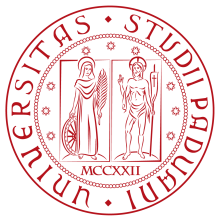
UNIVERSITÀ DEGLI STUDI DI PADOVA



Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria dell'Automazione

Corso di Progettazione di sistemi di Controllo a.a. 2010/11
Prof. Luca Schenato

Analisi di algoritmi per la sincronizzazione temporale

Camilla Corfini, Matricola: 1014127
Emilano Milesi, Matricola: 603265

camilla.corfini@studenti.unipd.it
milesi.emiliano@alice.it

Padova, 25 Febbraio 2011

DEPARTMENT OF INFORMATION ENGINEERING
UNIVERSITY OF PADOVA

Indice

1 Introduzione

Le recenti evoluzioni tecnologiche nei campi dei sistemi di controllo, dell'elettronica e l'avvento della tecnologia della miniaturizzazione dei sistemi elettromeccanici hanno elevato l'interesse nel campo della comunicazione autonoma tra agenti in grado di svolgere, anche in presenza di incertezza ed avversità, compiti che vanno oltre la capacità dei singoli. Le aree di applicazione coprono una vasta gamma di settori, quali ad esempio esplorazione, ricognizione, sorveglianza di aree, sistemi di targeting, controlli, comunicazione, monitoraggio ambientale, intrusion detection, monitoraggio e mappatura dei veicoli, applicazioni militari, e molte altre. Nonostante ciascuna di queste aree ponga le proprie specifiche difficoltà, si possono tuttavia ritrovare parecchi aspetti in comune.
Nella maggior parte dei casi, gli agenti interagiscono tra loro per svolgere il compito loro affidato, ma sono altresì disaccoppiati l'un l'altro, nel senso che il comportamento di uno non influisce direttamente sul comportamento degli altri. Ognuno di essi utilizza lo stesso algoritmo condiviso da tutti, che permette di prendere una decisione sulla base della sola informazione disponibile localmente, informazione che può essere soggetta ad incertezze e ritardi di trasmissione. Questo approccio di gestione dell'informazione è detto per tale ragione "distribuito". La sincronizzazione temporale costituisce un blocco fondamentale per l'infrastruttura di ogni sistema distribuito. In ognuna delle applicazioni sopracitate è essenziale che i nodi agiscano in maniera sincronizzata e coordinata richiedendo al sistema un andamento di sincronizzazione globale degli orologi, si abbia cioè che ogni nodo della rete faccia riferimento ad una comune nozione di tempo, detta "tempo assoluto". Le reti di sensori wireless (WSN) fanno un uso particolarmente estensivo della sincronizzazione che per esempio è utilizzata: per integrare una serie di rilevazioni di prossimità per stimare il movimento di uno o più veicoli; per misurare il tempo di propagazione del suono e localizzarne la sorgente o, ancora, per sopprimere messaggi ridondanti riconoscendo che questi messaggi descrivono duplicazioni dello stesso evento registrato da diversi sensori. Per tali ragioni sono infatti richiesti accurati riferimenti temporali (*timestamps*) nello scambio di informazioni tra i nodi della rete. La larga natura delle applicazioni riguardanti le reti di agenti porta ad esigenze di temporalizzazione che solitamente non sono così scontate da implementare. Un approccio abbastanza ingenuo per raggiungere la sincronizzazione è quello di creare una struttura gerarchica all'interno della rete, ma questo tipo di strategia, oltre al fatto di portare alla perdita dell'approccio totalmente distribuito, può non garantire buone prestazioni dal punto di vista della robustezza e della scalabilità. Ad ogni modo, affinché una strategia di controllo cooperativo sia efficace, il gruppo di agenti deve essere in grado di rispondere simultaneamente ad imprevisti o cambiamenti dell'ambiente condiviso. Un esempio tipico di condivisione dell'informazione per la coordinazione è il raggiungimento di un medesimo valore da parte di tutti gli agenti relativamente ad un dato rilevante per la coordinazione del gruppo. In letteratura il raggiungimento di un valore comune è chiamato *problema di consensus*.
Il tipo di algoritmi che verranno presentati all'interno di questo lavoro sono a carat-

tere totalmente distribuito in cui ogni agente è in grado di prendere autonomamente decisioni riguardo il proprio stato. Lo scopo di questo progetto è quello di illustrare e cercare di mettere a confronto tre algoritmi di sincronizzazione temporale: "Average-TimeSynch" di Schenato et al. [3], "A Distributed Time Synchronization Protocol" di Kumar et al. [5], "A PI Consensus Controller for Networked Clocks Syncronization" di Zampieri, Carli, Schenato et al.[9].
Valutare le proprietà di un algoritmo di sincronizzazione per un sistema distribuito è una cosa piuttosto complicata. Il problema è stabilire una serie di metriche standard da utilizzare per confrontare gli algoritmi in modo da valutarli equamente. Le principali metriche che sono state utilizzate per confrontare le prestazioni degli algoritmi di sincronizzazione presi in considerazione sono: la precisione (valore legato all'errore di sincronizzazione, cioè alla varianza delle stime che ciascun nodo fa della propria legge oraria), robustezza a parametri di disturbo e velocità di convergenza. Nello svolgimento del progetto sono stati considerati rumori di trasmissione e di misura, e tempo-varianza delle frequenze degli orologi che tenessero conto della non idealità del modello di evoluzione dello stato e di possibili eventuali ritardi o disturbi nei canali di comunicazione tra gli agenti. Gli algoritmi verranno inizialmente descritti ed implementati in una versione sincrona (fortemente irrealistica) in cui si suppone che i nodi comunichino e si aggiornino tutti negli stessi istanti; successivamente verrà presentata una implementazione (più realistica) asincrona.

2 Problemi di consenso e coordinazione multi-agente

2.1 Accenni di teoria dei grafi

E' naturale modellare lo scambio di informazione, tra un gruppo di agenti coordinati, tramite grafi orientati o non orientati. Un grafo diretto o digrafo consiste in coppie $(\mathcal{N}, \epsilon)$ dove $\mathcal{N}$ è un insieme finito (non vuoto) di nodi detto insieme dei vertici e $\epsilon \in \mathcal{N}^2$ è un set di coppie ordinate di nodi chiamate archi. L'insieme $\mathcal{N}^2$ è detto insieme dei lati. Al contrario, le coppie di nodi in un grafo non orientato, o indiretto, non sono ordinate. Un percorso diretto in un digrafo è una sequenza ordinata di archi della forma (v_{i1}, v_{i2}), (v_{i2}, v_{i3}),..., dove $v_{ij} \in \mathcal{N}$.
Un digrafo è detto fortemente connesso se esiste un cammino diretto per ogni nodo verso ogni altro nodo. Un grafo indiretto è detto connesso se esiste un cammino tra ogni distinto paio di nodi. Un albero ricoprente o albero di connessione (spanning tree) di un digrafo, è un albero diretto, formato dai lati del grafo che connettono tutti i nodi del grafo. Si dice che un grafo ha (o contiene) un albero ricoprente, se esiste un albero ricoprente che è un sottografo del grafo in considerazione. Quest'ultima assunzione equivale a dire che esiste un nodo del grafo che ha un cammino diretto verso tutti gli altri nodi.
Un nodo v_i di un digrafo $(\mathcal{N}, \epsilon)$ è detto bilanciato se e solo se il numero di archi entranti e uscenti si equivalgono. Conseguentemente si definisce grafo bilanciato un digrafo i cui nodi sono tutti bilanciati. Ogni grafo indiretto è bilanciato.
La matrice di adiacenza $A = [a_{ij}]$ di un digrafo pesato è definita come $a_{ij} = 0$ se $i = j$ e $a_{ij} > 0$ se $(j, i) \in \epsilon$ dove $i \neq j$. La matrice Laplaciana di un digrafo pesato è definita come $L = [l_{ij}]$, dove $l_{ii} = \sum_j^n a_{ij}$ e $l_{ij} = -a_{ij}$ dove $i \neq j$. Per un grafo indiretto la matrice Laplaciana è simmetrica e semi-definita positiva.

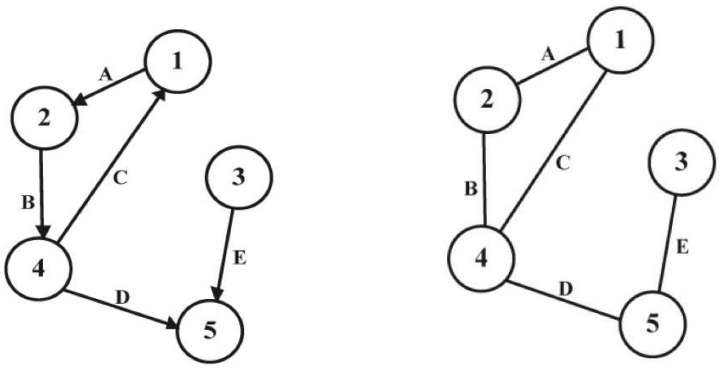


Figura 1: grafo orientato (digrafo) e non orientato (indiretto) con 5 nodi e 5 archi.

Rappresentazione Matriciale dei grafi

Sia $M_n(\mathbb{R})$ il set di matrici $n \times n$ reali. Data la matrice $A = [a_{ij}] \in M_n(\mathbb{R})$, il digrafo di A denotato da $\gamma(A)$ è il digrafo di n vertici v_i, con $i \in I$ così che ci sia un lato diretto in $\gamma(A)$ da v_j a v_i se e solo se $a_{ij} \neq 0$. Una matrice è non negativa (positiva) se tutti i suoi elementi sono non negativi (positivi). Ancora, se la somma di ogni sua riga è uguale ad 1, la matrice è detta matrice stocastica . Una matrice stocastica P è detta indecomponibile ed aperiodica (SIA) se $\lim_{k\to 0} P_k = \mathbf{1}v^T$ dove $\mathbf{1}$ è un vettore colonna formato da tutti 1 e v è un vettore colonna . Il raggio spettrale di una matrice A è definito come il massimo modulo degli autovalori della matrice: $\rho(A) = max_k|\lambda_k(A)|$ dove $\lambda_k(A)$ sono gli autovalori di A.
Il teorema di Perron-Frobenius afferma che se una matrice A è non negativa ed irriducibile (ovvero non riducibile nella forma diagonale a blocchi), e quindi il digrafo della matrice è fortemente connesso, allora $\rho(A)$ è costituito da un autovalore semplice di A associato ad un autovettore positivo, dove $\rho(A)$è il raggio spettrale della matrice. Sia A una matrice positiva. Diremo che essa è primitiva se esiste $n > 1$ tale che A^n abbia tutti i suoi elementi > 0. Se A è una matrice primitiva, e quindi A è irriducibile e $\rho(A)$ è un unico autovalore di massimo modulo, allora $\lim_{k\to\infty} k[\rho(A)^{-1}A^k] \to wv^T$ dove v e w sono rispettivamente l'autovettore destro e sinistro della matrice A associati all'autovalore $\rho(A)$ soddisfacente $w^Tv = 1$. I risultati sulle catene Markoviane mostrano che se una matrice stocastica A soddisfa la condizione $A^m > 0$ per qualche intero positivo m , allora $lim_{k\to\infty}A^k \to \mathbf{1}v^T$, dove v è un vettore colonna positivo soddisfacente $\mathbf{1}^Tv = \mathbf{1}$. Infatti la condizione $A^m > 0$ per un intero m positivo equivale alla condizione che A è irriducibile e $\rho(A)$ è costituito da un unico valore positivo a massimo modulo.

2.2 Il protocollo di consenso

Si consideri un insieme di n agenti e sia x_i lo stato dell' iesimo agente. L'informazione contenuta nello stato di ogni agente è proprio l'informazione che serve per la coordinazione degli agenti. Questa informazione può essere: la posizione, la velocità, l'orientamento, la fase, la variabile di decisione. Generalmente si indica con $\mathcal{N}_i$ l'insieme dei vicini del nodo i e con $d(i)$ la somma di questi, cioè il suo grado.
In tempo discreto il protocollo del consenso può essere espresso come:

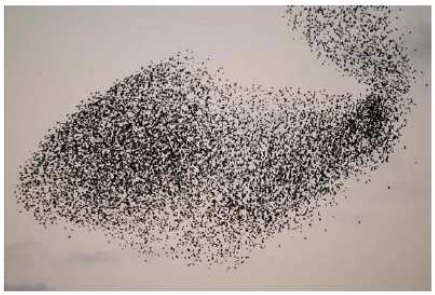
$$x_i(t+1) = \sum_{j\in\mathcal{N}_i(t)} W_{ij}(t)x_j(t) \tag{1}$$

dove $\sum_{j\in\mathcal{N}_i(t)} W_{ij}(t) = 1$ e $W_{ij}(t) > 0$ per $j \in \mathcal{N}_i(t)\bigcup i$, $W_{ij}(t) = 0$ altrimenti.
In altre parole, il prossimo stato di ogni agente è aggiornato attraverso il peso del proprio stato corrente e il peso degli stati correnti dei suoi vicini. Notare che ogni agente, ad ogni passo, mantiene il suo stato attuale se non vi è scambio di informazione con gli altri agenti in quell'istante.
Il protocollo del consenso (1) può essere scritto in forma matriciale come:

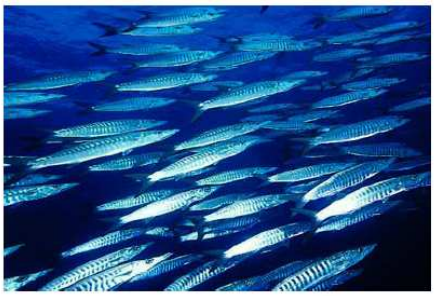
$$x(t+1) = P(t)x(t) \quad \text{con} \quad P(t) = \begin{cases} W_{ij} & \text{se } j \in \mathcal{N}_i(t) \\ 0 & \text{altrimenti} \end{cases} \tag{2}$$

dove $P(t)$ è una matrice stocastica con la diagonale positiva. Per una squadra di agenti si dice che si raggiunge il consenso se:

$$\|x_i - x_j\| \to 0 \qquad \text{per } t \to \infty \ , \forall i \neq j \ .$$



(a) *Stormo di uccelli.*



(b) *Branco di barracuda.*

(c) *Mandria di gazzelle.*

(d) *sciame di api.*

Figura 2.2: Alcuni esempi di flocking in sistemi multi-agente in natura.

Convergenza per un sistema tempo-invariante

Considerando una topologia con scambio d'informazione tempo-invariante, si assume che se un agente può accedere all'informazione di un altro agente in un istante di tempo, esso può ottenere l'informazione dal medesimo agente per ogni altro istante. Per il protocollo del consenso tempo discreto (1), si può vedere che tutti gli autovalori di P che non sono uguali ad 1 sono dentro il cerchio unitario del teorema di Gerschgorin. Se 1 è un autovalore semplice di P e tutti gli altri autovalori hanno modulo minore di uno, sappiamo che $\lim_{t\to\infty} P^t = \mathbf{1}\mu^T$, dove μ è un vettore colonna. Questo implica che $\|x_i - x_j\| \to 0$. Troviamo che la matrice stocastica P ha un unico autovalore di modulo 1 se e solo se il suo digrafo ha un albero ricoprente. Quindi,

considerando un protocollo del consenso tempo-discreto con topologia dell'informazione tempo invariante, questo raggiunge il consenso asintoticamente se e solo se la topologia dello scambio di informazione ha un albero ricoprente. Ora che sappiamo sotto quali condizioni il protocollo del consenso converge, il passo successivo è determinare a quale stato di equilibrio il protocollo converge. Nel caso in cui la topologia di scambio di informazione ha un albero ricoprente, sappiamo che $\lim_{t\to\infty} P^t = \mathbf{1}\mu^T$, dove $\mu^T = [\mu_1, \ldots, \mu_n]^T$ è autovettore non negativo sinistro di P, corrispondente all'autovalore 1 e soddisfa $\sum \mu_j = 1$.
Come risultato $x(t) \to \sum \mu_j x_j(0)$. Quindi lo stato finale di equilibrio è la media pesata della condizione iniziale di ogni agente. Comunque non è chiaro quale sia effettivamente il contributo di ogni agente per raggiungere lo stato di equilibrio finale. Nel caso che la topologia dello scambio di informazione sia fortemente connessa, sappiamo che μ_j è positivo. Quindi, in questo caso, ogni condizione iniziale di ogni singolo agente, contribuisce al raggiungimento dello stato del consenso. Ancora, se $\mu_i = \mu_j = \frac{1}{n}$ dove $i \neq j$, il consenso finale sarà la media delle condizioni iniziali di ogni agente. In questo caso lo stato finale del consenso è chiamato *average consensus*. L'*average consensus* è raggiunto nel caso in cui la topologia di scambio dell'informazione sia fortemente connessa e bilanciata.

Convergenza per un sistema tempo-variante

Anche se i problemi del consenso sono sensibilmente semplificati assumendo una topologia di scambio dell'informazione tempo-invariante, in realtà questa topologia può cambiare dinamicamente. Per esempio la comunicazione tra alcuni agenti può essere irrealizzabile in certi istanti a causa di disturbi e/o limitazioni. Se le informazioni sono date dal rilevamento diretto, i vicini localmente visibili di un agente probabilmente cambieranno col passare del tempo. Questo chiaramente porta ad un cambiamento del grafo di connessione ad ogni istante. Un approccio per studiare questa continua variazione di topologia è considerare la struttura algebrica corrispondente alla matrice del grafo. La soluzione del protocollo del consenso in tempo discreto $x(t+1) = P(t)x(t)$, può essere scritta come: $x(t) = P(t)\ldots P(1)P(0)x(0)$. Equivalentemente si può studiare la proprietà del prodotto di matrici stocastiche. Si dimostra che se $S = S_1, S_2, \ldots, S_k$ è un set di matrici SIA (Stocastiche Indecomponibili Aperiodiche), con la proprietà che per ogni sequenza $S_{i1}, S_{i2}, \ldots, S_{ij}$ di lunghezza positiva, la matrice prodotto $S_{ij}S_{ij-1}, \ldots, S_{i1}$ è SIA, allora per ogni infinita sequenza $S_{i1}, S_{i2}, \ldots$ esiste un vettore colonna tale che $\lim_{j\to\infty} S_{ij}S_{ij-1}, \ldots, S_{i1} = \mathbf{1}v^T$. Numerose ricerche hanno mostrato che il consenso è raggiunto asintoticamente se l'unione dei grafi dello scambio di informazione, per il gruppo di agenti, è connessa per la maggior parte del tempo in cui evolve il sistema. Ovvero, il consenso può essere raggiunto asintoticamente se i grafi dello scambio delle informazioni hanno abbastanza alberi ricoprenti man mano che il sistema evolve.

3 Il problema della sincronizzazione temporale

La sincronizzazione temporale è un elemento chiave per ogni sistema distribuito. Negli ultimi anni in particolar modo si è affermato l'interesse per lo studio delle WSNs (*Wireless Sensor Networks*); queste sono utilizzate in applicazioni industriali e commerciali per monitorare dati i quali sarebbero difficilmente o inconvenientemente monitorabili utilizzando apparecchiature cablate. Queste applicazioni includono il *monitoraggio di ambienti*, il *controllo di macchine industriali*, le *apparecchiature domestiche*, il *tracking di oggetti* e la *event detection*, ecc.

Figura 2: Esempio di sonda utilizzata per il monitoraggio di ambienti.

Molte di queste applicazioni richiedono una collaborazione distribuita per l'esecuzione di compiti ripartiti in un grande insieme di agenti sincronizzati. Inoltre le operazioni di *data fusion*, *power management* e *transmission scheduling* richiedono che tutti i nodi si riferiscano ad istanti di tempo comuni. Comunque, ogni singolo sensore in una WSN ha il proprio orologio interno. La distanza "temporale" tra orologi differenti può essere dovuta a vari fattori, come imperfezione degli oscillatori e cambiamenti ambientali. Questo rende la sincronizzazione temporale un requisito indispensabile nella progettazione di sistemi di controllo.

3.1 Problematiche

In generale, tra le varie problematiche che si possono incontrare nello sviluppo di un algoritmo di sincronizzazione temporale per una rete di nodi, ci sono: la scelta tra una implementazione di tipo centralizzato o distribuito, sincrono o asincrono, il raggiungimento di un certo grado di **robustezza** in termini di *node failure* e *lossy comunication*, la **velocità di convergenza** al consenso, la **stabilità** dell'intero sistema, l'**ottimizzazione** del protocollo stesso e la decisione relativa alle assunzioni

semplificative che si adottano. Vediamo ora in dettaglio cosa si intende per:

- **Approccio distribuito** si ha quando ogni nodo ha eguale potere decisionale in merito alle azioni da compiere.
- **Approccio centralizzato** si ha quando la decisione sull'operato è affidata ad un unico nodo (*leader*) che trasmette a tutti gli altri le azioni da intraprendere.
- **Implementazione asincrona** si ha quando la comunicazione tra i nodi avviene ad istanti temporali non prefissati e pertanto non è necessaria una particolare coordinazione.
- **Implementazione sincrona** si ha quando la comunicazione tra i nodi avviene ad istanti temporali prefissati.
- ***Node failure*** si ha quando all'interno di una rete di comunicazione uno o più nodi cessano definitivamente di comunicare a causa di guasti.
- ***Lossy comunication*** si ha quando nella comunicazione tra due nodi il pacchetto di informazione viene perso in seguito a *random interference* o rumore.

3.2 Approcci proposti

La strategia più banale per la sincronizzazione di due orologi consiste nell'eleggere uno dei due come riferimento e quindi aggiornare periodicamente l'*offset* dell'altro in base alla differenza dei due "tempi". Comunque, se i due orologi hanno differenti velocità, la differenza di tempo tende a divergere mostrando la caratteristica evoluzione a *saw-tooth*. Un metodo per compensare le differenti velocità temporali è di stimare la velocità relativa ed usarla per predire correttamente il tempo del nodo di riferimento. Una naturale estensione della precedente strategia di sincronizzazione temporale per una rete di orologi è quella di eleggere innanzitutto una radice e di creare un albero a partire dal grafo di comunicazione, e in seguito usare la precedente strategia dove ciascun figlio sincronizza se stesso rispetto al padre. Un altro comune approccio è quello di dividere la rete in distinti *clusters*, ciascuno dei quali con un proprio *cluster head*. Tutti i nodi all'interno dello stesso cluster sincronizzano le proprie leggi orarie con il corrispondente cluster head, e ciascun cluster head a sua volta sincronizza sè stesso con un altro cluster head, si veda [4].

