Università degli Studi di Roma "La Sapienza"
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Elettronica
Dipartimento di Informatica e Sistemistica

Tesi di Laurea

Analisi di competitività di algoritmi online per il problema del riparatore viaggiatore

Luca Allulli

Relatore
Prof. Giorgio Ausiello

Correlatore
Prof. Luigi Laura

Indice

Elenco dei principali simboli utilizzati

$\widetilde{\mathbb{R}}$	$\mathbb{R} \cup \{-\infty, +\infty\}$
$\mathbb{N}^+$	$\mathbb{N} \backslash \{0\}$
L_N-OlTRP	Problema del riparatore viaggiatore online con la funzione obiettivo della latenza netta
L-OlTRP	Problema del riparatore viaggiatore online con la funzione obiettivo della latenza
$F_{\max}$-OlTRP	Problema del riparatore viaggiatore online con la funzione obiettivo del massimo tempo di completamento
$\mathcal{L}_k^{(R)}$	Insieme degli algoritmi per l'OlTRP con lookahead sulle successive k richieste
$\mathcal{L}_\Delta^{(T)}$	Insieme degli algoritmi per l'OlTRP con lookahead sulle prossime Δ unità di tempo
$R(\mathcal{A})$	Rapporto di competitività della classe di algoritmi $\mathcal{A}$
$R(\mathcal{A}, \mathcal{B})$	Rapporto di comparatività tra le classi di algoritmi $\mathcal{A}$ e $\mathcal{B}$
$\mathcal{I}$	Insieme delle istanze di input per un problema

Introduzione

Questa tesi si occupa dei problemi dial-a-ride, inquadrandoli nel naturale contesto online.

Un problema dial-a-ride è un problema di scheduling di un servizio di trasporto su richiesta: alcuni server devono trasportare degli oggetti da un punto ad un altro di uno spazio metrico. Esistono numerose varianti di problemi dial-a-ride, che differiscono per lo spazio metrico utilizzato, per le caratteristiche dei server (numero, capacità, velocità), delle richieste e per la funzione obiettivo. Esempi concreti di applicazioni di problemi dial-a-ride sono lo scheduling di un servizio di radio-taxi, di un impianto di ascensori o di un servizio di recapito della posta.

Tra le varianti dei problemi dial-a-ride troviamo problemi di notevole interesse teorico e pratico, come il problema del commesso viaggiatore e il problema del riparatore viaggiatore. Questi problemi possono essere pensati come problemi dial-a-ride di tipo degenere, in cui la sorgente e la destinazione di ogni richiesta di trasporto coincidono: per soddisfare la richiesta è sufficiente che il server visiti il punto in cui la richiesta è rilasciata, carichi l'oggetto da "trasportare" e lo scarichi immediatamente.

Nella maggior parte delle applicazioni pratiche le richieste di trasporto sono rilasciate man mano che passa il tempo. Se fosse possibile conoscere fin dal principio tutte le richieste che saranno rilasciate, allora si potrebbe organizzare il miglior servizio possibile. In effetti questa situazione non si realizza quasi mai, in quanto le richieste diventano note soltanto quando sono rilasciate. Questa situazione si definisce *online*, mentre quella in cui si conoscono tutte le richieste fin dal principio si chiama *offline*. Il metodo più comunemente utilizzato per valutare la qualità di un algoritmo online è l'analisi di competitività, che si basa sul confronto delle prestazioni dell'algoritmo online con quelle del miglior algoritmo offline, nel caso peggiore per l'algoritmo online.

Per poter misurare le prestazioni di un algoritmo occorre prima di tutto definire la funzione obiettivo, la funzione che assegna un costo ad ogni soluzione. La soluzione ottima è naturalmente quella di costo più basso: occorre

valutare accuratamente la funzione obiettivo che si vuole adottare per un particolare problema dial-a-ride, perché da essa dipende il modo in cui le richieste sono servite in una soluzione ottima.

Una funzione obiettivo molto utilizzata e studiata in letteratura per i problemi dial-a-ride è il tempo di completamento, cioè il tempo complessivo che i server impiegano per servire tutte le richieste; è una funzione obiettivo che interessa particolarmente i server, perché renderla minima significa che i server termineranno di lavorare il più presto possibile. Invece, quando fornire un buon servizio significa servire "presto" le richieste, è più opportuno scegliere un'altra funzione obiettivo, quale la latenza netta (essenzialmente il tempo medio impiegato per il servizio di una richiesta) o il massimo tempo di servizio (il tempo impiegato per servire la richiesta più "sfortunata").

Il presente lavoro si propone anzitutto di fornire un quadro generale degli studi riguardanti gli algoritmi online deterministici per i problemi dial-a-ride con un unico server. Le varianti di problemi dial-a-ride più studiate in letteratura sono quelle che utilizzano la funzione obiettivo del tempo di completamento. Infatti per la latenza netta e per il massimo tempo di servizio esiste un risultato negativo, che afferma che non esistono algoritmi online competitivi. Per aggirare questo scoglio sono state tentate diverse strade, tra cui quella di cambiare leggermente la funzione obiettivo. In questo modo si è giunti alla latenza come funzione obiettivo (la somma degli istanti di tempo in cui le richieste sono servite), che differisce dalla latenza netta per una costante che dipende soltanto dall'istanza di input; per la latenza esistono degli algoritmi competitivi, ma purtroppo un buon rapporto di competitività per un algoritmo non garantisce un buon comportamento dell'algoritmo stesso in molte applicazioni pratiche.

Data la grande rilevanza della funzione obiettivo della latenza netta, in questa tesi si tenta un altro approccio: si studiano gli algoritmi dotati di lookahead per i problemi dial-a-ride online con la funzione obiettivo della latenza netta. Un algoritmo online è dotato di lookahead se conosce una limitata porzione del futuro. In particolare si introducono due modelli di lookahead: nel primo modello l'algoritmo online conosce in anticipo le prossime k richieste che saranno rilasciate, nel secondo l'algoritmo online conosce tutte le richieste che saranno rilasciate entro Δ unità di tempo. Per analizzare gli algoritmi con lookahead si utilizza sia l'analisi di competitività classica, che mette a confronto tali algoritmi con l'ottimizzatore offline, sia l'analisi di comparatività, che mette a confronto due classi di algoritmi online, per esempio due classi di algoritmi dotati di un numero differente di richieste di lookahead.

Ancora una volta si trovano risultati negativi: dimostreremo che nessun algoritmo dotato di lookahead di k richieste è competitivo, mentre per gli

algoritmi con un lookahead in una finestra temporale di ampiezza Δ si prenderanno in considerazione spazi metrici limitati di diametro D, dimostrando che non esistono algoritmi competitivi in nessuno spazio metrico quando $\Delta < 2D$. Per di più, negli spazi metrici isomorfi ad un aperto di $\mathbb{R}^n$ con $n \geq 2$ nessun algoritmo dotato di lookahead è competitivo, per quanto grande sia Δ. Questi risultati sono dimostrati per il problema del riparatore viaggiatore online; poiché esso è un caso particolare del problema dial-a-ride con un unico server più generale, gli stessi risultati sono validi per tale problema.

Poiché l'analisi di competitività non ci consente di apprezzare il vantaggio offerto dal lookahead, tenteremo di confrontare direttamente le prestazioni di diverse classi di algoritmi dotati di lookahead tramite l'analisi di comparatività, ma questo strumento si rivelerà inadeguato ai nostri scopi. L'analisi sperimentale delle prestazioni di alcuni algoritmi con lookahead, invece, ci permette sia di apprezzare l'utilità del lookahead, sia di selezionare fra gli algoritmi proposti quello che, nella pratica, sembra comportarsi meglio degli altri.

Il presente lavoro è organizzato nel seguente modo. Nel Capitolo 1 sono presentati i problemi dial-a-ride; specificando gli elementi che intervengono nella definizione di un problema dial-a-ride se ne presentano le differenti varianti, con particolare enfasi a quelle di maggior interesse teorico e applicativo. Il Capitolo 2 contiene un'introduzione ai problemi online: si definisce l'analisi di competitività e se ne presentano i limiti e gli approcci alternativi. Nel Capitolo 3 si presentano i principali risultati noti in letteratura riguardanti l'analisi deterministica online dei problemi dial-a-ride. Nel Capitolo 4 si studia, tramite l'analisi di competitività e l'analisi di comparatività, il vantaggio offerto dal lookahead agli algoritmi online per il problema del riparatore viaggiatore con la funzione obiettivo della latenza netta; al termine del capitolo si considerano brevemente alcune possibili direzioni della ricerca futura, e si estendono alla funzione obiettivo del massimo tempo di servizio alcuni dei risultati presentati nel capitolo. Il Capitolo 5 contiene uno studio sperimentale delle prestazioni degli algoritmi dotati di lookahead per il problema del riparatore viaggiatore online con la funzione obiettivo della latenza netta. In appendice è riportata una descrizione sintetica del programma di simulazione che è stato progettato per condurre gli esperimenti.

I risultati originali di questa tesi sono quelli contenuti nei Capitoli 4 e 5.

Capitolo 1

I problemi dial-a-ride e il problema del riparatore viaggiatore

In questo capitolo definiamo il problema del riparatore viaggiatore, inquadrandolo nel contesto più generale dei problemi dial-a-ride. Il problema del riparatore viaggiatore, infatti, è un particolare problema dial-a-ride. Inoltre esiste un'ampia classe di problemi dial-a-ride tale che, per ogni problema $\mathcal{P}$ della classe, il problema del riparatore viaggiatore può essere considerato un caso particolare di $\mathcal{P}$. Il problema del riparatore viaggiatore quindi è tra i più "facili" problemi di un'ampia classe di problemi dial-a-ride. Questa considerazione ci permetterà di estendere la validità di molti dei risultati che troveremo per il problema del riparatore viaggiatore ad altri problemi dial-a-ride. Presentiamo una panoramica sui problemi dial-a-ride.

1.1 Che cos'è un problema dial-a-ride

Risolvere un problema di tipo dial-a-ride significa trovare il modo "migliore" in cui un insieme di server riesce a soddisfare una sequenza di richieste.

Un *server* è un punto che si muove su uno spazio metrico fissato, con una velocità non superiore ad un limite massimo. Per soddisfare una *richiesta* (o *corsa*), il server deve visitare, nell'ordine, un punto S (*sorgente*) e un punto D (*destinazione*) dello spazio metrico; dopo aver visitato S e prima di aver visitato D diremo che il server sta trasportando un *oggetto* da S a D.

In genere una richiesta porta con sé alcuni vincoli temporali; il più comune è il *tempo di rilascio*, che rappresenta l'istante di tempo a partire dal quale la richiesta può essere soddisfatta. Il tempo di rilascio può avere ulteriori

significati. Ad esempio, nel prossimo capitolo vedremo che, in un contesto online, il tempo di rilascio di una richiesta σ_i rappresenta l'istante di tempo a partire dal quale è nota l'esistenza di σ_i.

Ogni server ha una *capacità* (finita o infinita) che rappresenta il numero massimo di oggetti che può trasportare in un qualsiasi istante di tempo. In alcuni casi un server può decidere di abbandonare, nel punto in cui il server si trova attualmente, uno degli oggetti che sta trasportando; in seguito l'oggetto dovrà essere nuovamente prelevato da uno qualsiasi dei server, per portare a termine la corsa interrotta.

I problemi dial-a-ride permettono di modellare un gran numero di problemi di grande importanza pratica. L'esempio più classico, quello che ha dato origine al nome *dial-a-ride* (chiama-una-corsa), è il servizio di radio-taxi: una compagnia gestisce un insieme di taxi che si muovono per le strade della città. Quando un cliente telefona alla centrale operativa, richiede una corsa: vuole essere raccolto da un taxi e trasportato in un (altro) punto della città. Per poter organizzare un buon servizio, la centrale operativa può risolvere un problema dial-a-ride in cui ogni taxi è rappresentato da un server che si muove nel grafo delle strade della città. Ogni server ha capacità unitaria perché un taxi può trasportare un solo cliente alla volta. Ad ogni chiamata corrisponde una richiesta, rilasciata nell'istante stesso in cui la chiamata arriva alla centrale.

In molti casi un problema dial-a-ride porta con sé, come casi particolari, altri problemi teorici noti ed importanti. Ad esempio quando la sorgente e la destinazione di ogni corsa coincidono, il servizio di una richiesta comporta semplicemente la visita del punto in cui la richiesta è rilasciata. In questo caso risolvere il problema dial-a-ride significa trovare il modo "migliore" per visitare un insieme di punti: specificando meglio le altre ipotesi ci si riconduce facilmente a problemi quali il problema del commesso viaggiatore online (OlTSP) o quello del riparatore viaggiatore online (OlTRP).

1.2 Classificazione dei problemi dial-a-ride

Per poter fare riferimento a uno specifico problema dial-a-ride occorre chiarire la natura degli oggetti che intervengono nella sua definizione: lo spazio metrico, i server, le richieste (con i relativi vincoli temporali), e la funzione obiettivo da minimizzare, che rappresenta la misura della qualità della soluzione ottenuta (si ricordi che stiamo cercando il modo "migliore" per servire tutte le richieste).

In [20] de Paepe introduce una notazione che consente di definire, in forma sintetica e rigorosa, una vasta varietà di problemi dial-a-ride; successivamen-

te fa uso della notazione per studiare la classe di complessità computazionale di ogni problema. Dopo aver accorpato problemi "equivalenti", ossia problemi che si possono immediatamente ridurre l'uno nell'altro e che, in sostanza, costituiscono uno stesso problema, de Paepe analizza i rimanenti 7930 problemi "distinti": soltanto per una piccola parte di essi (180) si può dimostrare che sono risolubili in tempo polinomiale, di 77 problemi attualmente non si conosce la classe di complessità, e tutti gli altri sono NP-hard.

Nel seguito ripercorreremo essenzialmente la classificanzione di de Paepe, adattandola in alcuni punti, ma non introdurremo un formalismo per denotare i problemi. Metteremo in risalto alcuni dei problemi dial-a-ride più rilevanti nella teoria e nella pratica, e i problemi di cui ci occuperemo in questa tesi.

Spesso incontreremo problemi con un diverso grado di generalità: i problemi meno generali possono essere visti come casi particolari di quelli più generali. Alcune delle informazioni implicite nella definizione del problema meno generale devono essere fornite, come parte dell'istanza di input, al problema più generale. Chiaramente, un problema è tanto più difficile da analizzare e da risolvere quanto più è generale.

1.2.1 Lo spazio metrico

Un problema dial-a-ride è definito su uno spazio metrico $M = (X, d)$, dove X è un insieme di punti e $d : X \times X \to \mathbb{R}_0^+$ è la distanza definita su X. d è simmetrica e gode della disuguaglianza triangolare, ed inoltre $d(P, P) = 0 \; \forall P \in X$. Chiediamo che M si comporti bene per un problema dial-a-ride: intuitivamente, occorre poter immaginare il movimento di un server su di esso. Pertanto ci limiteremo ad accettare due tipi di spazi metrici:

Spazi metrici continui. Se $M = (X, d)$ è uno spazio metrico continuo, allora prendendo due qualsiasi punti $P, Q \in X$ esiste un arco di curva continuo $\gamma(P, Q)$, composto da tutti punti di X, che congiunge P a Q ed ha lunghezza $d(P, Q)$. In altri termini lo spazio metrico è continuo se il percorso più breve tra due punti P e Q attraversa esclusivamente punti di X.

Spazi metrici discreti. Sia $M = (X, d)$ uno spazio metrico. Diremo che M è *discreto* se, per ogni coppia di punti $P, Q \in X$ esiste un insieme finito, eventualmente vuoto, $\{P_1, \ldots, P_n\} \subseteq X$ tale che, indicando $P_0 := P$ e $P_{n+1} := Q$,

- $d(P, Q) = \sum_{i=0}^{n} d(P_i, P_{i+1})$

- $\forall i \in \{0, \ldots, n\}$, una delle seguenti affermazioni è vera:
  - esiste un arco di curva continuo $\gamma(P_i, P_{i+1})$, avente lunghezza $d(P_i, P_{i+1})$ e composto interamente da punti di X, oppure
  - non esiste un punto $P^* \in X$ tale che $d(P_i, P_{i+1}) = d(P_i, P^*) + d(P^*, P_{i+1})$;

inoltre chiediamo che esistano almeno due punti P e Q tali che nessun un arco continuo che li unisce ha lunghezza $d(P, Q)$.

In altre parole in uno spazio metrico discreto esistono coppie di punti (P, Q) tali che, per spostarsi da P a Q tramite il percorso più breve, non è possibile muoversi rimanendo all'interno dello spazio metrico, ma bisogna compiere un "salto" discreto. Possiamo allora supporre che P e Q siano collegati da un segmento di lunghezza $d(P, Q)$ esterno allo spazio metrico, che costituisce una sorta di tunnel fra i due punti; una volta imboccato il tunnel, il server si muove dentro di esso e non lo può abbandonare. Può però fermarsi e invertire il verso di marcia. In altri contesti è più utile supporre che il server, una volta partito da P, non possa fermarsi o tornare indietro prima che sia giunto in Q. Per esempio un veicolo (server) che viaggia su di una una autostrada non può invertire il senso di marcia prima che sia giunto all'uscita successiva. In altri contesti ancora è utile discretizzare anche il tempo. Per esempio se lo spazio metrico è un grafo in cui tutti gli archi hanno lunghezza unitaria, si può supporre che ogni istante di tempo sia un numero naturale; se nell'istante h il server si trova su un vertice, allora nell'istante $h+1$ esso si può trovare sullo stesso vertice oppure su un vertice adiacente; ogni richiesta è rilasciata in un istante di tempo intero.

Indichiamo con $\mathcal{M}$ l'insieme degli spazi metrici che siano (i) continui oppure discreti, e (ii) non banali, ossia dotati di almeno due punti che distano più di zero. Un esempio di spazio metrico che non si trova in $\mathcal{M}$ è $\mathbb{Q}$ (dotato della della distanza euclidea): infatti $\mathbb{Q}$ non è continuo, però è denso e quindi non è neanche discreto.

Un problema dial-a-ride è definito o su un fissato spazio metrico $M = (X, d) \in \mathcal{M}$, oppure sul generico spazio metrico di $\mathcal{M}$; in quest'ultimo caso, occorre specificare in ciascuna istanza di input qual è lo spazio metrico effettivamente utilizzato. Occorre definire o fornire in input anche un punto $O \in X$ che chiameremo *origine*, in cui si trovano tutti i server nell'istante 0; e inoltre, se lo spazio metrico è discreto, occorre definire come avvengono i "salti" tra due punti non altrimenti collegati da punti di X.

Spazi metrici particolarmente importanti per i problemi dial-a-ride sono l'albero, il grafo, il segmento, la semiretta, la retta, e lo spazio euclideo $\mathbb{R}^n$.

Con un piccolo abuso di notazione nel seguito della tesi faremo riferimento allo *spazio metrico* $\mathbb{R}^n$ o a un suo sottoinsieme, dotato della consueta distanza euclidea, semplicemente con il simbolo dell'insieme (ad esempio: $\mathbb{R}^2$, $\mathbb{R}_0^+$, $[-1, 1]$), invece di indicare più correttamente la coppia (insieme, distanza).

In questa tesi studieremo problemi dial-a-ride definiti su spazi metrici generici (astratti), oppure su $\mathbb{R}^n$ e sui suoi sottoinsiemi.

1.2.2 I server

Le proprietà dell'insieme dei server possono variare parecchio a seconda del problema; vediamo quali sono le principali.

Numero. In generale il numero dei server è un naturale $k \in \mathbb{N}^+$. Un caso particolarmente importante è quello in cui c'è un unico server ($k = 1$). Nel corso della tesi ci occuperemo esclusivamente di questo caso.

Capacità. In generale, ciascuno dei server ha una propria capacità, finita o infinita. Un caso particolare è quello in cui tutti i server hanno la stessa capacità. Un caso ancora più particolare, ma importante, è quello in cui i server hanno tutti capacità unitaria.

Un taxi è ben rappresentato da un server a capacità unitaria; un taxi collettivo, un furgone portapacchi o un ascensore si possono modellare con server a capacità finita; infine, un furgone portalettere può essere modellato con un server a capacità infinita.

Velocità. Nel caso più generale ogni server ha una propria velocità (massima), che è una funzione (presumibilmente non crescente) del numero di oggetti che attualmente trasporta. È molto più comune supporre che ogni server abbia una velocità fissata, indipendente dal numero di oggetti che trasporta; spesso questa velocità è comune a tutti i server, e in tal caso si può sempre supporre che essa sia unitaria (a meno di cambiamenti di scala nello spazio metrico e nell'asse dei tempi). In questa tesi supporremo sempre che il nostro unico server si muova a velocità unitaria.

1.2.3 Le richieste

Per caratterizzare una richiesta occorre fornire alcuni dati che dipendono dal problema.

Definiamo, per il momento, una richiesta come una terna $\sigma_i = (t_i, S_i, D_i) \in \mathbb{R}_0^+ \times M \times M$ dove $M \in \mathcal{M}$ è lo spazio metrico su cui è

definito il problema, t_i è l'istante di rilascio della richiesta, S_i è la sorgente e D_i è la destinazione. Come abbiamo detto, per servire σ_i il server deve trasportare un oggetto da S_i a D_i, prelevandolo non prima dell'istante t_i.

L'istanza di un problema dial-a-ride comprende sempre una stringa di richieste $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$, e spesso si limita ad essa. Con un piccolo abuso di notazione indicheremo con σ anche l'*insieme* delle richieste, il che ci permetterà di scrivere, ad esempio, "$\sigma_i \in \sigma$".

Tra i problemi particolari, troviamo quelli in cui tutte le sorgenti coincidono con un unico punto, diciamo l'origine: $S_1 = \ldots = S_n = O$ (si pensi ad esempio ad un servizio di spedizione posta, in cui i veicoli si caricano all'ufficio postale); quelli in cui tutte le destinazioni coincidono con l'origine (servizio di raccolta della spazzatura); e quelli in cui, per ogni richiesta, la sorgente coincide con la destinazione. In quest'ultimo caso non c'è nulla da trasportare, il server deve semplicemente visitare i punti in cui si trovano le richieste. Per caratterizzare una richiesta di questi problemi particolari è sufficiente una coppia: $\sigma_i = (t_i, S_i)$ oppure $\sigma_i = (t_i, D_i)$.

È possibile introdurre altri vincoli temporali sulle richieste: ad esempio l'istante di scadenza, prima del quale la richiesta deve essere servita; oppure dei vincoli aggiuntivi sull'istante in cui la richiesta è prelevata dalla sorgente o sull'istante in cui è depositata sulla destinazione. Si noti che un problema dial-a-ride con vincoli di questo tipo potrebbe ammettere soluzione non per tutte le istanze di input. È possibile peraltro introdurre dei vincoli di *precedenza* tra le richieste. Non faremo più riferimento a vincoli aggiuntivi.

Infine, occorre specificare la possibilità o meno di prelazionare le richieste. Se il server può *prelazionare* le richieste, allora può depositare uno degli oggetti che sta trasportando in un qualsiasi punto dello spazio metrico diverso dalla destinazione. L'oggetto dovrà essere raccolto, in seguito, da qualche server per portare a compimento la richiesta relativa. Se la prelazione non è ammessa, invece, ogni volta che un server carica un oggetto non può depositarlo fino a quando non si trova nella destinazione della relativa richiesta.

1.2.4 La funzione obiettivo

Nel contesto dei problemi di minimizzazione, la *funzione obiettivo* (o *funzione di costo*) è una funzione che associa ad ogni soluzione del problema un "costo" reale e non-negativo. Risolvere il problema (in modo ottimo) significa trovare una soluzione di costo minimo.

Una funzione di costo ampiamente utilizzata per i problemi dial-a-ride è il tempo di completamento, l'ultimo istante di tempo in cui è servita una richiesta dell'istanza:

Definizione 1.1 (Tempo di completamento) *Dato un problema dial-a-ride, siano $\sigma = \sigma_1 \dots \sigma_n$ un'istanza di input, s una sua soluzione, e sia τ_i il tempo di servizio in s della richiesta σ_i. Allora il* tempo di completamento *della soluzione s è la quantità*

$$C := \max_i \{\tau_i\} .$$

Dal punto di vista del server, il tempo di completamento è una funzione obiettivo "egoista": renderla minima significa far sì che esso termini il proprio lavoro il più presto possibile.

In un atteggiamento più altruista il server dovrebbe servire ogni richiesta nel più breve tempo possibile a partire dal momento in cui essa è rilasciata. Una funzione che misura il ritardo complessivo con cui sono servite le richieste è la latenza netta:

Definizione 1.2 (Latenza netta) *Dato un problema dial-a-ride, siano $\sigma = \sigma_1 \dots \sigma_n$ un'istanza di input, s una sua soluzione, e siano t_i e τ_i rispettivamente il tempo di rilascio e il tempo di servizio in s della richiesta σ_i. Allora la* latenza netta *della soluzione s è la quantità*

$$L_N := \sum_{i=1}^{n} (\tau_i - t_i) .$$

Una funzione obiettivo per certi versi simile alla latenza netta è la latenza, la somma dei tempi di servizio delle richieste:

Definizione 1.3 (Latenza) *Dato un problema dial-a-ride, siano $\sigma = \sigma_1 \dots \sigma_n$ un'istanza di input, s una sua soluzione, e sia τ_i il tempo di servizio in s della richiesta σ_i. Allora la* latenza *della soluzione s è la quantità*

$$L := \sum_{i=1}^{n} \tau_i .$$

Per una fissata istanza di input $\sigma = \sigma_1 \dots \sigma_n$, si ha che $L_N = L - \sum_{i=1}^{n} t_i$; poiché il termine $\sum_{i=1}^{n} t_i$ non dipende dalla soluzione ma soltanto dall'istanza σ del problema, minimizzare la latenza netta significa anche minimizzare la latenza, e viceversa. Tuttavia la latenza ha un significato meno intuitivo rispetto a quello della latenza netta.

Più in generale è possibile definire una versione pesata rispettivamente della latenza netta e della latenza, ovvero $L_N := \sum_{i=1}^{n} (w_i(\tau_i - t_i))$ e $L := \sum_{i=1}^{n} (w_i \tau_i)$, in cui w_i è il *peso* della richiesta σ_i. Nel seguito supporremo sempre che ogni richiesta abbia peso unitario ($w_i = 1$), riconducendoci alle definizioni precedenti.

La latenza netta si presta bene a rappresentare l'insoddisfazione complessiva dei clienti per il ritardo del servizio delle rispettive richieste; minimizzando la latenza netta si ottiene, nella media, un buon servizio (anzi, il miglior servizio possibile); tuttavia alcune richieste potrebbero essere servite con grande ritardo.

Un'alternativa è quella di minimizzare l'insoddisfazione del cliente più "sfortunato", quello la cui richiesta impiega più tempo per essere servita. Per far ciò è sufficiente assumere come funzione obiettivo da minimizzare il massimo tempo di servizio (o massimo tempo di flusso):

Definizione 1.4 (Massimo tempo di servizio) *Dato un problema dial-a-ride, siano $\sigma = \sigma_1 \dots \sigma_n$ un'istanza di input, s una sua soluzione, e siano t_i e τ_i rispettivamente il tempo di rilascio e il tempo di servizio in s della richiesta σ_i. Allora il* massimo tempo di servizio *della soluzione s è la quantità*

$$F_{\max} := \max_i \{\tau_i - t_i\} .$$

In questo caso il servizio è più scadente nella media, ma nessun cliente può essere "troppo" insoddisfatto.

1.3 Alcuni problemi dial-a-ride rilevanti. Il problema del riparatore viaggiatore

Ora che abbiamo tutti gli elementi per definire completamente un problema dial-a-ride, diamo la definizione di alcuni problemi rilevanti nella teoria o nella pratica.

Il problema dei postini o dei furgoni portapacchi. Un ufficio postale dispone di k postini che devono rifornirsi di posta e consegnarla nelle abitazioni. Per rendere efficiente il servizio è possibile risolvere un problema dial-a-ride, definito su un opportuno spazio metrico (per esempio su $\mathbb{R}^2$ o sul grafo delle strade della città) con k server a capacità infinita (postini) o finita (furgoni portapacchi); la velocità dei server si può assumere costante; tutte le richieste hanno sorgente nell'origine; non è ammessa prelazione. La funzione obiettivo più naturale è il tempo di completamento.

Il servizio radio-taxi. Come abbiamo visto nella Sezione 1.1, un servizio radio-taxi è il più classico esempio di situazione che è possibile affrontare tramite un particolare problema dial-a-ride. Vogliamo qui aggiungere che una buona funzione obiettivo per tale problema è la latenza netta.

Gli ascensori. In un grattacielo si trovano k ascensori, ciascuno dei quali ha una capacità finita e una velocità che non dipende dal carico. Un buon modello è quello di un problema dial-a-ride generico in cui k server si muovono su un albero lineare, con un nodo per ogni piano. Funzione obiettivo: latenza netta, oppure massimo tempo di servizio.

Il problema del commesso viaggiatore. Il problema del commesso viaggiatore (Traveling Salesman Problem, TSP) è un problema di enorme interesse teorico e pratico, ampiamente studiato. Ne presentiamo la versione con tempi di rilascio, detta anche *online* (OlTSP).

L'OlTSP è il problema dial-a-ride con un server che si muove a velocità unitaria su un fissato spazio metrico; ogni richiesta ha un tempo di rilascio; la sorgente e la destinazione di ogni richiesta coincidono; la funzione obiettivo è il tempo di completamento.

Il TSP deve il suo nome ad una sua possibile utilizzazione da parte di un commesso viaggiatore che deve visitare le case dei potenziali clienti (ciascuna non prima di un certo orario, nella versione online) per proporre l'acquisto di spazzole. Il suo obiettivo è quello di finire le visite il più presto possibile (oltre naturalmente a quello di vendere il maggior numero possibile di spazzole).

Si noti che un gran numero di problemi dial-a-ride minimizzanti il tempo di completamento ammettono, come caso particolare, l'OlTSP. Questo implica che tutte le "difficoltà" che si incontrano nella risoluzione dell'OlTSP valgono per qualsiasi problema dial-a-ride più generale. Ad esempio è noto che il TSP su un grafo generico è un problema NP-hard: dunque, tutti i problemi dial-a-ride più generali sono NP-hard.

Infine osserviamo che nel TSP classico (detto anche TSP offline) non compaiono i tempi di rilascio delle richieste. Ogni istanza del TSP classico si può pensare come un'istanza dell'OlTSP in cui tutte le richieste hanno tempo di rilascio pari a 0.

Il problema del riparatore viaggiatore. Un'altro problema importante dal punto di vista teorico è il problema del riparatore viaggiatore online (Online Traveling Repairman Problem, OlTRP).

