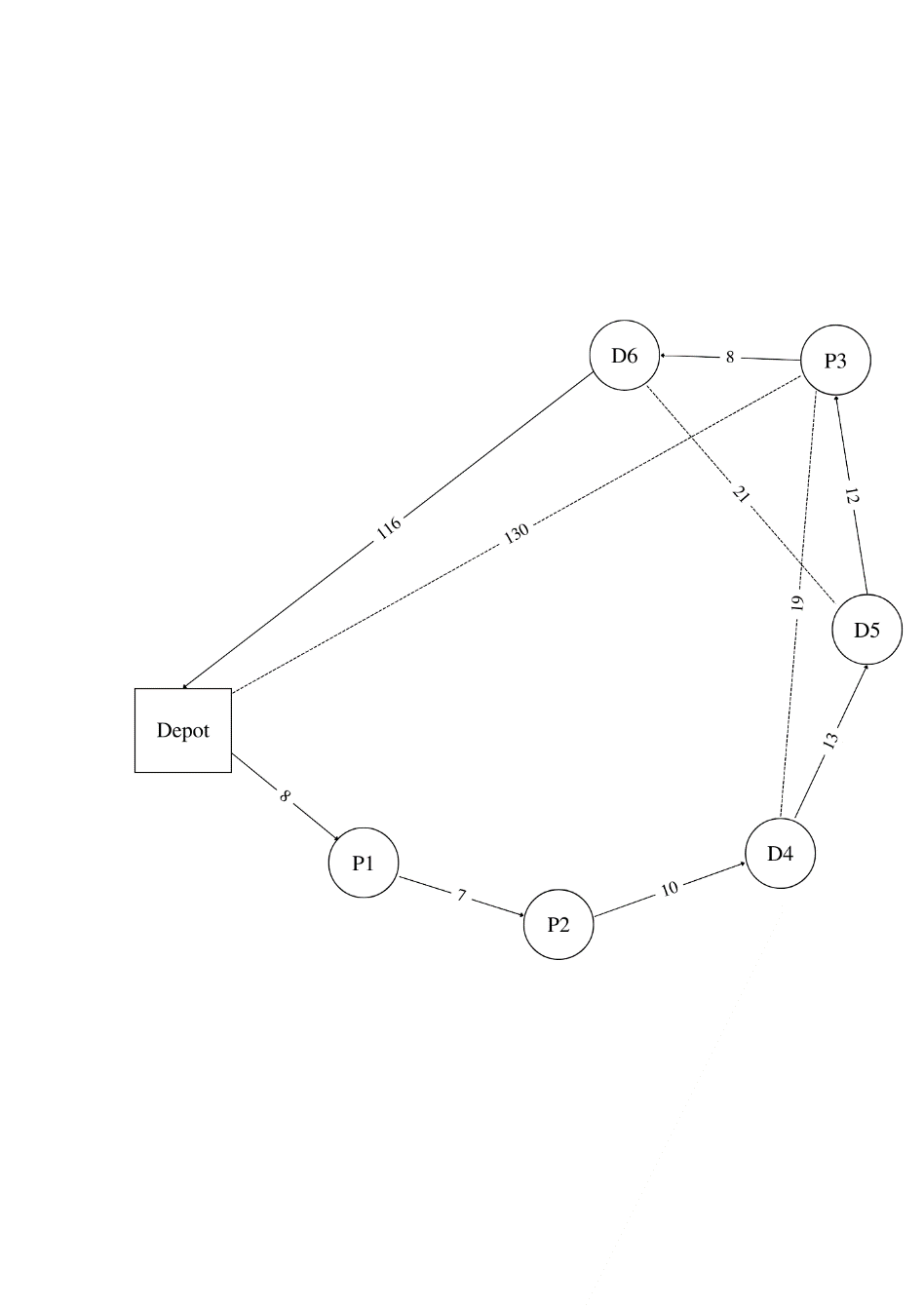
### Considerazioni su altri esempi

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Dep (o/d)** | **P1** | **P2** | **P3** | **D4** | **D5** | **D6** |
| **Dep**  **(o/d)** | / | 8 | 13 | 130 | 19 | 28 | 116 |
| **P1** | 8 | / | 7 | 22 | 14 | 15 | 11 |
| **P2** | 13 | 7 | / | 25 | 10 | 27 | 14 |
| **P3** | 31 | 22 | 25 | / | 79 | 88 | 57 |
| **D4** | 19 | 14 | 10 | 41 | / | 13 | 28 |
| **D5** | 28 | 15 | 27 | 37 | 13 | / | 65 |
| **D6** | 5 | 11 | 14 | 27 | 28 | 19 | / |

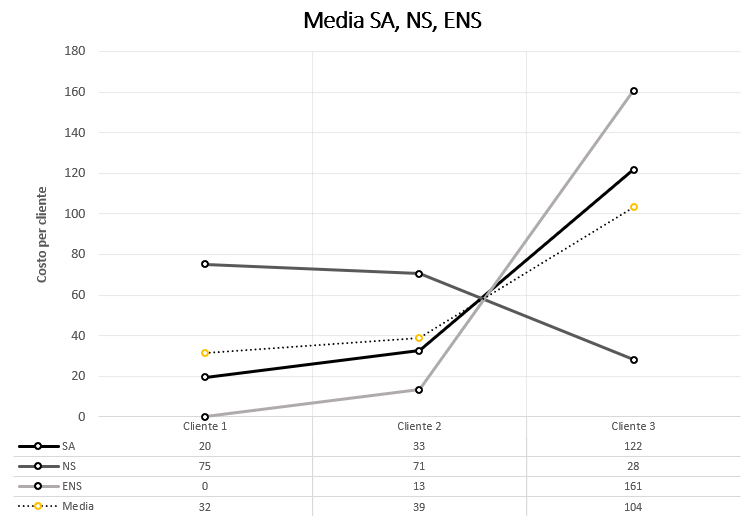
|  |  |  |  |
| --- | --- | --- | --- |
| **RULE** | **Cliente 1** | **Cliente 2** | **Cliente 3** |
| % SA | 10% | 16% | 74% |
| % NS | 47% | 43% | 10% |
| % ENS | 0% | 10% | 90% |
| **% Media** | **19%** | **23%** | **58%** |



Partendo dallo stesso esempio di partenza, si consideri il caso in cui, il cliente 3 è sempre posto a distanza maggiore dal deposito, ma è anche più vicino agli altri clienti.

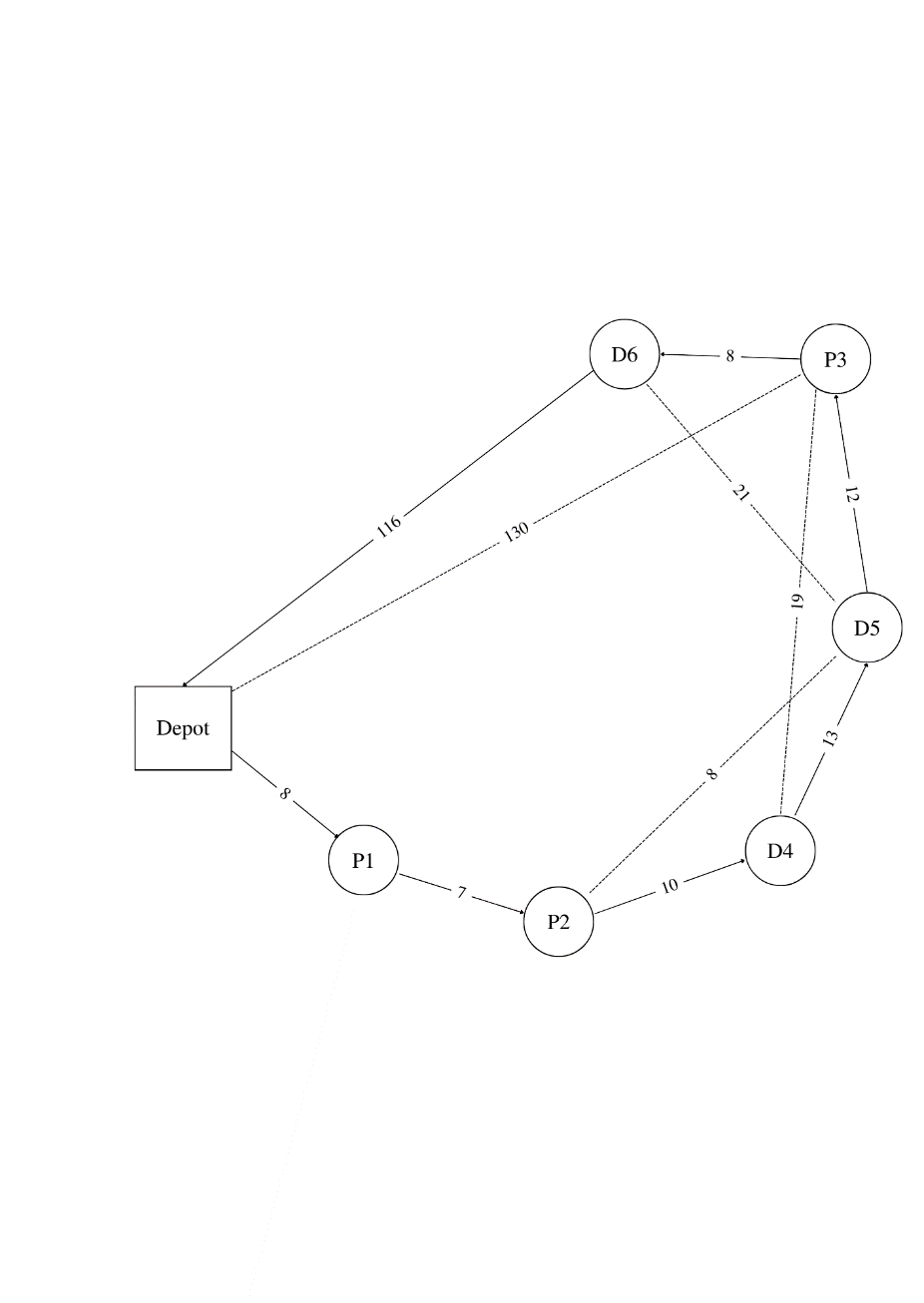
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Dep (o/d)** | **P1** | **P2** | **P3** | **D4** | **D5** | **D6** |
| **Dep**  **(o/d)** | / | 8 | 13 | 130 | 19 | 28 | 116 |
| **P1** | 8 | / | 7 | 22 | 14 | 15 | 11 |
| **P2** | 13 | 7 | / | 25 | 10 | 27 | 14 |
| **P3** | 31 | 22 | 25 | / | 19 | 12 | 8 |
| **D4** | 19 | 14 | 10 | 41 | / | 13 | 28 |
| **D5** | 28 | 15 | 27 | 37 | 13 | / | 21 |
| **D6** | 5 | 11 | 14 | 27 | 28 | 19 | / |