Sebbene queste due strategie possano essere facilmente implementate ed abbiano mostrato notevoli performance esse soffrono di due principali problemi. Il primo problema è la robustezza; infatti, se un nodo muore o un nuovo nodo viene aggiunto alla rete, diventa necessario ricostruire l'albero o i clusters, al prezzo di un'implementazione addizionale e la possibile comparsa di lunghi periodi nei quali la rete, o parte di essa, può trovarsi inadeguatamente sincronizzata. Il secondo problema si verifica quando due orologi, fisicamente vicini e potenzialmente in grado di comunicare l'un l'altro, appartengono invece a differenti rami dell'albero o differenti clusters portandoli ad avere grandi differenze temporali. Questo è particolarmente sconveniente in

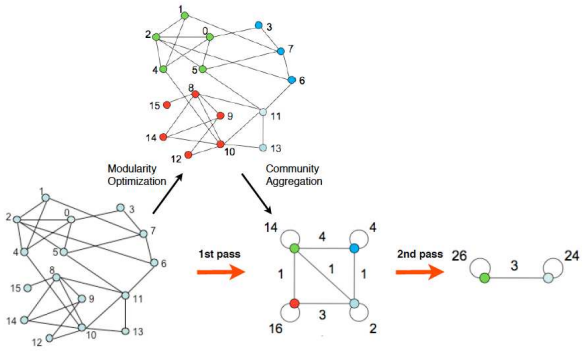


Figura 3: Visualizzazione dei passi dell'algoritmo [4].

quelle applicazioni come la comunicazione TDMA (*Time Division Medium Access*) ove è richiesta una buona sincronizzazione di ciascun nodo con i propri vicini.
Un'altra questione che sorge nell'interconnessione di sistemi è il ritardo che si crea durante le "letture temporali" di due dispositivi distinti. Infatti molti degli algoritmi di sincronizzazione presenti in letteratura assumono che queste vengano compiute istantaneamente. Se questo non è il caso, poiché ad esempio è richiesto dal software un tempo d'accesso non nullo per leggere l'orologio locale o si hanno ritardi di propagazione o di lettura dell'orologio ricevente, allora le performance di sincronizzazione possono peggiorare e addirittura generare instabilità negli algoritmi. Sebbene ci siano algoritmi che tentano di mitigare questi effetti, si veda [5], il trend generale è quello di risolvere questo problema a livello hardware, facendo in modo di leggere correttamente le leggi orarie prima che avvenga la trasmissione del pacchetto e non appena il pacchetto giunge a destinazione.

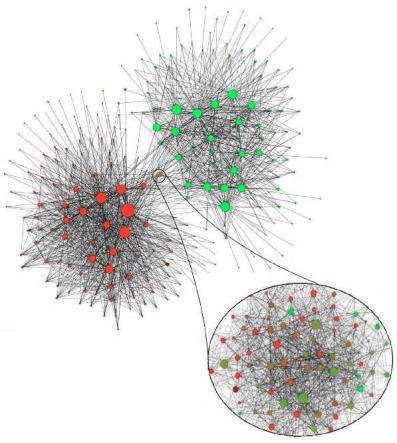


Figura 4: Esempio di rete telefonica Belga (circa 2 milioni di nodi).

4 Algoritmi proposti

In questo lavoro si è deciso di affrontare lo studio di tre particolari algoritmi di sincronizzazione temporale, ciascuno operante secondo specifici criteri che verranno esposti in questa sezione. Si tratta degli algoritmi discussi in [3], [5] e [9].

4.1 Scenario

Per ciascuno di questi algoritmi si è deciso di effettuare una triplice implementazione, una sincrona e due asincrone; in quest'ultimo caso i protocolli di comunicazione utilizzati son stati quelli di tipo broadcast e gossip simmetrico.
Nel caso sincrono l'idea di base è quella per la quale tutti i nodi, ad istanti temporali prefissati, si mobilitano effettuando uno scambio di pacchetti con tutti i nodi che hanno una connessione con essi.
Il protocollo di comunicazione broadcast prevede che, ad istanti di tempo random, un solo nodo della rete invii il proprio pacchetto d'informazione a tutti i suoi vicini, i quali aggiornano il proprio stato a ricezione avvenuta.
Per quanto riguarda invece il protocollo gossip (simmetrico) vengono messi in comunicazione due nodi vicini, ciascuno scelto in maniera random. In tutti i casi si considera che la trasmissione dei dati da nodo a nodo avvenga in maniera istantanea.

Dal punto di vista dell'implementazione MATLAB sono stati sviluppati due *Main*, uno per il caso sincrono ed uno per quello asincrono. In entrambi, i parametri sui quali è possibile agire sono:

- Numero di nodi: N
- Quanto temporale: T
- Numero di iterazioni dell'algoritmo: $Iter$
- Raggio di connessione del grafo: r
- Rumore: $sigma\_n$
- Parametro per l'inizializzazione delle velocità: $sigma$

Per tutte e tre le tipologie gli orologi sono stati modellizzati con offset variabili in maniera random all'interno dell'intervallo [0,10]. E' stata introdotta poi la possibilità di intervenire rendendo tempo-varianti le frequenze di oscillazione degli orologi (*random walk*), il che è stato realizzato attraverso l'uso di una variabile aleatoria uniformemente distribuita, nel codice indicata con $value\_vel$, facendo attenzione che le velocità saturassero nel caso di valori esterni all'intervallo $[0.9, 1.1]$. La stessa strategia è stata poi sfruttata nel caso asincrono per modellizzare la comunicazione che, a differenza del caso sincrono, non è prodotta da un tempo assoluto scandito in istanti equispaziati. Ciò è evidente se si pensa che nel caso asincrono non è possibile prevedere gli istanti in cui i nodi, prossimi a comunicare, "si sveglieranno". Con la

variabile *sigma_n* si sono volute rappresentare tutte le possibili cause di disturbo che si generano nella comunicazione tra i nodi della rete, mentre con la variabile *sigma* viene generata una variabile aleatoria uniforme con la quale vengono inizializzate le velocità degli N nodi. In uscita, ogni algoritmo produce la varianza delle stime dei tempi.
Và detto infine che tutte le simulazioni sono state prodotte secondo il metodo *Montecarlo Simulations*, dove la risposta di ciascun algoritmo è stata mediata sulla base di 50 realizzazioni, ciascuna delle quali utilizza un diverso grafo di tipo *random geometric* e differenti condizioni iniziali. Per variare il numero di realizzazioni si è utilizzata la variabile *cycle*. Ciascun plot riporta solo le parti salienti di ogni test effettuato (transitorio) appartenente a simulazioni della durata totale di 10000 iterazioni.

4.2 Algoritmo ATS

In questa sezione si analizzerà e testerà il protocollo ATS (Average TimeSync) [3] che si prefigge di sincronizzare una rete di sensori wireless sfruttando il principio dell'*average consensus*. Esso fa parte della classe degli algoritmi distribuiti ed è costituito dalla cascata di due algoritmi di consenso. Questo protocollo si basa su un'idea di gestione dell'informazione che è completamente distribuita e localizzata, cioè la sincronizzazione è fatta localmente senza il bisogno di un'inizializzazione da parte di nodi *leader*. Per tale ragione ATS risulta molto robusto in caso di guasto di alcuni nodi. Grazie a questo tipo di configurazione ogni nodo può inizializzare o terminare un processo di sincronizzazione. L'idea principale dell'algoritmo è quella di livellare tutti i valori dei clock di ciascun nodo alla loro media globale. Un nodo aggiorna il proprio valore di clock incrementandolo o decrementandolo, in base al pacchetto d'informazione ricevuto dai suoi vicini; dopo alcune iterazioni il clock di ciascun orologio raggiungerà lo stesso valore.

MODELLO

Modellizziamo la rete di orologi con un grafo $\mathscr{G}(V, \mathscr{E})$ di N nodi, dove
$V = \{1, \ldots, N\}$ è l'insieme dei "vertici" ed $\mathscr{E} \subseteq V \times V$.
Consideriamo la comunicazione $j \longrightarrow i$ e limitiamoci a descrivere il caso sincrono in cui tutti i nodi comunicano nello stesso istante hT; chiamiamo con N_i l'insieme dei nodi che comunicano al nodo i il loro tempo locale: $N_i = \{j | (j, i) \in \mathscr{E}\}$.

Orologio locale

Ciascun orologio i modellizza il proprio oscillatore con la seguente legge oraria:

$$\tau_i(t) = \alpha_i t + o_i$$

in cui τ_i è il valore del clock, α_i il coefficiente di velocità e o_i quello di offset.

Consideriamo che la comunicazione avvenga agli istanti t_1, t_2, con $t_2 > t_1$.

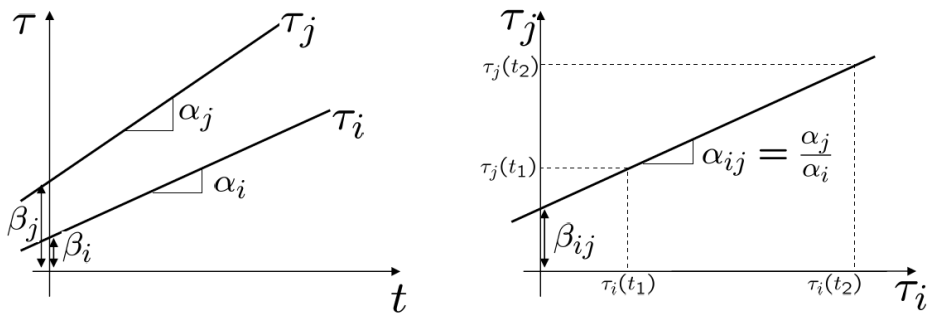


Figura 5: Dinamiche degli orologi come funzione del tempo assoluto (sx), e relativa tra i due (dx). L'offset in questi grafici è rappresentato col simbolo β.

Ogni orologio non è a conoscenza della propria velocità e del proprio offset, ma può solo leggere e trasmettere il suo orologio locale τ_i. Pertanto la prima cosa che tenta di fare è stimare $\alpha_{ij} = \frac{\alpha_j}{\alpha_i}$, cioè la sua velocità relativa rispetto alle velocità dei suoi vicini:

$$\eta_{ij}^{+}(t) = \rho\eta_{ij}(t) + (1-\rho)\frac{\tau_j(t_2) - \tau_j(t_1)}{\tau_i(t_2) - \tau_i(t_1)}$$

η_{ij} rappresenta la stima del rapporto $\frac{\alpha_j}{\alpha_i}$ che il nodo i possiede.

Possiamo considerare il parametro ρ di tuning come il fattore che pesa l'importanza dell'informazione che il nodo possiede all'istante t e quella che il nodo possiede dopo la successiva ricezione. Tale parametro compensa quindi la presenza di disturbi:

senza disturbi $i \to j$ ($\rho = 0$), si ha

$$\begin{aligned} \eta_{ij}^{+}(t) &= \frac{\tau_j(t_2) - \tau_j(t_1)}{\tau_i(t_2) - \tau_i(t_1)} \\ &= \frac{\alpha_j t_2 + \beta_j - \alpha_j t_1 + \beta_j}{\alpha_i t_2 + \beta_i - \alpha_i t_1 + \beta_i} \\ &= \frac{\alpha_j(t_2 - t_1)}{\alpha_i(t_2 - t_1)} \\ &= \frac{\alpha_j}{\alpha_i} \end{aligned}$$

con elevati disturbi $i \to j$ ($\rho = 1$), si ha invece

$$\eta_{ij}^{+}(t) = \eta_{ij}(t) \qquad .$$

Stima locale dell'orologio virtuale

Come premesso, lo scopo dell'algoritmo è di sincronizzare tutti i nodi rispetto ad un clock virtuale di riferimento

$$\tau_v(t) = \alpha_v t + o_v \quad .$$

Ciascun orologio effettua una stima di questo in base alla propria legge temporale:

$$\hat{\tau}_i = \hat{\alpha}_i \tau_i + \hat{o}_i \quad .$$

Quello che vorremmo è che $\hat{\tau}_i \to \tau_v$, per ogni nodo i della rete.

Aggiornamento della stima delle velocità

Una volta che si ha la stima η_{ij} della velocità relativa, è possibile ricavare la stima della velocità di ciascun orologio locale:

$$\hat{\alpha}_i(hT) = P_{ii}\hat{\alpha}_i((h-1)T) + \sum_{j \in N_i} P_{ij}\eta_{ij}(hT)\hat{\alpha}_j((h-1)T)$$

dove P rappresenta la matrice di consenso che verrà definita in seguito.

Aggiornamento della stima degli offset

Per trovare la stima dell'offset di ciascun orologio è necessario conoscere le stime dell'orologio virtuale fatte dal nodo i, ricevente, e dai nodi vicini j, trasmettenti:

$$\begin{aligned} \hat{o}_i(hT) &= \hat{o}_i((h-1)T) + \sum_{j \in N_i} P_{ij}(\hat{\tau}_j((h-1)T) - \hat{\tau}_i((h-1)T)) \\ &= \hat{o}_i((h-1)T) + \sum_{j \in N_i} P_{ij}\hat{\tau}_j((h-1)T) - \sum_{j \in N_i} P_{ij}\hat{\tau}_i((h-1)T) \\ &= \hat{o}_i((h-1)T) + \sum_{j \in N_i} P_{ij}\hat{\tau}_j((h-1)T) - (1 - P_{ii})\hat{\tau}_i((h-1)T) \end{aligned}$$

Definiamo ora le seguenti matrici che torneranno utili per l'implementazione dell'algoritmo in MATLAB:

- matrice delle stime delle velocità relative

  $$H(hT) \quad | \quad \{H(hT)\}_{ij} = \eta_{ij}(hT) \qquad (\eta_{ii}(hT) = 1 \qquad \forall h \in \mathbb{N})$$

- vettore delle stime delle velocità

  $$\hat{\alpha}(hT) = \begin{bmatrix} \hat{\alpha}_1(hT) \\ \vdots \\ \hat{\alpha}_N(hT) \end{bmatrix}$$

- vettore delle stime degli offset

  $$\hat{o}(hT) = \begin{bmatrix} \hat{o}_1(hT) \\ \vdots \\ \hat{o}_N(hT) \end{bmatrix}$$

- vettore delle stime degli orologi locali

  $$\hat{\tau}(hT) = \begin{bmatrix} \hat{\tau}_1(hT) \\ \vdots \\ \hat{\tau}_N(hT) \end{bmatrix}$$

- matrice di consenso con assegnazione dei pesi secondo la strategia Metropolis

  $$P = \begin{cases} P_{ii} = 1 - \sum_{j \in N_i} P_{ij} \\ P_{ij} = \frac{1}{max\{d(i),d(j)\}+1} \\ P_{ij} = 0 \quad \text{altrimenti} \end{cases}$$

Si ha che

1. $\hat{\alpha}(hT) = (P \odot H(hT))\hat{\alpha}((h-1)T)$
2. $\hat{o}(hT) = \hat{o}((h-1)T) + (P - I)\hat{\tau}((h-1)T)$
3. $\hat{\tau}(hT) = (\hat{\alpha}(hT) \odot \tau(hT)) + \hat{o}(hT)$

La variabile z utilizzata nel codice è

$$z(hT) = \frac{1}{N} \cdot \mathrm{var}(\hat{\tau}(hT))$$

cioè la varianza normalizzata delle stime dei tempi.

RICAPITOLANDO

$$i \longleftrightarrow j$$

nodo i trasmette $\quad \tau_i((h-1)T),\ \tau_i(hT),\ \hat{\alpha}_i((h-1)T),\ \hat{\tau}_i((h-1)T)$

nodo i riceve $\quad \tau_j((h-1)T),\ \tau_j(hT),\ \hat{\alpha}_j((h-1)T),\ \hat{\tau}_j((h-1)T), \quad \forall j \in N_i$

1. **condizioni iniziali**

$$\begin{aligned} \eta_{ij}(0) &= \widehat{\frac{\alpha_j(0)}{\alpha_i(0)}} = 1 \\ \hat{\alpha}_i(0) &= 1 \\ \hat{o}_i(0) &= 0 \end{aligned}$$

   Si suppone approssimativamente che in partenza gli orologi abbiano la stessa velocità; da qui la ragione di porre $\eta_{ij}(0) = 1$.

2. **aggiornamento della stima delle velocità relative**

$$\eta_{ij}(hT) = \rho\ \eta_{ij}((h-1)T) + (1-\rho)\frac{\tau_j(hT) - \tau_j((h-1)T)}{\tau_i(hT) - \tau_i((h-1)T)} \qquad \forall j \in N_i$$

3. **aggiornamento della stima delle velocità**

$$\begin{aligned} \hat{\alpha}_i(hT) &= P_{ii}\hat{\alpha}_i((h-1)T) + \sum_{j \in \eta_i} P_{ij}\eta_{ij}(hT)\hat{\alpha}_i((h-1)T) \\ \hat{\alpha}(hT) &= (P \odot H(hT))\hat{\alpha}((h-1)T) \end{aligned}$$

4. **aggiornamento delle stime degli offset**

$$\begin{aligned} \hat{o}_i(hT) &= (\hat{o}_i((h-1)T) + \sum_{j \in N_i} P_{ij}(\hat{\tau}_j((h-1)T)) - \hat{\tau}_i((h-1)T)) \\ \hat{o}(hT) &= \hat{o}((h-1)T) + (P - I)\hat{\tau}((h-1)T) \end{aligned}$$

5. **stime degli orologi locali**

$$\begin{aligned} \hat{\tau}_i(hT) &= (\hat{\alpha}_i(hT)\tau_i(hT)) + \hat{o}_i(hT) \\ \hat{\tau}(hT) &= (\hat{\alpha}(hT) \odot \tau(hT)) + \hat{o}(hT) \end{aligned}$$

Per quanto riguarda la memoria, ciascun nodo i, per ogni $t \in ((h-1)T, hT)$, memorizza i valori $\tau_i((h-1)T)$, $\hat{\alpha}_i((h-1)T)$, $\hat{o}_i((h-1)T)$, $\hat{\tau}_i((h-1)T)$, $\{\eta_{ij}((h-1)T)\}_{j\in N_i}$ e ρ.

Quanto detto si può facilmente estendere anche al caso asincrono, a patto di considerare un protocollo di comunicazione nel quale non tutti i nodi comunicano contemporaneamente.

4.2.1 Caso sincrono e relative simulazioni

Vediamo in questa sezione l'andamento dell'algoritmo nel caso sincrono testato avendo imposto per tutte le simulazioni i seguenti parametri:

- $N = 10$
- $T = 1$
- $Iter = 10000$
- $r = 0.28$

Per quanto riguarda le condizioni iniziali di stima si è supposto, approssimativamente, che in partenza gli orologi avessero stessa velocità e offset nullo; da qui la ragione di porre $\eta_{ij}(0) = 1$. Da notare che tra gli N nodi si è scelto di fissarne uno prendendolo a riferimento come andamento ideale di legge temporale.
La seguente tabella riporta i valori relativi ad alcuni test scelti a titolo d'esempio tra tutti quelli prodotti in sede di simulazione, per ciascuno dei quali si è variato il valore del parametro ρ in corrispondenza al variare del rumore e delle frequenze di oscillazione introdotte. Come si può osservare in questo paragrafo si è scelto di testare le prestazioni del protocollo in relazione all'introduzione di disturbi.

	rho	$sigma\_n$	$value\_vel$
DISTURBI NULLI	0	0	0
DISTURBI MEDI	0.5	10^{-3}	10^{-9}
DISTURBI ELEVATI	0.9	10^{-4}	10^{-5}

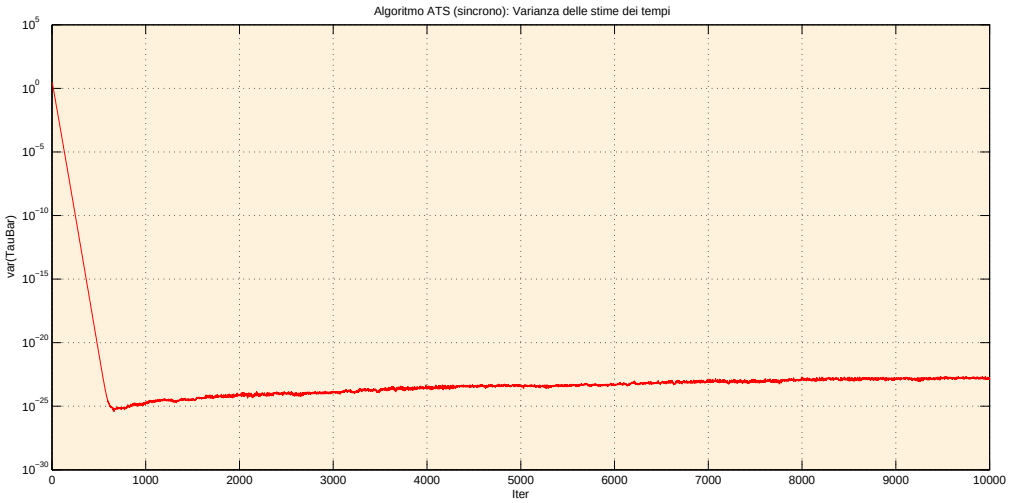


Figura 6: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso ideale, ossia assenza di rumore e velocità tempo-invarianti. Ritenendo quindi che avvenga una comunicazione perfetta il rho è stato impostato a zero. Ciò ha garantito una velocità di convergenza molto buona con varianza "bounded".

