Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea in Fisica

TITOLO TESI

Relatore:
Prof. Armando Bazzani

Correlatore: (eventuale)
Prof./Dott. Nome Cognome

Presentata da:
Gregorio Berselli

Anno Accademico 2021/2022

Abstract

Indice

Introduzione **4**

1 Costruzione del modello **5**

2 Implementazione **6**
2.1 Classi 6
2.1.1 VehicleType 6
2.1.2 Vehicle 6
2.1.3 Street 7
2.1.4 Graph 8
2.2 Esecuzione 9
2.3 Performance 9

3 Risultati **10**

Elenco delle figure

1 *Prova.* 4

2.1 Velocità nel modello 7
2.2 Temperatura nel modello 8

Introduzione

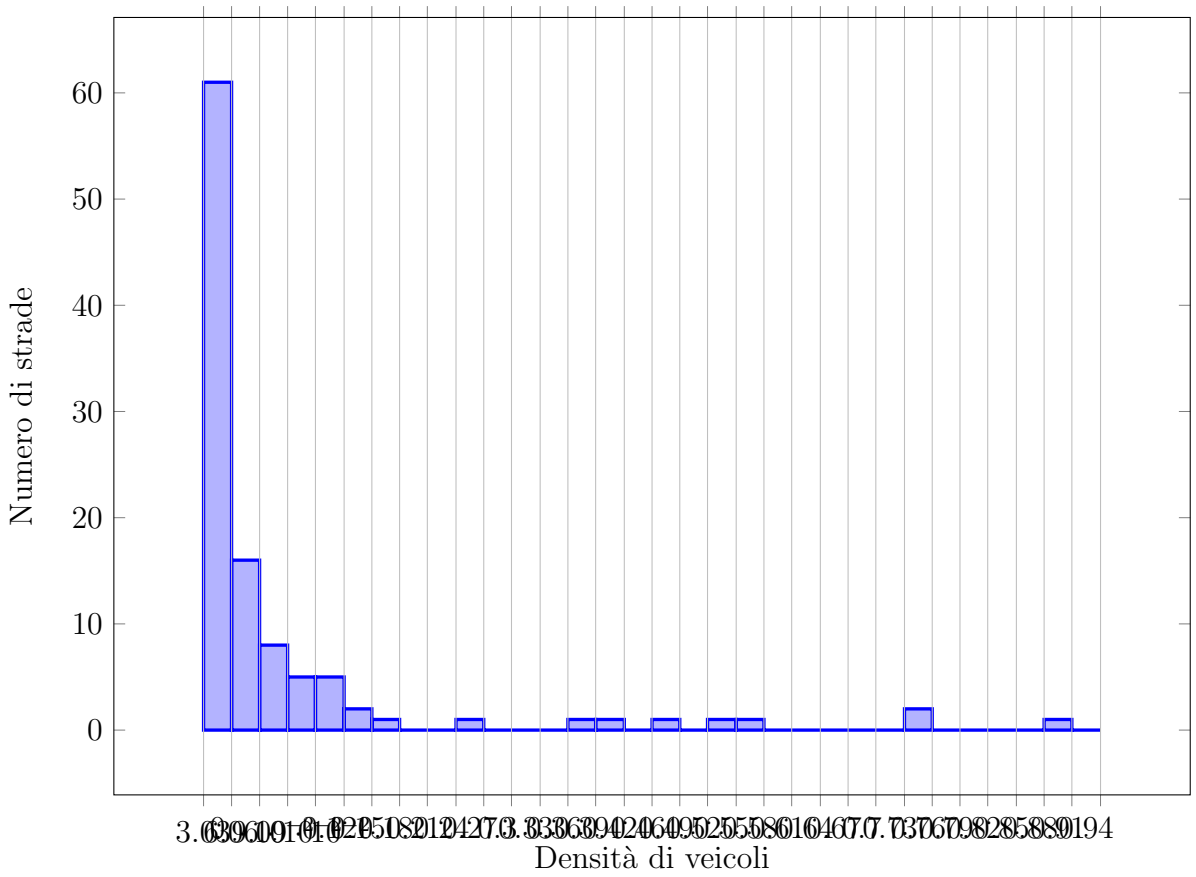

Figura 1: *Prova.*

Random Walk su network

Modelli di traffico

Capitolo 1

Costruzione del modello

Capitolo 2

Implementazione

Il modello descritto in precedentza è stato implementato tramite un software scritto in C++.

2.1 Classi

Sfruttando la programmazione a oggetti su cui è basato il linguaggio C++ si è diviso il modello in varie classi.