|  |  |  |  |
| --- | --- | --- | --- |
| **RULE** | **Cliente 1** | **Cliente 2** | **Cliente 3** |
| % SA | 11% | 19% | 70% |
| % NS | 43% | 41% | 16% |
| % ENS | 0% | 8% | 92% |
| **% Media** | **18%** | **22%** | **60%** |

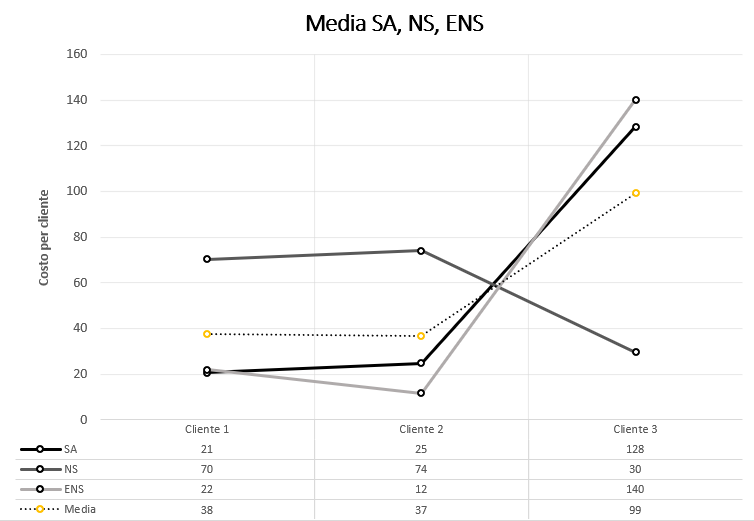


Da notare come, anche nel caso in cui la vicinanza tra il cliente 3 e gli altri, sia minore, il risultato rimane quasi costante. La regola *Extreme* *Neighbors savings*, “tira” l’aumento dei costi verso il cliente 3, insieme alla regola *Stand Alone*. Si prova a ridistribuire la situazione, facendo in modo che non vada a 0.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Dep (o/d)** | **P1** | **P2** | **P3** | **D4** | **D5** | **D6** |
| **Dep**  **(o/d)** | / | 8 | 13 | 130 | 19 | 28 | 116 |
| **P1** | 8 | / | 7 | 22 | 14 | 15 | 11 |
| **P2** | 13 | 7 | / | 25 | 10 | 8 | 14 |
| **P3** | 31 | 22 | 25 | / | 19 | 12 | 8 |
| **D4** | 19 | 14 | 10 | 41 | / | 13 | 28 |
| **D5** | 28 | 15 | 27 | 37 | 13 | / | 21 |
| **D6** | 5 | 11 | 14 | 27 | 28 | 19 | / |



|  |  |  |  |
| --- | --- | --- | --- |
| **RULE** | **Cliente 1** | **Cliente 2** | **Cliente 3** |
| % SA | 12% | 14% | 74% |
| % NS | 40% | 43% | 17% |
| % ENS | 13% | 7% | 80% |
| **% Media** | **22%** | **21%** | **57%** |



Per concludere queste considerazioni, anche in quest’ultimo caso non vi è una variazione troppo significativa della ripartizione del costo. In tutti i casi trattati, la media finale assume un andamento simile alla regola *Stand Alone*, indice del fatto la distanza tra il deposito e i clienti da servire è il parametro con peso maggiore. Ma questo viene smorzato dalle altre due regole, mantenendo il risultato finale sullo stesso ordine di grandezza e riuscendo nell’obiettivo trovare di una strategia di allocazione dei costi al di fuori del paradigma dello Shapley value che potesse essere consistente.

# Capitolo 2 – Lo Shapley Value

## Storia dello Shapley value

Con l’uscita nel 1953 del suo paper “*A value for n-person games*” Lloyd S. Shapley formula un approccio atto ad aggirare tutta quella gamma di problemi della Teoria dei giochi, che consiste nell’integrazione strategica complessa. Infatti egli proponeva che fosse possibile valutare, in modo numerico, il "value", il valore nel partecipare ad un determinato gioco. Da questo studio è emerso lo Shapley Value, che, dalla data della sua prima pubblicazione, continua a suscitare interesse in ricercatori e studiosi ed è stato il motivo per cui, nel 2012, è stato assegnato il premio Nobel per l’economia al suo autore. Per comprendere a pieno i motivi per i quali questo valore sia così importante in ambito accademico, è necessario fare un passo indietro di quasi dieci anni, fino 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 quelle che sono tutte le condizioni che portino a delle preferenze in un giocatore, che messo davanti ad una serie di opzioni lo portino a scegliere quella che massimizzi la propria funzione di utilità, massimizzando 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 ad un unico numero: l’utilità attesa. La classe di giochi persa in esame è la così detta “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 in tramite un procedimento concordato tra tutti i membri, ma senza alcun costo aggiuntivo.

Il core dei TU games è trovare la serie della distribuzione dei payoff di ciascun giocatore e con il vincolo che la somma di tali 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) interessati ad un potenziale oggetto in possesso di 1.

|  |  |
| --- | --- |
| **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 |

* 1 valuta l’oggetto €10
* 2 valuta l’oggetto €20
* 3 valuta l’oggetto €30

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 mordicare la 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 soluzione 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 giocare può esibire, dovuti principalmente dalle 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, ma anche in questo caso, 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. Queste sono in ampio spettro un riflesso della complessità dell’interazione strategica tra giocatori e facendo aumentare la necessità nel trovare un metodo più semplice per effettuare una valutazione preliminare dei giochi. È in questo contesto che si inserisce lo Shapley value.

## La forma dello 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 che esprimesse quanto fossero convenienti le 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”) di partecipare ad un dato gioco.

Indichiamo con U, l’universo di tutti i possibili giocatori, da questo punto di partenza, Shapley aveva in mente di considerare tutti i giochi ai quali avrebbero potuto partecipare tutti i giocatori appratenti ad U. Viene definita una funzione che assegni ad ogni gioco , un numero per ogni giocatore . viene chiamato “*Shapley value*” ed è il valore, espresso sotto forma di numero, associato al gioco . 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 :

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** | |
|  |  |
|  |  |
|  |  |
|  | |

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 , allora

1. **Carrier axiom**: questo assioma è diviso in spesso considerato da dividere in due assiomi distinti: efficency axiom & dummy player property axiom:
   1. **Efficiency axiom**: preso ogni gioco v e il valore dato dal gioco, questo deve essere una ripartizione di quello che il “tutto” riesce ad ottenere. In altre parole la somma dei su tutti i giocatori i di un qualsiasi sottoinsieme deve essere pari a .

* 1. **Dummy player property axiom**: assegniamo il valore di 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 (, il numero reale: . Se prendiamo tale che , con , allora i è un dummy player e quindi .

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

per ogni

Shapley dimostra che esiste una e una sola funzione che soddisfa contemporaneamente tutti e tre gli assiomi, ed è :

[ 1. ]

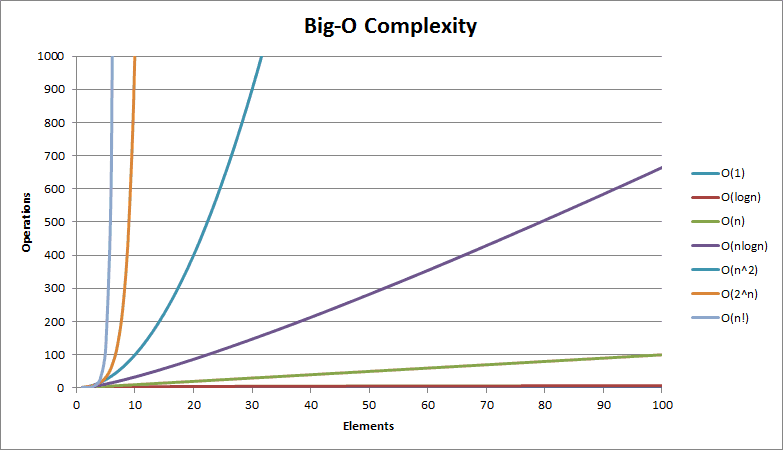
Lo Shapley Value si presenta come il *contributo marginale* medio di un giocatore i in una specifica coalizione, rispetto le possibili permutazioni dei giocatori. In un gioco cooperativo, un giocatore può avere un peso molto piccolo, ma, determinante da date condizioni, il suo effetto è determinante 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 definire il problema, immaginiamo di porci in una stanza vuota, con 3 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).