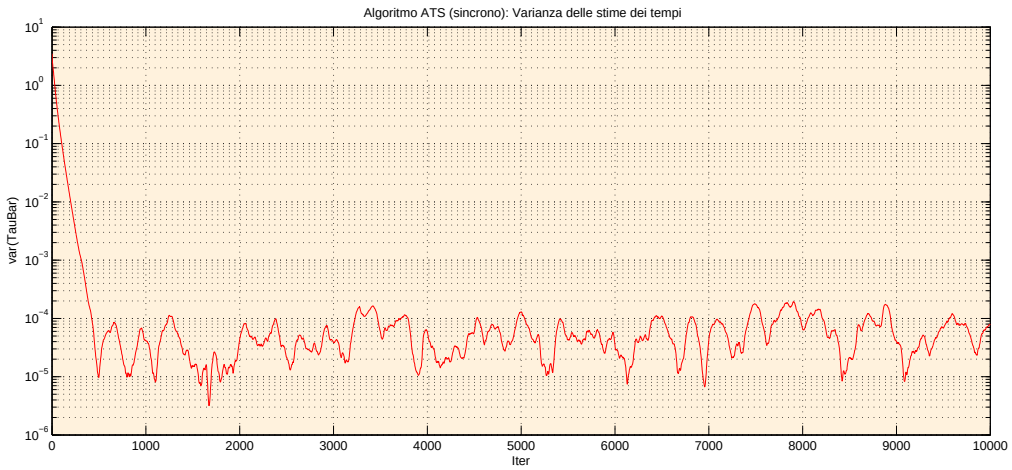


Figura 7: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso di $rho = 0.5$, $sigma\_n = 10^{-3}$ e $value\_vel = 10^{-9}$. Si può vedere che qui il consenso viene raggiunto già dopo circa 500 iterazioni, a scapito però di una varianza sensibilmente maggiore rispetto al caso precedente. Questo porta a pensare che l'aumento del parametro di tuning permetta di gestire bene la presenza di disturbi, ma rallenti leggermente il raggiungimento del consenso.

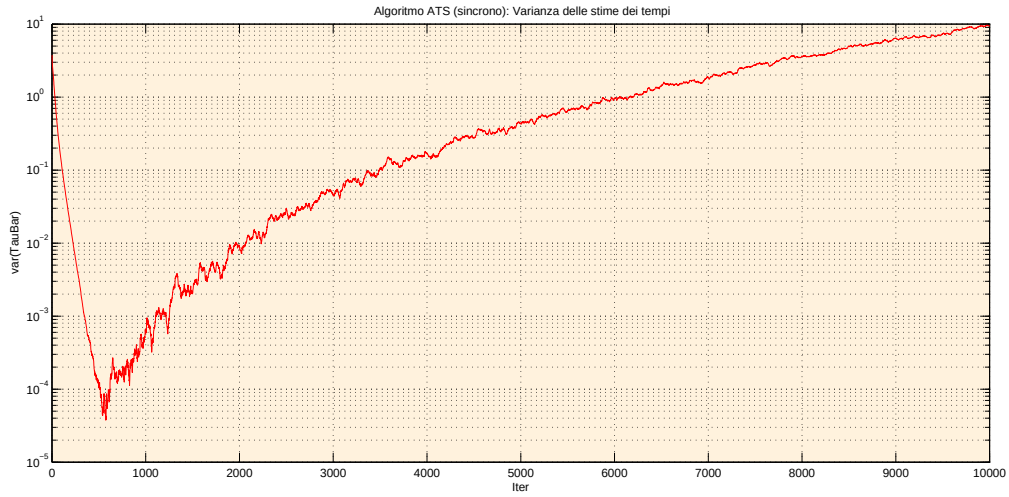


Figura 8: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso di $rho = 0.9$, $sigma\_n = 10^{-4}$ e $value\_vel = 10^{-5}$. Si può vedere che in questo caso la varianza tende a divergere in quanto l'algoritmo non riesce a compensare la presenza di una considerevole variazione delle frequenze nonostante il rho fosse stato spinto al massimo del suo range di variazione.

Dalla campagna di simulazioni effettuate si è osservato che all'aumentare del parametro rho, che può essere visto come un filtro, diminuisce la pendenza della retta che rappresenta la velocità dell'algoritmo al raggiungimento del consenso. Si è giunti quindi a considerare come ottimale il valore 0.5 per il parametro ρ. Esso infatti fornisce, a livello empirico, un buon compromesso tra velocità di convergenza al consenso e gestione di disturbi in termini di varianza delle stime.

4.2.2 Caso asincrono e relative simulazioni

In questa sezione si è deciso di valutare quale dei due protocolli di comunicazione fornisse prestazioni migliori per questo tipo di algoritmo. Vediamo quindi l'andamento nel caso asincrono avendo imposto per tutte le simulazioni i seguenti parametri:

- $N = 10$
- $T = 1$
- $Iter = 10000$
- $r = 0.28$

Le condizioni iniziali sono state fissate identiche al caso sincrono.
Si riporta anche qui, a titolo d'esempio, uno studio relativo all'andamento dell'algoritmo nelle sue due versioni asincrone in corrispondenza all'introduzione di disturbi.

	rho	$sigma\_n$	$value\_vel$
DISTURBI NULLI	0	0	0
DISTURBI MEDI	0.5	10^{-4}	10^{-9}
DISTURBI ELEVATI	0.9	10^{-4}	10^{-5}

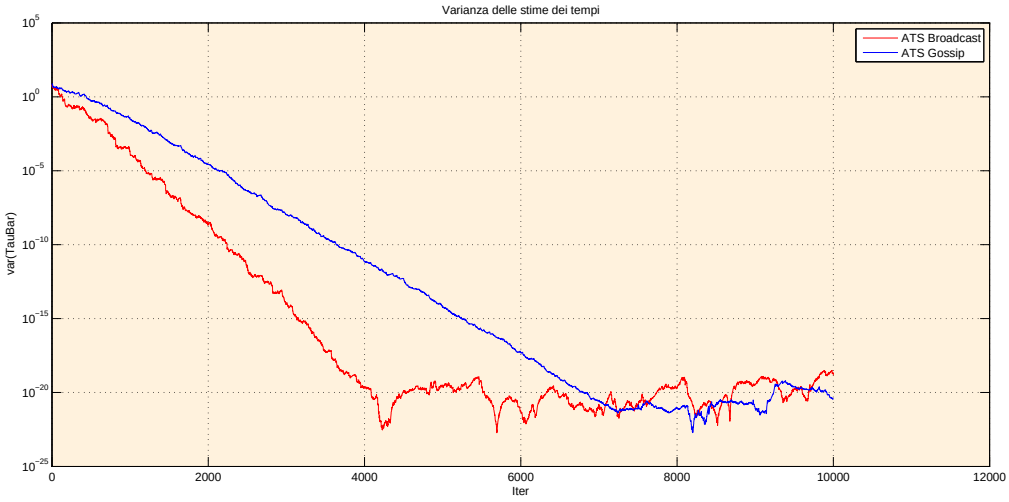


Figura 9: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso ideale, ossia assenza di rumore e velocità tempo-invarianti, quindi $rho = 0$. In entrambi i casi si giunge al consenso con una varianza limitata a valori prossimi a 10^{-25}.

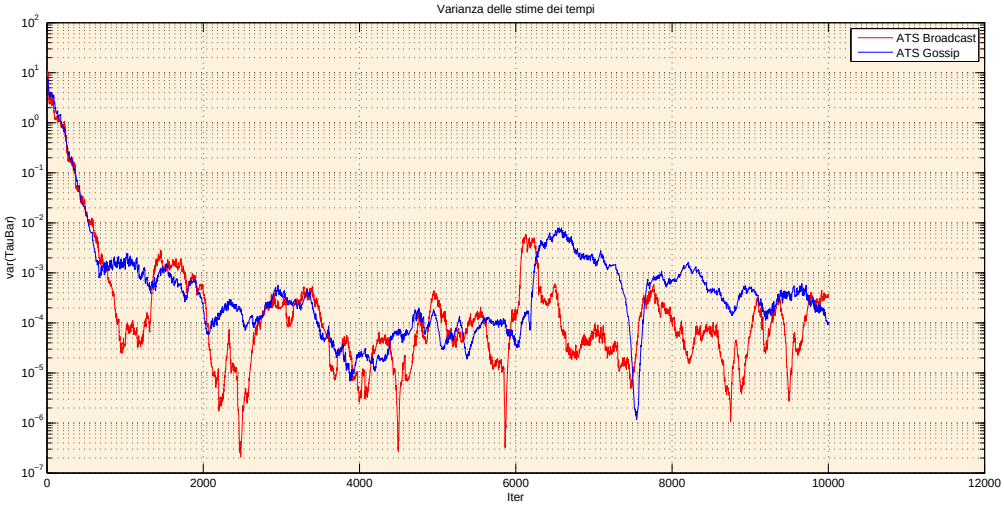


Figura 10: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso di $rho = 0.5$, $sigma\_n = 10^{-4}$ e $value\_vel = 10^{-9}$. Si può vedere che qui il consenso viene raggiunto attorno alle 1000 iterazioni, a scapito però di una varianza sensibilmente maggiore rispetto al caso precedente. Di nuovo si osserva come l'aumento del parametro di tuning permetta di gestire bene la presenza di disturbi, ma rallenti leggermente il raggiungimento del consenso.

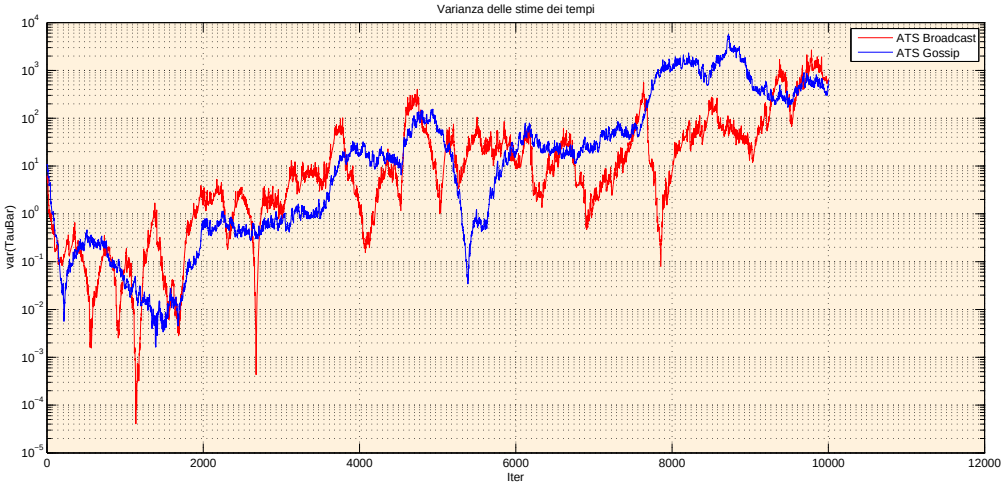


Figura 11: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso di $rho = 0.9$, $sigma\_n = 10^{-4}$ e $value\_vel = 10^{-5}$. Si può vedere che in questo caso la varianza tende a divergere in quanto l'algoritmo non riesce a compensare la presenza di una considerevole variazione delle frequenze nonostante il rho fosse stato spinto al massimo del suo range di variazione.

Effettuando un confronto tra le due implementazioni asincrone si evidenzia una miglior prestazione nel caso di comunicazione broadcast; infatti la varianza in questo caso resta sempre prossima allo zero o comunque a valori inferiori di quelli generati nel caso gossip. Ciò è intuibile considerando il funzionamento dei due protocolli di comunicazione.
Come si vede dai primi due grafici le varianze sono "*bounded*", il che è stato possibile utilizzando negli aggiornamenti la differenza tra le leggi orarie calcolate negli ultimi due istanti anziché computando il tempo considerato solo l'ultimo campione.
Anche qui, come per il caso sincrono, si è giunti a considerare come ottimale il valore 0.5 per il parametro ρ. Esso infatti fornisce, a livello empirico, un buon compromesso tra velocità di convergenza al consenso e gestione di disturbi e tempo-varianza delle frequenze, in termini di varianza delle stime.

4.3 Algoritmo di Kumar

In questa sezione si analizzerà e testerà il protocollo "Solis, Borkar, Kumar" [5] che si prefigge di sincronizzare una rete di sensori wireless sfruttando il principio *least-squares smoothing* per effettuare uno smoothing dell'insieme delle stime ottenute grazie ad una procedura di scambi di pacchetto. Anche questo protocollo fa parte della classe degli algoritmi distribuiti. L'idea da cui parte questo algoritmo è quella di fissare dei vincoli a livello globale e sfruttarli per condurre una stima a livello locale.

MODELLO

Modellizziamo la rete di orologi con un grafo $\mathscr{G}(V, \mathscr{E})$ di N nodi, dove
$V = \{1, \ldots, N\}$ è l'insieme dei "vertici" ed $\mathscr{E} \subseteq V \times V$.
Consideriamo la comunicazione $j \longrightarrow i$ e limitiamoci a descrivere il caso sincrono in cui tutti i nodi comunicano nello stesso istante hT; chiamiamo con N_i l'insieme dei nodi che comunicano al nodo i il loro tempo locale: $N_i = \{j | (j, i) \in \mathscr{E}\}$.

Orologio locale

Le leggi degli orologi i e j sono

$$\begin{aligned} \tau_i &= \alpha_i \, t + o_i \\ \tau_j &= \alpha_j \, t + o_j \end{aligned}$$

j invia:

τ_j = valore del suo orologio locale
$\hat{\alpha}_j$ = stima della sua velocità
$\hat{o}_j$ = stima del suo offset

Si vuole stimare la differenza dei due offset, cioè: $\hat{o}_{ij} = \widehat{o_i - o_j}$. Per farlo è necessario stimare l'istante temporale assoluto con

$$\hat{t}_i = \frac{\tau_i - \hat{o}_i}{\hat{\alpha}_i} \quad .$$

Segue quindi:

$$\begin{aligned} \widehat{o_i - o_j} &= \tau_i - \tau_j - (\hat{\alpha}_i - \hat{\alpha}_j)\,\hat{t}_i \\ &= \tau_i - \hat{\alpha}_i\hat{t}_i - (\tau_j - \hat{\alpha}_j\hat{t}_i\,) \quad . \end{aligned}$$

L'algoritmo si propone di raffinare questa stima utilizzando dei vincoli globali, come ad esempio

$$\sum_{k=1}^{n} o_{i_k i_{k+1}} = 0 \quad \text{per ogni ciclo di nodi } i_1, \ldots, i_n = i_1 \quad .$$

Pertanto il nodo i calcola la stima del suo offset come

$$\hat{o}_i = \sum_{j \in N_i} \frac{1}{|N_i|}(\widehat{o_i - o_j} + \hat{o}_j) \quad .$$

Una volta che il nodo ha calcolato la stima del suo offset deve ricavare la stima della velocità. Per farlo si passa attraverso il metodo *Recursive Least Squares* [1] :

$$\hat{\alpha}_{ij}(hT) = argmin_\alpha \underbrace{\sum_{l=0}^{h-1} \lambda^{l+1-h}\{x_j((l+1)T) - x_j(lT) - \alpha[x_i((l+1)T) - x_i(lT)]\}^2}_{f(\alpha)}$$

$$\begin{aligned} f'(\alpha) &= -2\left(\sum_{l=0}^{h-1} \lambda^{l+1-h}\{x_j((l+1)T) - x_j(lT) - \alpha[x_i((l+1)T) - x_i(lT)]\}\cdot\right. \\ &\quad \left.\cdot[x_i((l+1)T) - x_i(lT)]\right) \end{aligned}$$

$$\alpha^* = \frac{\sum_{l=0}^{h-1} \lambda^{l+1-h}(x_j((l+1)T) - x_j(lT))(x_i((l+1)T) - x_i(lT))}{\sum_{l=0}^{h-1} \lambda^{l+1-h}(x_i((l+1)T) - x_i(lT))^2}$$

[1] In MATLAB le somme sono state calcolate con l'ausilio di una matrice indicata con S.

Il nodo i grazie a queste informazioni computa $\eta_{ji} = \widehat{\frac{\alpha_i}{\alpha_j}} = \alpha^*$ e si ricava la stima della propria velocità, cioè

$$\hat{\alpha}_i \simeq \eta_{ji}\,\hat{\alpha}_j \qquad .$$

Per sfruttare la stessa tecnica utilizzata nel calcolo delle stime degli offset, conviene ora ricondursi alla scala logaritmica visto che il prodotto delle velocità attorno ad un ciclo deve essere uguale a 1. Bisogna inoltre tenere a mente che ci si trova nel caso in cui si considera che la comunicazione avviene simultaneamente con tutti i vicini, pertanto si ottiene:

$$\begin{aligned} \log\ \hat{\alpha}_i(hT) &\simeq \frac{1}{|N_i|}\sum_{j\in N_i}\log\left[\eta_{ji}(hT)\hat{\alpha}_j((h-1)T)\right] \\ &= \frac{1}{|N_i|}\sum_{j\in N_i}\left[\log\ \eta_{ji}(hT) + \log\ \hat{\alpha}_j((h-1)T)\right] \qquad . \end{aligned}$$

Si possono ora ricavare le stime dei tempi come:

$$\hat{\tau}_i(hT) = \frac{\tau_i(hT) - \hat{o}_i(hT)}{\hat{\alpha}_i(hT)} \qquad .$$

La variabile z utilizzata nel codice è

$$z(hT) = \frac{1}{N}\cdot \mathrm{var}(\hat{\tau}(hT))$$

cioè la varianza normalizzata delle stime dei tempi.

4.3.1 Caso sincrono e relative simulazioni

Vediamo in questa sezione l'andamento dell'algoritmo nel caso sincrono testato avendo imposto per tutte le simulazioni i seguenti parametri:

- $N = 10$
- $T = 1$
- $Iter = 10000$
- $r = 0.28$

Per quanto riguarda le condizioni iniziali di stima si è supposto, approssimativamente, che in partenza gli orologi avessero stessa velocità e offset nullo. Anche qui si è scelto di fissare uno tra gli N nodi prendendolo a riferimento come andamento ideale di legge temporale.
La seguente tabella riporta i valori relativi ad alcuni test scelti a titolo d'esempio tra tutti quelli prodotti in sede di simulazione, per ciascuno dei quali si è variato il valore del parametro λ in corrispondenza al variare del rumore e delle frequenze di oscillazione introdotte. Come si può osservare in questo paragrafo si è scelto di testare le prestazioni del protocollo in relazione all'introduzione di disturbi.

	lambda	*sigma_n*	*value_vel*
DISTURBI NULLI	1000	0	0
DISTURBI MEDI	5	10^{-7}	0
DISTURBI ELEVATI	1.1	10^{-6}	0
DISTURBI ELEVATI	1.1	0	10^{-7}

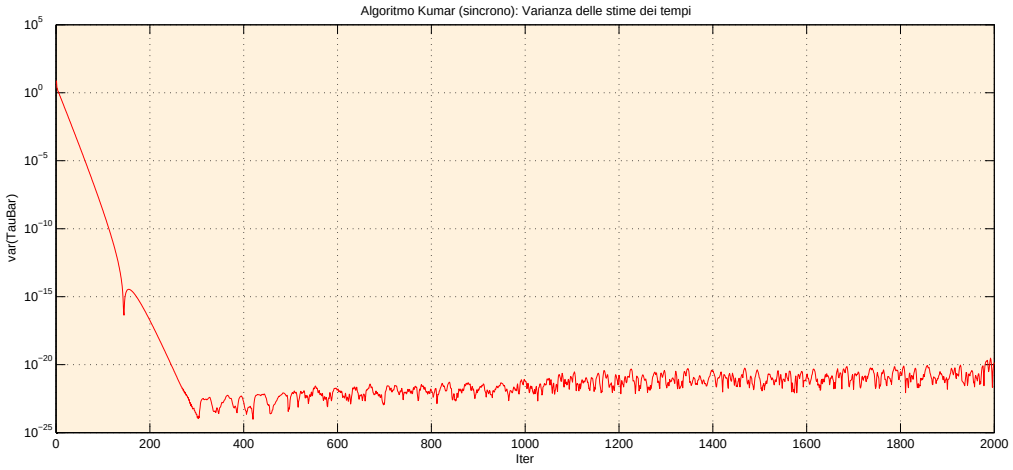


Figura 12: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso ideale, ossia assenza di rumore e velocità tempo-invarianti. Ritenendo quindi che avvenga una comunicazione perfetta il *lambda* è stato impostato ad un valore elevato (1000). Ciò ha garantito una velocità di convergenza molto buona con varianza "bounded".

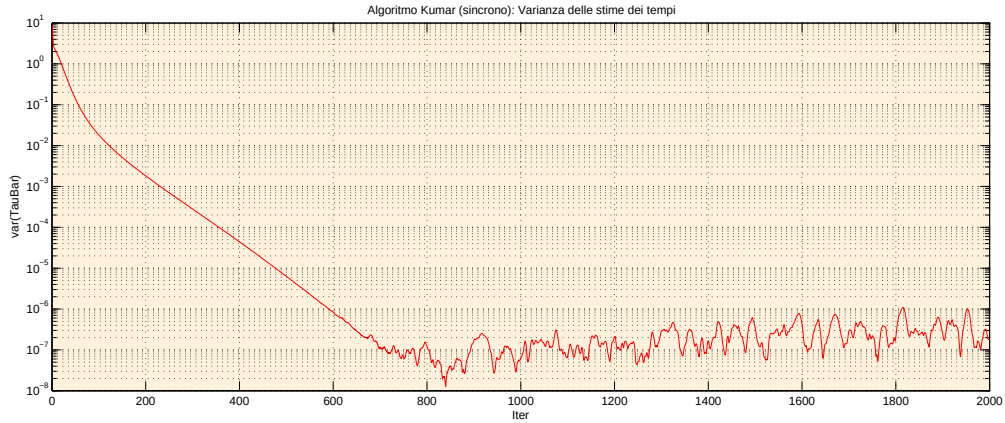


Figura 13: In figura è riportata la varianza dell'algoritmo in scala logaritmica in presenza di rumore ($sigma\_n = 10^{-7}$), con velocità tempo-invarianti. Il *lambda* è stato impostato ad un valore medio-basso, pari a 5. In questo caso si raggiunge comunque il consenso ma con una velocità di convergenza nettamente inferiore rispetto al caso precedente.

