
Progetto di Sistemi Complessi: Modelli e Simulazione

SUPERMARKET SIMULATION

TONELLI LIDIA LUCREZIA - 813114

MARCO GRASSI - 829664

GIANLUCA GIUDICE - 830694

Contents

1	Introduzione	2
2	Stato dell'arte	4
3	Descrizione del modello	6
3.1	Framework adottato	6
3.2	Sistema	6
3.3	Agente di tipo Cliente	7
3.3.1	Workflow	8
3.4	Agente di tipo Cassa	10
3.4.1	Tipi di cassa	11
3.4.2	Workflow	13
3.5	Ambiente	19
3.5.1	Interazione tra agenti	19
3.5.2	Caratteristiche dell'ambiente	20
3.6	Considerazioni sul modello	20
3.7	Parametri del modello	21
3.7.1	Configurazione delle casse e struttura del supermercato	21
3.7.2	Parametri di jockeying	21
3.7.3	Basket size	21
3.7.4	Parametri di tempo	26
3.7.5	Distibuzione dei clienti in entrata nel supermercato	27
4	Implementazione dei comportamenti e delle strategie	34
4.1	Fase di scelta della coda	34
4.2	Fase di attesa in coda e jockeying	36
5	Simulazione e analisi dei risultati	39
5.1	Simulazione con scopo di validazione	39
5.2	Simulazione con strategie <i>jockey</i>	41
5.3	Simulazione con coda condivisa	45
5.4	Simulazione con casse self-scan	48
5.5	Simulazione non deterministica	50
6	Conclusioni	52
6.1	Sviluppi futuri	52
	Bibliografia	54

Chapter 1

Introduzione

In questo progetto è stato costruito un modello basato su agenti con lo scopo di simulare il comportamento dei clienti all'interno di un supermercato durante la fase di scelta della coda relativa a diversi tipi di casse.

Un grande supermercato è composto da molte casse diverse, ognuna di queste ha un comportamento diverso. Come vederemo in dettaglio nel capitolo 3, esistono diversi tipi di casse: la cassa standard, in cui si trova una cassiera che passa i prodotti provenienti dal nastro, la cassa self-service, in cui è il cliente stesso a scannerizzare i prodotti e la cassa self-scan, comparsa negli ultimi anni, in cui il cliente deve soltanto pagare, in quanto ha già effettuato la scannerizzazione dei prodotti man mano che li ha raccolti nel carrello. La configurazione delle casse nel supermercato può prevedere che ogni cassa abbia la sua coda dedicata, oppure che tante casse condividano la stessa coda.

Dal momento che in un grande supermercato sono presenti più clienti che casse, è fondamentale l'utilizzo di code per gestire la grande quantità di clienti. Un supermercato può essere considerato a tutti gli effetti un sistema complesso in quanto si verificano diversi aspetti che considerati contemporaneamente fanno emergere un comportamento complessivo difficilmente prevedibile, tra questi:

- Flusso di clienti in ingresso variabile (che influisce sulle persone in coda)
- Numero di prodotti che un cliente acquista durante la spesa (che influisce sul tempo passato in cassa, scatenando eventualmente il cambio della coda da parte di altri clienti)
- Numero di casse aperte contemporaneamente nel supermercato
- Strategia di scelta della coda dei clienti (un cliente può avere diversi criteri per la scelta della coda)
- Strategia di cambio della coda dei clienti

Lo scopo di questo lavoro è costruire un modello basato su agenti per simulare il comportamento del supermercato. Dopo una prima fase di validazione, sfruttando i pochi dati a disposizione, vengono considerati diversi casi "what-if" con lo scopo di sperimentare diverse configurazioni di casse e strategie di scelta della coda per una gestione ottimale del flusso dei clienti.

Questa relazione è organizzata come segue: nel capitolo 2 vengono descritte brevemente le fonti da cui prende ispirazione il progetto e giustificate le scelte riguardo i tipi di casse e le strategie usate dai clienti. Nel capitolo 3 sono descritti il sistema supermercato e gli agenti, con rimandi alla teoria e una descrizione in dettaglio dell'implementazione dei componenti;

in questo capitolo vengono anche descritti i parametri usati nel modello che possono essere cambiati a piacere. Nel capitolo 4 vengono descritte le strategie di scelta della coda e di cambio di coda (jockeying); le strategie vengono introdotte molto dettagliatamente perché rivestono un ruolo importante nel progetto, infatti l'obiettivo finale è capire quale di esse porta a un guadagno maggiore in termini di tempi di attesa in cassa. Nel capitolo 5 si vedono le simulazioni del modello, in particolare la prima simulazione servirà da confronto con l'articolo da cui prende principalmente spunto questo progetto; le altre simulazioni serviranno quindi a indagare sugli elementi che abbiamo voluto introdurre per espandere e rendere più realistico il nostro supermercato.

Chapter 2

Stato dell'arte

In questo capitolo si scorreranno brevemente gli articoli principali da cui abbiamo preso spunto per il nostro progetto.