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 se, 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 per tutte le permutazioni degli n elementi che compongono i nostri giocatori, sommando tutti i valori marginali del giocatore i per poter trovare il suo *contributo marginale*.

### Appendice 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 stanzia 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.

Pertanto, è possibile calcolare lo Shapley Value in tempo polinomiale, ma solo nei casi in cui il numero di giocatori non è elevato. Già se il numero di giocatori fosse 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.

## 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:

[ 1. ]

= Insieme di tutti i giocatori che precedono i nella permutazione

= Insieme di tutti gli ordinamenti possibili dati dalle n permutazioni

= *Contributo marginale* del giocatore i secondo *funzione di costo* v

Si definisce come l’insieme di tutti i giocatori che precedono il giocatore i nella permutazione (insieme di tutti gli ordinamenti possibili dati dalle n permutazioni), pertanto il *contributo marginale* 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:***

|  |  |
| --- | --- |
| **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 |

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

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 |

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:

|  |  |
| --- | --- |
| **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 |



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:***

Se già i giocatori non fossero 3, ma 4, la complessità computazione andrebbe ad aumentare in maniera esponenziale, come visto nell’appendice sulla complessità 1. Prendiamo il caso della tabella sulla destra, che riporta i valori della funzione di costo .

Ecco che i calcoli per lo Shapley value diverranno:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **i = 1** | | **i = 2** | | **i = 3** | | **i = 4** | |
| **P4** | **Contributo marginale** | **Calc.** | **Contributo marginale** | **Calc.** | **Contributo marginale** | **Calc.** | **Contributo marginale** | **Cal.** |
| 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 |

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:

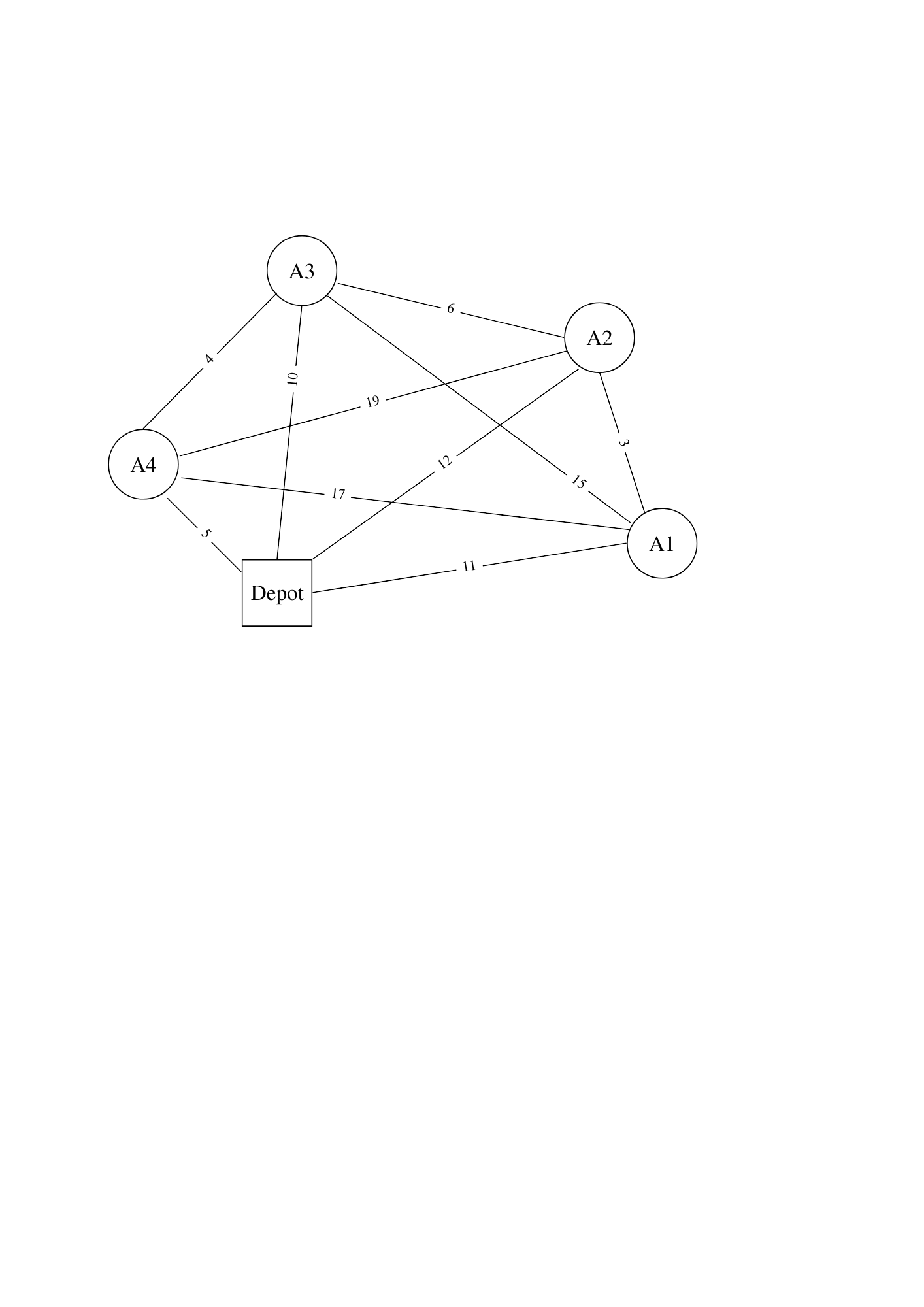


### Applicare ApproShapley ad un problema di TSP

Una volta compreso il funzionamento dello Shapley value, bisogna comprendere 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 |

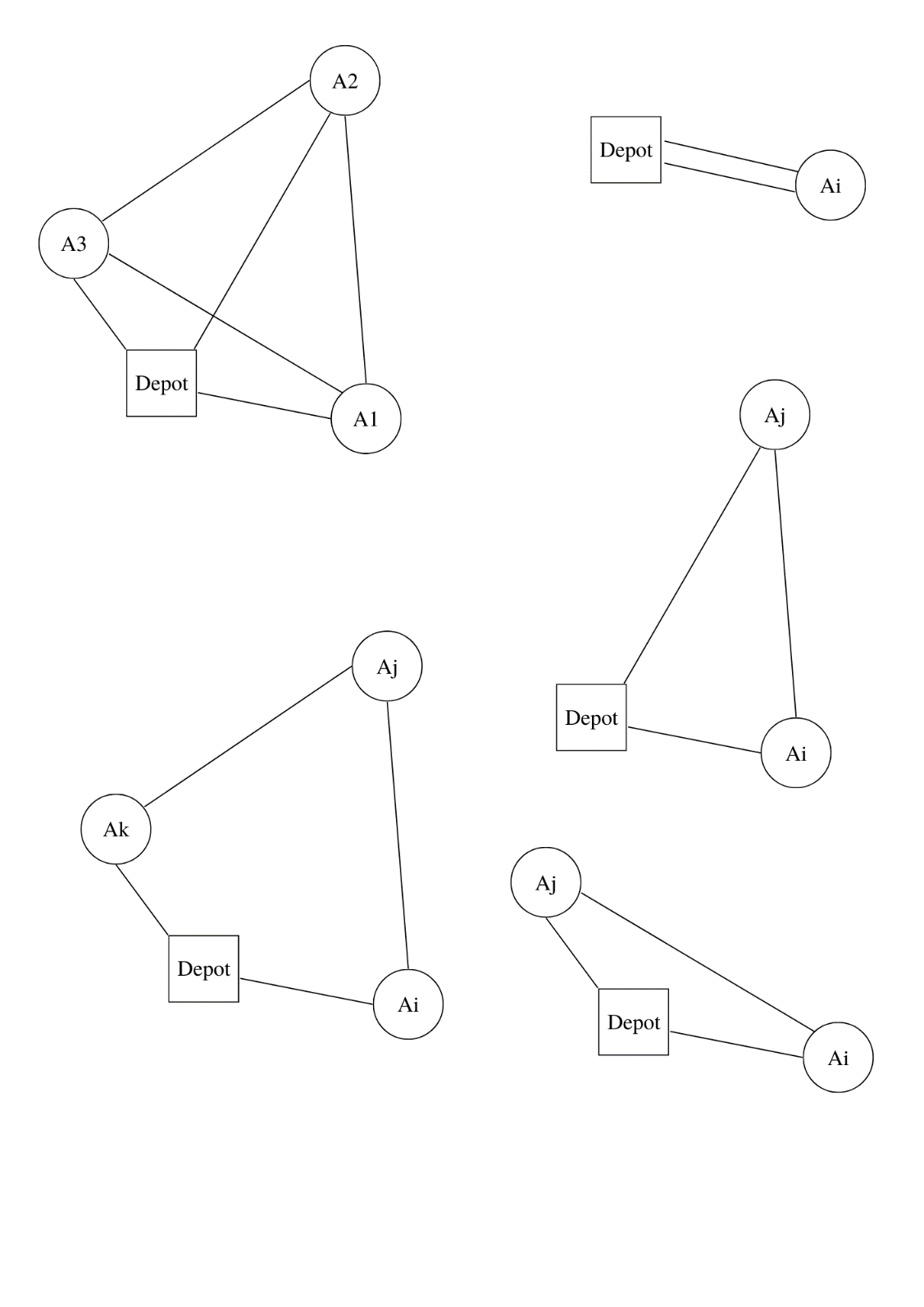


Per andare a creare la matrice *funzione di costo*, bisogna partire dalla definizione: questa rappresenta il costo v (i, j, k) che bisogna sostenere per manere in soluzione i nodi i, j, k. Va da se quindi, che il costo in soluzione di v (1, 2, 3) consiste nel costo minore per raggiungere questi 3 nodi. 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 passo passo:

* **v (0)**

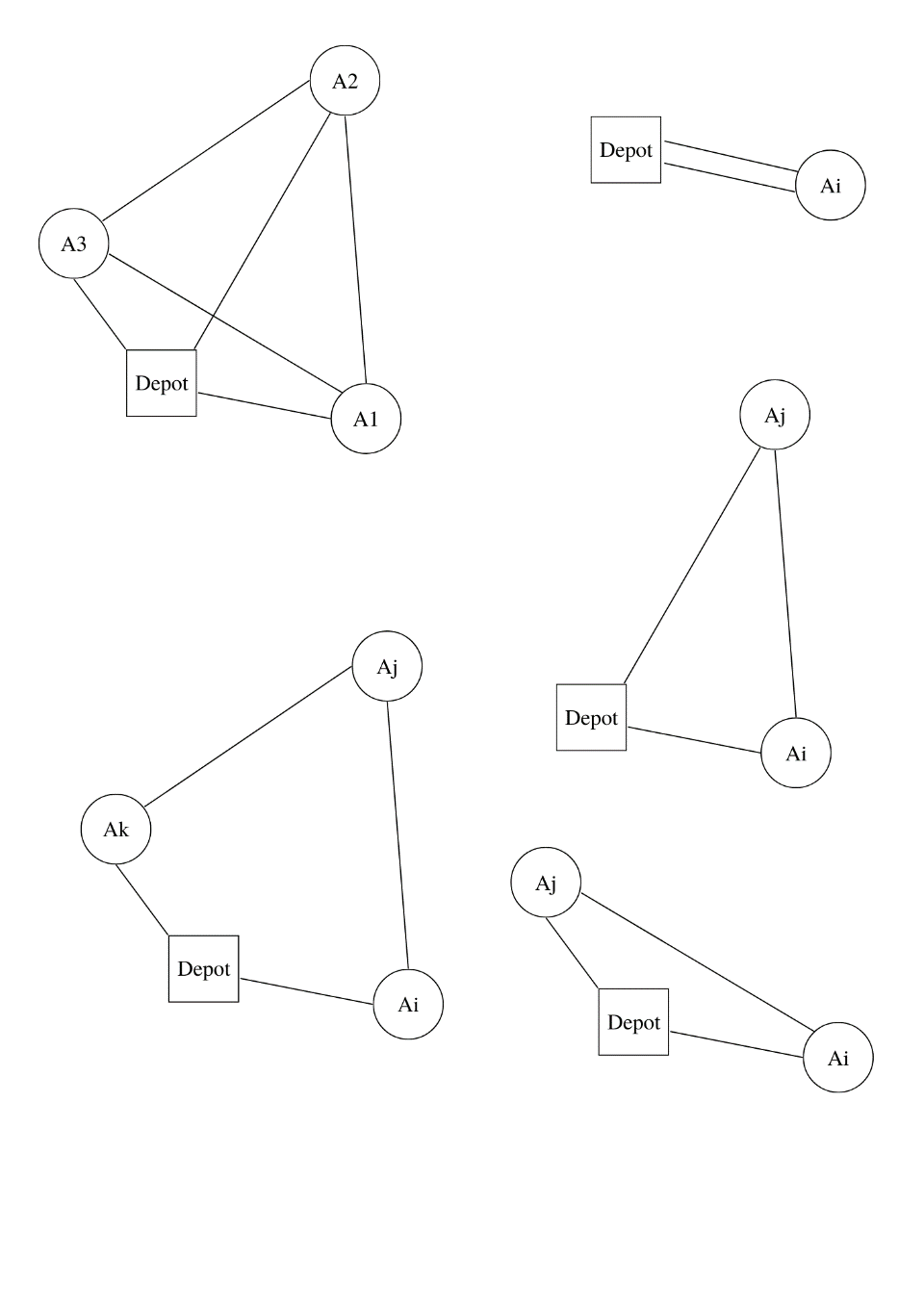
La *funzione costo* v (0) è esattamente pari a 0 dato che è l’insieme vuoto

|  |  |  |
| --- | --- | --- |
| **Funzione di costo** | **Calcoli** | **Valore** |
| v(0) | Insieme vuoto | 0 |

* **v (i)**

Avendo messo in soluzione un unico nodo, il costo ottimo per partire dal nodo 0 (deposito) e ritornarvici, è esattamente 2 \* d\_0i

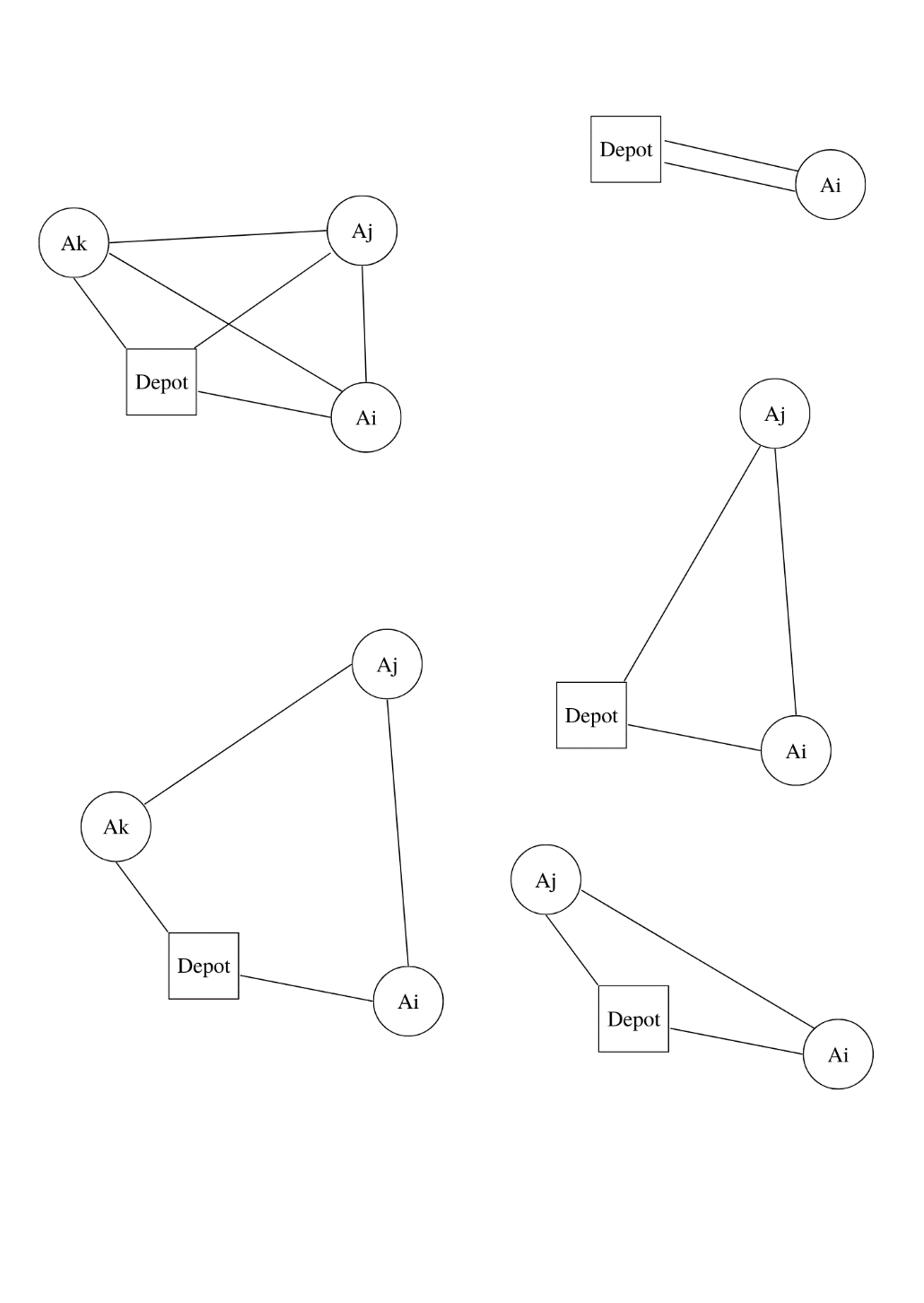
|  |  |  |
| --- | --- | --- |
| **Funzione di costo** | **Calcoli** | **Valore** |
| v(1) | 2 \* d\_01 | 22 |
| v(2) | 2 \* d\_02 | 24 |
| v(3) | 2 \* d\_03 | 20 |
| v(4) | 2 \* d\_04 | 10 |



