Università degli studi di Roma Tre
Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea in Matematica

Sintesi relativa alla Tesi di Laurea in Matematica di

Giulio Simeone

Barriere assorbenti nelle catene di Markov e una loro applicazione al web

Relatore
Prof. Marco Liverani

Il Candidato Il Relatore

Anno Accademico 2001/2002
Luglio 2002

Classificazione AMS: 68C05, 68C25, 68E10
Parole chiave: grafi, ottimizzazione, Catene di Markov

In questa Tesi intendiamo illustrare un problema che si può porre a un Webmaster e descrivere un algoritmo che risolve in maniera soddisfacente tale problema: abbiamo implementato tale algoritmo tramite un programma scritto in C.

Sulla rete Internet esistono forme di pubblicità che funzionano in questo modo: un Webmaster ed un'azienda possono stipulare un contratto nel modo seguente. Il Webmaster accetta di ospitare in una pagina web del sito un banner pubblicitario, ovvero un link che conduce ad una pagina dove vengono pubblicizzati i prodotti messi in commercio dall'azienda, e quest'azienda lo ricompensa con del denaro.

La situazione è cioè quella raffigurata (figura 1):

Il Webmaster può stipulare più di un contratto di questo tipo: il nostro algoritmo prende appunto in considerazione la situazione in cui il Webmaster abbia stipulato contratti con una o più aziende, e debba inserire uno o più banner pubblicitari all'interno delle pagine del sito.

Noi possiamo modellizzare questo sito web come un grafo orientato (figura 2)

I nodi rappresentano le pagine HTML, e i link i collegamenti tra di esse.

Lo scopo del nostro algoritmo sarà di trovare la disposizione migliore per tali banner pubblicitari all'interno di queste pagine HTML, ovvero la disposizione che massimizza il profitto. Più precisamente, ogni banner pubblicitario va collocato in una ed una sola pagina HTML: una pagina HTML può però

ospitare più banner.

Ad esempio, rappresentiamo i banner pubblicitari in questo modo:

Una possibile collocazione per tali banner all'interno del sito web è

Per ogni banner X l'algoritmo prende in input due coefficienti:

1) a_X esprime il coefficiente di attrattività associato al banner, cioè la misura in cui esso può attirare l'attenzione di un visitatore. Esso può variare da 0,5 a 2.

2) p_x esprime il profitto associato al banner, cioè il guadagno conseguito dal webmaster per ogni visitatore che clicca su di esso.

Vediamo in che modo l'algoritmo trova una soluzione soddisfacente per il problema che abbiamo illustrato. Ovviamente, quando il numero di banner è piccolo, sarà tale anche il numero di disposizioni possibili: in questo caso chiaramente l'algoritmo le valuta tutte e trova quella migliore.

Tuttavia, quando il numero di banner da collocare nel sito è grande, l'algoritmo non può valutare tutte le disposizioni possibili perchè ciò sarebbe troppo dispendioso in termini di tempo; allora ne valuta solo alcune, e in questo modo non viene trovata sempre la soluzione migliore, ma ne viene trovata una soddisfacente.

Vediamo adesso come viene valutata ogni disposizione di banner.

Prima di tutto dobbiamo parlare del concetto di flusso all'interno di un sito.

Lo stesso server che pubblica il sito produce dei file di log, dei file di testo in cui sono registrati tutti gli accessi degli utenti di Internet alle pagine del sito, e tutti gli spostamenti degli stessi utenti da una pagina all'altra. Elaborando i file di log, l'algoritmo rileva il percorso compiuto da ciascun utente che visita il sito in un certo tempo t prefissato, dal momento in cui entra nel sito fino al momento in cui esce da esso.

