ALMA MATER STUDIORUM
UNIVERSITA` DI BOLOGNA

FACOLTA` DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Triennale in Informatica

IMPLEMENTAZIONE E VALUTAZIONE DI MODELLI DI MOBILITÀ SULLA PIATTAFORMA GAIA/ARTIS

Tesi di Laurea in Reti di Calcolatori

Relatore:
Gabriele D'Angelo

Presentata da:
PIRAZZINI SERENA

Sessione III
Anno Accademico 2010/2011

INDICE

INTRODUZIONE

La finalità di questo progetto è quella di sviluppare una serie di modelli di mobilità, sviluppati nell'ambito della simulazione parallela e distribuita, e verificarne le varie differenze tramite l'implementazione di un meccanismo d'infezione.

La simulazione parallela e distribuita si riferisce alla tecnica utilizzata nell'esecuzione della simulazione quando questa è affidata ad un insieme di unità di elaborazione (physical execution unit, PEU), interconnesse da una rete di comunicazione.

In questo caso la rete di comunicazione è composta da workstation interconnesse da una rete ad hoc. Una rete ad-hoc mobile (Mobile Ad-hoc NETwork, MANET) è un sistema autonomo di terminali mobili che sono liberi di muoversi casualmente e di auto organizzarsi, nonostante la topologia wireless cambi rapidamente e in modo imprevedibile [5].

Lo scopo del progetto è quello di implementare e studiare diversi modelli di mobilità che, utilizzati nella simulazione di una rete ad hoc, rappresentano accuratamente e realisticamente il movimento di utenti mobili (o nel nostro caso, nodi mobili) all'interno di un sistema, in modo da offrire più informazioni possibili nella scelta del modello da utilizzare per un protocollo di rete.

In questo sistema, costituito da uno spazio toroidale bidimensionale (ossia una superficie a forma di ciambella), i nodi mobili si muovono secondo un determinato modello di mobilità scelto inizialmente, e si scambiano messaggi tra loro utilizzando un raggio d'azione definito.

Il meccanismo d'infezione implementato permette di studiare i modelli di mobilità implementati e verificare quale impiega il maggior o il minor tempo per raggiungere l'infezione totale del sistema.

Successivamente sono state studiate le prestazioni del simulatore utilizzato, simulando il sistema implementato su una macchina del cluster di simulazione, un gruppo di computer interconnessi che garantiscono la simulazione parallela e distribuita.

CAPITOLO 1 : MODELLI DI MOBILITÀ

Per simulare un nuovo protocollo per una rete ad hoc, è necessario usare un modello di mobilità che rappresenta accuratamente i nodi mobili che eventualmente utilizzeranno il protocollo dato [1]. Un modello di mobilità, infatti, rappresenta il movimento di utenti all'interno di un sistema, e come cambiano nel tempo la loro posizione, velocità e accelerazione [2]. Un modello di mobilità dovrebbe tentare di imitare i movimenti di nodi mobili reali. Solo con la scelta del corretto modello di mobilità sarà possibile determinare se il protocollo proposto sarà utile quando verrà implementato.
In letteratura abbiamo due gruppi principali di modelli di mobilità: i modelli di mobilità a entità, che rappresentano nodi mobili che si muovono indipendentemente gli uni dagli altri, e i modelli di mobilità a gruppi, dove il movimento di un nodo è dipendente da quello dei nodi vicini.

I tipici modelli di mobilità a entità sono:

- Random Walk
- Random Waypoint
- Random Direction
- Boundless Area Simulation
- Gauss Markov
- City Section

I modelli di mobilità a gruppo, invece, sono:

- RPGM (Reference Point Group Mobility Model)
- modello di mobilità a colonna (Column Mobility Model)
- modello di mobilità della comunità nomade (Nomadic Community Mobility Model)
- modello di mobilità a inseguimento (Pursue Mobility Model)

Più nello specifico, questi modelli, possono essere divisi in varie classi in base alle loro caratteristiche di mobilità.

In alcuni modelli il movimento di un nodo è condizionato dal suo movimento precedente. In questo caso si parla di modelli di mobilità con dipendenza temporale, e vengono classificati in questa categoria il modello Gauss Markov e il Boundless Simulation area.
In altri scenari di mobilità, i nodi tendono a viaggiare in modo correlato, e chiamiamo questa categoria modelli di mobilità con dipendenza spaziale. Appartengono a questo gruppo gli RPGM (Reference Point Group Model) e tutti gli altri modelli a gruppo. Un'altra classe è quella dei modelli di mobilità con restrizioni geografiche, come il City Section, dove il movimento dei nodi è limitato da strade, autostrade o ostacoli.
L'ultima categoria è quella dei modelli di mobilità casuali, ai quali appartengono il Random Walk, Random Direction e Random Waypoint, dove i nodi si muovono all'interno dello spazio, liberamente e senza restrizioni [2].

1.1 Modelli di mobilità casuali

In questi modelli di mobilità, i nodi si muovono indipendentemente gli uni dagli altri. Viene selezionata casualmente una destinazione, e i nodi la raggiungono utilizzando una velocità e una direzione, anch'esse casuali e indipendenti dagli altri nodi [2].

1.1.1 Random Walk

Il Random Walk [1] fu creato per descrivere il movimento irregolare ed estremamente imprevedibile che possono avere diverse entità in natura. In questo modello un nodo si muove dalla sua posizione attuale a una nuova locazione scegliendo casualmente una direzione e una velocità a cui viaggiare. Le velocità e le direzioni dei nodi sono scelte in un intervallo predefinito, rispettivamente [velocitàMin, velocitàMax] e [0, 2π]. Ogni movimento avviene durante un intervallo di tempo t, o una distanza costante viaggiata d, al termine del quale ogni direzione e velocità viene ricalcolata.

Se un nodo durante il suo movimento raggiunge un bordo della simulazione, rimbalza con un angolo determinato dalla direzione in arrivo. Per tutta la simulazione i nodi si muovono in questo modo.
In questo modello, i nodi si muovono sempre attorno al punto di partenza con una probabilità di 1, che assicura che nessuno si allontani.
I problemi relativi a questo modello sono principalmente due: innanzitutto non vengono conservate informazioni su dove è stato e qual'era la sua velocità, poiché l'attuale velocità e direzione sono indipendenti da quelle precedenti. Inoltre se si scelgono parametri piccoli (una piccola frazione di tempo, o pochi passi prima di cambiare direzione), il movimento avverrà solo in una porzione dell'area di simulazione.

1.1.2 Random Waypoint

Il modello di mobilità Random Waypoint [1] aggiunge delle pause di tempo tra il cambio di direzione e velocità. Un nodo sceglie casualmente un punto di partenza e rimane fermo in questo punto per un certo periodo di tempo. Dopodichè il nodo sceglie una destinazione casuale all'interno dell'area di simulazione e una velocità distribuita uniformemente nell'intervallo [velocitàMin, velocitàMax] e viaggia fino alla nuova destinazione scelta alla velocità selezionata. Dopo l'arrivo, si ferma per una frazione di tempo, poi riprende il processo.
Il movimento dei nodi nel Random Waypoint, è come il movimento nel Random Walk, se la pausa di tempo fosse uguale a zero e gli intervalli delle velocità fossero le stesse.

1.1.3 Random Direction

Quando il modello del Random Waypoint veniva applicato a un piano cartesiano, inizialmente i nodi erano distribuiti casualmente all'interno dell'area di simulazione e muovendosi si avvicinavano tra loro.
Questo causava un problema iniziale: ogni nodo, era caratterizzato da un continuo cambiamento del numero e della tipologia dei nodi vicini, facendo

rilevare dati sempre più disomogenei. Quindi fino alla stabilizzazione del sistema i nodi continuavano a muoversi, causando raggruppamenti in diverse parti dell'area di simulazione.

Il modello del Random Direction venne creato per superare questi cambiamenti di densità iniziale dei nodi. In questo modello, infatti, un nodo sceglie una direzione casuale verso la quale viaggiare, e continua a muoversi in quella direzione, finché non raggiunge un bordo dell'area di simulazione [1].

Quando il bordo è stato raggiunto, il nodo si ferma per un periodo di tempo, sceglie un altro angolo di direzione (tra 0 e 180 gradi) e continua il processo. In questo modo, i nodi saranno sempre distribuiti equamente all'interno dell'area.

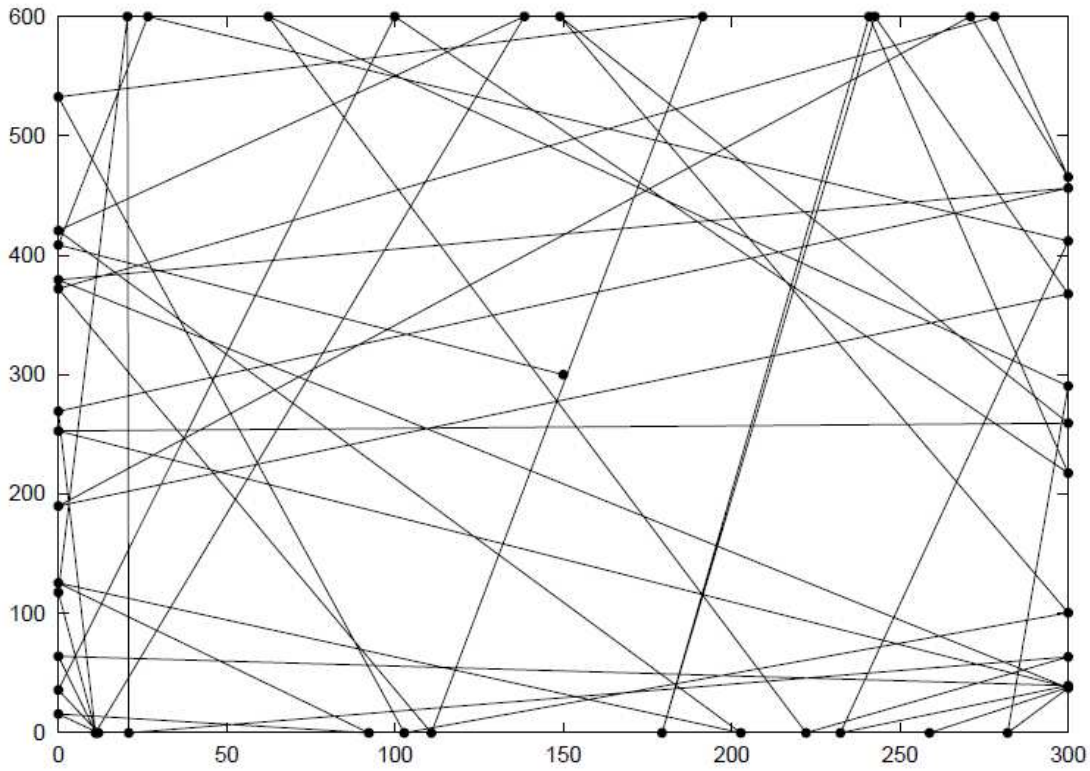


Figura 1: Movimento dei nodi nel modello di mobilità Random Direction

1.2 Modelli di mobilità con dipendenza temporale

La mobilità di un nodo, può essere forzata e limitata dalle leggi fisiche dell'accelerazione, della velocità e dal tasso di variazione della direzione.
La velocità corrente di un nodo mobile, può dipendere dalla sua velocità precedente, perciò le velocità di un singolo nodo mobile, a spazi di tempo differenti, sono correlate [2].

1.2.1 Boundless Simulation Area

Nel Boundless Simulation Area [1] c'è un collegamento tra le precedenti direzioni e velocità del nodo, e le nuove direzioni e velocità. Un vettore di velocità $V = (v, \Theta)$ è usato per descrivere la velocità V di un nodo mobile e la sua direzione Θ; la posizione del nodo mobile è rappresentata dalle coordinate (x,y).
Sia la velocità che la posizione sono aggiornate ad ogni step di tempo dt, in base alle seguenti formule:

Nuova velocità:
$v(t+dt) = min\ (max[v(t)+dv, 0], V_{max})$

Nuova direzione:
$\Theta(t+dt) = \Theta(t)+d\Theta$

Nuova posizione:
$x(t+dt) = x(t) + v(t) * cos\Theta\ (t)$
$y(t+dt) = y(t) + v(t) * sen\Theta\ (t)$

dove V_{max} è la massima velocità di simulazione, dv è il cambio di velocità, che è uniformemente distribuito nell'intervallo $[-A_{max}*dt, A_{max}*dt]$, A_{max} è la massima accelerazione di un dato nodo, dΘ è il cambio di direzione che è uniformemente distribuito in $[-\alpha*dt, \alpha*dt]$, e α è il massimo cambiamento angolare nella direzione, quando il nodo sta viaggiando.

In questo modello di mobilità, non ci sono destinazioni da raggiungere, ma il nodo continua a viaggiare all'interno dell'area di simulazione, e se questa ha dei bordi, una volta raggiunto un lato dell'area, il nodo riappare sul lato opposto per poi continuare il suo percorso.

1.2.2 Modello di mobilità Gauss-Markov

Nel modello di mobilità Gauss-Markov [1] inizialmente ad ogni nodo è assegnata una velocità e una direzione. A intervalli di tempo *n* fissati, i movimenti dei nodi avvengono aggiornando la loro velocità e direzione. In modo specifico, il valore della direzione e della velocità all'istanza *n* è calcolata basandosi sul valore di direzione e velocità all'istanza *(n-1)* e con una variabile casuale, usando le seguenti equazioni:

$$s_n = \alpha s_n\text{-}1 + (1-\alpha)\overline{s} + \sqrt{(1-\alpha^2)}\, s_{x(n-1)}$$

$$d_n = \alpha d_n\text{-}1 + (1-\alpha)\overline{d} + \sqrt{(1-\alpha^2)}\, d_{x(n-1)}$$

dove s_n e d_n sono la nuova velocità e nuova direzione del nodo all'intervallo di tempo n; α, compreso tra 0 e 1, è il parametro usato per variare la casualità; $\overline{s}$ e $\overline{d}$ sono costanti che rappresentano il valore medio di velocità e direzione per $n\rightarrow\infty$; infine $s_{x(n-1)}$ e $d_{x(n-1)}$ sono variabili casuali prese da una distribuzione gaussiana (generalmente con media 0 e deviazione standard 1). Valori completamente casuali (che permettono di ottenere un movimento browniano, ossia come quello delle particelle che si muovono in un liquido), sono ottenuti settando $\alpha = 0$, mentre il movimento lineare (come nel caso del Random Waypoint), è ottenuto con $\alpha = 1$. Tutti i livelli intermedi di casualità, sono ottenuti variando il valore di α tra 0 e 1.
Ad ogni intervallo di tempo la prossima posizione del nodo, è calcolata basandosi sulla posizione corrente, velocità e direzione di movimento. Quindi ad ogni intervallo n, la posizione è data dall'equazione:

$x_n = x_{n-1} + s_{n-1} * cos\ d_{n-1}$
$y_n = y_{n-1} + s_{n-1} * sin\ d_{n-1}$

dove (x_n , y_n) e (x_{n-1} , y_{n-1}) sono rispettivamente le coordinate x e y della posizione del nodo all'intervallo di tempo *n* e *n-1*.
Per assicurarsi che un nodo non rimanga vicino a un bordo dell'area per un lungo periodo di tempo, i nodi sono forzati ad allontanarsi. Questo viene fatto modificando la variabile della direzione media $\bar{d}$ nell'equazione della direzione.
Questo avviene, ovviamente, se ci si trova all'interno di un'area delimitata da bordi. Nel caso del progetto in esame, dove viene utilizzato un ambiente toroidale bidimensionale, quindi senza bordi a cui avvicinarsi, la direzione media rimane costante.

1.3 Modelli di mobilità con vincoli geografici

Nel Random Waypoint i nodi possono muoversi liberamente nell'area di simulazione. Tuttavia, nelle applicazioni della vita reale, il movimento del nodo è spesso soggetto all'ambiente nel quale si trova. In particolare, il movimento dei veicoli è limitato alle autostrade o alle strade urbane, e i pedoni, camminando, possono essere bloccati dagli edifici che si trovano davanti.
Quindi i nodi si possono muovere in modo pseudo-casuale su percorsi predefiniti nell'area di simulazione. Così sono stati creati modelli di mobilità con percorsi indotti o ostacoli [2].

1.3.1 City Section Mobility Model

Nel modello di mobilità City Section [4], l'area di simulazione è una rete stradale che rappresenta la sezione di una città. Le strade e i limiti di velocità sulle strade sono basati sul tipo di città che si vuole simulare.
Questo modello, costringe i nodi a muoversi su questa griglia di strade, dove tutte le strade sono a una corsia e bidirezionali. I veicoli (nodi)

selezionano casualmente una posizione iniziale rappresentata da un qualsiasi punto su una strada. Successivamente scelgono una delle intersezioni tra le strade come loro destinazione e si muovono verso di essa con velocità costante, con almeno un movimento orizzontale o verticale.
La velocità dipende dalla strada sulla quale si trova il nodo, e possiamo dividerle in due tipi di strade in particolare: strade ad alta e a bassa velocità. Ogni nodo, quindi, imposta la sua velocità in base alla strada sulla quale sta viaggiando. Le iterazioni tra i nodi lungo le strade sono ignorate, poiché tutti viaggiano alla stessa velocità (quindi non si possono sorpassare), e ai nodi è permesso sovrapporsi negli incroci tra strade.
Non sono previste pause ne alle intersezioni, ne al raggiungimento della destinazione.

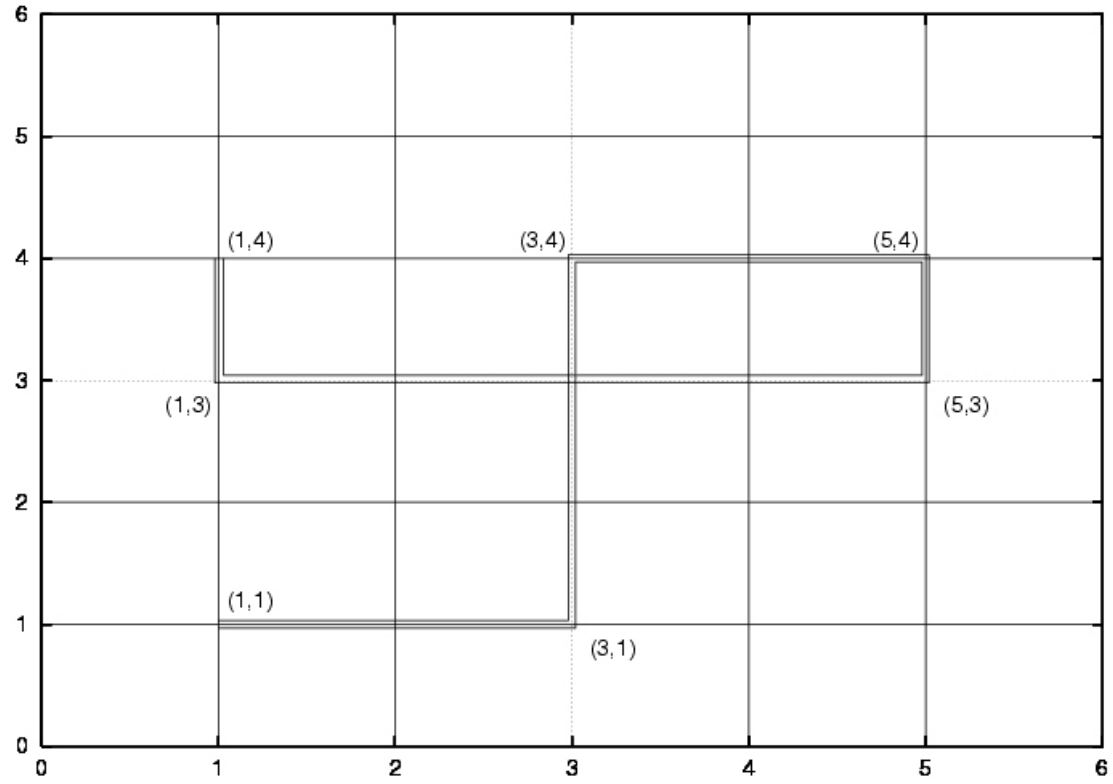


Figura 2: Esempio di mobilità usando il City Section. Il nodo si muove sulla griglia, partendo dal punto (1, 4) e scegliendo varie destinazioni, fino ad arrivare a (1, 1);

1.4 Modelli di mobilità con dipendenza spaziale

In questa classe di modelli di mobilità, possiamo far rientrare tutti i modelli di mobilità a gruppi. In questi modelli troviamo un insieme di nodi, le cui azioni sono correlate agli altri nodi presenti nel gruppo. In una rete ad hoc, infatti, ci sono molte situazioni dove è necessario modellare il comportamento di nodi che si muovono insieme, come ad esempio nel caso di un gruppo di soldati ai quali è richiesto di lavorare insieme per raggiungere un obiettivo [1]. Ci sono diversi modelli di mobilità a gruppo, ed il più generico è il Reference Point Group (RPGM), dal quale sono poi implementati tutti gli altri.

1.4.1 Reference Point Group Mobility Model (RPGM)

Il modello RPGM rappresenta sia il movimento casuale di un gruppo di nodi mobili sia il movimento casuale di ogni nodo individuale dentro al gruppo.
In questo modello ogni gruppo ha un centro, che in genere è un centro logico o un nodo leader del gruppo. Per semplicità assumiamo che il centro del gruppo, sia un nodo leader. Quindi ogni gruppo è composto da un leader e da un numero di membri e il movimento del leader determina il comportamento dell'intero gruppo. Il leader e gli altri nodi, hanno funzioni differenti: il leader si muove utilizzando un vettore di movimento V, che non determina solo il proprio spostamento, ma anche quello di tutto il gruppo, mentre gli altri membri si muovono sia con tutto il gruppo seguendo il leader, sia attorno a un proprio punto di riferimento.
Quando i punti di riferimento individuali si muovono da t a t+1, la loro posizione è aggiornata in modo che restino sempre vicino al loro leader. Una volta che i punti di riferimento vengono aggiornati alla nuova posizione, questi vengono combinati con un vettore casuale, RM, che rappresenta come il nodo si muoverà attorno al proprio punto di riferimento. Sia il movimento del leader di ogni gruppo, sia il movimento casuale di ogni nodo all'interno del gruppo, sono implementati usando

l’algoritmo del Random Waypoint. La differenza, però, è che i nodi che si muovono individualmente, non usano pause mentre il gruppo si muove. Le pause vengono usate solo quando il punto di riferimento del gruppo raggiunge una destinazione, allora tutti i nodi del gruppo si fermano per un periodo di tempo. Questo modello fu creato inizialmente per rappresentare scenari di soccorso in caso di valanga, dove viene utilizzato un team composto da risorse umane e canine, che lavorano insieme. Le guide creano un percorso da seguire per i cani, e i cani creano il loro percorso casuale attorno all’area scelta dalle guide, nella ricerca di eventuali vittime [1].

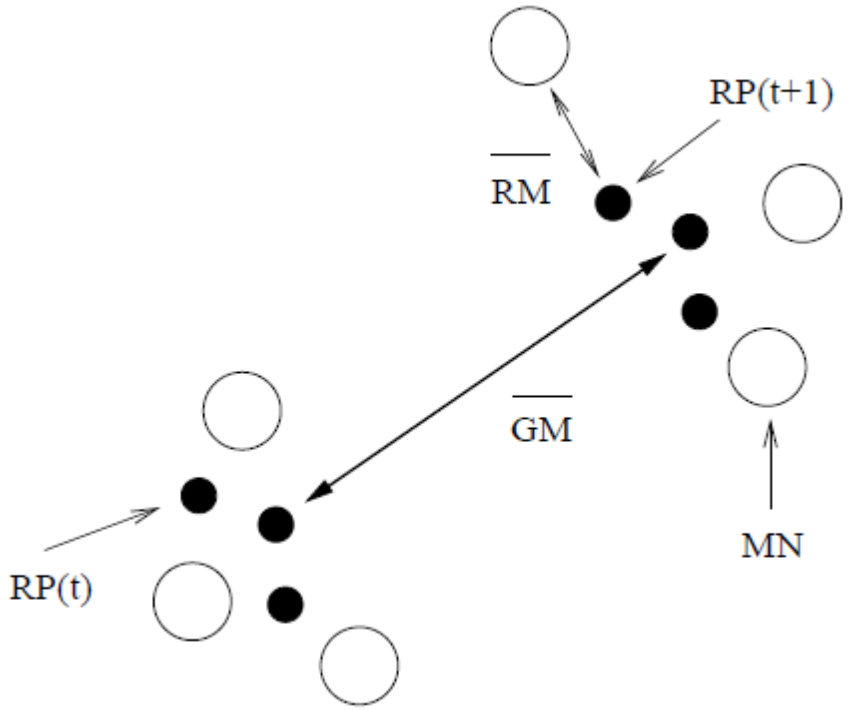


Figura 3: Disposizione dei nodi nel modello di mobilità RPGM. GM è il vettore che sposta il leader e i punti di riferimento (RP), RM è il vettore che permette ai nodi di muoversi attorno al punto di riferimento (pallino nero).

CAPITOLO 2: STRUTTURA DEL SISTEMA

Il sistema sul quale si lavora è composto da un insieme di entità simulate (Simulated Model Entities, SME) e ognuna di esse rappresenta un componente del sistema (nel caso specifico si esaminano dei dispositivi wireless).
Nella simulazione parallela e distribuita un insieme di unità di esecuzione si occupano della simulazione, e queste unità sono rappresentate da Logical Process (LP), ognuna delle quali è responsabile di una parte della simulazione.
Una o più SME sono contenute da vari LP, che sono allocati a loro volta su una unità fisica d'esecuzione (Physical Execution Unit, PEU), che fornisce ad ogni LP le risorse computazionali e di comunicazione [3]. Gli SME appartenenti ad un LP possono comunicare con tutti gli SME del sistema, anche se questi sono allocati su un LP diverso.
Per ottenere una corretta esecuzione della simulazione distribuita, tutte le unità di esecuzione devono essere sincronizzate durante l'intero periodo di tempo della simulazione. L'algoritmo utilizzato per la sincronizzazione, in questo caso, è basato sul time-stepped. Questo comporta che la simulazione sia divisa in frazioni di tempo tutte della stessa lunghezza, dette step o passi. Il simulatore non avanza al prossimo step di tempo finchè tutte le attività di simulazione associate al passo corrente non siano state completate.
Ogni entità gestirà l'evoluzione di una piccola parte del sistema e interagirà con le altre entità usando dei messaggi. In particolare, gli SME sullo stesso LP potranno interagire tra loro con bassa latenza e bassi costi. In questo caso parliamo di comunicazione intra-LP.
Per quanto riguarda la comunicazione inter-LP, cioè tra SME che si trovano su LP diversi, è di primaria importanza considerare le caratteristiche dell'allocazione fisica dei differenti processi ed aggiustare la comunicazione in base alle prestazioni della rete. Ad esempio, gli LP su sistema multi-core (che dispone cioè di più processori) dovrebbero comunicare tramite memoria condivisa, mentre LP connessi tramite LAN,

WAN o internet, dovrebbero invece utilizzare appropriati protocolli di comunicazione (ad esempio TCP).