* **v (i, j)**

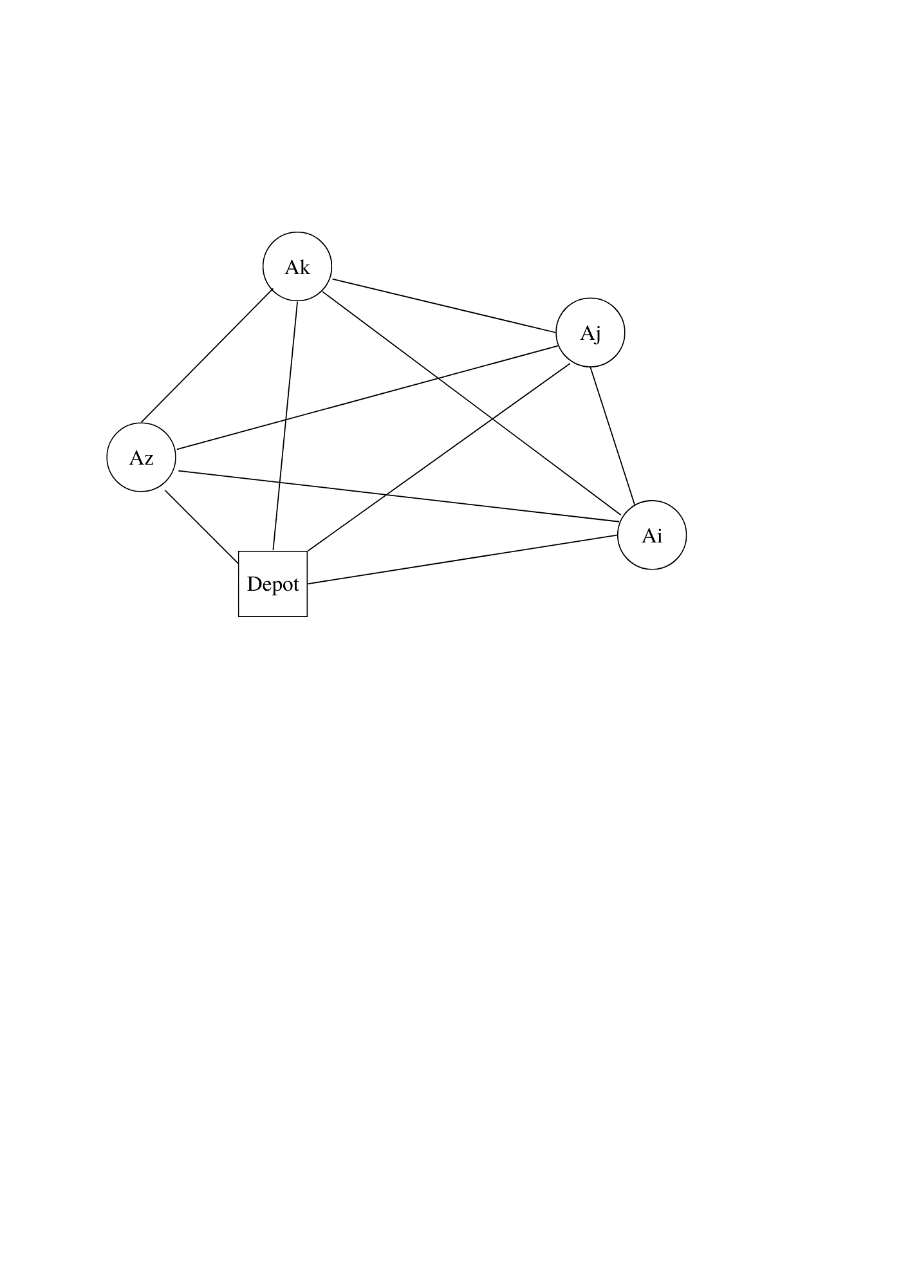
In soluzione adesso si trovano due nodi, ma anche qui il problema di TSP è di facile risoluzione, dato che il risultato non è altro che la somma delle distanze d\_01 + d\_12 + d\_02

|  |  |  |
| --- | --- | --- |
| **Funzione di costo** | **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), e trovando il risultato minore.

|  |  |  |
| --- | --- | --- |
| **Funzione di costo** | **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, … ,p12) per trovare il percorso minimo e risolvere il problema del TSP. Si noti che questo è tutto il grafo.

|  |  |  |
| --- | --- | --- |
| **Funzione di costo** | **Calcoli** | **Valore** |
| v(1, 2, 3, 4) | min(p1, … ,p12) = d\_01 + d\_12 + d\_23 + d\_34 + d\_04 | 29 |

Ecco che dalla matrice delle distanze si è passati alla *funzione di costo*, indispensabile per la risoluzione dello Shalpey value. È molto importante notare come, qualsiasi sia n, la *funzione di costo* che comprende tutti i nodi, sia proprio la risoluzione del TSP dell’intero grafo. Riprendendo le soluzioni di questo esempio, il percorso ottimo è:

[0 – 1 – 2 – 3 – 4 – 0] con costo totale Ctot = 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 è esattamente la 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.

## ApproShapley O(1) – prima approssimazione

Come già anticipato, il problema principale, nell’applicare lo Shapley Value a questa tipologia di problemi, è la complessità. Questo verrà analizzato 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, me 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, il così chiamato Airport problem.

### Airport problem – caso monodimensionale

Immagine che contiene schizzo, diagramma, disegno, Disegno tecnico

Descrizione generata automaticamenteL’airport problem, formulato da Littlechild nel 1973 (verifica) è ampiamente noto in letteratura come un caso particolare dello Shapley value. Come ipotesi di base, di 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 Aj, con j che va da 1 a n. Per cui, come si può notare in figura, è presente un vettore delle distanze dj per ogni nodo j e un vettore rj che corrisponde alla distanza incrementale che separa un cliente dal suo predecessore. Pertanto, rj non è altro che il *contributo marginale* che il giocatore j apporta alla soluzione finale e possiamo rappresentarlo come:

[1.3]

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:

[ 1. ]

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

* Tutti gli n clienti, avranno l’onere di pagare 2r1, che rappresenta il viaggio di andata e ritorno dal deposito al nodo A1 e il costo sarà ripartito tra tutti gli n clienti.
* Tutti gli n – 1 clienti (tranne A1), avranno l’onere di pagare 2r2 e il costo sarà ripartito tra tutti gli n – 1 clienti.
* Tutti gli n – 2 clienti (tranne A1 e A2), avranno l’onere di pagare 2r3 e il costo sarà ripartito tra tutti gli n – 2 clienti.

Questo fino a raggiungere il cliente An. Come è già stato riportato, il valore rj 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 (A1, A2, A3). La strada che serve tutti gli n clienti è unica condivisa tra tutti e i tratti che separano gli N nodi sono equidistanti (r1 = 5, r2 = 5, r3 = 5) generando la seguente matrice delle distanze (d1 = 5, d2 = 10, d3 = 15). Ovviamente la condizione del TSP è che si debba partire e rientrare al deposito, per cui la massima distanza percorsa sarà 2d3 = 30. Nella forma originale dello Shapley value, quest’ultimo valore non è altro che e dato il fatto che la validità dei 3 assiomi non viene in alcun modo alterata, allora .

* = 3.33

Si nota subito che l’assioma 3 è rispettato ( 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 multi-dimensionale, che sarà lo scopo anche di questa ricerca, ma andiamo per gradi.

Per sostituzione diretta dei valori rj dalla [ 1.3 ] nella [ 1.4 ], è possibile ottenere lo Shapley value del nodo coda An.

[ 1. ]

La logica alla base sta nel fatto di prendere in esame la distanza massima, ovvero il costo che An 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:



Possiamo estrapolare questo principio generare una funzione di *contributo condiviso* Sc per un determinato numero di argomenti .

[ 1. ]

Partendo dalla [ 1.5 ], l’operazione trasla “inverte” il vettore preso in esame, per cui è stato scelto il vettore che rappresenta il vettore ordinato in maniera decrescente. A questo punto possiamo riscrivere la [ 1.5 ] come segue:

[ 1. ]

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

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

[ 1. ]

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 vi ci si sottrae il *contributo condiviso* di tutti i partecipanti che lo hanno preceduto. Da notare il fattore appare n – k volte nella funzione Sc come il contributo che va da k + 1 ad n.

Per completare l’esempio:

* Nota: appare 2 volte dato che n – k = 2
* Nota: appare 1 volta dato che n – k = 1

Immagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamenteSono stati forniti quindi due metodi equivalenti, [ 1.4 ] e [ 1.8 ], 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.

Come si nota in figura, a sinistra del deposito esiste un vettore di clienti Bj, con j che va da 1 a n e con un vettore lj 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:

[ 1. ]