Ad esempio, il percorso compiuto da un utente può essere il seguente:

Nel caso specifico l'utente parte da una pagina esterna al sito: però egli potrebbe anche entrare nel sito tramite accesso diretto, cioè digitando l'indi-

rizzo nell'apposita finestra del browser. Tuttavia, ai fini del nostro algoritmo ci interessa ciò che accade all'interno del sito, e quindi il fatto che un utente entri nel sito da una pagina esterna al sito o per accesso diretto non ha importanza. Quindi, rappresenteremo le due possibilità suddette tramite un unico nodo, che chiameremo nodo sorgente. Similmente, ogni utente al termine del suo percorso si può disconnettere da Internet oppure raggiungere una pagina esterna al sito: ancora una volta rappresenteremo questa possibilità tramite un unico nodo, che chiameremo pozzo di disconnessione.

A questo punto il nostro grafo risulta ampliato con altri due nodi:

Torniamo alle rilevazioni compiute sul percorso degli utenti all'interno del sito, grazie ad esse è possibile determinare il numero di utenti che accedono

ad ogni pagina del sito e, per ogni coppia (a, b) di pagine, il numero di utenti che va dalla pagina a alla pagina b.

Tali informazioni possono essere riassunte in un grafo pesato ed orientato del tipo seguente, che esprime il flusso all'interno del sito:

I pesi associati agli archi vengono poi riassegnati dall'algoritmo in chiave probabilistica: ad esempio, nel grafo schematizzato sopra 8 visitatori raggiungono il nodo 2. Di questi 8, 3 transitano verso il nodo 3 e 5 verso il pozzo di disconnessione. Questi dati possono essere interpretati in chiave probabilistica: si può affermare infatti che un visitatore che raggiunge il nodo 2 si sposta verso il nodo 3 con probabilità 3/8, e verso il pozzo di disconnessione con probabilità 5/8.

Qua sotto rappresentiamo il grafo con i pesi che vengono assegnati nel modo suddetto;

Tale grafo può essere interpretato come una catena di Markov; spieghiamo

in due parole cosa si intende con questo termine.

Una catena di Markov è un sistema che può passare attraverso un dato insieme di stati e che si evolve nel tempo, più precisamente in una successione finita o numerabile di tempi t = 1, 2, ... , k . Per ogni tempo t il sistema passa per uno stato s, e le probabilità con cui il sistema passa da uno stato s1 al tempo t ad uno stato s2 al tempo t + 1 sono note e indipendenti dal tempo t.

Tenuta presente tale definizione, possiamo ben capire perché il grafo schematizzato sopra può essere interpretato come una catena di Markov: l'idea è che un dato utente circola nelle pagine del sito, e si sposta da una pagina all'altra con una probabilità prefissata.

Le probabilità con cui l'utente si sposta da un nodo all'altro si possono riassumere in una matrice, detta matrice di transizione.

Nel caso specifico schematizzato sopra, la matrice di transizione sarà la seguente:

	SORGENTE	1	2	3	4	*POZZO*
SORGENTE	0	6/19	5/19	5/19	3/19	0
1	0	0	1/2	0	0	1/2
2	0	0	0	3/8	0	5/8
3	0	0	0	0	5/8	3/8
4	0	0	0	3/8	0	5/8
POZZO	0	0	0	0	0	0

Ricordiamo che il nostro scopo era quello di valutare ogni possibile disposizione dei banner all'interno del sito.

Ogni banner condurrà ad una pagina esterna al sito web: pertanto, per ogni disposizione dei banners, il sito verrà espanso in un certo modo, ad esempio consideriamo la disposizione di banner seguente:

Dato ogni banner collocato in una pagina, l'algoritmo calcola prima di tutto la probabilità con cui un utente che visita la pagina selezioni il banner. Supponiamo che il banner B sia collocato nella pagina 4 e ad esso risulti associato un coefficiente di attrattività pari ad a_B; supponiamo inoltre che dalla pagina $U(4)$ fuoriescano $U(4)$ archi, e indichiamo con T_{4P} la proporzione di utenti che dalla pagina 4 si disconnette dal sito.

Allora la formula con cui viene stimata la suddetta probabilità è

$T_{4B} = a_B/U(4) * (1 - T_{4P})$

Una volta calcolate queste probabilità, le altre probabilità vengono aggiornate di conseguenza: infatti, data una pagina interna al sito, la somma delle probabilità che da quella pagina un utente vada verso altre pagine oppure si disconnetta deve essere uguale ad 1. Quindi, data la matrice di transizione ampliata con le nuove probabilità, ogni riga viene 'normalizzata' in modo che la somma di tutti i suoi elementi risulti uguale ad 1.