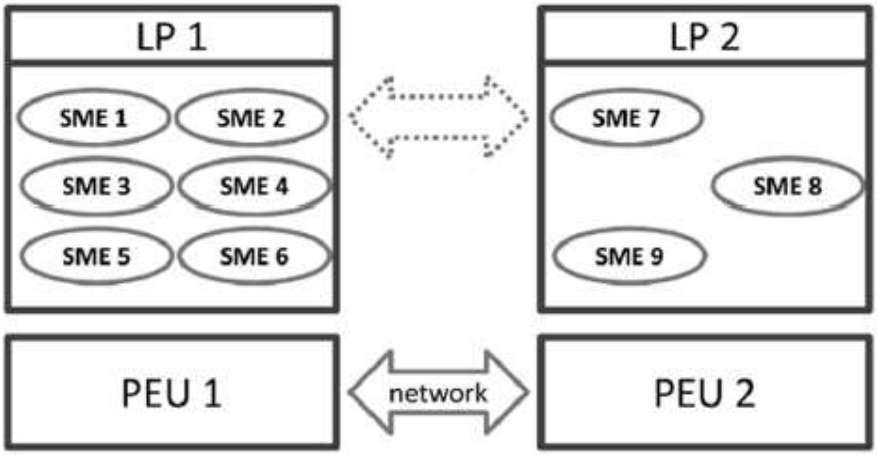


Figura 4: una simulazione distribuita composta da 2 LP

2.1 Riduzione del sovraccarico

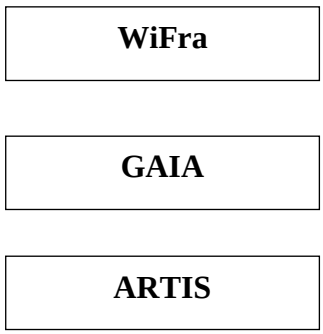
Per ridurre i costi di comunicazione, si controlla il comportamento di ogni SME, in modo che quelli che interagiscono maggiormente tra loro, vengano spostati e riallocati sullo stesso LP. In questo modo si riducono i costi della comunicazione inter-LP.
L'algoritmo analizza la comunicazione di ogni SME e determina se deve migrare. Alla fine di ogni time-step di simulazione, durante la fase di sincronizzazione, è analizzato il comportamento di ogni SME che ha fatto almeno un'iterazione durante la comunicazione. Un SME che ha la maggior parte di iterazioni consegnate a SME che si trovano all'esterno del suo LP, è un buon candidato per la migrazione. Il destinatario della migrazione, è l'LP che ha avuto la maggior percentuale di interazioni in uscita [3].
La migrazione viene implementata come il trasferimento delle strutture dati di ogni SME.

2.2 Bilanciamento del sistema

Il bilanciamento è basato sulla migrazione delle entità simulate e dalla presenza di barriere di sincronizzazione durante l'esecuzione della simulazione, rappresentate dal meccanismo del time-step.
Focalizzandosi sulla sincronizzazione, una simulazione basata sul time-step può essere vista come una serie di punti consecutivi di sincronizzazione, dove ad ogni LP è permesso di passare dallo step di tempo *n* allo step *n+1*, solo se tutti gli LP hanno completato le proprie operazioni relative allo step *n*. Uno svantaggio di questo sistema, è che tutti gli LP dovranno aspettare che l'LP più lento abbia terminato le sue operazioni. La lentezza di un LP può essere dovuta a due fattori principali: l'unità di elaborazione è sovraccarica, oppure la rete di comunicazione usata dall'LP ha un forte ritardo rispetto agli altri componenti della simulazione.
In entrambi i casi, la soluzione è data dal trasferimento di alcuni SME dell'LP più lento, all'LP più veloce. Per determinare quali sono gli LP più lenti, ogni LP durante la simulazione colleziona dati derivanti dall'algoritmo di sincronizzazione e, arrivato alla barriera di sincronizzazione, li confronta con gli altri LP. Successivamente ogni LP calcola una sorta di sua posizione basata sulla velocità di esecuzione degli altri LP. Gli LP che si trovano in cima sono segnalati come "veloci", mentre quelli in fondo sono i "lenti". Un LP marcato come "lento", può migrare alcuni SME locali all'LP marcato come "veloce". Ogni LP "lento" deve determinare quanti e quali SME devono migrare e l'LP che dovrà riceverli [3].
Quindi, il meccanismo di bilanciamento innesca migrazioni aggiuntive per migliorare il bilanciamento del sistema distribuito.

CAPITOLO 3: STRUTTURA DEL SIMULATORE

Il simulatore utilizzato per lo svolgimento del progetto è costituito da un’architettura sviluppata su più livelli.
In cima si trova il Wireless Framework (WiFra), una libreria composta da modelli e funzioni ausiliarie, progettata e implementata per semplificare le valutazioni di performance della simulazione basata su reti wireless a larga scala. Nel livello al centro c'è GAIA, un altro framework che fornisce le funzionalità per ridurre il sovraccarico di comunicazione e per gestire il bilanciamento nella simulazione distribuita. Il nucleo del simulatore è ARTIS (Advanced RTI System), un middleware che si occupa di fornire un ambiente efficiente e facile all’uso per la simulazione parallela e distribuita.



3.1 ARTIS (Advanced RTI System)

ARTIS è un middleware, termine che indica un’entità (programma, protocollo o altro) che si interpone principalmente tra l’hardware della macchina e il software applicativo [5]. È disegnato specificatamente per supportare un alto grado di scalabilità, ovvero non solo è possibile aggiungere ulteriori funzionalità senza doverne modificare le caratteristiche fondamentali, ma permette la sua esecuzione su computer di potenza diversa.
La scalabilita' e' un fattore critico per le applicazioni distribuite e indica

anche la capacità di adattarsi all'aumento di utenti, all'incremento dei dati e alla diversificazione delle funzionalita' richieste [10]. Artis infatti è disegnato per architetture di esecuzione composte da un gran numero di unità di elaborazione (PEU).
Dato che nella simulazione distribuita una gran parte delle iterazioni sono consegnate tramite la comunicazione di rete, la velocità di esecuzione è influenzata dalle performance della rete. Quindi è di primaria importanza considerare le caratteristiche dell'allocazione fisica di ogni LP, ed aggiustare la comunicazione in base alle performance della rete. Gli LP che appartengono allo stesso PEU (Physical Execution Unit) dovrebbero comunicare con memoria condivisa, mentre quelli su PEU diversi, che utilizzano la LAN, WAN o internet, dovrebbero basarsi sul protocollo appropriato.
ARTIS, come meccanismo di comunicazione, utilizza la memoria condivisa dove possibile, e in tutti gli altri casi TCP/IP.
Come detto nei paragrafi precedenti, la sincronizzazione è uno dei problemi principali della simulazione parallela e distribuita, perciò questo middleware deve fornire i servizi per la gestione del tempo ai componenti della simulazione.
ARTIS mette a disposizione librerie per la sincronizzazione e gestione dei messaggi. I due algoritmi utilizzabili tramite le API sono il Chandy-Misra-Bryant o il time-stepped [1].

3.1.1 SIMA (SImulation MAnager)

L'architettura dell'implementazione risulta parzialmente centralizzata per la presenza di un SImulation MAnager [9] che ricopre vari compiti tra i quali la coordinazione degli LP, l'inizializzazione e terminazione della simulazione e la gestione di barriere di sincronizzazione durante l'esecuzione. Il SIMA è a conoscenza del numero totale di LP e della natura dei canali di comunicazione che li collegano. Esso rileva la disposizione topologica dei PEU ed indica ad ognuno di essi come instaurare il dialogo con gli altri. In pratica, è necessario eseguire prima

della simulazione vera e propria una fase di inizializzazione dei canali di trasmissione, così da predisporre ogni LP alla comunicazione con gli altri nel modo più efficiente possibile.
Le primitive messe a disposizione per utilizzare il SIMA sono le seguenti:

`void SIMA_Initialize(int port, int numpeers, char *netfile);` Questa funzione viene invocata per indicare al SIMA quanti siano gli LP partecipanti alla simulazione, su quale porta avverranno le comunicazioni di inizializzazione e quale sia il file da leggere relativo ai canali di comunicazione. La funzione non ritorna finchè non sono stati contattati tutti gli LP;

`void SIMA_Finalize( );` Chiude i socket di comunicazione tra SIMA e LP. Alla chiamata di questa funzione, la fase di inizializzazione è terminata, tutti gli LP sono in possesso delle informazioni che permettono loro di instaurare connessioni con gli altri LP e le risorse allocate in fase di inizializzazione possono essere rilasciate;

`void SIMA_Barrier( );` Istituisce un punto di sincronizzazione durante l'esecuzione. Questa riprenderà solamente quando tutti gli LP si saranno sincronizzati con il SIMA.

Il codice del SIMA, quindi, dovrà avere una struttura di questo tipo:

```
SIMA_Initialize(porta, lps,channels.txt);
...
SIMA_Finalize( );
```

Dopodichè potrà iniziare il programma vero e proprio.
Il file di configurazione "channels.txt" indica al runtime quali siano i valori di lookahead su ogni singolo canale, oppure specifica un valore globale e comune a tutti gli LP. Il lookahead, nel caso l'algoritmo sia quello del time

step, indica l'intervallo di tempo che alle diverse entità coinvolte occorre a compiere una o più azioni.
Nel caso specifico del progetto tale misura coincide con uno step di computazione.
Il file dovrebbe essere composto in due parti: la prima, per definire i canali di comunicazione ed i loro nomi, e la seconda, per specificare un valore di lookahead. Quest'ultimo potrà essere di tipo globale, cioè comune a tutti gli LP, oppure distinto per ognuno di essi (solo nel caso Chandy-Misra-Bryant) , come nell'esempio seguente:

```
# DEFINIZIONE DEI CANALI:
:MILANO 0
:BOLOGNA 1
:ROMA 2
#DEFINIZIONE DEI LOOKAHEAD
GLOBAL LA=0 # Look-Ahead Globale Disabilitato
MILANO: BOLOGNA 5 ROMA 7
ROMA: MILANO 7 BOLOGNA 3
BOLOGNA: MILANO 5 ROMA 3
```

Nella prima parte vengono associati ID numerici alle etichette che identificano gli LP, quindi si procede con la specifica del lookahead per ognuno di essi se il modello lo prevede, oppure con l'inserimento di un valore positivo nel campo GLOBAL_LA. Esso segnala che il lookahead sarà lo stesso per tutti gli LP e lo quantifica (nell'esempio non viene usato lookahead globale).
In questo caso, ad esempio, Milano comunicherà con Bologna in 5 unità di tempo, e con Roma in 7. Se fosse stato GLOBAL_LA = 2, non sarebbe stato necessario definire i singoli lookahead, poiché tutti avrebbero comunicato tra loro con 2 unità di tempo.

3.2 GAIA (Generic Adaptive Interaction Architecture)

GAIA è un framework basato sulla migrazione, ossia è un insieme di classi ed interfacce di base, che costituiscono l'infrastruttura di una applicazione e permette la migrazione di entità da un processo all'altro [5].
Le funzioni di GAIA sono quelle di controllare il comportamento di ogni entità simulata durante tutto il tempo di simulazione, fornire le iterazioni col nucleo della simulazione e i servizi di comunicazione, e gestire la distribuzione e localizzazione dei dati. Lo scopo di GAIA è quello di fornire la migrazione e mettere a disposizione dell'utente un insieme di API, per garantire un alto livello di astrazione [1].
Una serie di euristiche, valutano il comportamento della comunicazione, e innescano la riallocazione di entità per ridurre il costo di comunicazione e migliorare l'equilibrio nell'architettura d'esecuzione.
Per mantenere le informazioni sullo stato delle SME (Simulated Mobile Entities), nei modelli vengono aggiunte al codice le strutture dati relative ai nodi. GAIA migra le strutture dati degli SME, cioè le informazioni sul loro stato, tra gli LP [8].
È tramite questo framework che vengono implementati, infatti, la riduzione del sovraccarico e il meccanismo di bilanciamento descritti sopra
Per quando riguarda il bilanciamento, c'è la primitiva:

```
Void GAIA_SetLoadBalancing (int state);
```

dove tramite "state" si seleziona se il bilanciamento è attivato (ON) o disattivato (OFF), e nel primo caso, il meccanismo di GAIA reagisce automaticamente all'hardware eterogeneo e rialloca dinamicamente gli SME dall'LP più "lento" a quello più "veloce" [6].

Per la migrazione, invece, è necessario chiamare la funzione:

```
Void GAIA_Migrate(int id, String data, int size);
```

Questa viene utilizzata da un LP locale per migrare un SME ad un LP remoto. L'LP locale non è libero di scegliere quale SME deve essere migrato. Durante la simulazione, il meccanismo di migrazione definisce quali SME devono migrare e informa l'LP usando un messaggio di migrazione (`NOTIF_MIGR`) [6].
Ovviamente, GAIA, fornisce primitive per lo scambio di messaggi, con la

```
GAIA_Send();
```

per inviare messaggi, e per riceverli:

```
GAIA_Receive(max_len);
```

Per inizializzare GAIA, ogni LP che vuole entrare nella simulazione, deve chiamare

```
Int GAIA_Initialize();
```

mentre per uscirne, allo stesso modo, l'LP chiama

```
void GAIA_Finalize();
```

Il meccanismo di GAIA è basato sull'algoritmo del time-step, quindi ogni LP deve attendere che lo step di tempo corrente sia terminato, prima di procedere al prossimo step di tempo.

CAPITOLO 4: STRUTTURA INIZIALE DEL PROGRAMMA

Inizialmente il progetto era composto da 2 cartelle, contenenti i sorgenti principali.

Nella cartella INCLUDE, ci sono i sorgenti .h, in particolare tutte le API utilizzate nel programma: le API di GAIA (`gaia.h`), le API relative al SIMA (`RTIComm.h`), le API per utilizzare e manipolare un file con estensione .ini (`ini.c`), le API per i generatori di numeri casuali (`rnd.h`), un file con le strutture dati per gestire gli LP (`LPinfo.h`), e varie API per poter utilizzare algoritmi di sincronizzazione come il Three Barrier, Time Warp, Time Stepped o Chandry Misra Briant. Tra questi ovviamente, viene utilizzato solo il Time Stepped (`ts.h`) che viene incluso in `gaia.h`, poiché GAIA utilizza la sincronizzazione basata sul Time Step.

Nella cartella /MIGRATION-WIRELESS, invece, troviamo i sorgenti veri e propri del programma: `wireless.c`, `sima.c` e l'eseguibile `run`, che una volta compilati i sorgenti tramite un Makefile, permetterà di far partire l'esecuzione della simulazione.

4.1 `Makefile` e partenza della simulazione

Nella cartella troviamo un `Makefile`, un file apposito che collega tra loro e crea i file che eseguiranno la simulazione.
È necessario aprire il terminale, posizionarsi all'interno della cartella e digitare `make`. In questo modo il Makefile genera gli eseguibili `./sima` e `./wireless`, dai file `wireless.c` e `sima.c`.
Successivamente per eseguire la simulazione bisogna digitare `./run NLP NLP NSMH`.
Dove NLP sono il numero di LP (processi) e NSMH sono il numero di Simulated Mobile Host (nodi).

Gli SMH vengono divisi in modo uguale tra il numero di LP, quindi se viene digitato ad esempio `./run 2 2 1000`, vengono creati 2 LP con 500 nodi l'uno.

4.2 `run`

Questo file esegue uno o più run della simulazione della rete wireless. In questo caso è impostato per eseguire una simulazione.
Il suo compito è quello di leggere i 3 parametri di input:

```
NLP=$1
SLP=$2
TOT_SMH=$3
```

I primi due indicano entrambi il numero di LP nel sistema, e il terzo è il numero totale di nodi nel sistema.
Successivamente suddivide il numero di nodi tra gli LP (`SMH=TOT_SMH/NLP`), e chiama l'eseguibile `./sima` (creato precedentemente col Makefile) con il parametro `NLP`, in modo che il SIMA sappia quanti sono gli LP partecipanti.
Terminata l'esecuzione del sima, ogni LP nel sistema chiama l'eseguibile `./wireless` con i parametri `NLP` e `SMH`.
In questo modo ogni LP viene inizializzato col proprio numero di nodi e gli viene associato un processo che lo gestirà per l'intera durata della simulazione.

4.3 `sima.c`

Come detto nel Capitolo 3, per poter utilizzare ARTIS è necessario chiamare per primo il SIMA (SImulation MAnager).
Questo file inizializza il SIMA con la funzione:

```
SIMA_Initialize(5000, atoi(argv[1]), ”channels.txt");
```

la porta alla quale deve collegarsi è la 5000, il numero di LP nella simulazione vengono dati come parametro, quindi non fa altro che leggerli da input (gli vengono passati dall'eseguibile `./run`), e il file di configurazione per la rete è il file "channels.txt".
A questo punto la simulazione è pronta per partire, poiché gli LP conoscono qual è il valore del lookahead.
In questo caso, nel file di input, c'è un lookahead globale fissato a 1.

4.4 `wireless`

Questo è il file che esegue la simulazione vera e propria, viene chiamato da ogni LP per gestire i nodi e inviare messaggi tra di essi.

4.4.1 *`wireless.ini`*

I file col formato .ini sono degli standard per configurare le piattaforme o i software. L'estensione .ini deriva dalla parola "inizializzazione" poiché questi file contengono le proprietà che serviranno al sorgente .c.
In questo caso, vengono inizializzati i valori delle variabili contenute in wireless.c.
Viene settata la grandezza dell'area di simulazione tramite:

```
[MODEL]
; Simulated area
MAX_X=10000.0
MAX_Y=10000.0
```

L'area ha un formato toroidale 2D, quindi le limitazioni dell'area sono solamente per indicarne la grandezza, poichè un nodo che arriverà alla coordinata X pari a 10000, non si fermerà, ma riprenderà dalla coordinata X=0.

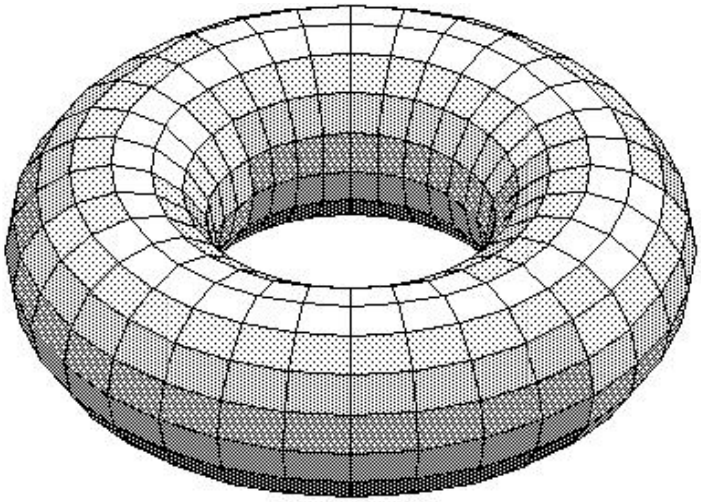


Figura 5: uno spazio toroidale 2D: i nodi si muovono sulla parte esterna dell'anello

Il file imposta anche la velocità che ogni nodo dovrà utilizzare e il raggio entro il quale un nodo può mandare i messaggi:

```
; Communication range
RADIUS=250.0
SPEED=10
```

Infine vengono inizializzate le informazioni che serviranno anche al SIMA per iniziare la simulazione:

```
; Simulation Manager
[SIMA]
HOST=localhost
PORT=5000
```

4.4.2 `wireless.c`

All'interno di questo file è implementato il main, e vengono definite tutte le strutture dati utilizzate per memorizzare gli SMH (Simulated Mobile Host), per scambiare messaggi tra loro e per effettuare la migrazione.
Ad ogni LP viene associato un ID, e in questo programma, è il primo LP (quello con ID=0) che gestirà le statistiche, ossia nel suo file di output, saranno indicati: il tempo usato in questo step, il numero dello step in cui ci troviamo, il numero di SHM in questo LP, le entità migrate da questo LP,

la percentuale di comunicazioni locali e remote, e il totale di migrazioni tra tutti gli LP in questo step di tempo.
Gli altri LP, invece, avranno nel file di output solo le informazioni essenziali, ovvero il tempo usato in questo step, il numero del time-step, il numero di host nell'LP e le entità migrate dall'LP durante questo step di tempo.

Poiché la simulazione non è infinita, ma dura un determinato periodo di tempo, viene definita la sua durata, settando la variabile `END_CLOCK=1000`, cioè la simulazione durerà 1000 step di tempo.
Lo step di tempo in cui si trova la simulazione viene salvato nella variabile `CLOCK` con la chiamata alla funzione:

```
GAIA_TimeAdvance();
```

che restituisce il prossimo step di tempo. Se il prossimo step è maggiore o uguale al valore di `END_CLOCK`, la simulazione termina, chiudendo tutte le comunicazioni e terminando GAIA, con la funzione: `GAIA_Finalize()`.

Il Wall Clock Time (WCT, letteralmente "tempo che indica l'orologio sulla parete", ovvero il tempo effettivamente trascorso), invece, viene salvato nelle variabili `t1` e `t2`, che indicano il momento in cui è iniziata la simulazione e il momento in cui si trova. Quando la simulazione termina, tramite queste variabili, sarà possibile conoscerne la sua durata.

Per rendere il sistema il più realistico possibile, vi sono delle percentuali che "limitano" il comportamento degli host. In questo caso, la percentuale che l'host si muova nel time-step attuale è del 70%, la probabilità che sia equipaggiato di un dispositivo wireless è del 50%, e la probabilità di inviare un'iterazione a un host nel suo raggio d'azione è del 20%.

Un nodo può mandare 3 tipi di messaggi:

`W`: messaggio wireless: il nodo è dotato di un dispositivo wireless e può comunicare con i nodi attorno a lui;
`P`: il nodo invia un messaggio PING ai nodi vicini.
`M`: il nodo si è mosso verso una nuova posizione, e la sua posizione deve essere aggiornata.

Per la memorizzazione dei nodi nel sistema, si usano due tabelle hash:
`stable`: contiene gli host simulati all'interno di un singolo LP.
`table`: contiene gli host simulati in tutto il sistema.
Sono entrambe inizializzate chiamando la funzione `hash_init,` che crea le tabelle impostandone la dimensione e allocando lo spazio per le strutture dati dei nodi che dovranno ospitare.
`NSIMULATE*NLP` sarà il numero di nodi nella tabella `table`, mentre `NSIMULATE` sarà la dimensione di `stable.`
Il valore di `NSIMULATE` è stato passato come parametro dall'eseguibile `./run`, che l'aveva calcolato con la formula `SMH=TOT_SMH/NLP.`

Per ogni nodo nell'LP, viene chiamato `GAIA_Register(MIGRABLE),` che genera e rende migrabili i nodi della tabella `stable`.
Ovviamente sarà necessario abilitare il meccanismo di migrazione chiamando `GAIA_SetMigration(MIGR_ON)`.

Entrambe le tabelle hash, sono strutturate in questo modo:

```
typedef struct hash_t {
     struct hash_node_t **bucket;
     int count;
     int size;
} hash_t;
```

Dove nel campo bucket, abbiamo una struttura fatta in questo modo:

```
typedef struct hash_node_t {
     struct hash_data_t *data;
     struct hash_node_t *next;
} hash_node_t;
```

in questa struttura c'è puntatore al prossimo elemento e un campo `data`, che contiene tutte le informazioni relative al nodo, come la sua chiave (indispensabile in una tabella hash), l'ID dell'LP in cui si trova, la sua posizione attuale, il suo target da raggiungere, e altre informazioni sul suo stato.

Per effettuare la migrazione, c'è una lista unica, che contiene tutti i nodi che dovranno migrare al termine del time-step corrente: `mlist.`
Ogni elemento della lista ha queste informazioni:

```
typedef struct smh_list {
     struct smh_list_n *head;
     struct msg_type smh_list_n *tail;
     int size;
} smh_list;
```

dove sia `head` che `tail`, sono una struttura con un campo `hash_node_t`, e un puntatore all'elemento successivo, in modo da poter scorrere la lista.

L'algoritmo utilizzato in questo programma per muovere i nodi, è il Random Waypoint. Il nodo viene inizializzato con una posizione random all'interno dell'area di simulazione, dopodiché, sempre in modo casuale, gli viene assegnata una destinazione che deve raggiungere con una certa velocità `SPEED` (definita precedentemente nel file .ini). Prima di muoversi, il nodo controlla se è stata raggiunta la destinazione, e in questo caso ne calcola una nuova. Altrimenti calcola l'angolo necessario per dirigersi verso il target e si sposta. Vengono fatti anche i controlli nel caso la nuova

posizione del nodo sia maggiore di 10000 o minore di 0, e in questo caso viene spostato nella posizione giusta (se ad esempio si trova a 10001, deve riprendere dalla posizione 1, secondo la struttura toroidale dell'area).

Dopo le varie inizializzazioni e definizioni di funzioni, nel `main`, abbiamo un ciclo principale, che si occupa di gestire tutte le richieste.
Come prima operazione, viene fatto:

```
msg_type = GAIA_Receive(&from, &to, &Ts, (void *)data, &max_data);
```

che salva il tipo del messaggio ricevuto da GAIA nella stringa `msg_type`, e tramite uno switch chiama il gestore opportuno:

`STATE_CHANGE`: la topologia della simulazione sta cambiando, poiché un LP sta lasciando la simulazione prima che questa sia terminata. Se si tratta dell'LP che gestisce le statistiche, incremento il contatore degli ID in modo che le statistiche vengano calcolate dall'LP con ID successivo. Se è LP dove ci troviamo che sta lasciando il sistema, segnalo con una variabile che sta lasciando la simulazione.

`REGISTER`: è stata registrata una nuova entità: viene chiamata la funzione `register_event_handler(id, lp)` (dove `id` ed `lp` sono il `from` e `to` della `GAIA_Receive`), che inserisce il nodo nella `table` globale e nella `stable` dell'LP, e modifica il campo `data` del nodo aggiungendo una posizione iniziale e un target. Se il nodo viene dotato di un dispositivo wireless (in base alle percentuali indicate sopra), viene creato un messaggio `W` che viene mandato a tutti gli altri nodi tramite il `GAIA_Broadcast()` per avvertirli dello stato del nuovo dispositivo.

`NOTIF_MIGR`: è avvenuta una notifica di migrazione, significa che il nodo deve migrare al termine dell'attuale time step (solo al termine perché

è possibile che nel time step corrente possa essere destinatario di messaggi in sospeso).
Chiama il gestore `notify_migration_event_handler(from, to)`, che cerca il nodo nella `table` (tramite l'`id`, che era il `from` passato in input) e lo aggiunge alla lista `mlist`. All'interno del campo `data`, modifica l'`id` dell'LP in cui si trovava, aggiornandolo al nuovo valore (campo `to` passato in input), che sarà LP nel quale dovrà migrare.

`NOTIF_MIGR_EXT`: deve avvenire una migrazione all'esterno dell'LP, e questo deve essere messo a conoscenza di questa operazione anche se non è direttamente coinvolto.
Viene chiamato il gestore `notify_ext_migration_event_handler (from, to)` che, come prima, cerca il nodo con l'`id=from` nella `table` globale, e viene cambiato l'`id` dell'LP in cui si trova con il `to` passato in input.

`EXEC_MIGR`: l'LP locale sta ricevendo un nodo che sta migrando da un altro LP. Il gestore deve allocare la memoria e le strutture dati necessarie per accoglierlo. Viene ricevuto un messaggio di migrazione, nel quale è contenuto lo stato del nodo che migra.
Viene chiamato `migration_event_handler(from, msg)`, che cerca il nodo con `id=from` nella `table` globale, e lo inserisce nella `stable` di questo LP, aggiornando l'`id` dell'LP nella struttura `data`. Il messaggio di migrazione contiene il target che deve essere raggiunto dal nodo, quindi anche questo viene aggiornato nella struttura.