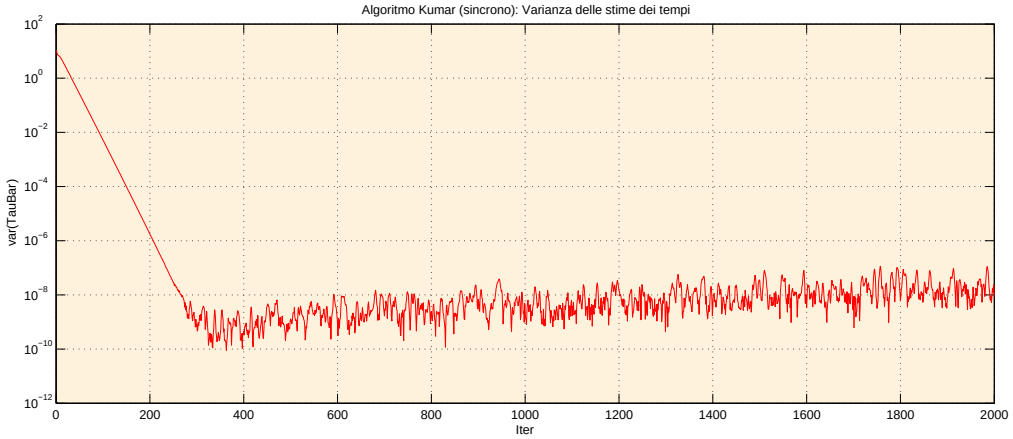


Figura 14: In figura è riportata la varianza dell'algoritmo in scala logaritmica in presenza di rumore ($sigma\_n = 10^{-6}$), con velocità tempo-invarianti. Il *lambda* è stato impostato ad un valore molto basso, pari a 1.1. La velocità di convergenza si vede essere nettamente superiore rispetto al caso in cui il *lambda* aveva valori superiori. Questo porta a pensare che la diminuzione del parametro comporti un aumento della velocità e una miglior gestione dei rumori.

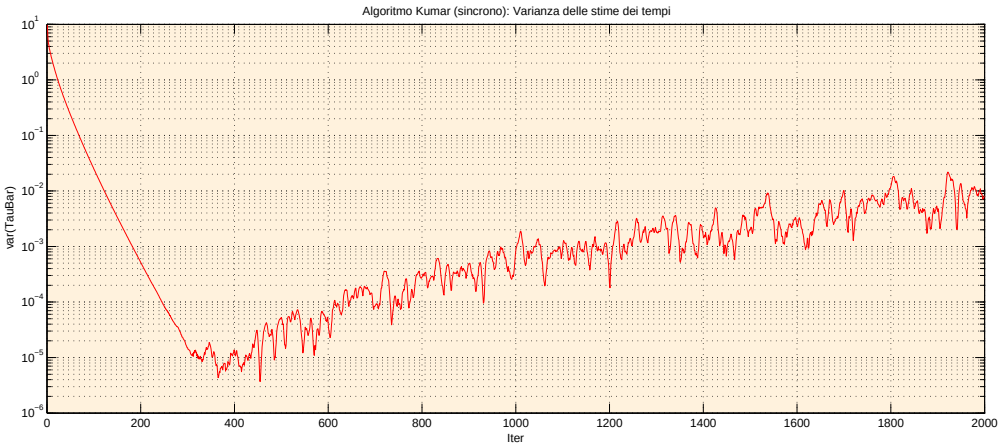


Figura 15: In figura è riportata la varianza dell'algoritmo in scala logaritmica in assenza di rumore ma avendo introdotto tempo-varianza nelle velocità ($value\_vel = 10^{-7}$). Il *lambda* anche qui è stato impostato al valore 1.1 a fronte di ciò che era stato osservato precedentemente. Tuttavia la convergenza non è garantita; come si vede già dalla 400-esima iterazione la varianza tende a divergere.

Dalla campagna di simulazioni effettuate si è osservato che al diminuire del parametro *lambda* ($\lambda \in (1, +\infty)$) aumenta la pendenza della retta che rappresenta la velocità dell'algoritmo al raggiungimento del consenso e migliora altresì la gestione del rumore. Purtroppo però questo protocollo non è in grado di gestire la tempo-varianza delle frequenze di oscillazione degli orologi ed i rumori oltre ad un certo valore limite.
In seguito a tutto ciò si è arrivati a considerare come ottimale il valore $\lambda = 1.1$, per il quale l'algoritmo di Kumar offre mediamente buone prestazioni per un'implementazione di tipo sincrono.

4.3.2 Caso asincrono e relative simulazioni

L'implementazione asincrona di questo protocollo che è stata realizzata purtroppo non ha offerto buone prestazioni sia nel caso broadcast che in quello gossip. Ciò crea perplessità sulla sensatezza dell'implementazione da noi prodotta, poiché nell'articolo [5] veniva dimostrata la convergenza proprio in una versione asincrona. Per tale ragione non si sono ritenuti attendibili i risultati prodotti dal codice sviluppato che viene comunque riportato per completezza in appendice.

4.4 Algoritmo PI

In questa sezione si presenterà il protocollo di sincronizzazione temporale PI [9]. Si tratta di un algoritmo in grado di compensare sia le differenze iniziali di offset che quelle delle velocità degli orologi interni. L'idea di partenza si basa su un controllore Proporzionale-Integrativo (PI) che modellizza le differenti velocità come disturbi costanti sconosciuti e i differenti offsets come diverse condizioni iniziali. Sebbene questo protocollo nasca per un'implementazione di tipo sincrono si è cercato in questo lavoro di testarne una versione asincrona. Vengono garantite sia la convergenza che l'ottimalità del sistema una volta fornito il grafo di comunicazione. Questo protocollo può essere usato senza problemi per studiare l'effetto del rumore e dei disturbi esterni nelle performance a regime.

MODELLO

Modellizziamo la rete di orologi con un grafo $\mathscr{G}(V, \mathscr{E})$ di N nodi, dove
$V = \{1, \ldots, N\}$ è l'insieme dei "vertici" ed $\mathscr{E} \subseteq V \times V$.
Consideriamo la comunicazione $j \longrightarrow i$ e limitiamoci a descrivere il caso sincrono in cui tutti i nodi comunicano nello stesso istante hT; chiamiamo con N_i l'insieme dei nodi che comunicano al nodo i il loro tempo locale: $N_i = \{j | (j, i) \in \mathscr{E}\}$.

Orologio locale

In questo algoritmo si hanno N orologi modellizzati come oscillatori di periodo Δ_i non noto.
Le leggi dei rispettivi orologi si scrivono come per gli altri algoritmi:

$$\begin{aligned} \tau_i &= \alpha_i\, t + o_i \\ \tau_j &= \alpha_j\, t + o_j \qquad . \end{aligned}$$

Ad ogni istante di comunicazione i invia ad ogni nodo vicino j la sua stima del tempo assoluto:

$$\hat{\tau}_i(hT) = x'_i(hT)$$

e al tempo stesso riceve da ognuno di essi la loro stima del tempo assoluto:

$$\hat{\tau}_j(hT) = x'_j(hT) \quad .$$

La legge oraria di ogni orologio locale all'istante t si può riscrivere come:

$$\tau_i(t) = \lfloor \frac{t - t_{0i}}{\Delta_i} \rfloor$$

da cui deriva che la stima della legge oraria è data da:

$$\hat{\tau}_i(t) = \Delta \cdot \tau_i(t) + \hat{\tau}_i(t_{0i})$$

dove $\hat{\tau}_i(t_{0i})$ è l'offset iniziale pari a una stima dell'istante iniziale di tempo t_{0i}.

Vengono introdotti due stati: x_i' che rappresenta la stima del tempo e x_i'' con $\Delta x_i''$ che rappresenta invece la stima del periodo di oscillazione.
Si applica poi un controllo del tipo

$$u_i(hT) = -\sum_{j=1}^{N} K_{ij} F x_j(hT)$$

con F matrice 2×2 e K costante al variare di h, simmetrica e tale che $K\mathbf{1} = 0$.
Si può quindi tradurre il problema con il seguente modello di stato, dove l'uscita del sistema $y_i(hT)$ rappresenta la stima della legge oraria del nodo i:

$$\begin{aligned} x_i((h+1)T) &= \begin{bmatrix} 1 & \Delta\alpha_i(hT) \\ 0 & 1 \end{bmatrix} [x_i((hT) - u_i(hT)] \\ y_i(hT) &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_i(hT) \\ x_i(0) &= \begin{bmatrix} y_i(0) \\ 1 \end{bmatrix} \end{aligned}$$

dove $\alpha_i(hT) = \frac{T}{\Delta_i}$ è la velocità dell'i-esimo orologio.
Riportandolo in forma vettoriale si ottiene:

$$\underbrace{\begin{bmatrix} x'((h+1)T) \\ x''((h+1)T) \end{bmatrix}}_{\substack{x((h+1)T) \\ 2N \times 1}} = \begin{bmatrix} I_N & TD \\ 0 & I_N \end{bmatrix} \left(\begin{bmatrix} I_N & 0 \\ 0 & I_N \end{bmatrix} - \begin{bmatrix} f_{11}K & f_{12}K \\ f_{21}K & f_{22}K \end{bmatrix} \right) \begin{bmatrix} x'(hT) \\ x''(hT) \end{bmatrix}$$

dove $D = diag\{\Delta/\Delta_1, \ldots, \Delta/\Delta_N\} = diag\{d_1, \ldots, d_N\}$ è una matrice $N \times N$.
L'obiettivo ora è di trovare F e K tali che si abbia errore di sincronizzazione

$$e(h) = \begin{bmatrix} \Omega & 0 \end{bmatrix} x(hT) \longrightarrow 0 \quad \text{per } h \to \infty$$

con

$$\Omega = I - \frac{1}{N}\mathbf{1}\mathbf{1}^* \quad \text{e} \quad \lim_{h\to\infty} [x'(h) - (ah+b)\mathbf{1}] = 0 \quad .$$

Dobbiamo cioè determinare F e K tali che:

- il sistema abbia un autovalore in 1 con *molteplicità algebrica* 2 e *molteplicità geometrica* 1. Questo assicura che le traiettorie contengano i modi del tipo $ah + b$;
- i due modi associati all'autovalore 1 siano non osservabili rispetto all'errore e;
- tutti gli altri autovalori siano interni al cerchio aperto di raggio unitario.

In conclusione il modello di controllo che verrà implementato sarà del tipo:

$$u_i(hT) = \begin{bmatrix} u_i'(hT) \\ u_i''(hT) \end{bmatrix} = \begin{bmatrix} f_{11} \\ f_{21} \end{bmatrix} \sum_{j \in N_i} K_{ij}(x_j'(hT) - x_i'(hT)) \tag{3}$$

dove si sono posti $f_{12} = f_{22} = 0$ e f_{11}, f_{21} tali che:

$$\begin{aligned} f_{11} > 0 \quad , \quad f_{21} > 0 \\ 0 < \lambda_i < \frac{4}{2f_{11} + Tf_{21}} \end{aligned}$$

Aggiornamento

Gli aggiornamenti degli stati sopra introdotti sono dati da:

$$\begin{aligned} x_i'((h+1)T) &= x_i'(hT) + u_i'(hT) \\ x_i''((h+1)T) &= x_i''(hT) + u_i''(hT) \end{aligned}$$

Notando che per $hT \leq t < (h+1)T$ si ha

$$x_i'(t) = x_i'(hT^+) + \alpha_i(t - hT)x_i''(hT^+)$$

la legge di aggiornamento degli stati risulta essere la seguente:

$$\begin{aligned} x_i''((h+1)T) &= x_i''(hT) + u_i''(hT) \\ x_i'((h+1)T) &= x_i'(hT) + \alpha_i T[x_i''((h+1)T)] + u_i'(hT) \end{aligned}$$

dove

$$\begin{aligned} u_i'(hT) &= h(x_j'(hT), \forall j \in N_i) \\ u_i''(hT) &= g(x_j'(hT), \forall j \in N_i) \end{aligned}$$

che giustificano (3).

4.4.1 Caso sincrono e relative simulazioni

Vediamo in questa sezione l'andamento dell'algoritmo nel caso sincrono testato avendo imposto per tutte le simulazioni i seguenti parametri:

- $N = 10$
- $T = 1$
- $Iter = 10000$
- $r = 0.28$

Per quanto riguarda le condizioni iniziali di stima si è supposto, approssimativamente, che in partenza gli orologi avessero stessa velocità e offset nullo.
La seguente tabella riporta i valori relativi ad alcuni test scelti a titolo d'esempio tra tutti quelli prodotti in sede di simulazione, per ciascuno dei quali si è variato il valore dei parametri f_{11}, f_{21} in corrispondenza al variare del rumore e delle frequenze di oscillazione introdotte. Come si può osservare in questo paragrafo si è scelto di testare le prestazioni del protocollo in relazione all'introduzione di disturbi.

	$f11$, $f21$	$sigma\_n$	$value\_vel$
DISTURBI NULLI	0.1	0	0
DISTURBI ELEVATI	0.5	10^{-4}	10^{-5}
DISTURBI ELEVATI	0.9	10^{-4}	10^{-5}

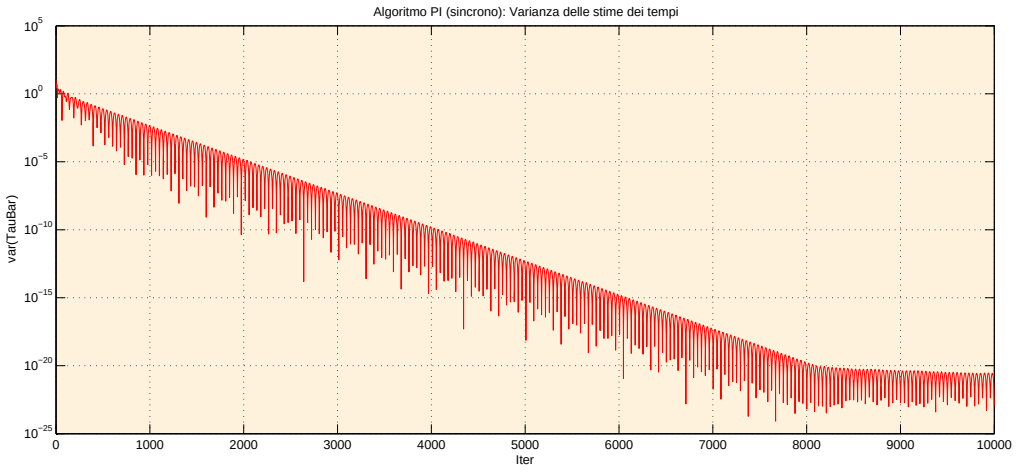


Figura 16: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso ideale, ossia assenza di rumore e velocità tempo-invarianti. Ritenendo quindi che avvenga una comunicazione perfetta si sono imposti i parametri ad un valore pari a $f11 = f21 = 0.1$. Ciò ha garantito una convergenza al consenso con una varianza "bounded".

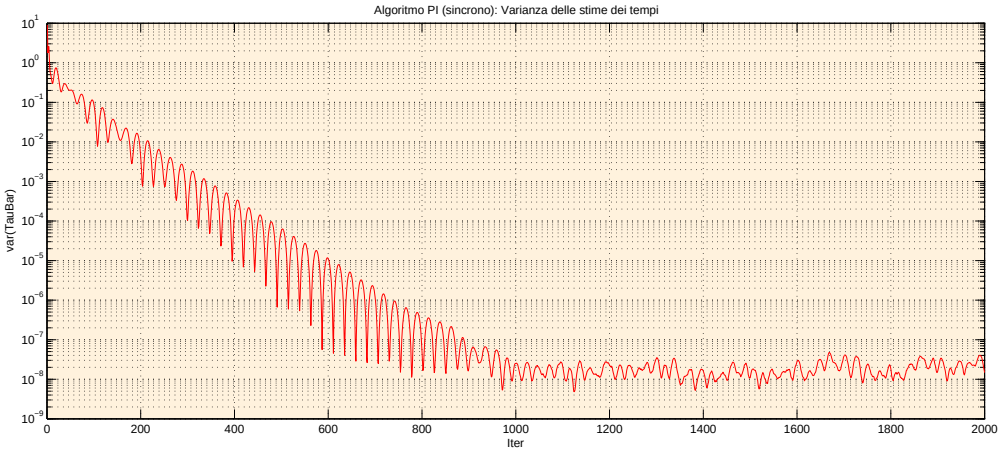


Figura 17: In figura è riportata la varianza dell'algoritmo in scala logaritmica in presenza di rumore ($sigma\_n = 10^{-4}$) e con l'introduzione di tempo-varianza nelle velocità ($value\_vel = 10^{-5}$). I parametri sono stati imposti ad un valore "medio" pari a $f11 = f21 = 0.5$. La velocità di convergenza al consenso si osserva essere superiore rispetto al caso precedente a scapito però di una perdita di prestazioni in termini di varianza. Questo porta a pensare che un piccolo aumento dei parametri comporti il raggiungimento di una maggiore velocità e di una buona gestione dei rumori.

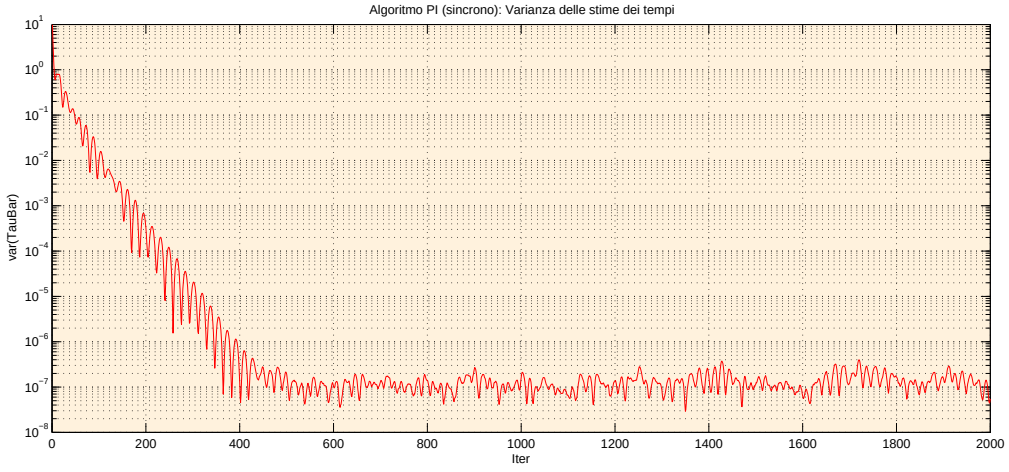


Figura 18: In figura è riportata la varianza dell'algoritmo in scala logaritmica in presenza di rumore ($sigma\_n = 10^{-4}$), e con l'introduzione di tempo-varianza nelle velocità ($value\_vel = 10^{-5}$). I parametri sono stati imposti ad un valore "alto" pari a $f11 = f21 = 0.9$. Rispetto al caso appena discusso, la velocità di convergenza è superiore ma l'aver aumentato i parametri ha comportato un aumento della varianza.

Dalla campagna di simulazioni effettuate si è osservato che all'aumentare dei parametri f_{11} e f_{21} aumenta la pendenza della retta che rappresenta la velocità dell'algoritmo al raggiungimento del consenso ma peggiora la gestione dei disturbi. Tuttavia questo protocollo è in grado di gestire senza problemi sia la presenza di rumore che quella di tempo-varianza delle frequenze. In seguito a tutto ciò si è arrivati a considerare come ottimali i valori $f_{11} = f_{21} = 0.5$, per i quali l'algoritmo di PI offre mediamente ottime prestazioni per un'implementazione di tipo sincrono.

4.4.2 Caso asincrono e relative simulazioni

Vediamo in questa sezione l'andamento dell'algoritmo nel caso asincrono testato avendo imposto per tutte le simulazioni i seguenti parametri:

- $N = 10$
- $T = 1$
- $Iter = 10000$
- $r = 0.28$

Per quanto riguarda le condizioni iniziali di stima si è supposto, approssimativamente, che in partenza gli orologi avessero stessa velocità e offset nullo.
La seguente tabella riporta i valori relativi ad alcuni test scelti a titolo d'esempio tra tutti quelli prodotti in sede di simulazione, per ciascuno dei quali si è variato il valore dei parametri f_{11}, f_{21} in corrispondenza al variare del rumore e delle frequenze di oscillazione introdotte. Come si può osservare in questo paragrafo si è scelto di testare le prestazioni del protocollo in relazione all'introduzione di disturbi. Rimarchiamo il fatto che i grafici presentati riportano solo il transitorio delle simulazioni, ove la durata totale era, come sopra dichiarato, di 10000 iterazioni.

	$f11$, $f21$	$sigma\_n$	$value\_vel$
DISTURBI NULLI	0.1	0	0
DISTURBI ELEVATI	0.5	10^{-4}	10^{-5}
DISTURBI ELEVATI	0.9	10^{-4}	10^{-5}

Effettuando un confronto tra le due implementazioni asincrone si evidenzia una leggermente miglior prestazione nel caso di comunicazione broadcast; infatti la varianza in questo caso resta sempre prossima allo zero o comunque a valori inferiori di quelli generati nel caso gossip. Ciò è intuibile considerando il funzionamento dei due protocolli di comunicazione.

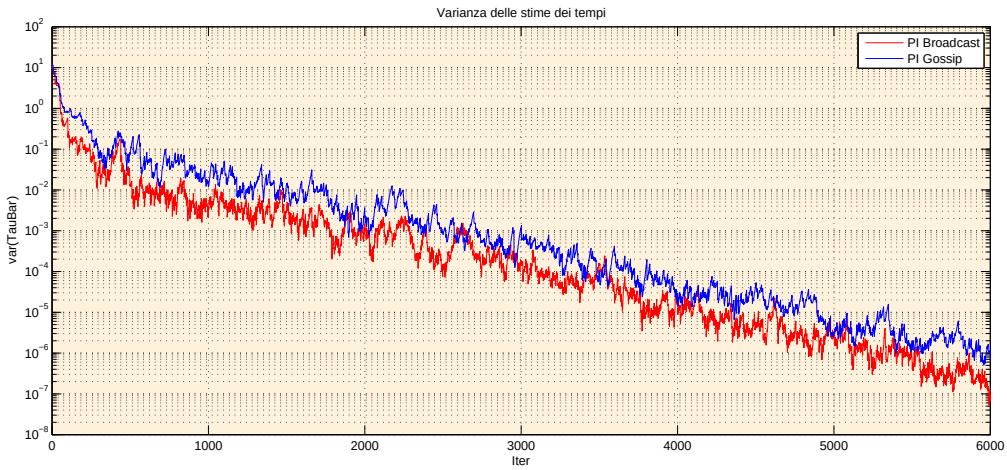


Figura 19: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso ideale, ossia assenza di rumore e velocità tempo-invarianti, ove si è imposto $f11 = f21 = 0.1$. In entrambi i casi di comunicazioni si giunge al consenso con una varianza limitata.