Così come il commesso viaggiatore, il riparatore viaggiatore deve visitare le abitazioni dei clienti, ma la sua intenzione è diversa: riparare una lavatrice guasta. I clienti sono ansiosi di ricevere la sua visita (al contrario di quanto avveniva nel caso del commesso viaggiatore), quindi per renderli soddisfatti il riparatore viaggiatore deve minimizzare una funzione obiettivo "altruista", per esempio la latenza netta, oppure il massimo tempo di servizio.

Più formalmente il problema del riparatore viaggiatore online è il proble-

ma dial-a-ride con un server che si muove a velocità unitaria in un fissato spazio metrico. Ogni richiesta ha un tempo di rilascio; la sorgente e la destinazione di ogni richiesta coincidono. La funzione obiettivo è la latenza netta (o equivalentemente la latenza). In effetti si possono considerare varianti del problema del riparatore viaggiatore con differenti funzioni obiettivo, ad esempio il massimo tempo di completamento. Inoltre in alcuni contesti è necessario distinguere la latenza dalla latenza netta, come meglio vedremo in seguito. Per distinguere le varianti dell'OlTRP introduciamo la seguente notazione:

Simbolo	**Funzione obiettivo della variante del TRP indicata**
L-OlTRP	Latenza
L_N-OlTRP	Latenza netta
$F_{\max}$-OlTRP	Massimo tempo di servizio

Ciascuna variante dell'OlTRP è utile perché è un caso particolare di un'ampia classe di problemi dial-a-ride, in cui si vuole minimizzare la latenza netta, la latenza oppure il massimo tempo di servizio. Al riguardo, valgono gli stessi ragionamenti che abbiamo fatto nel caso del TSP.

In letteratura è stata ampiamente studiata la versione senza i tempi di rilascio del L-OlTRP, che equivale al L-OlTRP in cui tutte le richieste sono rilasciate nell'istante 0. Questo problema, chiamato anche *Minimum Latency Problem* o MLP, ha un'altra suggestiva interpretazione. È dato un insieme di n isole; in una e una sola isola è nascosto un tesoro. L'isola i-esima contiene il tesoro con probabilità $p_i \in [0, 1]$; naturalmente $\sum_{i=1}^{n} p_i = 1$. Un cercatore di tesori desidera determinare il giro T che deve compiere per rendere minimo il tempo di ricerca atteso. Per trovare T, è sufficiente risolvere il MLP pesato, in cui la funzione obiettivo è la latenza pesata con le probabilità p_i.

Capitolo 2

I problemi online

2.1 Il modello online

Consideriamo un qualsiasi problema dial-a-ride, ad esempio il problema dei radio-taxi, e immaginiamo di applicarlo nella pratica. L'istanza di input del problema è costituita dalla sequenza delle richieste che giungono alla centrale operativa nell'arco di una giornata. Conoscendo l'intera sequenza possiamo risolvere il problema, e ottimizzare così gli spostamenti dei taxi nell'ambito dell'intera giornata. L'ottimizzazione naturalmente deve essere compiuta *prima* dell'inizio del servizio.

Ecco che sorge una difficoltà: al momento dell'ottimizzazione ancora non si conoscono tutte le richieste che arriveranno alla centrale. In altri termini, l'intera istanza di input non è nota fin dal principio, ma viene "rivelata" man mano che passa il tempo; ciononostante è opportuno iniziare il servizio di taxi fin dal principio, cercando di utilizzare al meglio le informazioni parziali sull'input di cui si dispone.

Una situazione analoga a quella che abbiamo descritto si presenta molto spesso nella pratica: ci si trova a dover risolvere un problema la cui istanza di input è rivelata gradualmente, ma non è opportuno – oppure non è possibile – attendere di conoscere l'intera istanza prima di fornire la risposta. Spesso non è possibile sapere quanto è lunga la sequenza di input: per esempio in un problema dial-a-ride ogni richiesta che si riceve potrebbe essere l'ultima, ma nel momento in cui la si riceve non è dato saperlo.

Un problema in cui l'istanza di input è rivelata man mano che scorre il tempo è detto *problema online*. Il concetto dello "scorrere del tempo" giace sotto ogni problema online, ma spesso il modello matematico del problema risulta più semplice se si rappresenta il tempo in maniera più astratta rispetto alla consueta rappresentazione del tempo fisico. Ad esempio in alcuni

problemi il tempo è discretizzato, e le richieste possono arrivare soltanto negli istanti discreti. In altri problemi occorre soddisfare una sequenza di richieste, una per volta: ogni volta che una richiesta è soddisfatta, ne è presentata un'altra. Si pensi al problema del paging, o a quello dello scheduling del processore: è naturale supporre che soltanto quando una richiesta (page-fault o scheduling di un processo) è stata soddisfatta, giunge una nuova richiesta.

Nel caso dello scheduling, per di più, l'ordine in cui arrivano le richieste dipende dall'ordine in cui esse vengono servite (infatti, un processo può attivarne degli altri). Il paging è un esempio di problema *intrinsecamente online*, per il quale non è possibile fornire l'istanza di input separatamente da una sua soluzione, ovvero dalla strategia impiegata per risolvere il problema.

Quando si ha a che fare con problemi dial-a-ride online è naturale supporre che una richiesta σ_i sia nota a partire dal suo istante di rilascio t_i. Come avevamo accennato nella Sezione 1.1, abbiamo qui esteso la semantica del tempo di rilascio di una richiesta.

Un *algoritmo online* A è un algoritmo che risolve un problema online utilizzando un input parziale, che va completandosi man mano che scorre il tempo; più precisamente, A associa ad ogni istanza di input una soluzione *causale*, il cui comportamento "fino ad un certo istante" non dipende da come sarà fatto l'ingresso "dopo quell'istante". Si consideri un algoritmo online A per un problema dial-a-ride. La sua soluzione è il tragitto che i server devono compiere per servire le richieste; la natura online di A aggiunge il vincolo che il tragitto compiuto nell'intervallo $[0, t]$ non dipende dalle richieste rilasciate dopo l'istante t. Se A è deterministico, allora il tragitto in $[0, t]$ è funzione delle richieste rilasciate in $[0, t]$; se A è randomizzato allora il tragitto può dipendere anche da scelte casuali compiute da A.

2.2 L'analisi di competitività

Un algoritmo online non riesce quasi mai a trovare una soluzione ottima per un'istanza di un problema. Diventa estremamente importante costruire degli strumenti che ci permettano di misurare le prestazioni degli algoritmi online; l'obiettivo ultimo è quello di poter scegliere, tra tutti gli algoritmi online per un dato problema, il "migliore" – ma è più opportuno dire il "preferito", poiché vedremo presto che non esiste un unico modo per valutare la qualità di un algoritmo online.

Lo strumento di gran lunga più utilizato è l'analisi di competitività, introdotta formalmente da Sleator e Tarjan nel 1985 [23] ma utilizzata già in precedenza per l'analisi di alcuni problemi online, ad esempio lo scheduling multiprocessore [10] e il bin packing [12].

L'analisi di competitività è stata definita sia per gli algoritmi deterministici, sia per quelli randomizzati. Faremo riferimento soltanto ai primi.

2.2.1 Definizioni

Consideriamo un problema online di minimizzazione (quanto diremo vale anche per problemi di massimizzazione, con le dovute inversioni di segno); sia $\mathcal{I}$ l'insieme delle istanze di input. Per un qualsiasi algoritmo deterministico A che risolve il problema, indichiamo con $A(\sigma)$ il valore della funzione obiettivo calcolata sulla soluzione fornita da A, quando in input è fornita l'istanza $\sigma \in \mathcal{I}$. Indicheremo con OPT l'algoritmo ottimo offline, ossia l'algoritmo offline che risolve in maniera ottima il problema.

Definizione 2.1 (Algoritmo ρ-competitivo) *Sia $\rho \in \mathbb{R}^+$. Diremo che un algoritmo (online) A per un problema è* ρ-competitivo *se, per ogni istanza $\sigma \in \mathcal{I}$, risulta:*

$$A(\sigma) \leq \rho \cdot OPT(\sigma)\,. \tag{2.1}$$

Dalla definizione risulta che, se A è ρ-competitivo, allora $\rho \geq 1$, perché A non può comportarsi meglio di OPT. Alcuni autori consentono di aggiungere una costante al secondo membro della (2.1), per tener conto di una differenza di stato iniziale tra l'algoritmo ottimo offline e A.

Abbiamo posto tra parentesi il termine "online" perché non è necessario ai fini della definizione, che risulta essere più generale.

Definizione 2.2 (Rapporto di competitività) *Il* rapporto di competitività *di un algoritmo (online) A per un problema è la quantità:*

$$\inf\{\rho \in \mathbb{R}^+ | A \text{ è } \rho-\text{competitivo}\}\,,$$

se esiste; invece, se non esiste $\rho \in \mathbb{R}^+$ per cui A è ρ-competitivo, diremo che A non è competitivo.

Da queste defininzioni si evince che il rapporto di competitività è una misura delle prestazioni dell'algoritmo nel caso peggiore, perché il confronto con le prestazioni dell'algoritmo ottimo offline deve reggere per *tutte* le istanze di input; tra le quali, appunto, la peggiore.

Osserviamo che il rapporto di competitività di un algoritmo A si può anche scrivere come:

$$\sup_{\sigma \in \mathcal{I}} \frac{A(\sigma)}{OPT(\sigma)}\,.$$

Definizione 2.3 (Lower bound di un problema) *Diremo che $l \in \mathbb{R}^+$ è un* lower bound di competitività *di un problema online se ogni algoritmo online A per il problema ha un rapporto di competitività $\rho_A \geq l$, oppure A non è competitivo.*

Diremo che un problema possiede lower bound di competitività infinito *se non esistono algoritmi online competitivi per il problema.*

Data una classe di algoritmi (online) $\mathcal{A}$, è naturale chiedersi quale sia il miglior algoritmo di $\mathcal{A}$ e quali siano le sue prestazioni. L'analisi di competitività ci dice che il miglior algoritmo di $\mathcal{A}$ è quello che ha il rapporto di competitività più basso. Possiamo allora definire il rapporto di competitività di una classe di algoritmi:

Definizione 2.4 (Rapporto di competitività di una classe) *Sia $\mathcal{A}$ una classe di algoritmi online per un problema; chiameremo* rapporto di competitività della classe $\mathcal{A}$ *l'estremo inferiore dell'insieme dei rapporti di competitività degli algoritmi di $\mathcal{A}$; ovvero la quantità:*

$$R(\mathcal{A}) := \inf_{A \in \mathcal{A}} \sup_{\sigma \in \mathcal{I}} \frac{A(\sigma)}{OPT(\sigma)} .$$

2.2.2 L'analisi di competitività come una partita

L'analisi di competitività di un algoritmo A può essere interpretata come una partita disputata tra A e un avversario malevolo che ne conosce il comportamento. L'avversario costruisce un'istanza di input che mette in difficoltà A ma non svantaggia troppo l'algoritmo ottimo offline, in modo da massimizzare il rapporto $\frac{A(\sigma)}{OPT(\sigma)}$.

È anche possibile immaginare che l'algoritmo ottimo offline e l'avversario costituiscano complessivamente un'unica entità, chiamata *avversario offline* oppure *giocatore offline*, che conoscendo il funzionamento di A genera delle richieste molto sfavorevoli per A, ma che esso stesso riesce a servire facilmente.

Questa interpretazione è suggestiva, rende più snello il linguaggio nelle dimostrazioni e più intuitive alcune considerazioni, come quelle che ci apprestiamo a compiere che riguardano i limiti dell'analisi di competitività.

2.2.3 I limiti dell'analisi di competitività

L'analisi di competitività è stata criticata da molti autori perché presenta numerosi limiti; vogliamo qui elencare i principali.

Essendo una misura della prestazione di un algoritmo nel caso peggiore, il rapporto di competitività rischia di dare un'informazione troppo pessimista sulla reale "bontà" dell'algoritmo. Sono noti algoritmi online che nella pratica si comportano molto bene, ma con pessimi rapporti di competitività oppure del tutto non competitivi.

Possiamo reinterpretare questa critica utilizzando il linguaggio della partita contro un avversario offline: l'avversario appare troppo potente rispetto all'algoritmo online. Le sue grandi potenzialità derivano da una conoscenza completa di quanto sta accadendo: di tutte le richieste che saranno rilasciate nel futuro e del comportamento di A in ogni istante. Al contrario A può solo guardare al passato, e in generale non sa nulla di ciò che sta compiendo l'avversario. Si può concludere che esiste un ampio gap tra il regime informativo di A e quello dell'avversario; nella prossima sezione vedremo come si può ovviare all'eccessivo potere dell'avversario, ricorrendo ad alcuni stratagemmi con cui essenzialmente si tenta di ridurre questo gap informativo.

Utilizzare degli approcci alternativi all'analisi di competitività classica, eventualmente ricorrendo a stratagemmi, in alcuni casi è una necessità. In questa tesi ci occuperemo di problemi per cui non esistono algoritmi online competitivi.

Un altro limite dell'analisi di competitività è il seguente. Per potersi comportare in maniera accettabile con *tutte* le istanze di input, spesso un algoritmo competitivo si comporta in maniera "paranoica": ad esempio può attendere molto tempo prima di prendere una decisione che potrebbe in qualche modo comprometterlo, "sprecando" del tempo prezioso; la maggior parte delle istanze ne sarà svantaggiata, ma nel caso peggiore andrà un po' meglio. Per avere una migliore conoscenza della qualità di un algoritmo è utile affiancare all'analisi di competitività una valutazione sperimentale delle sue prestazioni. Supponiamo di dover decidere quale algoritmo online adottare per risolvere un problema. Possiamo partire da un insieme di algoritmi candidati; studiarne il loro rapporto di competitività (o alcuni upper bound al loro rapporto di competitività), e quindi selezionare un sottoinsieme di algoritmi con rapporto di competitività abbastanza basso. Infine si sottopongono gli algoritmi selezionati ad un'analisi sperimentale delle prestazioni, in base alla quale si decide l'algoritmo da adottare.

Si noti che l'analisi di competitività è ortogonale all'analisi della complessità computazionale degli algoritmi. Questo significa che, per un dato problema, gli algoritmi con i rapporti di competitività più bassi possono essere non utilizzabili nella pratica, per via del loro eccessivo uso delle risorse di calcolo. Se si vuole tenere conto anche della complessità computazionale è sufficiente cercare, tra tutti gli algoritmi online appartenenti ad una certa classe di complessità (ad esempio P), quelli con i migliori rapporti di

competitività.

2.3 Metodi di analisi alternativi all'analisi di competitività deterministica

Come abbiamo visto, uno dei principali limiti dell'analisi di competitività è l'eccessivo potere di cui dispone l'avversario offline a causa della sua onniscienza. Studiamo alcuni stratagemmi che ci permettano di aggirare questo ostacolo.

2.3.1 Indebolire l'avversario

Una prima idea è quella di mantenere l'avversaro onnisciente, ma di non permettergli di utilizzare le proprie conoscenze in modo tale da svantaggiare troppo l'algoritmo online. L'idea nasce dalla seguente osservazione: l'avversario, conoscendo le richieste che saranno rilasciate in futuro, può comportarsi in maniera innaturale, ovvero molto diversa dal modo in cui ci aspetteremmo comportarsi un algoritmo (online) sulla base delle richieste rilasciate nel passato. Ad esempio l'avversario può decidere di trascurare momentaneamente il servizio di tutte richieste già presentate, perché sa che nel futuro verrà rilasciata una richiesta particolarmente "vantaggiosa", e si prepara a servirla al meglio.

Una possibile contromossa è quella di imporre che il comportamento dell'avversario in un istante t non si discosti eccessivamente da un comportamento che si ritiene "ragionevole" sulla base delle richieste presentate fino all'istante t. In questo modo l'avversario continua ad essere onnisciente, ma non può sfruttare le sue conoscenze in modo eccessivamente "scorretto" nei confronti dell'algoritmo online.

Un esempio di avversario indebolito utilizzato nei problemi dial-a-ride è l'*avversario leale*, che nell'istante di tempo t si può muovere soltanto all'interno del cono convesso dell'origine O e dei punti delle richieste che sono state rilasciate negli istanti di tempo precedenti t (si veda [5]).

Un altro modo di colmare parzialmente il gap informativo è quello di utilizzare un avversario che comunichi all'algoritmo delle informazioni aggiuntive sul proprio comportamento, ad esempio la propria posizione in determinati istanti di tempo. Si veda, per esempio, l'*avversario benevolo* definito in [6].

Ovviamente, l'uso di avversari indeboliti ha anche degli svantaggi. L'analisi di competitività classica fornisce un indicatore molto naturale della qualità dell'algoritmo che si sta analizzando, perché lo confronta con il migliore possibile tra *tutti* gli algoritmi, l'ottimo offline: è un indicatore chiaro

ed assoluto. Non è altrettanto naturale l'interpretazione del significato di un rapporto di competitività contro un avversario indebolito, in quanto l'avversario è un oggetto introdotto con un certo grado di arbitrarietà. In alcuni lavori si sono confrontate le prestazioni di un avversario indebolito con quelle di un algoritmo online dotato di poteri superiori a quelli dell'avversario; il significato del rapporto di competitività ottenuto in questo modo può essere arduo da interpretare.

2.3.2 Lo studio di algoritmi dotati di lookahead

Un algoritmo online è dotato di *lookahead* se all'istante t è a conoscenza di un sottoinsieme S_t dell'insieme σ di tutte le richieste rilasciate; S_t, oltre a contenere tutte le richieste rilasciate prima dell'istante t, può contenere alcune delle richieste che sarannno rilasciate dopo l'istante t.

Definire un *modello di lookahead* significa fornire la funzione che associa ad ogni coppia (σ, t) il corrispondente sottoinsieme di richieste note S_t.

Quando si misura il rapporto di competitività di algoritmi dotati di lookahead, in sostanza si cerca di ridurre il gap informativo rispetto all'avversario fornendo ulteriori informazioni all'algoritmo sfidante, piuttosto che togliendole all'avversario.

2.3.3 L'uso di funzioni obiettivo differenti

Se non esistono algoritmi competitivi per un dato problema, si può cercare di modificarlo leggermente cambiando la funzione obiettivo: occorre cercare una funzione obiettivo che abbia un significato simile a quella originale, e che porti alla definizione di un problema per cui esistono algoritmi online competitivi.

Ciò richiede qualche cautela, in quanto sostituendo la funzione obiettivo con una simile è possibile snaturare il significato del rapporto di competitività, come vedremo nella Sezione 3.2.4.

2.3.4 L'uso di algoritmi randomizzati

Nell'analisi di competitività di un algoritmo A, l'avversario offline è molto avvantaggiato dal fatto di poter prevedere esattamente come A si comporterà; se A non è deterministico ma *randomizzato*, ossia in grado di compiere scelte casuali durante la sua esecuzione, allora non è sufficiente che l'avversario conosca "come è fatto A" affinché sappia esattamente "quello che farà A". Per sapere quello che farà A, l'avversario deve conoscere anche i risultati di tutte le scelte casuali compiute da A.

Queste considerazioni spingono a studiare le prestazioni degli algoritmi randomizzati. Prima di tutto diamo la definizione di algoritmo randomizzato.

Definizione 2.5 (Algoritmo randomizzato) *Un* algoritmo randomizzato *A è una distribuzione di probabilità definita su un insieme di algoritmi deterministici.*

È necessario ridefinire anche il rapporto di competitività, e di conseguenza il concetto di avversario. Partiamo da quest'ultimo. In letteratura sono stati definiti diversi modelli di avversario di algoritmi randomizzati; il modello più usato è quello di avversario *oblivio*, che conosce soltanto la descrizione dell'algoritmo A (e non le scelte casuali che compirà). Un avversario più potente è quello *adattativo online*, in grado di costruire l'istanza di input online: all'istante t conosce, oltre al funzionamento di A, tutte le scelte casuali che A ha compiuto fino all'istante t. Infine l'avversario *adattativo offline*, al momento di scegliere l'istanza di input, ha una conoscenza completa sia di A che di tutte le scelte casuali che A compirà: questo è chiaramente il modello di avversario più potente in assoluto.

Scegliamo di lavorare con il modello dell'avversario oblivio e definiamo il rapporto di competitività.

Definizione 2.6 (Algoritmo randomizzato ρ-competitivo) *Sia A un algoritmo randomizzato distribuito sull'insieme di algoritmi deterministici $\{A_x\}$ per un problema di minimizzazione. Diremo che A è ρ-competitivo se vale la seguente relazione:*

$$\mathbf{E}_x[A_x(\sigma)] \leq \rho \cdot OPT(\sigma) \ \forall \sigma \in \mathcal{I} .$$

$\mathbf{E}_x[A_x(\sigma)]$ è il valore atteso della funzione obiettivo conseguita dall'algoritmo randomizzato avente in input l'istanza σ.

In questa tesi non faremo uso degli algoritmi randomizzati.

2.3.5 L'aumento di risorse

Alcuni problemi sono parametrizzati rispetto ad un valore, la *quantità di risorse disponibili*: in genere, tanto più alto è il parametro, tanto più facile è risolvere il problema in modo efficiente. Si pensi ad esempio al problema della paginazione, in cui la quantità di risorse disponibili è il numero di elementi della memoria veloce.

Nell'analisi di competitività di un algoritmo A, per facilitare l'algoritmo rispetto all'avversario è possibile concedergli una quantità maggiore di risorse. Cambiando punto di vista, si può affermare che si sta indebolendo l'avversario rispetto ad A concedendogli meno risorse: ci riconduciamo al caso dell'uso di avversari indeboliti.

2.3.6 Il confronto diretto tra algoritmi online

Quando nessun algoritmo online è competitivo rispetto all'ottimo offline, ci si domanda se, tra tutti gli algoritmi online, ci sia una classe di algoritmi "migliori" di tutti gli altri. Per rispondere a questa domanda sono necessari degli strumenti che consentano il confronto tra le prestazioni di *classi* di algoritmi.

Uno di questi strumenti è l'*analisi di comparatività*, introdotta da Koutsoupias e Papadimitriou [13].

Definizione 2.7 (Rapporto di comparatività) *Fissato un problema online $\mathcal{P}$, consideriamo due classi $\mathcal{A}$ e $\mathcal{B}$ di algoritmi per $\mathcal{P}$. Chiameremo* rapporto di comparatività *fra $\mathcal{A}$ e $\mathcal{B}$ la quantità:*

$$R(\mathcal{A}, \mathcal{B}) := \sup_{B \in \mathcal{B}} \inf_{A \in \mathcal{A}} \sup_{\sigma \in \mathcal{I}} \frac{A(\sigma)}{B(\sigma)},$$

se esiste; con $\mathcal{I}$ abbiamo indicato l'insieme delle istanze del problema.

Questa definizione si comprende bene interpretando l'analisi di comparatività come una partita tra la classe $\mathcal{A}$ e la classe $\mathcal{B}$. Lo scopo di $\mathcal{A}$ è quello di rendere più bassa possibile la quantità $\frac{A(\sigma)}{B(\sigma)}$, che dovrà pagare a $\mathcal{B}$ al termine del gioco. La partita si svolge nel modo seguente: $\mathcal{B}$ sceglie un algoritmo "campione" $B \in \mathcal{B}$. In risposta, $\mathcal{A}$ sceglie un proprio campione $A \in \mathcal{A}$. Infine $\mathcal{B}$ sceglie l'istanza $\sigma \in \mathcal{I}$ ed $\mathcal{A}$ paga $\frac{A(\sigma)}{B(\sigma)}$.

Si noti che l'analisi di comparatività è un'estensione dell'analisi di competitività classica, alla quale si riduce quando $\mathcal{B}$ coincide con l'insieme di tutti gli algoritmi (di cui il "campione" B è, ovviamente, l'ottimo offline). In questo caso dunque:

$$R(\mathcal{A}, \mathcal{B}) = R(\mathcal{A}).$$

Per trovare un lower bound a un rapporto di comparatività non è necessario conoscere l'algoritmo "campione" $B \in \mathcal{B}$ che è stato scelto secondo la definizione. Consideriamo un qualsiasi $B' \in \mathcal{B}$; risulta che:

$$\rho' = \inf_{A \in \mathcal{A}} \sup_{\sigma \in \mathcal{I}} \frac{A(\sigma)}{B'(\sigma)} \leq \sup_{B \in \mathcal{B}} \inf_{A \in \mathcal{A}} \sup_{\sigma \in \mathcal{I}} \frac{A(\sigma)}{B(\sigma)} = \rho;$$

e quindi ρ' è un lower bound al rapporto di comparatività ρ.

Questa osservazione risulta utile perché ci consente, in alcune circostanze, di trovare il rapporto di comparatività senza conoscere l'algoritmo campione B: ad esempio dimostrando che ρ' è contemporaneamente un lower bound e un upper bound per ρ; oppure quando ρ' è infinito.

Osserviamo poi che, se $\mathcal{A} \supseteq \mathcal{B}$, allora $R(\mathcal{A}, \mathcal{B}) \leq 1$. Infatti, qualunque sia l'algoritmo campione $B \in \mathcal{B}$, la classe $\mathcal{A}$ può scegliere, in risposta, il medesimo algoritmo B ($\in \mathcal{A}$).

Gli strumenti che, come l'analisi di comparatività, ci permettono di confrontare direttamente due classi di algoritmi, possono essere impiegati in diversi modi. Abbiamo detto che una loro applicazione naturale è la ricerca, tra tutti gli algoritmi online, di classi di algoritmi che si comportino particolarmente bene.

Un'altra possibilità è quella di considerare due classi di algoritmi di *natura* differente, che non siano entrambe costituite esclusivamente da algoritmi online, e confrontarne le prestazioni. Per esempio possiamo confrontare la classe $\mathcal{A}$ di tutti e soli gli algoritmi online con la classe $\mathcal{B}$ (più generale di $\mathcal{A}$) di tutti e soli gli algoritmi online dotati di lookahead.

Anche l'analisi di comparatività cerca di arginare la superiorità dell'avversario dovuta alla differenza di disponibilità di informazioni di cui godono gli algoritmi in gioco; osserviamo inoltre che è immediato ridefinire l'analisi di competitività contro un avversario indebolito in termini di analisi di comparatività, prendendo come $\mathcal{B}$ l'insieme di tutti e soli gli avversari indeboliti.

In generale, però, lo spirito del confronto diretto tra classi di algoritmi è profondamente diverso dallo spirito dell'analisi di competitività contro avversari indeboliti. Nel primo caso si vogliono confrontare le prestazioni di *differenti varianti di algoritmi online*, lasciando cadere definitivamente l'ambizione di una confronto con i "migliori" algoritmi offline – ambizione ancora presente, in qualche modo, quando si fa uso di avversari indeboliti.

Capitolo 3

Principali lower bound ed algoritmi noti per i problemi dial-a-ride

In questo capitolo esaminiamo i principali risultati che si sono ottenuti nell'analisi online di problemi dial-a-ride. In ogni sezione prendiamo in considerazione una funzione obiettivo. La funzione obiettivo più studiata in letteratura è il tempo di completamento, di cui ci occupiamo nella Sezione 3.1. Nelle Sezioni 3.2 e 3.3 ci occupiamo della latenza e della latenza netta rispettivamente: mentre per quest'ultima esiste soltanto un risultato negativo, per la latenza sono noti degli algoritmi competitivi, che comunque non raggiungono il lower bound attualmente noto. Per comprendere l'utilità di ulteriori indagini sulla latenza netta osserveremo che l'informazione fornita da un rapporto di competitività sulla latenza in molti casi non è adeguata. Infine nella Sezione 3.4 vediamo che, per la funzione obiettivo del massimo tempo di servizio, nonostante esista un risultato negativo generale (simile a quello relativo alla latenza netta), tuttavia è noto un algoritmo competitivo contro un particolare tipo di avversario.

Nel corso del capitolo, affrontiamo sempre problemi dial-a-ride con un singolo server che si muove a velocità unitaria nel proprio spazio metrico.

3.1 Il tempo di completamento

In questa sezione ci occupiamo delle varianti dei problemi dial-a-ride online più studiate, quelle in cui la funzione obiettivo da minimizzare è il tempo di completamento. In tale ambito è stata posta particolare attenzione al

problema del commesso viaggiatore online; il problema è stato affrontato estensivamente da Ausiello et al. in [4] e da Lipmann in [18].

Nella Sezione 3.1.1 affrontiamo il generico problema dial-a-ride con minimizzazione del tempo di completamento, così come lo abbiamo definito nel Capitolo 1. Nella Sezione 3.1.2 affrontiamo una versione leggermente diversa del problema dial-a-ride, in cui si richiede che il server ritorni nell'origine quando ha servito tutte le richieste. In letteratura quest'ultima è nota come *versione Home* (HOLDARP), mentre quella in cui non si richiede il ritorno del server nell'origine è nota come *versione Nomadic* (NOLDARP). In particolare, indicheremo le due varianti del problema del commesso viaggiatore online con NOLTSP per la versione Nomadic, HOLTSP per la versione Home.

Per entrambe le versioni sono state proposte particolari forme di avversari e di algoritmi, di cui si parlerà brevemente nella Sezione 3.1.3. Infine, è stata proposta un'interessante limitazione al regime informativo degli algoritmi online per i problemi dial-a-ride, secondo la quale si suppone che l'algoritmo online possa conoscere la destinazione di una corsa soltanto quando inizia a servire la corsa stessa: ce ne occupiamo nella Sezione 3.1.4.

3.1.1 Versione "Nomadic"

3.1.1.1 Il problema del commesso viaggiatore

Il seguente lower bound al rapporto di competitività di algoritmi per il problema del commesso viaggiatore online è stato fornito da Lipmann in [18].

Teorema 3.1 *Ogni algoritmo online ρ-competitivo per il* NOLTSP *su un generico spazio metrico ha $\rho > 2.02976$. Questo lower bound si ottiene sulla retta dei reali.*

Dimostrazione. Si veda [18]. □

Il miglior algoritmo noto per il TSP online su un generico spazio metrico è RETURN HOME$_\alpha$ (RH$_\alpha$), che è stato presentato sempre da Lipmann [18]. Il comportamento di RH$_\alpha$ è specificato completamente descrivendo le azioni che esso compie ogni volta che è rilasciata una nuova richiesta.

Algoritmo 3.2 (RETURN HOME$_\alpha$) *In ogni istante t in cui* RH$_\alpha$ *riceve una nuova richiesta, esso ritorna nell'origine tramite il percorso più breve. Una volta che si trova nell'origine, nell'istante t_0, calcola il giro ottimo $T^*_{t_0}$ su tutte le richieste presentate fino all'istante t_0.* RH$_\alpha$ *inizia a seguire questo giro $T^*_{t_0}$,*

mantenendosi entro una distanza di $\alpha t'$ dall'origine ($\alpha \leq 1$) ad ogni istante t', regolando la propria velocità soltanto quando ciò diventa indispensabile.