`EOS`: End Of Step. L'attuale step di simulazione è terminato, sono da svolgere tutte le attività pendenti.
Viene fermato il tempo d'esecuzione in `t2`, e si controlla se `CLOCK` è ancora minore di `END_CLOCK`. Se è maggiore o uguale, la simulazione termina, chiudendo i file di output e terminando il while principale.
Altrimenti se la simulazione non è terminata, si modifica la posizione di tutti i nodi, chiamando `ScanActivity()`. Se un nodo è selezionato per

muoversi (in base alle percentuali scritte sopra) chiama `rwp_move(node, &posX, &posY)`, che muove i nodi con l'algoritmo del Random Waypoint, aggiornando la sua posizione X e Y, e invia un messaggio con la sua posizione da aggiornare tramite `GAIA_Broadcast()`.
Viene controllato se c'è tempo prima che termini la simulazione per mandare dei messaggi ping. In questo caso ogni LP, con la funzione `PingRange()`, scandisce tutte le entità simulate di cui è responsabile (tabella `stable`). Nel caso un nodo sia abilitato a comunicare (cioè dispone di un dispositivo wireless), sarà necessario verificare quali sono tutti i destinatari della comunicazione, cioè tutti i nodi che circondano il mittente e che sono all'interno di un raggio predefinito. Le entità riceventi potrebbero essere gestite da un LP diverso da quello in cui ci troviamo, quindi viene chiamata la funzione `signal_to_range(table, node)`, che scorre la tabella globale e manda i messaggi a tutti i nodi nel raggio d'azione del nodo dato in input, tramite la funzione `GAIA_Send()`.
Al termine dello step di tempo, è il momento di eseguire la migrazione vera e propria dei nodi che sono stati spostati nella lista `mlist`. La funzione `ScanMigrating()` scorre la lista dei nodi, e uno ad uno li elimina e li fa migrare con `GAIA_Migrate(key, msg, sizeof msg)`. Il messaggio che viene inviato ogni volta è un messaggio di migrazione che contiene il target X e Y del nodo.

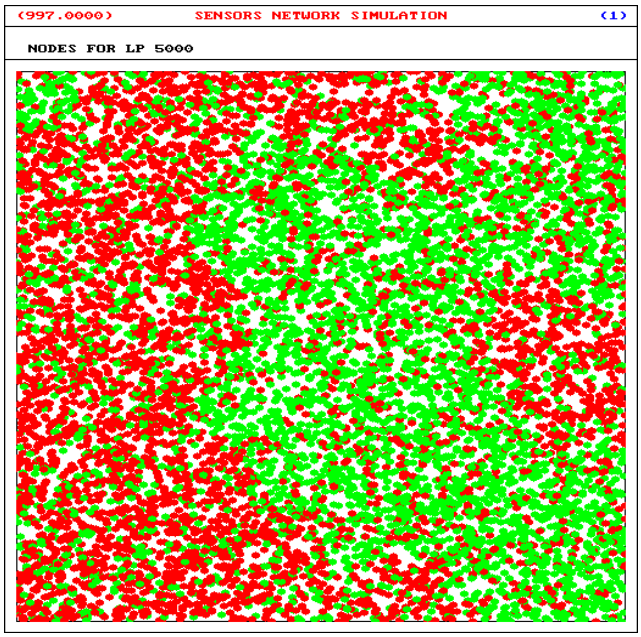


Figura 6: si può notare come i nodi in 2 LP diversi (uno rappresentato dal rosso e uno dal verde) al termine della simulazione appaiono vicini tra loro come conseguenza della migrazione.

Al termine di queste operazioni, viene avanzato il `CLOCK` allo step successivo e vengono scritte le statistiche sui file di output.
`UNSET`: gestisce la ricezione dei messaggi che sono stati inviati precedentemente, o con la `GAIA_Send()` o con la `GAIA_Broadcast()`. Controlla il type del messaggio: se è "`M`" chiama `move_event_handler(from, msg)`, che legge il messaggio contenente la nuova posizione, e aggiorna la posizione X e Y del nodo che ha `id=from`; se type è "`W`", significa che un dispositivo wireless è stato acceso o spento, quindi si chiama `comm_event_handler(from, msg)` che cerca il nodo `from` nella `table` globale e modifica il campo `data` in base ai valori contenenti nel messaggio (YES o NO); se il messaggio ha come tipo "`P`" significa che è stato ricevuto un messaggio ping, e viene chiamata `ping_event_handler(from, to, msg)` che controlla se il nodo `to` è nello stesso LP di `from` e in questo caso aumenta il numero di ping inviati in questo LP.

Terminato il main, viene terminato GAIA in modo che tutti gli LP possano uscire dalla simulazione.

CAPITOLO 5: MODELLI DI MOBILITÀ IMPLEMENTATI

Per implementare nuovi modelli di mobilità, in modo che i nodi possano scegliere quale modello utilizzare per muoversi, sono stati aggiunti alcuni sorgenti e alcune strutture dati.

Nella cartella INCLUDE, è stato aggiunto il file `types.h`, che contiene tutte le strutture dati relative alle tabelle hash che memorizzano i nodi.
Nel campo `data`, descritto nel capitolo precedente, è stato aggiunto un campo nel quale si salva il modello di mobilità utilizzato dal nodo per muoversi e si impostano tutti i suoi parametri:

```
typedef struct hash_data_t {

     ...
     model    *parameters;

} hash_data_t;
```

dove la struttura `model` è impostata in questo modo:

```
typedef struct model {
     char model;
     float speed;
     int x_grid_angle;
     int y_grid;
     float acc_meanspeed;
     float meandir;
     struct timeval dt;
     float vmax_alpha;
} model;
```

Per evitare spreco di memoria, sono stati inseriti dei campi che possono assumere valori diversi in base al modello utilizzato. Il campo `int x_grid_angle`, nel caso del CitySection indica la densità della griglia X, mentre nel Boundless Area indica il massimo cambiamento angolare che serve al nodo per muoversi.
Il campo `float acc_meanspeed`, nel caso del Boundless Area indica la massima accelerazione, mentre nel Gauss Markov la velocità media del nodo, infine il campo `float vmax_alpha`, nel Boundless è il valore massimo che può raggiungere la velocità, mentre nel Gauss Markov è il valore alpha che determina la casualità.
Un altro file è stato aggiunto in questa cartella, e si tratta di `const.h`. Sono state messe al suo interno tutte le macro (funzioni definite con `#define`) e le costanti che prima erano definite in `wireless.c`, e sono state aggiunte due nuove macro:

```
#define SET_MODEL(_struct, _model_name)
     ((_struct)=(_model_name))

#define SET_RADIUS(_radius, _new_radius)
((_radius)=(_new_radius))
```

Che servono appunto per impostare il modello utilizzato dal nodo e per modificare il raggio entro il quale i nodi possono mandare i messaggi.
Sempre in questo file, sono stati aggiunti come costanti i valori che può assumere il campo `model_name`, che indicano i 4 modelli di mobilità implementati in questo progetto:

```
#define RANDOMWAYPOINT 0
#define CITY_SECTION   1
#define BOUNDLESS_AREA 2
#define GAUSS_MARKOV   3
```

In questo capitolo verranno spiegati nello specifico i modelli di mobilità City Section, Boundless Simulation Area e Gauss Markov, poiché il Random Waypoint era già stato spiegato nel capitolo precedente.

5.1 Modello di mobilità City Section

Per utilizzare questo modello sono state definite, oltre alla funzione per il movimento del nodo, una funzione per posizionare inizialmente il nodo nel sistema, e una funzione per impostare una destinazione.

Per la posizione iniziale, il nodo deve essere sopra a qualche strada e non è necessario che sia su un incrocio.
La funzione:

```
void city_initial_position (hash_node_t *node, float *posX, float *posY);
```

prende in input il nodo, che all'interno ha la struttura dati del modello con tutte le informazioni, e la posizione x e y, che saranno riempite con i valori della posizione iniziale.
Viene calcolata una posizione (x, y) casuale all'interno dell'area di simulazione. Dato che l'area sarà divisa in tanti quadrati (definiti dalla griglia X e Y), la funzione calcola a quale lato del quadrato a cui appartiene il punto, è più vicino e lo posiziona su di esso. Se il punto è più vicino a un lato parallelo all'asse X, la X rimane costante, mentre la Y viene modificata per poter giacere su quel lato. Viceversa nel caso il nodo sia capitato in un punto più vicino a un lato parallelo all'asse Y.

Per quanto riguarda il calcolo della destinazione, questa, invece, deve trovarsi su un incrocio.
Quindi la funzione viene chiamata una volta per scegliere la coordinata X, e una volta per la coordinata Y

```
float city_destination(hash_node_t *node, char axis);
```

dove axis può avere il valore “x” o “y”.
Viene chiamata la prima volta all'inizio del programma, e successivamente, quando il target è già stato raggiunto, per impostare una nuova destinazione.
Come nella funzione precedente, si calcola una coordinata casuale e viene posizionata sull'incrocio più vicino tramite una serie di calcoli, che controllano (nel caso della coordinata X) se l'incrocio più vicino è più avanti o più indietro, e nel caso della Y, se è più in alto o più in basso.

Per quanto riguarda il movimento del nodo, è stata definita la funzione:

```
void city_move(hash_node_t *smh_node, float *posX, float *posY);
```

Dove `smh_node` è il nodo, e `posX` e `posY` indicano la posizione nella quale il nodo dovrà spostarsi.
Nel nostro caso, i bordi della simulazione (quindi se la posizione X o Y vale 10000 o 0) sono considerate le superstrade, quindi la velocità viene incrementata di 10.
Subito, come nel Random Waypoit, controlla se la destinazione è stata raggiunta e in questo caso chiama `city_destination()`per entrambi gli assi per definire una nuova destinazione. Se invece la destinazione non è raggiunta, il nodo si sta muovendo su una griglia, quindi non possono essere modificate entrambe le coordinate, ma solo una di esse.
Con la funzione modulo, viene controllato se il nodo si sta muovendo su una linea parallela all'asse X o Y. Un altro controllo che viene fatto è se è stato raggiunto il target sull'asse su cui si sta muovendo il nodo. Infatti il nodo può già trovarsi sulla coordinata Y della destinazione, ma deve ancora raggiungere l'incrocio, dove anche la X equivale al target.

Ad esempio se il target è (400, 1600), il nodo potrebbe trovarsi in (100, 1600), quindi il valore Y non deve cambiare, deve continuare a muoversi sulla X finchè non raggiunge la posizione 400.
Per evitare che la destinazione venga superata, una volta capita la posizione del nodo, viene controllato se aggiungendo il valore di `SPEED`, il nodo raggiungerebbe il target su quella strada, e in questo caso, la posizione viene aggiornata col valore del target. Se invece il nodo è ancora lontano, si calcola se il percorso più breve sarebbe passando attraverso l'area o utilizzando i bordi (essendo l'area toroidale), e si assegna il nuovo valore alla coordinata.
La funzione è strutturata in modo che il nodo continua a muoversi sulla linea sulla quale è posizionato con la posizione iniziale, e quando viene raggiunto il target su quella linea, viene incrementata (o decrementata) l'altra coordinata fino al raggiungimento della destinazione.

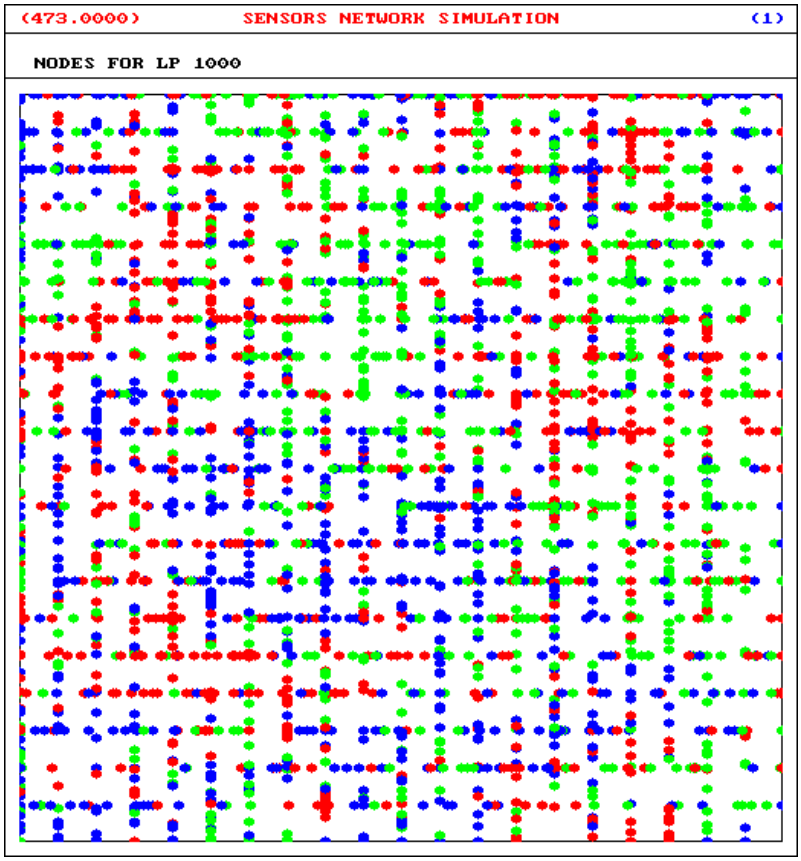


Figura 7: Uno screenshot di come i nodi si muovono con l’algoritmo del City Section

5.2 Modello di mobilità Boundless Simulation Area

La posizione iniziale, nel caso di questo modello, viene stabilità con la stessa funzione del Random Waypoint:

```
void rwp_initial_position (float *posX, float *posY);
```

Per quanto riguarda il movimento, invece, viene chiamata la funzione:

```
void boundless_move(hash_node_t *smh_node, float
     *posX, float *posY);
```

Dove `smh_node` è sempre la struttura del nodo, e `posX` e `posY` saranno la sua nuova posizione dopo lo spostamento.
In questo caso il nodo non ha un target da raggiungere, ma continua a muoversi in continuazione nell'area di simulazione, secondo la seguente formula:

```
X= smh_node->data->posX + (smh_node->data->parameters-
>speed * cos(angle));
```

```
Y= smh_node->data->posY + (smh_node->data->parameters-
>speed * sin(angle));
```

dove `angle` è la nuova direzione che dovrà prendere il nodo. Viene ricalcolato ogni volta, utilizzando la funzione `rand()`, che prende un valore nell'intervallo `[-x_grid_angle, +x_grid_angle]`.

La velocità viene anche questa ricalcolata ogni volta: prima si sceglie un'accelerazione casuale presa nell'intervallo `[-acc_meanspeed, +acc_meanspeed]` e la si moltiplica per la frazione di tempo passata dal movimento precedente (`dt`), poi si controlla che non sia stata superata la velocità `vmax`, e che il valore appena calcolato non sia minore di zero,

infine si assegna al campo `smh_node->data->parameters->speed` il valore opportuno, che sarà da utilizzare nel prossimo step di tempo.

5.3 Modello di mobilità Gauss Markov

Come nel Boundless Simulation Area, anche in questo modello, il nodo non deve raggiungere una destinazione, e la posizione iniziale viene scelta con la stessa funzione del Random Waypoint.

Per quanto riguarda il movimento, è sempre casuale all'interno dell'area, ma viene influenzato da una variabile casuale con distribuzione gaussiana, che agisce sulla velocità e sulla direzione.
La funzione per il movimento è:

```
void gauss_markov_move(hash_node_t *node, float *posX, float *posY);
```

dove i parametri sono gli stessi descritti sopra.
Viene subito calcolata la nuova posizione X o Y e viene assegnata ai parametri passati in input (`posX` e `posY`), poi vengono calcolate la nuova direzione verso la quale il nodo dovrà muoversi, e la velocità da usare nel prossimo step di tempo.
Questi valori sono calcolati con la formula:

```
angleg = (int) (alpha*angleg)+((1-alpha)*node->data->parameters->meandir)+sqrt(1-(pow(alpha,2)))*dg;
```

la stessa formula viene usata per la velocità, usando i valori `meanspeed` al posto di `meandir`, `node->data->parameters->speed` al posto di `angleg` e `sg` al posto di `dg`.
`angleg` è il prossimo angolo d'inclinazione che dovrà usare il nodo, `meandir` è la direzione media del nodo, `alpha` è la variabile che determina la casualità e `dg` è la variabile gaussiana per la direzione.

Per calcolarla è stata usata la funzione:

```
dg = gsl_ran_gaussian(r, 1.0);
```

che preso un generatore casuale di numeri `r`, genera un numero con distribuzione gaussiana con media 0 e sigma 1.

CAPITOLO 6: FUNZIONI UTILIZZABLI TRAMITE LE API

All’interno della cartella MIGRATION-WIRELESS-MOBILE/ si trova il file `models.h` che contiene le API utilizzabili dall’utente per gestire i modelli di mobilità.

6.1 Funzioni generiche per i modelli

Nelle API abbiamo a disposizione funzioni generiche per i modelli. Questo significa che basta chiamare nel main la funzione principale, che accede al campo del nodo che identifica il modello, controlla che modello utilizza, e chiama la funzione opportuna, nascondendo ogni passaggio all'utente.

6.1.1 Funzione per l'inizializzazione del modello

Una volta che sono stati impostati i parametri per il modello, è necessario inizializzare la posizione del nodo nel sistema. Così nelle API abbiamo la funzione

```
void model_inizialization(hash_node_t *node);
```

che legge il campo `parameters-> model` del nodo, e chiama la funzione appropriata per dare un punto iniziare al nodo nel sistema.
Nel caso del modello di mobilità Gauss Markov, Boundless Simulation Area e Random Waypoint, la posizione sarà casuale, mentre per il City Section, la posizione deve essere su una strada della griglia X o Y.

6.1.2 Funzione per impostare la destinazione

Alcuni modelli prevedono che il nodo debba raggiungere una destinazione. Al contrario del Gauss Markov e del Boundless, che il nodo continua a muoversi senza un target, nel City Section e nel Random Waypoint, è necessario salvare dei valori nei parametri `targetX` e `targetY` del nodo.

```
void model_destination(hash_node_t *node);
```

Quindi viene sempre controllato che tipo di modello usa il nodo in questione, e viene calcolata la destinazione, tenendo presente che nel City Section non deve essere completamente casuale, poiché deve giacere su un incrocio tra strade.

6.1.3 Funzione per il movimento del nodo

Una volta definiti i parametri, la posizione iniziale, e (per quelli che lo richiedono) la destinazione, le API mettono a disposizione la funzione per muovere il nodo secondo il modello di mobilità scelto.
Così deve essere chiamata la funzione:

```
void model_movement(hash_node_t *node, float *posX, float *posY);
```

che controlla il modello utilizzato dal nodo, e chiama la funzione opportuna. Questa modifica il valore di `posX` e `posY`, aggiornando la sua nuova posizione.

6.2 Funzione per impostare i parametri di default

Quando viene scelto il modello di mobilità per un nodo tramite la macro `SET_MODEL()`, è necessario impostare anche i suoi parametri.
Nelle API abbiamo diverse funzioni che svolgono questo compito, una funzione che imposta il modello di mobilità con i parametri di default, e le singole funzioni per ogni modello, nel caso l'utente voglia impostare dei parametri scelti da lui.

Se si vogliono usare i parametri di default, è necessario chiamare questa funzione, subito dopo la scelta del nodo:

```
void model_parameters(hash_node_t *node);
```

che accede alla struttura `parameters` del nodo e imposta i valori in base al modello di mobilità. I valori che verranno assegnati sono i seguenti:

Random Waypoint: l'unico parametro da impostare è la velocità, e viene impostata a 10;

```
speed = 10;
```

City Section: oltre alla velocità, che vale sempre 10, sono da definire anche la densità della griglia X e Y sulla quale i nodi si muovono. In questo caso abbiamo:

```
speed = 10;
grid X = 500;
grid Y = 500;
```

che significa che la dimensione dell'area X che è impostata a 10000, ogni 100 posizioni, avrà una strada.

Boundless Simulation Area: in questo caso la velocità viene calcolata di volta in volta, utilizzando l'accelerazione, la frazione di tempo trascorsa dal movimento precedente, e una massima velocità che il nodo può raggiungere. I parametri vengono impostati in questo modo:

```
speed = 0;
TIMER_NOW(dt);
acceleration = 2;
max speed = 10;
angle = 90;
```

Dove l'angolo è il massimo cambiamento angolare che può fare il nodo, e l'accelerazione e la massima velocità, indicano i valori più alti che possono raggiungere durante la simulazione. Questi valori sono utilizzati per definire gli intervalli, all'interno dei quali saranno da scegliere i valori casuali.
Per quanto riguarda il tempo, invece, viene salvato il momento attuale nella struttura timeval `dt`, ed ogni volta che si calcola una nuova posizione, viene aggiornato.

Gauss Markov: in questo modello, i parametri da impostare sono la velocità media del nodo, la sua direzione media, e alpha, un valore che determina la casualità del movimento del nodo. Con alpha=1 abbiamo un movimento lineare, con alpa = 0 invece è del tutto casuale. Nei parametri di default, alpha assume un valore intermedio:

```
speed = 0;
mean speed = 10;
mean direction = 90;
alpha = 0.5;
```

dove la velocità parte da zero, poiché inizialmente il nodo è fermo.

6.2 Funzioni specifiche per i modelli: impostare i parametri manualmente

Nel caso i parametri di default non siano quelli di cui ha bisogno l'utente, le API forniscono una serie di funzioni che permettono di modificare manualmente il valore dei parametri:

```
void parameter_rwp(model *parameters, float velocity);

void parameter_city(model *parameters, float velocity,
int gridx, int gridy);

void parameter_boundless(model *parameters, float
acceleration, float speedMax, int angle);

void parameter_gauss_markov(model *parameters, float
meanspeed, float meandir, float alpha);
```

come parametri deve essere passata la struttura del nodo da modificare (`data->parameters`), e i valori per i parametri del modello.
Ad esempio per avere, nel modello di Gauss Markov, alpha = 1, sarà necessario chiamare:

```
parameter_gauss_markov(node->data->parameters, 10, 90,
1.0);
```

CAPITOLO 7: PROGETTAZIONE E IMPLEMENTAZIONE DI UN MECCANISMO D'INFEZIONE

Per confrontare e analizzare i diversi modelli di mobilità implementati, è stato sviluppato un meccanismo che diffonde un'infezione tra i nodi all'interno del sistema.
Questo modello a infezione permette di studiare come il diverso movimento dei nodi in base al modello di mobilità scelto, influenzi l'andamento della simulazione.
Il sistema all'interno del quale è stato implementato questo meccanismo, è lo stesso descritto precedentemente: un'area toroidale all'interno della quale si trovano dei nodi mobili gestiti da diversi Logical Process.
Il meccanismo d'infezione introdotto garantisce che ogni nodo che si avvicina ad un altro, abbia la possibilità di infettarlo, e questa possibilità è limitata da una percentuale d'infezione. Ogni nodo che viene infettato, informa i propri vicini del suo cambiamento di stato tramite un messaggio.
Obiettivo della simulazione è quello di studiare le differenze tra i modelli di mobilità implementati, mettendo in evidenza come il diverso movimento dei nodi favorisce o meno lo sviluppo dell'infezione.
I parametri presi in analisi per lo studio dei diversi modelli di mobilità, sono in alcuni casi la percentuale d'infezione e il tempo che viene impiegato per raggiungere l'infezione totale del sistema, in altri la percentuale raggiunta a determinati step di tempo.
Questi parametri sono influenzati dalla densità dei nodi nell'area di simulazione, dalla loro velocità, dalla percentuale con la quale si diffonde l'infezione e dal modello di mobilità utilizzato.
All'interno del sistema viene scelto casualmente un nodo dal quale partirà l'infezione e si svilupperà attraverso tutta l'area.

Sono stati studiati 3 tipi diversi d'infezione all'interno del sistema:

1- tutti i nodi si muovono con lo stesso modello di mobilità e la simulazione termina quando tutti i nodi del sistema sono stati infettati. In questo caso viene utilizzato il raggio iniziale di 250 unità;

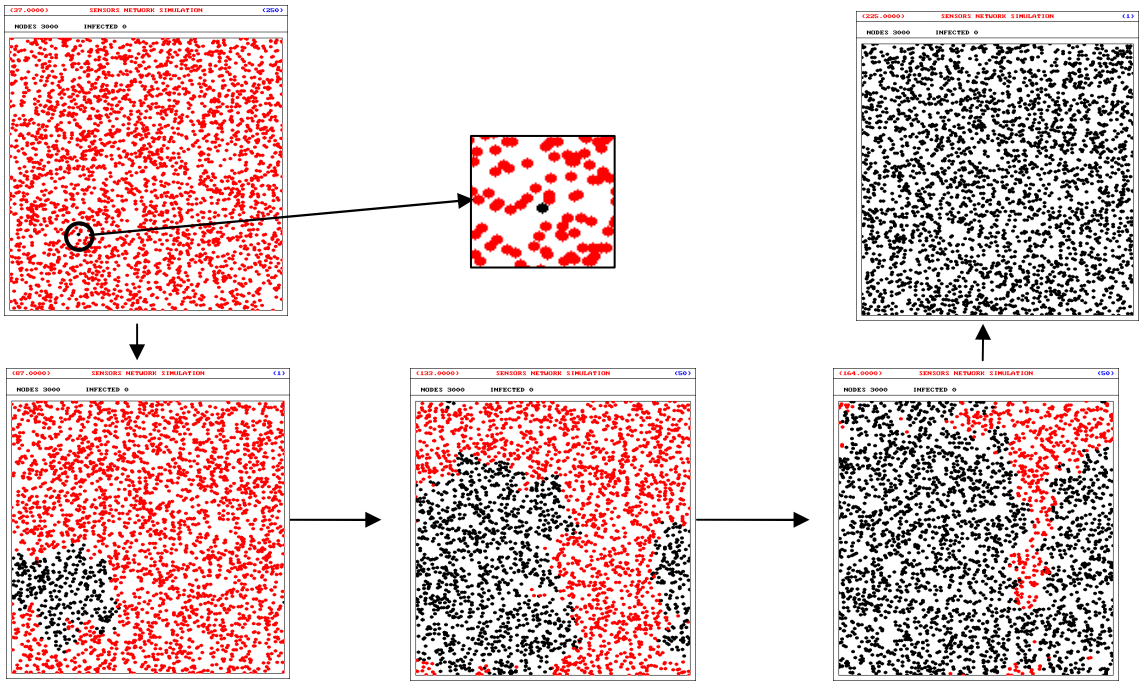


Figura 8: Diffusione dell'infezione nel sistema: i nodi infetti sono disegnati in nero, quelli sani in rosso

2- i nodi infetti e quelli non infetti si muovono con modelli di mobilità differenti. Questo significa che durante la simulazione, quando un nodo viene infettato, cambia il suo movimento in base al nuovo modello di mobilità. L'infezione avviene solo tra il contatto di nodi vicini, per questo è stato impostato un raggio di 100 unità. In questo modo si ha un'infezione del sistema molto più lenta e la simulazione termina quando sono stati raggiunti 500 step di tempo;

3- inizialmente i nodi sono ripartiti in classi e ognuna delle classi ha caratteristiche diverse per modello di mobilità o parametri di applicazione del modello. L'infezione avviene sempre per contatto, e la simulazione termina quando si sono raggiunti i 1000 step di tempo, o quando tutto il

sistema è stato infettato, poiché in base alla scelta delle diverse classi si ottengono diversi scenari d'infezione.

7.1 Implementazione

Per garantire lo sviluppo dell'infezione all'interno del sistema, è stata aggiunta una percentuale di probabilità con la quale i nodi possono infettare i vicini. Questa percentuale è indicata con `PERC_INF` all'interno del file `const.h`.
Alla struttura `data` dei nodi è stato aggiunto il campo

```
char infected;
```

che indica se il nodo è stato infettato o meno.
Alla struttura `table`, che contiene tutti i nodi presenti nel sistema, è stato aggiunto anche qui il campo

```
int infected;
```

che conta il numero totale di nodi infetti nel sistema. Da questo valore, sarà successivamente calcolata la percentuale che indica qual è il livello di infezione totale.

Inizialmente nel sistema abbiamo solo un nodo infettato, e il suo `ID` viene scelto casualmente all'inizio della simulazione. Quando questo nodo viene inizializzato, viene propagata l'informazione a tutta la simulazione tramite un messaggio di tipo "`I`" (Infected), e il campo `table->infected` viene incrementato.
Le funzioni per diffondere l'infezione all'interno del sistema, sono state inserite all'interno del file `wireless.c`.
Al termine di ogni time step (all'interno del case: EOS) per prima cosa viene verificata la condizione per terminare la simulazione, ossia se la percentuale di nodi infetti nel sistema è uguale al 100%.
Se questa condizione non è ancora verificata, viene chiamata la funzione

```
InfectionRange();
```

che scorre la tabella locale dell'LP in cui si trova il programma e controlla quali sono i nodi infetti. Se un nodo è infetto e la percentuale di infezione (che indica la probabilità con la quale un nodo infetta i suoi vicini) lo consente, viene chiamata la funzione

```
infect_nodes(table, node);
```

Questa funzione viene chiamata usando come parametro la tabella globale `table`, poiché può essere che un nodo abbia come vicino un'altro nodo che però non appartiene allo stesso LP, ma che è comunque da infettare.
Si scorre tutta la tabella, alla ricerca dei nodi all'interno del raggio d'azione del nodo passato come parametro. Ogni volta che si prende in esame un nodo, si controlla che questo non sia già infettato e in caso contrario viene propagato un messaggio di tipo "`I`" alla simulazione contenente il nuovo stato del nodo utilizzando la funzione `GAIA_Broadcast();`.