Riprendendo il nostro esempio, prendendo in considerazione la disposizione schematizzata, e supponendo che $a_A = 1$, $a_B = 1/2$, $a_C = 1/3$, si ha

$a_A = 1 \Longrightarrow T_{1A} = /U(1) * (1 - T_{1P}) = 1/3 * 1/2 = 1/6$

$a_B = 1/2 \Longrightarrow T_{4B} = a_B/U(4) * (1 - T_{4P}) = 1/2 * 3/8 = 3/16$

$a_C = 2/3 \Longrightarrow T_{4C} = a_C/U(4) * (1 - T_{4P}) = 2/3 * 3/8 = 1/4$

(infatti dal disegno vediamo che $U(1) = 3, U(4) = 2, T_{1P} = 1/2, T_{4P} = 5/8$).

La matrice di transizione T viene quindi ampliata e aggiornata in questo modo:

	SOR	1	2	3	4	A	B	C	$POZZO$
SOR	0	6/19	5/19	5/19	3/19	0	0	0	0
1	0	0	3/7	0	0	1/7	0	0	3/7
2	0	0	0	3/8	0	0	0	0	5/8
3	0	0	0	0	5/8	0	0	0	3/8
4	0	0	0	27/64	0	0	7/64	9/32	45/32
$POZZO$	0	0	0	0	0	0	0	0	0

Ora, interpretando il grafo come una catena di Markov, possiamo distinguere tra stati ricorrenti e transienti.

Si dice ricorrente uno stato tale che, se il sistema passa per quello stato, si ha la certezza che prima o poi vi torna. Se non si ha questa certezza, uno stato si dice transiente.

Osservando il grafo, è facile notare che gli stati SORGENTE,1, 2, 3, e 4, corrispondenti alle pagine del sito originario, sono transienti, gli stati A, B, C e POZZO, corrispondenti ai link aggiunti, sono ricorrenti.

Chiamiamo Q la matrice che esprime le probabilità di transizione tra gli stati ricorrenti; essa corrisponderà ad una porzione dell' intera matrice di transizione T.

Sopra abbiamo evidenziato in rosa gli elementi appartenenti alla matrice Q.

L'algoritmo utilizza il seguente

Teorema. Data una catena di Markov, ordiniamo i suoi stati in modo che quelli transienti precedano quelli ricorrenti e suddividiamo di conseguenza la sua matrice di transizione T in 4 blocchi:

$$\begin{matrix} D & 0 \\ B & Q \end{matrix}$$

Allora:

1) La matrice che esprime il numero di visite atteso su uno stato i a partire

da uno stato j è

$$\begin{matrix} E & 0 \\ F & U \end{matrix}$$

dove $U = (I - Q)^{-1}$

2) La matrice che esprime la probabilità che, a partire da uno stato i transiente, il sistema finisca su uno stato j ricorrente è

$L = UB.$

L'algoritmo effettua il calcolo della matrice inversa $(I - Q)^{-1}$ tramite l'algoritmo della fattorizzazione LU.

Per ogni disposizione dei link aggiunti, il teorema precedente permette in particolare di calcolare le probabilità che un visitatore che entra nel sito (cioè che parte del nodo sorgente) finisca su uno dei link aggiunti: infatti i link aggiunti corrispondono agli stati ricorrenti della catena di Markov.

Se chiamiamo V il numero di visitatori che entra nel sito, n il numero di banner, $p(B_i)$ la probabilità con cui ogni visitatore che entra nel sito va a finire sul banner B_i , e $P(B_i)$ il profitto associato al banner, allora il profitto atteso per la disposizione è

$$V * \sum_{i=1}^{n} p(B_i)P(B_i)$$

La disposizione per cui il profitto risulterà massimo sarà quella ottimale.