Il principio di funzionamento di RH_α è quello di rimanere sempre abbastanza vicino all'origine, perché quando è rilasciata una nuova richiesta non sia troppo dispendioso ritornare nell'origine per incominciare un nuovo giro. L'origine è un punto privilegiato in quanto il giro ottimo del server offline, che RH_α tenta di seguire, incomincia proprio dall'origine. Il valore di α non può essere troppo piccolo, altrimenti si renderebbe l'algoritmo lento, né troppo grande, per evitare di allontanarsi eccessivamente dall'origine.

Teorema 3.3 *L'algoritmo* RETURN HOME$_\alpha$ *è* $(\sqrt{2}+1)$*-competitivo per il* NOLTSP*, nel generico spazio metrico, quando* $\alpha = (\sqrt{2}-1)$.

Dimostrazione. [18] Sia t l'istante in cui è stata rilasciata l'ultima richiesta: evidentemente sussistono due lower bound per la soluzione ottima, $OPT(\sigma) \geq t$ e $OPT(\sigma) \geq |T_t^*|$. Distinguiamo due situazioni:

- RH_α *non deve mai regolare la sua velocità a partire dall'istante* t.
  Poiché nell'istante t RH_α si trova a una distanza di al più αt dall'origine, RH_α ritorna nell'origine non dopo l'istante $t + \alpha t$; perciò $\mathrm{RH}_\alpha(\sigma) \leq (1+\alpha)t + |T_t^*| \leq (1+\alpha)OPT(\sigma) + OPT(\sigma) = (2+\alpha)OPT(\sigma)$.

- RH_α *deve regolare la sua velocità dopo l'istante* t.
  Sia $\sigma_w = (t_w, P_w)$ l'ultima richiesta (nell'ordine di T_t^*) che costringe RH_α a regolare la velocità. Poiché RH_α regola la velocità il più tardi possibile, serve σ_w nell'istante $w = \frac{d(O,P_w)}{\alpha}$. Dopo w, RH_α continua a seguire T_t^* a piena velocità; indichiamo con $T(w)$ la parte di T_t^* che non è stata ancora completata nell'istante w. Chiaramente

  $$OPT(\sigma) \geq d(O,\, P_w) + |T(w)|,$$

  mentre

  $$\begin{aligned} \mathrm{RH}_\alpha(\sigma) &= w + |T(w)| = \frac{d(O,\, P_w)}{\alpha} + |T(w)| \\ &\leq \frac{1}{\alpha}\Big(d(O,\, P_w) + |T(w)|\Big) \leq \frac{1}{\alpha}\, OPT(\sigma). \end{aligned}$$

Dunque in ogni caso $\mathrm{RH}_\alpha(\sigma) \leq \max\left\{\alpha + 2,\, \frac{1}{\alpha}\right\} OPT(\sigma)$. Il valore più piccolo del massimo si ottiene per $\alpha = \sqrt{2} - 1$, che fornisce il rapporto di competitività di $\sqrt{2} + 1$. □

Sulla retta dei reali si è riusciti a fare di meglio: utilizzando le informazioni fornite dalla dimostrazione del lower bound, che è costruito proprio sull'asse reale, Lipmann è riuscito a fornire un algoritmo 2.06-competitivo, chiamato WAIT AND ANTICIPATE. Rimandiamo a [18] per la descrizione dell'algoritmo e per il calcolo del suo rapporto di competitività.

Sulla semiretta è stato trovato un lower bound inferiore:

Teorema 3.4 *Ogni algoritmo online ρ-competitivo per il* NOLTSP *sulla semiretta dei reali non-negativi ha* $\rho \geq \approx 1.60$.

Dimostrazione. Si veda [18]. □

Tuttavia il miglior algoritmo noto per la semiretta è ancora WAIT AND ANTICIPATE [18], ereditato dal caso della retta, con lo stesso rapporto di competitività di 2.06.

3.1.1.2 Il generico problema dial-a-ride

Per il generico problema dial-a-ride online in cui si vuole minimizzare il tempo di completamento, il più alto lower bound attualmente conosciuto è di nuovo il lower bound per il problema del commesso viaggiatore online, che ne costituisce un caso particolare (il valore è circa 2.03: vedi Teorema 3.1).

Il miglior algoritmo noto per il generico problema dial-a-ride online, per qualsiasi spazio metrico, è WAIT OR IGNORE$_\alpha$ (WI$_\alpha$), presentato da Lipmann in [18]. WI$_\alpha$ è descritto completamente specificando il suo comportamento ogni volta che è rilasciata una nuova corsa. WI$_\alpha$ deve il suo nome al fatto che esso, a volte, ignora temporaneamente alcune delle corse che sono state rilasciate: indichiamo con S l'insieme delle corse ignorate nell'istante t. Indichiamo con T_t^* il giro ottimo su tutte le corse rilasciate fino all'istante t, e con T_t^{WI} il giro ottimo su tutte le richieste non ignorate nell'istante t.

Algoritmo 3.5 (WAIT OR IGNORE$_\alpha$) *Ogni volta che, nell'istante t, è rilasciata una nuova corsa:*

- *Se* WI$_\alpha$ *può ritornare nell'origine prima dell'istante $\alpha|T_t^*|$, allora torna nell'origine e svuota l'insieme S. Attende nell'origine fino l'istante $\alpha|T_t^*|$, e quindi inizia a seguire il giro ottimo T^{WI} su tutte le richieste ancora non servite.*

- *Altrimenti,* WI$_\alpha$ *continua a seguire T^{WI}. La nuova richiesta è aggiunta all'insieme S. Quando* WI$_\alpha$ *ha terminato, svuota S e calcola il giro ottimo su tutte le richieste non servite che inizia dall'origine.* WI$_\alpha$ *inizia a seguire questo giro, spostandosi direttamente tramite il percorso più breve sulla prima richiesta del giro.*

Teorema 3.6 *L'algoritmo* WAIT OR IGNORE$_\alpha$ *è* $\frac{3+\sqrt{5}}{2}$*-competitivo per il* NOLDARP *con funzione obiettivo tempo di completamento, in ogni spazio metrico, quando* $\alpha = \frac{1+\sqrt{5}}{2}$.

Dimostrazione. [18] Sia t l'istante di tempo in cui è rilasciata l'ultima corsa. Distinguiamo due situazioni:

- WI$_\alpha$ *può ritornare nell'origine prima dell'istante* $\alpha|T_t^*|$.
  Si ha che $OPT(\sigma) = |T_t^*| \geq |T^{WI}|$. D'altra parte, WI$_\alpha$ attende fino all'istante $\alpha|T_t^*|$ e poi serve le corse rimanenti in tempo $|T^{WI}|$, quindi: $\text{WI}_\alpha(\sigma) = \alpha|T_t^*| + |T^{WI}| \leq (\alpha+1)OPT(\sigma)$.

- WI$_\alpha$ *non può ritornare nell'origine prima dell'istante* $\alpha|T_t^*|$.
  Indichiamo con d_{WI} la destinazione dell'ultima corsa di T^{WI}, s_q la sorgente della prima corsa in S servita in una soluzione ottima (offline), e $T_q(S)$ il giro più breve che parte da s_q e serve tutte le corse in S. Indichiamo poi con t_l l'ultimo istante prima di t in cui l'insieme S era vuoto. Poiché WI$_\alpha$ stava servendo le corse di T^{WI} quando è arrivata la prima corsa in S, WI$_\alpha$ aveva già lasciato l'origine e quindi $t_l \geq \alpha|T^{WI}|$. Tutte le corse in S sono state rilasciate dopo t_l, per cui $OPT(\sigma) \geq t_l + |T_q(S)| \geq \alpha|T^{WI}|$. D'altra parte $\text{WI}_\alpha(\sigma) \leq t_l + |T^{WI}| + d(d_{WI},\, s_q) + |T_q(S)|$ e, dal momento che naturalmente $d(d_{WI},\, s_q) \leq OPT(\sigma)$:

$$\begin{aligned}\frac{\text{WI}_\alpha(\sigma)}{OPT(\sigma)} &\leq \frac{t_l + |T_q(S)|}{OPT(\sigma)} + \frac{d(d_{WI},\, s_q)}{OPT(\sigma)} + \frac{|T^{WI}|}{OPT(\sigma)} \\ &\leq 1 + 1 + \frac{|T^{WI}|}{\alpha|T^{WI}|} \leq 2 + \frac{1}{\alpha}.\end{aligned}$$

Dunque in ogni caso $\text{WI}_\alpha(\sigma) \leq \max\left\{\alpha+1,\, 2+\frac{1}{\alpha}\right\} OPT(\sigma)$. Il valore più piccolo del massimo si ottiene per $\alpha = \frac{1+\sqrt{5}}{2}$, che fornisce il rapporto di competitività di $\frac{3+\sqrt{5}}{2}$. □

A differenza di quanto avveniva per il problema del commesso viaggiatore online, non sono stati trovati algoritmi con un migliore rapporto di competitività per il problema dial-a-ride su spazi metrici particolari.

3.1.2 Versione "Home"

Come abbiamo accennato, nella versione *Home* di un problema dial-a-ride si impone che il server torni nell'origine dopo aver servito tutte le richieste. Evidentemente una tale imposizione ha senso soltanto quando la funzione

obiettivo riguarda il server (tempo di completamento, distanza percorsa), non ha senso se riguarda soltanto le richieste (latenza, latenza netta, massimo tempo di servizio).

Nell'analisi del problema del commesso viaggiatore online, versione Home (HOLTSP) e del problema dial-a-ride online con minimizzazione del tempo di completamento, versione Home (HOLDARP) si sono spesso trovati rapporti di competitività inferiori rispetto ai problemi omologhi nella versione Nomadic. Probabilmente, ciò si deve al fatto che, quando si impone un vincolo aggiuntivo (nel nostro caso il ritorno all'origine), si fornisce un'informazione in più all'algoritmo online.

3.1.2.1 Il problema del commesso viaggiatore

Iniziamo con il lower bound generale per l'HOLTSP, che è stato fornito da Ausiello et al. in [4]:

Teorema 3.7 *Ogni algoritmo online ρ-competitivo A per l'*HOLTSP *nel generico spazio metrico ha $\rho \geq 2$.*

Dimostrazione. Presentiamo una versione alternativa della dimostrazione, tratta da [18].

Lo spazio metrico è un grafo "a ragno" (vedi Figura 3.1) con insieme di vertici $X = \{1, 2, \ldots, n\} \cup \{O\}$ e la funzione distanza $d : X \times X \to \mathbb{R}_0^+$, con $d(O, i) = 1$ e $d(i, j) = 2$ $\forall i, j \in X \backslash O$.

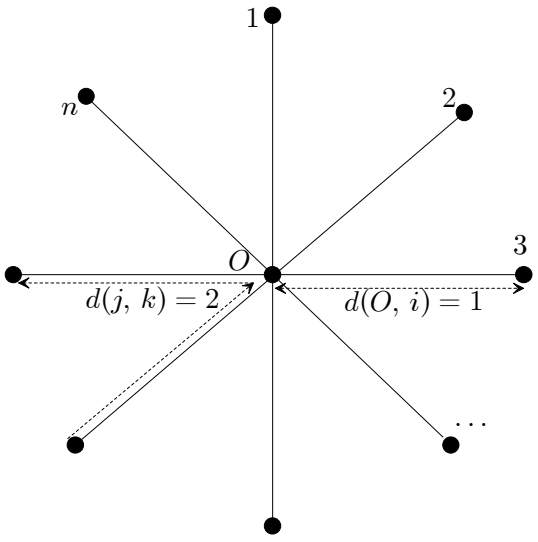

Figura 3.1: Il grafo a ragno.

Nell'istante 0 c'è una richiesta in ciascuno degli n punti di $X \backslash O$. Fino all'istante $2n-1-\varepsilon$, l'avversario si comporta nel modo seguente: se A serve la richiesta nel punto i nell'istante t, allora una nuova richiesta è rilasciata nel punto i nell'istante $t+\varepsilon$.

In questo modo, nell'istante $2n-1$ A deve ancora servire una richiesta in ciascuno degli n punti di X. Quindi $A(\sigma) \geq 2n-1+2n-1 = 4n-2$. È facile vedere che $OPT(\sigma) = 2n$, per cui $\frac{A(\sigma)}{OPT(\sigma)}$ si può rendere arbitrariamente vicino a 2, prendendo n sufficientemente grande. □

Il lower bound è conseguito dall'algoritmo PLAN AT HOME (PAH), presentato sempre da Ausiello et al. in [4]. Quando PAH sta servendo delle richieste e ne arriva una nuova, allora deve scegliere se interrompere il giro che sta compiendo oppure continuarlo e rimandare il servizio delle nuove richieste. Dal momento che il server dovrà comunque ritornare nell'origine, PAH decide di non interrompere il giro se la nuova richiesta è stata rilasciata in un punto "vicino" all'origine:

Algoritmo 3.8 (PLAN AT HOME**)** *Indichiamo con* $p = p_{PAH}(t)$ *la posizione del server mosso da* PAH *all'istante* t*.*

1. *Ogniqualvolta il server si trova nell'origine, inizia a seguire un giro ottimo che serve tutte le richieste non ancora servite e ritorna nell'origine.*

2. *Se nell'istante* t *una nuova richiesta è presentata nel punto* x*, allora* PAH *compie una delle seguenti azioni, a seconda della sua posizione attuale* p*:*

   2a. *Se* $d(x, O) > d(p, O)$*, allora il server ritorna nell'origine (seguendo il percorso più breve da* p*), dove si trova nella situazione del Caso 1.*

   2b. *Se* $d(x, O) < d(p, O)$*, allora il server la ignora fino a quando arriva nell'origine, dove nuovamente rientra nel Caso 1.*

Le richieste presentate tra un'occorrenza del Caso 2a e l'arrivo nell'origine non fanno deviare il server dal suo attuale percorso più breve verso l'origine.

Teorema 3.9 PLAN AT HOME *è 2-competitivo per l'*HOLTSP.

Dimostrazione. [4] Sia t l'istante di tempo in cui è presentata l'ultima richiesta; indichiamo con T^* il giro ottimo che parte dall'origine, serve tutte le richieste presentate e ritorna nell'origine. Chiaramente $OPT(\sigma) \geq t$ e $OPT(\sigma) \geq |T^*|$. Distinguiamo due casi:

1. Nell'istante t, PAH si trova nell'origine. Allora inizia subito a seguire il giro ottimo sulle richieste non servite che ritorna nell'origine, impiegando al più $|T^*|$. Pertanto $\text{PAH}(\sigma) \leq t + |T^*| \leq 2OPT(\sigma)$.

2. Nell'istante t, PAH non si trova nell'origine. Distinguiamo ancora due sottocasi:

   2a. $d(x, O) > d(p, O)$. Allora PAH ritorna in O, dove arriva prima dell'istante $t + d(x, O)$; poi calcola e segue il giro ottimo delle richieste ancora da servire che ritorna nell'origine. Dunque $\text{PAH}(\sigma) \leq t + d(x, O) + |T^*|$.
   Poiché OPT dopo aver servito la richiesta (t, x) deve ritornare nell'origine, allora $OPT(\sigma) \geq t + d(x, O)$. Da questo e dal fatto che $OPT(\sigma) \geq |T^*|$ si ricava che $\text{PAH}(\sigma) \leq 2OPT(\sigma)$.

   2b. $d(x, O) < d(p, O)$. Indichiamo con $\mathcal{R}$ il giro che PAH sta compiendo, S l'insieme delle richieste temporaneamente ignorate a partire dall'istante in cui è stato calcolato $\mathcal{R}$ (quando il server si trovava nell'origine); sia q la posizione della prima richiesta di S servita in una soluzione ottima, t_q l'istante di rilascio di tale richiesta. Infine, sia P_S^* il giro più corto che parte da q, serve tutte le richieste di S e termina in O. Chiaramente, $OPT(\sigma) \geq t_q + |P_S^*|$ e $OPT(\sigma) \geq |\mathcal{R}|$.
   Nell'istante t_q è stata rilasciata una delle richieste temporaneamente ignorate. Questo significa che era "vicina" all'origine, ossia $d(O, p(t_q)) \geq d(O, q)$; allora PAH nell'istante t_q ha percorso una parte di $\mathcal{R}$ lunga almeno $d(O, q)$, e ne rimane da percorrere una parte lunga al più $|\mathcal{R}| - d(O, q)$. Pertanto ritornerà nell'origine prima dell'istante $t_q + |\mathcal{R}| - d(O, q)$. Dopodiché, PAH seguirà il giro ottimo T_S^* che copre l'insieme S delle richieste non ancora servite: risulta che $|T_S^*| \leq d(O, q) + |P_S^*|$. Dunque:

$$\begin{aligned} \text{PAH}(\sigma) &\leq (t_q + |\mathcal{R}| - d(O, q)) + |T_S^*| \\ &\leq t_q + |\mathcal{R}| - d(O, q) + d(O, q) + |P_S^*| \\ &= |\mathcal{R}| + (t_q + |P_S^*|) \leq 2OPT(\sigma) \end{aligned}$$

□

Il lower bound di 2 è valido per il generico spazio metrico, e naturalmente per i grafi; tuttavia sulla retta e sulla semiretta si può fare di meglio.

Sulla retta vale il seguente lower bound (Ausiello et al., [4]):

Teorema 3.10 *Ogni algoritmo ρ-competitivo per l'*HOLTSP *sulla retta dei reali ha* $\rho \geq \frac{9+\sqrt{17}}{8} \approx 1.64$.

Dimostrazione. Vedi [4]. □

Anche questo lower bound è il più alto possibile, perché Lipmann [18] ha fornito un algoritmo, WAIT DELIBERATELY, $\frac{9+\sqrt{17}}{8}$-competitivo:

Teorema 3.11 WAITE DELIBERATELY *è un algoritmo* $\frac{9+\sqrt{17}}{8}$*-competitivo per l'*HOLTSP *sulla retta dei reali.*

Dimostrazione. Vedi [18]. □

Sulla semiretta il lower bound si abbassa (Blom et al., [5]):

Teorema 3.12 *Ogni algoritmo ρ-competitivo per l'*HOLTSP *sulla semiretta dei reali non-negativi ha* $\rho \geq 3/2$.

Dimostrazione. Vedi [5]. □

Blom et al. hanno anche fornito un semplice algoritmo che è il migliore algoritmo possibile, MOVE RIGHT IF NECESSARY (MRIN) [5]:

Teorema 3.13 MRIN *è un algoritmo* 3/2*-competitivo per l'*HOLTSP *sulla semiretta dei reali non-negativi.*

Dimostrazione. Vedi [5]. □

3.1.2.2 Il generico problema dial-a-ride

Il lower bound di Ausiello et al. per l'HOLTSP ([4], vedi Teorema 3.7) fornisce un lower bound di 2 anche per il più generale problema HOLDARP. Anche in questo caso il lower bound è il più alto possibile, infatti Krumke ha fornito un algoritmo 2-competitivo, SMARTSTART (vedi [14] per la definizione):

Teorema 3.14 SMARTSTART *è 2-competitivo per l'*HOLDARP *sul generico spazio metrico.*

Dimostrazione. Vedi [14]. □

Sulla retta e sulla semiretta Krumke [14] ha determinato dei lower bound un po' più alti rispetto a quelli forniti dal caso particolare del HOLTSP:

Teorema 3.15 *Ogni algoritmo ρ-competitivo per l'*HOLDARP *sulla retta dei reali ha $\rho \geq 7/4$.*

Dimostrazione. Vedi [14]. □

Teorema 3.16 *Ogni algoritmo ρ-competitivo per l'*HOLDARP *sulla semiretta dei reali non-negativi ha $\rho \geq \frac{2+\sqrt{2}}{2}$.*

Dimostrazione. Vedi [14]. □

Tuttavia non sono noti algoritmi che lavorano in questi spazi metrici particolari conseguendo un lower bound inferiore a 2, per cui il miglior algoritmo conosciuto è SMARTSTART.

3.1.3 Uso di particolari forme di avversari ed algoritmi

Come abbiamo visto nella Sezione 2.3, l'avversario offline a volte è considerato eccessivamente potente nei confronti dell'algoritmo online, per cui si è cercato di trovare dei modelli di avversario che, in qualche modo, si comportassero in modo non eccessivamente dissimile da un comportamento online.

Un modello di avversario che è stato estensivamente utilizzato nell'analisi di competitività di problemi dial-a-ride in cui si vuole minimizzare il tempo di completamento è l'avversario leale (Blom et al., [5]):

Definizione 3.17 (Avversario leale) *L'*avversario leale *muove il suo server sul percorso ottimo che verifichi la seguente condizione: in ogni istante t, il server si trova nel cono convesso dell'origine e dei punti delle richieste rilasciate fino all'istante t.*

È molto interessante indagare sull'influenza che determinate caratteristiche di un algoritmo hanno sulle sue prestazioni. Per esempio ci si può domandare se sia conveniente per un algoritmo online A aspettare un po' di tempo prima di iniziare a servire le richieste, in modo tale che, se sono rilasciate nuove richieste, A non si trovi eccessivamente svantaggiato per via di decisioni "affrettate".

Per rispondere a questa domanda, Blom et al. in [5] hanno introdotto il modello di algoritmo diligente:

Definizione 3.18 (Algoritmo diligente) *Il server mosso da un* algoritmo diligente*, cioè un* server diligente*, non può stare fermo se ci sono delle richieste da servire, ma si deve muovere alla velocità massima. Se ci sono*

delle richieste da servire, allora la direzione in cui si muove il server diligente non può cambiare, tranne che in un istante in cui (i) una nuova richiesta è rilasciata, oppure (ii) il server si trova nell'origine o (iii) il server si trova sul punto di una richiesta.

Diversi autori hanno compiuto indagini sui problemi dial-a-ride con la funzione obiettivo del tempo di completamento, nelle versioni Home e Nomadic, utilizzando avversari leali, algoritmi diligenti, ed entrambi in combinazione, in diversi spazi metrici. Nelle Tabelle 3.1 e 3.2, tratte da [18], sono riportati i lower bound e gli upper bound sui rapporti di competitività degli algoritmi online per tali problemi, con i riferimenti ai lavori in cui ciascun risultato è stato pubblicato. Naturalmente, i casi in cui si fanno competere algoritmi "standard" con avversari "standard" sono quelli che abbiamo esposto nelle sezioni precedenti.

Un aspetto di grande interesse che emerge dall'analisi di queste tabelle è il fatto che, per quasi tutti i problemi considerati, il miglior algoritmo non è un algoritmo diligente. Non sono diligenti, ad esempio, Return Home e Wait or Ignore (come suggerisce il nome), che abbiamo presentato e analizzato in precedenza; invece, Plan At Home è diligente.

Questo conferma l'intuizione secondo cui attendere un po' prima di iniziare a servire le richieste porta giovamento nel caso peggiore. Rimane il dubbio su quello che sia meglio nella pratica, dal momento che l'attesa potrebbe abbassare le prestazioni dell'algoritmo su un gran numero di istanze di input: per questo, come abbiamo già osservato, conviene affiancare all'analisi di competitività un'analisi sperimentale, possibilmente compiuta su istanze desunte dalle applicazioni pratiche.

3.1.4 Introduzione di ulteriori limitazioni al regime informativo online

Il modello online di problema dial-a-ride non corrisponde pienamente a quello che avviene in molte situazioni reali. Si immagini un impianto di ascensori: l'utente chiama l'ascensore premendo il pulsante di chiamata del piano p, e in questo modo l'impianto viene a conoscere l'esistenza di una corsa avente, per sorgente, il piano p. Tuttavia l'impianto non conosce la destinazione della corsa finché l'utente non entra nella cabina e preme il pulsante relativo al piano a cui vuole essere trasportato.

In un classico problema dial-a-ride online, una corsa è rivelata per intero all'algoritmo online nell'istante di rilascio della corsa stessa. In un problema dial-a-ride con regime informativo limitato, invece, nell'istante di rilascio l'algoritmo online viene a conoscenza della sorgente della corsa, e di essa

		non diligenti		diligenti	
		LB	UB	LB	UB
Homing generale		2 [4]	2 [4]	2 [4]	2 [4]
Homing sulla retta	standard	$\frac{9+\sqrt{17}}{8}$ [4]	$\frac{9+\sqrt{17}}{8}$ [18]	$\frac{7}{4}$ [5]	$\frac{7}{4}$ [4]
	leale	$\frac{5+\sqrt{57}}{8}$ [18]	$\frac{5+\sqrt{57}}{8}$ [18]	$\frac{8}{5}$ [21]	$\frac{7}{4}$ [4]
Homing sulla semiretta	standard	$\frac{3}{2}$ [5]	$\frac{3}{2}$ [5]	$\frac{3}{2}$ [5]	$\frac{3}{2}$ [5]
	leale	$\frac{1+\sqrt{17}}{4}$ [5]	$\frac{1+\sqrt{17}}{4}$ [5]	$\frac{4}{3}$ [5]	$\frac{4}{3}$ [5]
Nomadic generale		≈ 2.03 [18]	$1+\sqrt{2}$ [18]	$\frac{2\sqrt{21}-3}{3}$ [18]	$\frac{5}{2}$ [4]
Nomadic sulla retta	standard	≈ 2.03 [18]	2.06 [18]	$\frac{2\sqrt{21}-3}{3}$ [18]	$\frac{7}{3}$ [4]
	leale	$\frac{1+\sqrt{97}}{6}$ [18]	2.06 [18]	$\frac{\sqrt{33}}{3}$ [18]	$\frac{7}{3}$ [4]
Nomadic sulla semiretta	standard	≈ 1.63 [18]	2.06 [18]	$\frac{7}{4}$ [18]	$\frac{7}{3}$ [4]
	leale	≈ 1.60 [18]	2.06 [18]	$\sqrt{3}$ [18]	$\frac{7}{3}$ [4]

Tabella 3.1: Lower bound ed upper bound sui rapporti di competitività degli algoritmi deterministici per varianti del problema del commesso viaggiatore online.

soltanto. Per conoscere la destinazione della corsa, il server online deve iniziare la corsa stessa, ossia deve andare nel punto di sorgente e prendere l'oggetto da trasportare.

Il modello di regime informativo limitato è stato introdotto da Lipmann et al. in [19] con due obiettivi principali:

- fornire un modello adeguato all'analisi di molte situazioni reali, ad esempio un impianto di ascensori, oppure molti servizi radio-taxi: infatti spesso si richiede al cliente che telefona alla centrale operativa di fornire soltanto il punto in cui si trova, mentre la destinazione da raggiungere sarà comunicata direttamente all'autista;

- indagare sull'utilità di investire su un migliore sistema informativo. Gli autori di [19] hanno osservato il fatto che spesso la mancanza di informazioni non è una proprietà strutturale del problema pratico, ma è una scelta "di progetto". Per esempio una centrale di taxi potrebbe richiedere anche la destinazione di una corsa quando il cliente telefona; in un impianto di ascensori, in ogni piano potrebbe essere posta una pulsantiera con la quale l'utente chiama l'ascensore specificando subito il piano a cui vuole essere trasportato. Naturalmente ampliare il siste-

		non diligenti		diligenti	
		LB	UB	LB	UB
Homing generale		2 [4]	2 [14]	2 [4]	$\frac{5}{2}$ [8, 14]
Homing sulla retta	standard	$\frac{7}{4}$ [18]	2 [14]	2 [18]	$\frac{5}{2}$ [8, 14]
	leale	$\frac{1+\sqrt{5}}{2}$ [18]	2 [14]	2 [18]	$\frac{5}{2}$ [8, 14]
Homing sulla semiretta	standard	$\frac{2+\sqrt{2}}{2}$ [14]	2 [14]	2 [18]	$\frac{5}{2}$ [8, 14]
	leale	$\frac{1+\sqrt{5}}{2}$ [18]	2 [14]	2 [18]	$\frac{5}{2}$ [8, 14]
Nomadic generale		≈ 2.03 [18]	$\frac{3+\sqrt{5}}{2}$ [18]	$\frac{5}{2}$ [18]	3 [14]
Nomadic sulla retta	standard	≈ 2.03 [18]	$\frac{3+\sqrt{5}}{2}$ [18]	$\frac{5}{2}$ [18]	3 [14]
	leale	$\frac{1+\sqrt{22}}{3}$ [18]	$\frac{3+\sqrt{5}}{2}$ [18]	2 [18]	3 [14]
Nomadic sulla semiretta	standard	$\frac{1+\sqrt{22}}{3}$ [18]	$\frac{3+\sqrt{5}}{2}$ [18]	$\frac{5}{2}$ [18]	3 [14]
	leale	$\frac{1+\sqrt{22}}{3}$ [18]	$\frac{3+\sqrt{5}}{2}$ [18]	2 [18]	3 [14]

Tabella 3.2: Lower bound ed upper bound sui rapporti di competitività degli algoritmi deterministici per problemi dial-a-ride online minimizzanti il tempo di completamento.

ma informativo ha un costo, e ci si chiede se il vantaggio sia tale da giustificare il costo da sostenere.

Nella Tabella 3.3 sono riassunti i risultati determinati in [19], in termini di lower bound di problemi e upper bound di algoritmi. I problemi considerati sono problemi dial-a-ride versione Homing ad un singolo server, con capacità unitaria, costante pari a $c \geq 2$ oppure infinita; la prelazione può essere o meno ammessa.

Prelazione	**Capacità**	**LB**	**UB**
Sì	1, c, ∞	3	3
No	1	$1 + \frac{3}{2}\sqrt{2}$	4
No	c	$\max\left\{1 + \frac{3}{2}\sqrt{2},\ c\right\}$	$2c + 2$
No	∞	3	3

Tabella 3.3: Lower bound e upper bound sui rapporti di competitività degli algoritmi deterministici per problemi dial-a-ride con modello di regime informativo limitato.

Come si può notare, i lower bound sono sensibilmente più alti (almeno del

50%) rispetto al lower bound di 2 (Teorema 3.7) del problema HOLDARP con un server a capacità unitaria. Nel caso in cui è ammessa la prelazione esiste un algoritmo 3-competitivo, SNIFFER (per la definizione vedi [19]), che raggiunge il lower bound di 3. È interessante osservare che SNIFFER è basato sull'algoritmo 2-competitivo PLAN AT HOME (Algoritmo 3.8, [4]), e che sfrutta la prelazione, a grandi linee, nel modo seguente: prima di servire le corse, SNIFFER effettua un giro delle sorgenti delle corse; ogni volta che si trova in una sorgente inizia la corsa, venendo così a conoscenza della destinazione della corsa stessa; immediatamente dopo, prelaziona la corsa e continua il giro delle sorgenti.