Quando nel ciclo principale del main, viene ricevuto un messaggio di tipo "`I`" dalla `GAIA_Receive()`, si entra nel case: UNSET, che chiama la funzione

```
infection_event_handler(from, msg);
```

che cerca nella tabella globale `table` il nodo con `id=from`. Una volta trovato, viene aggiornato il suo campo `data->infected` col valore contenuto nel messaggio e viene aggiornato il numero di nodi infetti nel sistema, incrementando il campo `table->infected`.
Poichè ogni LP esegue una simulazione indipendente, viene controllato se è già stato ricevuto un messaggio d'infezione per quel nodo, per evitare che venga infettato nuovamente da un messaggio pendente.

CAPITOLO 8: VALUTAZIONE DEI MODELLI DI MOBILITÀ TRAMITE IL MECCANISMO D'INFEZIONE

Sono state eseguite diverse simulazioni per verificare il comportamento del sistema. Per ognuno dei 3 tipi diversi d'infezione sono state eseguite 10 simulazioni ed è stato studiato il comportamento dei nodi, facendo anche raffronti con sistemi reali.
Al termine delle 10 simulazioni per ogni tipo d'infezione, sono stati calcolati gli intervalli di confidenza. Un intervallo di confidenza è un particolare tipo di stima intervallare di un parametro della popolazione [11]. La frequenza con la quale l'intervallo osservato contiene il parametro in analisi, è determinata dal livello di confidenza. In questo caso viene utilizzato un livello di confidenza del 95%, ciò significa che nel 95% delle prove effettuate il parametro avrà il valore all'interno dell'intervallo, mentre per il 5% dei casi non ci rientrerà.

8.1 I nodi si muovono con lo stesso modello di mobilità

Per ogni diverso modello di mobilità, sono state eseguite 10 simulazioni, che venivano interrotte quando la percentuale di nodi infetti era pari al 100% dei nodi nel sistema. In questo modo si può notare come il movimento dei nodi secondo un modello di mobilità o un altro, aumenta o diminuisce il tempo d'infezione.
Per quanto riguarda la percentuale d'infezione, per ogni modello di mobilità sono state fatte 10 simulazioni con una percentuale del 15%, e 10 simulazioni con la percentuale del 10%. Si può ovviamente notare, come più si abbassa la percentuale, più si alza il tempo di infezione.
Tutte le simulazioni sono state eseguite con 2 LP (Logical Process) e 3000 nodi, che si muovevano su un area di 10000x10000 unità di spazio.

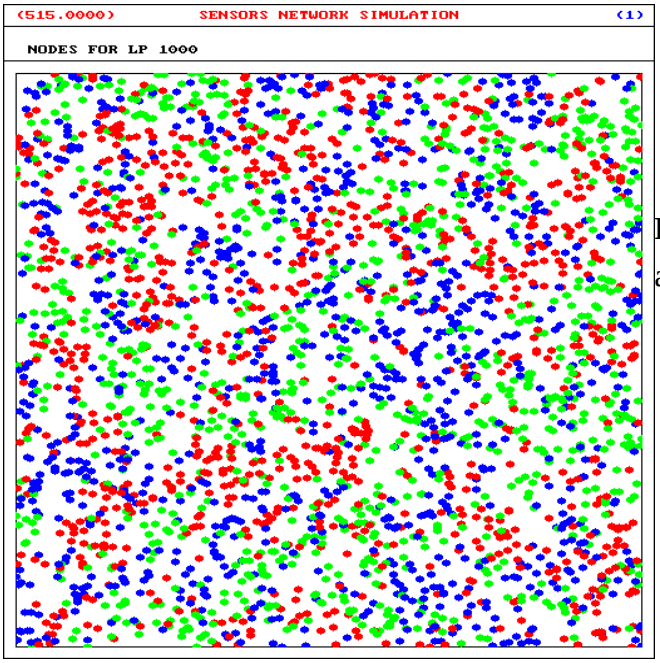
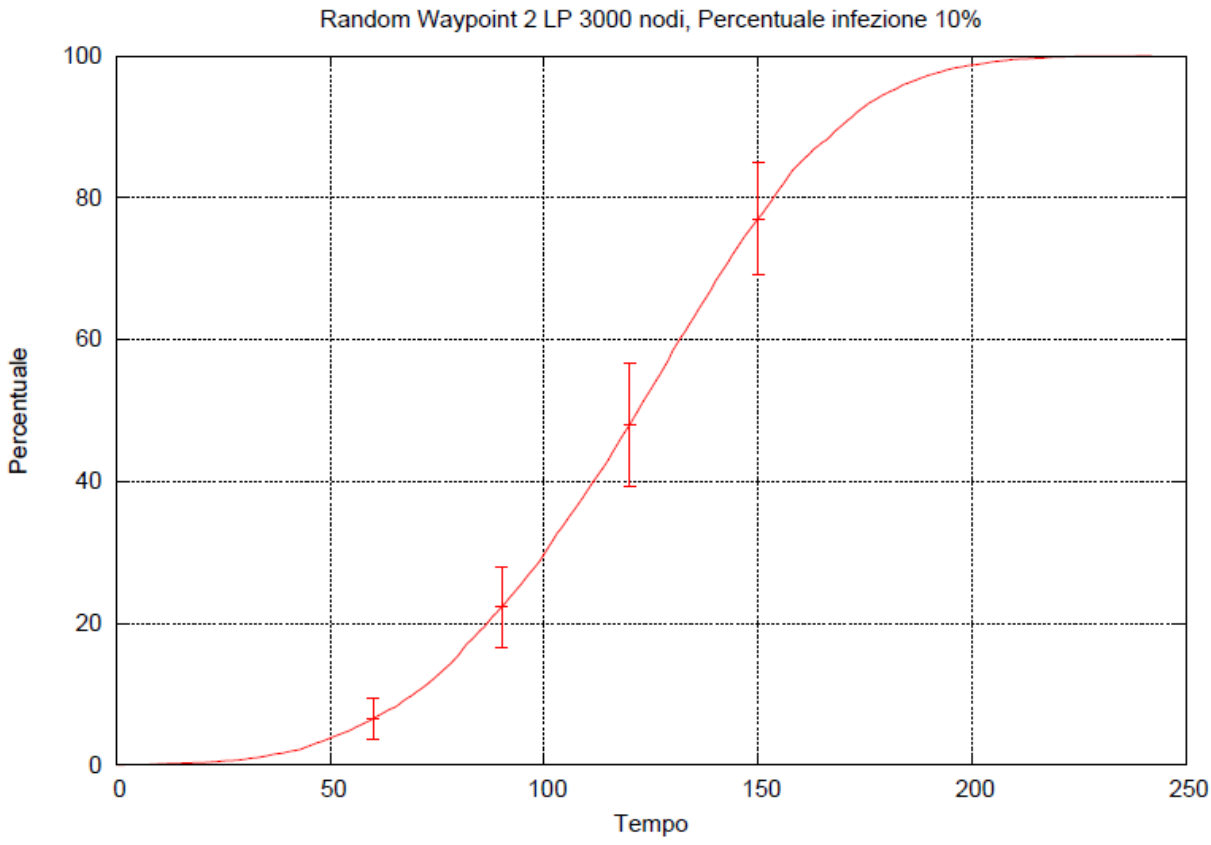


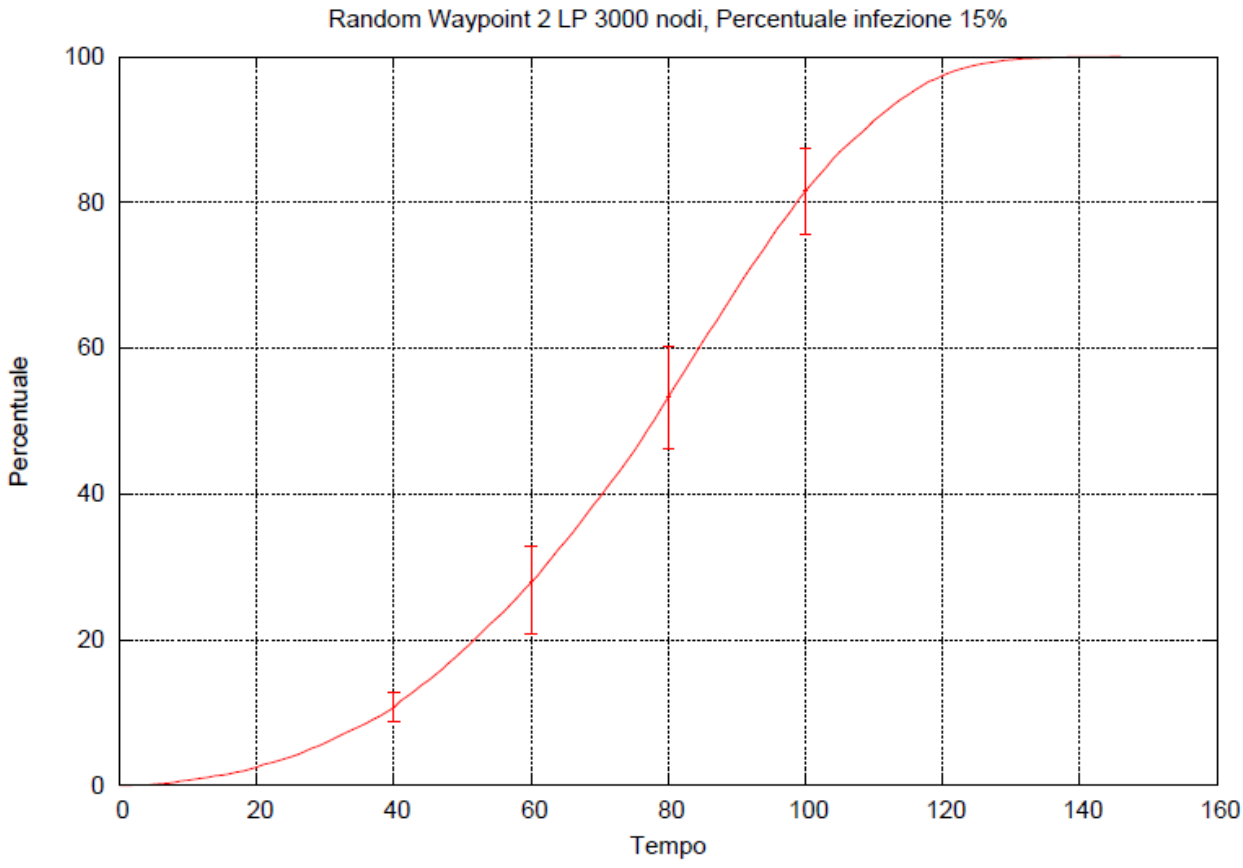
Figura 9: Densità di 3000 nodi su un area di 10000x10000 unità di spazio

I grafici nei paragrafi successivi, sono la media di 10 diverse simulazioni per ogni modello, sulle quali sono stati calcolati gli intervalli di confidenza con un livello di confidenza del 95%.

8.1.1 Valutazioni sull'algoritmo del Random Waypoint



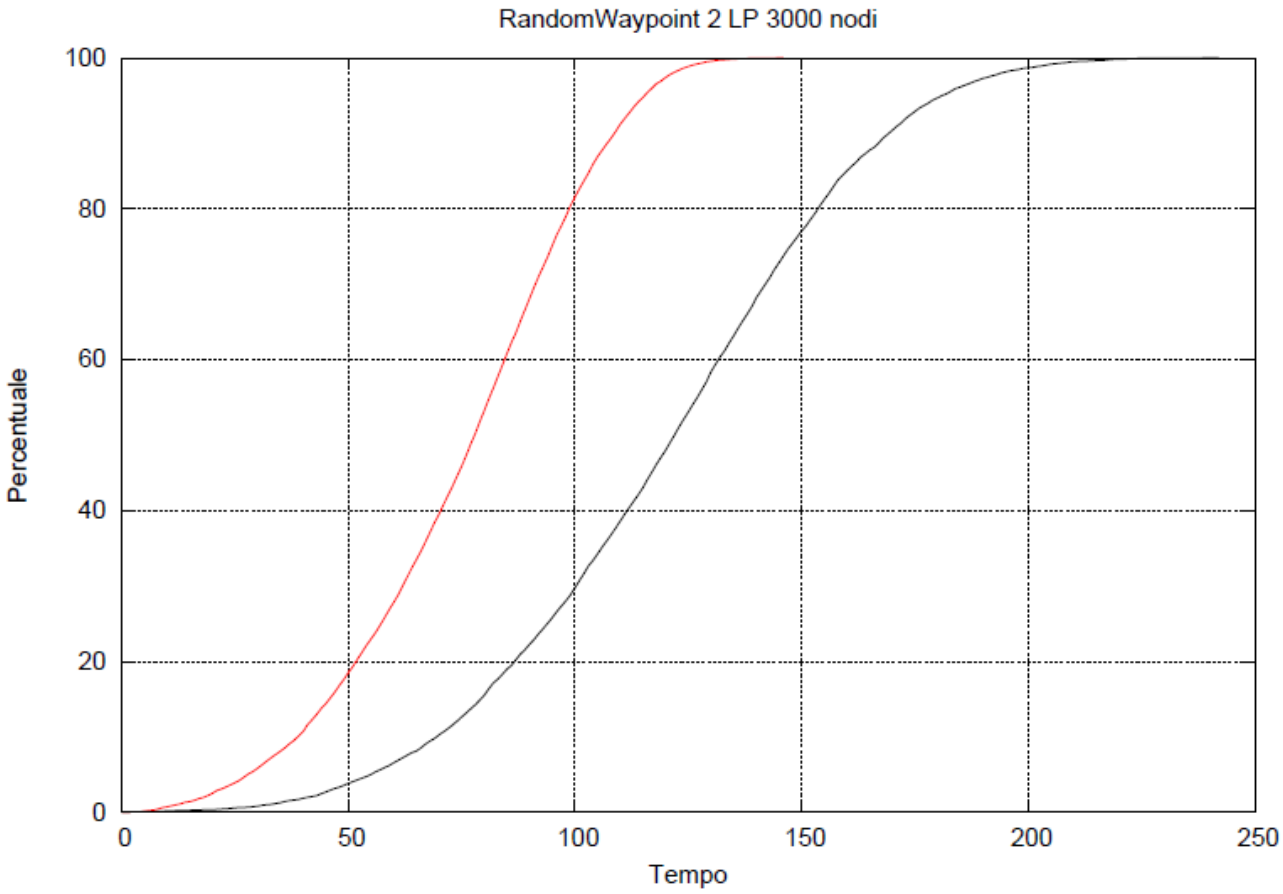
Inizialmente si ha una percentuale bassa di nodi infetti, poiché un basso numero di nodi sono stati infettati. Dopo lo step di tempo 50, infatti, si può notare una crescita esponenziale dell'infezione nel sistema, poiché il numero di nodi infetti è abbastanza alto da permettere un'infezione più veloce.
Gli intervalli di confidenza sono rappresentati sul grafico dalle linee rosse verticali e per certi step di tempo è stato calcolato l'intervallo della percentuale:
step 60: [3.67; 9.43]
step 90: [16.65; 27.88]
step 120: [39.32; 56.58]
step 150: [69.05; 85.04]



Con una percentuale d'infezione del 15%, la simulazione termina attorno allo step di tempo 160, al contrario della simulazione precedente, dove la simulazione col tempo maggiore, era terminata allo step 250.

Poiché la percentuale d'infezione è più alta, si può notare come la fase iniziale termini allo step di tempo 20, dove inizia ad esserci un discreto aumento dei nodi infetti nel sistema.

In questo caso, gli intervalli di confidenza calcolati per gli step di tempo sono:
step 40: [8.65; 12.70]
step 60: [20.76; 32.87]
step 80: [46.21; 60.32]
step 100: [75.57; 87.50]

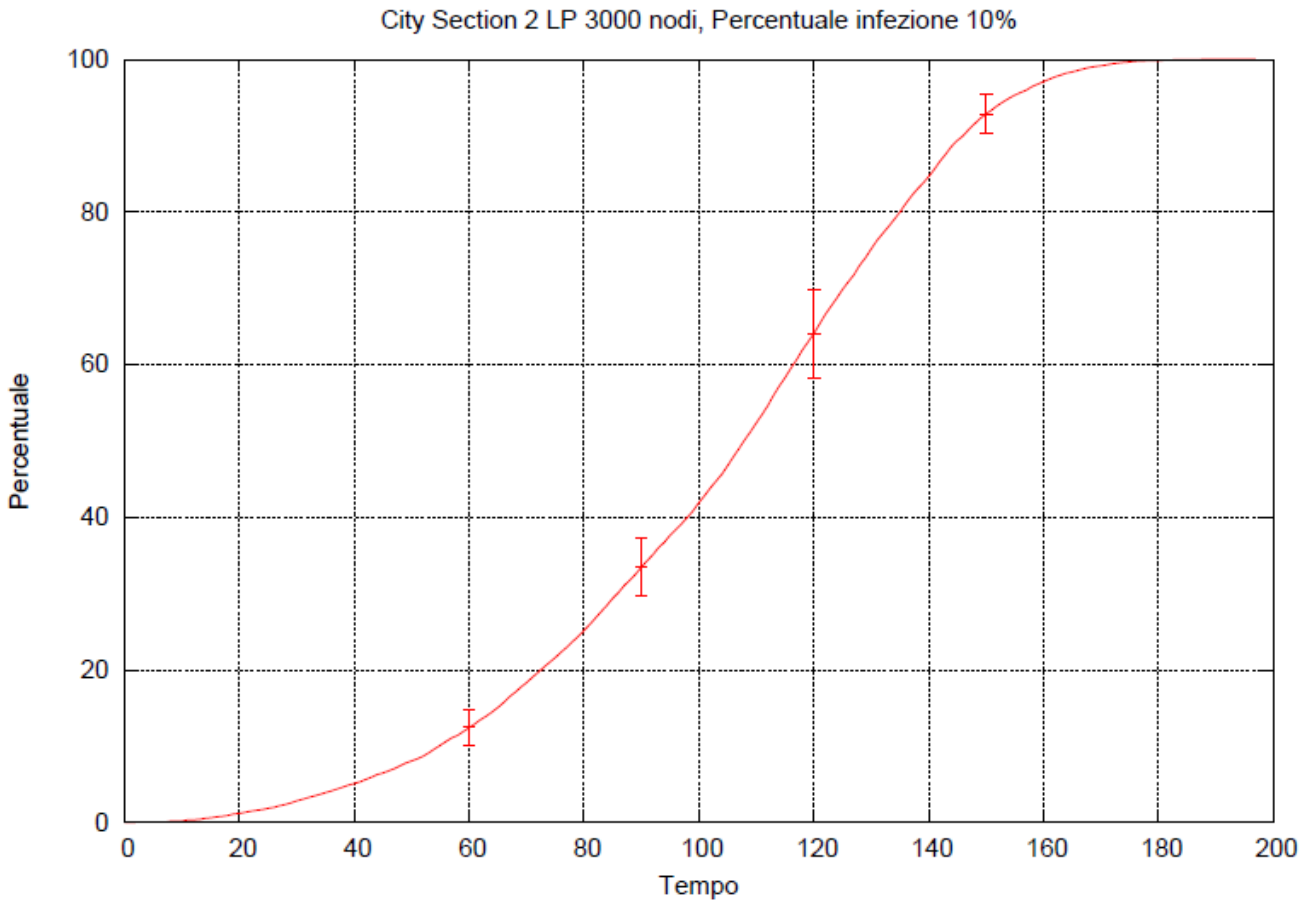


In questo grafico si possono osservare disegnate in rosso la linea con la percentuale del 15% e in nero la percentuale del 10%.
Più è bassa la percentuale, più tempo viene impiegato per infettare tutto il sistema. Inoltre si può notare che con la percentuale del 15% l'infezione

cresce più velocemente, mentre con la percentuale del 10% il grafico è più lineare.

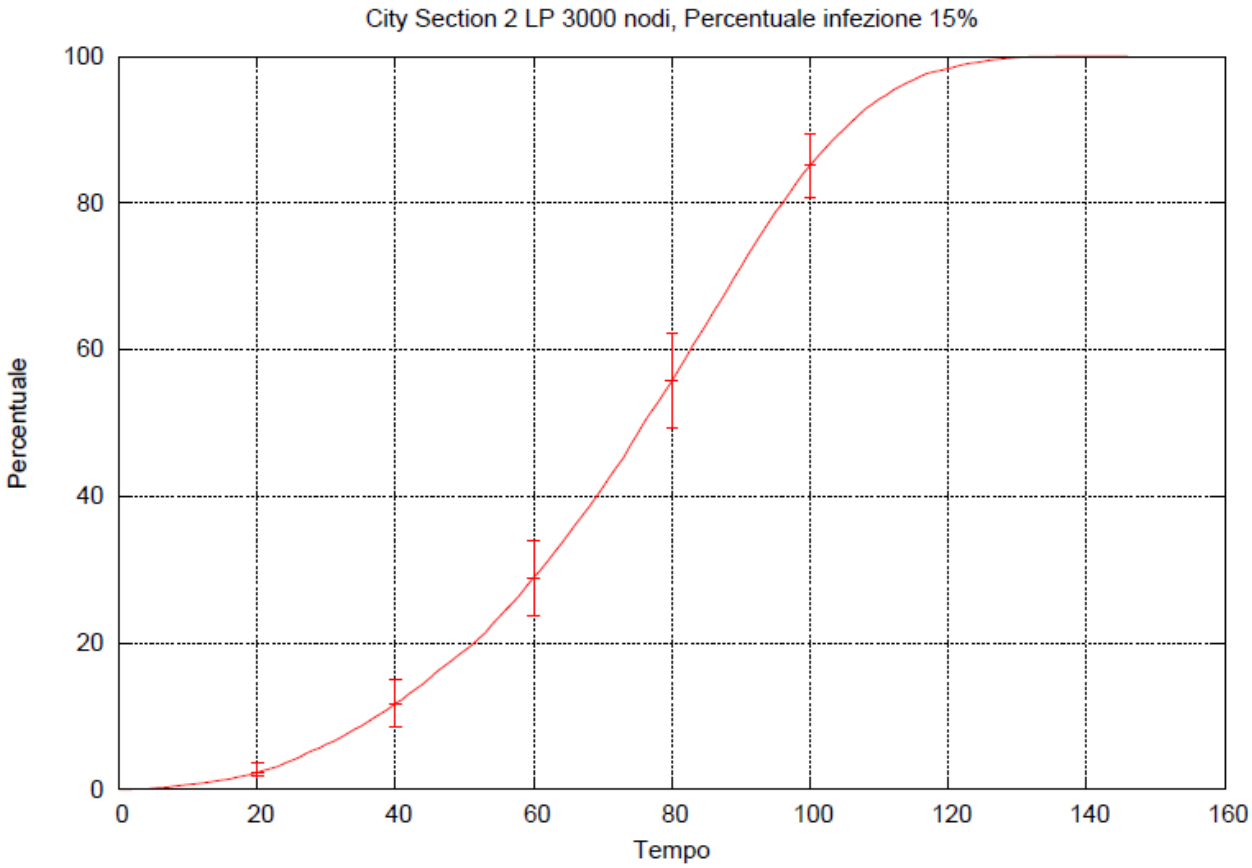
È stato calcolato un intervallo di confidenza per verificare mediamente a quale step di tempo termina la simulazione nel caso la percentuale sia del 10% o del 15%, e questi sono i risultati ottenuti:
10%: [195; 222]
15%: [132; 142]

8.1.2 Valutazioni sull'algoritmo del City Section



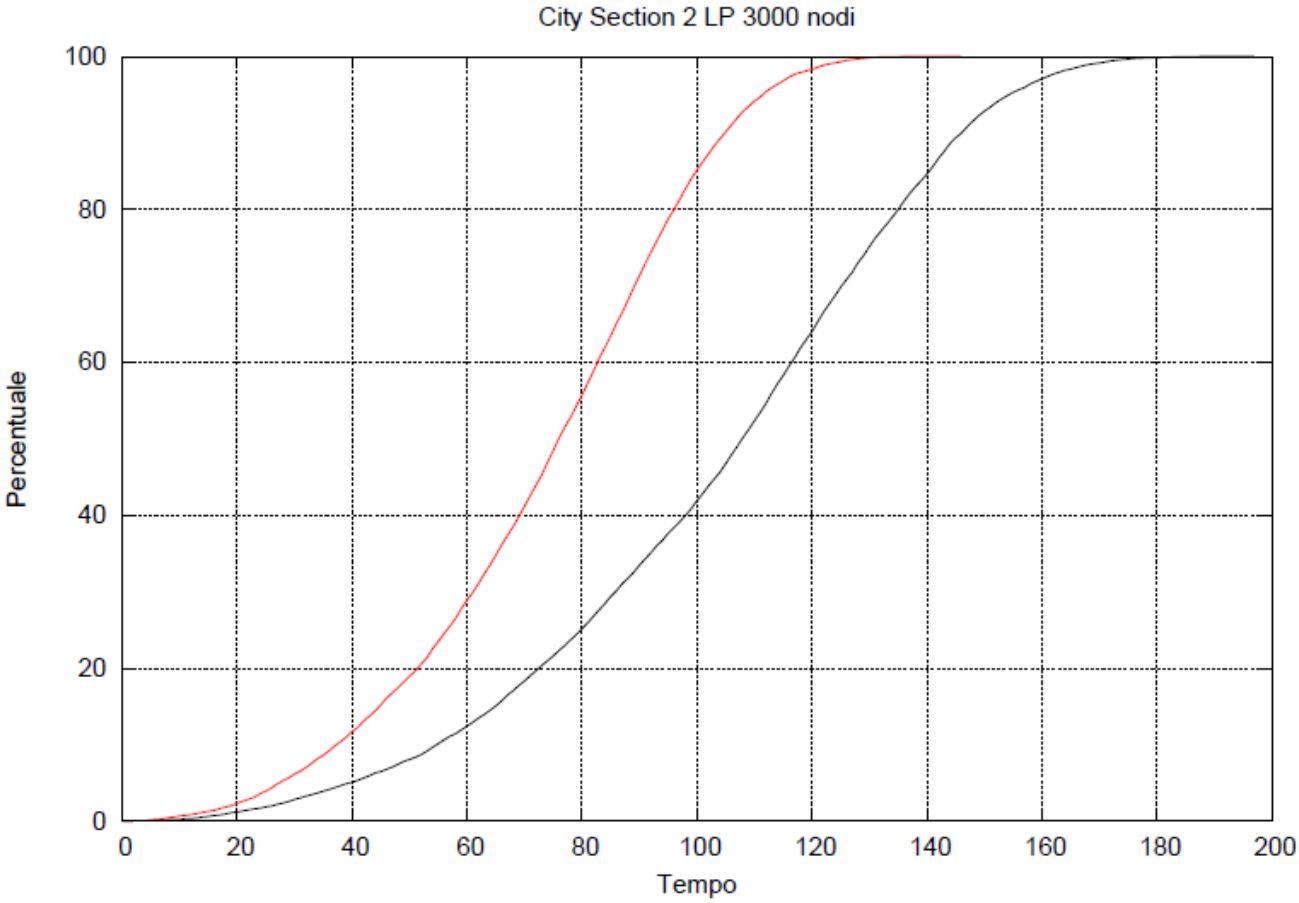
Si può notare in questo algoritmo, la differenza nella parte iniziale con il RandomWaypoint nelle stesse condizioni. La percentuale di nodi infetti nel sistema, infatti, aumenta in modo particolare poco prima dello step di tempo 40, al contrario del modello precedente che aumentava dallo step 50.