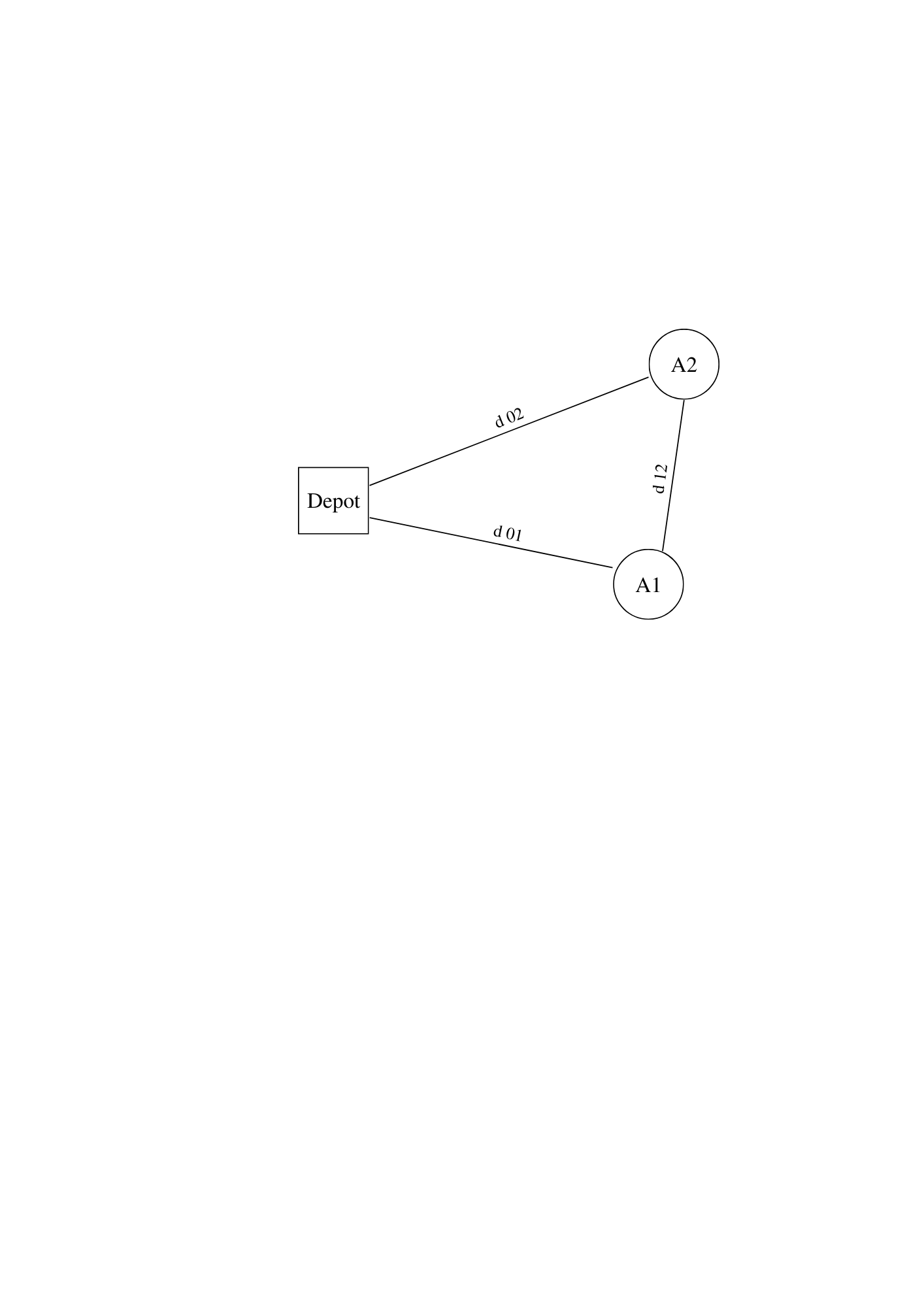
I coefficienti e dipenderanno solo e soltanto della posizione relativa del generico k preso in esame e dal loro numero di predecessori. La prima sommatoria, infatti, identifica la parte sinistra del deposito, la seconda invece la parte destra. Nel caso specifico della [ 1.9 ] si sta cercando di un generico k-esimo elemento di A e si può dimostrare che tutti i coefficienti debbano essere esattamente equivalenti della [ 1.5 ] e di conseguenza tutti i coefficienti 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 modo-dimensionali distinti, il loro Shapley value non dipenderà dai valori assunti dalla controparte.

Si è arrivati quindi alla conclusione che il caso di studio del “airport problem”, e quindi il caso mono-dimensionale del problema del TSP, si può risolvere scomponendolo in due “giochi” distinti. Da un lato avremo Ai clienti situati alla destra del deposito, dall’altro avremo Bi clienti situati a sinistra del deposito. Per il calcolo dello Shapley value di un generico Ak, è possibile utilizzare o la [ 1.4 ] o la [ 1.8 ], prendendo in considerazione tutti i clienti Aj, con j che va da 1 a n e tralasciando i clienti Bj. Allo stesso modo, per il calcolo dello Shapley value di un generico Bk, è possibile utilizzare o la [ 1.4 ] o la [ 1.8 ], prendendo in considerazione tutti i clienti Bj, con j che va da 1 a m e tralasciando i clienti Aj. Questa indipendenza sarà fondamentale nel caso multi-dimensionale.

### Appendice 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(nlogn), 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 in tempi polinomiali, dall’altro non è un’approssimazione accettabile.

### La forma di Appro O(1) nel caso multidimensionale

Si passi ora al caso multidimensionale, ma prima di andare avanti si vuole concentrare il focus sul fatto che tutti gli esperimenti da qui in avanti saranno focalizzati sul caso in due dimensioni, affidandoci solo sulle distanze reciproche tra le località e con l’ipotesi che la distanza reciproca tra le località si identica e che soddisfi la disuguaglianza triangolare. Questo caso particolare, infatti, è mirato per andare ad analizzare il caso non euclideo delle reti stradali (TSP).

Si assuma di quindi di partire da un deposito D marcato come 0 e dover servire n clienti , con una matrice delle distanze simmetrica .

Per arrivare alla prima grande approssimazione, si consideri il caso in cui si hanno 2 clienti e e applichiamo per la prima volta lo Shapley value ad un problema di TSP non monodimensionale.

|  |  |
| --- | --- |
| **Funzione di costo** | **Valore** |
| v(0) | 0 |
| v(1) |  |
| v(2) |  |
| v(1, 2) |  |

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 deposido 0. Pertanto, partendo dalla matrice delle distanze, ci costruiamo, come spiegato in precedenza, la *funzione di costo*.

Questo è reso più comprensibile tramite il proseguo <dell’esempio:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **i = 1** | | **i = 2** | |
| **Permutazioni** | **Contributo marginale** | **Calcolo** | **Contributo marginale** | **Calcolo** |
| 1, 2 | v(1) - v(0) |  | v(1,2) - v(1) |  |
| 2, 1 | v(1,2) - v(2) |  | v(2) - v(0) |  |
|  | Somma: |  | Somma: |  |

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

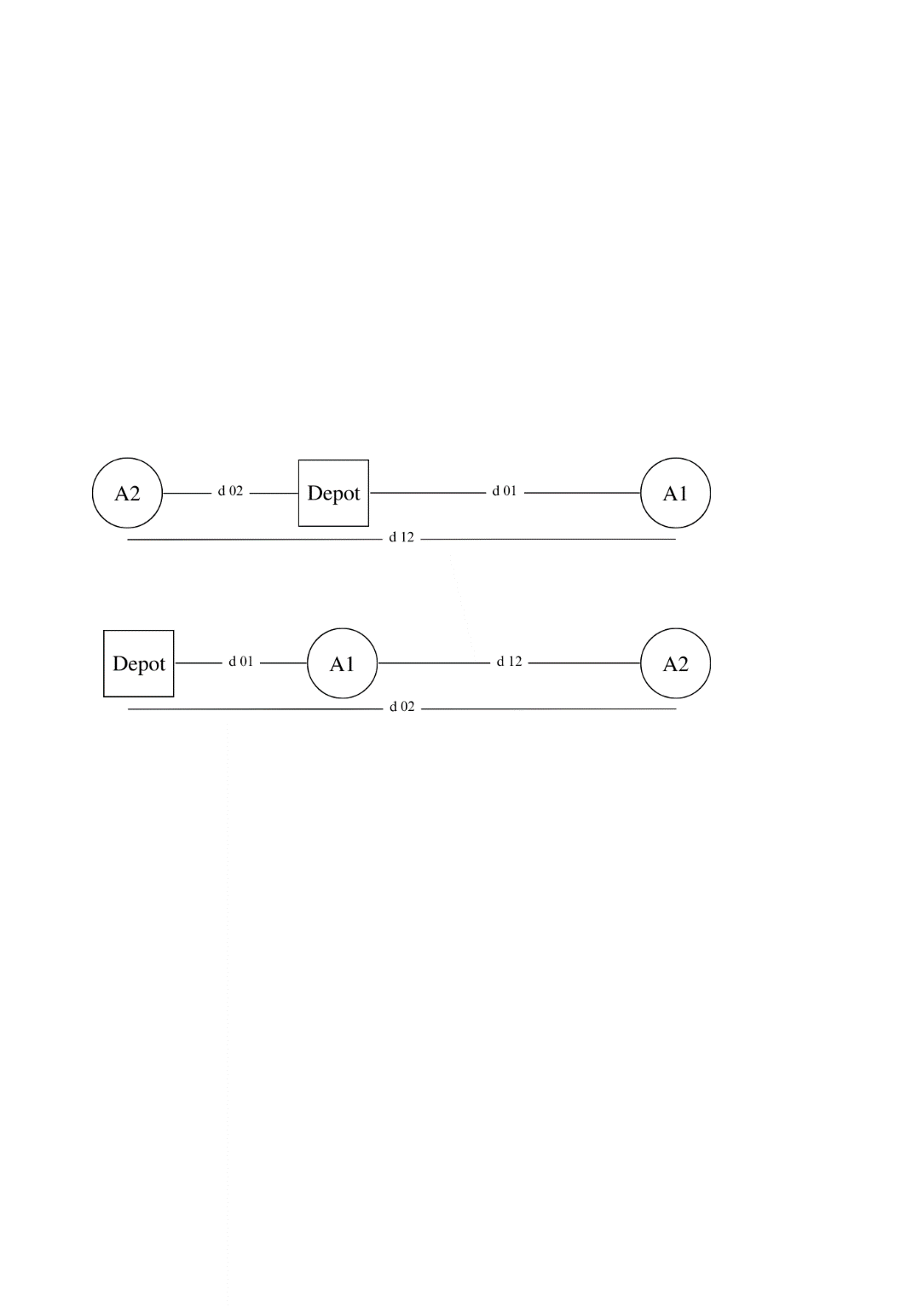
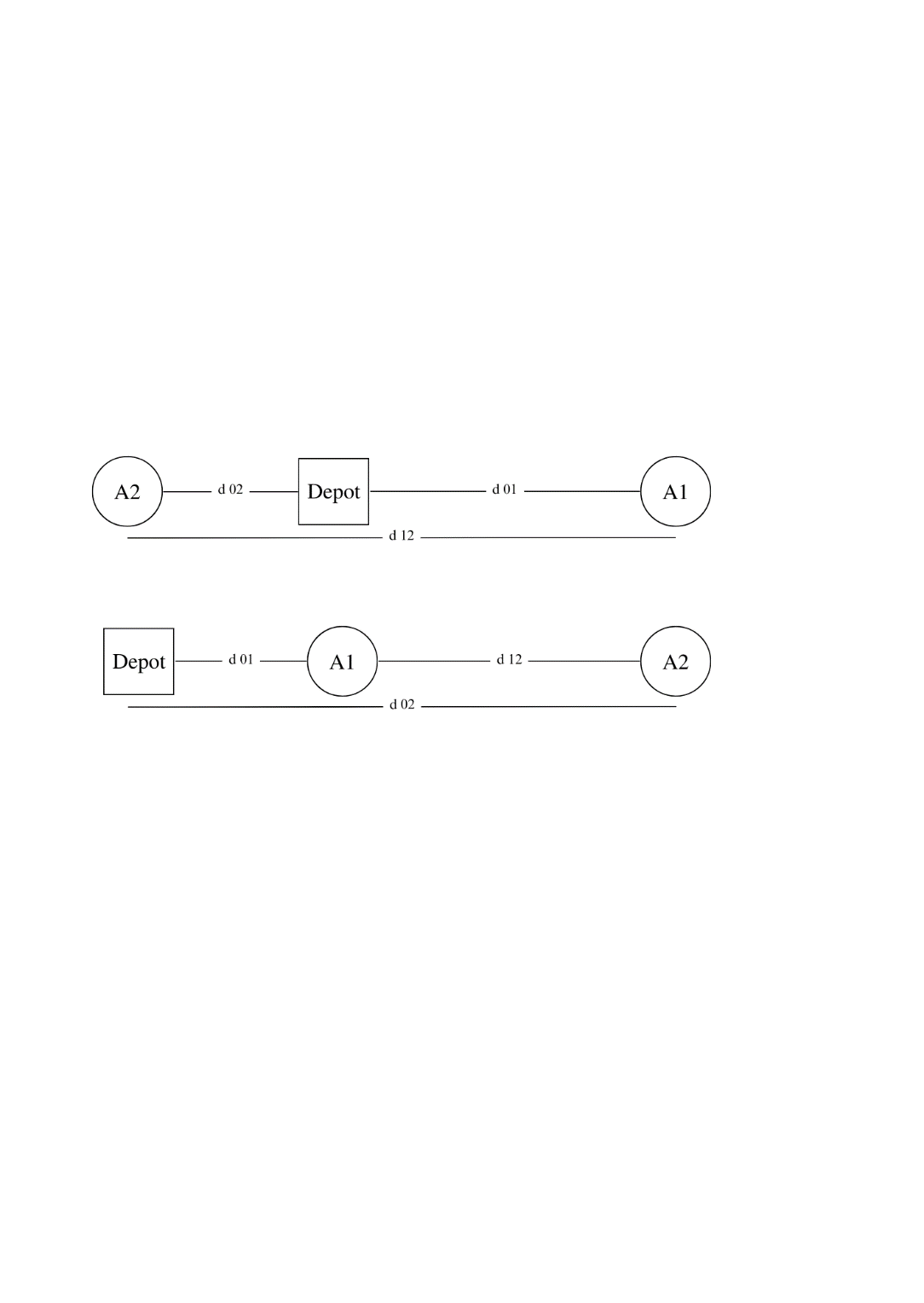
* )