Se la prelazione non è ammessa, come spesso avviene nella pratica (si immagini quello che succederebbe se un taxi implementasse l'algoritmo SNIFFER...) i lower bound aumentano ulteriormente, in particolare con l'aumentare della capacità del server; l'algoritmo proposto, BOUNCER, utilizza PLAN AT HOME come sottoprocedura e consegue un rapporto di competitività di $2c+2$ (4 quando $c = 1$). Quando la capacità diventa infinita, tuttavia, la prelazione non è più necessaria dal momento che il server può mantenere dentro di sé tutti gli oggetti che desidera piuttosto che lasciarli per poi riprenderli. In questo modo diventa possibile utilizzare il 3-competitivo SNIFFER.

3.2 La latenza

3.2.1 I lower bound

I problemi dial-a-ride online in cui la funzione obiettivo è la latenza netta sono stati analizzati per la prima volta da E. Feuerstein e L. Stougie in [8], dove sono stati forniti i seguenti lower bound per il problema del riparatore viaggiatore online e per il generico problema dial-a-ride.

Teorema 3.19 *Se A è un algoritmo ρ-competitivo per il problema del riparatore viaggiatore online definito sulla retta, in cui la funzione obiettivo è la latenza (L-OLTRP), allora $\rho \geq 1 + \sqrt{2}$.*

Dimostrazione. [8] L'avversario presenta una richiesta $\sigma_1 = (0, -1)$. Se A non la serve entro l'istante $1 + \sqrt{2}$, allora l'avversario non presenta altre richieste, e serve la richiesta σ_1 nell'istante stesso in cui è rilasciata. Quindi $\frac{A(\sigma)}{OPT(\sigma)} \geq 1 + \sqrt{2}$.

Altrimenti, sia T l'istante in cui A serve σ_1. L'avversario presenta N nuove richieste $\sigma_2 = \ldots = \sigma_{N+1} = (T, T)$: tali richieste potranno essere servite da A non prima dell'istante $2T + 1$, per cui

$$A(\sigma) \geq T + N(2T + 1).$$

L'avversario serve per prime $\sigma_2, \ldots, \sigma_{N+1}$, nell'istante T, poi si sposta in -1 e serve σ_1 pagando

$$OPT(\sigma) = NT + 2T + 1.$$

Quando N è grande, $\frac{A(\sigma)}{OPT(\sigma)}$ tende a $f(T) = \frac{2T+1}{T}$, che è una funzione monotona decrescente per $T \in [1, 1+\sqrt{2}]$ (questo è l'intervallo di tempo in cui A può servire σ_1). Dunque $\frac{A(\sigma)}{OPT(\sigma)} \geq f(1+\sqrt{2}) = 1+\sqrt{2}$. □

Teorema 3.20 *Se A è un algoritmo ρ-competitivo per il problema dial-a-ride online definito sulla retta in cui la funzione obiettivo è la latenza (L-*OLTRP*), allora $\rho \geq 1+\sqrt{2}$.*

Dimostrazione. Vedi [8]. □

In [16] Laura ha fornito i lower bound di altre varianti di problemi dial-a-ride con minimizzazione della latenza, prendendo in considerazione anche gli spazi metrici del segmento unitario e della semiretta, ed i problemi in cui il server ha capacità non unitaria. I risultati sono riassunti nella Tabella 3.4.

Spazio metrico	**Capacità**	**LB**
Intervallo	1	$1+\sqrt{2} \approx 2.41$
Intervallo	c, ∞	$\frac{1+\sqrt{5}}{2} \approx 1.62$
Semiretta	1	$1+\sqrt{2} \approx 2.41$
Semiretta	c, ∞	$\frac{1+\sqrt{5}}{2} \approx 1.62$
Retta	1	3
Retta	c, ∞	$1+\sqrt{2} \approx 2.41$

Tabella 3.4: Lower bound sui rapporti di competitività degli algoritmi dial-a-ride con minimizzazione della latenza.

3.2.2 Il migliore algoritmo noto

Il miglior algoritmo che si conosce per il problema della latenza è INTERVAL$_\alpha$, fornito da Krumke et al. in [15]. INTERVAL$_\alpha$ è stato studiato per la più generale funzione obiettivo della latenza pesata. L'idea di base è quella di suddividere l'asse dei tempi in finestre la cui ampiezza cresce in maniera esponenziale. INTERVAL$_\alpha$ lavora in fasi: fondamentalmente, ad ogni fase corrisponde una finestra temporale. Durante una fase INTERVAL$_\alpha$ segue un

percorso tale da massimizzare il peso delle richieste servite; in altre parole, in ogni fase INTERVAL$_\alpha$ cerca di servire la maggiore quantità di richieste possibile.

Algoritmo 3.21 (INTERVAL$_\alpha$) *L'algoritmo lavora in fasi.*
Fase 0: *inizializzazione.*
Sia L il primo istante in cui una qualsiasi richiesta può essere completata da OPT. Possiamo assumere che $L > 0$, poiché $L = 0$ significa che ci sono delle richieste $(0, O, O)$ che possono essere servite a costo 0 sia da OPT che da INTERVAL$_\alpha$. *Per $i = 1, 2, \ldots$, definiamo $B_i := \alpha^{i-1}L$, dove $\alpha \in]1, 3]$ è una costante scelta una volta per tutte.*
Fase *i, per $i = 1, 2, \ldots$*
Nell'istante B_i calcola uno schedule S_i per l'insieme delle richieste rilasciate fino all'istante B_i che ancora non sono state servite, con le seguenti proprietà:

(i) Lo schedule S_i parte dall'origine O.

(ii) Lo schedule S_i termina con il server vuoto in un punto x_i tale che $d(O, x_i) \leq B_i$.

(iii) La lunghezza dello schedule S_i, denotata da $l(S_i)$, soddisfa: $l(S_i) \leq B_i$;

(iv) S_i è lo schedule, fra tutti quelli che soddisfano le proprietà (i)-(iii), che massimizza la somma dei pesi delle richieste servite.

A partire dall'istante βB_i, dove $\beta := \frac{2}{\alpha-1}$, segui lo schedule S_i. Non appena lo schedule S_i è terminato, ritorna nell'origine.

Si noti che β è scelto in modo tale che il server possa partire all'istante βB_i, percorrere il tragitto lungo $l(S_i) \leq B_i$ e ritornare nell'origine, dove si troverà prima dell'istante $\beta B_i + 2B_i = \left(\frac{2}{\alpha-1} + 2\right) B_i = \frac{2}{\alpha-1}\alpha B_i = \beta B_{i+1}$: proprio in tempo per iniziare a seguire lo schedule S_{i+1}.

Teorema 3.22 INTERVAL$_\alpha$ *è un algoritmo $(1+\sqrt{2})^2$-competitivo per il problema dial-a-ride online con funzione obiettivo latenza, su qualsiasi spazio metrico, quando $\alpha = (1 + \sqrt{2})$.*

Dimostrazione. Vedi [15]. □

INTERVAL$_\alpha$ è anche il migliore algoritmo online attualmente conosciuto per il L-OLTRP; si noti che esiste un gap ancora piuttosto ampio tra il lower bound, di $(1 + \sqrt{2})$ per il L-OLTRP o di 3 per il problema dial-a-ride generico, e l'upper bound di $(1 + \sqrt{2})^2 \approx 5.825$ fornito da INTERVAL$_\alpha$.

3.2.3 Un algoritmo polinomiale per la retta

INTERVAL$_\alpha$ è un algoritmo esponenziale, a meno che $P = NP$. Recentemente, Pini [22] ha fornito un algoritmo polinomiale per lo spazio metrico della retta, con un rapporto di competitività pari a 6.14.

3.2.4 Discussione

Consideriamo nuovamente l'algoritmo INTERVAL$_\alpha$. Nell'istante B_i l'algoritmo calcola lo schedule S_i. Supponiamo che, nell'istante $t^* = B_i + \varepsilon$ sia rilasciata una nuova richiesta σ^*. La richiesta σ^* per il momento è ignorata; sarà presa in considerazione soltanto nell'istante $B_{i+1} = 2B_i \approx 2t^*$, ovvero quando è calcolato lo schedule S_{i+1}; e sarà servita non prima dell'istante βB_{i+1}, quando lo schedule S_{i+1} ha inizio.

Questo significa che una richiesta può essere servita da INTERVAL$_\alpha$ ad una distanza di tempo almeno pari al tempo in cui essa è stata rilasciata. Si consideri ad esempio un servizio di radio-taxi che implementi INTERVAL$_\alpha$, a partire dal 1° gennaio. Le prime corse saranno servite molto rapidamente; ma una corsa richiesta a giugno dello stesso anno potrebbe tranquillamente essere servita a dicembre.

Naturalmente la "colpa" di questo comportamento non è di INTERVAL$_\alpha$, ma della funzione obiettivo della latenza, o meglio dell'analisi di competitività compiuta utilizzando la funzione obiettivo della latenza. La difficoltà è generata dal fatto che al costo di una soluzione (ottima oppure online) contribuisce non solo il tempo di servizio di ogni richiesta (calcolato a partire dall'istante in cui la richiesta è rilasciata), ma anche il tempo trascorso dall'origine dei tempi all'istante di rilascio di ogni richiesta. Quindi, finché il tempo di servizio di ogni richiesta si mantiene proporzionale al suo istante di rilascio, siamo sicuramente competitivi.

Per mostrare che l'analisi di competitività sulla latenza in molti casi non fornisce indicazioni adeguate a studiare la bontà di un algoritmo per le applicazioni pratiche, supponiamo che esista un algoritmo online A per il L-OLTRP che gode della seguente proprietà. Per ogni istanza di input σ, per ogni richiesta $\sigma_i = (t_i, x_i)$, il server controllato da A serve σ_i nell'istante $\tau_i = \rho \cdot t_i$, dove $\rho = 1 + \sqrt{2}$. Evidentemente l'algoritmo A è $(1 + \sqrt{2})$-competitivo; quindi, dal punto di vista dell'analisi di competitività, è ottimale per il L-OLTRP. D'altra parte si osservi che per servire una fissata richiesta il server impiega $\sqrt{2}$ volte il tempo in cui è stata rilasciata la richiesta. In altri termini, il tempo di servizio di una richiesta dipende fortemente dall'istante di tempo in cui la richiesta è stata rilasciata, ed aumenta man

mano che scorre il tempo. Questa caratteristica renderebbe A inutilizzabile nella pratica in quasi tutte le applicazioni.

Osserviamo per inciso che l'algoritmo 6.14-competitivo di E. Pini, pur conseguendo un rapporto di competitività più alto di quello di INTERVAL$_\alpha$, non possiede un comportamento "esponenziale" simile a quello di INTERVAL$_\alpha$, e che quindi nella pratica potrebbe comportarsi meglio.

Per ovviare agli inconvenienti emersi in questa sezione, la scelta più naturale sembrerebbe quella di utilizzare un'altra funzione obiettivo (ad esempio la latenza netta) per la quale soltanto i tempi di attesa delle richieste contribuiscono al costo della soluzione. Nella prossima sezione vedremo che utilizzare la latenza netta non è affatto semplice.

3.3 La latenza netta

3.3.1 I risultati negativi dell'analisi di competitività classica

Quando la funzione obiettivo da minimizzare è la latenza netta, l'analisi di competitività non ci viene in aiuto. Infatti vale il seguente lower bound.

Teorema 3.23 *Qualsiasi algoritmo A per il problema del riparatore viaggiatore online in cui la funzione obiettivo è la latenza netta, definito sullo spazio metrico $M = (X, d) \in \mathcal{M}$, non è competitivo.*

Dimostrazione. La dimostrazione che segue è stata adattata da [16].

Indichiamo con $p_S(t)$ la posizione del server mosso dall'algoritmo S all'istante t.

Poiché M non è banale (vedi Sezione 1.2.1), oltre all'origine O in cui si trovano entrambi i server all'istante 0 esiste un punto $P \in X$ tale che $d := d(P, O) > 0$. L'avversario offline presenta esattamente una richiesta nell'istante di tempo d, in un punto che dipende dalla posizione del server mosso da A nel medesimo istante. L'idea è la seguente: almeno uno dei due punti O e P dista dal server online almeno $d/2$ nell'istante d, in virtù della disuguaglianza triangolare. Presentando la richiesta in un punto che dista almeno $d/2$ dal server online, sicuramente questi impiegherà almeno $d/2$ per servirla.

Ecco la richiesta presentata:

$$\sigma_1 = \begin{cases} (d,\, P), & d(O, p_A(d)) \leq d/2; \\ (d,\, O), & d(O, p_A(d)) > d/2. \end{cases}$$

Il server ottimo offline è in grado di servire la richiesta nell'istante stesso in cui è rilasciata, per cui $OPT(\sigma) = 0$; al contrario, il server mosso da A raggiunge il punto in cui è rilasciata σ_1 nell'istante $\tau_1 \geq d + d/2$. Quindi $A(\sigma) \geq d/2 > 0$. □

Corollario 3.24 *Qualsiasi algoritmo A per il problema dial-a-ride online in cui la funzione obiettivo è la latenza netta, definito sullo spazio metrico $M \in \mathcal{M}$, non è competitivo.*

Osserviamo che la dimostrazione del teorema si basa fortemente sulla capacità del server ottimo offline di servire la richiesta non appena è stata presentata, senza incorrere in alcun costo. Questo suggerisce di modificare la funzione obiettivo da minimizzare. L'idea è quella di aggiungere alla latenza netta un costo fisso TS che deve essere pagato ogniqualvolta si serve una richiesta. TS si può interpretare come il costo di servizio di una richiesta. Dunque la nuova funzione obiettivo diventa:

$$f = \sum_{i=0}^{n} (\tau_i - t_i) + n\,TS\,.$$

Tuttavia questo approccio non fornisce l'aiuto sperato se lo spazio metrico non è limitato, come è affermato da un altro teorema che compare in [16]:

Teorema 3.25 *Se A è un algoritmo per il problema del riparatore viaggiatore online avente come funzione obiettivo la latenza netta con costi di servizio, definito sulla retta dei reali, allora A non è competitivo.*

Dimostrazione. Come prima, sia $p_S(t)$ la posizione del server mosso da S all'istante t.

L'avversario rilascia esattamente una richiesta, all'istante $T \in \mathbb{R}^+$, a seconda della posizione del server mosso da A in T:

$$\sigma_1 = \begin{cases} (T,\, T), & p_A(T) \leq 0; \\ (T,\, -T), & p_A(T) > 0. \end{cases}$$

L'avversario è in grado di servire la richiesta nell'istante in cui è rilasciata, pagando solamente il costo del servizio TS; invece A raggiunge la richiesta almeno nell'istante $2T$, per cui $A(\sigma) \geq T + TS$. Ne risulta che:

$$\frac{A(\sigma)}{OPT(\sigma)} \geq \frac{T + TS}{TS}\,.$$

Questa quantità può essere resa maggiore di un qualsiasi $\rho \in \mathbb{R}^+$, scegliendo un valore opportuno per T. □

Se lo spazio metrico è limitato questo lower bound non è più valido, come vedremo in seguito.

3.3.2 Una forma di analisi alternativa: l'analisi sotto condizioni ragionevoli di carico

Data l'impossibilità di utilizzare l'analisi di competitività classica per l'analisi dei problemi dial-a-ride con la funzione obiettivo della latenza netta, Hauptmeier et al. [11] hanno osservato che questi problemi sono utili per modellizzare sistemi che operano continuativamente, ad esempio un servizio di radio-taxi o un impianto di ascensori. In questi sistemi ci si aspetta che, in un intervallo di tempo sufficientemente lungo, il numero delle richieste rilasciate sia pari al numero delle richieste servite. In altri termini, ci si aspetta che il carico di lavoro del sistema sia "ragionevole", nel senso che il sistema deve avere la possibilità, entro un tempo ragionevole, di servire tutte le richieste che sono state rilasciate.

Partendo da questa osservazione, Hauptmeier et al. introducono il concetto di insieme Δ_0-ragionevole di richieste. A grandi linee, l'insieme di richieste R è Δ_0-*ragionevole* se, per ogni $\Delta \geq \Delta_0$, per ogni intervallo di tempo $[t, t+\Delta[$ di ampiezza Δ, un server che parte dall'origine nell'istante $t + \Delta$ è in grado di servire entro l'istante $t + 2\Delta$ tutte le richieste rilasciate nell'intervallo $[t, t + \Delta[$. Un insieme R di richieste è detto *ragionevole* se è Δ_0-ragionevole per qualche $\Delta_0 \in \mathbb{R}_0^+$.

La nozione di Δ_0-ragionevolezza è usata per studiare le prestazioni degli algoritmi online, con particolare riguardo alla stabilità del sistema. Un sistema continuativamente operante è *stabile* se il numero di richieste in attesa di essere servite non diventa arbitrariamente grande.

Nel lavoro sono presi in considerazione in particolare due algoritmi classici: IGNORE e REPLAN. IGNORE ha una coda di richieste e si può trovare in uno tra due stati, 0 e 1. Ogni volta che è rilasciata una nuova richiesta, essa è messa in coda. Inizialmente IGNORE si trova nello stato 0. L'algoritmo rimane nello stato 0 finché la coda è vuota. Appena la coda contiene almeno una richiesta, IGNORE svuota la coda, calcola il giro ottimo offline T per le richieste estratte dalla coda, ossia il giro di lunghezza minima che serve le richieste estratte dalla coda, ed entra nello stato 1. Durante lo stato 1, IGNORE semplicemente segue il giro T; al termine del giro, ritorna nello stato 0.

REPLAN, invece, lavora nel seguente modo: ogni volta che è rilasciata una nuova richiesta, REPLAN (i) interrompe l'eventuale giro che sta percorrendo, (ii) calcola il giro più corto T che gli consente di servire tutte le richieste che sono state rilasciate ma non ancora servite, e (iii) incomincia a seguire il giro T.

Presentiamo ora i risultati che sono stati ottenuti dall'analisi di IGNORE e di REPLAN, osservando che in un sistema continuativamente operante è opportuno utilizzare, al posto della latenza netta, la funzione obiettivo (equivalente) del tempo medio di servizio.

Teorema 3.26 *Sia $\Delta_0 \in \mathbb{R}^+$. Per ogni insieme di richieste R che sia Δ_0-ragionevole, il massimo tempo di servizio conseguito su R da* IGNORE *non è superiore a $2\Delta_0$.*

Dimostrazione. Vedi [11]. □

Corollario 3.27 *Sia $\Delta_0 \in \mathbb{R}^+$. Per ogni insieme di richieste R che sia Δ_0-ragionevole, il tempo medio di servizio conseguito su R da* IGNORE *non è superiore a $2\Delta_0$.*

Dimostrazione. È sufficiente osservare che il tempo medio di servizio è non superiore al massimo tempo di servizio. □

Teorema 3.28 *Esiste un $\Delta_0 \in \mathbb{R}^+$ per cui vale la seguente proprietà. Per ogni $K \in \mathbb{R}^+$ esiste un'istanza di input Δ_0-ragionevole R tale che il massimo tempo di servizio e il tempo medio di servizio di* REPLAN *con input R sono maggiori di K.*

Dimostrazione. Vedi [11]. □

Nei Teoremi precedenti il parametro Δ_0 viene visto, in qualche modo, come una misura della difficoltà combinatoria dell'insieme R di richieste; le prestazioni degli algoritmi sono espresse in funzione di tale difficoltà.

Per comprendere l'idea che si trova dietro al Teorema 3.26, supponiamo che IGNORE abbia appena terminato di seguire il giro T_i, e che T_i abbia una lunghezza pari a Δ_0. Il giro successivo T_{i+1} deve servire in modo ottimo le richieste rilasciate mentre il server stava percorrendo T_i, ossia durante un intervallo temporale della durata di Δ_0. Per l'ipotesi di Δ_0-ragionevolezza, queste richieste possono essere servite in modo ottimo in un intervallo di tempo di durata non superiore a Δ_0: dunque ogni richiesta è servita al massimo $2\Delta_0$ dopo essere stata rilasciata.

Quello che abbiamo appena esaminato è il caso peggiore per IGNORE: infatti non è difficile dimostrare (vedi [11]) che la durata di un giro T_i si mantiene sempre non superiore a Δ_0. In altri termini, il fatto che IGNORE sia un algoritmo online fa sì che esso serva le richieste con un certo ritardo; con il trascorrere del tempo, questo ritardo può aumentare; ma proprio grazie al ritardo, ogni volta che IGNORE compie una nuova pianificazione ha a disposizione un insieme di richieste più ricco, che gli permette una migliore ottimizzazione. Quindi, intuitivamente, il ritardo cresce via via sempre più lentamente; l'ipotesi di Δ_0-ragionevolezza serve proprio a garantire che il ritardo non possa superare il valore limite di Δ_0.

Il motivo per cui REPLAN si comporta tanto male è da attribuirsi alla sua mancanza di memoria. Infatti, ogni volta che arriva una nuova richiesta, REPLAN effettua una nuova pianificazione senza alcun riguardo per l'istante di rilascio delle richieste: le vecchie richieste sono trattate esattamente come le nuove. In questo modo le richieste più svantaggiose da servire possono essere trascurate e rimanere in attesa per un tempo arbitrariamente elevato.

3.4 Il massimo tempo di servizio

Se fosse possibile fornire un algoritmo con un rapporto di competitività finito sul massimo tempo di servizio, si offrirebbero delle garanzie sull'insoddisfazione del cliente più sfortunato; in questo caso il massimo tempo di servizio costituirebbe una valida alternativa alla latenza netta per la quale, abbiamo visto, non esistono algoritmi competitivi.

Purtroppo la situazione della latenza netta si ripete per il massimo tempo di servizio, in quanto vale il seguente lower bound:

Teorema 3.29 *Ogni algoritmo A per il problema del riparatore viaggiatore online, definito sullo spazio metrico $M \in \mathcal{M}$, con funzione obiettivo massimo tempo di servizio, non è competitivo.*

Dimostrazione. Vale la stessa dimostrazione del Teorema 3.23, in quanto nella dimostrazione è utilizzata un'istanza di input contenente una sola richiesta. Quando $|\sigma| = 1$ evidentemente la latenza netta e il massimo tempo di servizio coincidono. □

Corollario 3.30 *Ogni algoritmo A per il problema dial-a-ride online, definito sullo spazio metrico $M \in \mathcal{M}$, con funzione obiettivo massimo tempo di servizio, non è competitivo.*

Per poter fornire indicazioni utili sulle prestazioni di algoritmi per il $F_{\max}$-OlTRP sulla retta dei reali, Laura et al. [17] hanno definito un nuovo tipo di avversario, meno potente dell'avversario leale (anche contro l'avversario leale non esistono algoritmi competitivi):

Definizione 3.31 (Avversario non-abusivo) *Un avversario per il $F_{\max}$-OlTRP su $\mathbb{R}$ è* non-abusivo *se in ogni istante il server è immobile, oppure si muove verso una richiesta che è stata rilasciata ma non ancora servita.*

Il $F_{\max}$-OlTRP contro l'avversario non-abusivo non è un problema banale, in quanto vale il seguente lower bound:

Teorema 3.32 *Nessun algoritmo online A per il $F_{\max}$-OlTRP su $\mathbb{R}$ può ottenere un rapporto di competitività inferiore a 2 contro un avversario non-abusivo.*

Dimostrazione. Vedi [17]. □

Laura et al. forniscono un algoritmo, Detour, con un rapporto di competitività finito per il $F_{\max}$-OlTRP sulla retta:

Teorema 3.33 Detour *è un algoritmo 8-competitivo contro un avversario non-abusivo per il $F_{\max}$-OlTRP.*

Dimostrazione. Vedi [17]. □

Capitolo 4

Il potere del lookahead per il L_N-OlTRP

In questo capitolo ci occuperemo dei problemi dial-a-ride in cui la funzione obiettivo da minimizzare è la latenza netta.

Abbiamo visto che la latenza netta è una funzione obiettivo molto naturale per un problema dial-a-ride, in quanto essa in molti contesti rappresenta l'insoddisfazione complessiva dei clienti. Tuttavia l'analisi di competitività classica per questi problemi porta a risultati non molto interessanti: abbiamo mostrato nel corollario del Teorema 3.23 che non esistono algoritmi competitivi per nessun problema dial-a-ride minimizzante la latenza netta.

Per l'enorme interesse di questi problemi vale la pena di tentare approcci alternativi, come quelli che abbiamo descritto nella Sezione 2.3.

Nel corso del capitolo ci occuperemo principalmente del problema del riparatore viaggiatore online con la funzione obiettivo della latenza netta (L_N-OlTRP), che si può immediatamente ridurre al più generale problema dial-a-ride con minimizzazione della latenza netta. Per questo problema, e più in generale per ogni problema dial-a-ride, definiremo due differenti modelli di lookahead. Nel primo modello l'algoritmo online conosce un numero costante di richieste future; nel secondo modello l'algoritmo online conosce le richieste che saranno rilasciate nel futuro all'interno di una finestra temporale di ampiezza fissata. Studieremo entrambi i modelli di lookahead sia tramite l'analisi di competitività classica, confrontandoli con l'avversario ottimo offline, sia tramite l'analisi di comparatività, confrontandoli con algoritmi online privi di lookahead o dotati di un diverso "grado" di lookahead. In quest'ultimo caso troveremo un risultato curioso, che fa pensare che l'analisi di comparatività sia uno strumento non adeguato per l'analisi del L_N-OlTRP. Invece con l'analisi di competitività classica vedremo che gli algoritmi dotati di lookahead del primo tipo (noto un numero costante di

richieste future) non sono mai competitivi, mentre quelli dotati di lookahead del secondo tipo (note le richieste future che cadono in una finestra) hanno un comportamento che varia a seconda dello spazio metrico in cui ci si trova, in particolare del suo diametro.

4.1 Modelli di lookahead

In questa sezione definiamo due differenti modelli di lookahead per i problemi dial-a-ride online. Richiamiamo che per definire un modello di lookahead occorre fornire la regola che consente di stabilire quale sia l'insieme S_t delle richieste note all'algoritmo online nell'istante di tempo t, conoscendo l'istanza di input σ del problema.

Nel primo modello di lookahead l'algoritmo online A nell'istante t conosce le prossime k richieste che saranno presentate nel futuro:

Definizione 4.1 (Lookahead in termini di richieste) *Un algoritmo online A si dice* dotato di lookahead di k richieste*, se l'insieme S_t delle richieste note ad A nell'istante t contiene tutte e sole le seguenti richieste:*

1. *le richieste (t', x') con $t' \leq t$;*
2. *altre k richieste $\sigma_1 = (t_1, x_1), \ldots, \sigma_k = (t_k, x_k)$ che verificano le seguenti proprietà:*
   - *(a)* $t < t_i \ \forall i \in \{1, \ldots, k\}$
   - *(b)* $\forall \sigma' = (t', x') \in \sigma \backslash S_t, \ \forall i \in \{1, \ldots, k\} : \ t' \geq t_i$.

Il punto 2. ci dice semplicemente che le k richieste di lookahead devono essere scelte tra le prime k richieste rilasciate dopo l'istante attuale t.

Indichiamo con $\mathcal{L}_k^{(R)}$ l'insieme degli algoritmi online per il problema del riparatore viaggiatore dotati di lookahead di k richieste.

Nel secondo modello di lookahead l'algoritmo online A conosce, all'istante di tempo t, tutte le richieste che saranno rilasciate nelle successive Δ unità di tempo:

Definizione 4.2 (Lookahead di una finestra temporale) *Un algoritmo online A si dice* dotato di lookahead di una finestra temporale di ampiezza Δ*, se l'insieme S_t delle richieste note ad A nell'istante t contiene tutte e sole le richieste $\sigma' = (t', x')$ tali che $t' \leq t + \Delta$.*

Indichiamo con $\mathcal{L}_\Delta^{(T)}$ l'insieme degli algoritmi online dotati di lookahead in una finestra di ampiezza Δ per il problema del riparatore viaggiatore online.

Si osservi che entrambi i modelli di lookahead che abbiamo presentato in questa sezione sono stati definiti facendo riferimento unicamente all'istante di tempo in cui le richieste sono presentate, e sono pertanto applicabili ad ogni tipo di problema dial-a-ride, indipendentemente, per esempio, dalla funzione obiettivo utilizzata, dallo spazio metrico o dal numero di server. Invece, gli insiemi $\mathcal{L}_k^{(R)}$ e $\mathcal{L}_\Delta^{(T)}$ contengono gli algoritmi (dotati di lookahead) per alcuni particolari problemi dial-a-ride, cioè il problema del riparatore viaggiatore online con tutte le possibili funzioni obiettivo (chiaramente la scelta di una differente funzione obiettivo non influenza la correttezza di un algoritmo: se $A \in \mathcal{L}_k^{(R)}$ è un algoritmo progettato per il L_N-OlTRP, A è un algoritmo corretto anche per il TSP online).

Osserviamo inoltre che $\mathcal{L}_0^{(R)} = \mathcal{L}_0^{(T)}$ è l'insieme degli algoritmi online privi di lookahead.

4.2 Algoritmi con lookahead in termini di richieste

In questa sezione ci occuperemo degli algoritmi online per il L_N-OlTRP dotati di un lookahead di k richieste. Per non appesantire troppo la notazione, con l'espressione "lookahead" si indicherà il lookahead in termini di richieste; inoltre l'insieme $\mathcal{L}_k^{(R)}$ sarà indicato semplicemente con $\mathcal{L}_k$.

Utilizzeremo due metodi di analisi. Dapprima vedremo, con l'ausilio dell'analisi di competitività classica, che nessun algoritmo online dotato di lookahead è competitivo per il L_N-OlTRP.

Ci chiederemo allora se sia possibile misurare il vantaggio offerto dal lookahead. Utilizzando l'analisi di comparatività troveremo un risultato curioso: se $k > h$, allora $R(\mathcal{L}_h, \mathcal{L}_k) = +\infty$. Sembrerebbe che gli algoritmi dotati di un lookahead maggiore si comportino molto meglio; in realtà anche algoritmi molto stupidi di $\mathcal{L}_k$ riescono a battere tutti gli algoritmi di $\mathcal{L}_h$ su delle istanze di input "tagliate su misura" per loro. Ne trarremo la conclusione che l'analisi di comparatività è uno strumento non adeguato all'analisi degli algoritmi dotati di lookahead per il L_N-OlTRP.

4.2.1 Analisi di competitività degli algoritmi con lookahead di k richieste

Nell'apprestarsi a studiare il comportamento degli algoritmi dotati di lookahead vengono alla mente due domande.

- Gli algoritmi dotati di lookahead sono vantaggiosi rispetto a quelli privi di lookahead?
- Un lookahead di un numero maggiore di richieste è realmente vantaggioso?