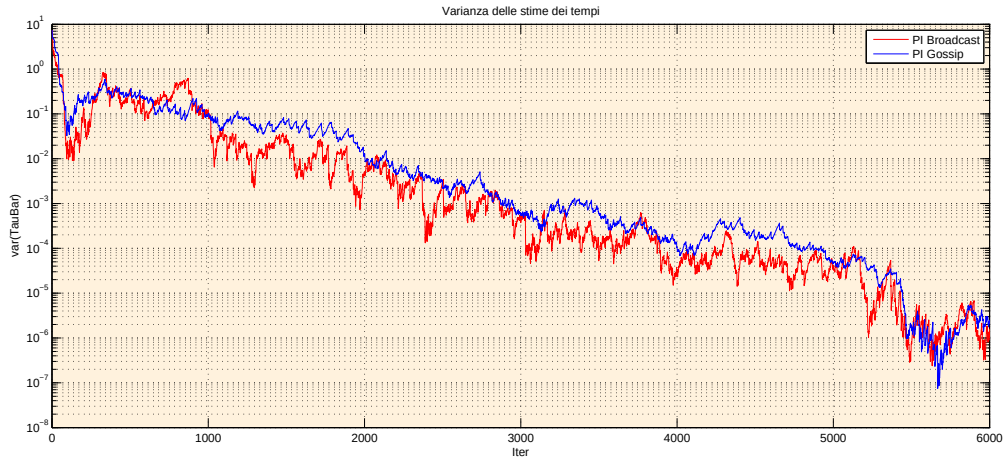


Figura 20: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso di $f11 = f21 = 0.5$, $sigma\_n = 10^{-4}$ e $value\_vel = 10^{-5}$. Si può vedere che anche qui il consenso viene raggiunto più o meno con la stessa velocità del caso precedente. Di nuovo si osserva come l'algoritmo, una volta aumentati i parametri, sia in grado di gestire bene la presenza di disturbi.

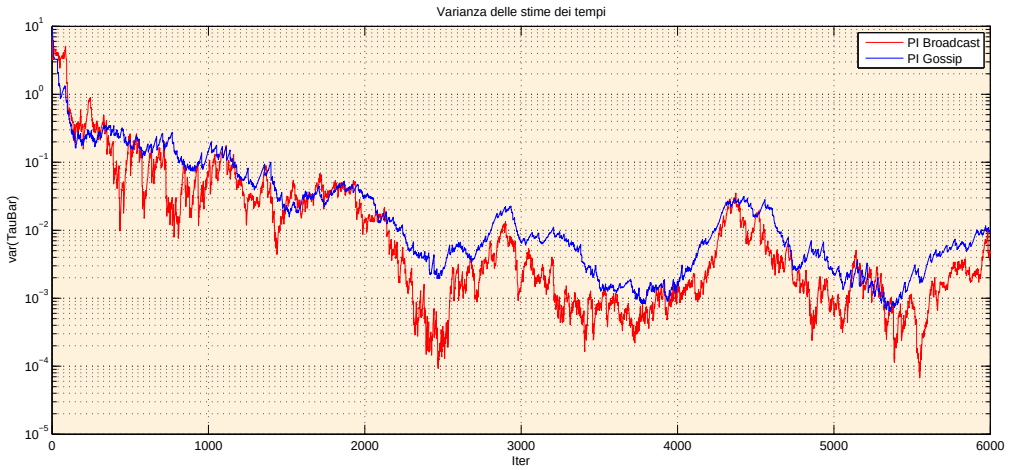


Figura 21: In figura è riportata la varianza dell'algoritmo in scala logaritmica nel caso di $f11 = f21 = 0.9$, $sigma\_n = 10^{-4}$ e $value\_vel = 10^{-5}$. Si può vedere che anche qui il consenso viene raggiunto più o meno con la stessa velocità dei casi precedenti. Tuttavia si vede come l'aumento dei parametri abbia portato ad un aumento della varianza delle stime, come già dedotto nel caso sincrono.

Come si vede in tutti grafici le varianze sono "*bounded*". Anche qui, come per il caso sincrono, si è giunti a considerare come ottimale il valore 0.5 per i parametri f_{11} e f_{21}. Esso infatti fornisce, a livello empirico, un buon compromesso tra velocità di convergenza al consenso e gestione di disturbi in termini di varianza delle stime. Nonostante questo algoritmo fosse stato inizialmente pensato per una implementazioni sincrona, le sue prestazioni nella versione asincrona offrono, come si è visto, buone garanzie in termini di convergenza.

5 Confronti e conclusioni

5.1 Confronti

In questo capitolo presenteremo un confronto, sia per il caso sincrono che per il caso asincrono, tra gli algoritmi che sono stati esposti nei capitoli precedenti, con l'unica eccezione per il caso asincrono del protocollo di Kumar.
Si è deciso di partire da condizioni sui parametri caratteristici per le quali ciascun algoritmo presentasse l'andamento mediamente ottimale che è stato ottenuto nelle simulazioni viste nel capitolo 4.
Si tratta di $rho = 0.5$ per l'algoritmo ATS, $lambda = 1.1$ per l'algoritmo di Kumar (sincrono) e $f11 = f21 = 0.5$ per l'algoritmo PI. Ciascun confronto si è svolto quindi valutando il comportamento delle rispettive varianze a parità di condizioni. Anche qui infatti si è utilizzato un programma *Main* per effettuare le simulazioni [2], garantendo che, ad ogni test si avesse lo stesso grafo, grado di disturbi, stesse condizioni iniziali, numero di iterazioni ...
Analizziamo per primo il caso sincrono per il quale sono state effettuate le simulazioni corrispondenti ai valori dei parametri che vengono riportati nella seguente tabella:

Simulazione	$sigma\_n$	$value\_vel$
1	0	0
2	10^{-4}	0
3	10^{-3}	0
4	0	10^{-9}
5	0	10^{-5}
6	10^{-3}	10^{-9}
7	10^{-4}	10^{-5}

La simulazione 1 mostra gli andamenti delle varianze nel caso ideale cioè con **disturbi nulli**, le simulazioni 2, 3, 4, 5 mostrano gli andamenti delle varianze in presenza di **disturbi medi** e le simulazioni 6 e 7, invece, mostrano gli andamenti delle varianze in presenza di **disturbi elevati**.

[2] Tutte mediate, come per le simulazioni precedenti, mediante strategia Montecarlo.

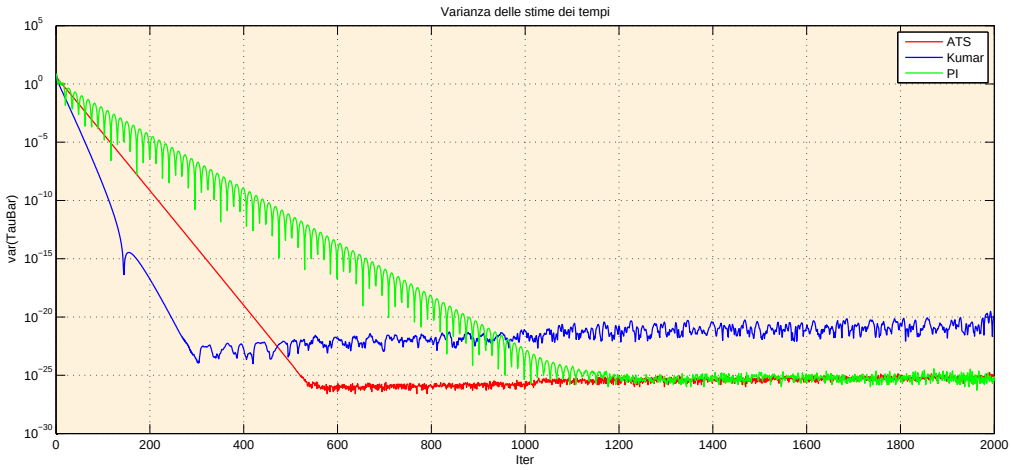


Figura 22: In figura sono riportate le varianze nel caso ideale in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di disturbo a: $sigma\_n = 0$, $value\_vel = 0$. Dalla figura si osserva come l'algoritmo di Kumar abbia velocità maggiore degli altri due, ma si assesti ad un valore di varianza "bounded" superiore, prossimo a 10^{-20}; quello che è evidente è che, in questo caso, l'ATS offra le migliori prestazioni a livello di compromesso velocità-varianza.

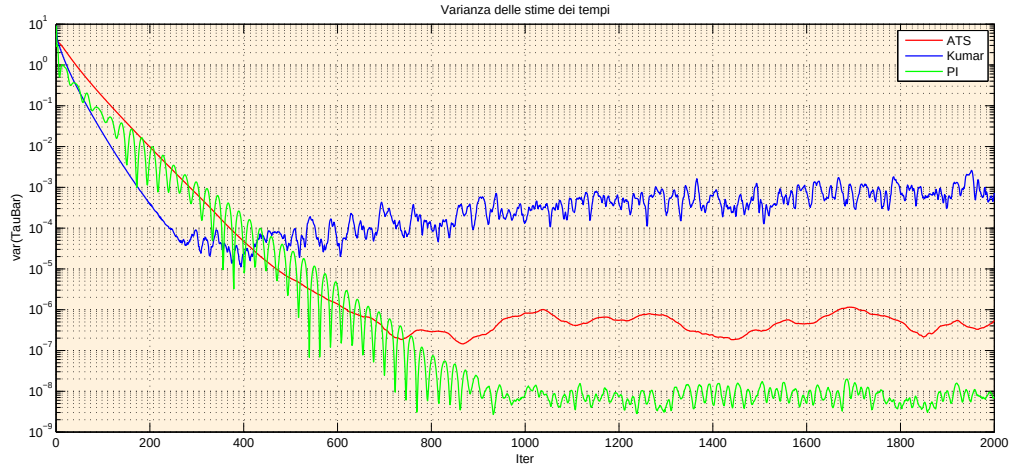


Figura 23: In figura sono riportate le varianze nel caso in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di rumore a: $sigma\_n = 10^{-4}$, $value\_vel = 0$. Dalla figura si osserva come sia l'ATS che il PI riescano a gestire meglio la presenza di rumore rispetto all'algoritmo di Kumar che offre comunque una varianza "bounded". Al solito si nota come Kumar e ATS siano mediamente più veloci del PI, il quale però presenta varianza minore di tutti.

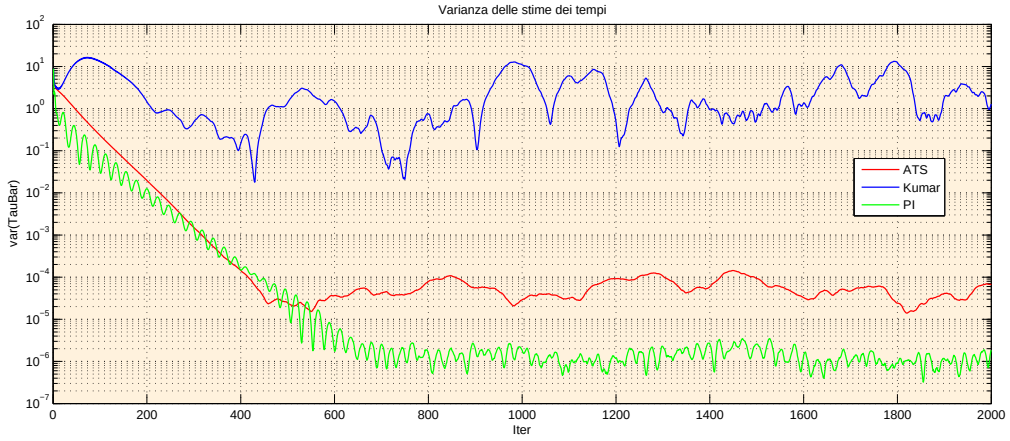


Figura 24: In figura sono riportate le varianze nel caso in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di rumore a zero: $sigma\_n = 10^{-3}$, $value\_vel = 0$. Dalla figura si osserva come sia l'ATS che il PI riescano a gestire anche la presenza di un rumore più elevato a differenza del Kumar che mai giunge al consenso.

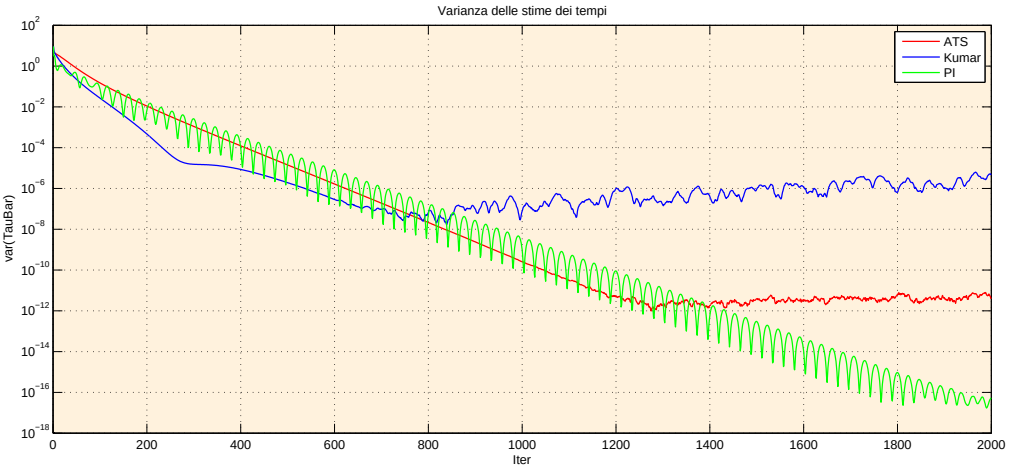


Figura 25: In figura sono riportate le varianze nel caso in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di rumore a zero: $sigma\_n = 0$, $value\_vel = 10^{-9}$. Dalla figura si osserva come sia l'ATS che il PI riescano a gestire anche la presenza della tempo-varianza delle velocità a differenza del Kumar che non è in grado di compensarla (come del resto era stato evidenziato nel capitolo 4).

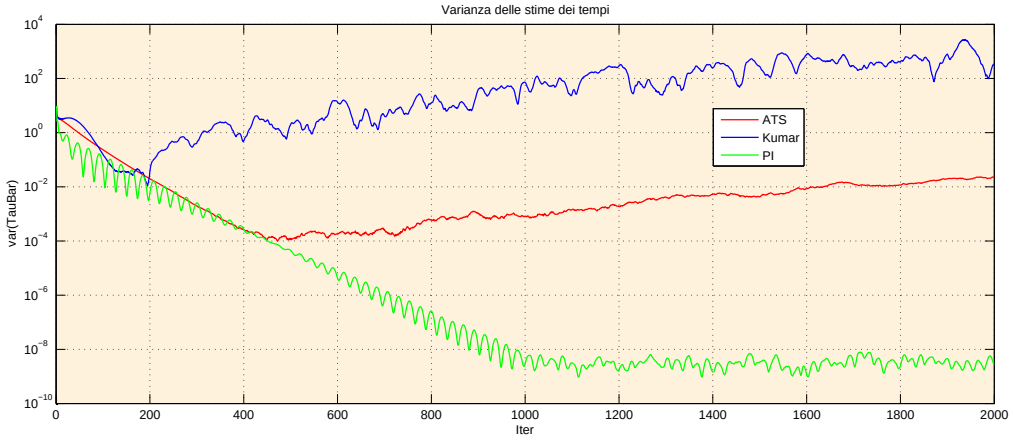


Figura 26: In figura sono riportate le varianze nel caso in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di rumore a zero: $sigma\_n = 0$, $value\_vel = 10^{-5}$. Dalla figura si osserva come solo il PI sia in grado di gestire un considerevole aumento di disturbo in termini di tempo-varianza delle velocità. Sia l'ATS che il Kumar non riescono in questo caso a compensarla.

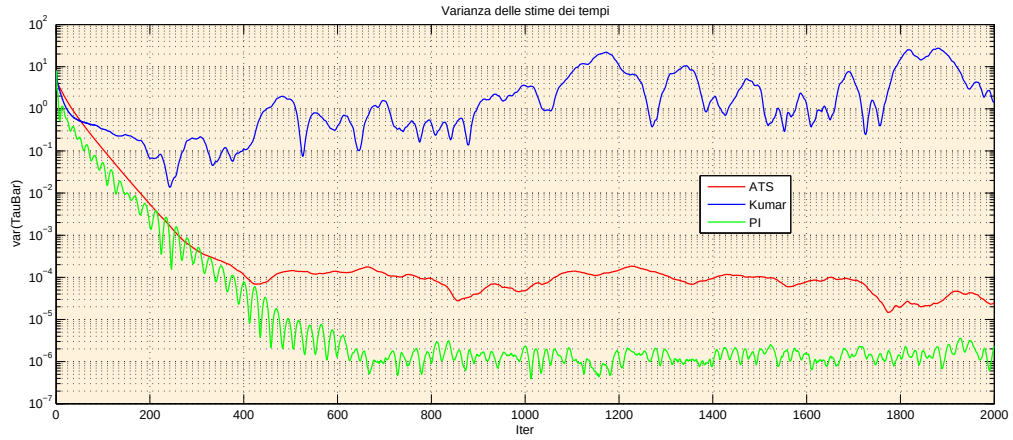


Figura 27: In figura sono riportate le varianze nel caso in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di rumore a zero: $sigma\_n = 10^{-3}$, $value\_vel = 10^{-9}$. Dalla figura si osserva come sia l'ATS che il PI riescano a gestire la simultanea presenza di tempo-varianza delle velocità e rumore. Come ci si aspettava il Kumar, non compensando la tempo-varianza, diverge all'introduzione di questi valori dei disturbi.

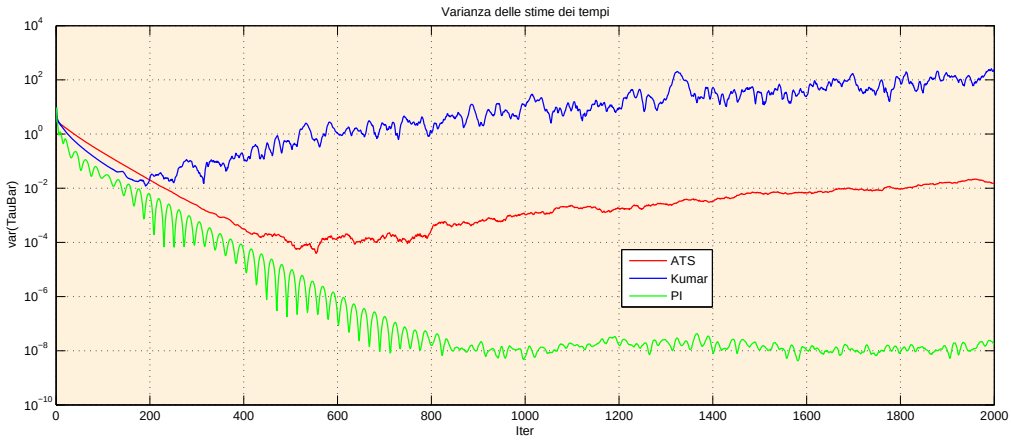


Figura 28: In figura sono riportate le varianze nel caso in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di rumore a zero: $sigma\_n = 10^{-4}$, $value\_vel = 10^{-5}$. Dalla figura si osserva come solo il PI sia in grado di gestire un considerevole aumento di disturbo sia in termini di tempo-varianza delle velocità che di rumore. Esso come in tutti i grafici presenta ottime prestazioni dal punto di vista di un compromesso tra la gestione dei disturbi e la velocità di convergenza.

Vediamo adesso il caso asincrono per il quale sono state effettuate le simulazioni corrispondenti ai valori dei parametri che vengono riportati nella seguente tabella:

$sigma\_n$	$value\_vel$
0	0
10^{-4}	0
10^{-3}	0
0	10^{-9}
0	10^{-5}
10^{-3}	10^{-9}
10^{-4}	10^{-5}

La simulazione 1 mostra gli andamenti delle varianze nel caso ideale cioè con **disturbi nulli**, le simulazioni 2, 3, 4, 5 mostrano gli andamenti delle varianze in presenza di **disturbi medi** e le simulazioni 6 e 7, invece, mostrano gli andamenti delle varianze in presenza di **disturbi elevati**.

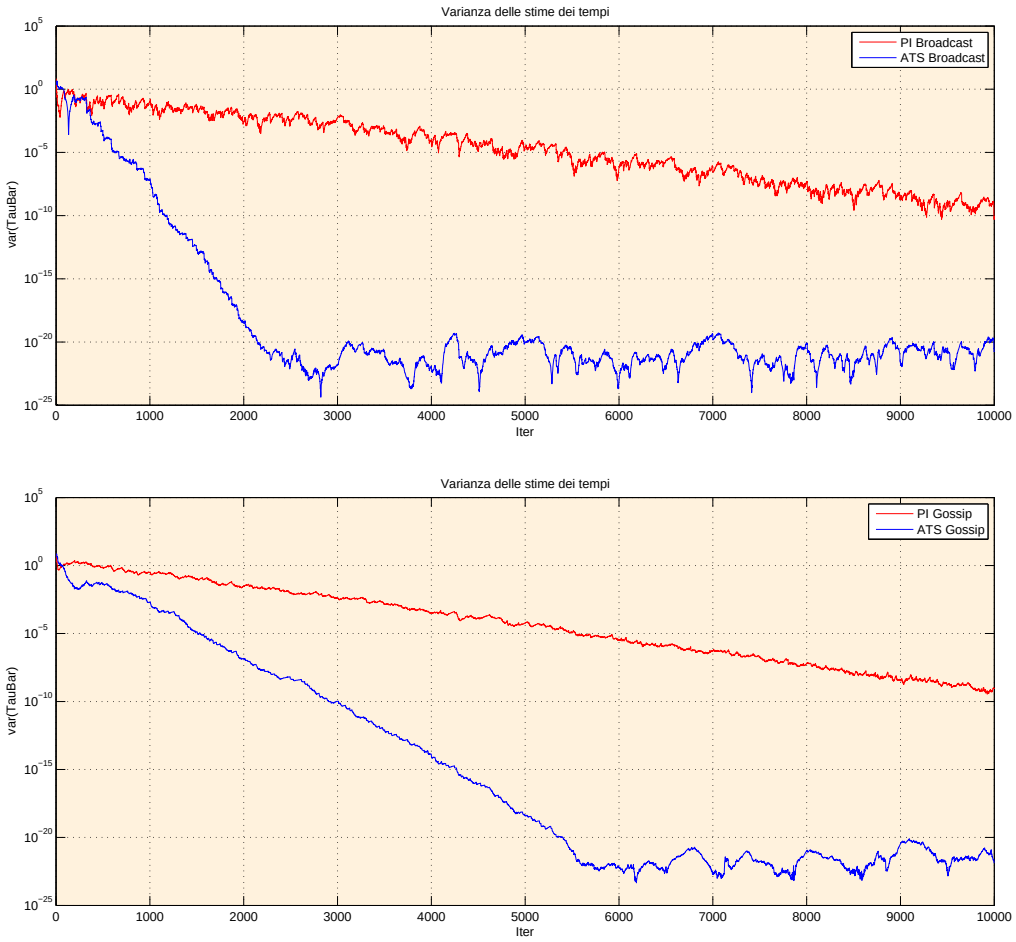


Figura 29: Nelle figure sono riportate le varianze nel caso Broadcast e Gossip in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di rumore a zero: $sigma\_n = 0$, $value\_vel = 0$. Si osserva come in entrambi i casi l'ATS converga ad una velocità superiore rispetto al PI.