Che possono essere riscritti come:

* )

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

La *distanza condivisa* che è stata trovata riscontra delle analogie con il *contributo condiviso* 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.



Se a questo ci aggiungiamo che la *distanza condivisa* di un cliente con sé stesso, è esattamente il doppio della propria (, allora è possibile riscrivere il risultato dello Shapley value per i clienti 1 e 2 come:

Riprendendo ora il *contributo marginale* dal caso monodimensionale e applicandolo al caso specifico, questo diventa .

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:

[ 1. ]

è quindi l’approssimazione di ordine 1 dello Shapley value, anche detta Appro O(1). Essenzialmente viene fornito un valore approssimato del cliente k partendo dalla *distanza condivisa* della posizione con se stesso () a meno del *contributo condiviso* di tutte le *distanze condivise* tra e tutti gli altri () 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* è pari a zero. Altro elemento da considerare è che nel caso in cui i clienti siano 2, allora 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* .

Nel momento in cui si voglia utilizzare *Appro O(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, ma per completezza essi erano:



|  |  |
| --- | --- |
| **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 |

Immagine che contiene schizzo, disegno, diagramma, linea

Descrizione generata automaticamente

Il costo ottimo della soluzione del TSP per visitare tutti i nodi è [0 – 1 – 2 – 3 – 4 – 0] con costo totale Ctot = 29. Per applicare Appro O(1): la prima iterazione consiste nel calcolare la matrice delle distanze condivise

|  |  |  |
| --- | --- | --- |
| **Distanze condivise** |  | **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 |

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

Applicando con che rappresenta il vettore ordinato in maniera decrescente, riscriviamo:

* con
* con
* con
* con

Sostituendo i rispettivi valori numerici si ottiene:

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 a vicina a Ctot, con un certo margine di errore. Pertanto tali risultati andranno normalizzati sul valore totale del costo di percorrenza. Su questo, come già anticipato, si approfondirà nei successivi capitoli relativi ai calcoli.

## 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 O(1) fornisce un primo grado di approssimazione introducendo il concetto di *distanza condivisa*, la quale, solo sotto specifiche 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 ,* generato dalla *funzione di costo* v() e la *distanza condivisa .* Per spiegare come questo possa essere possibile, si prenda in considerazione il seguente esempio:

***Esempio:***

Immagine che contiene diagramma, linea, cerchio

Descrizione generata automaticamenteCi si metta nel caso di n = 3 clienti con un deposito di partenza e una matrice delle distanze dell’ordine (n + 1) \* (n + 1). Da questa matrice delle distanze, si supponga 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 [1.1], esistono 3 coalizioni di cui 1 non fa parte:

* Immagine che contiene Carattere, diagramma, linea, testo

  Descrizione generata automaticamente**Coalizione 1**: il cliente 1 è preso da solo e quindi la S è vuoto ( e quindi la *funzione di costo* è pari alla distanza percorsa in andata e in ritorno

Immagine che contiene diagramma, schizzo, linea

Descrizione generata automaticamente

* Immagine che contiene diagramma, schizzo, linea

  Descrizione generata automaticamente**Coalizione 2**: il cliente 1 è preso in sequenza con uno degli altri clienti e quindi l’insieme S avrà cardinalità pari a 2 (). Avremo pertanto due *funzioni costo*:

Immagine che contiene diagramma, schizzo, linea

Descrizione generata automaticamente

* **Coalizione 3**: il cliente 1 è preso in sequenza con gli altri 2 clienti e quindi l’insieme S avrà cardinalità pari a 1 (). 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.

nel nostro caso.

NOTA: ricordiamo che corrisponde alla soluzione del problema del TSP. Più in generale, nel caso di n clienti da servire, la *funzione costo* corrisponde alla soluzione del problema di TSP.