La seconda domanda nasce da un'osservazione classica nello studio degli algoritmi online (si veda per esempio [3]). Consideriamo un algoritmo online $A \in \mathcal{L}_k$, con $k > 1$. Se un avversario offline vuole vanificare l'utilità di un lookahead così "grande", può presentare le richieste in blocchi di k richieste identiche. In questo modo A di volta in volta è a conoscenza soltanto del *blocco* successivo di richieste: contro l'avversario offline, A ha lo stesso potere di un algoritmo dotato di lookahead di 1 richiesta. Tutto ciò è precisato dal seguente teorema.

Teorema 4.3 *Sia* $R(\mathcal{L}_k)$ *il rapporto di competitività dell'insieme degli algoritmi online con lookahead di* k *richieste, per il* L_N-OLTRP*, contro l'avversario ottimo offline. Risulta:*

$$R(\mathcal{L}_k) = R(\mathcal{L}_1) \ \forall k \in \mathbb{N}^+$$

Dimostrazione. Poiché $\mathcal{L}_k \supseteq \mathcal{L}_1$, allora evidentemente

$$R(\mathcal{L}_k) \leq R(\mathcal{L}_1) \ \forall k \in \mathbb{N}^+. \tag{4.1}$$

Rimane da dimostrare che $R(\mathcal{L}_k) \geq R(\mathcal{L}_1) \ \forall k \in \mathbb{N}^+$.

Definiamo una funzione $f : \mathcal{L}_k \to \mathcal{L}_1$ che associa, ad ogni algoritmo $A \in \mathcal{L}_k$, un algoritmo $A' = f(A) \in \mathcal{L}_1$ che si comporti non peggio di A: più precisamente, per quanto male si possa comportare A', A si comporta almeno altrettanto male.

A' opera nel seguente modo. Per prima cosa A' trasforma l'istanza di input ricevuta utilizzando la funzione $g : \mathcal{I} \to \mathcal{I}$ che ora definiamo. L'istanza di input $\sigma' = \sigma_1 \cdots \sigma_n$ di A' è trasformata nella sequenza

$$\sigma = g(\sigma') := \underbrace{\sigma_1 \ldots \sigma_1}_{k} \ldots \underbrace{\sigma_n \ldots \sigma_n}_{k} .$$

Successivamente A' esegue l'algoritmo A, fornendogli in input l'istanza σ; in output, A fornisce il giro T_σ. Il server mosso da A' percorre proprio T_σ servendo tutte le richieste che trova durante il percorso.

È evidente che T_σ è anche una soluzione del problema per l'istanza σ', e che inoltre vale la seguente relazione:

$$A(\sigma) = kA'(\sigma') ,$$

in quanto le richieste omonime sono servite con lo stesso ritardo da A e da A'.

Per quel che riguarda l'ottimo offline, è facile dimostrare che il percorso ottimo per servire σ coincide con il percorso ottimo per servire σ'. Pertanto vale la relazione seguente:

$$OPT(\sigma) = kOPT(\sigma') .$$

Dunque risulta che:

$$\frac{A(\sigma)}{OPT(\sigma)} = \frac{kA'(\sigma')}{kOPT(\sigma')} = \frac{A'(\sigma')}{OPT(\sigma')} \ \forall \sigma' \in \mathcal{I} . \tag{4.2}$$

Ora, supponiamo che $R(\mathcal{L}_1) = \rho \in \widetilde{\mathbb{R}}$. Nel seguito considereremo ρ finito; la dimostrazione si può adattare immediatamente anche al caso di ρ infinito.

Richiamiamo alla mente che per la definizione di rapporto di competitività $R(\mathcal{L}_k)$ si ottiene nel seguente modo: si sceglie $A \in \mathcal{L}_k$ in modo da minimizzare la quantità $\max_{\sigma \in \mathcal{I}} \frac{A(\sigma)}{OPT(\sigma)}$.

Scelto A, consideriamo $A' = f(A)$; poiché A' è, al meglio, ρ-competitivo, risulta che:

$$\forall \varepsilon \in \mathbb{R}^+ \ \exists \sigma' : \ \frac{A'(\sigma')}{OPT(\sigma')} \geq \rho - \varepsilon .$$

Ciò implica che

$$\forall \varepsilon \in \mathbb{R}^+ \ \exists \sigma = g(\sigma') : \ \frac{A(\sigma)}{OPT(\sigma)} = \frac{A'(\sigma')}{OPT(\sigma')} \geq \rho - \varepsilon ,$$

e quindi A non può essere meglio che ρ-competitivo. Dunque:

$$R(\mathcal{L}_k) \geq \rho = R(\mathcal{L}_1) \ \forall k \in \mathbb{N}^+ . \tag{4.3}$$

Dalla (4.1) e dalla (4.3) si ottiene l'asserto. □

Vedremo in seguito che la validità del Teorema 4.3 non si limita al problema che stiamo ora analizzando.

Risultati simili a quelli del Teorema 4.3 sono emersi dall'analisi dell'efficacia del lookahead per altri problemi online. Albers [2, 3] per i problemi del list update e dello scheduling ha proposto un modello di *lookahead forte* in cui l'algoritmo online, al tempo t, è a conoscenza di un insieme S_t di $k' \geq k$ richieste rilasciate immediatamente dopo l'istante t. S_t contiene esattamente k richieste *distinte*, cioè rilasciate in k "punti" distinti. Un modello simile di lookahead potrebbe essere definito per il L_N-OLTRP, ma per molti spazi

metrici non ci sarebbe alcun vantaggio: potremmo ripetere l'argomentazione della dimostrazione del Teorema 4.3, proponendo all'algoritmo dotato di lookahead forte blocchi di k richieste molto vicine tra di loro, ma distinte.

Per completare l'analisi di competitività degli algoritmi dotati di lookahead non rimane che da investigare sulla classe degli algoritmi con 1 richiesta di lookahead. Il risultato che si trova non è positivo.

Teorema 4.4 *Ogni algoritmo online A dotato di lookahead di* 1 *richiesta per il* L_N-OLTRP *definito su uno spazio metrico $M = (X, d) \in \mathcal{M}$, non è competitivo.*

Dimostrazione. Siano:

- $O \in X$ l'origine,
- $P \in X$ un punto tale che $d := d(O, P) > 0$,
- $p_S(t)$ la posizione del server mosso dall'algoritmo S nell'istante t.

L'avversario offline presenta esattamente $N + 1$ richieste, di cui anticipa la prima: $\sigma_1 = (d, P)$.

Le restanti N richieste sono rilasciate tutte nell'istante $d + \varepsilon$, quindi A viene a conoscenza di tali richieste tra l'istante d e l'istante $d+\varepsilon$. La posizione in cui esse sono rilasciate dipende dalla posizione del server mosso da A all'istante d, in modo tale che, per servirle, A debba percorrere una distanza di almeno $d/2$ a partire dall'istante d:

$$\sigma_2 = \ldots = \sigma_{N+1} = \begin{cases} (d+\varepsilon,\, P), & d(O, p_A(d)) \leq d/2; \\ (d+\varepsilon,\, O), & d(O, p_A(d)) > d/2. \end{cases}$$

È evidente che $A(\sigma) \geq (d/2 - \varepsilon)N$; l'algoritmo ottimo offline è in grado di servire le N ultime richieste nell'istante stesso in cui sono rilasciate, e poi passare a servire σ_1 con un ritardo massimo di $2d + \varepsilon$.

Pertanto risulta:

$$\frac{A(\sigma)}{OPT(\sigma)} \geq \frac{(d/2 - \varepsilon)N}{2d + \varepsilon}\,.$$

La tesi segue facendo $\varepsilon \to 0$, $N \to \infty$. □

4.2.2 Analisi di comparatività tra classi di algoritmi con lookahead

Abbiamo visto che nessun algoritmo online A è competitivo contro l'avversario ottimo offline, neanche se A è dotato di lookahead. L'analisi di competitività contro l'avversario ottimo offline non ci consente di apprezzare il vantaggio offerto dal lookahead, perché l'avversario è troppo potente rispetto a tutti gli algoritmi online, e in questo modo "appiattisce" la misura delle loro prestazioni.

Ora ci proponiamo di confrontare direttamente tra loro le prestazioni di algoritmi online dotati di un diverso grado di lookahead, o privi del tutto di lookahead.

Cercheremo il rapporto di comparatività $R(\mathcal{L}_h, \mathcal{L}_k)$ tra le classi degli algoritmi online dotati di lookahead di h e k richieste rispettivamente, per il L_N-OLTRP; si rimanda alla Sezione 2.3.6 per le nozioni fondamentali relative all'analisi di comparatività.

D'ora in avanti supporremo di avere $h < k$, in modo tale che $\mathcal{L}_h \subset \mathcal{L}_k$. Ricordiamo che per trovare un lower bound di $R(\mathcal{A}, \mathcal{B}) = \rho$ possiamo cercare un algoritmo $B \in \mathcal{B}$ che gode della seguente proprietà: per ogni $A \in \mathcal{A}$, esiste un'istanza $\sigma \in \mathcal{I}$ tale che:

$$\frac{A(\sigma)}{B(\sigma)} \geq b \; ;$$

in tal caso concluderemo che b è un lower bound per ρ, cioè che $b \leq \rho$.

Utilizzeremo questa tecnica sia per confrontare $\mathcal{L}_1$ con l'insieme $\mathcal{L}_0$ degli algoritmi online privi di lookahead, sia per confrontare $\mathcal{L}_k$ con $\mathcal{L}_h$ nel caso in cui $0 < h < k$; in entrambi i casi troveremo un lower bound infinito. Iniziamo dal secondo confronto.

Teorema 4.5 *Sia $\mathcal{L}_n$ la classe degli algoritmi online dotati di lookahead di n richieste per il L_N-OLTRP, definito su uno spazio metrico $M \in \mathcal{M}$; siano inoltre h, k due naturali tali che $0 < h < k$. Allora*

$$R(\mathcal{L}_h, \mathcal{L}_k) = +\infty \; .$$

Dimostrazione. Per alleggerire la notazione dimostreremo il teorema sullo spazio metrico segmento $[-1, 1]$, ma la dimostrazione si estende immediatamente a qualsiasi spazio metrico in $\mathcal{M}$, usando la tecnica vista nella dimostrazione del Teorema 3.23.

Indichiamo con $p_S(t)$ la posizione del server mosso dall'algoritmo S nell'istante t.

Vogliamo trovare un lower bound infinito al rapporto di comparatività; per fare questo scegliamo un algoritmo $B \in \mathcal{L}_k$ che determinerà il valore

del lower bound. La descrizione di B sarà data più in avanti nel corso della dimostrazione, per il momento lo si consideri un algoritmo fissato.

Sia $A \in \mathcal{L}_h$ un qualsiasi algoritmo che "sfida" B. Per rendere grande il rapporto $\frac{A(\sigma)}{B(\sigma)}$, dobbiamo scegliere un'opportuna istanza $\sigma \in \mathcal{I}$ (si può immaginare che σ sia scelta da B per mettere in difficoltà A). L'istanza σ che scegliamo è composta o da $h+1$ richieste, oppure da $h+1+N$ richieste, a seconda del comportamento di A. Le prime $h+1$ richieste non dipendono da A, e sono le seguenti:

$$\begin{aligned} \sigma_1 &= (0,\, 0)\,; \\ \sigma_2 = \ldots = \sigma_{h+1} &= (1,\, 1)\,. \end{aligned}$$

Fino all'istante 1, A conoscerà le richieste $\sigma_1, \ldots, \sigma_{h+1}$ ed esse soltanto.
Se $p_A(1) \leq 0$, non sono rilasciate altre richieste; dunque $A(\sigma) \geq h$.
Se $p_A(1) > 0$, invece, rilasciamo altre N richieste:

$$\sigma_{h+1+1} = \ldots = \sigma_{h+1+N} = (1+\varepsilon,\, -1)\,.$$

Ora, l'algoritmo B è in grado di conoscere se ci troviamo nel caso $p_A(1) \leq 0$ oppure nel caso $p_A(1) > 0$, in quanto il suo lookahead maggiore consente di sapere, fin dall'istante 0, se ci sono $k-h$ delle ultime N richieste rilasciate nel punto -1, oppure no. L'idea è quella di costruire B in modo tale che serva queste ultime richieste nell'istante stesso in cui sono rilasciate, se sono presenti, perché sono molto numerose (faremo $N \to \infty$); altrimenti, B serve le h richieste nell'istante in cui sono rilasciate.

Più formalmente, l'algoritmo B ha il seguente comportamento:

- se c'è una richiesta in $(0,\, 0)$, la serve immediatamente;
- se nella sua coda di lookahead S_0 relativa all'istante 0 c'è una richiesta nel punto -1, allora muove subito il server in -1, e rimane fermo in tale punto servendo le richieste ivi presentate, fino all'istante t in cui la coda di lookahead S_t non contiene alcuna richiesta nel punto -1;
- per il resto, B serve le richieste nel medesimo ordine in cui esse arrivano.

Se $p_A(1) \leq 0$ allora, come abbiamo visto, $A(\sigma) \geq h$, mentre $B(\sigma) = 0$; quindi il rapporto tra $A(\sigma)$ e $B(\sigma)$ è infinito.

Se invece $p_A(1) > 0$, allora $A(\sigma) > N$, mentre $B(\sigma) = (2+\varepsilon)h$, e:

$$\lim_{N\to\infty} \lim_{\varepsilon\to 0^+} \frac{N}{(2+\varepsilon)h} = +\infty\,.$$

□

Teorema 4.6 *Sia* $\mathcal{L}_n$ *la classe degli algoritmi online dotati di lookahead di* n *richieste per il* L_N-OlTRP*, definito su uno spazio metrico* $M \in \mathcal{M}$*. Vale la seguente relazione:*

$$R(\mathcal{L}_0, \mathcal{L}_1) = +\infty .$$

Dimostrazione. Anche qui per alleggerire la notazione dimostreremo il teorema sullo spazio metrico segmento $[-1, 1]$.

Indichiamo con $p_S(t)$ la posizione del server mosso dall'algoritmo S nell'istante t.

Vogliamo trovare un lower bound infinito al rapporto di comparatività; per fare questo scegliamo un algoritmo $B \in \mathcal{L}_1$ che determinerà il valore del lower bound. L'algoritmo B semplicemente serve le richieste nello stesso ordine in cui sono rilasciate, muovendosi verso una richiesta appena possibile (cioè appena è a conoscenza dell'esistenza della richiesta, e non ci sono richieste più vecchie da servire).

Sia $A \in \mathcal{L}_0$ un qualsiasi algoritmo online privo di lookahead, che consideriamo lo sfidante di B. A partire da A decidiamo l'istanza di input σ, che contiene esattamente una richiesta:

$$\sigma_1 = \begin{cases} (1, -1), & p_A(1) \geq 0\,; \\ (1, 1), & p_A(1) < 0\,. \end{cases}$$

Chiaramente $A(\sigma) \geq 1$, mentre B serve σ_1 nell'istante in cui è presentata, per cui $B(\sigma) = 0$. □

Corollario 4.7 *Se* $k \in \mathbb{N}^+$*, allora:* $R(\mathcal{L}_0, \mathcal{L}_k) = +\infty$*.*

Dimostrazione. È sufficiente osservare che $\mathcal{L}_1 \subseteq \mathcal{L}_k$. □

Ciò che maggiormente sorprende dei Teoremi 4.5 e 4.6 è la loro dimostrazione: in entrambi i casi per trovare un lower bound a $R(\mathcal{A}, \mathcal{B})$ abbiamo scelto per la classe più generale $\mathcal{B}$ un algoritmo "campione" B, rispetto al quale abbiamo misurato le prestazioni di ogni algoritmo A della classe meno generale $\mathcal{A}$. B è risultato infinitamente migliore di A per opportune istanze di input. Eppure in tutte e due le dimostrazioni l'algoritmo "campione" B è decisamente stupido. È stato costruito in modo da sfruttare il lookahead per apprendere informazioni su un input particolarmente malevolo (per A) fin dal principio. Ma è chiaro che B si comporta bene solo sulle istanze di input che sono state utilizzate ai fini della dimostrazione.

Questa osservazione ci fa riflettere sul valore dei risultati che abbiamo trovato: come possiamo affermare che gli algoritmi di $\mathcal{B}$ sono migliori di

quelli di $\mathcal{A}$, se per dimostrare questa affermazione confrontiamo tutti gli algoritmi di $\mathcal{A}$ con un algoritmo di $\mathcal{B}$ chiaramente mediocre? Ci sembra di poter affermare che l'analisi di comparatività non si presta a questo tipo di confronto, per il L_N-OlTRP.

Concludiamo con un'altra osservazione. Per dimostrare la superiorità degli algoritmi dotati di lookahead rispetto a quelli che ne sono privi è stato sufficiente farli competere su un'istanza di input σ contenente esattamente una richiesta; per il confronto di due classi di algoritmi con lookahead, invece, abbiamo fatto ricorso a istanze di input con un numero arbitrariamente grande di richieste. Questo ci suggerisce che:

- potrebbe esistere una differenza tra algoritmi con e algoritmi senza lookahead più sostanziale della differenza tra classi di algoritmi con lookahead, anche se né l'analisi di competitività né quella di comparatività sono in grado di coglierla;
- con un problema dial-a-ride differente, in cui la funzione obiettivo sia meno influenzata dal numero $|\sigma|$ di richieste presenti nell'istanza σ (per esempio il massimo tempo di servizio), questa differenza potrebbe emergere proprio dall'analisi di comparatività.

4.3 Algoritmi con lookahead di una finestra temporale

Ci occupiamo ora dell'altro modello di lookahead, quello in cui all'algoritmo online A sono note tutte le richieste che saranno rilasciate in una finestra temporale di ampiezza Δ. Anche qui per non appesantire troppo la notazione ci riferiremo a questo modello di lookahead semplicemente con il termine "lookahead", ed indicheremo la classe $\mathcal{L}_\Delta^{(T)}$ con $\mathcal{L}_\Delta$.

Per dare significato all'ampiezza Δ della finestra di lookahead occorre confrontare tale valore con un'altra grandezza. La scelta più naturale è quella di confrontare Δ con il diametro D dello spazio metrico: vedremo che se lo spazio metrico non è limitato, ossia $D = +\infty$, allora non esistono algoritmi competitivi in $\mathcal{L}_\Delta$; indagheremo poi su quello che avviene al variare di Δ nel caso in cui $D \in \mathbb{R}^+$.

Osserviamo che, se $\Delta_1 \leq \Delta_2$, allora $\mathcal{L}_{\Delta_1} \subseteq \mathcal{L}_{\Delta_2}$; pertanto un qualsiasi lower bound sul rapporto di competitività di $\mathcal{L}_{\Delta_2}$ è anche un lower bound sul rapporto di competitività di $\mathcal{L}_{\Delta_1}$.

Nel seguito troveremo un lower bound infinito per una classe abbastanza ampia di algoritmi, ossia per un insieme $\mathcal{L}_{\Delta_2}$ in cui l'ampiezza Δ_2 della finestra di lookahead è piuttosto elevato rispetto al diametro D. Questo implica

che il lower bound infinito è valido anche per tutti i valori inferiori $\Delta_1 \leq \Delta_2$ di ampiezza della finestra. Tuttavia non affronteremo subito il problema così in generale. Dimostreremo il lower bound infinito prima per le classi più piccole di algoritmi online; questo ci permetterà di impiegare tecniche di dimostrazione più semplici, e di conseguenza più facili da generalizzare a problemi simili.

4.3.1 Lower bound nel caso $\Delta < D/2$

Teorema 4.8 *Sia A un algoritmo online dotato di lookahead in una finestra temporale di ampiezza Δ per il problema del riparatore viaggiatore online con la funzione obiettivo della latenza netta, definito su uno spazio metrico $M = (X, d) \in \mathcal{M}$ di diametro $D > 0$. Se $\Delta < D/2$, allora A non è competitivo.*

Dimostrazione. Indichiamo con $p_S(t)$ la posizione del server mosso dall'algoritmo S nell'istante t.

Per ipotesi per ogni $\varepsilon \in \mathbb{R}^+$ esistono due punti $P, Q \in X$ tali che $d(P, Q) > D - \varepsilon$. Scegliamo ε sufficientemente piccolo affinché risulti

$$\Delta < \frac{D}{2} - \varepsilon \,. \tag{4.4}$$

L'avversario offline presenta un'istanza di input σ contenente un'unica richiesta, che dipende dalla posizione del server mosso da A nell'istante $D-\Delta$; lo scopo è quello di posizionare la richiesta "lontano" dalla posizione del server:

$$\sigma_1 = \begin{cases} (D, Q), & d(P, p_A(D-\Delta)) \leq D/2; \\ (D, P), & d(P, p_A(D-\Delta)) > D/2. \end{cases}$$

L'algoritmo ottimo offline è in grado di servire la richiesta nell'istante D in cui è rilasciata.

Invece A arriva sicuramente in ritardo nel punto in cui è rilasciata σ_1: l'algoritmo online viene a conoscenza della richiesta nell'istante $D - \Delta$. A partire da quest'istante, se σ_1 è rilasciata in P allora A impiega almeno $D/2$ per raggiungerla. Dunque A raggiunge il punto P almeno nell'istante:

$$(D - \Delta) + D/2 > D/2 + D/2 = D \,.$$

Se invece σ_1 è rilasciata in Q, allora A impiega per raggiungerla almeno $(D - \varepsilon) - D/2 = D/2 - \varepsilon$, e quindi il punto Q è raggiunto non prima dell'istante:

$$(D - \Delta) + (D/2 - \varepsilon) = D + \underbrace{((D/2 - \varepsilon) - \Delta)}_{>0 \text{ (per la (4.4))}} > D \,.$$

Il teorema è così dimostrato. □

Corollario 4.9 *Sia A un algoritmo online dotato di lookahead in una finestra temporale di ampiezza* Δ *per il problema del riparatore viaggiatore online con la funzione obiettivo della latenza netta, definito su uno spazio metrico non limitato. Allora A non è competitivo.*

4.3.2 Lower bound nel caso $\Delta < D$

Teorema 4.10 *Sia A un algoritmo online dotato di lookahead in una finestra temporale di ampiezza* Δ *per il problema del riparatore viaggiatore online, con la funzione obiettivo della latenza netta, definito su uno spazio metrico* $M = (X, d) \in \mathcal{M}$ *di diametro* $D > 0$*. Se* $\Delta < D$*, allora A non è competitivo.*

Dimostrazione. Indichiamo con $p_S(t)$ la posizione del server mosso dall'algoritmo S nell'istante t.

Per ogni $\varepsilon \in \mathbb{R}^+$ esistono due punti $P, Q \in X$ tali che $d(P, Q) > D - \varepsilon$. Scegliamo ε sufficientemente piccolo in modo tale che risulti:

$$d := d(P, Q) > \Delta\,.$$

L'avversario offline presenta un'istanza di input σ contenente esattamente $N + 1$ richieste. La prima richiesta è la seguente:

$$\sigma_1 = (D, P)\,.$$

Sia T l'istante di tempo in cui A servirebbe la richiesta σ_1 se essa fosse l'unica richiesta dell'istanza di input. Le rimanenti $N > 0$ richieste di σ sono le seguenti:

$$\sigma_2 = \cdots = \sigma_{N+1} = (T + \Delta, Q)\,.$$

Poiché queste ultime richieste sono note ad A solamente a partire dall'istante T, allora $p_A(T) = P$. Dunque A può trovarsi in Q per servire le ultime N richieste non prima dell'istante $T + d$; ne segue che:

$$A(\sigma) > (d - \Delta)N > 0\,.$$

Un avversario offline (non migliore dell'ottimo) può servire le ultime N richieste nell'istante stesso in cui sono rilasciate, e poi passare a servire σ_1 nell'istante $T + \Delta + d$: pertanto

$$OPT(\sigma) \leq (T + \Delta + d) - D\,,$$

e la tesi è dimostrata poiché

$$\lim_{N\to\infty} \frac{(d-\Delta)N}{T+\Delta+d-D} = +\infty\,.$$

□

Come abbiamo preannunciato all'inizio della Sezione 4.3, sono stati esposti due teoremi, il Teorema 4.8 e il Teorema 4.10, dei quali il primo è meno generale, essendo implicato dal secondo. D'altra parte, per dimostrare il lower bound infinito nel caso in cui $\Delta < D/2$ è stato sufficiente far competere gli algoritmi su un'istanza formata da una richiesta; quando abbiamo aumentato l'ampiezza della finestra di lookahead, abbiamo fatto ricorso ad un'istanza di input di lunghezza tendente all'infinito. Dunque il primo caso può essere facilmente generalizzato a problemi in cui la quantità di richieste dell'istanza influenza meno direttamente la funzione obiettivo.

Ci chiediamo che cosa succeda quando Δ cresce ulteriormente. Possiamo ancora generalizzare il nostro lower bound, ma occorre affinare la tecnica di dimostrazione.

4.3.3 Lower bound nel caso $\Delta < 2D$

Quando l'ampiezza Δ della finestra di lookahead si mantiene al di sotto di 2 volte il diametro D dello spazio metrico, possiamo ancora dimostrare un lower bound infinito. Procederemo per gradi, trovando dapprima un (piccolo) lower bound facendo uso di un'istanza composta da tre richieste; poi faremo vedere come si può aumentare indefinitamente il lower bound aggiungendo nuove richieste all'istanza.

Per semplificare la trattazione utilizziamo, come spazio metrico, il segmento $[-1, 1]$ di diametro $D = 2$; i risultati possono essere immediatamente generalizzati a qualsiasi spazio metrico limitato, procedendo in modo simile a quanto abbiamo visto nelle dimostrazioni dei teoremi precedenti.

Prima di entrare nel vivo della questione, abbiamo bisogno di alcuni strumenti che ci aiuteranno a trattarla più agevolmente.

Sia S un qualsiasi algoritmo per L_N-OLTRP. Indicheremo con $C_S(\sigma)$ il *tempo di completamento* dell'algoritmo S con input σ, ossia l'istante in cui S serve l'ultima richiesta di σ.

Il secondo strumento è la latenza netta parziale. Intuitivamente, la latenza netta parziale dell'algoritmo S nell'istante T rappresenta il contributo dato alla latenza netta da tutto ciò che è accaduto fino all'istante T. In altre parole, la latenza netta parziale è composta da:

- la somma dei tempi impiegati per il servizio delle richieste già servite in T;
- la somma dei tempi trascorsi da quando sono state rilasciate le richieste non ancora servite.

Diamone ora la definizione formale.

Definizione 4.11 (Latenza netta parziale) *Sia S un algoritmo per il L_N-OLTRP, $\sigma \in \mathcal{I}$ un'istanza di input, $T \in \mathbb{R}^+$ un istante di tempo. Supponiamo che sia $\sigma = \sigma_1 \dots \sigma_n$, e che la richiesta $\sigma_i = (t_i, x_i)$ sia servita da S nell'istante τ_i. Chiameremo* latenza netta parziale *di S con input σ all'istante T la quantità:*

$$S(\sigma, T) := \sum_{i:\, \tau_i \leq T} (\tau_i - t_i) + \sum_{i:\, (\tau_i > T) \wedge (t_i \leq T)} (T - t_i)\,.$$

Ora siamo pronti ad affrontare il lower bound nel caso $\Delta < 2D = 4$.

Sia $A \in \mathcal{L}_\Delta$ un algoritmo online con lookahead $\Delta \in [2, 4[$. Si presti attenzione al fatto che abbiamo scelto $\Delta \geq 2$ per rendere più agevole la dimostrazione: utilizzeremo questa disuguaglianza in molti passaggi, senza citarla esplicitamente. Come abbiamo fatto notare all'inizio della Sezione 4.3, il lower bound che troveremo sarà valido anche per valori più piccoli di Δ.

Consideriamo l'istanza di input $\sigma^{(3)} = \sigma_1\sigma_2\sigma^*$, di cui le prime due richieste sono:

$$\begin{aligned} \sigma_1 &= (4,\, 1); \\ \sigma_2 &= (4,\, -1). \end{aligned}$$

Supponiamo, senza restrizione di generalità, che A serva σ_1 per prima, nell'istante T ($T \geq 4$). Rilasciamo la terza richiesta nell'istante $T + \Delta$:

$$\sigma^* = (T + \Delta,\, 1).$$

L'algoritmo ottimo offline serve le richieste nell'ordine $\sigma_2\sigma_1\sigma^*$ ottenendo

$$OPT(\sigma^{(3)}) = 0 + 2 + \underbrace{0}_{\Delta \geq 2} = 2\,.$$

Osserviamo inoltre che nell'istante $T + \Delta$, OPT ha terminato di servire le richieste di $\sigma^{(3)}$, ovvero che

$$C_{OPT}(\sigma^{(3)}) = T + \Delta\,.$$

Quando A viene a conoscere σ^*, nell'istante T, ha due possibilità: può servire prima σ_2 e poi σ^*, oppure prima σ^* e poi σ_2.

Nel primo caso ($\sigma_2\sigma^*$), A raggiunge il punto -1 (almeno) nell'istante $T+2$, e ritorna nel punto 1 (almeno) nell'istante $T+4$. Il valore minimo di $A(\sigma^{(3)})$ si ottiene per $T=4$; pertanto risulta:

$$A(\sigma^{(3)}) \geq 0 + 2 + ((T+4) - (T+\Delta)) = 2 + (4-\Delta)\,.$$

Sempre analizzando il caso ($\sigma_2\sigma^*$), facciamo alcune considerazioni di cui inizialmente non si comprenderà l'importanza, che sarà chiarita nel seguito. Indichiamo con T' l'istante in cui A serve la penultima richiesta σ_2 di $\sigma^{(3)}$. Stimiamo la latenza netta parziale di A in T', utilizzando il fatto che $T \geq 4$ e $T' \geq T+2$. Ai nostri fini, nella stima è sufficiente considerare il contributo delle richieste σ_1 e σ_2:

$$A(\sigma^{(3)}, T') \geq (T-4) + (T'-4) \geq 2\,.$$

Infine, calcoliamo il valore della quantità $C_{OPT}(\sigma^{(3)}) - T'$:

$$C_{OPT}(\sigma^{(3)}) - T' \leq \underbrace{(T+\Delta)}_{C_{OPT}} - \underbrace{(T+2)}_{T' \geq T+2} = \Delta - 2\,.$$

Affrontiamo ora il secondo caso ($\sigma^*\sigma_2$), quello in cui A serve prima σ^* e poi σ_2. A attende nel punto 1 almeno fino all'istante $T+\Delta$ e poi si sposta in -1, dove arriva non prima dell'istante $T+\Delta+2$. Anche qui il valore minimo di $A(\sigma^{(3)})$ si ottiene quando $T=4$, quindi:

$$A(\sigma^{(3)}) \geq 0 + 0 + ((T+\Delta+2)-4) \overset{(T\geq 4)}{\geq} 2 + \Delta \overset{(\Delta \geq 2)}{\geq} 2 + (4-\Delta)\,;$$

per quanto riguarda la latenza netta parziale nell'istante $T' \geq T+\Delta$ in cui A serve la penultima richiesta σ^* di $\sigma^{(3)}$, ci interessa soltanto il contributo dato da σ_2: la richiesta è rilasciata in 4 e in T' ancora non è stata servita, quindi

$$A(\sigma^{(3)}, T') \geq (T'-4) \geq \Delta \geq 2\,.$$

Ed ora calcoliamo:

$$C_{OPT}(\sigma^{(3)}) - T' \leq (T+\Delta) - (T+\Delta) = 0 \leq \Delta - 2\,.$$

Questo ragionamento ci ha permesso di stabilire che A ottiene sempre una latenza netta almeno di $4-\Delta$ superiore a quella dell'ottimo offline. Non sembra molto, ma iterando il procedimento si può aumentare indefinitamente questo scarto, come sarà mostrato fra breve.