Questo succede perché i nodi non si trovano in posizioni casuali all'interno del sistema garantendo una minore densità, ma a causa delle loro costrizioni a seguire le strade del modello, la densità aumenta e di conseguenza anche il numero di nodi in contatto tra loro per poter essere infettati.
Gli intervalli di confidenza per questo modello, sono stati calcolati agli step di tempo 60, 90, 120, 150:
step 60: [10.04; 14.81]
step 90: [29.61; 37.27]
step 120: [58.27; 69.78]
step 150: [90.21; 95.38]



Con la percentuale del 15% la fase iniziale dove l'infezione rimane bassa, equivale a quella degli altri modelli, ossia l'infezione inizia a crescere dopo lo step di tempo 20.

Gli intervalli di confidenza rappresentati sul grafico dalle righe rosse verticali, sono i seguenti e indicano gli intervalli per le percentuali in un determinato step di tempo:
step 20: [1.77; 3.57]
step 40: [8.49; 14.95]
step 60: [23.76; 34.02]
step 80: [49.21; 62.19]
step 100 [80.71; 89.49]

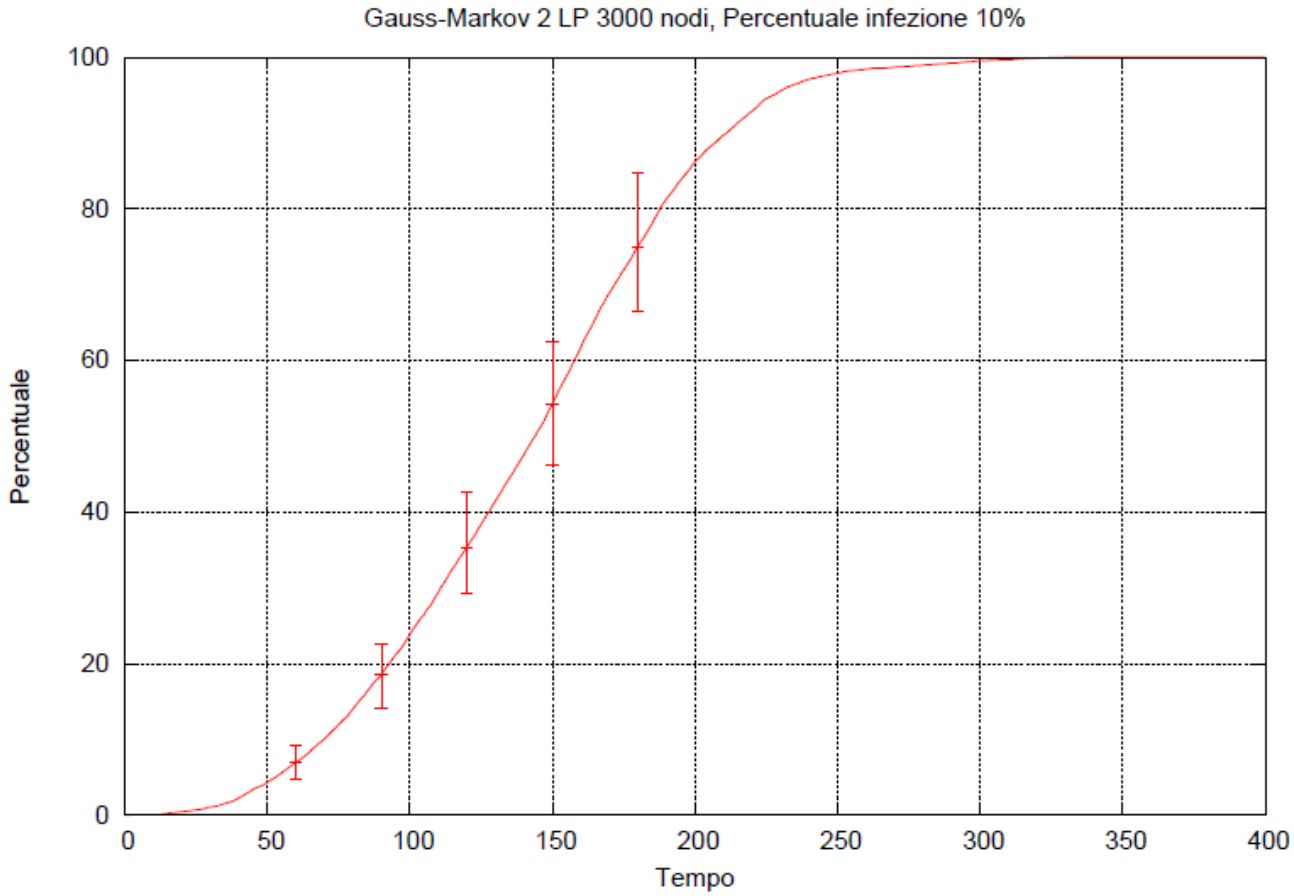


In questo grafico la linea rossa indica la percentuale del 15%, mentre quella nera la percentuale del 10%.
Per quanto riguarda la fase iniziale, allo step di tempo 20 le due percentuali si equivalgono, ma mentre con la percentuale più bassa l'infezione rimane più o meno stabile fino allo step 40, con la percentuale del 15% la linea del grafico subito dopo lo step di tempo 20 cresce velocemente.

Sono stati calcolati i seguenti intervalli di confidenza:
10%: la simulazione termina mediamente in un intervallo tra [173; 185] step di tempo
15%: la simulazione termina mediamente in un intervallo tra [125; 135] step di tempo

Confrontandoli con l'algoritmo del RandomWaypoint (e anche con gli algoritmi successivi), si nota come gli intervalli di tempo siano più bassi, a causa della densità più alta dei nodi che caratterizza questo modello.

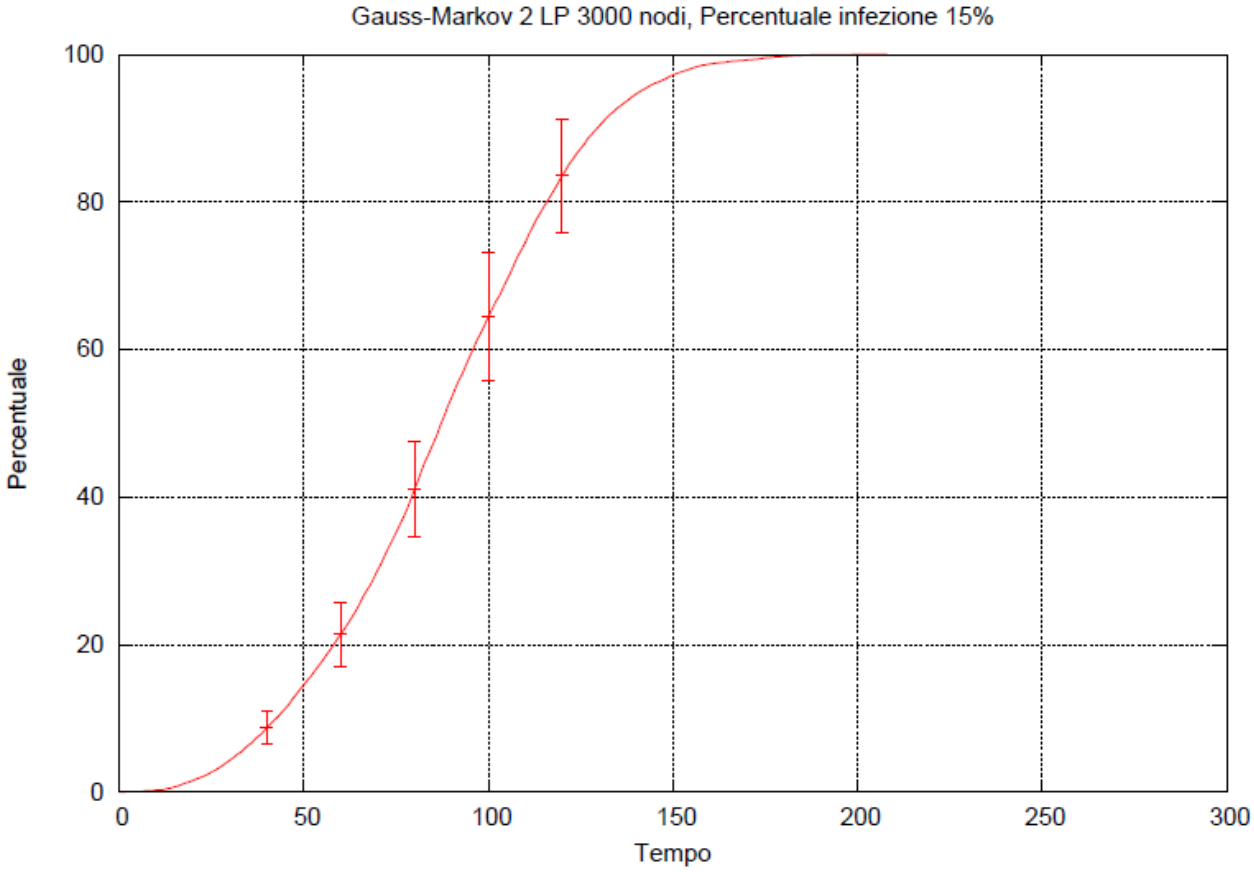
8.1.3 Valutazioni sull' algoritmo Gauss-Markov



In questo modello, come nel caso del Random Waypoint, la fase iniziale dura circa 50 step di tempo. La simulazione in media termina attorno allo step di tempo 320, e si può notare, confrontandolo con gli altri modelli che il modello Gauss-Markov è quello che impiega più tempo a terminare la simulazione.

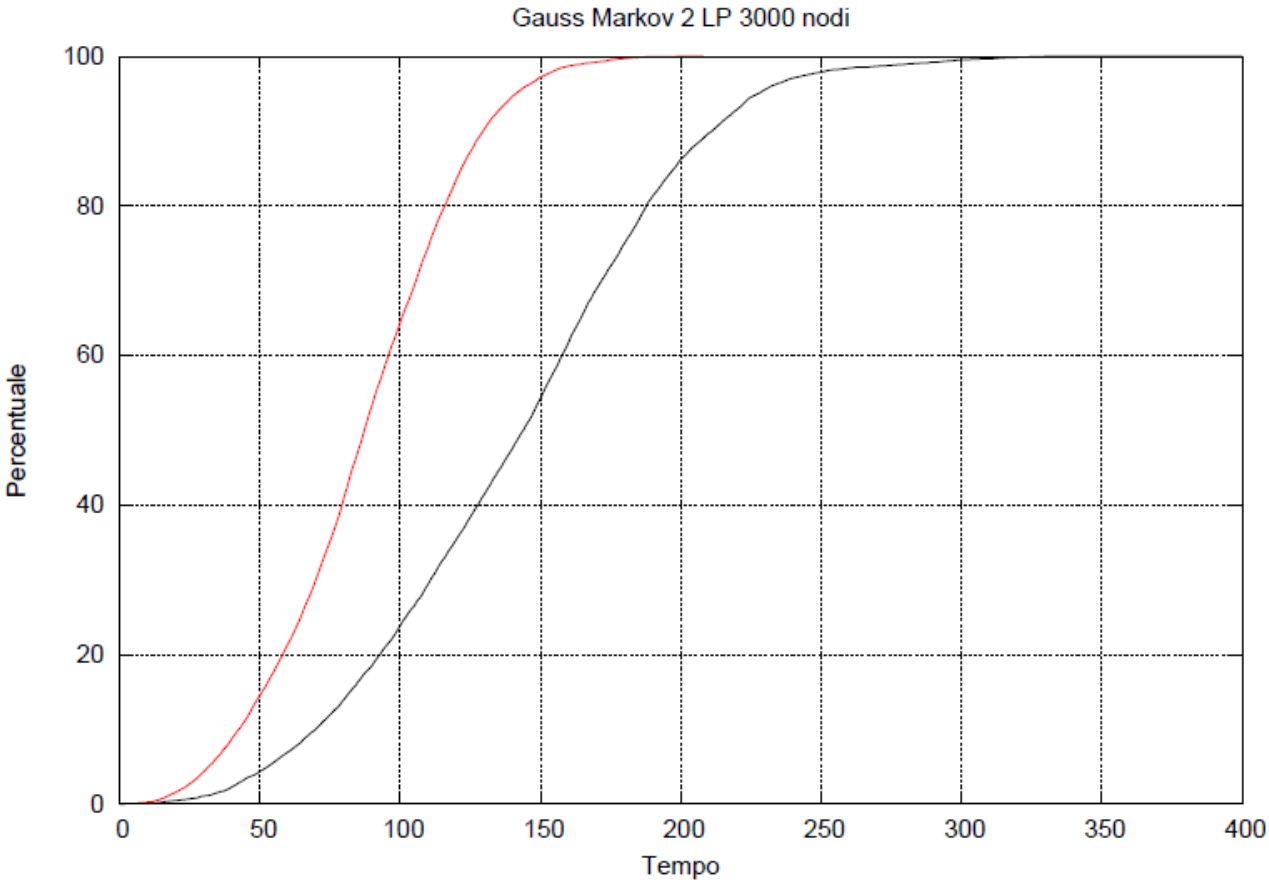
Gli intervalli di confidenza calcolati, indicano le seguenti percentuali per gli step di tempo presi in analisi:

step 60: [4.74; 9.11]
step 90: [14.15; 22.55]
step 120: [29.2; 42.55]
step 150: [46.13; 62.38]
step 180: [66.56; 84.85]



Come gli altri modelli, con la percentuale del 15%, la fase iniziale in cui la percentuale rimane bassa è attorno allo step 20, e si può facilmente notare come il 100% viene raggiunto in minor tempo rispetto al grafico precedente. Al cambiare della percentuale questo modello rimane quello che impiega più tempo, terminando mediamente allo step di tempo 210, al contrario degli altri modelli che rimangono tutti sotto allo step 200.

Gli intervalli di confidenza calcolati per questo modello sono i seguenti:
step 40: [6.42; 11.03]
step 60: [17.10; 25.71]
step 80: [34.62; 47.49]
step 100: [55.70; 73.15]
step 120: [75.84; 91.23]

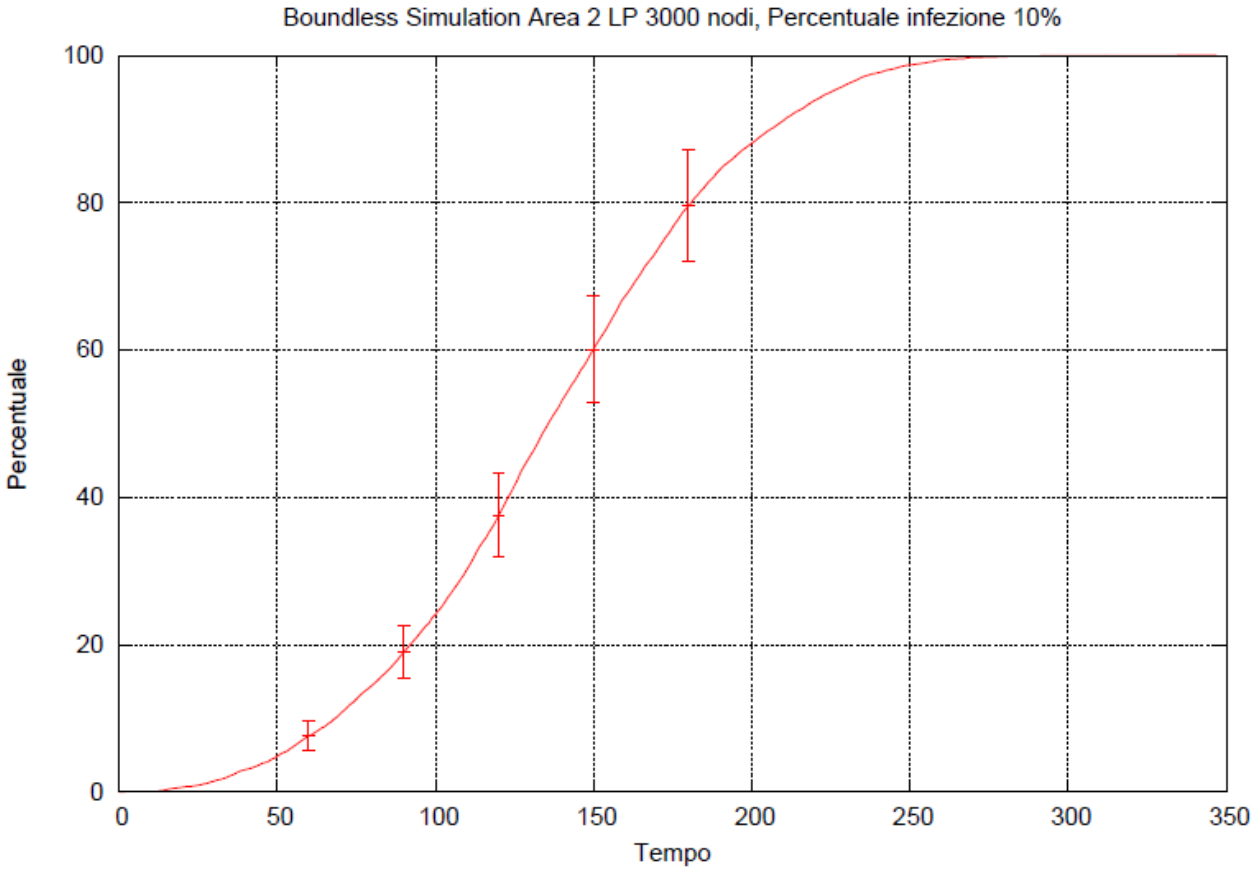


In questo grafico, dove la percentuale del 15% è disegnata in rosso, e quella del 10% in nero, è evidente la differenza tra i due grafici.
Si nota la differenza tra la fase iniziale, dove con la percentuale del 10% il grafico subisce piccole variazioni fino allo step 50, mentre con la percentuale del 15, la linea sale velocemente subito dopo lo step di tempo 20.
Anche per quanto riguarda la fase finale si può notare la differenza: con la percentuale del 15% una volta raggiunto lo step 150 al quale la linea di stabilizza, nel giro di poco tempo viene infettato tutto il sistema, mentre

con la percentuale del 10%, raggiunta la stabilità allo step 250, occorrono più di 50 step di tempo per completare l'infezione totale.

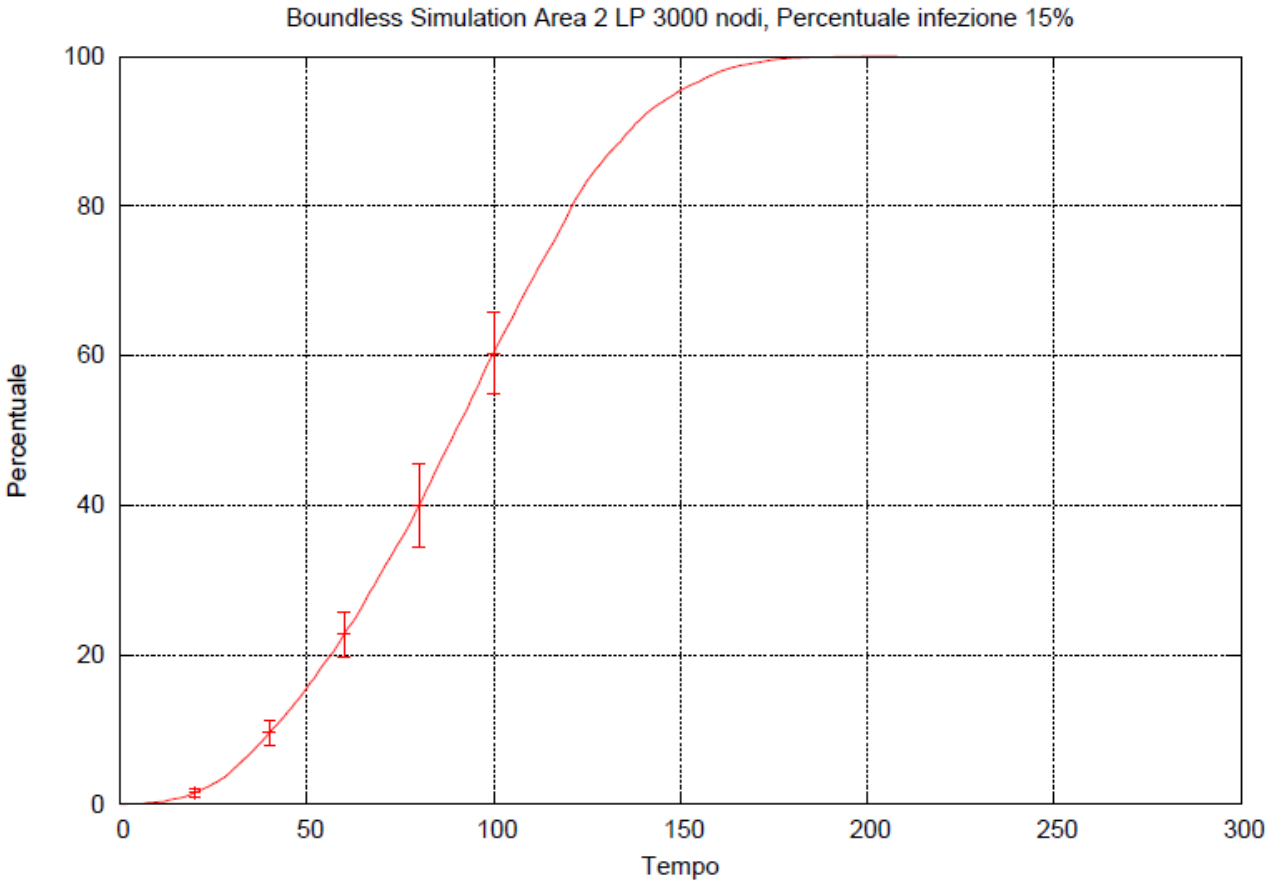
Calcolando gli intervalli di confidenza quando la percentuale ha raggiunto il 100%, si può notare la differenza di tempo di durata tra la simulazione con la percentuale del 10% e del 15%:
percentuale del 10%: [239; 303]
percentuale del 15%: [158; 198]

8.1.4 Valutazione sull'algoritmo del Boundless Simulation Area



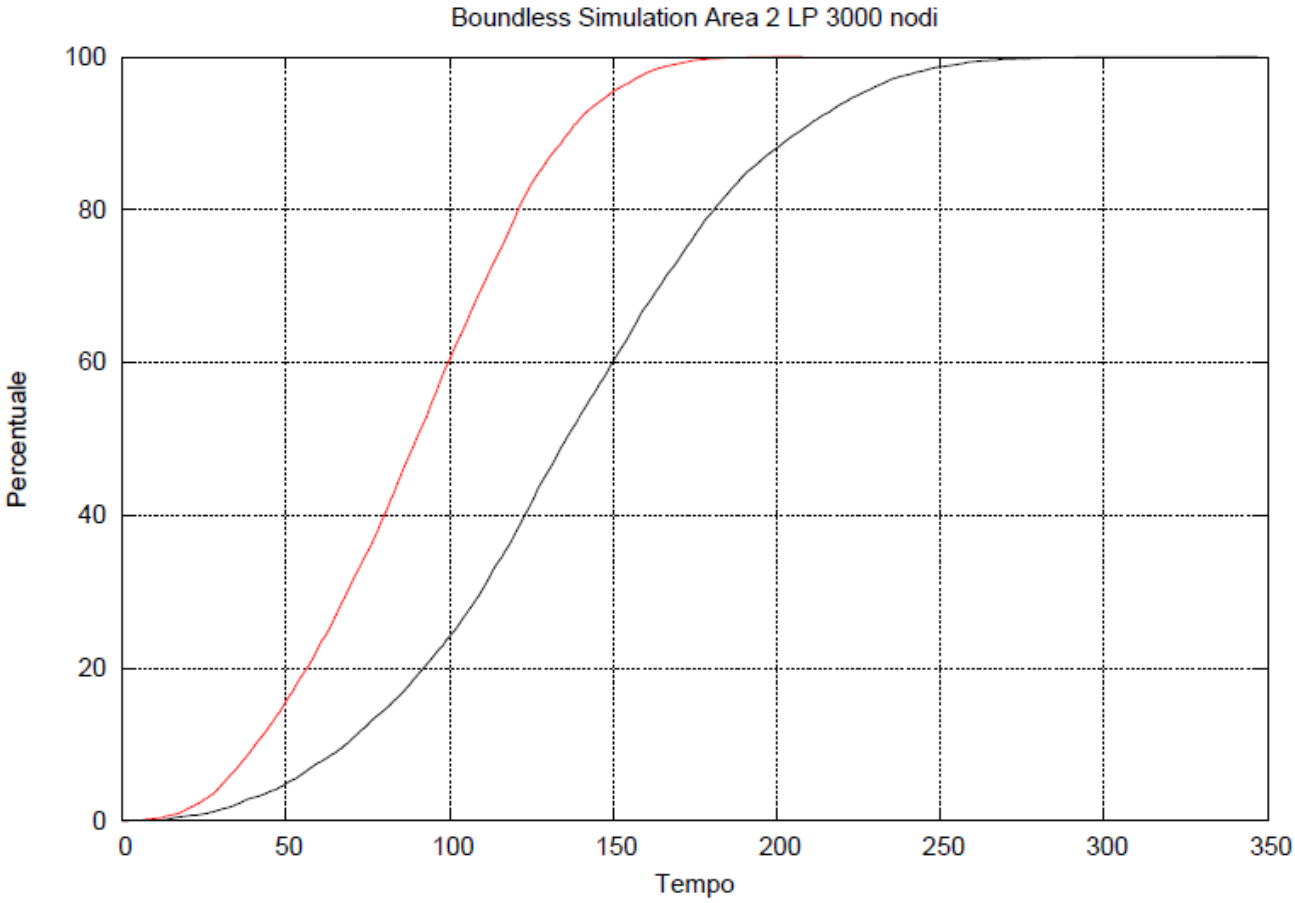
Come nei modelli precedenti, eccetto il City Section, si può notare la fase iniziale, dove l'infezione vera e propria inizia a partire circa dallo step 50, dopodiché aumenta esponenzialmente.
La simulazione termina mediamente attorno allo step di tempo 280, e confrontandolo con gli altri modelli di mobilità, è il secondo modello più lento, dopo il Gauss-Markov.

Gli intervalli di confidenza calcolati per le percentuali sono i seguenti:
step 60: [5.52; 9.69]
step 90: [15.38; 2.51]
step 120 [31.9; 43.19]
step 150: [52.91; 67.32]
step 180: [71.94; 87.24]



Si può notare anche qui rispetto al grafico precedente, come con una percentuale più alta l'infezione si diffonda più rapidamente.
Dato che la linea è una media dei valori ottenuti nelle 10 simulazioni, si prolunga fino circa allo step 300, ma alcune simulazioni sono terminate molto prima.
Come in tutti gli altri modelli, la fase iniziale termina attorno allo step di tempo 20.