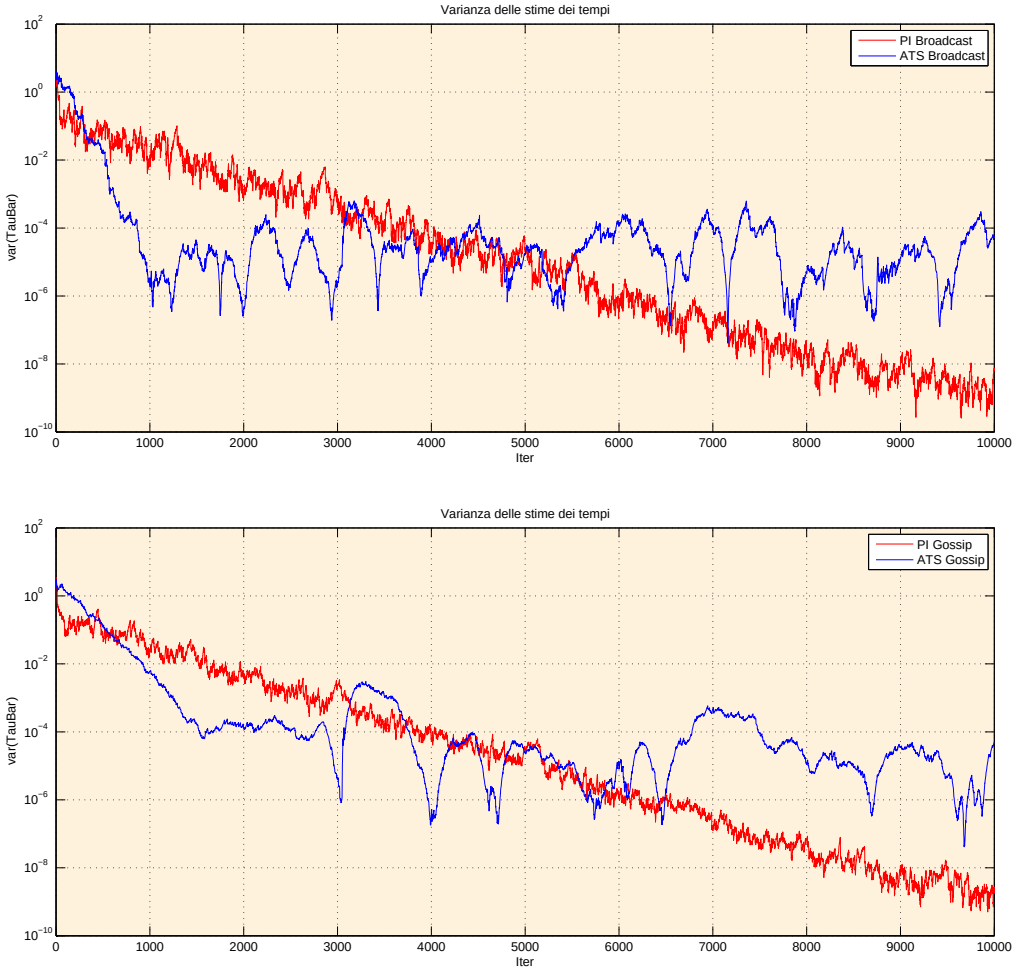


Figura 30: Nelle figure sono riportate le varianze nel caso Broadcast e Gossip in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di disturbo: $sigma\_n = 10^{-4}$, $value\_vel = 0$. Si osserva come in entrambi i casi l'ATS converga ad una velocità superiore rispetto al PI, ma si assesti a valori della varianza superiori.

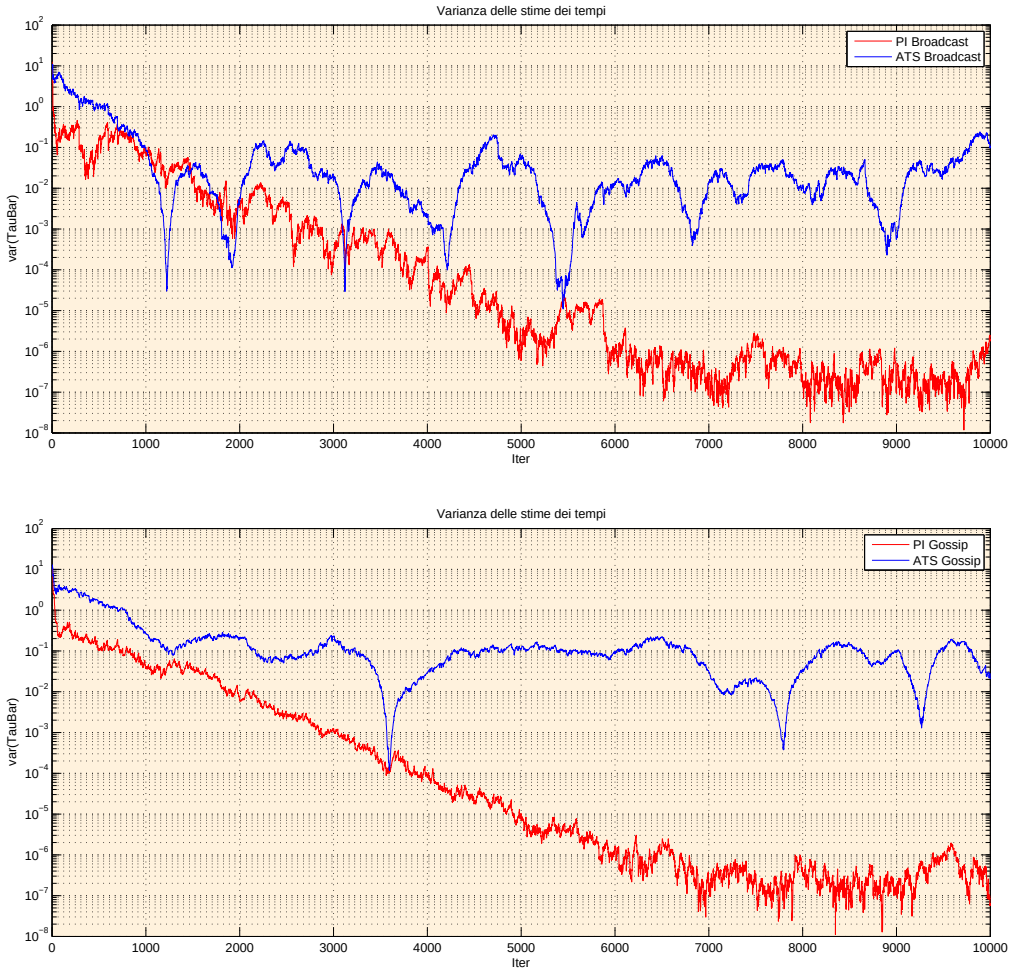


Figura 31: Nelle figure sono riportate le varianze nel caso Broadcast e Gossip in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di disturbo: $sigma\_n = 10^{-3}$, $value\_vel = 0$. Si osserva che, nonostante le prestazioni dell'ATS siano peggiorate anche in termini di velocità, esso offre comunque una varianza "bounded". Il PI invece gestisce senza grossi rallentamenti la presenza di un consistente rumore.

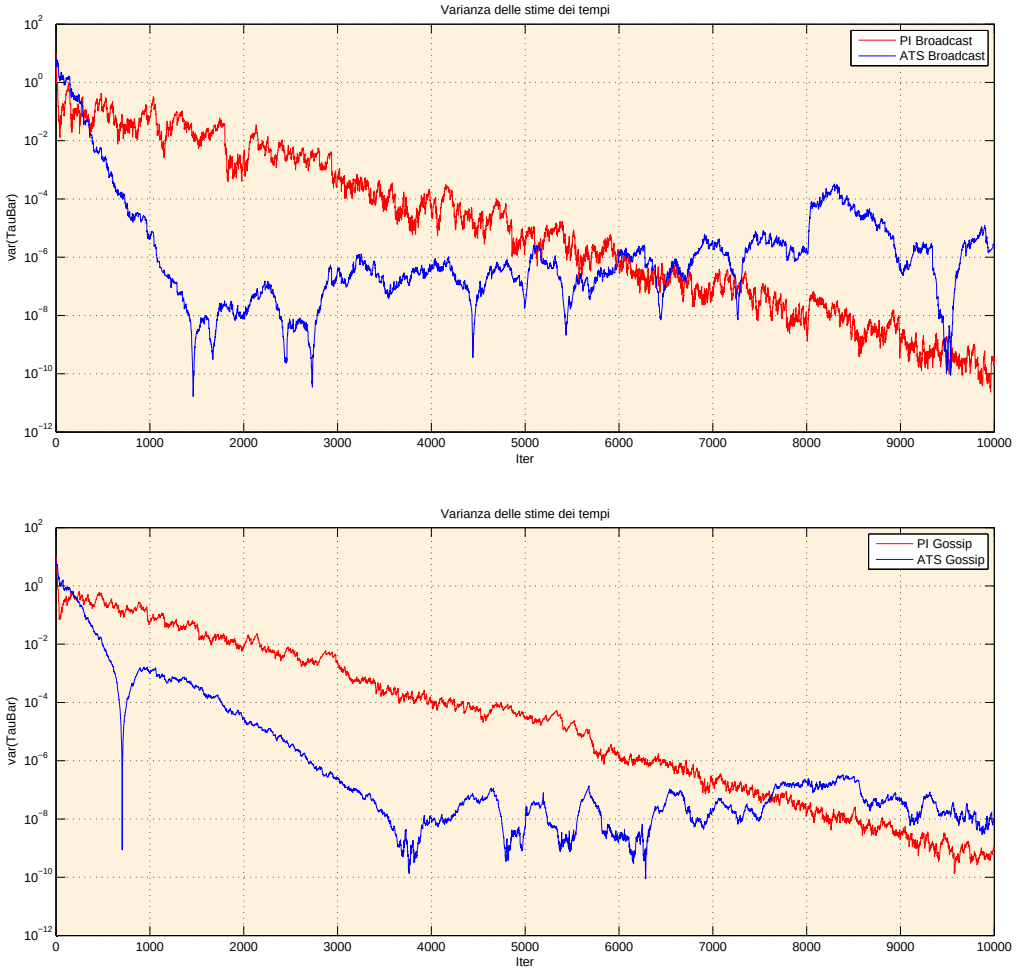


Figura 32: Nelle figure sono riportate le varianze nel caso Broadcast e Gossip in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di disturbo: $sigma\_n = 0$, $value\_vel = 10^{-9}$. Si osserva come in entrambi i casi l'ATS converga ad una velocità superiore rispetto al PI, ma si assesti a valori della varianza superiori. Questa comportamento è analogo alla risposta vista in figura (30), dove però veniva introdotto solo un considerevole rumore.

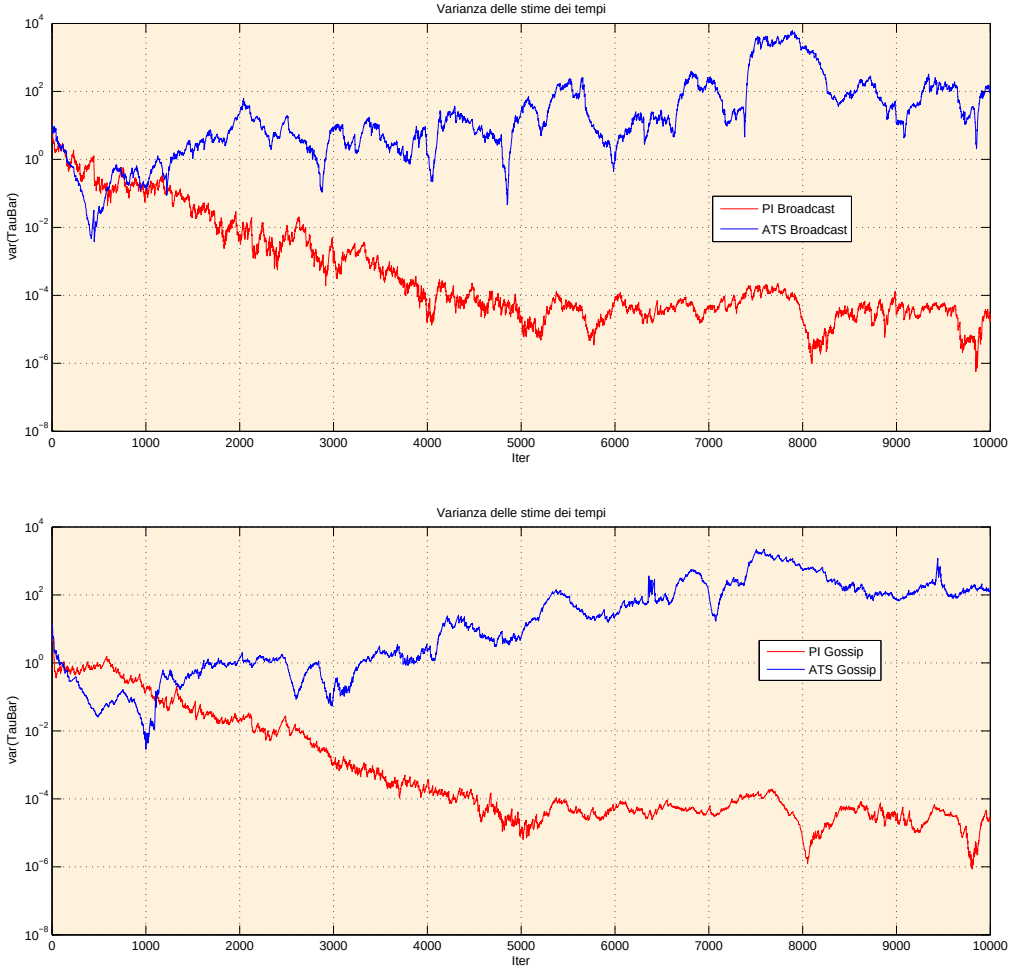


Figura 33: Nelle figure sono riportate le varianze nel caso Broadcast e Gossip in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di disturbo: $sigma\_n = 0$, $value\_vel = 10^{-5}$. Si osserva un grave deterioramento delle prestazioni dell'ATS a fronte di buone prestazioni del PI alla presenza di una forte tempo-varianza delle frequenze (come del resto era stato evidenziato nel capitolo 4).

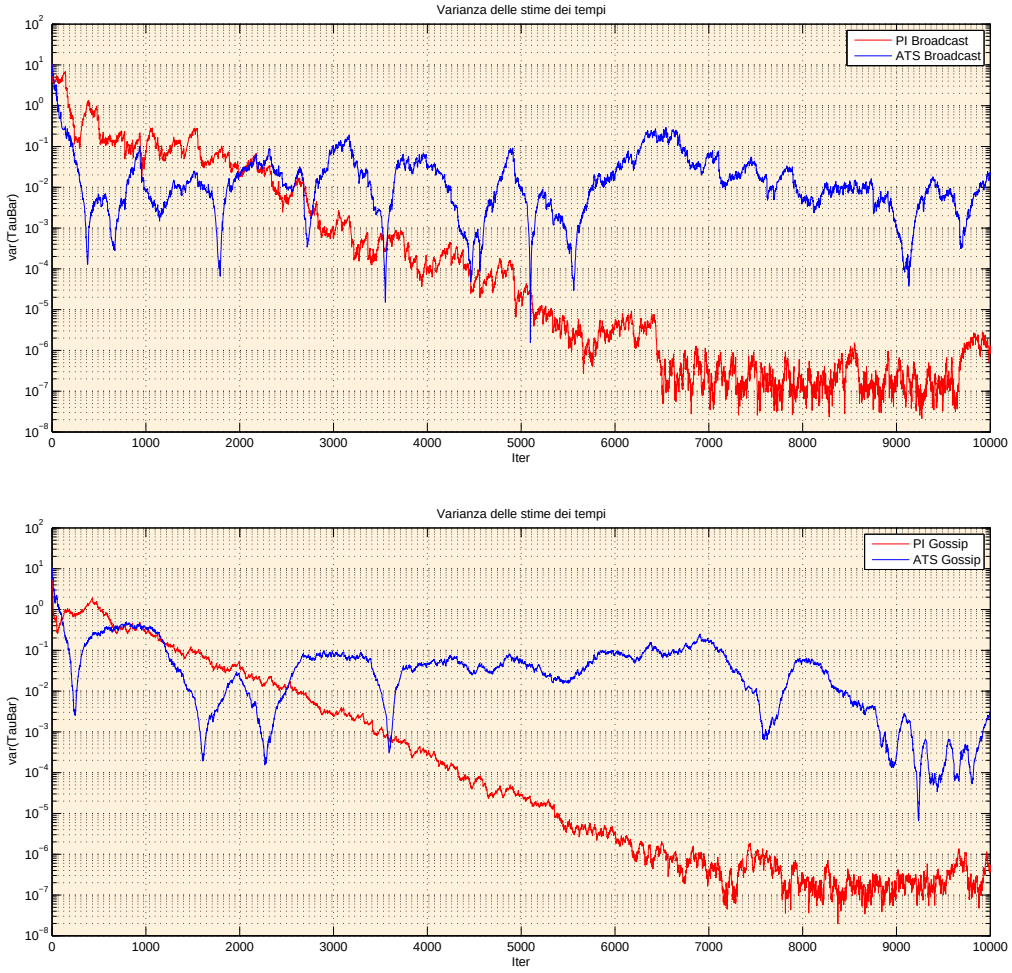


Figura 34: Nelle figure sono riportate le varianze nel caso Broadcast e Gossip in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di disturbo: $sigma\_n = 10^{-3}$, $value\_vel = 10^{-9}$. Si osserva che la presenza di piccoli disturbi viene gestita da entrambi gli algoritmi. Tuttavia il PI si dimostra più performante in termini di varianza delle stime.

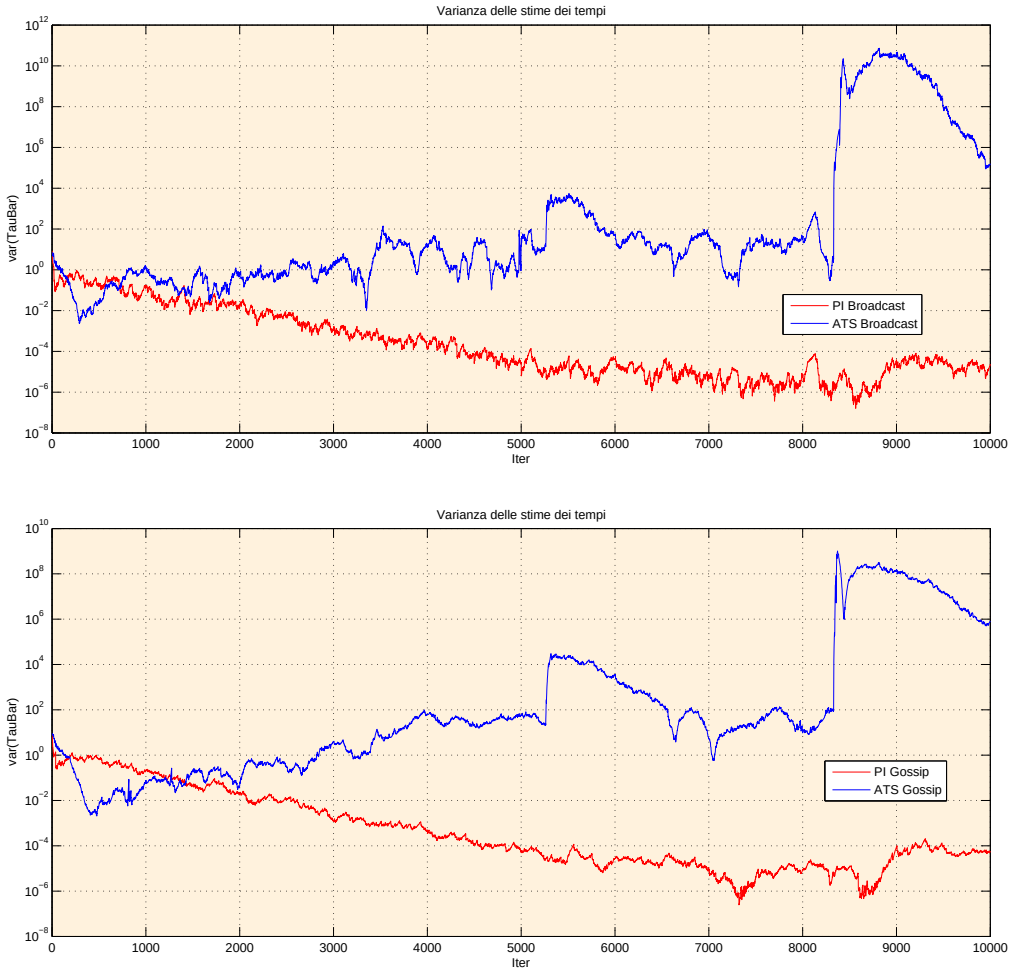


Figura 35: Nelle figure sono riportate le varianze nel caso Broadcast e Gossip in cui i parametri caratteristici degli algoritmi sono settati ai valori ottimali e i parametri di disturbo: $sigma\_n = 10^{-4}$, $value\_vel = 10^{-5}$. Si osserva che la presenza di elevati disturbi viene gestita unicamente dall'algoritmo PI che anche questa volta offre una discreta velocità di convergenza al consenso e una varianza "bounded" prossima a 10^{-4}.