2.1.1 VehicleType

La classe a livello inferiore è *VehicleType* che, come suggerisce il nome, definisce una tipologia di veicolo. In questo modello ogni veicolo è caratterizzato dai parametri di input nodo sorgente e nodo destinazione. Tuttavia, per muoversi sul network, ogni tipologia di veicolo necessita anche di una matrice di transizione contenente le probabilità di effettuare o meno un passo in una specifica dimensione. Questa matrice viene solo dichiarata come parametro della classe e viene impostata dopo la definizione del network.

2.1.2 Vehicle

Dopo aver definito le tipologie di veicoli è necessario definire anche i veicoli stessi. La classe *Vehicle* rappresenta dunque gli agenti che andranno a muoversi sul network stradale. Un vettore statico di *VehicleType* permette ad ogni veicolo di avere una tipologia definita, tramite un parametro indiciale che determina la posizione nel vettore. Ogni agente ha inoltre due coordinate che ne definiscono la posizione: nodo attuale e strada attuale. Per permetterne il movimento nel tempo sono presenti altri due parametri, non necessari in input, rappresentanti la velocità del veicolo, dettata dalla strada sulla quale si trova, e la penalità di tempo che questo deve scontare, dipendente sia dalla velocità del veicolo stesso sia alla densità di veicoli presente sulla strada in cui si trova.

2.1.3 Street

Una volta definiti i veicoli è necessario definire le proprietà dei collegamenti tra i vari nodi (incroci) della rete. Ogni istanza della classe *Street* rappresenta dunque un collegamento tra due nodi. I parametri da fornire come input per distinguere una strada in maniera univoca sono dunque l'indice del nodo sorgente e l'indice del nodo destinazione. Si presti ora attenzione al fatto che ogni strada abbia una direzione: considerando due nodi generici i e j, la strada che connette $i \to j$ sarà differente dalla strada che connette $j \to i$. Questa distinzione permette sia una gestione delle densità di veicoli più efficiente e coerente con la realtà, non avendo interferenza tra le corsie, sia l'inserimento di strade a senso unico nella rete.
Nella classe *Street* viene poi definita un parametro di controllo del modello, ossia la lunghezza media dei veicoli. Altri parametri della strada sono la sua lunghezza, il numero n di veicoli su di essa, la velocità massima consentita ,il numero di corsie (direzionate come la strada stessa) e la capacità massima n_{max} di veicoli presenti contemporaneamente. Si noti come mentre un veicolo conosce esattamente la strada in cui si trova ciò non sia vero per la strada in quanto quest'ultima possiede informazione solamente sul numero totale di veicoli presenti. La velocità effettiva mantenibile su una strada, essendo un valore altamente dinamico, non viene considerato come parametro (quindi immagazzinato in memoria) ma viene calcolato tramite una funzione quando necessario. In particolare, l'andamento della velocità su una strada segue la funzione

$$v(n) = v_{max}\left(1 - 0.75\frac{n}{n_{max}}\right) \tag{2.1}$$

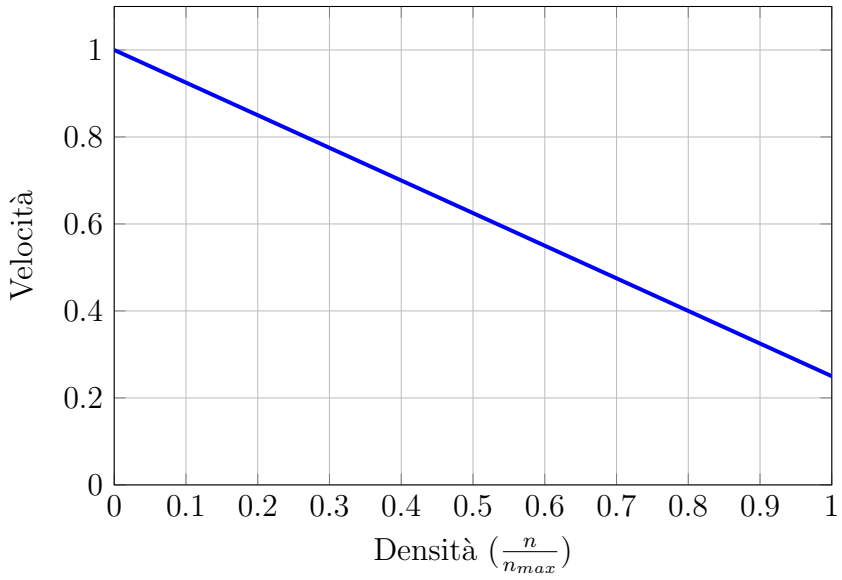

Figura 2.1: *Ipotesi dell'andamento della velocità in funzione della densità.*

Come visibile in Fig. 2.1 anche una volta raggiunta la densità massima i veicoli non si fermano ma si immettono sulla strada, quando si libera sufficiente spazio, con una velocità minima pari al 25% della velocità massima.