Gli intervalli di confidenza calcolati sono i seguenti:
step 20: [1.03; 2.13]
step 40: [7.81; 11.28]
step 60: [19.73; 25.75]
step 80: [34.33; 45.55]
step 100: [54.82; 65.81]

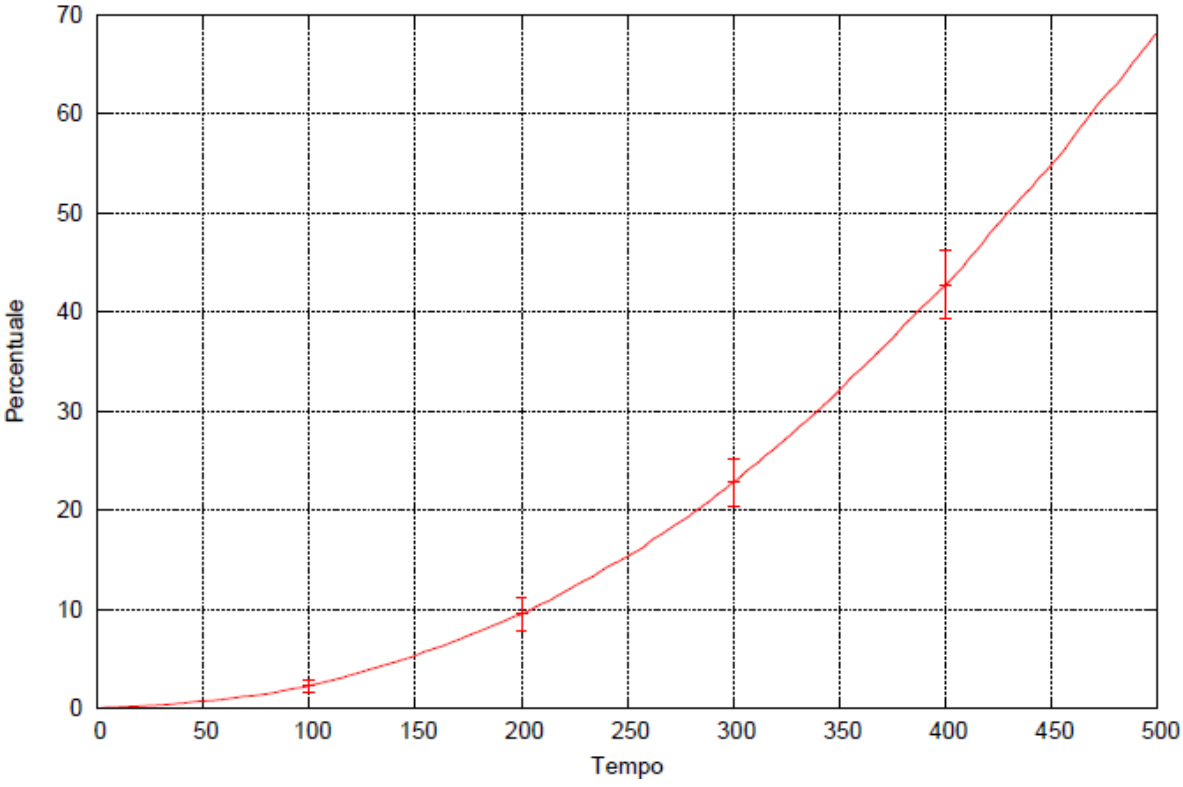


Come i grafici precedenti, la linea rossa indica le simulazioni con la percentuale del 15%, mentre la linea nera quelle con la percentuale del 10%. Si può notare la fase iniziale, come sia equivalente per le due percentuali fino allo step di tempo 20, per poi cambiare completamente negli step successivi.
Per quanto riguarda la durata delle simulazioni, sono stati calcolati i seguenti intervalli di confidenza che mettono in evidenza le differenze tra le due percentuali utilizzate:
10%: [258; 313]
15%: [172; 220]

8.2 Nodi infetti e non infetti utilizzano diversi modelli di mobilità

In questo scenario, all'inizio della simulazione un nodo si muove con un modello di mobilità diverso dagli altri nodi nel sistema. L'infezione avviene per contatto, questo significa che viene utilizzato un raggio d'azione molto inferiore a quello precedente. Inoltre in questo caso non esiste una percentuale che indica la probabilità con cui un nodo infetta il suo vicino, poichè ogni nodo sano che entra a contatto con uno infetto, viene immediatamente infettato a sua volta.
Per ogni coppia di modelli di mobilità utilizzati, sono state eseguite 10 simulazioni e sono stati calcolati gli intervalli di confidenza con un livello di confidenza al 95%.

8.2.1 Nodi non infetti: Boundless Simulation Area, nodi infetti: Random Waypoint



Nel grafico si può notare la media tra le 10 simulazioni effettuate. Dopo 500 step di tempo trascorsi, si è ottenuta in media una percentuale del 68% d'infezione. In base all'angolo scelto nei parametri per il Boundless, i nodi

si muovono da sinistra verso destra senza una destinazione. Nell'algoritmo del Random Waypoint, invece, i nodi si muovono casualmente su tutta l'area. Questo porta a un'alta percentuale d'infezione, perchè i nodi infetti si muovono su tutta l'area, aumentando la probabilità di contatto con quelli non infetti.

Per questo modello, sono stati calcolati i seguenti intervalli di confidenza, per determinati step di tempo:
step 100: [1.65; 2.91]
step 200: [7.87; 11.16]
step 300: [20.41; 25.21]
step 400: [39.25; 46.21]

Il seguente screenshot mette in evidenza come i nodi infetti (neri) si diffondono su tutta l'area, mettendo in evidenza il movimento da sinistra verso destra dei nodi non infetti (rossi). Infatti abbiamo una densità maggiore di nodi infetti sulla sinistra:

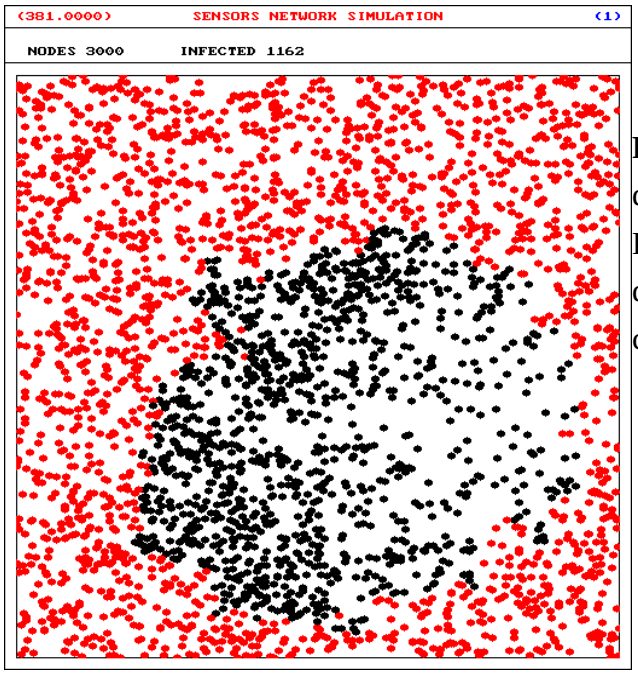
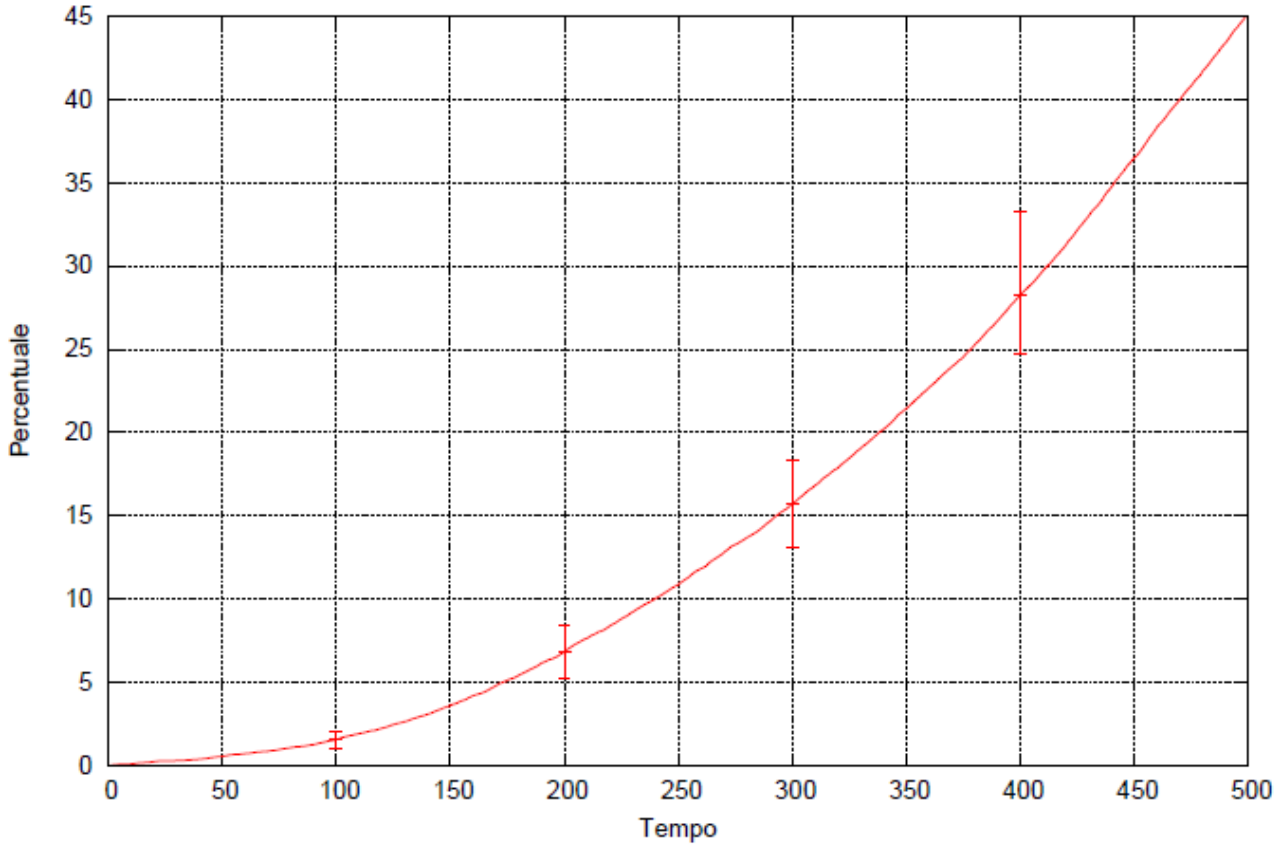


Figura 10: I nodi infetti si muovono da sinistra a destra con l'algoritmo del Boundless Simulation Area, mentre quelli infetti si muovono su tutta l'area con il Random Waypoint

8.2.2 Nodi non infetti: Boundless Simulation Area, nodi infetti: City Section



In questo caso si può notare come la percentuale raggiunta sia inferiore a quella precedente, infatti abbiamo una percentuale media del 45%. I nodi non infetti si muovono col il Boundless Simulation Area, mentre quelli infetti con il City Section. Nonostante anche con l'algoritmo del City Section, una volta infettati, i nodi si muovono su tutta l'area, abbiamo una percentuale d'infezione più bassa, poichè i nodi sono limitati nel loro movimento dalla griglia. Quindi perché un nodo venga infettato deve sia passare vicino a una strada, che entrare in contatto con un nodo infetto, abbassando notevolmente le probabilità d'infezione.
Gli intervalli di confidenza calcolati sono i seguenti:
step 100: [1.09; 2.10]
step 200: [5.26; 8.42]
step 300: [13.07; 18.38]
step 400: [24.70; 33.25]

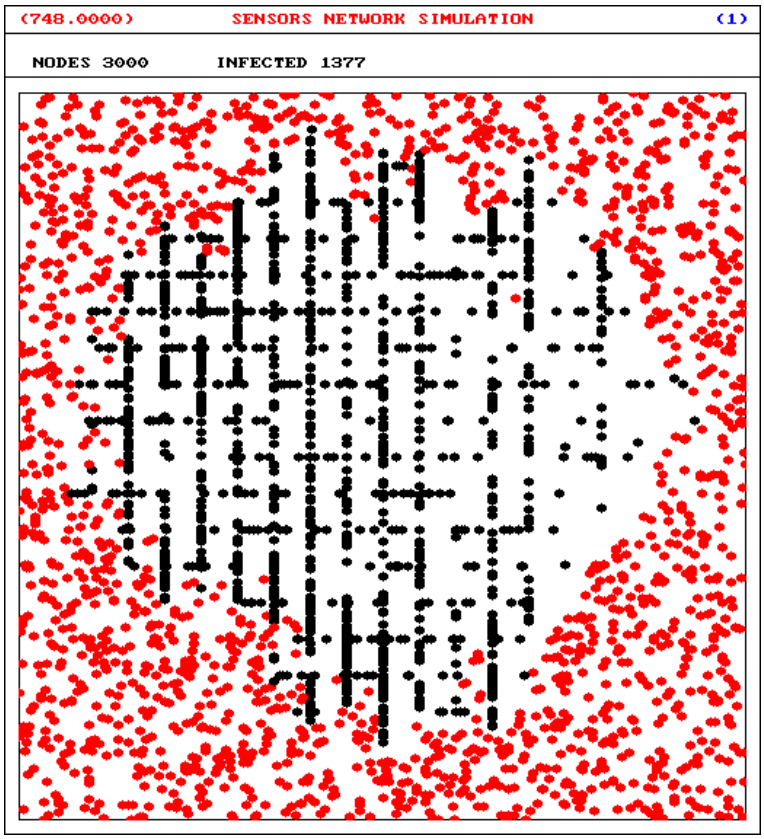
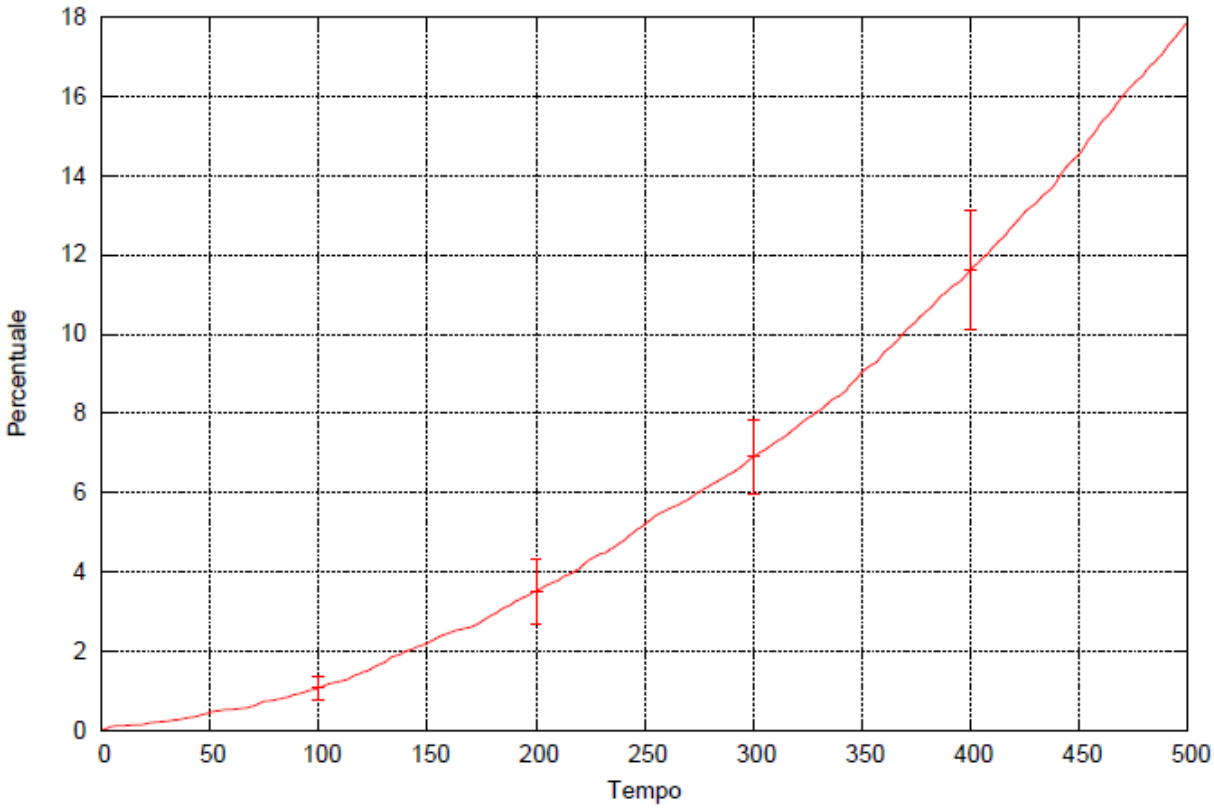


Figura 11: Si può notare la differenza di movimento tra i nodi non infetti che si muovono con l'algoritmo del Boundless Simulation Area, e quelli infetti che utilizzano il City Section.

8.2.3 Nodi non infetti: Random Waypoint, nodi infetti: Gauss-Markov



Utilizzando per i nodi non infetti il modello di mobilità RandomWaypoint, mentre per quelli infetti il Gauss Markov, si raggiunge una percentuale media del 17%. Il valore della percentuale risulta così basso, a causa del movimento dei nodi infetti. Con il modello di mobilità Gauss Markov, infatti, i nodi si muovono continuamente sull'area dall'alto verso il basso, infettando i nodi che incontrano. Col il Random Waypoint invece i nodi si muovono su tutta l'area, quindi c'è una probabilità di contagio più bassa rispetto a tutti gli altri scenari. Nello screenshot si può notare come sopra ai nodi infetti si crei del vuoto nell'area, poiché i nodi che infettano i vicini li "portano" con loro a viaggiare verso il basso.
Gli intervalli di confidenza calcolati a determinati step di tempo sono i seguenti:
step 100: [0.79; 1.38]
step 200: [2.69; 4.35]
step 300: [3.98; 7.84]
step 400: [10.10; 13.13]

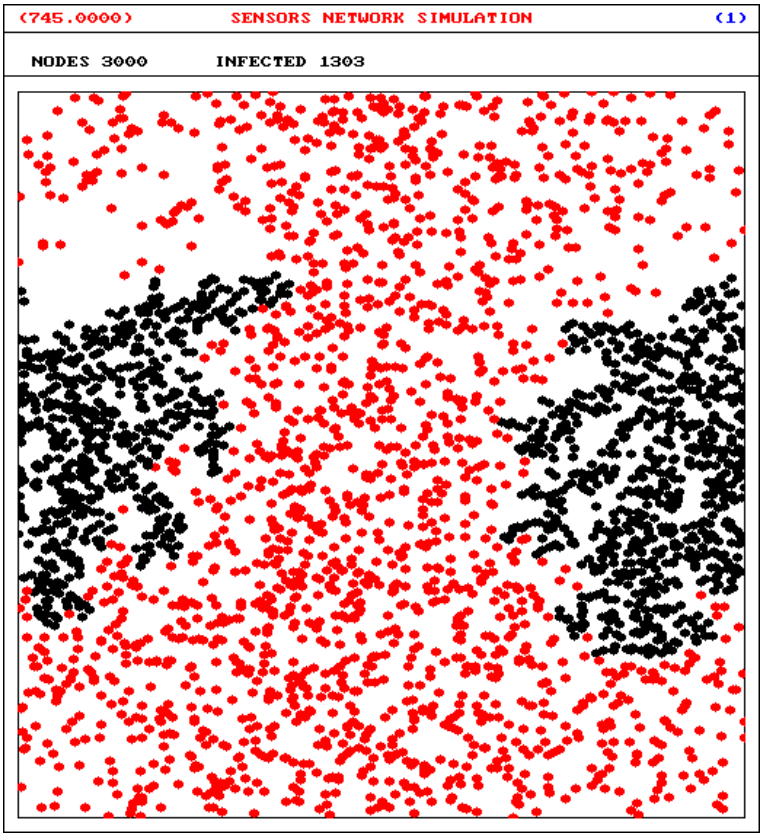
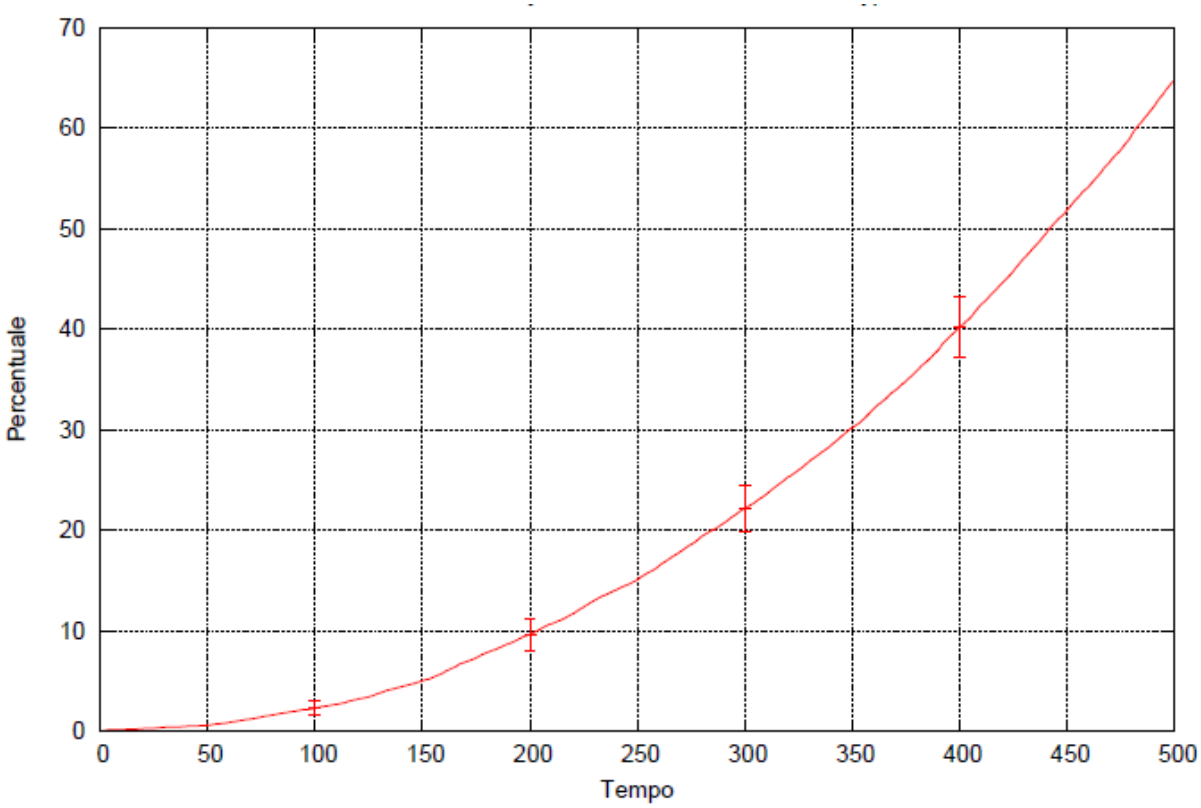


Figura 12: Si possono notare i nodi in nero (infetti) che si muovono verso il basso, creando del vuoto nella parte superiore dell'area

8.2.4 Nodi non infetti: City Section, nodi infetti: RandomWaypoint



Utilizzando questi modelli di mobilità, si può osservare una percentuale media al termine dei 500 step del 64%.

La percentuale ottenuta è molto simile a quella del primo scenario utilizzato, dove i nodi non infetti si muovevano con il Boundless Simulation Area, mentre quelli infetti con il Random Waypoint. Questo accade perché il modello che infetta è lo stesso, quindi i nodi si muovono su tutta l'area entrando in contatto facilmente con gli altri nodi sani del sistema. Tuttavia si ottiene una percentuale media lievemente più bassa, perché anche con l'algoritmo del City Section i nodi si muovono su tutta l'area, e può succedere che si allontanino dai nodi infetti. Inoltre muovendosi su delle strade, si hanno delle porzioni vuote dell'area dove non ci sono nodi da infettare.

Gli intervalli di confidenza calcolati per questo modello, sono i seguenti:

step 100: [1.61; 2.98]

step 200: [8.04; 11.20]

step 300: [19.84; 24.44]

step 400: [37.22; 43.20]

Nel seguente screenshot si può osservare la differenza di movimento tra nodi infetti e non infetti:

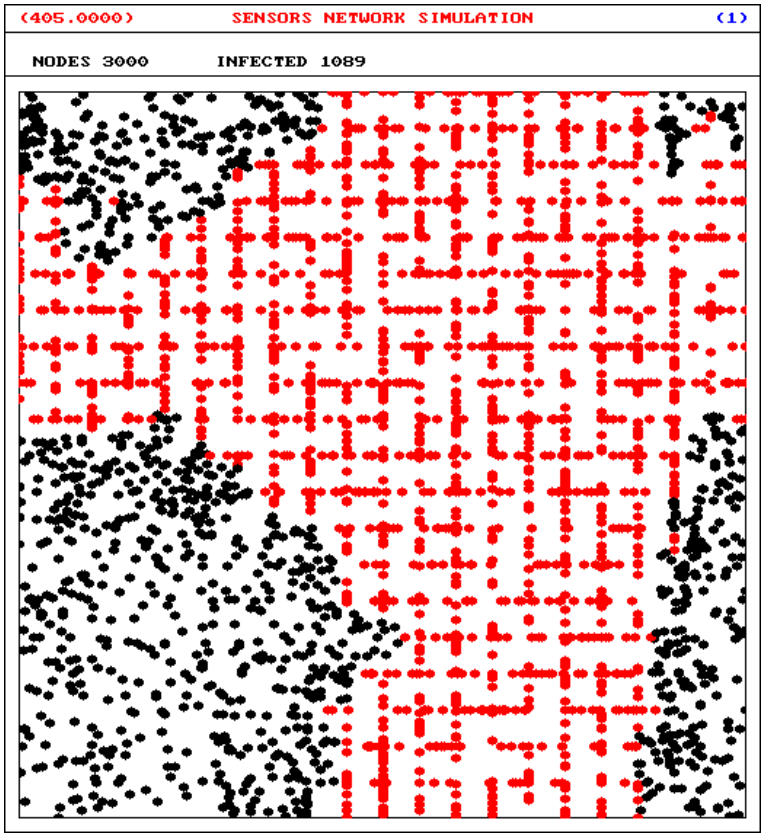


Figura 13: I nodi non infetti si muovono con il City Section, mentre i nodi infetti con il Random Waypoint

8.3 Nodi suddivisi in classi

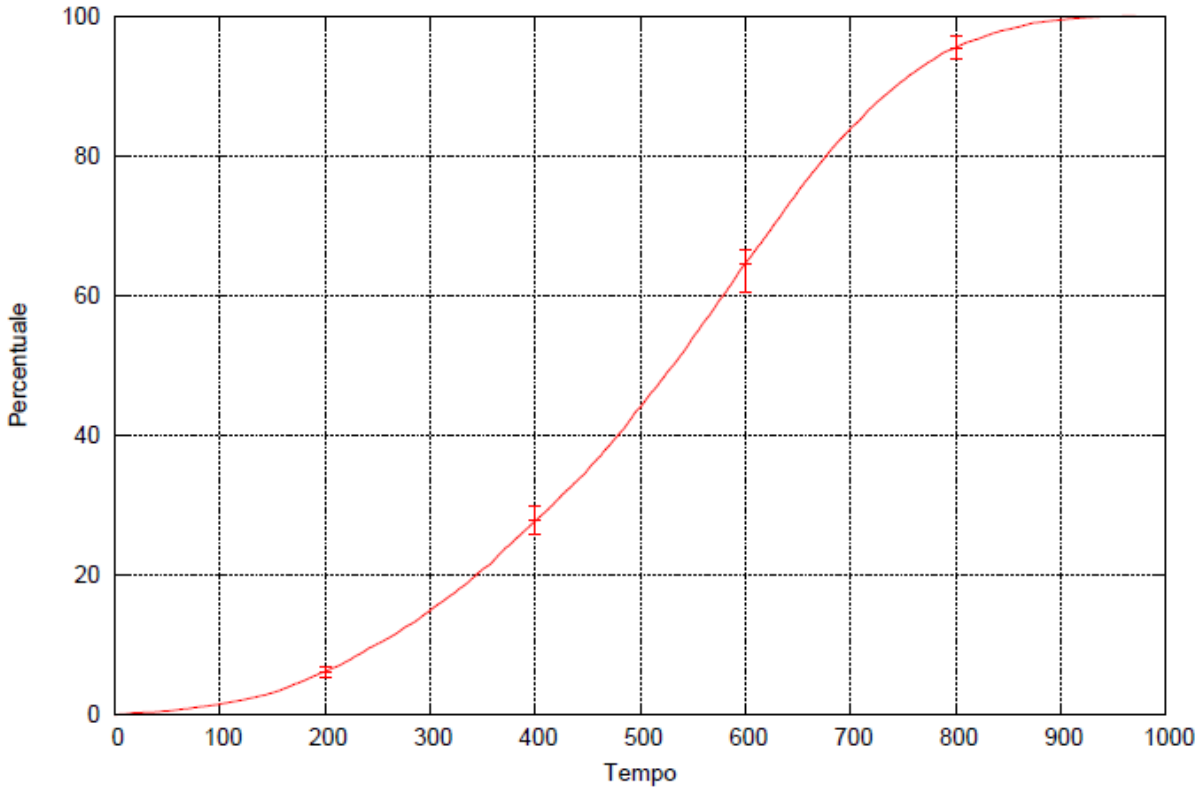
Un ultimo studio sui modelli di mobilità, e su come i nodi si muovono nel sistema e interagiscono tra loro, è stato eseguito suddividendo i nodi in classi.