Questo progetto prende spunto principalmente dal lavoro di Tomasz Antczak, Rafal Weron e Jacek Zabawa [1], in cui viene costruito un modello ad agenti realistico per simulare la scelta della coda dei clienti di un supermercato; lo scopo principale è il supporto alle decisioni nelle operazioni di vendita. Essi mostrano che quando i clienti scelgono la coda in modo da minimizzare il tempo d'attesa previsto, questo porta in generale a tempi d'attesa minori nelle code per tutti i clienti e porta anche benefici per la gestione del supermercato, in quanto permette di ridurre i turni di lavoro dei cassieri.

L'articolo sopracitato prende in considerazione 5 strategie diverse di scelta della coda:

1. La coda è scelta in modo random
2. È scelta la coda con il numero minore di clienti
3. È scelta la coda con il numero minore di prodotti in tutti i carrelli
4. È scelta la coda con il minor tempo d'attesa previsto calcolato come il prodotto del numero di clienti in coda e il tempo di servizio medio per tutte le casse
5. È scelta la coda con il minor tempo d'attesa previsto calcolato come il prodotto del numero di clienti in coda e la somma dei tempi di transazione e di pausa attesi

Nel nostro progetto in particolare, non abbiamo considerato la prima strategia di scelta random, ma soltanto le strategie 2-5.

Il modello di riferimento [1] implementato in NetLogo non corrisponde esattamente al modello reso disponibile nella repository ¹in quanto questo presenta aspetti non considerati nel paper. Questo fattore accompagnato dalla nostra inesperienza con NetLogo e la relativamente scarsa documentazione del modello ha portato alla decisione di reimplementare le parti del modello descritte nel paper [1] e le nostre espansioni in linguaggio Python nel framework Mesa.

Dal repository del loro progetto, abbiamo estratto i dati su cui si basa il loro modello, in particolare la distribuzione di arrivo dei clienti nel supermercato e la distribuzione della grandezza dei loro carrelli (più avanti chiamata *basket size*); si parlerà in dettaglio delle distribuzioni estratte nella sezione 3.7.

¹<https://github.com/tant2002/NetLogo-Supermarket-Queue-Model>

L'articolo [1] non modella il *jockeying*, ovvero il fatto che un cliente possa cambiare coda prima di essere servito, se calcola che in un'altra cassa ci sia un tempo d'attesa minore; abbiamo deciso di estendere il loro lavoro includendo la possibilità per i clienti di fare *jockeying*, per valutare se questo portasse a una riduzione ulteriore dei tempi d'attesa. Per implementare questa estensione, abbiamo preso spunto dall'articolo [2]. In questo articolo l'autore considera la strategia di *jockeying* nelle code, perché in molte situazioni reali si utilizza per ridurre il tempo totale speso in coda. L'autore prende in considerazione tre situazioni di coda diverse, in particolare nella terza (*Tellers' Windows with Jockeying*) definisce una strategia di jockey probabilistico con un *threshold k*, che rappresenta la verosimiglianza di un cliente di lasciare la propria coda per un'altra. Nel nostro modello è stata implementata questa strategia, includendo anche una randomicità, che esprime il fatto che non tutte le persone fanno jockeying.

Uno studio che abbiamo voluto includere nel nostro progetto è la differenza dei tempi di attesa al variare della disposizione delle code rispetto alle casse: nell'articolo [3] i ricercatori prendono in considerazione due tipi di disposizioni delle code: **parallela**, che si trova quando ogni cassa ha una coda dedicata, e **N-Fork**, che si trova quando c'è un'unica coda condivisa tra più casse. Essi hanno l'obiettivo di estendere la *queueing theory* considerando la distanza da percorrere per arrivare a una coda, allo scopo di rendere più efficiente il sistema di code. Nel nostro progetto, noi non abbiamo incluso nelle strategie di scelta della coda la distanza da essa, e di fatti questa potrebbe essere un'estensione, abbiamo però considerato le due disposizioni di casse studiate.

Un'ulteriore estensione che abbiamo apportato a questi modelli è stata l'introduzione delle casse *self-scan*: nell'articolo [1] vengono considerate le casse standard, con il cassiere, e le casse self-service, in cui il cliente passa in autonomia i prodotti allo scanner; nell'articolo [3], invece, viene considerata solamente la cassa standard. Negli ultimi anni molti supermercati hanno adottato le casse self-scan, che permettono al cliente di scannerizzare i prodotti durante la spesa e, arrivati alla cassa, semplicemente pagare, senza passare i prodotti su un nastro; questo tipo di cassa sembra essere molto efficiente, in quanto richiede soltanto il pagamento, che diventa ancora più veloce se fatto in forma telematica. Per evitare furti, i supermercati decidono di effettuare in maniera randomica delle rilettture della spesa.