Per il momento riassumiamo il risultato che abbiamo ottenuto nella seguente proposizione.

Proposizione 4.12 *Sia A un qualsiasi algoritmo online dotato di lookahead in una finestra temporale di ampiezza Δ per il problema del riparatore viaggiatore online con la funzione obiettivo della latenza netta, definito sul segmento $[-1, 1]$. Se $2 \leq \Delta < 4$, allora esiste una sequenza di input $\sigma^{(3)}$, $|\sigma^{(3)}| = 3$, che gode delle seguenti proprietà:*

- *Ogni richiesta di $\sigma^{(3)}$ è rilasciata nel punto -1 oppure nel punto 1; in particolare, la penultima e l'ultima richiesta servite da A si trovano in due punti distinti;*
- *$OPT(\sigma^{(3)}) = 2$;*
- *$A(\sigma^{(3)}) \geq 2 + (4 - \Delta) \quad [= 2 + (3 - 2)(4 - \Delta)]$;*
- *se A serve la penultima richiesta di $\sigma^{(3)}$ nell'istante T, allora $C_{OPT}(\sigma^{(3)}) - T \leq \Delta - 2$;*
- *infine $A(\sigma^{(3)}, T) \geq 2 \quad [= 2 + (3 - 3)(4 - \Delta)]$.*

A questo punto iteriamo il procedimento che abbiamo seguito. L'idea di base è la seguente: ogniqualvolta l'algoritmo online A serve la penultima richiesta di cui attualmente è a conoscenza, l'avversario ne aggiunge una nuova nel punto stesso in cui si trova A, a distanza di tempo di Δ. OPT è in grado di servire questa nuova richiesta nell'istante stesso in cui è rilasciata, dunque la sua latenza netta non aumenta; mentre A, che "viaggia" con un ritardo di almeno $4 - \Delta$ rispetto ad OPT (il ritardo è stato accumulato nel servizio delle prime tre richieste), serve la nuova richiesta almeno $4 - \Delta$ dopo il suo tempo di rilascio, aumentando così il valore della propria latenza netta.

Più formalmente, vale il seguente teorema.

Teorema 4.13 *Sia A un qualsiasi algoritmo online dotato di lookahead temporale in una finestra di ampiezza Δ per il problema del riparatore viaggiatore online con la funzione obiettivo della latenza netta, definito sul segmento $[-1, 1]$. Se $2 \leq \Delta < 4$, allora per ogni $n \in \mathbb{N}$, $n \geq 3$, esiste una sequenza di input $\sigma^{(n)}$, $|\sigma^{(n)}| = n$, che gode delle seguenti proprietà:*

- *Ogni richiesta di $\sigma^{(n)}$ è rilasciata nel punto -1 oppure nel punto 1; in particolare, la penultima e l'ultima richiesta servite da A si trovano in due punti distinti;*
- *$OPT(\sigma^{(n)}) = 2$;*
- *$A(\sigma^{(n)}) \geq 2 + (n - 2)(4 - \Delta)$;*

- *se A serve la penultima richiesta di $\sigma^{(n)}$ nell'istante T, allora $C_{OPT}(\sigma^{(n)}) - T \leq \Delta - 2$;*
- *infine $A(\sigma^{(n)}, T) \geq 2 + (n-3)(4-\Delta)$.*

Dimostrazione. Procediamo per induzione.

Passo base ($n = 3$). Coincide con la Proposizione 4.12.

Passo induttivo ($n = h + 1$). La tesi è vera per $n = h$. Ordiniamo le richieste di $\sigma^{(h)} = \sigma_1 \cdots \sigma_h$ secondo l'ordine nel quale esse sono servite da A.

Supponiamo che la penultima richiesta σ_{h-1} sia servita da A nell'istante T e che, senza restrizione di generalità, si trovi nel punto 1.

L'ultima richiesta σ_h si trova nel punto -1.

Costruiamo la nuova sequenza $\sigma^{(h+1)} = \sigma^{(h)}\sigma^*$ a partire da $\sigma^{(h)}$ aggiungendo la richiesta

$$\sigma^* = (T + \Delta,\, 1)\,.$$

OPT per ipotesi induttiva può servire tutte le richieste di $\sigma^{(h)}$ terminando nell'istante $T - 2 + \Delta$, e quindi servire σ^* nell'istante stesso $T + \Delta$ in cui è rilasciata, senza aumentare la sua latenza netta. Pertanto valgono le seguenti relazioni:

$$OPT(\sigma^{(h+1)}) = OPT(\sigma^{(h)}) + 0 = 2\,;$$
$$C_{OPT}(\sigma^{(h+1)}) = T + \Delta\,.$$

Passiamo all'algoritmo online A. Poiché σ^* è nota ad A solamente a partire dall'istante T, fino a tale istante il server di A si muove come se stesse servendo $\sigma^{(h)}$.

Dopo l'istante T, A ha due possibilità: servire prima σ_h e poi σ^*, oppure prima σ^* e poi σ_h. In ogni caso, queste due sono la penultima e l'ultima richiesta di $\sigma^{(h+1)}$ servite da A, e si trovano in due punti distinti.

Nel primo caso ($\sigma_h\sigma^*$), le due richieste sono servite rispettivamente negli istanti $\tau_h \geq T + 2$ e $\tau^* \geq T + 4$, e sono state rilasciate rispettivamente negli istanti $t_h \leq C_{OPT}(\sigma^{(h)})$ (perché (ovviamente!) OPT serve le richieste *dopo* che sono state rilasciate, e *poi* termina il servizio) e $t^* = T + \Delta$. Stimiamo ora il contributo fornito a $A(\sigma^{(h+1)})$ da ciascuna delle richieste di $\sigma^{(h+1)}$. Il contributo delle richieste $\sigma_1, \ldots, \sigma_{h-1}$ è interamente contenuto in $A(\sigma^{(h)}, T)$; per il resto, $A(\sigma^{(h)}, T)$ può contenere soltanto una parte del contributo fornito da σ_h, la parte relativa al tempo trascorso tra il rilascio di σ_h e T. Il contributo *complessivo* fornito a $A(\sigma^{(h+1)})$ da σ_h è $\tau_h - t_h \geq (T+2) - C_{OPT}(\sigma^{(h)}) \geq (T+2) - (T+\Delta-2) = (T+2) - \bar{T}$. In quest'ultimo termine abbiamo posto $\bar{T} := (T + \Delta - 2)$ per osservare che $\bar{T} \geq T$:

ciò significa che questo termine contiene soltanto una parte del contributo fornito da σ_h, che si riferisce a quello che avviene *dopo* l'istante T; quindi questa parte del contributo non è presente in $A(\sigma^{(h)}, T)$, che può contenere soltanto la parte del contributo di σ_h *fino* all'istante T. Infine, il contributo della richiesta σ^* è $\tau^* - t^* \geq (T+4) - (T+\Delta)$. Pertanto risulta:

$$\begin{aligned} A(\sigma^{(h+1)}) &\geq \underbrace{A(\sigma^{(h)}, T)}_{\text{contributo di tutte le richieste in } [0,T]} + \underbrace{((T+2) - (T+\Delta-2))}_{\text{contributo di } \sigma_h \text{ dopo } T} \\ &+ \underbrace{((T+4) - (T+\Delta))}_{\text{contributo di } \sigma^*} \\ &\geq A(\sigma^{(h)}, T) + (4-\Delta) + (4-\Delta) \\ &\geq 2 + (h-3)(4-\Delta) + (4-\Delta) + (4-\Delta) \\ &= 2 + ((h+1)-2)(4-\Delta)\,. \end{aligned}$$

In particolare si osservi che, se $T' = \tau_h$ è l'istante in cui è servita la penultima richiesta σ_h di $\sigma^{(h+1)}$, allora una stima della latenza netta parziale in T' (che tiene conto solo delle richieste $\sigma_1, \ldots, \sigma_h$, il che è sufficiente ai nostri scopi) risulta:

$$\begin{aligned} A(\sigma^{(h+1)}, T') &\geq \underbrace{A(\sigma^{(h)}, T)}_{\text{contributo di tutte le richieste in } [0,T]} + \underbrace{((T+2) - (T-2+\Delta))}_{\text{contributo di } \sigma_h \text{ dopo } T} \\ &\geq 2 + (h-3)(4-\Delta) + (4-\Delta) \\ &= 2 + ((h+1)-3)(4-\Delta)\,. \end{aligned}$$

Per finire, $C_{OPT}(\sigma^{(h+1)}) - T' \leq (T+\Delta) - (T+2) = \Delta - 2$. La tesi è stata dimostrata nella sua interezza.

Nel secondo caso ($\sigma^*\sigma_h$), A può servire σ^* nell'istante stesso in cui è rilasciata $(T+\Delta)$, e poi si sposta nel punto -1 che raggiunge non prima dell'istante $T+\Delta+2$. Procedendo come nel primo caso, stimiamo:

$$\begin{aligned} A(\sigma^{(h+1)}) &\geq \underbrace{A(\sigma^{(h)}, T)}_{\text{contributo di tutte le richieste in } [0,T]} + \underbrace{0}_{\text{contributo di } \sigma^*} \\ &+ \underbrace{((T+\Delta+2) - (T-2+\Delta))}_{\text{contributo di } \sigma_h \text{ dopo } T} \\ &= A(\sigma^{(h)}, T) + 4 \\ &\geq A(\sigma^{(h)}, T) + 2(4-\Delta) \\ &\geq 2 + (h-3)(4-\Delta) + 2(4-\Delta) \\ &= 2 + ((h+1)-2)(4-\Delta)\,. \end{aligned}$$

Se $T' = \tau^*$ è l'istante in cui A serve la penultima richiesta σ^*, allora la latenza netta parziale risulta:

$$\begin{aligned} A(\sigma^{(h+1)}, T') &\geq \underbrace{A(\sigma^{(h)}, T)}_{\text{contributo di tutte le richieste in } [0,T]} + \underbrace{0}_{\text{contributo di } \sigma^*} \\ &+ \underbrace{(T+\Delta) - (T-2+\Delta)}_{\text{contributo di } \sigma_h \text{ dopo } T \text{ e prima di } T'} \\ &= A(\sigma^{(h)}, T) + 2 \\ &\geq A(\sigma^{(h)}, T) + (4-\Delta) \\ &\geq 2 + (h-3)(4-\Delta) + (4-\Delta) \\ &= 2 + ((h+1)-3)(4-\Delta)\,. \end{aligned}$$

Ora dobbiamo soltanto calcolare: $C_{OPT}(\sigma^{(h+1)}) - T' \leq (T+\Delta) - (T+\Delta) = 0 \leq \Delta - 2$.

Il teorema è stato dimostrato. □

Corollario 4.14 *Sia A un algoritmo online dotato di lookahead in una finestra temporale di ampiezza Δ per il problema del riparatore viaggiatore con minimizzazione della latenza netta, definito sullo spazio metrico $[-1, 1]$. Se $\Delta < 4$, allora A non è competitivo.*

4.3.4 Lower bound in $\mathbb{R}^n$, $n \geq 2$, per ogni Δ

Nella precedente Sezione 4.3.3 abbiamo trovato un lower bound valido per qualsiasi spazio metrico. Per dimostrarne l'esistenza abbiamo fatto sì che il server online servisse, di volta in volta, la richiesta per lui più svantaggiosa: infatti, ogni volta che il server online visita un punto per servire una richiesta, allora una nuova richiesta è rilasciata a breve distanza temporale nel medesimo punto.

La breve distanza temporale deve essere comunque maggiore o uguale al lookahead Δ; quindi è "breve" nel senso che, prima che la nuova richiesta sia effettivamente rilasciata, il server online non è in grado di servire le altre richieste attive e di ritornare nel punto della nuova richiesta. Per ottenere ciò abbiamo introdotto delle richieste situate in punti sufficientemente distanti rispetto a Δ. Questo è il motivo per cui il Corollario 4.14 è valido finché il diametro dello spazio metrico è sufficientemente grande (più di $\Delta/2$).

Se lo spazio metrico è un aperto di $\mathbb{R}^n$, $n \geq 2$, possiamo sfruttare un'altra idea per rendere "breve" la distanza temporale alla quale rilasciamo di volta in volta la nuova richiesta, qualunque sia il rapporto tra Δ e il diametro

dello spazio metrico. L'idea è quella di rilasciare richieste in parecchi punti distinti. Per esempio, se sono attive richieste in $2N$ punti distinti, a due a due separati da una distanza di almeno d, allora per servire tutte le richieste è necessario un tempo maggiore o uguale a $2dN$. Nella dimostrazione del teorema che segue vedremo che è possibile rendere $2dN$ grande a piacere, in modo tale che la distanza temporale a cui rilasciamo le nuove richieste sia "breve" rispetto a $2dN$.

Teorema 4.15 *Sia Ω un aperto qualsiasi di $\mathbb{R}^n$, con $n \geq 2$; sia A un algoritmo online per il problema del riparatore viaggiatore online con la funzione obiettivo della latenza netta, definito sullo spazio metrico Ω dotato della distanza euclidea, con un lookahead in una finestra temporale di ampiezza $\Delta \in \mathbb{R}^+$. Allora, qualunque sia il valore di $\Delta \in \mathbb{R}^+$, A non è competitivo.*

Dimostrazione. Come di consueto, indichiamo con $O \in \Omega$ il punto (origine) in cui i server si trovano nell'istante 0; indichiamo con $p_X(t)$ la posizione del server mosso dall'algoritmo X nell'istante t.

Dimostreremo il teorema nel caso bidimensionale, ossia supponendo che $\Omega \subseteq \mathbb{R}^2$. Nel caso pluridimensionale è sufficiente considerare l'intersezione di Ω con un piano bidimensionale passante per l'origine, ottenendo così un insieme isomorfo ad un aperto di $\mathbb{R}^2$, e lavorare su tale insieme.

Ci limiteremo a rilasciare le richieste in un intorno dell'origine $B_O(\frac{\sqrt{2}}{2}a) \subseteq \Omega$ di raggio $\frac{\sqrt{2}}{2}a < \Delta$ (vedi Figura 4.1). Vogliamo disporre le richieste sui punti di una griglia quadrata $k \times k$ di lato al più a in modo tale che il server debba impiegare un tempo di almeno 2Δ per toccare tutti i punti della griglia. La distanza minima tra due punti è di $\frac{a}{k}$. Quindi per toccare tutti i punti della griglia un server impiega almeno $k^2\frac{a}{k} = ka$. Prendendo $k \geq \lceil \frac{2\Delta}{a} \rceil$, (e, per comodità, k pari), e aggiustando il lato del quadrato ad $a' := \frac{2\Delta}{k} \leq a$, abbiamo risolto il problema.

Nel seguito per comodità indicheremo con $N := \frac{k^2}{2}$ la metà del numero dei punti della griglia; osserviamo che per visitare N punti occorre un tempo di almeno Δ, e che la distanza minima di due punti della griglia è di Δ/N.

Assumendo senza restrizione di generalità che l'origine abbia coordinate $(0, 0)$, per $i, j \in \{-k/2, \ldots, -2, -1, +1, +2, \ldots, +k/2\}$ indichiamo con P_{ij} il punto $P_{ij} := (f(i), f(j))$ (vedi Figura 4.1), dove:

$$f(i) := \begin{cases} i\frac{a'}{k} - \frac{a'}{2k}, & 1 \leq i \leq \frac{k}{2}; \\ i\frac{a'}{k} + \frac{a'}{2k}, & -\frac{k}{2} \leq i \leq -1. \end{cases}$$

Confronteremo la prestazione di A con quella di B, l'algoritmo dell'avversario offline. L'avversario ha anche il compito di costruire l'istanza di input.

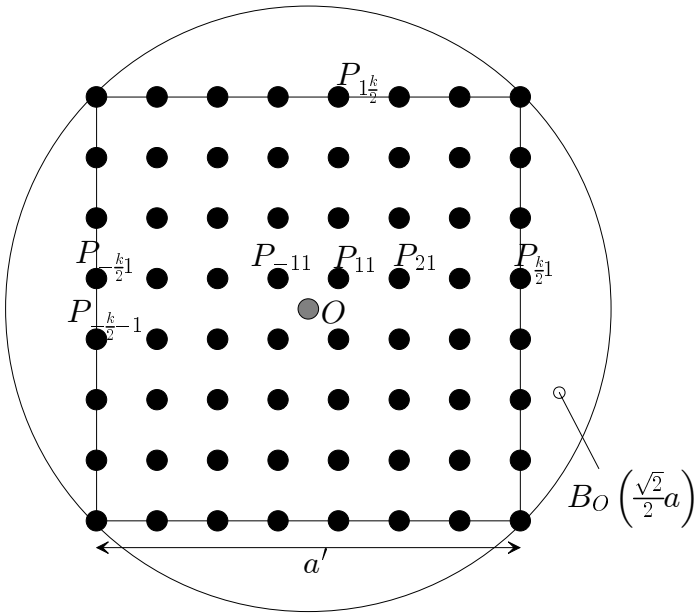

Figura 4.1: La griglia dei punti in cui sono rilasciate le richieste.

Si noti che B, pur essendo un algoritmo offline, non necessariamente risolve il problema di ottimizzazione in modo ottimo.

Nell'intervallo $[\Delta, 2\Delta[$ l'avversario rilascia una sequenza σ_0 di richieste che chiameremo *richieste iniziali*. La sequenza σ_0 ha la proprietà di contenere almeno una richiesta in ciascuno dei punti P_{ij}.

Si supponga che A serva per prime le richieste di σ_0 rilasciate "sotto" l'origine, ovvero nei punti P_{ij} con $j < 0$, e che la prima richiesta servita da A sia situata in $P_{-\frac{k}{2}-1}$; si supponga inoltre che, nell'istante 2Δ, A abbia terminato di servire tutte e sole le richieste situate sotto l'origine. Chiamiamo questa ipotesi *ipotesi* ***H***. In seguito faremo vedere che l'ipotesi **H** non è arbitraria, in quanto è possibile scegliere una sequenza di richieste iniziali σ_0 tale che, se A non verifica l'ipotesi **H**, allora A non è competitivo. Per il momento anticipiamo soltanto che, nella sequenza σ_0, tutte le richieste rilasciate "sopra" l'origine sono rilasciate nell'istante Δ.

Oltre alle richieste iniziali, l'avversario rilascia altre richieste a partire dall'istante 2Δ. Per ogni t, $2\Delta \leq t \leq t_{stop}$, se A serve almeno una richiesta nell'istante $t - \Delta$, allora l'avversario rilascia esattamente una richiesta $(t, p_A(t - \Delta))$. Dal punto di vista di A questo significa che ogni volta che A serve una o più richieste, l'avversario rilascia una nuova richiesta nel medesimo punto in cui A si trova, ad una distanza temporale di esattamente Δ; A è immediatamente a conoscenza della nuova richiesta grazie al lookahead di

Δ. Dunque in ogni istante t, $\Delta \leq t \leq t_{stop} - \Delta$, A vede almeno una richiesta (già rilasciata oppure nella coda di lookahead) in ciascuno dei $2N$ punti distinti P_{ij}. L'avversario termina di rilasciare richieste nell'istante t_{stop}, che determineremo in seguito.

Calcoliamo ora il costo pagato da A. Sia A_0 il costo pagato da A per servire le richieste iniziali. Per calcolare un lower bound al costo pagato da A per servire le altre richieste, dividiamo l'asse dei tempi in finestre di ampiezza 2Δ e calcoliamo il contributo al costo complessivo fornito dalle richieste che, all'inizio di ciascuna finestra, sono state già rilasciate ma non ancora servite (*richieste attive*), oppure sono nella coda di lookahead.

Sia $t_i := 2\Delta i$. Per $i \geq 1$, calcoliamo il contributo dato dalle richieste attive o in lookahead nell'istante t_i relativamente all'intervallo $[t_i, t_{i+1}[$ di ampiezza 2Δ. Nell'istante t_i sono note ad A almeno $2N$ richieste, situate in esattamente $2N$ punti distinti. Nell'istante $t_i + \Delta$ ciascuna di queste richieste sarà divenuta attiva, ed eventualmente servita. Osserviamo però che, nell'intervallo $[t_i, t_i + \Delta[$, A riesce a servire richieste in al più N punti distinti. Quindi nell'istante $t_i + \Delta$ sono attive richieste in almeno N punti. Per servire la j-esima di queste richieste, A impiega un tempo di almeno $(j-1)\frac{\Delta}{N}$, per un costo complessivo di almeno

$$\frac{\Delta}{N}\sum_{j=1}^{N}(j-1) = \frac{\Delta}{N}\frac{(N-1)N}{2} = \frac{(N-1)\Delta}{2}.$$

Osserviamo che non tutte le $2N$ richieste sono necessariamente servite da A entro l'istante t_{i+1}; il costo che abbiamo calcolato è relativo unicamente all'intervallo $[t_i, t_{i+1}[$, ma le eventuali richieste rimanenti fanno parte anche dell'insieme delle $2N$ richieste note ad A nell'istante t_{i+1}, e quindi contribuiscono anche al costo dell'intervallo successivo $[t_{i+1}, t_{i+2}[$.

Dunque, supponendo che $t_{stop} = t_m$, il costo in cui incorre A è:

$$A(\sigma) \geq A_0 + (m-1)\frac{(N-1)\Delta}{2} \geq (m-1)\frac{(N-1)\Delta}{2},$$

e può essere reso grande a piacere agendo su m.

Vediamo ora come si comporta B, l'algoritmo dell'avversario, sempre supponendo che l'ipotesi **H** sia verificata. In tal caso:

1. nell'intervallo $[0, \Delta[$, B porta il server sulla prima richiesta che dovrà servire;

2. nell'intervallo $[\Delta, 2\Delta[$, B serve tutte le richieste che si trovano "sopra" l'origine, terminando con la richiesta situata in $P_{-\frac{k}{2}1}$, e poi sposta il

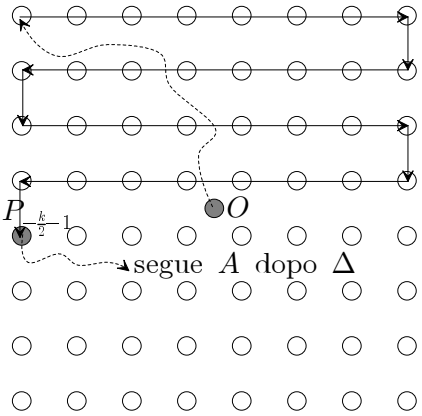

Figura 4.2: Il percorso del server mosso da B.

server nel punto $P_{-\frac{k}{2}-1}$, immediatamente sottostante. È immediato verificare che ciò è possibile mediante una semplice scansione orizzontale dei punti della griglia, a cominciare dall'alto e scendendo, riga dopo riga, verso l'origine (vedi Figura 4.2);

3. nell'istante 2Δ, il server mosso da B si trova nel punto $P_{-\frac{k}{2}-1}$; qui attende l'istante t_{start} tale che, nell'istante $t_{start} - \Delta$, A ha servito le richieste posizionate proprio nel punto $P_{-\frac{k}{2}-1}$ (si ricordi che questo è il primo punto visitato dal server mosso da A);

4. infine, a partire dall'istante t_{start}, B segue A con un ritardo di Δ, ossia: $p_B(t) = p_A(t-\Delta)$; B serve tutte le richieste attive che incontra nel suo tragitto.

Osserviamo che nell'istante 3Δ tutte le richieste iniziali sono state servite da B, perché nell'istante 2Δ tutte le richieste iniziali sotto l'origine sono state servite da A. Questo significa che il costo B_0 del servizio delle richieste iniziali è già noto nell'istante 2Δ, e quindi che B_0 non dipende da m (ossia da t_{stop}).

D'altra parte B_0 è l'unico costo pagato da B. Si ricordi infatti che B è in ritardo di Δ rispetto ad A, e che le richieste sono rilasciate nei punti visitati da A dopo esattamente Δ: quindi B riesce a servire tutte le richieste non-iniziali nell'istante stesso in cui sono rilasciate. Dunque abbiamo:

$$B(\sigma) = B_0.$$

Pertanto risulta:

$$\frac{A(\sigma)}{OPT(\sigma)} \geq \frac{A(\sigma)}{B(\sigma)} \geq \frac{(n-1)\frac{(N-1)\Delta}{2}}{B_0},$$

che può essere reso grande a piacere agendo su n. Il teorema è dimostrato.

Ci rimane da far vedere che esiste una sequenza di richieste iniziali σ_0 tale che, se l'ipotesi **H** non è verificata, allora A non è competitivo. Come abbiamo preannunciato, tutte le richieste iniziali sopra l'origine sono rilasciate nell'istante Δ; in particolare, rilasciamo esattamente una richiesta in ogni punto della griglia.

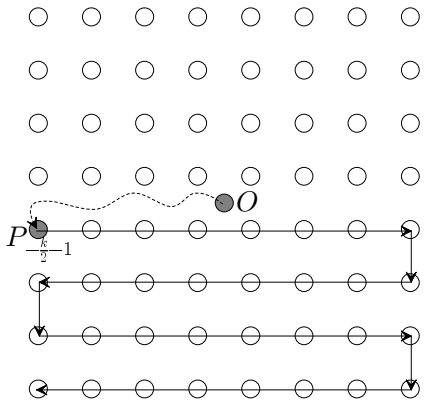

Figura 4.3: Il percorso del server virtuale A_v.

Per quanto riguarda le richieste iniziali sotto l'origine, procediamo come segue. Costruiamo il percorso che vorremmo far percorrere al server mosso da A, muovendo un server virtuale A_v (vedi Figura 4.3). A_v parte dal primo punto $P_{-\frac{k}{2}-1}$, che visita nell'istante Δ, e scandisce le righe della griglia ad una ad una, procedendo verso il basso, e senza mai fermarsi; si vede immediatamente che A_v termina il proprio percorso nell'istante $2\Delta - \Delta/N$. In ogni punto della griglia sotto l'origine rilasciamo M richieste nell'istante in cui il server virtuale A_v lo visita.

Ora, se il server mosso da A non verifica l'ipotesi **H**, esso visita almeno un punto della griglia sotto l'origine con un ritardo di almeno Δ/N rispetto ad A_v; quindi $A(\sigma) \geq M\Delta/N$. Sempre se l'ipotesi **H** non è verificata, l'avversario si comporta nel seguente modo:

1. rilascia richieste non-iniziali così come abbiamo descritto in precedenza, con $t_{stop} = 3\Delta$;

2. B serve per prime le richieste iniziali situate sotto l'origine, nell'istante in cui sono rilasciate, semplicemente seguendo il percorso di A_v; poi attende l'istante 3Δ e infine, con una scansione completa della griglia, serve tutte le richieste ancora attive, che sono al più $2N$, terminando nell'istante 5Δ.

Dunque B paga non più di $2N(5\Delta - \Delta) = 8N\Delta$, e quindi

$$\frac{A(\sigma)}{OPT(\sigma)} \geq \frac{A(\sigma)}{B(\sigma)} \geq \frac{\frac{M\Delta}{N}}{8N\Delta} = \frac{M}{8N^2},$$

che può essere reso grande a piacere agendo su M. □

4.4 Estensioni e problemi aperti

4.4.1 Il potere del lookahead per il L_N-OlTRP: problemi aperti

In questo capitolo abbiamo visto che non esistono algoritmi online dotati di lookahead che siano competitivi per il L_N-OlTRP, nelle seguenti situazioni:

- sempre, quando il lookahead è di $k \in \mathbb{N}^+$ richieste;
- sempre, quando il lookahead riguarda una finestra temporale di ampiezza Δ e lo spazio metrico è isomorfo a un aperto di $\mathbb{R}^n$, $n \geq 2$;
- per $\Delta < 2D$, quando il lookahead riguarda una finestra temporale di ampiezza Δ, dove D è il diametro dello spazio metrico.

Ancora non si conoscono lower bound ed upper bound sui rapporti di competitività degli algoritmi online dotati di lookahead in una finestra temporale di ampiezza $\Delta \geq 2D$, quando lo spazio metrico è generale; casi particolarmente interessanti sono il segmento e i grafi. A proposito dei grafi, osserviamo che il Teorema 4.15 non si estende ai grafi a griglia: infatti la sua dimostrazione usa un'argomentazione che si basa sulla densità $\mathbb{R}^2$, ovvero sul fatto che, in ogni aperto di $\mathbb{R}^2$, l'avversario ha la possibilità di infittire a piacimento i punti delle richieste; al contrario, in un grafo a griglia i punti sono fissati dallo spazio metrico.

Sotto l'ipotesi di ragionevolezza del carico (vedi Sezione 3.3.2) è immediato dimostrare che non esistono algoritmi online competitivi: perciò in letteratura le prestazioni degli algoritmi online sono state misurate in un modo

differente, confrontandole con il valore Δ_0 per cui il carico è Δ_0-ragionevole. Invece si può tentare di utilizzare l'analisi di competitività per studiare gli algoritmi online dotati di lookahead; potrebbero emergere interessanti relazioni che coinvolgono il rapporto di competitività, il diametro D dello spazio metrico, l'ampiezza Δ della finestra di lookahead e il valore Δ_0.

4.4.2 Il potere del lookahead per il $F_{\max}$-OLTRP: estensione della validità di alcuni risultati

Sostituendo alla latenza netta la funzione obiettivo del massimo tempo di servizio, alcuni dei risultati presentati in questo capitolo continuano ad essere validi, mentre per gli altri la dimostrazione perde di significato e occorre affrontare il problema in maniera diversa. La differenza principale tra le due funzioni obiettivo è costituita dal numero di richieste per le quali l'algoritmo deve pagare: per una, la più "sfortunata", nel caso del massimo tempo di servizio; per tutte quante, nel caso della latenza netta. Osserviamo inoltre che, quando $|\sigma| = 1$, allora la latenza netta e il massimo tempo di servizio di ogni soluzione coincidono: quindi ogni dimostrazione che si basa su un'istanza di input composta da una sola richiesta continua ad essere valida nel caso del massimo tempo di servizio.