5.2 Conclusioni

In questo lavoro è stato quindi svolto un confronto tra tre possibili approcci totalmente distribuiti che risolvono il problema della sincronizzazione temporale. Ciascuno di questi ha evidenziato pregi e difetti relativi alle diverse strategie di risoluzione del problema. Gli algoritmi di controllo proposti hanno mostrato che, sotto opportune condizioni, è garantito il raggiungimento della sincronizzazione in assenza di disturbi. L'obbiettivo del lavoro consisteva nell'analisi di questi stessi algoritmi applicati a grandezze stimate anziché a grandezze esatte per tenere conto anche di una eventuale presenza di disturbi di processo e di errori di misura. I risultati sperimentali, illustrati nel capitolo 4, mostrano che tutti e tre gli algoritmi considerati, sotto opportune condizioni, portano al raggiungimento della sincronizzazione, anche in presenza di adeguati parametri di disturbo. Nella maggior parte degli esperimenti l'algoritmo PI sembra essere il più efficace in termini di robustezza. Questo può essere dovuto sia alla filosofia di tipo lineare con cui esso nasce, che permette, in linea di principio, di non essere gravemente influenzato dalla presenza di disturbi sia dal basso numero di informazioni che ciascun nodo deve scambiarsi. Il medesimo algoritmo sembra soffrire di più in termini di velocità di convergenza nel caso ideale rispetto agli altri due di tipo non lineare. L'algoritmo di Kumar risulta infatti essere il più veloce e allo stesso tempo il più debole quando vengono introdotti sostanziosi disturbi. Il compromesso tra velocità di convergenza e robustezza a piccoli disturbi si ha con l'ATS, il quale però sembra soffrire maggiormente della presenza di tempo-varianza delle frequenze piuttosto che della presenza di rumore.
Un possibile approfondimento dell'analisi effettuata, potrebbe consistere la ricerca di altre vie di tipo lineare grazie alle quali si garantirebbe che gli effetti dei disturbi restino sempre limitati. Un'altra idea potrebbe essere quella di migliorare la strategia adottata dal PI, introducendo ad esempio una parte derivativa o facendo intervenire la parte integrativa solo da un certo istante in poi. Lo studio dell'evoluzione della topologia del grafo e delle sue caratteristiche in funzione del protocollo di sincronizzazione impiegato, insieme allo studio del numero di dati da trasmettere (e quindi memorizzare) da ogni agente, restano comunque problemi aperti come testimoniano recenti lavori in letteratura.

A Codice MATLAB

A.1 SINCRONO

Graph.m

```
% Metodo di creazione random geometric graph (Ad matrice di adiacenza).
% In ingresso forniti il numero di nodi N ed il raggio r entro il quale i
% nodi vengono connessi.
% In uscita si ha la matrice di adiacenza Ad e di consenso P.

function [P,Ad] = Graph(N,r)

connect=0;          % variabile test di connessione

while ~connect

    Ad=zeros(N);
    d=zeros(1,N);

    v=rand(N,2);  % matrice Nx2 delle coordinate degli N punti

    for i=1:N
        for j=i+1:N

            if norm(v(i,:)-v(j,:))<r      % nodi connessi se loro distanza
                                          % minore di r
                Ad(i,j)=1;
                Ad(j,i)=1;

            end

        end
        d(i) = sum(Ad(i,:));              % grado del nodo i
    end

    P= zeros(N);                          % matrice di consenso


    % assegnazione dei pesi secondo strategia Metropolis-Hastings


    for i=1:N
        for j=i+1:N

            if Ad(i,j)

                P(i,j) = 1/(max(d(i),d(j))+1);
                P(j,i) = P(i,j);

            end

        end
```

```
        P(i,i) = 1-sum(P(i,:));
    end


    % verifica connessione grafo

    if min(min(P^N))>0
        connect = 1;
    end
end
```

Main.m

```
%% PROGRAMMA MAIN: grazie a questo programma è possibile simulare
%% l'andamento dei seguenti algoritmi sincroni:
%% 1) ATS;
%% 2) Kumar;
%% 3) PI;

clear all

Y =[];

Y1 =[];

Y2 =[];

for cycle=1:50

%% PARAMETRI DEL PROBLEMA

N=10;                           % numero di nodi

T=1;                            % quanto temporale

Iter=25000;                     % numero iterazioni

r=0.28;                         % raggio di connessione

[P,A]=Graph(N,r);               % matrici di adiacenza e consenso del grafo

s=sum(A);                       % vettore gradi nodi

sigma=0.1;                      % parametro velocità

sigma_n=0.0001;                 % parametro per rumore

value_vel= 0.00001;             % parametro per le velocità random


%% PARAMETRI RELATIVI AGLI ALGORITMI

Lambda=1.1;                     % PER KUMAR

rho = 0.5;                      % PER ATS
```

```
f11=1/2;                                % PER PI
f21=1/(2*T);

%% Alpha = velocità orologi locali
Alpha=1+(rand(N,1)-1/2)*sigma;
%% O = offeset orologi

O=rand(N,1)*10;

% orologio uno preso come riferimento

Alpha(1)=1;
O(1,1)=0;
% variabili aleatorie per random walk delle velocità (a)
a=[];      % matrice delle variabili aleatorie per le velocità
for e=1:Iter

    a=[a -value_vel + value_vel*2*rand(N,1)];     % variabile tra [-value_vel,
        value_vel]
end


%% Algoritmo PI-Sincrono in funzione
[x,y]=PI_Sinc(N,P,Alpha,O,Iter,T,f11,f21,sigma_n,a);

Y=[Y;y];
%% Algoritmo Kumar-Sincrono in funzione
[x1,y1]=Kumar_sinc(N,s,A,Alpha,O,Iter,sigma_n,Lambda,a);
Y1=[Y1;y1];
%% Algoritmo ATS-Sincrono in funzione

[x2,y2]=ATS_sincr(N,T,Iter,P,Alpha,O,sigma_n,rho,a);
Y2=[Y2;y2];
end
y=[];
y1=[];

y2=[];
for i=1:Iter
```

```
105     y =[y 1/cycle*sum(Y(:,i))];

        y1 =[y1 1/cycle*sum(Y1(:,i))];

        y2 =[y2 1/cycle*sum(Y2(:,i))];
110
    end


    %% PLOT PI
115
    figure(1)

    semilogy(y,'r')
    title('Algoritmo PI (sincrono): Varianza delle stime dei tempi');
120 xlabel('Iter');
    ylabel('var(TauBar)');
    grid on


125 figure(2)

    for l=1:N
        plot(x(l,:));
        hold on;
130 end

    title('Algoritmo PI (sincrono): Stime dei tempi');
    xlabel('Iter');
    ylabel('TauBar');
135 grid on

    % PLOT Kumar

    figure(3)
140
    semilogy(y1,'r')

    title('Algoritmo Kumar (sincrono): Varianza delle stime dei tempi');
    xlabel('Iter');
145 ylabel('var(TauBar)');
    grid on

    figure(4)


150
    for l=1:N
        plot(x1(l,:));
        hold on;
    end
155
    title('Algoritmo Kumar (sincrono): Stime dei tempi');
    xlabel('Iter');
    ylabel('TauBar');
    grid on
160

    %% PLOT ATS

    figure(5)
165
    semilogy(y2,'r')
```

```
title('Algoritmo ATS (sincrono): Varianza delle stime dei tempi');
xlabel('Iter');
ylabel('var(TauBar)');
grid on

figure(6)

for l=1:N
    plot(x2(l,:));
    hold on;
end

title('Algoritmo ATS (sincrono): Stime dei tempi');
xlabel('Iter');
ylabel('TauBar');
grid on

%% PLOT SOVRAPPOSTI

figure(7)

semilogy(y2,'r','Linewidth',1)
hold on;
semilogy(y1,'b','Linewidth',1)
hold on;
semilogy(y,'g','Linewidth',1)
legend('ATS','Kumar','PI')

title('Varianza delle stime dei tempi');
xlabel('Iter');
ylabel('var(TauBar)');
grid on
```

ATS_sincr.m

```
% Metodo per l'algoritmo ATS sincrono (cascata di due alg. di consenso).
%
% Input: numero nodi 'N', sampling time 'T', numero di iterazioni delle
% simulazioni 'Iter', matrice di consenso 'P', velocità 'Alpha',
% offsets 'O', varianza di rumore 'sigma_n', paremetro di tuning 'rho'.
%
% Output: stime delle leggi orarie TauBar e loro varianza z.

function [TauBar,z] = ATS_sincr(N,T,Iter,P,Alpha,O,sigma_n,rho,a)

    I=eye(N);

    Eta=ones(N);

    tau(:,1)=Alpha*1*T+O;                    % orologi locali inizializzati

    AlphaBar(:,1)=ones(N,1);

    OBar(:,1)=zeros(N,1);

    TauBar(:,1)=AlphaBar(:,1).*tau(:,1)+OBar(:,1);
```

```
    AlphaBar(:,2)=P.*Eta*AlphaBar(:,1);
    AlphaBar(1,2)=AlphaBar(1,1);                % velocità riferimento costante
    OBar(:,2)=OBar(:,1)+(P-I)*TauBar(:,1);
    OBar(1,2)=OBar(1,1);                        % offset riferimento costante


    for t=2:Iter

        %Rumore=zeros(N);
        Rumore=(rand(N)-1/2)*2*sigma_n;

        Rumore=Rumore-diag(diag(Rumore));

        for x=1:N

            if Alpha(x)+a(x,t) >= 0.9 && Alpha(x)+a(x,t) <= 1.1

                Alpha(x)=(Alpha(x)+a(x,t));

            end
        end

        tau(:,t)=Alpha*t*T+O;          % orologi locali aggiornati
        TauBar(:,t)= TauBar(:,t-1)+AlphaBar(:,t).*(tau(:,t)-tau(:,t-1))+OBar
            (:,t)-OBar(:,t-1);

        for i=1:N

            for j=1:N

                if i==j

                Eta(i,j)=1;

                else

                    Eta(i,j)=rho*Eta(i,j)+(1-rho)*((tau(j,t)-tau(j,t-1)+Rumore
                        (i,j))/(tau(i,t)-tau(i,t-1)));

                end

            end

        end

    AlphaBar(:,t+1)=P.*Eta*AlphaBar(:,t);

    AlphaBar(1,t+1)=AlphaBar(1,t);          % velocità riferimento costante

    OBar(:,t+1)=OBar(:,t)+(P-I)*TauBar(:,t);

    OBar(1,t+1)=OBar(1,t);                  % offset riferimento costante

    z(t)=var(TauBar(:,t),1);                % varianza delle stime normaliz.


    end

end
```

Kumar_sinc.m

```
% Metodo per l'algoritmo di Kumar sincrono.
%
% Input: numero di nodi 'N', vettore dei gradi dei nodi 's', matrice di
% adiacenza 'A', velocità 'Alpha', offsets 'O', numero di iterazoni 'Iter',
% varianza di rumore 'sigma_n', parametro di tuning 'Lambda'.
%
% Output: stime dei tempi 'TauBar' e varianza delle stesse 'z'.

function [TauBar,z] = Kumar_sinc(N,s,A,Alpha,O,Iter,sigma_n,Lambda,a)

%% AlphabarLog = stime velocità orologi locali in scala logaritmica

AlphabarLog=zeros(N,1);

%% OBar = stime offset orologi

Obar=zeros(N,1);                    % inizializzazione stima degli offset

%% tau = orologi locali

tau(:,1)=O;

tau_n(:,1)=O+(rand(N,1)-1/2)*2*0.0000001;


%% TauBar = stime orologi locali

TauBar(:,1)=tau(:,1);

S=zeros(N);

for k=1:Iter

    % n=0;                                  % rumore assente

    n=(rand(N,1)-1/2)*2*sigma_n;            % rumore presente


    % stima dell'istante di tempo

    for i=1:N

        % riga dei tempi fittizi (per ogni k è una riga)
        t_bar(i)=(tau(i,k)- Obar(i,k))/exp(AlphabarLog(i,k));

    end

    Sinv=(diag(s))^(-1);

    % aggiornamento stime offset

    for i=1:N

        Obar(i,k+1)=0;

```

```
        for j=1:N

            ORel= tau(i,k)-exp(AlphabarLog(i,k))*t_bar(i)-(tau_n(j,k) -...
                  exp(AlphabarLog(j,k))*t_bar(i));

            Obar(i,k+1)=Obar(i,k+1)+Sinv(i,i)*A(i,j)*(Obar(j,k)+ ORel);

        end

    end

    Obar(1,k+1)=0;                    % stima offset di rif. a zero

    for x=1:N

            if Alpha(x)+a(x,k) >= 0.9 && Alpha(x)+a(x,k) <= 1.1

                Alpha(x)=(Alpha(x)+a(x,k));

            end
    end

    tau(:,k+1)=Alpha*k+O;             % aggiornamento leggi orarie reali

    tau_n(:,k+1)=tau(:,k+1)+n;        % leggi orarie sporcate

    for i=1:N

        for j=1:N

            if j==i

                S(i,i)=1/Lambda*S(i,i)+(tau_n(i,k+1)-tau_n(i,k))*...
                       (tau_n(i,k+1)-tau_n(i,k));

            else

                S(i,j)=1/Lambda*S(i,j)+(tau(i,k+1)-tau(i,k))*...
                       (tau_n(j,k+1)-tau_n(j,k));

            end

        end

    end


    AlphabarLog(:,k+1)=zeros(N,1);


    % Aggiornamento stime velocità in scala logaritmica sfruttando
    % l'alpha i j ottimo soluzione di RLS


    for i=1:N

        for j=1:N

            AlphabarLog(i,k+1)=AlphabarLog(i,k+1)+1/s(i)*A(i,j)*...
                               (AlphabarLog(j,k)+log(S(i,j)/S(j,j)));

        end
```

```
    end

    AlphabarLog(1,k+1)=0;              % stima velocità di rif. a 0

    TauBar(:,k+1)=(tau(:,k+1)-Obar(:,k+1))./exp(AlphabarLog(:,k+1));

    z(k)= var(TauBar(:,k),1);

end
```

PI_Sinc.m

```
% Metodo PI sincrono.
%
% Input: numero nodi 'N', matrice di consenso 'P', velocità 'Alpha',
% offsets 'O', numero di iterazioni delle simulazioni 'Iter',
% sampling time 'T', parametri di tuning 'f11' ed 'f21',
% varianza di rumore 'sigma_n'.
%
% Output: stime delle leggi orarie 'x_primo' e loro varianza 'z'.

function [x_primo,z] = PI_Sinc(N,P,Alpha,O,Iter,T,f11,f21,sigma_n,a)

I=eye(N);

K=I-P;

x_primo(:,1)=O;        % stima del tempo naturale di ogni orologio

D=diag(Alpha);

x_secondo(:,1)=ones(N,1);

X=zeros(2*N,1);

X(:,1)=[x_primo(:,1)' x_secondo(:,1)']';

%% Aggiornamento algoritmo

for t=1:Iter

    for i=1:N

        Rumore=(rand(N)-1/2)*2*sigma_n;
        Rumore=Rumore-diag(diag(Rumore));

        S=0;

        for j=1:N

            % Rumore(j,i) rappresenta il rumore della comunicazione j-->i

            S=S+K(i,j)*(x_primo(j,t)-x_primo(i,t)+Rumore(j,i));

        end
```

```
            % Aggiornamento leggi di controllo

            u_primo=-f11*S;
            u_secondo=-f21*S;


            if Alpha(i)+a(i,t) >= 0.9 && Alpha(i)+a(i,t) <= 1.1

                    Alpha(i)=(Alpha(i)+a(i,t));

            end

            % Aggiornamento stato

            x_secondo(i,t+1)=x_secondo(i,t)+u_secondo;
            x_primo(i,t+1)=x_primo(i,t)+u_primo+Alpha(i)*T*x_secondo(i,t+1);

        end

        z(t)=var(x_primo(:,t),1);

    end
```

A.2 ASINCRONO

Graph_din.m

```
    % Metodo di creazione random geometric graph (Ad matrice di adiacenza).
    % In ingresso forniti il numero di nodi N ed il raggio r entro il quale i
    % nodi vengono connessi.
    % In uscita si ha la matrice di adiacenza Ad e di consenso P.


    function [P,Ad] = Graph_din(N,r,Iter,com)

    connect=0;          % variabile test di connessione


    while ~connect

        Ad=zeros(N);
        d=zeros(1,N);

        v=rand(N,2);  % matrice Nx2 delle coordinate degli N punti

        for i=1:N
            for j=i+1:N

                if norm(v(i,:)-v(j,:))<r      % nodi connessi se loro distanza
                                              % minore di r
                    Ad(i,j)=1;
                    Ad(j,i)=1;

                end

            end
```

```
        d(i) = sum(Ad(i,:));                % grado del nodo i
    end

    if min(min(Ad^N))>0                     % verifica connessione grafo
        connect = 1;
    end

end

    % assegnazione dei pesi secondo strategia Broadcast

    P= [];                                  % matrice di consenso

    for k=1:Iter

        P=[P Broadcast(Ad,com(k))];

    end
```

Broadcast.m

```
% ALGORITMO DISTRIBUITO PER GRAFI DIRETTI E CONTENENTI SELF LOOPS
% Broadcast
% CASO DINAMICO: si accende un solo nodo alla volta in ogni istante
% temporale

% input:  matrice di adiacenza associata a grafo diretto di N nodi e
% indice k del nodo "acceso" di cui si vuole conoscere la matrice dei pesi
% output: matrice Q dei pesi relativa ad ogni nodo (istante temporale)

function Q_B = Broadcast(A,k)

N=length(A);
I=eye(N);
q=0.5;
X=zeros(N,N);

for i=1:N
    S=zeros(N,N);

    for j=1:N
        if (j==i) || (j~=i && A(i,j)==0)
            Z=zeros(N);
            S=S+Z;

        else
            Z=I(:,j)*(I(:,j)-I(:,i))';
            S=S+Z;

        end

    end
```

```
    Q=I-q*S;
    if i==1
        X=Q;
    else
        X=[X,Q];
    end

end

Q_B=X(:,(k-1)*N+1:k*N);
```

Main_A.m

```
%% PROGRAMMA MAIN: grazie a questo programma è possibile simulare
%% l'andamento dei seguenti algoritmi asincroni:
%% 1) ATS;
%% 2) Kumar;
%% 3) PI;

clear all

Y_b=[];
Y_g=[];

Y2_b=[];
Y2_g=[];

for cycle=1:1

%% PARAMETRI DEL PROBLEMA

N=10;                               % numero di nodi

T=1;                                % quanto temporale

Iter=10000;                          % numero iterazioni (almeno 5000)

r=0.28;                             % raggio di connessione

%com=randi(N,1,Iter);
com=ceil(N.*rand(1,Iter));          % vettore di comunicazione

[Ptot,A]=Graph_din(N,r,Iter,com);    % matrici di adiacenza e consenso del
    grafo

s=sum(A);                           % vettore gradi nodi

sigma=0.1;                          % parametro velocità

sigma_n=0.0001;                      % parametro 2 per rumore

value_vel= 0;     % parametro per le velocità random

%% PARAMETRI RELATIVI AGLI ALGORITMI

rho = 0.5;                          % PER ATS

```

```
f11=1/2;                                % PER PI

f21=1/(2*T);

%% Alpha = velocità orologi locali

Alpha=1+(rand(N,1)-1/2)*sigma;

%% O = offeset orologi

O=rand(N,1)*10;


% orologio uno preso come riferimento

Alpha(1)=1;

O(1,1)=0;

% variabili aleatorie per random walk dei tempi (c) e delle velocità (a)

c=[];      % stringa dei tempi ("semi-aleatori")
a=[];      % matrice delle variabili aleatorie per le velocità

for e=1:Iter

    c=[c e+T.*rand(1,1)];              % variabile tra e+[0,1]
    a=[a -value_vel + value_vel*2*rand(N,1)];     % variabile tra [-value_vel,
        value_vel]

end



%% Algoritmo PI-Asincrono

%  Gossip

[x_g,y_g]=PI_Asinc_GS(N,A,Alpha,O,Iter,T,sigma_n,com,c,a);

Y_g=[Y_g;y_g];

%  Broadcast

[x_b,y_b]=PI_Asinc_B(N,A,Alpha,O,Iter,T,sigma_n,com,c,a);

Y_b=[Y_b;y_b];

%% Algoritmo ATS-Asincrono

%  Gossip

[x2_g,y2_g]=ATS_Asinc_GS(N,T,Iter,A,Alpha,O,sigma_n,rho,com,c,a);

Y2_g=[Y2_g;y2_g];


%  Broadcast
```

```
[x2_b,y2_b]=ATS_Asinc_B(N,T,Iter,Ptot,Alpha,O,sigma_n,rho,com,c,a);

Y2_b=[Y2_b;y2_b];


end

y_b=[];
y_g=[];

y2_b=[];
y2_g=[];

for i=1:Iter

    y_b =[y_b 1/cycle*sum(Y_b(:,i))];
    y_g =[y_g 1/cycle*sum(Y_g(:,i))];

    y2_b =[y2_b 1/cycle*sum(Y2_b(:,i))];
    y2_g =[y2_g 1/cycle*sum(Y2_g(:,i))];

end




%% ********************** SIMULAZIONI GOSSIP **********************

%% PLOT PI

figure(1)

semilogy(y_g,'r')
title('Algoritmo PI (gossip): Varianza delle stime dei tempi');
xlabel('Iter');
ylabel('var(TauBar)');
grid on


figure(2)

for l=1:N
    plot(x_g(l,:));
    hold on;
end

title('Algoritmo PI (gossip): Stime dei tempi');
xlabel('Iter');
ylabel('TauBar');
grid on

%% PLOT ATS

figure(5)

semilogy(y2_g,'r')

title('Algoritmo ATS (gossip): Varianza delle stime dei tempi');
```

```
xlabel('Iter');
ylabel('var(TauBar)');
grid on

figure(6)

for l=1:N
    plot(x2_g(l,:));
    hold on;
end

title('Algoritmo ATS (gossip): Stime dei tempi');
xlabel('Iter');
ylabel('TauBar');
grid on



%% ********************** SIMULAZIONI BROADCAST **********************
%% PLOT PI

figure(7)

semilogy(y_b,'r')
title('Algoritmo PI (broadcast): Varianza delle stime dei tempi');
xlabel('Iter');
ylabel('var(TauBar)');
grid on

figure(8)

for l=1:N
    plot(x_b(l,:));
    hold on;
end

title('Algoritmo PI (broadcast): Stime dei tempi');
xlabel('Iter');
ylabel('TauBar');
grid on


%% PLOT ATS

figure(11)

semilogy(y2_b,'r')

title('Algoritmo ATS (broadcast): Varianza delle stime dei tempi');
xlabel('Iter');
ylabel('var(TauBar)');
grid on

figure(12)

for l=1:N
    plot(x2_b(l,:));
    hold on;
```

```
end

title('Algoritmo ATS (broadcast): Stime dei tempi');
xlabel('Iter');
ylabel('TauBar');
grid on


%% PLOT varianze G&B sovrapposte per ciascun algoritmo

% ATS
figure(13)

semilogy(c,y2_b,'r')
hold on;
semilogy(c,y2_g,'b')
legend('ATS Broadcast','ATS Gossip')

title('Varianza delle stime dei tempi');
xlabel('Iter');
ylabel('var(TauBar)');
grid on

% PI
figure(15)

semilogy(c,y_b,'r')
hold on;
semilogy(c,y_g,'b')
legend('PI Broadcast','PI Gossip')

title('Varianza delle stime dei tempi');
xlabel('Iter');
ylabel('var(TauBar)');
grid on

%% PLOT confronto varianze PI Vs ATS

figure(16)

semilogy(c,y_b,'r')
hold on;
semilogy(c,y2_b,'b')
legend('PI Broadcast','ATS Broadcast')

title('Varianza delle stime dei tempi');
xlabel('Iter');
ylabel('var(TauBar)');
grid on

figure(17)

semilogy(c,y_g,'r')
hold on;
semilogy(c,y2_g,'b')
legend('PI Gossip','ATS Gossip')

title('Varianza delle stime dei tempi');
xlabel('Iter');
ylabel('var(TauBar)');
grid on
```