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

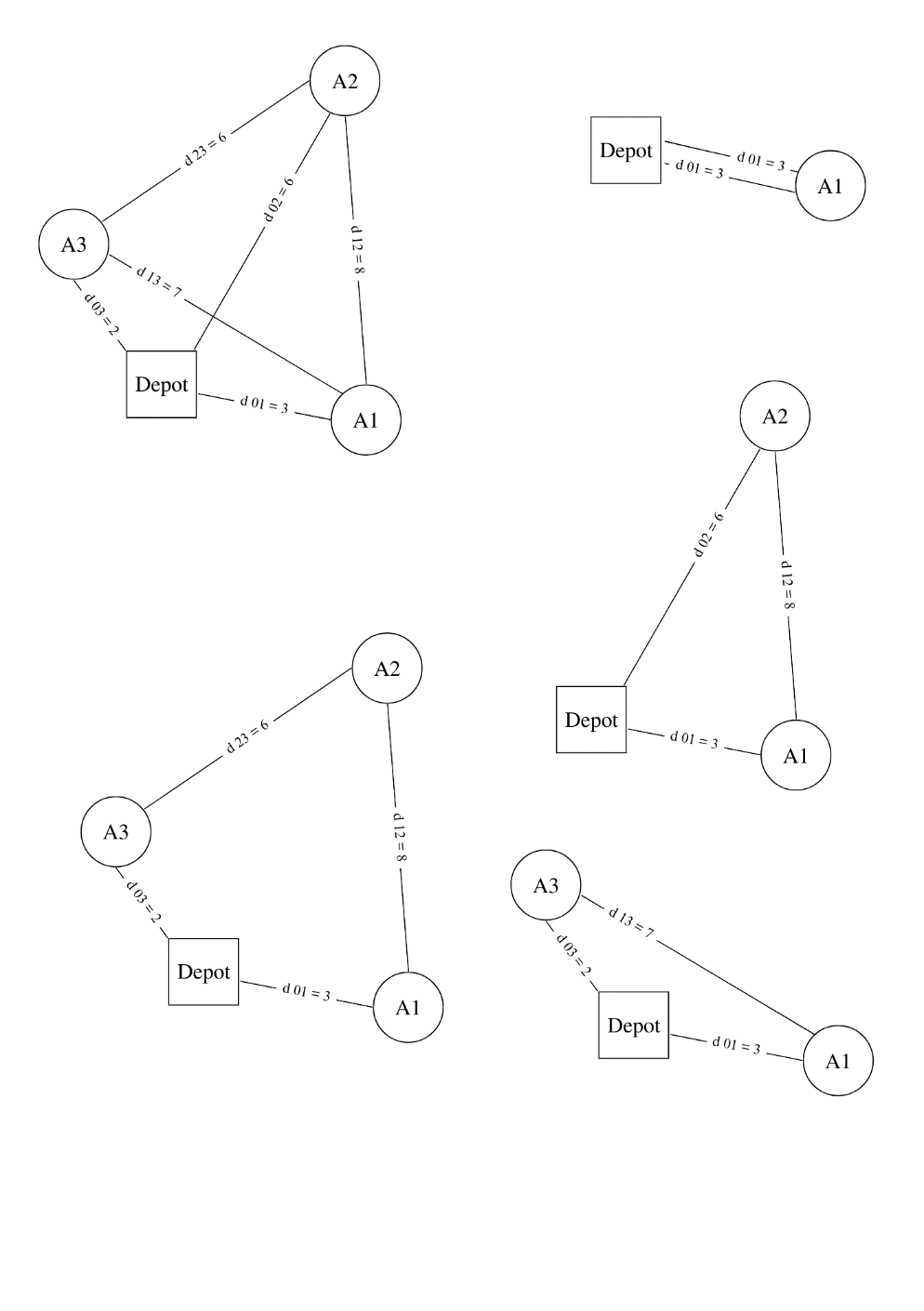
|  |  |  |
| --- | --- | --- |
| **Funzione di costo** | **Calcolo in funzione delle distanze** | **Valore** |
| v(0) | 0 | 0 |
| v(1) |  | 6 |
| v(2) |  | 12 |
| v(3) |  | 4 |
| v(1, 2) |  | 17 |
| v(1, 3) |  | 12 |
| v(2, 3) |  | 14 |
| v(1, 2, 3) |  | 19 |

|  |  |
| --- | --- |
| **Distanze** | **Valori** |
| d\_01 | 3 |
| d\_02 | 6 |
| d\_03 | 2 |
| d\_12 | 8 |
| d\_13 | 7 |
| d\_23 | 6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **0** | **A1** | **A2** | **A3** |
| **0** | / | 3 | 6 | 2 |
| **A1** | 3 | / | 8 | 7 |
| **A2** | 6 | 8 | / | 6 |
| **A3** | 2 | 7 | 6 | / |

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 |





Il cliente 3 è si 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. Opposto invece per il cliente 2, che si vede allocare circa il 50% del costo totale.

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

Che è possibile dividere in *funzione del costo* delle 3 coalizioni

Riscriviamo lo Shapley value utilizzando la *distanza condivisa* tra i clienti i e j introdotto nel paragrafo precedente

* **Coalizione 1**
* **Coalizione 2**
* **Coalizione 3**

Poniamoci nel caso in cui il percorso sia il minore:

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

e nel caso generale:

Alla fine, diventa:

Data la sua natura, questa riscrittura dello Shapley Value coincide con l’Appro O(1) , nel caso in cui il valore sia o o . Ci si ponga nel caso in cui il minore dei tre sia :

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

Dove corrisponde all’intersezione formale delle due distanze condivise e . È quindi possibile fornire una generalizzazione del caso di partenza [1.10] come:

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* on la *distanza condivisa*. La complessità si sposta in 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:

La complessità si ritrova in . Abbiamo visto come sia possibile ottenere un'approssimazione migliore di Appro-O(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, a approssimare qualsiasi m-intersezione formale sconosciuta per m ≥ 3 come segue:

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

Vengano rinominate come le n − 1 distanze condivise di un generico con senza preoccuparci troppo inizialmente di quale particolare ordine di indicizzazione scegliamo.

* Fissato il nodo k, le *distanze condivise* diventano

Si indichi con la matrice simmetrica (n − 1) × (n − 1) di 2-intersezioni formali, cioè per

la voce

***Esempio:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | / | 1 | 2 | 4 | 7 |
|  | 1 | / | 3 | 5 | 8 |
|  | 2 | 3 | / | 6 | 9 |
|  | 4 | 5 | 6 | / | 10 |
|  | 7 | 8 | 9 | 10 | / |

Approfondiamo il concetto della matrice , 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 e, scegliendo arbitrariamente i valori, l'elemento più basso di , è il successivo più alto e così via.



Con questo ordine dominato dall'indice delle voci in , è facile individuare quali intersezioni di ordine superiore sono approssimate da ciascuna particolare 2-intersezione, secondo .

In , tutti i sottoinsiemi di intersezione in cui appare la coppia saranno minimamente dominati per :

* La coppia dominerà minimamente, con la sua intersezione, tutti i sottoinsiemi di intersezione in cui compaiono e possono contenere anche , e
* Le coppie e domineranno minimamente, con le loro 2 intersezioni, tutti i sottoinsiemi di intersezione in cui compaiono e possono contenere anche e
* Le coppie , e domineranno minimamente, con le loro 3 intersezioni, tutti i sottoinsiemi di intersezione in cui compaiono e possono contenere anche
* Le coppie , , e non domineranno nessun elemento

Per qualsiasi dimensione n e per una qualunque matrice dominata dall'indice, vi sarà:

* Una coppia corrispondente all'elemento minimo di , la cui intersezione dominerà tutti gli insiemi di intersezione risultanti dall'aggregazione con qualsiasi con i restanti n − 3 valori di s: .
  + L’intersezione è un 2- intersezione di rango n – 3
* Le coppie e le cui intersezioni e domineranno minimamente tutti gli insiemi di intersezione da aggregazioni con uno qualsiasi dei n − 4 valori s: .
  + L’intersezione e sono un 2- intersezione di rango n – 4
* Le coppie , e le cui intersezioni , e domineranno minimamente tutti gli insiemi di intersezione da aggregazioni con uno qualsiasi dei n − 5 valori s: .
  + L’intersezione , e sono un 2- intersezione di rango n – 5
* Le coppie , , e le cui intersezioni , , e non domineranno nessun insieme
  + L’intersezione , , e 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: . Di tutte le 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 generata da n = 6:

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

Nella matrice , il rango di una qualsiasi delle sue voci, diciamo in posizione è facile da calcolare, in quanto il numero di indici l (con ) sia tale, che entrambe le seguenti condizioni tengano:

Sebbene sia improbabile avere un ordine dominato dall'indice come quello proposto nell’empio, si scopre che averne una versione simile, sotto qualche permutazione delle voci della matrice, è molto comune 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 . Ciò suggerisce che se siamo disposti ad accettare un altro errore minore nel file approssimazione, potremmo adattare tutto in un quadro uniforme. Questo è ottenuto dal raffinando della funzione rank(), definita per ogni voce di , come segue.

Per ogni dato permutiamo prima le sue righe e colonne con la permutazione che corrisponde all'ordine crescente di . 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 riassegnino rango pari a n - 3 al primo elemento in questo elenco ordinato, 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 viene assegnato un rango, .

Questa procedura non modifica le classifiche calcolate se la matrice ha il modello (molto comune) di un ordine dominato dall'indice, o una permutazione di tale matrice. Forza solo le poche istanze di 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):

L’approssimazione Appro (2) consiste in una approssimazione di “secondo ordine” e tiene conto non solo delle *distanze condivise* , ma anche delle loro 2-intersezioni formali . L'idea di questa approssimazione origina dalla stessa idea di Appro-O(1): ridurre i valori delle intersezioni sconosciute di ordine superiore a valori delle loro più piccole intersezioni note di ordine inferiore. Per Appro O(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 . 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 O(2) è esattamente la stessa di quella che risulterebbe da una ricerca esponenziale completa. Ciò significa che Appro O(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.

***Immagine che contiene schizzo, disegno, diagramma, linea

Descrizione generata automaticamenteEsempio:***

Per una maggior comprensione di Appro O(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

|  |  |  |
| --- | --- | --- |
| **Distanze condivise** |  | **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 |

|  |  |
| --- | --- |
| **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 |

Nodo k = 1

Il numero di nodi è n = 4 per cui, volendo calcolare lo Shapley value per il cliente 1, il rango sarà compreso tra 0 e 1. Per assegnare il rango all’intersezione si deve creare la matrice , tenendo in considerazione che

Per creare la matrice , bisogna riassegnare i valori di



In seguito inserire il valore corrispondente nell’intersezione degli s.

Costruiamo la matrice :

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | / | 6 | -2 |
|  | 6 | / | -1 |
|  | -2 | -1 | / |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | / | -2 | 6 |
|  | -2 | / | -1 |
|  | 6 | -1 | / |

Riscritta per far combaciare le dominanze:

# Capitolo 3 – Il programma Python

Testo

# Capitolo 4 – ApproShapley, O(1) e O(2) nel caso di studio

Test

**SHAPLEY VALUE**

= *Contributo marginale* del giocatore i secondo *funzione di costo* v

**APPRO O(1)**

**APPRO O(2)**

# Conclusioni

# Bibliografia

* “*A value for n-person games*” by Lloyd S. Shapley
* 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>
* “*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

<https://gryhasselbalch.com/> LEGGI

<https://github.com/jvkersch/pyconcorde> LEGGI