4.4.2.1 Lookahead in termini di richieste

Il rapporto di competitività delle classi $\mathcal{L}_h^{(R)}$ e $\mathcal{L}_k^{(R)}$ degli algoritmi online dotati di lookahead di h e k richieste rispettivamente, h, $k \in \mathbb{N}^+$ è il medesimo; infatti il Teorema 4.3 si estende facilmente:

Teorema 4.16 *Sia* $R\left(\mathcal{L}_k^{(R)}\right)$ *il rapporto di competitività dell'insieme degli algoritmi online con lookahead di* k *richieste, per il* $F_{\max}$-OLTRP*, contro l'avversario ottimo offline. Risulta:*

$$R\left(\mathcal{L}_k^{(R)}\right) = R\left(\mathcal{L}_1^{(R)}\right) \ \forall k \in \mathbb{N}^+$$

Dimostrazione. La dimostrazione è la stessa di quella del Teorema 4.3, con la seguente eccezione. Quando si calcolano i costi degli algoritmi, risulta:

$$A(\sigma) = A'(\sigma')$$

e

$$OPT(\sigma) = OPT(\sigma').$$

Da ciò si ricava che

$$\frac{A(\sigma)}{OPT(\sigma)} = \frac{A'(\sigma')}{OPT(\sigma')} \ \forall \sigma' \in \mathcal{I}$$

che coincide con la (4.2); tutto quello che segue continua ad essere valido. □

Invece la dimostrazione del Teorema 4.4 non si estende al caso del massimo tempo di servizio, perché si basa sulla seguente tecnica: tramite un numero di richieste N opportunamente grande, l'avversario fa pagare un costo arbitrariamente grande all'algoritmo online. Questa tecnica, che chiamiamo *tecnica della forza del numero*, evidentemente non si può applicare con il massimo tempo di servizio. Dunque non è noto il rapporto di competitività delle classi $\mathcal{L}_k^{(R)}$, $\forall k \in \mathbb{N}^+$, anche se è noto che tutte queste classi possiedono il medesimo rapporto di competitività.

Per quanto riguarda l'analisi di comparatività di classi di algoritmi dotati di lookahead, osserviamo che il Teorema 4.6 è valido anche per il massimo tempo di servizio, in quanto nella sua dimostrazione l'avversario rilascia un'istanza σ composta da una sola richiesta:

Teorema 4.17 *Sia $\mathcal{L}_n^{(R)}$ la classe degli algoritmi online dotati di lookahead di n richieste per il $F_{\max}$-*OLTRP*, definito su uno spazio metrico $M \in \mathcal{M}$. Vale la seguente relazione:*

$$R\left(\mathcal{L}_0^{(R)}, \mathcal{L}_1^{(R)}\right) = +\infty \ .$$

Invece la dimostrazione del Teorema 4.5 non si può estendere, basandosi sulla tecnica della forza del numero di richieste. Dunque non è noto il rapporto di comparatività $R\left(\mathcal{L}_h^{(R)}, \mathcal{L}_k^{(R)}\right)$ per $0 < h < k$.

Rimarchiamo che quanto esposto in questa sezione è in linea con l'osservazione con cui abbiamo chiuso la Sezione 4.2.2, in cui si congettura che la presenza del lookahead offra un vantaggio, rispetto all'assenza completa di lookahead, "migliore" del vantaggio offerto da un maggiore grado di lookahead.

4.4.2.2 Lookahead in una finestra temporale

Consideriamo il problema del riparatore viaggiatore online definito su uno spazio metrico limitato $M \in \mathcal{M}$ di diametro $D \in \mathbb{R}^+$. Se la funzione obiettivo è il massimo tempo di servizio, per gli algoritmi online dotati di lookahead in una finestra temporale di ampiezza Δ vale il seguente teorema:

Teorema 4.18 *Sia A un algoritmo online dotato di lookahead in una finestra temporale di ampiezza Δ per il problema del riparatore viaggiatore online con la funzione obiettivo della massimo tempo di servizio, definito su uno spazio metrico $M = (X, d) \in \mathcal{M}$ di diametro $D > 0$. Se $\Delta < D/2$, allora A non è competitivo.*

Dimostrazione. È identica alla dimostrazione del Teorema 4.8, nella quale l'avversario rilascia una sola richiesta. □

Tutti gli altri risultati relativi alla funzione obiettivo della latenza netta, cioè la non esistenza di algoritmi competitivi quando $D/2 \leq \Delta < 2D$ e, in aperti di $\mathbb{R}^2$, per ogni valore di Δ, sono stati dimostrati facendo ricorso a delle tecniche non riutilizzabili nel caso del massimo tempo di servizio. Per questa importante funzione obiettivo, quindi, rimangono aperti molti problemi.

4.4.3 La latenza netta con costi di servizio

Vogliamo chiudere il capitolo con un accenno ad un'altra funzione obiettivo, la latenza netta con costi di servizio. Il Teorema 3.25 afferma che, se lo spazio metrico è limitato, non esistono algoritmi online competitivi per il problema del riparatore viaggiatore con questa funzione obiettivo. Al contrario, quando lo spazio metrico è limitato è possibile trovare degli algoritmi competitivi. Per esempio consideriamo il classico algoritmo ASCENSORE per lo spazio metrico del segmento:

Algoritmo 4.19 (ASCENSORE) *Il server mosso da* ASCENSORE *si muove sempre a velocità massima, senza mai invertire il verso di marcia, se non quando raggiunge un estremo del segmento; serve tutte le richieste che trova lungo il percorso.*

È banale osservare che ASCENSORE è competitivo per il problema del riparatore viaggiatore quando la funzione obiettivo è la latenza netta con costi di servizio:

Teorema 4.20 ASCENSORE *è un algoritmo $\left(1 + \frac{2D}{TS}\right)$-competitivo per il problema del riparatore viaggiatore online definito su un segmento di lunghezza D, con la funzione obiettivo della latenza netta con costo di servizio pari a TS.*

Dimostrazione. Sia $\sigma \in \mathcal{I}$ una qualsiasi istanza di input. Per ogni richiesta di σ, OPT paga almeno il costo di servizio, quindi $OPT(\sigma) \geq |\sigma|TS$. ASCENSORE serve ogni richiesta con un ritardo non superiore a $2D$, per cui ASCENSORE$(\sigma) \leq |\sigma|(2D + TS)$. Il rapporto di competitività segue

immediatamente. Il valore è stretto: è sufficiente considerare un'istanza σ composta da esattamente una richiesta, rilasciata in un estremo del segmento subito dopo che il server ha lasciato l'estremo stesso. □

Osserviamo che questo teorema vale anche se sostituiamo alla latenza netta con costi di servizio la funzione obiettivo del massimo tempo di servizio con costi di servizio, ossia la quantità $\max_i \{(\tau_i - t_i) + TS\}$.

Ulteriori indagini potrebbero portare ad interessanti risultati di competitività per algoritmi online, eventualmente dotati di lookahead.

Capitolo 5

Analisi sperimentale delle prestazioni di algoritmi per il L_N-OlTRP

Nel capitolo precedente abbiamo visto che tramite l'analisi di competitività è molto difficile valutare la qualità di un algoritmo online per il problema del riparatore viaggiatore online con la funzione obiettivo della latenza netta, anche se l'algoritmo è dotato di lookahead.

In questo capitolo ci proponiamo di compiere delle indagini sperimentali, confrontando le prestazioni di diversi algoritmi online dotati di lookahead, tra di loro e rispetto alle prestazioni dell'algoritmo ottimo offline. Gli algoritmi online sono parametrizzati rispetto all'ampiezza della finestra di lookahead di cui dispongono. In questo modo potremo valutare sperimentalmente il vantaggio offerto dal lookahead.

Prima di tutto descriviamo il tipo di esperimenti compiuti, per poi passare ai risultati. Per una descrizione tecnica della struttura e del funzionamento del programma di simulazione si rimanda all'Appendice A.

5.1 Gli esperimenti compiuti

5.1.1 I problemi

I problemi che abbiamo considerato sono i problemi del riparatore viaggiatore online con la funzione obiettivo della latenza netta, definiti sugli spazi metrici del segmento unitario $[-1/2,\, 1/2]$ e del quadrato con diametro unitario $\left[-\sqrt{2}/4,\, \sqrt{2}/4\right]^2$.

Abbiamo scelto questi spazi metrici in quanto semplici esempi di spazi

metrici limitati, con diametro unitario, che siano sottoinsiemi di $\mathbb{R}$ e di $\mathbb{R}^2$ rispettivamente.

5.1.2 Gli algoritmi

Nelle simulazioni abbiamo utilizzato l'algoritmo ottimo offline e quattro algoritmi online dotati di lookahead in una finestra temporale di ampiezza $\Delta \geq 0$.

Gli algoritmi online sono delle varianti dei noti algoritmi REPLAN e IGNORE. In entrambi i casi l'algoritmo originale contiene un passo in cui compie un'ottimizzazione, utilizzando la porzione di input di cui attualmente dispone; fondamentalmente questa ottimizzazione consiste nella risoluzione di un'istanza di un particolare problema dial-a-ride. Dunque possiamo fornire una descrizione dell'algoritmo parametrizzata rispetto al problema dial-a-ride che esso utilizza nel passo di ottimizzazione. In particolare, il problema è un elemento $X \in \{\text{TSP, TRP}\}$, cioè uno tra il problema del commesso viaggiatore online e il problema del riparatore viaggiatore online minimizzante la latenza netta. Si noti che quello che distingue i due problemi è unicamente la funzione obiettivo utilizzata.

Algoritmo 5.1 (Δ-REPLAN$_X$) *L'algoritmo* Δ-REPLAN$_X$*, con* $\Delta \geq 0$*, è un algoritmo dotato di lookahead in una finestra temporale di ampiezza* Δ*. Se all'algoritmo non sono note richieste non ancora servite, il server rimane fermo. Altrimenti, ogni volta che l'algoritmo viene a conoscere una nuova richiesta (nell'istante* $t_i - \Delta$ *se la richiesta è rilasciata in* t_i*), l'algoritmo calcola il giro* T_X *che gli permette di servire tutte le richieste note ma non ancora servite, minimizzando la funzione obiettivo del problema* X*. Quindi il server inizia a seguire* T_X*.*

Il nome REPLAN deriva dal fatto che l'algoritmo, quando viene a conoscere una nuova richiesta, smette di percorrere il giro attuale T_X ed esegue una nuova pianificazione.

Algoritmo 5.2 (Δ-IGNORE$_X$) *L'algoritmo* Δ-IGNORE$_X$*, con* $\Delta \geq 0$*, è un algoritmo dotato di lookahead in una finestra temporale di ampiezza* Δ*. In ogni istante l'algoritmo si trova in uno stato di* $\{IDLE, READY, TOUR\}$*; descriviamo il suo comportamento in ciascuno degli stati.*

- $IDLE$*. Nello stato* $IDLE$ *il server è fermo. Appena l'algoritmo viene a conoscenza dell'esistenza di almeno una richiesta che ancora non è stata servita, entra nello stato* $READY$*.*

- *READY. Quando l'algoritmo si trova nello stato READY, il server è fermo, ed almeno una richiesta non servita è nota all'algoritmo. L'algoritmo calcola il giro* T_X *che gli permette di servire tutte le richieste note ma non ancora servite, minimizzando la funzione obiettivo del problema X. Inoltre calcola l'istante* t_{start} *tale che, partendo nell'istante* t_{start} *e procedendo a velocità unitaria, il server è in grado di servire la prima richiesta di* T_X *nell'istante in cui essa è rilasciata. Ogni volta che l'algoritmo viene a conoscere una nuova richiesta, aggiorna il giro ottimo* T_X *e l'istante* t_{start}*. Nel primo istante* $t \geq t_{start}$*, l'algoritmo passa nello stato TOUR.*

- *TOUR. Nello stato TOUR il server percorre il giro ottimo* T_X *servendo tutte le richieste appena possibile. Se nel frattempo viene a conoscenza di una nuova richiesta, la ignora. Quando il giro* T_X *è terminato, ritorna nello stato IDLE.*

Osserviamo che gli algoritmi 0-REPLAN$_{\text{TSP}}$ e 0-IGNORE$_{\text{TSP}}$, privi di lookahead, coincidono rispettivamente con REPLAN e IGNORE, di cui abbiamo parlato nella Sezione 3.3.2 e le cui prestazioni in presenza di un carico di lavoro ragionevole sono state studiate da Hauptmeier et al. in [11].

5.1.3 Le istanze di input

Nelle simulazioni ogni istanza di input σ è stata scelta in modo casuale, tenendo conto delle seguenti considerazioni sul costo computazionale degli algoritmi utilizzati:

- L'algoritmo di ottimizzazione offline è di costo esponenziale in $|\sigma|$. Infatti per il L_N-OLTRP non sono noti algoritmi polinomiali neppure se lo spazio metrico è la retta, ed il problema è NP-hard in spazi metrici più generali. Per di più, per il L_N-OLTRP attualmente non sono noti algoritmi approssimati.

- Ognuno degli algoritmi online impiegati contiene un passo di ottimizzazione, che abbiamo realizzato tramite un algoritmo di costo esponenziale nel numero di richieste note ma non ancora servite. Se questo numero si mantiene limitato, il costo complessivo dell'algoritmo online è lineare nel numero totale di richieste; altrimenti esso diventa esponenziale.

Nel confrontare le prestazioni di algoritmi online con quelle dell'ottimizzatore offline abbiamo dovuto utilizzare istanze di input di piccola taglia (composte, al più, da circa venti richieste). Nel confrontare direttamente

le prestazioni di algoritmi online è stato possibile aumentare la taglia delle istanze, aumentando l'ampiezza dell'intervallo di tempo simulato, e mantenendo costante (e non eccessivamente elevato) il numero medio di richieste rilasciate per unità di tempo.

Per l'estrazione delle richieste il simulatore utilizza due parametri T e δ. T rappresenta la durata dell'intervallo di tempo (simulato) in cui sono rilasciate le richieste; δ la "densità temporale" delle richieste, cioè il numero medio di richieste rilasciate per unità di tempo. I valori per T e δ forniti al modulo di estrazione delle richieste possono essere fissati dall'utente, oppure fatti variare all'interno di un dato intervallo, oppure ancora estratti secondo una distribuzione di probabilità fissata dall'utente. Nelle simulazioni abbiamo optato per le prime due scelte.

Successivamente il simulatore estrae le $T\delta$ richieste. Per ogni richiesta estrae:

- l'istante di tempo in cui essa è rilasciata, ossia un punto nell'intervallo di ampiezza T, secondo una fissata distribuzione di probabilità (in genere, uniforme nell'intervallo);
- il punto in cui la richiesta è rilasciata. Nel caso del segmento unitario, il punto della richiesta è una variabile aleatoria uniformemente distribuita in $[-1/2,\ 1/2]$; analogamente, nel caso del quadrato a diametro unitario ciascuna coordinata del punto della richiesta è una variabile aleatoria uniformemente distribuita in $\left[-\sqrt{2}/4,\ \sqrt{2}/4\right]$

5.1.4 Le statistiche

Una volta stabiliti tutti i parametri del problema (fondamentalmente, Δ, T e δ) il simulatore estrae N istanze di input, dove N è un valore che si è ritenuto sufficientemente grande ($N = 100$). Per ogni istanza σ, calcola il costo della soluzione relativa all'istanza σ fornita da ciascuno degli algoritmi sotto indagine; quindi, per ogni coppia di algoritmi A, B di cui si vogliono confrontare le prestazioni, calcola il rapporto $\frac{A(\sigma)}{B(\sigma)}$. Infine calcola il valore minimo, medio e massimo di questi rapporti su tutte le N istanze.

5.2 I risultati ottenuti

5.2.1 Confronto con l'ottimo offline

La Figura 5.1 mostra il risultato del confronto delle prestazioni di $\text{REPLAN}_{\text{TRP}}$ e $\text{IGNORE}_{\text{TRP}}$ con quelle dell'algoritmo ottimo offline, sullo spazio metrico del segmento untitario. I parametri utilizzati per la simulazione

sono la durata $T = 7$, e la densità temporale $\delta = 2.5$. Il lookahead è fatto variare tra 0 e 1.2, e compare in ascissa nel grafico in figura. In ordinata sono riportati i valori minimi, medi e massimi del rapporto di prestazioni $\frac{OPT(\sigma)}{A(\sigma)}$, dove A è l'algoritmo online di volta in volta utilizzato. Si noti che questo rapporto è il reciproco del più usuale $\frac{A(\sigma)}{OPT(\sigma)}$ che compare nella definizione dell'analisi di competitività.

Dal grafico emerge che, nella situazione considerata, REPLAN$_{\text{TRP}}$ si comporta molto bene nella pratica, significativamente meglio di IGNORE$_{\text{TRP}}$; in entrambi i casi, il lookahead contribuisce in modo significativo al miglioramento delle prestazioni, che è considerevole soprattutto nella prima parte del grafico, corrispondente a valori di lookahead compresi tra 0 e 0.6 circa. Per esempio il valore medio di $\frac{OPT(\sigma)}{\text{REPLAN}_{\text{TRP}}(\sigma)}$ varia tra un minimo di circa 0.45 in assenza di lookahead, che corrisponde a un rapporto di competitività medio di circa 2.2, a un valore di circa 0.91 quando il lookahead vale 0.6, corrispondente a un rapporto di competitività medio di circa 1.1.

Anche nel caso migliore e nel caso peggiore REPLAN$_{\text{TRP}}$ si è comportato meglio di IGNORE$_{\text{TRP}}$; sebbene la teoria affermi che non esistono algoritmi competitivi per il L_N-OLTRP sulla retta quando $\Delta < 2D$ (nel nostro caso, $D = 1$), tuttavia non sono state generate istanze in cui gli algoritmi online abbiano conseguito un rapporto di prestazioni nullo. In particolare, gli algoritmi dotati di lookahead $\Delta \geq 0.1$ nelle 1200 istanze generate hanno conseguito, nel caso peggiore, un rapporto di competitività di circa 5 (in figura, per $\Delta = 0.5$).

Se lo spazio metrico utilizzato è il quadrato a diametro unitario e gli altri parametri rimangono immutati, la situazione non cambia molto, come è evidenziato dalla Figura 5.2.

È naturale domandarsi che cosa accade cambiando i parametri di T e δ. Per quanto riguarda la durata T, manteniamo fissa la densità $\delta = 2.5$ e facciamo variare T fra 4 e 10. Con un lookahead di 0.5 si ottiene la situazione in Figura 5.3, in cui si può constatare che il rapporto di prestazioni medio si mantiene abbastanza costante (di poco inferiore a 0.9 per REPLAN$_{\text{TRP}}$, circa 0.75 per IGNORE$_{\text{TRP}}$). Purtroppo non sappiamo che cosa accade per valori molto più grandi di T, a causa della complessità computazionale del problema di ottimizzazione.

Facendo variare la densità media δ tra 1.5 e 4, nel caso in cui $T = 5$ e $\Delta = 0.5$, otteniamo la situazione descritta dal grafico in Figura 5.4. In questo caso REPLAN$_{\text{TRP}}$ sembra influenzato pochissimo dalla densità delle richieste, che comunque sembra svolgere un ruolo lievemente positivo sulle sue prestazioni. Invece, le prestazioni di IGNORE$_{\text{TRP}}$ diminuiscono quando δ aumenta, anche se in misura non eccessiva. Anche in questo caso sareb-

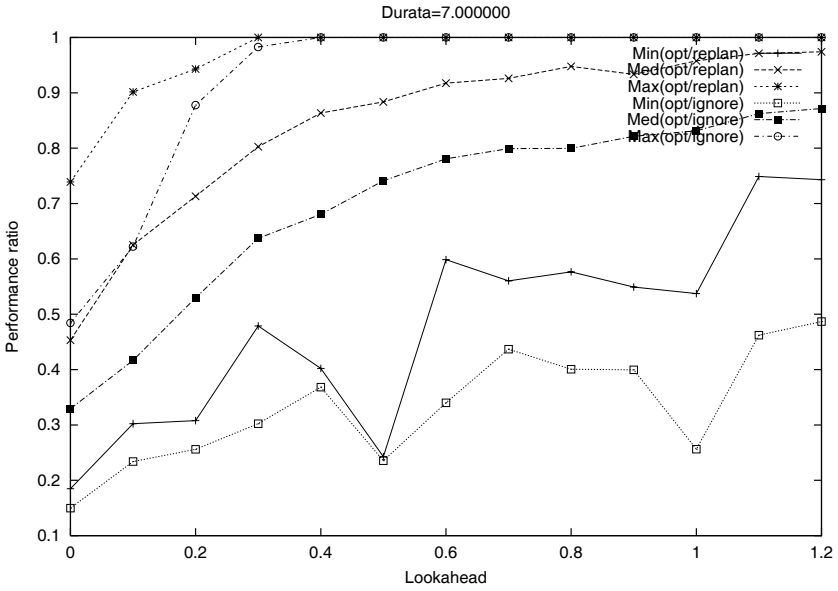

Figura 5.1: Confronto sperimentale delle prestazioni di OPT con $\text{REPLAN}_{\text{TRP}}$ (in figura, `replan`) e $\text{IGNORE}_{\text{TRP}}$ (in figura, `ignore`), sul segmento unitario.

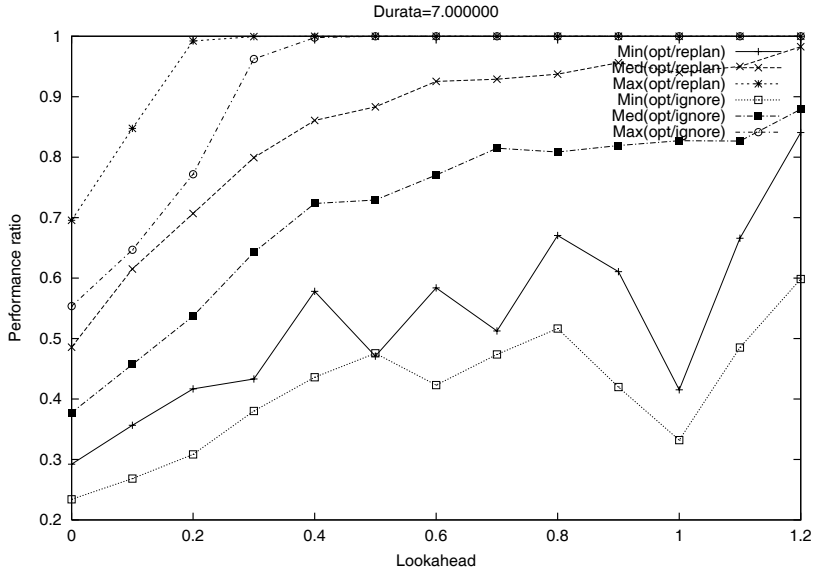

Figura 5.2: Confronto sperimentale delle prestazioni di OPT con $\text{REPLAN}_{\text{TRP}}$ (in figura, `replan`) e $\text{IGNORE}_{\text{TRP}}$ (in figura, `ignore`), sul quadrato a diametro unitario.

be interessante estendere l'intervallo di indagine, il che è contrastato dalla complessità computazionale del problema.

Passiamo ora a considerare gli algoritmi $\text{REPLAN}_{\text{TSP}}$ e $\text{IGNORE}_{\text{TSP}}$, per i quali effetueremo test analoghi a quelli compiuti per $\text{REPLAN}_{\text{TRP}}$ e $\text{IGNORE}_{\text{TRP}}$. La Figura 5.5 mostra, al variare del lookahead, il rapporto delle prestazioni di OPT con quelle dei due algoritmi online, sempre nel caso in cui $T = 7$ e $\delta = 2.5$, e sullo spazio metrico del segmento unitario. Si osserva immediatamente che le prestazioni di entrambi gli algoritmi sono nettamente inferiori rispetto a quelle di $\text{REPLAN}_{\text{TRP}}$ e $\text{IGNORE}_{\text{TRP}}$; in particolare, per valori del lookahead Δ superiori a circa 0.6, $\text{REPLAN}_{\text{TSP}}$ sembra non risentire positivamente di ulteriori aumenti del valore di Δ, comportandosi peggio di $\text{IGNORE}_{\text{TSP}}$.

Sul quadrato a diametro unitario ancora una volta otteniamo una situazione del tutto analoga a quella relativa al segmento unitario (Figura 5.6).

Quando facciamo variare la durata T tra 3 e 7, sembra che le prestazioni diminuiscano leggermente (vedi Figura 5.7). È interessante notare che le prestazioni calano drasticamente con l'aumento della densità temporale δ delle richieste, soprattutto per quel che concerne $\text{REPLAN}_{\text{TSP}}$, come si può osservare nella Figura 5.8.

5.2.2 Confronto diretto di algoritmi online

Se si vogliono confrontare direttamente le prestazioni degli algoritmi online è possibile, a volte, condurre le simulazioni su istanze di input di grande taglia.

Nella precedente Sezione 5.2.1 abbiamo notato che, all'aumentare della densità temporale delle richieste, il solo algoritmo online che sembra mantenere un rapporto di prestazioni costante rispetto all'ottimo offline è $\text{REPLAN}_{\text{TRP}}$; le prestazioni degli altri algoritmi diminuiscono. Questa osservazione è in qualche modo confermata dalle Figure 5.9 e 5.10, che mostrano il rapporto di prestazioni di $\text{REPLAN}_{\text{TRP}}$ con $\text{IGNORE}_{\text{TRP}}$, e di $\text{IGNORE}_{\text{TRP}}$ con $\text{IGNORE}_{\text{TSP}}$ rispettivamente, al variare di δ (in ascissa). In entrambi i casi il lookahead è $\Delta = 0.5$, la durata $T = 20$.

Vediamo ora quello che accade al variare della durata dell'intervallo in cui sono rilasciate le richieste. Il rapporto tra le prestazioni di $\text{REPLAN}_{\text{TRP}}$ e $\text{IGNORE}_{\text{TRP}}$ non cambia significativamente al variare di T, sia quando la densità temporale delle richieste è media ($\delta = 2.5$, Figura 5.11), sia quando essa è elevata ($\delta = 4.5$, Figura 5.12). Il rapporto tra le prestazioni di $\text{IGNORE}_{\text{TRP}}$ e $\text{IGNORE}_{\text{TSP}}$ si mantiene costante quando $\delta = 2.5$ (Figura 5.13), e sembra diminuire all'aumentare di T quando $\delta = 4$, anche se gli esperimenti in questo caso si riferiscono a valori della durata T molto più bassi (Figura 5.14).

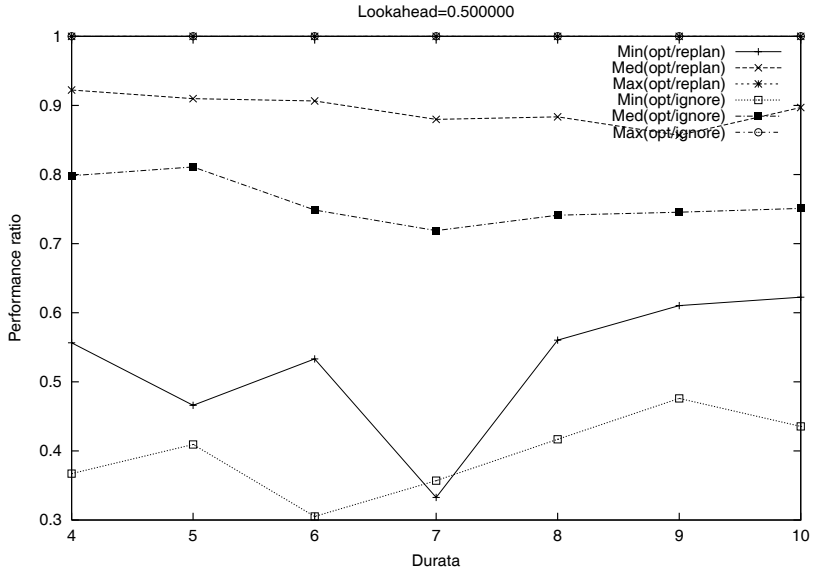

Figura 5.3: Che cosa succede quando varia la durata dell'intervallo in cui sono rilasciate le richieste.

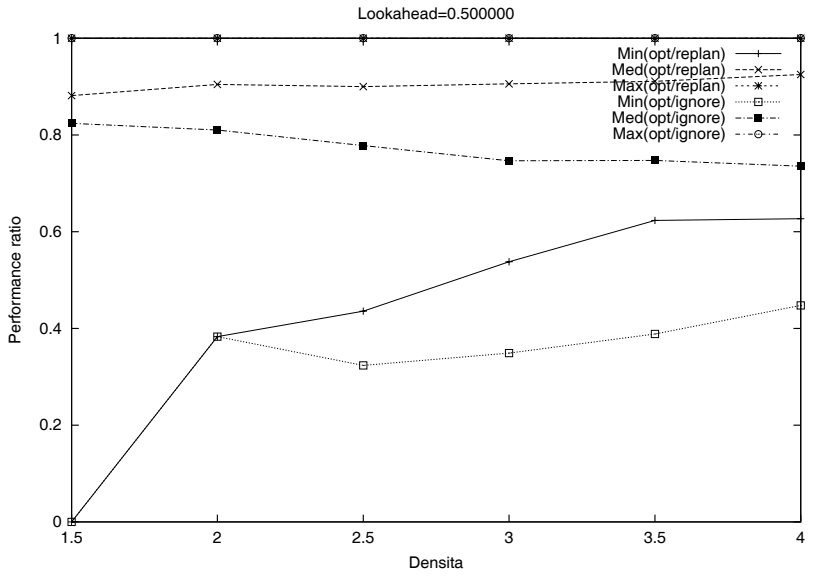

Figura 5.4: Che cosa succede quando varia la densità media temporale delle richieste.

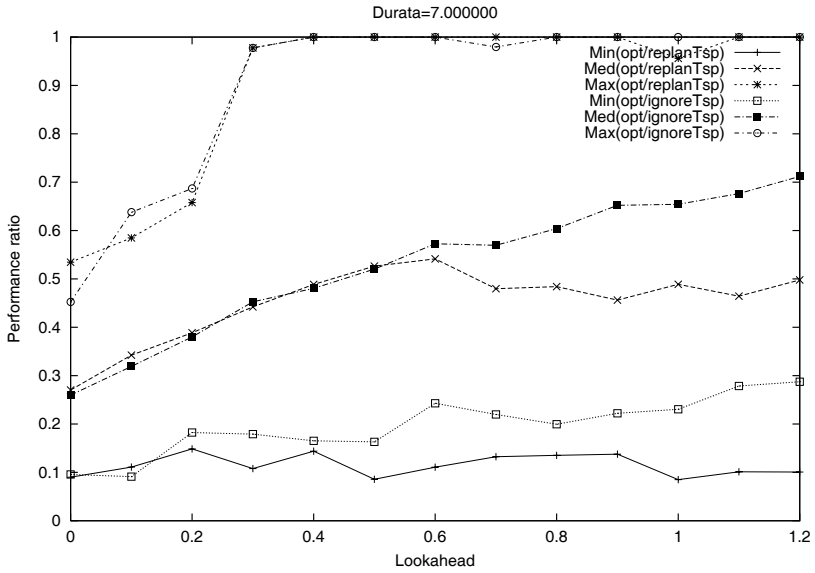

Figura 5.5: Confronto sperimentale delle prestazioni di OPT con REPLAN$_{\text{TSP}}$ (in figura, `replanTsp`) e IGNORE$_{\text{TSP}}$ (in figura, `ignoreTsp`), sul segmento unitario.

