**HITS**

L’algoritmo HITS è tipicamente utilizzato su grafi diretti e applica in maniera ciclica un Authority Update Rule (AUR) seguita da un Hub Update Rule (HUR).  
Effettuare tale algoritmo su un grafo non diretto porta ad ottenere lo stesso risultato sia per gli Hub che per le Authorities, come anche riscontrato dai primi esperimenti effettuati. L’AUR e l’HUR risultano quindi completamente equivalenti potendo così ricavare una generica Update Rule (UR) *X=UR(X)* tale che due applicazioni consecutive di UR producono lo stesso risultato dell’applicazione di un AUR seguita da un HUR.

I successivi algoritmi sono stati quindi definiti come l’applicazione ciclica di k UR che equivalgono all’applicazione di k/2 cicli di AUR e HUR. Attraverso vari esperimenti, si è potuto confermare come tale supposizione sia corretta, ottenendo in output gli stessi risultati.

L’algoritmo è stato implementato in due versioni, nella prima sono stati utilizzati i dizionari per il calcolo dello score (*dict version*), mentre la seconda strutta il prodotto tra matrice e vettore (*matrix version*).

In entrambe le versioni si è potuta notare una rapida convergenza dell’algoritmo, come mostrato nelle seguenti immagini in cui le ordinate rappresentano la sommatoria su ogni nodo dei valori assoluti delle differenze tra il rank del ciclo attuale e il rank del ciclo precedente.

|  |  |
| --- | --- |
| Dict version: | Matrix version: |
|  |  |

Possiamo subito vedere come la somma delle differenze si assesta velocemente nell’intorno di 0 già nella prima decina di cicli. La completa convergenza viene invece raggiunta dopo circa 150/200 iterazioni in entrambe le versioni, con un assestamento a 0 nel caso della *dict version* ed un assestamento a 3,5\*10-15 nella *matrix version*.

Al fine di ottenere una completa convergenza e dati anche i tempi di esecuzione molto brevi, si è scelto di impostare di k pari a 200. In particolare, la *dict version* risulta impiegare circa 63 secondi per effettuare 200 cicli, mentre la *matrix version* è risultata molto più veloce, completando i 200 cicli in circa 2 secondi.

Per quanto riguarda le versioni parallele di tali algoritmi, la necessità di dover conoscere in ogni thread e ad ogni ciclo gli score aggiornati dagli altri threads ha imposto l’utilizzo di una struttura dati condivisa tra i jobs. Tale scelta ha permesso quindi di ottenere gli stessi risultati delle versioni Naive, sia in termini di score che di convergenza, a discapito della velocità a causa dei tempi di gestione della parallelizzazione e del lock sulla struttura condivisa.  
In particolare, per la *dict\_version*, con un numero di job pari a 4 la gestione della parallelizzazione diventa troppo onerosa, impiegando circa 65 secondi per completare i 200 cicli (circa 2 secondi più lenta della Naive), mentre con 2 soli job si è potuto notare un discreto miglioramento nei tempi, ottenendo un tempo di esecuzione pari a 55 secondi.

Al contrario, per la *matrix version*, non si è riusciti ad ottenere nessun miglioramento dei tempi con la parallelizzazione a causa sia del brevissimo lasso di tempo che necessita la versione Naive per essere eseguita sia della complessità della gestione della parallelizzazione.  
Infatti, con un numero di job pari a 4 si ottiene un tempo di esecuzione di circa 11 secondi, mentre con 2 job il tempo scende a circa 7 secondi, comunque molto superiori ai 2 secondi che impiega mediamente la versione Naive.