Infine, diciamo due parole sull'algoritmo del Vecchio Scapolo, citato nell'articolo: Hu, Khang, Tsao, "Old Bachelor Acceptance: A New Class of Non Monotone Threshold Acceptance Methods" che permette di sfoltire il numero di disposizioni nel caso in cui il sito sia troppo grande.

L'algoritmo del Vecchio Scapolo funziona nel modo seguente: sia dato un generico problema di ottimizzazione globale e sia dato l'insieme S delle soluzioni

a questo problema e supponiamo che a ciascuna di queste soluzioni sia associato un numero reale $f(s_i)$ .Per prima cosa l'algoritmo genera una soluzione di partenza t_0 e fissa una soglia *soglia*: le altre soluzioni $t_1, ..., t_n$ vengono generate una dopo l'altra e ogni soluzione viene generata a partire dalla precedente tramite una procedura prefissata.

Nel nostro algoritmo, ad esempio, ogni disposizione t_{i+1}di link nelle pagine del sito viene generata a partire da quella precedente t_i: essa viene scelta a caso in modo tale che le due soluzioni differiscano solo per la collocazione di una pagina. Ad esempio, dalla disposizione

possono essere generate, tra le altre, le disposizioni seguenti

Tutte e tre queste disposizioni sono state generate spostando uno e uno solo dei banner.

Il profitto $f(t_{i+1})$di ciascuna soluzione viene confrontata con il profitto $f(t_i)$ della soluzione precedente, e se è verificata la condizione $f(t_{i+1}) > f(t_i) +$ *soglia* allora la soluzione t_{i+1}viene posta uguale a t_i: altrimenti essa verrà posta uguale a t_i, ovvero non verrà generata alcuna nuova soluzione e la soluzione t_{i+2} verrà ancora generata a partire dalla soluzione t_i .

Inoltre, nel primo caso il valore *soglia* verrà incrementato; nel secondo caso verrà decrementato.

Il concetto è il seguente. La procedura che genera le soluzioni può essere concepita in modo che da una soluzione possano essere generate soltanto soluzioni in qualche modo simili o 'vicine' ad essa, ed è il caso dell' esempio che abbiamo appena illustrato.

Di conseguenza, possiamo introdurre il concetto di massimo locale una soluzione costituisce un massimo locale quando tutte le soluzioni che possono essere generate a partire da essa sono meno buone.

Ora, se la soglia fosse costantemente uguale a 0 succederebbe che, nell' ipotesi in cui la soluzione t_i fosse un massimo locale, la nuova soluzione t_{i+1} verrebbe rifiutata in qualsiasi caso e l'algoritmo non potrebbe più generare nuove soluzioni. In questo modo, quando tale massimo locale coincide con il massimo assoluto di f, l'algoritmo funziona perfettamente: quando però ciò non avviene, si ha che l'algoritmo si blocca su una soluzione mentre invece potrebbe cercare soluzioni migliori.

Tale ragionamento si estende anche al caso in cui t_{i+1} non costituisce un massimo locale, ma una soluzione abbastanza buona. Essa avrebbe scarse probabilità di venire accettata e quindi l'algoritmo si bloccherebbe per molte iterazioni sulla stessa soluzione.

Invece, l'introduzione della variabile soglia che viene incrementata ogni volta che la soluzione viene rifiutata e decrementata ogni volta che la soluzione viene accettata, permette all'algoritmo di funzionare meglio.

Infatti, poniamo che t_i sia una buona soluzione; di conseguenza le nuove soluzioni che verranno generate tenderanno ad essere rifiutate, però ad ogni rifiuto l'algoritmo incrementa il valore soglia, rendendo più probabile l'accettazione di nuove soluzioni. In questo modo, si evita che l'algoritmo si

blocchi su una soluzione buona rispetto a quelle che si trovano 'intorno' ad essa che potrebbe non essere la migliore in assoluto.

Se viceversa t_i è una soluzione non tanto buona, è più probabile che la soluzione che viene generata a partire da essa venga accettata: in caso di accettazione della soluzione la soglia viene decrementata, in modo da rendere ancora più probabile l'accettazione di ulteriori soluzioni e di velocizzare l'ascesa verso soluzioni migliori.