All'inizio della simulazione, quando i nodi vengono registrati, vengono suddivisi casualmente in classi, che differiscono per modello di mobilità o per parametri del modello utilizzato. Dopodiché i nodi vengono infettati, ma ognuno continua a muoversi con il modello di mobilità scelto inizialmente. Per effettuare i test relativi a questo modello, si è cercato di studiare degli scenari che potessero raffrontarsi con dei sistemi reali.

In tutti gli scenari studiati, l'infezione avviene per contatto, con un raggio di 100 unità e il programma termina quando sono passati 1000 step di tempo, o se la percentuale raggiunge il 100%.

8.3.1 Suddivisione in due classi: City Section e Random Waypoint

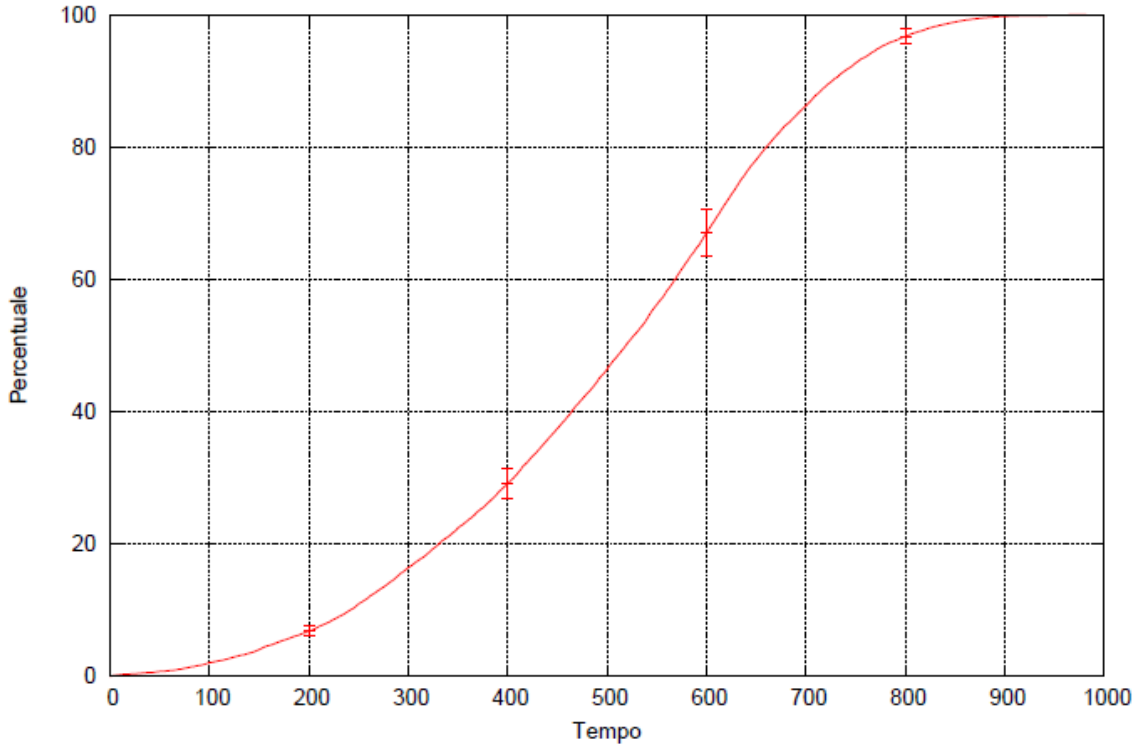
Quando all'inizio della simulazione avviene la suddivisione in classi, i nodi vengono suddivisi in quelli che utilizzano il City Section, e quelli che utilizzano il Random Waypoint. Questo modello è stato sviluppato pensando a una città in cui i pedoni (utilizzando il Random Waypoint) si muovono casualmente, mentre i veicoli (che utilizzano il City Section), sono obbligati a muoversi su delle strade. Il meccanismo d'infezione, può essere visto come un virus all'interno della rete, che si diffonde nei cellulari o dispositivi wireless, delle varie entità che entrano in contatto tra loro.
Ovviamente è stata modificata la velocità con cui si muovono veicoli e pedoni, per rendere il modello il più realistico possibile.



Il grafico è il risultato della media tra le 10 simulazioni eseguite, e si può osservare come si raggiunga l'infezione totale attorno ai 900 step di tempo. La simulazione che ha impiegato più tempo, è durata 971 step.
Si può notare dal grafico, come l'infezione cresca in modo omogeneo e abbastanza lineare.
Gli intervalli di confidenza calcolati a determinati step di tempo sono i seguenti:
step 200: [5.26; 6.97]
step 400: [25.79; 29.79]
step 600: [60.58; 66.55]
step 800: [93.96; 97.06]

8.3.2 Suddivisione in due classi: Random Waypoint con diverse velocità

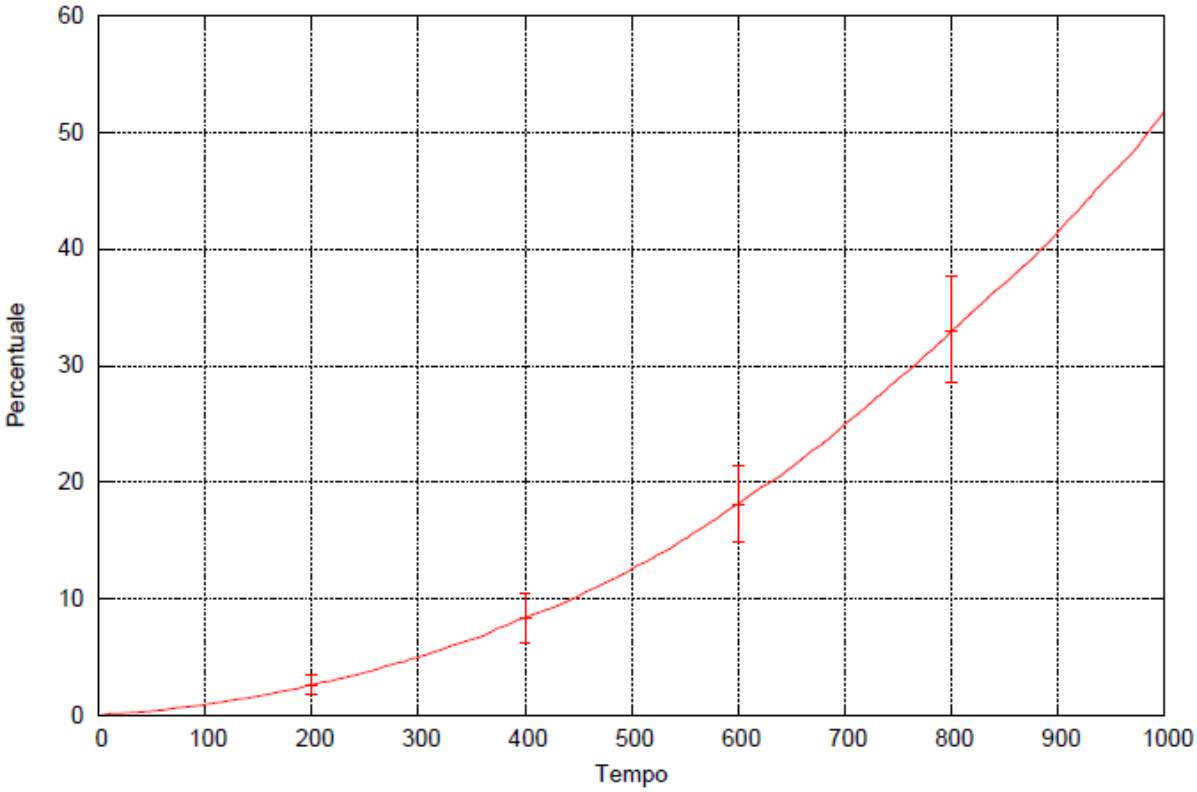
Quando si è implementato questo modello, si è pensato ad un parco in cui si muovono persone a piedi e in biciletta. Quindi entrambe le classi utilizzano lo stesso modello di mobilità, ma con velocità diverse. In particolare per le persone a piedi si è utilizzata una velocità di 5km/h, mentre per le persone in bicicletta 15km/h.



Dal grafico si può notare come l'infezione totale avvenga anche qui attorno ai 900 step di tempo, quindi si può affermare che questo scenario e quello precedente hanno un esito molto simile. Questo accade perché, sia che i nodi si muovano con il City Section e Random Waypoint, sia che si muovano entrambi con il Random Waypoint ma con velocità diverse, i nodi si dispongono su tutta l'area e si muovono in tutte le direzione all'interno di essa. Per questo, in poco tempo, è molto probabile che tutti entrino in contatto tra loro e vengano infettati.
Gli intervalli di confidenza calcolati per i valori delle percentuali sono i seguenti:
step 200: [6.02; 7.56]
step 400: [26.84; 31.41]
step 600: [63.56; 70.55]
step 800: [95.60; 97.86]

8.3.3 Suddivisione in due classi: Boundless Simulation Area e Gauss Markov

Questo scenario è stato creato per studiare cosa succederebbe se una parte dei nodi si muovessero con l'algoritmo del Boundless Simulation Area e l'altra parte con il Gauss-Markov.
Nei paragrafi precedenti è stato osservato che la simulazione terminava allo stesso modo utilizzando modelli di mobilità che permettono ai nodi di muoversi su tutta l'area e in tutte le direzioni. Con questo modello, dove i nodi si muovono sempre nella stessa direzione, è stato rilevato un risultato finale completamente diverso.
Questo perché, in base alle direzioni impostate inizialmente, con il Boundless Simulation Area i nodi si muovono da sinistra verso destra, mentre nel Gauss-Markov dall'alto verso il basso, causando un minore contatto tra i nodi, e di conseguenza una minore infezione.



In questo grafico si può appunto osservare come al termine dei 1000 step, sia stata superata da poco la percentuale del 50%.

L'infezione infatti, utilizzando questi due modelli, avviene molto più lentamente perché i nodi entrano molto meno in contatto tra loro.

Gli intervalli di confidenza calcolati risultano i seguenti:

step 200: [1.86; 3.46]

step 400: [6.24; 10.55]

step 600: [14.92; 21.38]

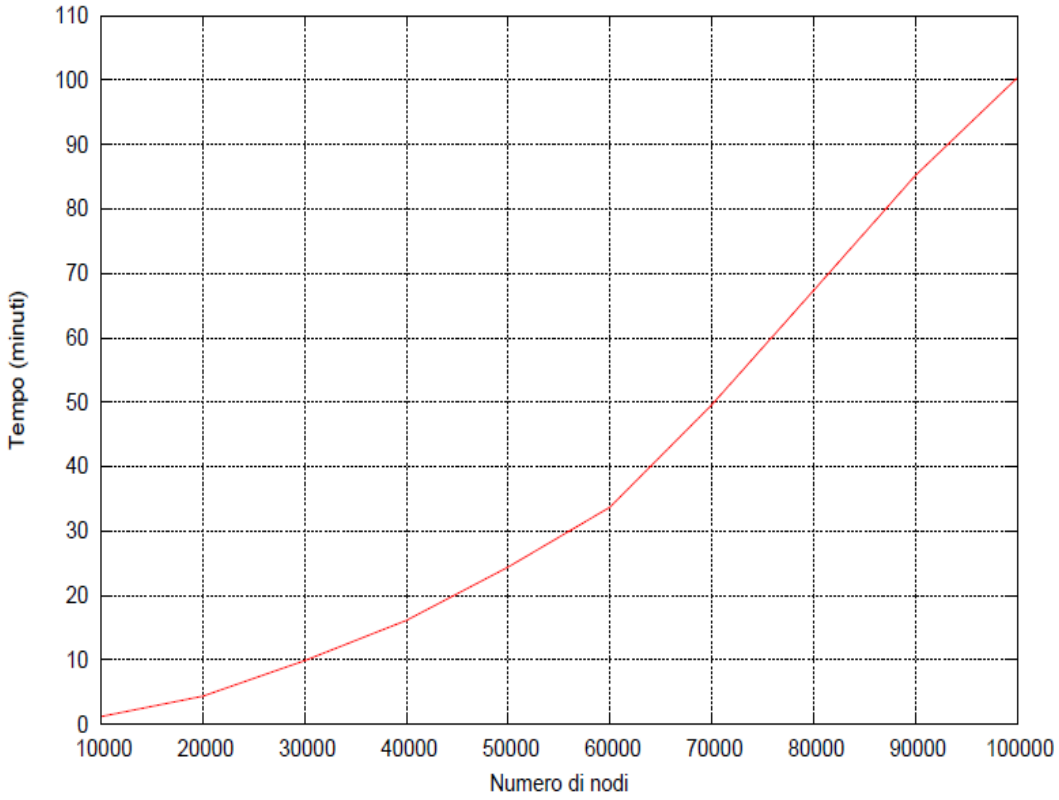
step 800: [28.53; 37.63]

CAPITOLO 9: PRESTAZIONI DEL SIMULATORE

Per verificare le prestazioni del simulatore, è stato utilizzato cassandra, un nodo del cluster di simulazione. Un cluster è un insieme di computer connessi tra loro che permettono di svolgere simulazioni parallele e distribuite. Cassandra è un terminale dual core Hyper-Threading, questo significa che per ogni processore ne vengono simulati due. Per questo è stato possibile utilizzare 4 processi per gestire i nodi.
Il programma è stato modificato per terminare la simulazione dopo 100 step di tempo, in modo che le simulazioni non risultassero troppo lunghe.
Per ogni modello di mobilità implementato sono state eseguite 10 simulazioni con diverse quantità di nodi, mantenendo però la stessa densità all'interno dell'area di simulazione. Ogni volta che il numero di nodi aumentava, infatti, venivano aumentati proporzionalmente i valori di MAX_X e MAX_Y nel file wireless.ini, che definiscono la grandezza dell'area.
Il numero di nodi simulati parte da 10000 per arrivare a 100000 e per ogni simulazione sono stati usati 4 LP (Logical Process).
Inoltre, per verificare al meglio le prestazioni del simulatore, sono state effettuate 10 simulazioni per ogni modello sia con la migrazione di GAIA abilitata che disabilitata.
Nei grafici seguenti si possono osservare il tempo (in minuti) della durata della simulazione sull'asse Y, mentre sull'asse X il numero di nodi utilizzati per la simulazione.

I grafici sono stati creati con il programma "gnuplot", utilizzando appunto la colonna del tempo e dei nodi.

9.1 Random Waypoint



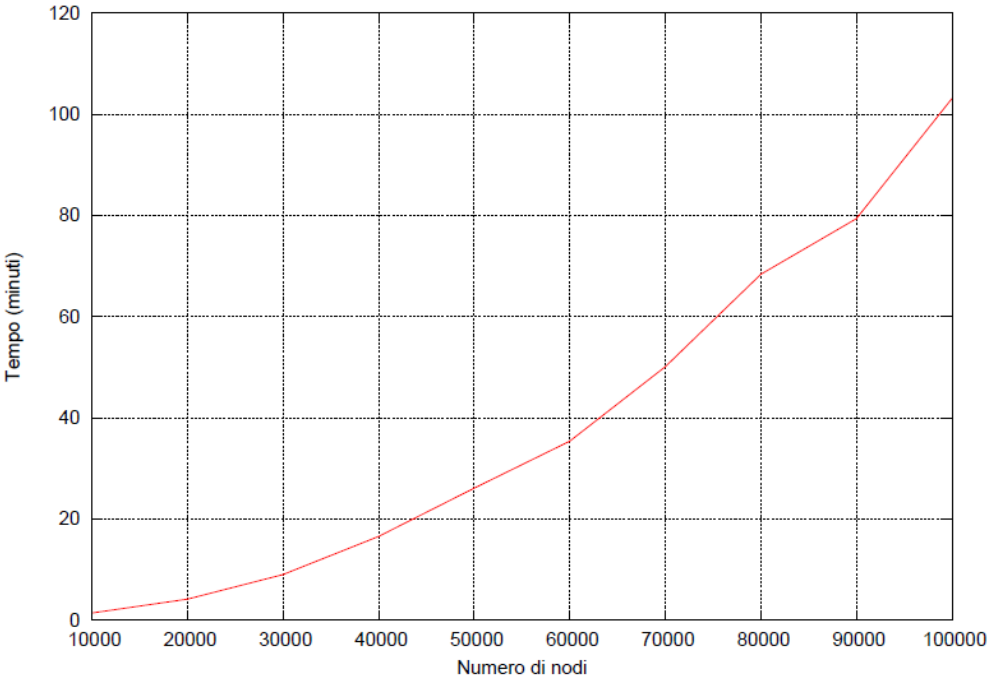
Tempo	Nodi
1.2	10000
4.37	20000
9.9	30000
16.13	40000
24.43	50000
33.7	60000
49.59	70000
67.34	80000
85.15	90000
100.2	100000

Questo grafico è il risultato di 10 prove effettuate con la migrazione abilitata: ha un andamento lineare poichè i tempi delle simulazioni all'aumentare dei nodi, incrementano in modo omogeneo. Anche la differenza tra i tempi di coppie di simulazioni consecutive cresce all'aumentare dei nodi in modo lineare, senza sbalzi particolarmente accentuati.

Ad esempio se tra 10000 e 20000 nodi abbiamo una differenza di 3.17 minuti, la differenza tra i tempi di successive coppie di simulazioni, è sicuramente maggiore.

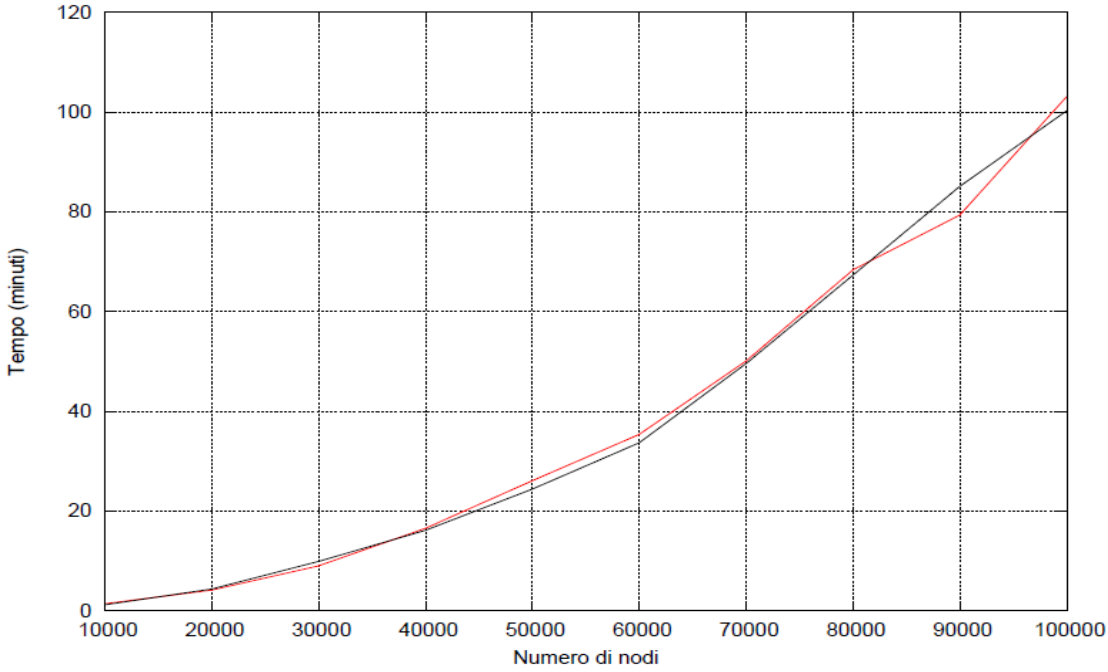
Già solo tra 20000 e 30000 la differenza è di 5.53 minuti. Se le differenze tra i tempi fossero state più accentuate sia in diminuzione che in aumento, avremmo ottenuto un grafico più disomogeneo.

Con la migrazione disabilitata, è stato ottenuto il seguente risultato:



Tempo	Nodi
1.4	10000
4.16	20000
9.0	30000
16.56	40000
26.11	50000
35.39	60000
50.10	70000
68.41	80000
79.39	90000
103.14	100000

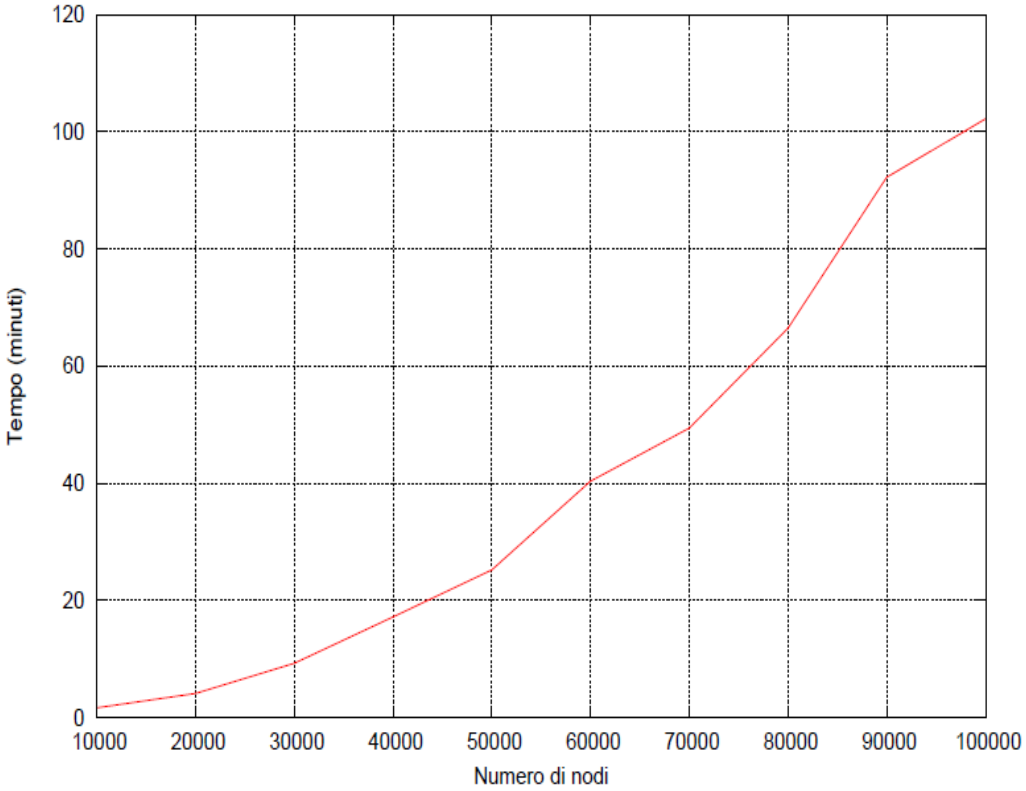
Si può facilmente osservare, come con la migrazione disabilitata, non ci siano differenze considerevoli all'interno del grafico.
È stato ottenuto un grafico lineare come il precedente, gli unici salti che si rilevano, sono in prossimità degli 80000 e 90000 nodi.
Questo perché tra 70000 e 80000 abbiamo una differenza nei tempi di 18 minuti, mentre tra 80000 e 90000, la differenza è molto inferiore, di 11.
Successivamente tra 90000 e 100000 la differenza tra i tempi continua a crescere, arrivando a 24 minuti. Questo causa un innalzamento veloce della linea del grafico.



In questo grafico, sono messe a confronto la migrazione attivata (linea nera) e disattivata (linea rossa).

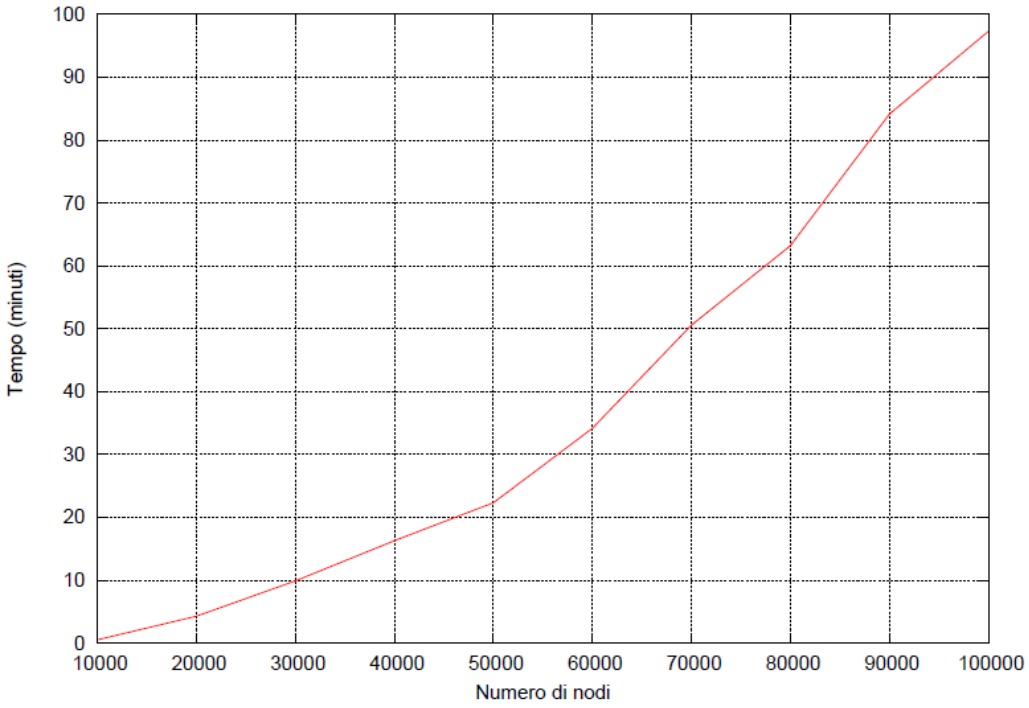
Come si può osservare, le linee hanno un andamento molto simile. Si poteva pensare che con la migrazione abilitata, i tempi di simulazione sarebbero stati più lunghi, a causa dello spostamento di nodi da un processo ad un altro. Tuttavia è stato rilevato che la migrazione non influenza in modo significativo l'andamento della simulazione, anzi, il tempo con cui terminano 100000 nodi con la migrazione attivata, è di 100 minuti, mentre con la migrazione disattivata è di 103. Questo scarto può essere dovuto a un ritardo nella rete durante la simulazione, ma indica maggiormente che utilizzando o meno la migrazione, non si influenza il risultato finale.