ATS_Asinc_B.m

```
% Metodo per l'algoritmo ATS Asincrono con metodo Broadcast.
%
% Input: numero nodi 'N', sampling time 'T', numero di iterazioni delle
% simulazioni 'Iter', matrice di consenso 'P', velocità 'Alpha',
% offsets 'O', varianza di rumore 'sigma_n', paremetro di tuning 'rho',
% vettore di comunicazione 'com'.
%
% Output: stime delle leggi orarie TauBar e loro varianza z.

function [TauBar,z] = ATS_Asinc_B(N,T,Iter,Ptot,Alpha,O,sigma_n,rho,com,c,a)

    I=eye(N);

    P=Ptot(:,1:N);

    Eta=ones(N);

    tau(:,1)=Alpha*1*T+O;              % orologi locali inizializzati

    AlphaBar(:,1)=ones(N,1);

    OBar(:,1)=zeros(N,1);

    TauBar(:,1)=AlphaBar(:,1).*tau(:,1)+OBar(:,1);

    AlphaBar(:,2)=P.*Eta*AlphaBar(:,1);
    AlphaBar(1,2)=AlphaBar(1,1);              % velocità riferimento costante
    OBar(:,2)=OBar(:,1)+(P-I)*TauBar(:,1);
    OBar(1,2)=OBar(1,1);                      % offset riferimento costante


    for t=2:Iter

        i=(com(t));

        l=c(t);                               % istante di tempo "random"

        P=Ptot(:,(t-1)*N+1:t*N);

        %Rumore=zeros(N);
        Rumore=(rand(N)-1/2)*2*sigma_n;

        Rumore=Rumore-diag(diag(Rumore));

        for x=1:N

            if Alpha(x)+a(x,t) >= 0.9 && Alpha(x)+a(x,t) <= 1.1

                Alpha(x)=(Alpha(x)+a(x,t));

            end
        end


        tau(:,t)= Alpha*l*T+O;             % orologi locali aggiornati

        TauBar(:,t)= TauBar(:,t-1)+AlphaBar(:,t).*(tau(:,t)-tau(:,t-1))+OBar
            (:,t)-OBar(:,t-1);
```

```
60
            for j=1:N
                if i==j
                Eta(i,j)=1;
                else
                    Eta(j,i)=rho*Eta(j,i)+(1-rho)*((tau(i,t)-tau(i,t-1)+Rumore
                        (j,i))/(tau(j,t)-tau(j,t-1)));
70              end
            end

75

    AlphaBar(:,t+1)=P.*Eta*AlphaBar(:,t);
    AlphaBar(1,t+1)=AlphaBar(1,t);          % velocità riferimento costante
80
    OBar(:,t+1)=OBar(:,t)+(P-I)*TauBar(:,t);
    OBar(1,t+1)=OBar(1,t);                  % offset riferimento costante
85  z(t)=var(TauBar(:,t),1);                % varianza delle stime normaliz.


    end

90 end
```

Kumar_Asinc_B.m

```
% Metodo per l'algoritmo di Kumar Asincrono con metodo Broadcast.
%
% Input: numero di nodi 'N', matrice di adiacenza'A', velocità 'Alpha',
5 % offsets 'O', numero di iterazoni 'Iter', varianza di rumore 'sigma_n',
% parametro di tuning 'Lambda' e vettore di comunicazione 'com'.
%
% Output: stime dei tempi 'TauBar' e varianza delle stesse 'z'.

10
function [TauBar,z] = Kumar_Asinc_B(N,A,Alpha,O,Iter,sigma_n,Lambda,com)


%% AlphabarLog = stime velocità orologi locali in scala logaritmica
15
AlphabarLog=zeros(N,1);

%% OBar = stime offset orologi

20 Obar=zeros(N,1);                  % inizializzazione stima degli offset


%% tau = orologi locali
```

```
tau(:,1)=O;
tau_n(:,1)=O+(rand(N,1)-1/2)*2*0.0000001;


%% TauBar = stime orologi locali
TauBar(:,1)=tau(:,1);
S=zeros(N);
for k=1:Iter
    j=com(k);
    c=rand(1,1);                % variabile aleatoria uniforme tra [0,T]
    l=k+c;                      % istante di tempo "random"

    % n=0;                                  % rumore assente
    n=(rand(N,1)-1/2)*2*sigma_n;            % rumore presente

    % stima dell'istante di tempo
    for i=1:N
        % riga dei tempi fittizi (per ogni k è una riga)
        t_bar(i)=(tau(i,k)- Obar(i,k))/exp(AlphabarLog(i,k));
    end
    % aggiornamento stime offset
    for i=1:N
        ORel= tau(i,k)-exp(AlphabarLog(i,k))*t_bar(i)-(tau_n(j,k)-...
              exp(AlphabarLog(j,k))*t_bar(i));
        Obar(i,k+1)=A(i,j)*(Obar(j,k)+ ORel);
    end
    Obar(1,k+1)=0;                  % stima offset di rif. a zero
    tau(:,k+1)=Alpha*l+O;           % aggiornamento leggi orarie reali
    tau_n(:,k+1)=tau(:,k+1)+n;      % leggi orarie sporcate
    for i=1:N
            if j==i
                S(i,i)=1/Lambda*S(i,i)+(tau_n(i,k+1)-tau_n(i,k))*...
                       (tau_n(i,k+1)-tau_n(i,k));
            else
                S(i,j)=1/Lambda*S(i,j)+(tau(i,k+1)-tau(i,k))*...
```

```
                        (tau_n(j,k+1)-tau_n(j,k));

            end
90
    end

    AlphabarLog(:,k+1)=zeros(N,1);
95

    % Aggiornamento stime velocità in scala logaritmica sfruttando
    % l'alpha i j ottimo soluzione di RLS

100
    for i=1:N

        AlphabarLog(i,k+1)=A(i,j)*(AlphabarLog(j,k)+log(S(i,j)/S(j,j)));

105 end

    AlphabarLog(1,k+1)=0;           % stima velocità di rif. a 0

    TauBar(:,k+1)=(tau(:,k+1)-Obar(:,k+1))./exp(AlphabarLog(:,k+1));
110
    z(k)= var(TauBar(:,k),1);

end
```

PI_Asinc_B.m

```
% Metodo PI Asincrono con metodo Broadcast.
%
% Input: numero nodi 'N', matrice di consenso 'P', velocità 'Alpha',
5 % offsets 'O', numero di iterazioni delle simulazioni 'Iter',
% sampling time 'T', parametri di tuning 'f11' ed 'f21',
% varianza di rumore 'sigma_n' e il vettore di comunicazione 'com'.
%
% Output: stime delle leggi orarie 'x_primo' e loro varianza 'z'.
10

function [x_primo,z] = PI_Asinc_B(N,A,Alpha,O,Iter,T,sigma_n,com,c,a)

x_primo(:,1)=O;       % stima del tempo naturale di ogni orologio
15
x_secondo(:,1)=ones(N,1);


%% Aggiornamento algoritmo
20
l_1 =T.*rand(1,1);


for t=1:Iter
25
    i=com(t);                       % nodo eletto

    l=c(t);                          % istante di tempo "random"
```

```
%      % i <--> j

        Rumore=(rand(N)-1/2)*2*sigma_n;
        Rumore=Rumore-diag(diag(Rumore));


       % Rumore(i,rk)  comunicazione i-->r1 , i-->r2 ...


        for r=1:N

          % Aggiornamento leggi di controllo

          if A(i,r) == 1

              if r==i

                  u_primo=0;
                  u_secondo=0;

              else

                  u_primo=1/2*(x_primo(i,t)-x_primo(r,t)+Rumore(i,r));
                  u_secondo=0.001/2*(x_primo(i,t)-x_primo(r,t)+...
                            Rumore(i,r));

              end

          else

              u_primo=0;
              u_secondo=0;

          end

          if Alpha(r)+a(r,t) >= 0.9 && Alpha(r)+a(r,t) <= 1.1

              Alpha(r)=(Alpha(r)+a(r,t));

          end

          % Aggiornamento stato

          x_secondo(r,t+1)=x_secondo(r,t)+u_secondo;
          x_primo(r,t+1)=x_primo(r,t)+u_primo+Alpha(r)*(l-l_1)*...
                         x_secondo(r,t+1);


        end

        l_1 = l;            % tengo mem dell'ist temporale app trascorso


    z(t)=var(x_primo(:,t),1);

end
```

ATS_Asinc_GS.m

```
% Metodo per l'algoritmo ATS Asincrono con metodo Gossip.
%
% Input: numero nodi 'N', sampling time 'T', numero di iterazioni delle
% simulazioni 'Iter', matrice di consenso 'P', velocità 'Alpha',
% offsets 'O', varianza di rumore 'sigma_n', paremetro di tuning 'rho',
% vettore di comunicazione 'com'.
%
% Output: stime delle leggi orarie TauBar e loro varianza z.

function [TauBar,z] = ATS_Asinc_GS(N,T,Iter,A,Alpha,O,sigma_n,rho,com,c,a)

    I=eye(N);
    q=0.5;
    P = I;

    i=(com(1));

    v_i=[];

    for p=1:N

        if A(i,p) == 1

            v_i= [v_i,p];            % vettore vicini nodo eletto

        end
    end

    %k=randi(length(v_i),1,1);
    k=ceil(length(v_i).*rand(1,1)); % indice di v_i

    j=v_i(k);                        % nodo vicino eletto


    P(1,1)=1-q;
    P(j,j)=1-q;
    P(1,j)=q;
    P(j,1)=q;


    Eta=ones(N);

    tau(:,1)=Alpha*1*T+O;                    % orologi locali inizializzati

    AlphaBar(:,1)=ones(N,1);

    OBar(:,1)=zeros(N,1);

    TauBar(:,1)=AlphaBar(:,1).*tau(:,1)+OBar(:,1);

    AlphaBar(:,2)=P.*Eta*AlphaBar(:,1);
    AlphaBar(1,2)=AlphaBar(1,1);             % velocità riferimento costante
    OBar(:,2)=OBar(:,1)+(P-I)*TauBar(:,1);
    OBar(1,2)=OBar(1,1);                     % offset riferimento costante


    %Eta=ones(N);

    for t=2:Iter
```

```
P = I;

i=(com(t));

l=c(t);                                % istante di tempo "random"

v_i=[];

for p=1:N

    if A(i,p) == 1

        v_i= [v_i,p];                  % vettore vicini nodo eletto

    end
end

%k=randi(length(v_i),1,1);
k=ceil(length(v_i).*rand(1,1)); % indice di v_i

j=v_i(k);                              % nodo vicino eletto

% Matrice di consenso P (comunicazione Gossip simmetrico)

P(i,i)=1-q;
P(j,j)=1-q;
P(i,j)=q;
P(j,i)=q;


%Rumore=zeros(N);
Rumore=(rand(N)-1/2)*2*sigma_n;

Rumore=Rumore-diag(diag(Rumore));

for x=1:N

    if Alpha(x)+a(x,t) >= 0.9 && Alpha(x)+a(x,t) <= 1.1

        Alpha(x)=(Alpha(x)+a(x,t));

    end
end

tau(:,t)=Alpha*l*T+O;          % orologi locali aggiornati
TauBar(:,t)= TauBar(:,t-1)+AlphaBar(:,t).*(tau(:,t)-tau(:,t-1))+OBar
    (:,t)-OBar(:,t-1);

Eta(i,i)=1;
Eta(j,j)=1;

Eta(j,i)=rho*Eta(j,i)+(1-rho)*((tau(i,t)-tau(i,t-1)+Rumore(j,i))/(tau(
    j,t)-tau(j,t-1)));
Eta(i,j)=rho*Eta(i,j)+(1-rho)*((tau(j,t)-tau(j,t-1)+Rumore(i,j))/(tau(
    i,t)-tau(i,t-1)));


AlphaBar(:,t+1)=P.*Eta*AlphaBar(:,t);

AlphaBar(1,t+1)=AlphaBar(1,t);         % velocità riferimento costante
```

```
120         OBar(:,t+1)=OBar(:,t)+(P-I)*TauBar(:,t);
            OBar(1,t+1)=OBar(1,t);                  % offset riferimento costante
            z(t)=var(TauBar(:,t),1);                % varianza delle stime normaliz.
125
    end

130 end
```

Kumar_Asinc_GS.m

```
% Metodo per l'algoritmo di Kumar Asincrono con metodo Gossip.
%
% Input: numero di nodi 'N', matrice di adiacenza 'A', velocità 'Alpha',
5 % offsets 'O', numero di iterazoni 'Iter', varianza di rumore 'sigma_n',
% parametro di tuning 'Lambda' e vettore di comunicazione 'com'.
%
% Output: stime dei tempi 'TauBar' e varianza delle stesse 'z'.

10
function [TauBar,z] = Kumar_Asinc_GS(N,A,Alpha,O,Iter,sigma_n,Lambda,com)

%% AlphabarLog = stime velocità orologi locali in scala logaritmica
15
AlphabarLog=zeros(N,1);

%% OBar = stime offset orologi

20 Obar=zeros(N,1);                     % inizializzazione stima degli offset

%% tau = orologi locali

25 tau(:,1)=O;

tau_n(:,1)=O+(rand(N,1)-1/2)*2*0.0000001;


30
%% TauBar = stime orologi locali

TauBar(:,1)=tau(:,1);

35 S=zeros(N);

for k=1:Iter

    j=com(k);
40
    c=rand(1,1);                    % variabile aleatoria uniforme tra [0,T]

    l=k+c;                              % istante di tempo "random"

45  v_j=[];
```

```
    for p=1:N

        if A(j,p) == 1
50
            v_j= [v_j,p];            % vettore vicini nodo eletto

        end
    end
55
    h=randi(length(v_j),1,1);        % indice di v_i

    r=v_j(h);                        % nodo vicino eletto

60
    % j <--> r

    % n=0;                                    % rumore assente

65  n=(rand(N,1)-1/2)*2*sigma_n;               % rumore presente


    % stima dell'istante di tempo

70  for i=1:N

        % riga dei tempi fittizi (per ogni k è una riga)
        t_bar(i)=(tau(i,k)- Obar(i,k))/exp(AlphabarLog(i,k));

75  end

    % aggiornamento stime offset

    ORel_rj= tau(r,k)-exp(AlphabarLog(r,k))*t_bar(r)-(tau_n(j,k) -...
80           exp(AlphabarLog(j,k))*t_bar(r));

    Obar(r,k+1)=A(r,j)*(Obar(j,k)+ ORel_rj);


85  ORel_jr= tau(j,k)-exp(AlphabarLog(j,k))*t_bar(j)-(tau_n(r,k) -...
           exp(AlphabarLog(r,k))*t_bar(j));

    Obar(j,k+1)=A(j,r)*(Obar(r,k)+ ORel_jr);

90


    Obar(1,k+1)=0;                   % stima offset di rif. a zero

    tau(:,k+1)=Alpha*l+O;            % aggiornamento leggi orarie reali
95
    tau_n(:,k+1)=tau(:,k+1)+n;       % leggi orarie sporcate


    S(j,j)=1/Lambda*S(j,j)+(tau_n(j,k+1)-tau_n(j,k))*(tau_n(j,k+1) -...
100        tau_n(j,k));

    S(r,r)=1/Lambda*S(r,r)+(tau(r,k+1)-tau(r,k))*(tau_n(r,k+1)-tau_n(r,k));


105 S(j,r)=1/Lambda*S(j,r)+(tau_n(j,k+1)-tau_n(j,k))*(tau_n(r,k+1) -...
        tau_n(r,k));
    S(r,j)=1/Lambda*S(r,j)+(tau_n(r,k+1)-tau_n(r,k))*(tau_n(j,k+1) -...
```

```
            tau_n(j,k));

110

    AlphabarLog(:,k+1)=zeros(N,1);

115 % Aggiornamento stime velocità in scala logaritmica sfruttando
    % l'alpha i j ottimo soluzione di RLS

    AlphabarLog(r,k+1)=A(r,j)*(AlphabarLog(j,k)+log(S(r,j)/S(j,j)));
    AlphabarLog(j,k+1)=A(j,r)*(AlphabarLog(r,k)+log(S(j,r)/S(r,r)));
120

    AlphabarLog(1,k+1)=0;           % stima velocità di rif. a 0

    TauBar(:,k+1)=(tau(:,k+1)-Obar(:,k+1))./exp(AlphabarLog(:,k+1));
125

    z(k)= var(TauBar(:,k),1);

end
```

PI_Asinc_GS.m

```
% Metodo PI Asincrono con metodo Gossip.
%
% Input: numero nodi 'N', matrice di consenso 'P', velocità 'Alpha',
5 % offsets 'O', numero di iterazioni delle simulazioni 'Iter',
% sampling time 'T', parametri di tuning 'f11' ed 'f21',
% varianza di rumore 'sigma_n' e il vettore di comunicazione 'com'.
%
% Output: stime delle leggi orarie 'x_primo' e loro varianza 'z'.
10

function [x_primo,z] = PI_Asinc_GS(N,A,Alpha,O,Iter,T,sigma_n,com,c,a)

x_primo(:,1)=O;       % stima del tempo naturale di ogni orologio
15
x_secondo(:,1)=ones(N,1);


%% Aggiornamento algoritmo
20
l_1 =T.*rand(1,1);


for t=1:Iter
25
    i=com(t);                       % nodo eletto

    l=c(t);                          % istante di tempo "random"

30  v_i=[];

    for p=1:N

        if A(i,p) == 1
35
```

```
            v_i= [v_i,p];              % vettore vicini nodo eletto

        end
    end

    %k=randi(length(v_i),1,1);
    k=ceil(length(v_i).*rand(1,1)); % indice di v_i

    j=v_i(k);                          % nodo vicino eletto

    % i <--> j

        Rumore=(rand(N)-1/2)*2*sigma_n;
        Rumore=Rumore-diag(diag(Rumore));


        % Rumore(j,i) rappresenta il rumore della comunicazione j-->i
        % Rumore(i,jk)  comunicazione i-->j1 , i-->j2 ...


        for r=1:N

           % Aggiornamento leggi di controllo

            if r~=i && r~=j

                u_primo=0;
                u_secondo=0;

            else if r==i

                u_primo=1/2*(x_primo(j,t)-x_primo(i,t)+Rumore(j,i));
                u_secondo=0.001/2*(x_primo(j,t)-x_primo(i,t)+Rumore(j,i));

                else

                    u_primo=1/2*(x_primo(i,t)-x_primo(j,t)+Rumore(i,j));
                    u_secondo=0.001/2*(x_primo(i,t)-x_primo(j,t)+...
                              Rumore(i,j));
                end

            end

            if Alpha(r)+a(r,t) >= 0.9 && Alpha(r)+a(r,t) <= 1.1

                Alpha(r)=(Alpha(r)+a(r,t));

            end

                    % Aggiornamento stato

            x_secondo(r,t+1)=x_secondo(r,t)+u_secondo;
            x_primo(r,t+1)=x_primo(r,t)+u_primo+Alpha(r)*(l-l_1)*...
                           x_secondo(r,t+1);

                    end

        l_1 = l;              % tengo mem dell'ist temporale app trascorso

    z(t)=var(x_primo(:,t),1);

end
```

Riferimenti bibliografici

[1] Federica Garin, Luca Schenato *A survey on distributed estimation and control applications using linear consensus algorithms*

[2] Luca Schenato, Giovanni Gamba *A distributed consensus protocol for clock syncronization in wireless sensor network*

[3] Luca Schenato, Federico Fiorentin *AverageTimeSynch: a consensus-based protocol for time synchronization in wireless sensor networks*

[4] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre *Fast unfolding of communities in large networks*

[5] Roberto Solis, Vivek S. Bokar, P. R. Kumar *A new Distributed Time Synchronization Protocol for Multihop Wireless Networks*

[6] Arvind Giridhar, P. R. Kumar *Distributed Clock Synchronization over Wireless Networks: Algorithms and Analysis*

[7] Ruggero Carli, Sandro Zampieri, *Networked clock synchronization based on second order linear consensus algorithms*

[8] R. Carli, A. Chiuso, L. Schenato, S. Zampieri, *A PI Consensus Controller for Networked Clocks Syncronization*

[9] Mei Lang, Yik-Chung Wu *Distributed clock synchronization for wireless sensor networks using belief propagation*