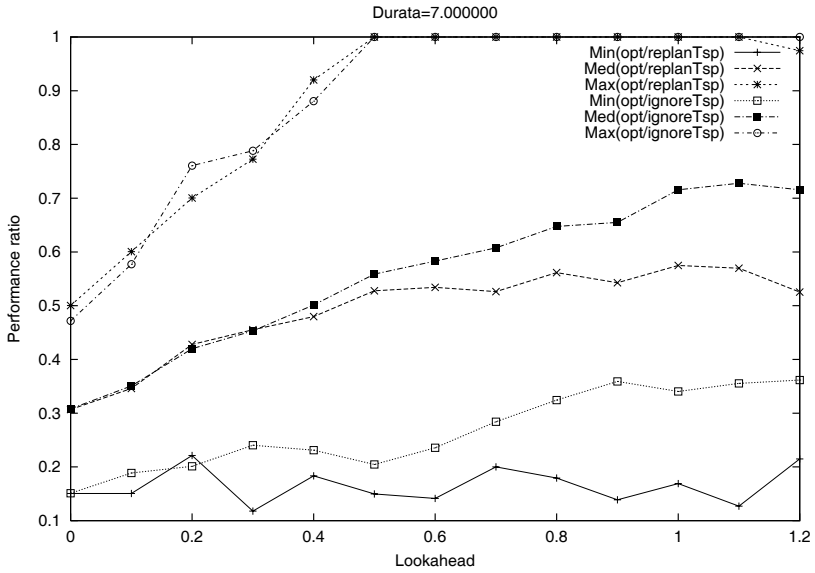

Figura 5.6: Confronto sperimentale delle prestazioni di OPT con REPLAN$_{\text{TSP}}$ (in figura, `replanTsp`) e IGNORE$_{\text{TSP}}$ (in figura, `ignoreTsp`), sul quadrato a diametro unitario.

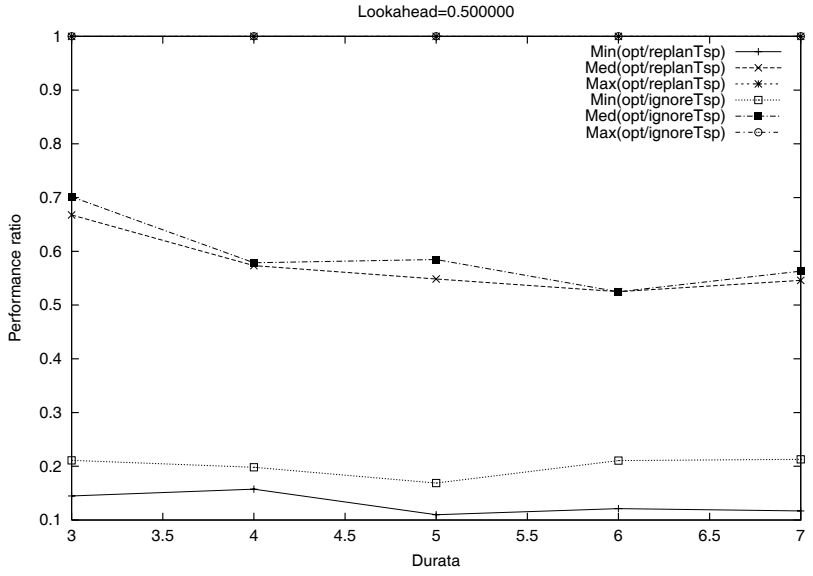

Figura 5.7: Che cosa succede quando varia la durata dell'intervallo in cui sono rilasciate le richieste.

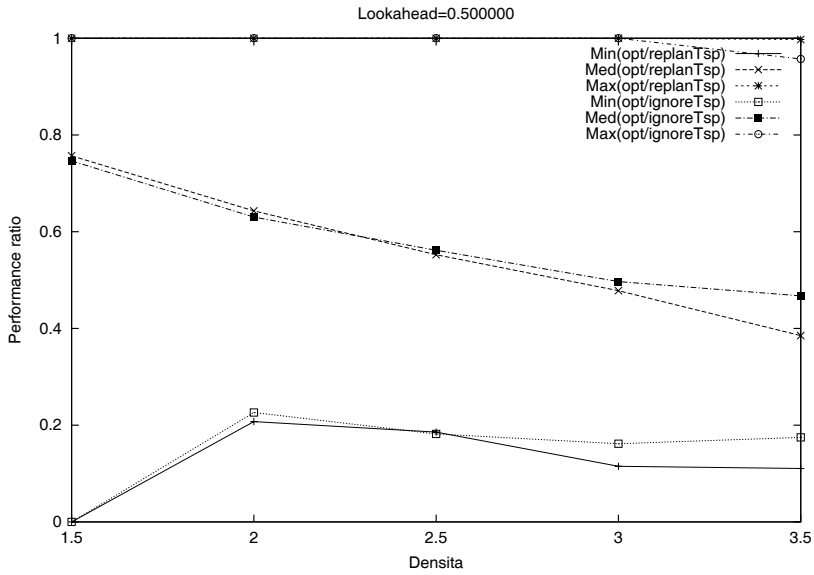

Figura 5.8: Che cosa succede quando varia la densità media temporale delle richieste.

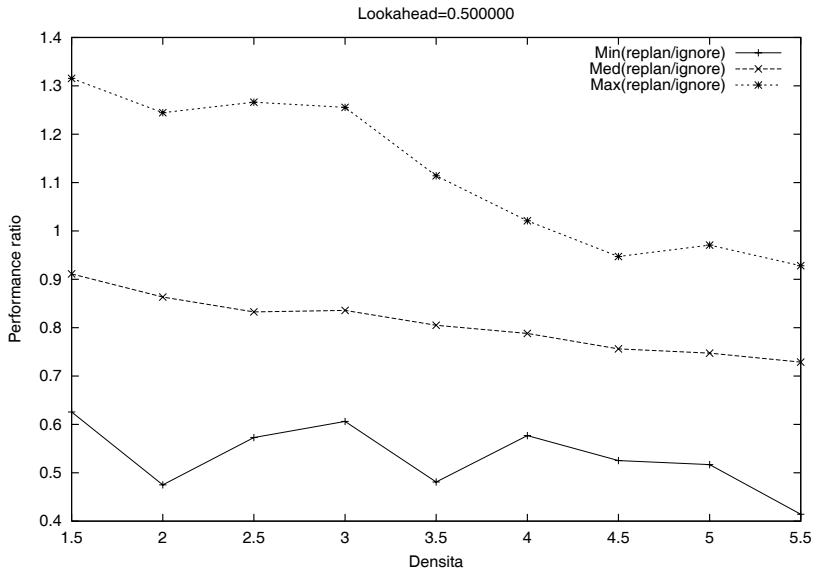

Figura 5.9: Rapporto tra le prestazioni di REPLAN$_{\text{TRP}}$ (`replan`) e IGNORE$_{\text{TRP}}$ (`ignore`) al variare di δ.

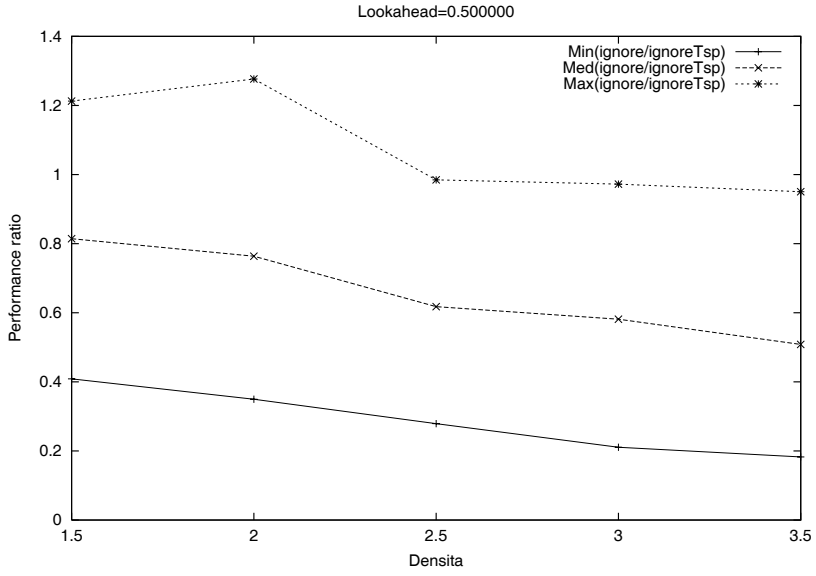

Figura 5.10: Rapporto tra le prestazioni di IGNORE$_{\text{TRP}}$ (`ignore`) e IGNORE$_{\text{TSP}}$ (`ignoreTsp`) al variare di δ.

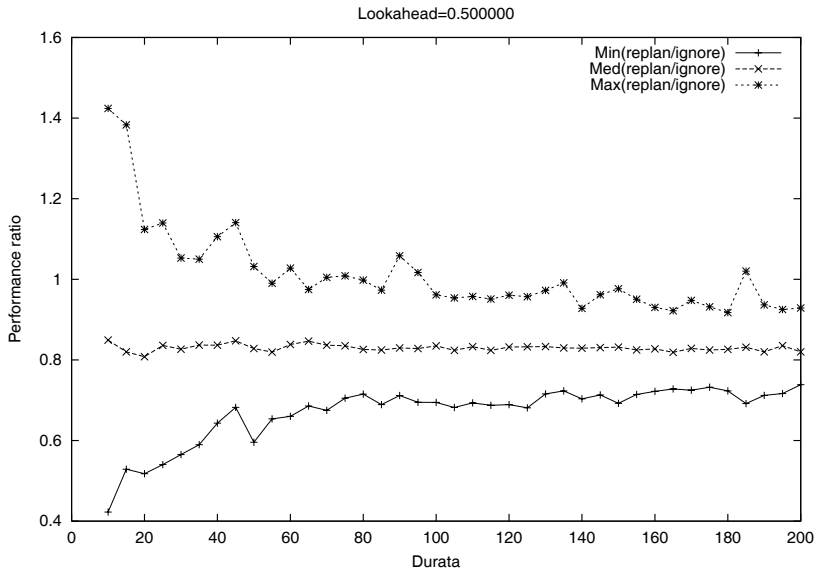

Figura 5.11: Rapporto tra le prestazioni di REPLAN$_{\text{TRP}}$ (`replan`) e IGNORE$_{\text{TRP}}$ (`ignore`) al variare di T, quando $\delta = 2.5$.

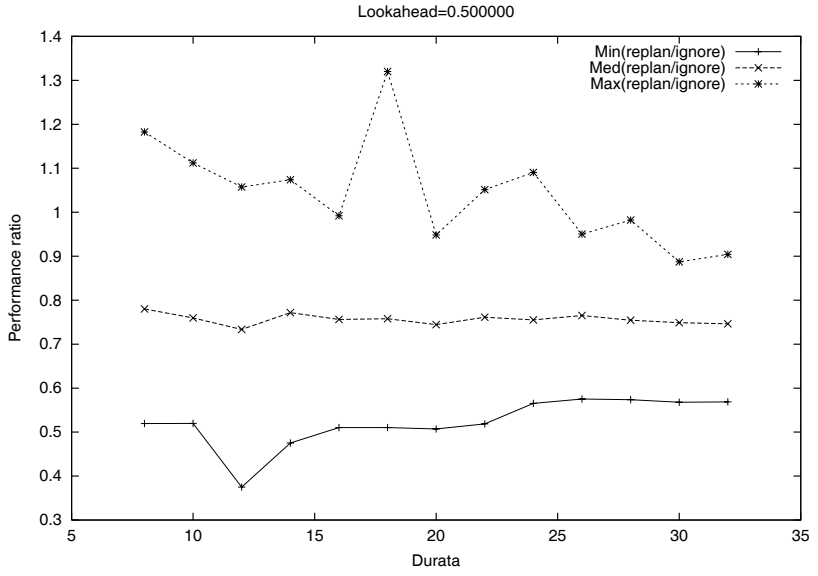

Figura 5.12: Rapporto tra le prestazioni di REPLAN$_{\text{TRP}}$ (`replan`) e IGNORE$_{\text{TRP}}$ (`ignore`) al variare di T, quando $\delta = 4.5$.

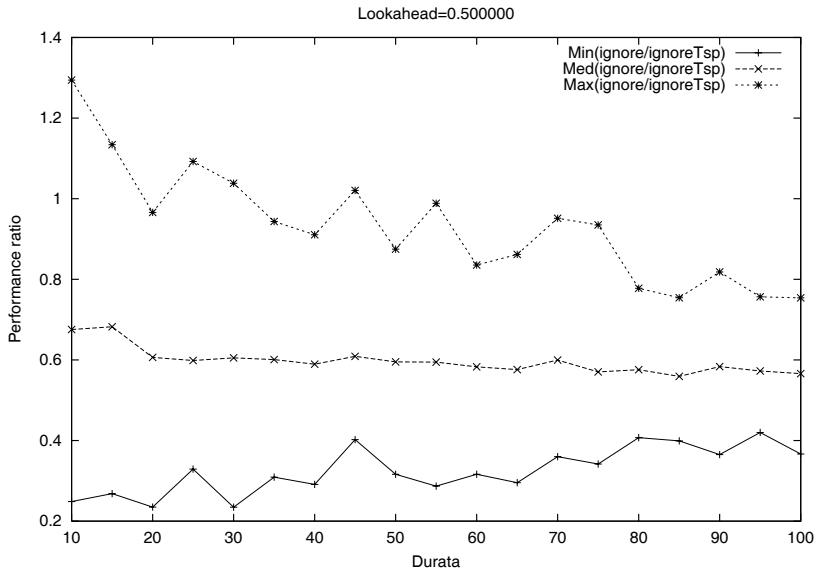

Figura 5.13: Rapporto tra le prestazioni di IGNORE$_{\text{TRP}}$ (`ignore`) e IGNORE$_{\text{TSP}}$ (`ignoreTsp`) al variare di T, quando $\delta = 2.5$.

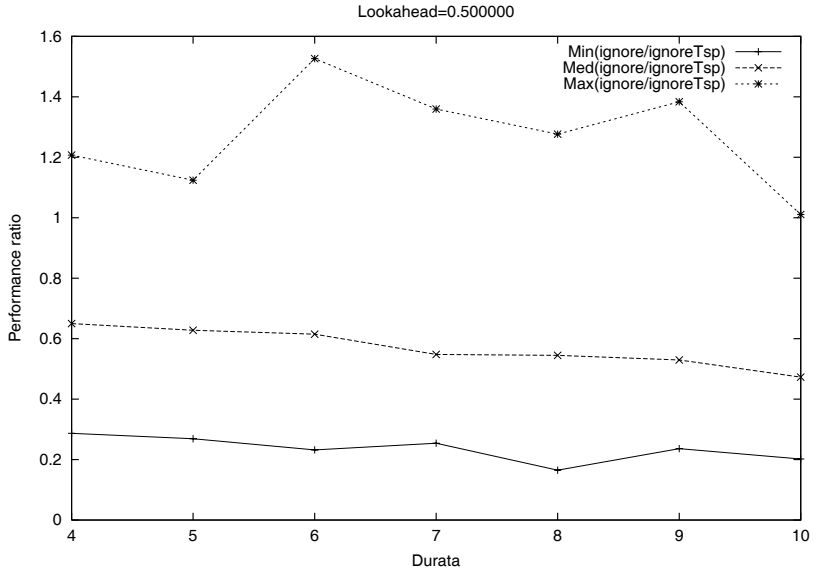

Figura 5.14: Rapporto tra le prestazioni di IGNORE$_{\text{TRP}}$ (`ignore`) e IGNORE$_{\text{TSP}}$ (`ignoreTsp`) al variare di T, quando $\delta = 4.0$.

5.3 Conclusioni

Dall'analisi sperimentale emerge chiaramente che il lookahead è una proprietà di grande rilevanza pratica per il problema del riparatore viaggiatore online. Abbiamo confrontato le prestazioni di quattro algoritmi online sia rispetto all'ottimo offline che tra di loro, ed è emersa una netta superiorità dell'algoritmo $\text{REPLAN}_{\text{TRP}}$. Infatti esso:

- ha conseguito i migliori rapporti di prestazioni rispetto all'ottimo offline, su istanze di input di piccola taglia;
- si è comportato meglio degli altri algoritmi online, anche su istanze di input di taglia maggiore;
- è stato l'unico algoritmo, tra quelli analizzati, le cui prestazioni non hanno risentito negativamente di un incremento della densità temporale delle richieste.

È interessante osservare che, nella pratica, $\text{REPLAN}_{\text{TRP}}$ si comporta considerevolmente meglio di $\text{IGNORE}_{\text{TSP}}$: ricordiamo che, se il carico di lavoro è Δ_0 ragionevole (vedi Sezione 3.3.2), 0-$\text{IGNORE}_{\text{TSP}}$ consegue un massimo tempo di servizio e un tempo medio di servizio non superiori a $2\Delta_0$ (Teorema 3.26).

Appendice A

Descrizione del programma di simulazione

A.1 La struttura generale

Il simulatore riceve in input un file di testo contenente la descrizione degli esperimenti da compiere, e fornisce in output il risultato degli esperimenti, sia in formato testuale che in un eventuale formato grafico (PostScript).

La Figura A.1 mostra lo schema concettuale del programma di simulazione.

Ogni *esperimento* è descritto da una stringa di input s. Per esperimento si intende l'estrazione di N istanze di input, la simulazione del comportamento di un insieme di algoritmi su ciascuna delle N istanze e il calcolo delle statistiche sui rapporti di prestazioni conseguiti, così come descritto nel Capitolo 5. È possibile introdurre $h \in \mathbb{N}^+$ parametri formali nella stringa che descrive l'esperimento: in questo modo si ottiene una funzione $\tilde{s} : \mathbb{R}^h \to \Sigma^*$ che, per ogni h-pla di valori di parametri $(p_1, \ldots, p_h)$ fornisce una stringa $\tilde{s}(p_1, \ldots, p_h)$ che descrive un esperimento.

Il modulo funzionale `SimulaConParametri` legge il file di input, che contiene essenzialmente la stringa parametrizzata $\tilde{s}$ e la descrizione dei valori da assegnare ai parametri; esegue di volta in volta una assegnazione $(p_1, \ldots, p_h)$ ai parametri ed invoca il modulo funzionale `SimulaSenzaParametri` sulla stringa $\tilde{s}(p_1, \ldots, p_h)$, eseguendo così un esperimento. Di volta in volta, `SimulaConParametri` riceve i risultati dell'esperimento e si occupa della costruzione dell'output, sia grafico (con l'ausilio del programma `GNUPlot`) che testuale.

Il modulo funzionale `SimulaSenzaParametri` possiede, oltre alla stringa s che descrive l'esperimento da compiere, un insieme di algoritmi (istanze

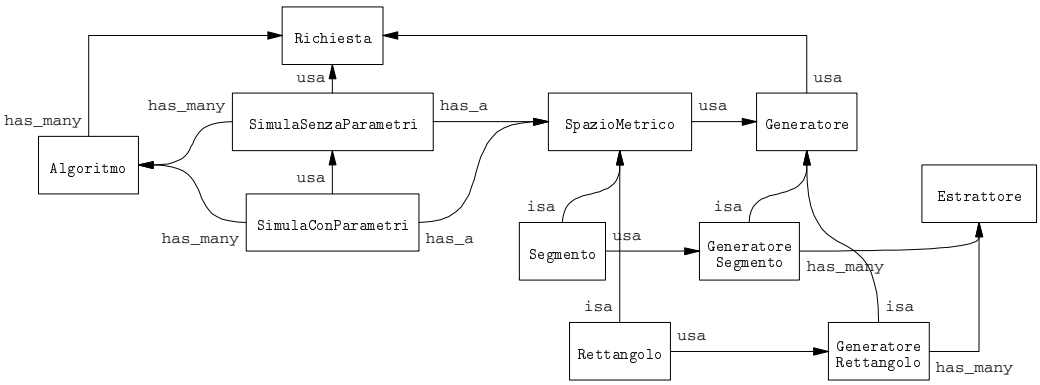

Figura A.1: Schema concettuale generale del simulatore.

di `Algoritmo`) e un'istanza di `SpazioMetrico`. Quest'ultima costituisce lo spazio metrico su cui eseguire gli esperimenti. Gli spazi metrici supportati sono il segmento $[a, b]$ (istanza di `Segmento`) e il rettangolo $[a, b] \times [c, d]$ (istanza di `Rettangolo`). Ad ogni tipo di spazio metrico è associato un tipo di generatore (`GeneratoreSegmento` e `GeneratoreRettangolo`, che derivano `Generatore`) il quale ha il compito di generare delle richieste per quel particolare spazio metrico. Quando un generatore necessita di estrarre una determinazione di una variabile aleatoria, utilizza un'istanza di `Estrattore` che incapsula la distribuzione di probabilità desiderata.

`SimulaSenzaParametri` lavora nel seguente modo:

1. crea un'istanza di `Generatore` (utilizzando lo `SpazioMetrico` che possiede); le passa in input la parte della stringa s che contiene la descrizione del modo in cui si vogliono estrarre le richieste (essenzialmente, i valori o le distribuzioni per la durata T, la densità temporale delle richieste δ e la distribuzione spaziale delle richieste, come descritto nella Sezione 5.1.3).

2. legge nella stringa s il valore di N; per N volte:

   (a) chiede al generatore di generare un'istanza di input σ, costituita da un insieme di richieste (una richiesta è un oggetto di tipo `Richiesta`).

   (b) per ogni algoritmo A descritto in s, fornisce ad A l'istanza σ, esegue l'algoritmo e memorizza il costo da esso conseguito;

   (c) per ogni coppia di algoritmi A e B di cui si vuole confrontare le prestazioni (secondo quanto indicato in s), calcola il rapporto dei costi;

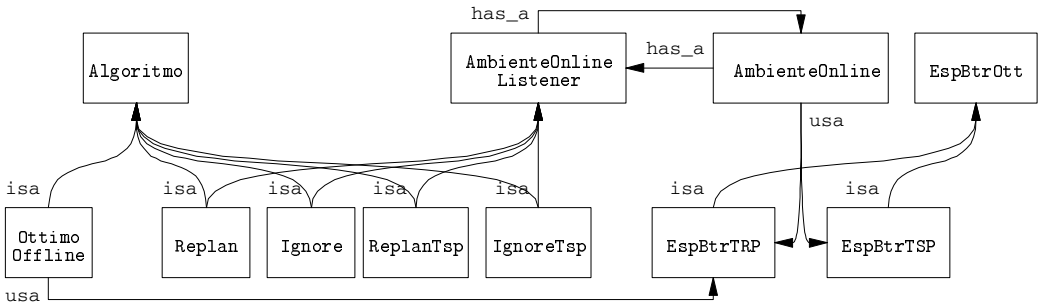

Figura A.2: Schema concettuale della parte algoritmica del simulatore.

3. infine calcola le statistiche sui rapporti dei costi conseguiti dagli algoritmi.

A.2 L'implementazione degli algoritmi online e dell'ottimizzatore

Per implementare un particolare algoritmo per il L_N-OlTRP, sia esso online oppure offline, occorre definire un modulo che deriva da `Algoritmo` e offre un metodo che, ricevendo in input l'istanza σ, ovvero un insieme di richieste (`Richiesta`), restituisce il costo della soluzione per σ calcolata dall'algoritmo in considerazione. Nella Figura A.2 sono mostrati i moduli attualmente disponibili.

L'algoritmo `OttimoOffline` è l'ottimizzatore offline. Come abbiamo accennato nel Capitolo 5, non sono noti algoritmi di costo polinomiale per il L_N-OlTRP: perciò `OttimoOffline` utilizza la tecnica algoritmica del backtracking per esplorare lo spazio di ricerca e determinare una soluzione ottima. L'esploratore dello spazio di ricerca è implementato dal modulo `EspBtrTRP`, che estende il modulo astratto `EspBtrOtt`, tratto da [7]: quest'ultimo fornisce la struttura di un generico esploratore dello spazio di ricerca che utilizza la tecnica del backtracking per risolvere un problema di ottimizzazione.

È stata creata un'infrastruttura avente lo scopo di semplificare lo sviluppo di nuovi moduli che implementano algoritmi online dotati di lookahead. Il funzionamento dell'infrastruttura è il seguente:

- un oggetto di tipo `AmbienteOnline` (che d'ora in avanti chiameremo *ambiente*) si occupa di condurre la simulazione. È a conoscenza dell'insieme delle richieste che costituisce l'istanza di input σ; possiede come

proprietà l'istante di tempo simulato (inizialmente 0) e la posizione del server (inizialmente l'origine);

- l'oggetto che rappresenta l'algoritmo online, che d'ora in avanti chiameremo semplicemente *algoritmo*, deve essere di tipo `AmbienteOnlineListener` per poter dialogare con `AmbienteOnline`: `AmbienteOnlineListener` è essenzialmente un'interfaccia che l'ambiente utilizza per richiedere l'intervento dell'algoritmo quando necessario;

- negli istanti di tempo simulato in cui avviene un evento significativo, l'ambiente invoca un opportuno metodo dell'algoritmo per comunicare l'evento e permettere all'algoritmo di reagire in modo opportuno. Esempi di eventi significativi sono il rilascio di una nuova richiesta (nuova per l'algoritmo, in quanto l'ambiente conosce tutte le richieste fin dal principio), l'arrivo del server in un punto dello spazio metrico, il termine del servizio di una richiesta o di una sequenza di richieste, l'entrata di una richiesta nella finestra di lookahead ecc.;

- l'algoritmo può ignorare un evento oppure utilizzare i metodi offerti dall'ambiente per reagire in modo opportuno, al fine di servire tutte le richieste presentate. Per esempio sono presenti metodi che chiedono all'ambiente di spostare il server verso un punto dello spazio metrico, di fermare il server, di servire una richiesta, di servire nell'ordine le richieste di una sequenza, di impostare un'allarme, di risolvere in modo ottimo un'istanza del problema del commesso viaggiatore online (fornitagli dall'algoritmo), e così via. Per offrire alcuni di questi metodi, l'ambiente fa uso dei moduli `EspBtrTRP` e `EspBtrTSP`.

Bibliografia

[1] F. Afrati, S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou, N. Papakonstantinou. *The complexity of the traveling repairman problem.* Theoretical Informatics and Applications **20**, pagine 79-87, 1986.

[2] S. Albers. *The influence of lookahead in competitive paging algorithms.* First Annual European Symposium on Algorithms, pagine 1-12, 1993.

[3] S. Albers. *A competitive analysis of the list update problem with lookahead.* Proceedings 6th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, pagine 201-210, 1994.

[4] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, M. Talamo. *Algorithms for the on-line travelling salesman.* Algorithmica **29** pagine 560-581, 2001.

[5] M. Blom, S. O. Krumke, W.E. de Paepe, L. Stougie. *The online*-TSP *against fair adversaries.* INFORMS Journal on Computing **13**, pagine 138-148, 2001.

[6] V. Bonifaci. *Algoritmi online per sistemi metrici di servizi.* Tesi di laurea in Ingegneria Informatica. Università La Sapienza, Roma, 2003.

[7] M. Cadoli, M. Lenzerini, P. Naggar, A. Shaerf. *Fondamenti della progettazione dei programmi: Princìpi, tecniche e loro applicazioni in C++.* Città*Studi*Edizioni, UTET Libreria, Torino, 1997.

[8] E. Feuerstein, L. Stougie. *On-line single-server dial-a-ride problems.* Theoretical Computer Science **268**, pagine 91-105, 2001.

[9] A. Fiat, G.J. Woeginger (eds.). *Online algorithms: The state of the art.* Lecture Notes in Computer Science, vol. 1442, 1998.

[10] R.L. Graham. *Bounds for certain multiprocessor anomalies.* Bell System Technical Journal **45**, 1966.

[11] D. Hauptmeier, S.O. Krumke, J. Rambau. *The online dial-a-ride problem under reasonable load.* Proceedings of the 4th Italian Conference on Algorithms and Complexity. Lecture Notes in Computer Science, vol. 1767, 2000.

[12] D.S. Johnson. *Near optimal bin-packing algorithms.* Doctoral thesis, MIT, 1973.

[13] E. Koutsoupias, C.H. Papadimitriou. *Beyond competitive analysis.* Proceedings 35th Annual Symposium on Foundations of Computer Science, pagine 394-400, 1994.

[14] S.O. Krumke. *Online optimization: Competitive analysis and beyond.* Habilitationsschrift. Technische Universität Berlin, 2001.

[15] S.O. Krumke, W.E. de Paepe, D. Poensgen, L. Stougie. *News from the online travelling repairman.* Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science. Lecture Notes in Computer Science, vol. 2136, pagine 487-499, 2001.

[16] L. Laura. *Risoluzione on-line di problemi dial-a-ride.* Tesi di laurea in Ingegneria Informatica. Università La Sapienza, Roma, 1999.

[17] L. Laura, M. Lipmann, A. Marchetti-Spaccamela, S.O. Krumke, W.E. de Paepe, D. Poensgen, L. Stougie. *Non-abusiveness helps: An $O(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem.* Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization. Lecture Notes in Computer Science, pagine 200-214, 2002.

[18] M. Lipmann. *On-Line Routing.* PhD Thesis. Da pubblicare.

[19] M. Lipmann, X. Lu, W.E. de Paepe, R. Sitters, L. Stougie. *Online dial-a-ride problems under a restricted information model.* Proceedings of the 10th Annual European Symposium on Algorithms. Lecture Notes in Computer Science, 2002.

[20] W.E. de Paepe. *Computer-aided complexity classification of dial-a-ride problems.* Master's thesis. University of Amsterdam, Faculty of Economics and Econometrics, 1998.

[21] W.E. de Paepe. *Complexity results and competitive analysis for vehicle routing problems.* Ph.D. thesis. Eindhoven University of Technology, 2002.

[22] E. Pini. *Comunicazione personale.*

[23] D.D. Sleator, R.E. Tarjan. *Amortized efficiency of list update and paging rules.* Communications of the ACM **28**, no. 2, pagine 202-208, 1985.