9.2 City Section



Tempo	Nodi
1.7	10000
4.13	20000
9.29	30000
17.2	40000
25.24	50000
40.36	60000
49.42	70000
66.48	80000
92.24	90000
102.2	100000

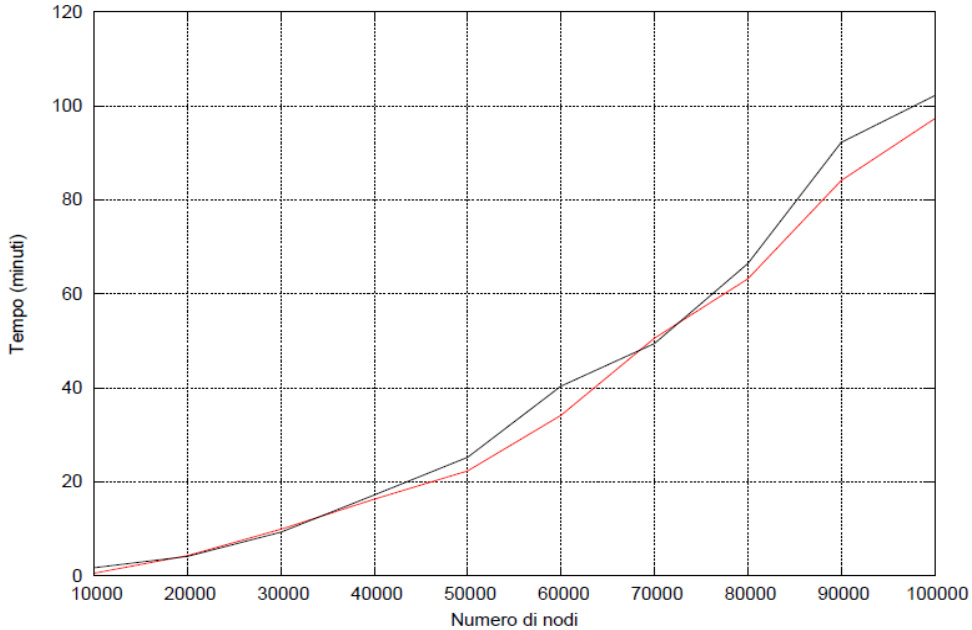
Questo grafico indica i tempi ottenuti con la migrazione attivata.
In questa serie di simulazioni il risultato è diverso dallo scenario precedente poiché sono state riscontrate differenze disomogenee tra i tempi di coppie di simulazioni consecutive.
Ad esempio tra 50000 e 60000 nodi abbiamo una differenza di tempo di 15.2 minuti, e ci si aspetta che la differenza tra 60000 e 70000 sia maggiore o comunque attorno a questo valore invece la differenza tra i tempi di queste due simulazioni è di 9.06 minuti. Nel grafico si può quindi notare come la linea crei uno sbalzo. Successivamente tra la coppia di simulazioni di 70000 e 80000 nodi si ottiene una differenza tra i tempi di 17.6 minuti, quindi il grafico continua a crescere.



Tempo	Nodi
0.54	10000
4.3	20000
9.9	30000
16.3	40000
22.33	50000
34.17	60000
50.58	70000
63.22	80000
84.13	90000
97.3	100000

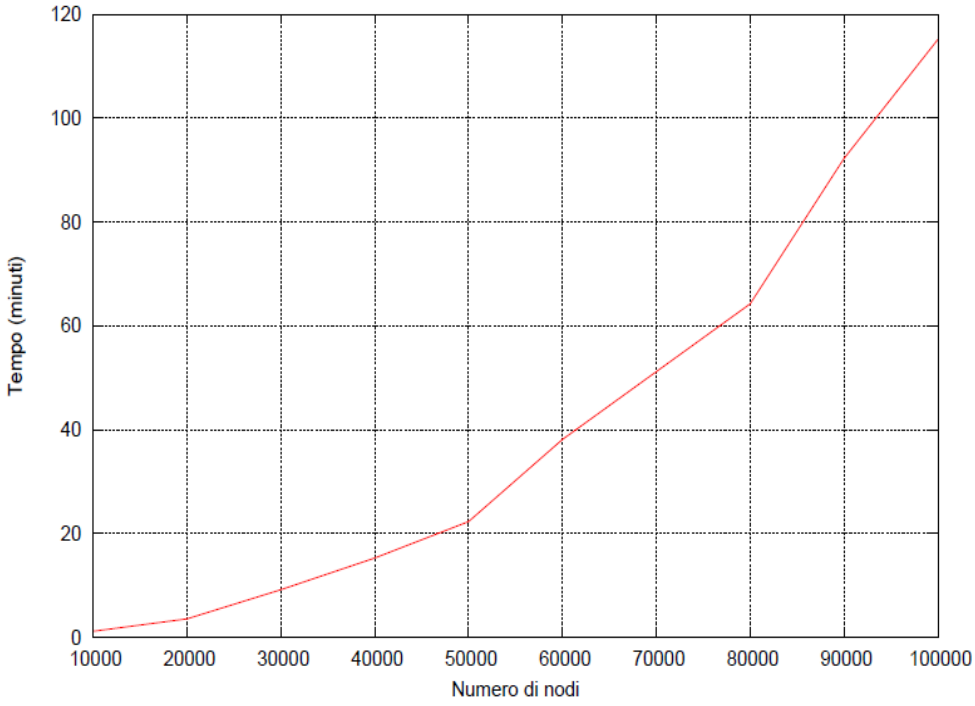
In questo grafico possiamo osservare l'andamento dei tempi con la migrazione disabilitata.
Dai 50000 nodi in poi, il grafico cresce velocemente, poiché ci sono grandi differenze tra i tempi di coppie di nodi successivi. Infatti il grafico ha qualche salto poco rilevante, poiché dopo i 50 nodi, la differenza tra i tempi delle simulazioni rimane tra i 12 e 16 minuti, facendo eccezione tra gli 80000 e 90000 dove la differenza è di ben 21 minuti.

Il seguente grafico, mette a confronto la migrazione abilitata (linea nera) e la migrazione disabilitata (linea rossa).



Come nel capitolo precedente, nel caso del Random Waypoint, non ci sono differenze considerevoli tra i due grafici. Una cosa che si può notare in questo caso, a differenza del precedente, è che i tempi con la migrazione disabilitata, siano quasi sempre inferiori. In questo caso si può dire che disabilitando la migrazione sono aumentate le prestazioni, seppur non in modo considerevole.

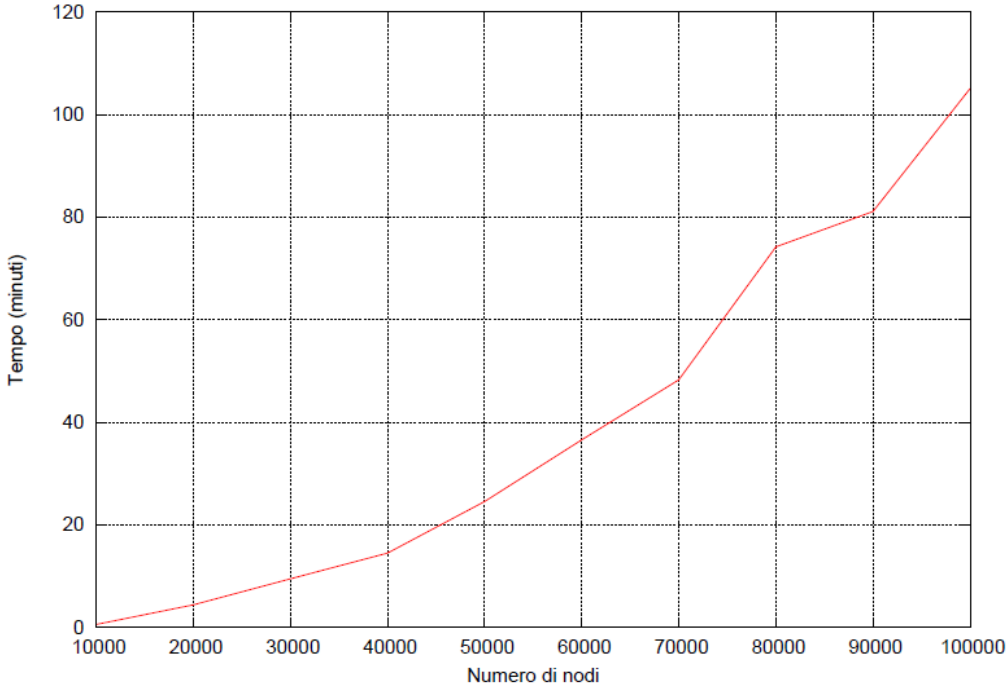
9.3 Gauss-Markov



Tempo	Nodi
1.2	10000
3.59	20000
9.2	30000
15.32	40000
22.32	50000
38.12	60000
51.19	70000
64.24	80000
92.22	90000
115.14	100000

Questo grafico indica i risultati ottenuti con la migrazione attivata.
Come si può subito osservare, questo modello di mobilità quando viene utilizzato con un gran numero di nodi, impiega molto più tempo degli altri modelli implementati a terminare la simulazione.
Al contrario degli altri modelli di mobilità che, utilizzando 100000 nodi, terminano la simulazione attorno ai 100 minuti, questo modello impiega quasi 2 ore. Questo può essere dovuto a un ritardo nella rete, o all'alto numeri di calcoli che il modello Gauss Markov effettua per il movimento dei nodi, causando un ritardo nella simulazione.
Nel grafico si possono notare due momenti in cui la linea subisce uno sbalzo. Tra i tempi delle coppie 40000-50000 e 50000-60000 c'è una

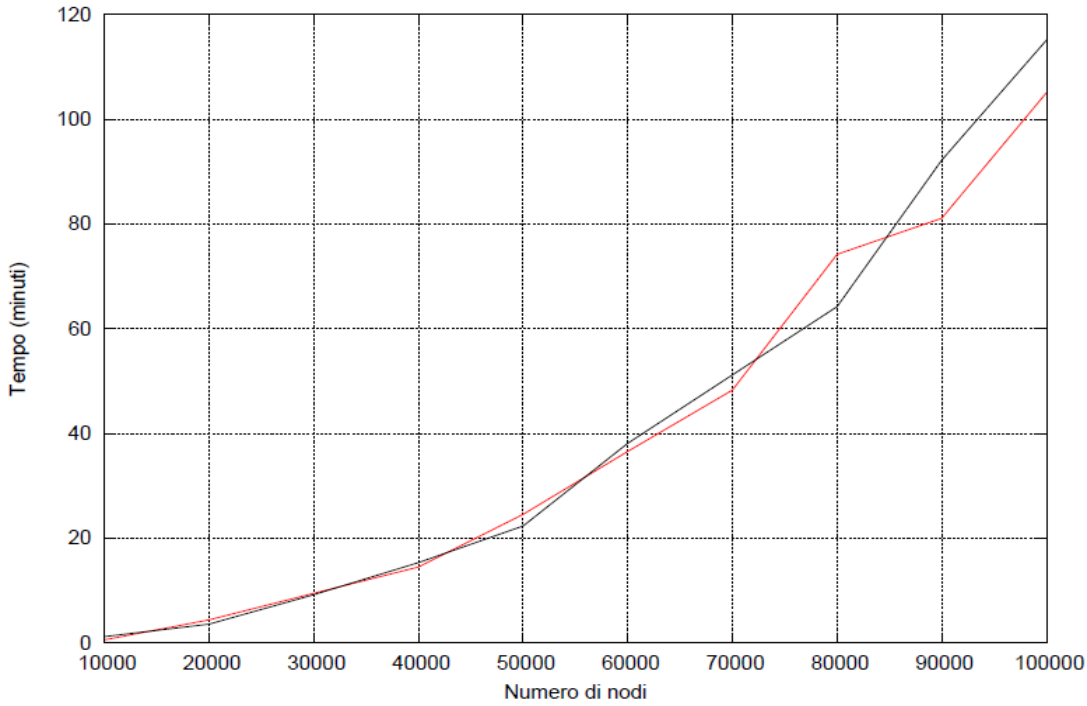
differenza abbastanza ampia che causa l'angolo nella linea. La prima coppia infatti ha una differenza nei tempi di 7 minuti, mentre la seconda ha una differenza di 15.8 minuti. I tempi tornano a crescere in modo lineare fino al secondo sbalzo che si verifica negli 80000 nodi. Nelle simulazioni con 70000-80000 nodi infatti si ha una differenza di tempi di 13,05 minuti, mentre nella coppia di simulazioni successive, 80000-90000 nodi, si ha una differenza di tempo di 27.90 minuti, che causa l'angolazione della linea verso l'alto.



Tempo	Nodi
0.58	10000
4.4	20000
9.48	30000
14.5	40000
24.5	50000
36.57	60000
48.29	70000
74.24	80000
81.12	90000
105.11	100000

In questo grafico è disegnata la linea dell'andamento della simulazione all'aumentare dei nodi, con la migrazione disattivata.

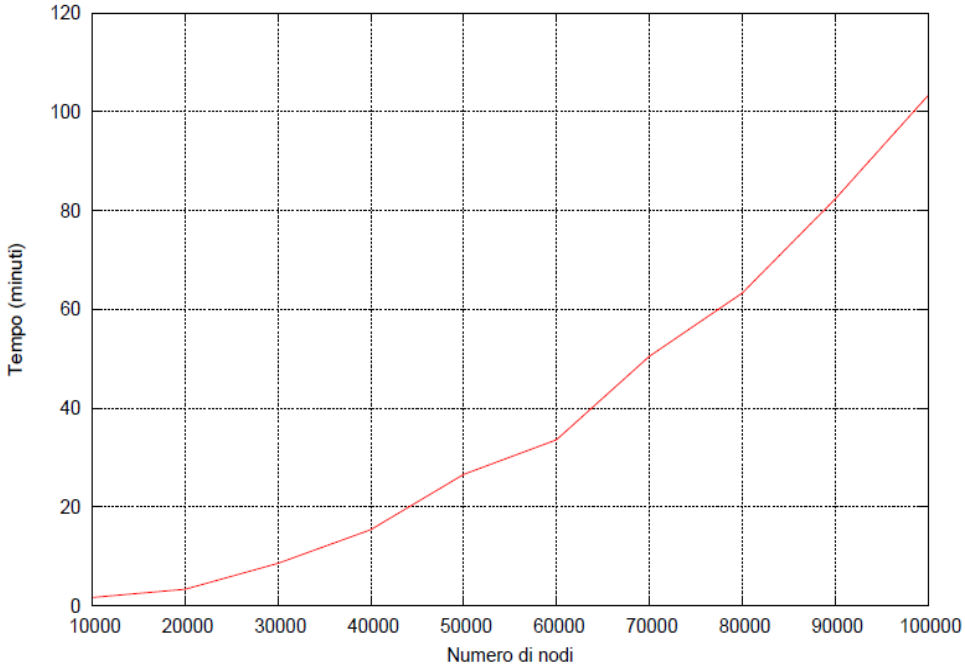
Si può notare un picco notevole in prossimità degli 80000 nodi, questo perché c'è una grande differenza tra i tempi nel passare da 70000 a 80000, e da 80000 a 90000. Passando da 70000 a 80000, infatti, abbiamo una differenza di tempo di 26 minuti, mentre passando da 80000 a 90000 la differenza è di soli 7 minuti. Anche in questo caso, l'algoritmo del Gauss-Markov, utilizzando il maggior numero di nodi, è stato il modello di mobilità che ha impiegato più tempo a terminare la simulazione. Si può affermare, quindi, che questo è dovuto ai calcoli complessi che deve effettuare il modello ogni volta, per muovere i nodi all'interno dell'area.



In questo grafico dove vengono messe a confronto la migrazione abilitata (linea nera) e disabilitata (linea rossa), si può notare che, al contrario dell'algoritmo del City Section, non c'è uno scenario che impiega minor o maggior tempo nella simulazione.

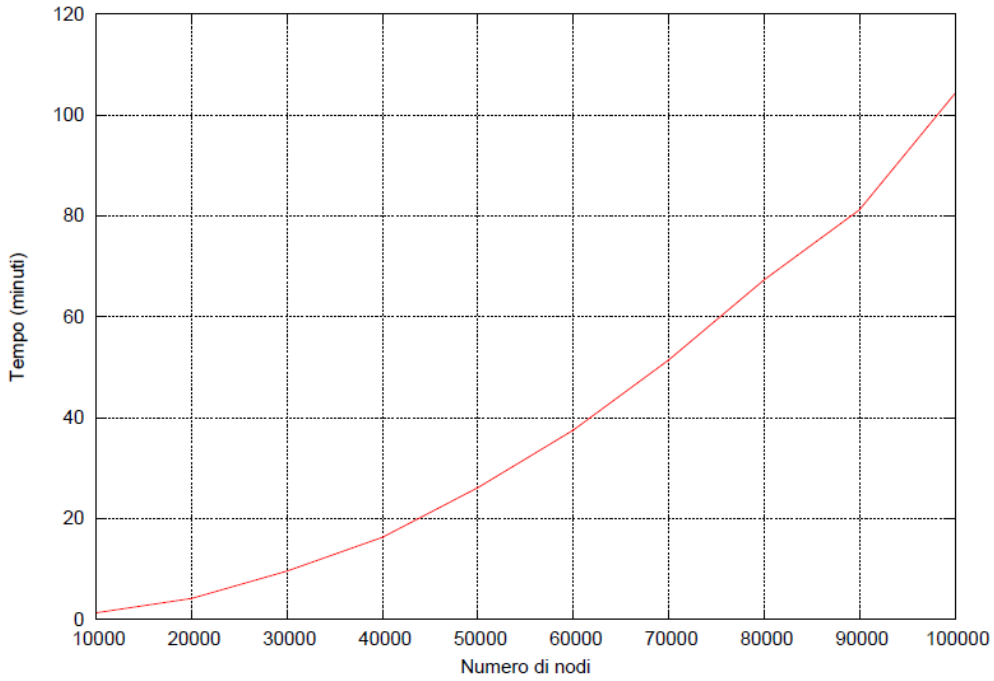
Infatti fino all'utilizzo di 70000 nodi, le linee sono pressoché uguali, dopodiché all'aumentare dei nodi, viene prima impiegato più tempo con la migrazione disattivata, dopodiché con la migrazione attivata.

9.4 Boundless Simulation Area



Tempo	Nodi
1.7	10000
3.38	20000
8.59	30000
15.42	40000
26.6	50000
33.6	60000
50.50	70000
63.30	80000
82.27	90000
103.2	100000

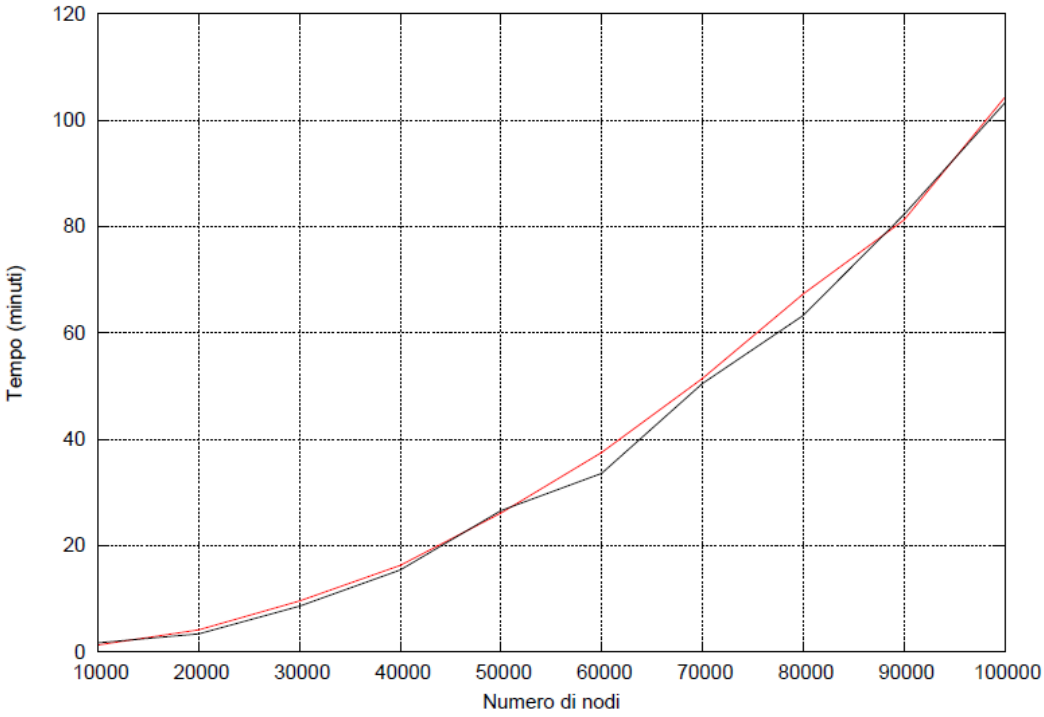
In questo grafico è rappresentata la linea delle simulazioni con la migrazione disattivata.
Questo grafico cresce in modo abbastanza lineare come quello del Random Waypoint, poichè le differenze riscontrate tra tempi di coppie di simulazioni consecutive sono abbastanza omogenee.
Si può notare una lieve increspatura tra 50000 e 70000 nodi dovuta a una differenza tra i tempi delle simulazioni di 40000-50000 nodi di 11.18 minuti, e una differenza tra i tempi delle simulazioni di 50000-60000 nodi di 7 minuti, che causa una specie di stallo nel grafico, per poi risalire nell'intervallo successivo. Infatti la differenza tra i tempi delle simulazioni di 60000-70000 nodi è di 16.9 e la linea continua a salire linearmente senza subire altri sbalzi.



Tempo	Nodi
1.3	10000
4.14	20000
9.59	30000
16.29	40000
26.13	50000
37.51	60000
51.45	70000
67.33	80000
81.24	90000
104.3	100000

In questo grafico si può osservare la linea delle simulazioni con la migrazione disabilitata.
Il grafico ha un andamento completamente lineare, infatti i tempi crescono in modo uniforme da 10000 a 100000 nodi utilizzati.
Questo modello, dopo il Gauss-Markov, è quello che impiega più tempo a terminare la simulazione, con 104 minuti trascorsi utilizzando 100000 nodi.

Nel seguente grafico si possono osservare a confronto la linea della simulazione con la migrazione abilitata (nera) e disabilitata (rossa).



Se come modello di mobilità viene utilizzato il Boundless Simulation Area, le differenze tra l'uso o meno della migrazione, sono ancor meno evidenti. Le due linee infatti si assomigliano molto, terminando addirittura quasi nello stesso punto.

Con l'utilizzo di 60000, 70000 e 80000 nodi, si può osservare che è stato impiegato meno tempo disabilitando la migrazione, anche se con una differenza di massimo 4 minuti. Dai 90000 nodi in poi, il tempo torna ad essere simile per entrambe le linee.

CAPITOLO 10: CONCLUSIONI

Nel complesso, osservando tutti gli scenari presi in analisi col modello a infezione, si possono trarre le seguenti conclusioni.
Nel caso i nodi infetti e non infetti si muovano con lo stesso modello di mobilità, all'aumentare della percentuale aumenta il tempo di simulazione, tuttavia non viene influenzato il comportamento del modello di mobilità utilizzato. Infatti il City Section, sia con la percentuale del 15%, sia con la percentuale del 10%, risulta il modello che impiega meno tempo a terminare la simulazione.
Nello stesso scenario, il modello più lento, invece, a parità di percentuale con gli altri, è il Gauss Markov, seguito dal Boundless Simulation Area, ed infine dal Random Waypoint.
Osservando la fase iniziale di tutti i modelli di mobilità con la percentuale del 10%, l'infezione inizia ad aumentare in modo esponenziale dopo lo step di tempo 50 nel Gauss-Markov, Boundless Simulation Area e RandomWaypoint. Nel City Section, invece, la fase iniziale termina attorno allo step di tempo 40. Questo avviene per il discorso fatto precedentemente, poiché il City Section è il modello che anche impiegherà meno tempo a raggiungere la percentuale totale.
Nel caso la percentuale sia il 15% invece, è stato osservato che tutti i modelli di mobilità terminano la fase iniziale e iniziano ad infettare maggiormente il sistema a partire dallo step di tempo 20.

Nello scenario in cui i nodi infetti e non infetti si muovono con modelli di mobilità differenti, è stato osservato che si ottiene una percentuale d'infezione più alta quando il modello di mobilità con il quale si muovono i nodi infetti è il Random Waypoint. Infatti i nodi si muovono su tutta l'area e in tutte le direzioni, entrando maggiormente in contatto con gli altri nodi sani.
Se come modello di mobilità per i nodi infetti viene utilizzato il City Section, abbiamo una percentuale inferiore, poiché nonostante si muovano su tutta l'area i nodi infetti hanno il movimento limitato da strade, di

conseguenza quelli sani per entrare in contatto con loro, devono passare in prossimità della griglia.
La percentuale inferiore d'infezione al termine dei 500 step, è stata riscontrata quando i nodi infetti si muovevano con il modello di mobilità Gauss Markov, poiché i nodi una volta infettati iniziano a muoversi sempre nella stessa direzione, con una bassa probabilità di entrare in contatto con i nodi sani. Poiché il movimento del Boundless Simulation Area assomiglia molto al movimento del Gauss-Markov, (i nodi non hanno destinazione, e viene impostata una direzione iniziale), si può dedurre che se venisse utilizzato il Boundless Simulation Area al posto del Gauss-MArkov, lo scenario finale d'infezione sarebbe molto simile.

Nell'ultimo modello studiato, nel quale i nodi venivano suddivisi in classi, sono stati osservati 3 scenari diversi.
Nel complesso di può dedurre che, se uno dei modelli utilizzati dalle classi è un modello che si muove su tutta l'area e in tutte le direzioni (come il City Section e il Random Waypoint), la simulazione termina prima dei 1000 step di tempo, poiché si raggiunge l'infezione totale del sistema. Se invece vengono utilizzati dei modelli di mobilità dove i nodi si muovono solo in una direzione (come il Boundless Simulation Area e il Gauss-Markov), l'infezione del sistema è molto bassa, poiché i nodi hanno una probabilità minore di entrare in contatto tra loro. Di conseguenza, la simulazione termina al raggiungimento dei 1000 step di tempo, senza che ci sia un infezione totale del sistema.

Analizzando i risultati ottenuti valutando le prestazioni del simulatore si possono trarre le seguenti conclusioni.
Se la migrazione viene attivata, si può affermare che l'algoritmo del Random Waypoint è il più veloce a eseguire la simulazione, poiché con un altro numero di nodi (100000) termina in 100 minuti.
Al contrario l'algoritmo del Gauss Markov risulta essere il più lento con lo stesso numero di nodi, poichè termina la simulazione in 115 minuti.
Disattivando la migrazione, i risultati cambiano leggermente.

L’algoritmo del Gauss Markov, rimane quello che impiega più tempo a portare a termine la simulazione col maggior numero di nodi, mentre l’algoritmo più veloce risulta quello del City Section, poiché impiega solamente 97 minuti.
Confrontando i grafici ottenuti dai vari modelli, con la migrazione abilitata e disabilitata, si può notare che utilizzando l’algoritmo del City Section e del Gauss Markov, all’aumentare dei nodi viene impiegato meno tempo per terminare la simulazione con la migrazione disabilitata. Con l’utilizzo dei modelli di mobilità Random Waypoint e Boundless Simulation Area, invece, l’abilitazione o meno della migrazione, non influenza la durata della simulazione.

BILIOGRAFIA

[1] Tracy Camp, Jeff Boleng, Vanessa Davies, “A Survey of Mobility Models for Ad Hoc Network Researc”. Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications, Vol. 2, No. 5, pp. 483-502, 2002.

[2] Fan Bai, Ahmed Helmy, “A Survey Of Mobility Models in Wireless Ad Hoc Network”.

[3] G.D’Angelo, M. Bracuto, “Distributed simulation of large-scale and detailed models”. International Journal of Simulation and Process Modelling, Vol. 5, No. 2, 2009.

[4] Marco Fiore, “Mobility Models in Inter-Vehicle - Communications Literature”

[5] Wikipedia - www.wikipedia.com

[6] GAIA APIs - http://pads.cs.unibo.it/dokuwiki/doku.php?id=pads:gaia-apis

[7] Vincent Gauthier - Mobility model in ad hoc network, 2009 – http://www-public.it-sudparis.eu/~gauthier/MobilityModel/mobilitymodel.html

[8] Luciano Bononi, Michele Bracuto, Gabriele D’Angelo, Lorenzo Donatiello, “A New Adaptive Middleware for Parallel and Distributed Simulation of Dynamically Interacting Systems”, Proceedings of the Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT ’04).

[9] ARTÌS: documentazione online - http://pads.cs.unibo.it/dokuwiki/doku.php?id=pads:artis-doc

[10] Glossario Informatico - http://www.pc-facile.com/glossario

[11] Averill M. Law, W. David Kelton - “Simulation Modeling and Analysis”