2.1.4 Graph

Ultima classe definita, che comprende tutte le precedenti, è la classe *Graph*, la quale costruisce effettivamente il network stradale. Parametro di input necessario per creare un'istanza è infatti la matrice di adiacenza, che definisce le connessioni tra i nodi. Tramite essa viene poi generato un vettore di puntatori a *Street* che genera le connessioni tra i nodi come strade.
Il movimento degli agenti sul network è determinato dalla matrice di transizione assegnata alle varie tipologie di veicoli. Questa viene generata tramite l'utilizzo del famoso algoritmo Dijkstra [1] e si basa quindi sulla ricerca del *best path* (a lunghezza inferiore) dalla posizione dell'agente alla destinazione definita dal suo *VehicleType*. Si è poi deciso di introdurre un parametro di temperatura al network per poter permettere ai veicoli di seguire un percorso differente rispetto al best path fornito dall'algoritmo Dijkstra. L'algoritmo di evoluzione, infatti, assegna peso 1 ai best path e un peso π variabile tra 0 e 1 ai percorsi più lunghi. Quest'ultimo peso varia in base alla temperatura del sistema secondo la funzione

$$\pi(T) = \tanh(kT) \tag{2.2}$$

dove k è un parametro di controllo del modello.

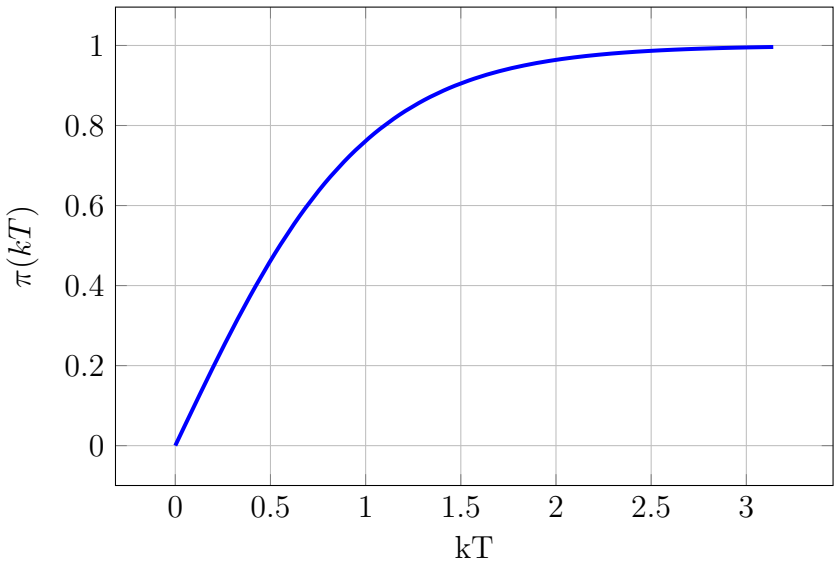

Figura 2.2: *Probabilità di errore in funzione della temperatura.*

Si procede poi alla normalizzazione a 1 di ogni vettore riga della matrice in modo tale da poter ottenere la probabilità di transizione. Una volta ottenute le probabilità di transizione il sistema può quindi evolvere tenendo presente che:

- se un agente prova a muovere su una strada piena questo movimento viene impedito e di fatto si perde uno step temporale;
- la velocità di ogni veicolo viene impostata all'ingresso in una strada e non più modificata fino all'ingresso nella strada successiva.

2.2 Esecuzione

2.3 Performance

Il programma è stato compilato utilizzando il compilatore gcc-9 su Ubuntu 20.04 nel Windows Subsystem for Linux. La compilazione è stata effettuata utilizzando le flag

```
-O3 -Wall -Wextra -fsanitize=address
```

in cui:

- *O3* indica il livello di ottimizzazione massima volto a ridurre il tempo di esecuzione del programma;
- *Wall* e *Wextra* consentono di correggere ogni tipo di warning che sorge in compilazione;
- *fsanitize=address* consente di ottenere un'ottima gestione della memoria.

Capitolo 3

Risultati

Bibliografia

[1] Thomas Cormen et al. *Introduction to Algorithms, Second Edition*. Gen. 2001, pp. 504–508. ISBN: 0-262-03293-7.