Infine, per rendere la simulazione non deterministica e più realistica, abbiamo voluto inserire delle variabili probabilistiche: l'entrata dei clienti e la loro *basket size* sono basate su distribuzioni di probabilità estratte dai dati di [1]; l'estrazione dei clienti delle casse self-scan per la rilettura della spesa è randomica; il *jockeying* da parte di un cliente è random, in quanto non tutte le persone in fila guardano le altre code allo scopo di minimizzare il tempo di attesa; la stima del *basket size* degli altri clienti da parte di un cliente, nel momento in cui utilizza la sua strategia per calcolare qual è la coda da scegliere, segue una distribuzione normale, rispecchiando così l'errore di stima che compie un umano quando deve quantificare una misura. Tutte queste distribuzioni di probabilità sono governate da parametri che, a parte per i dati estratti dal primo articolo, non sono giustificati da alcuno studio, per questo lo studio di essi rappresenta un'ulteriore estensione del nostro modello.

Abbiamo chiarito le fonti da cui abbiamo preso ispirazione per costruire il supermercato ed estenderlo con più funzionalità; nel prossimo capitolo verranno descritti l'ambiente e gli agenti nel dettaglio, le loro caratteristiche e le interazioni tra essi; quindi verranno descritti tutti i parametri che caratterizzano una simulazione nel nostro modello.

Chapter 3

Descrizione del modello

In questo capitolo verrà descritto in dettaglio il modello del supermercato da noi implementato; in particolare vedremo il framework utilizzato per implementare il modello, le motivazioni che ci hanno portato a costruire un modello ad agenti, gli agenti Cliente e Cassa visti nel particolare, l'ambiente fisico in cui agiscono gli agenti, come essi interagiscono, infine i pattern e i parametri implementati per rendere il modello flessibile.

3.1 Framework adottato

Mesa [4] è un framework in Python usato per la modellazione basata su agenti (ABM). Permette di creare il modello con componenti *built-in* come griglie spaziali e scheduler di agenti, e di visualizzare i componenti del modello con un'interfaccia browser. Sfruttando Mesa è possibile definire sia l'ambiente che gli agenti estendendo le relative classi messe a disposizione dal framework. Mesa infine comprende strumenti per l'analisi del modello creato.

Nel nostro modello di supermercato, il modello vero e proprio di Mesa è la classe **Supermarket** e gli agenti sono i clienti, classe **Customer**, e le casse, classe **CashDesk**.

3.2 Sistema

Il supermercato è stato progettato come **sistema multi-agente**, comprendente agenti Cliente e Cassa, con l'obiettivo di indagare sulle scelte individuali dei clienti, infatti lo scopo del progetto è trovare la migliore strategia di scelta della coda al fine di minimizzare i tempi d'attesa. La presenza di strategie individuali rende utile la modellazione ad agenti, infatti vorremmo che dai comportamenti dei singoli clienti emergano alcune proprietà del sistema, come la non linearità, che non si possono predire a priori.

La scelta del modello ad agenti è in realtà molto naturale: il supermercato è un sistema complesso in cui diverse entità interagiscono e cercano di risolvere alcuni problemi. I clienti effettuano scelte individuali, hanno obiettivi (la spesa, il tipo di pagamento) e i lavoratori e le casse sono lì per soddisfare i loro bisogni; i clienti e i lavoratori devono interagire per far "evolvere" il sistema supermercato durante la giornata, per raggiungere i loro obiettivi.

Qualunque sistema umano, in particolare che comprenda le decisioni umane, ha una componente impredicibile, stocastica; per questo abbiamo voluto inserire, soprattutto nelle decisioni dei clienti, delle variabili che dipendono da delle distribuzioni di probabilità. Per questo

motivo il sistema descritto in questa relazione non è deterministico ed ogni simulazione è diversa dalle altre.

Il sistema supermercato è multi-agente perché in esso agiscono tanti agenti, in particolare il numero di clienti presenti varia nel tempo, mentre il numero di casse resta costante durante una simulazione. Come si vedrà nelle sezioni 3.3 e 3.4, gli agenti Cliente e gli agenti Cassa sono di tipo diverso (goal-based e model-based reflex, rispettivamente), per cui il supermercato è un sistema **eterogeneo**.

3.3 Agente di tipo Cliente

I clienti sono gli agenti principali che compongono il modello e interagiscono con l'ambiente per raggiungere l'obiettivo di fare la spesa; per portare a termine questo compito l'agente esegue alcuni macro-step in cui è necessaria anche una fase di pianificazione e valutazione della bontà (*utility function*) della scelta.

Dal punto di vista dell'agente di tipo cliente, la fase più complicata è quella in cui ha raggiunto il target basket size e deve quindi mettersi in una coda per attendere il suo turno in cassa. A questo punto il **goal dell'agente** è essere servito da una cassa nel minor tempo possibile, pur dovendo necessariamente passare per una coda. Questa fase richiede una pianificazione da parte dell'agente in quanto la coda viene scelta in modo ottimale cercando di minimizzare il tempo passato in coda, ciò avviene basandosi su diverse strategie che saranno descritte nel capitolo 4. Ragionamenti analoghi vengono fatti nel momento in cui l'agente già in coda decide di fare jockeying, ovvero cambiare la coda attuale perché questa azione porta al raggiungimento del goal.

Da queste considerazioni è possibile classificare l'agente di tipo Cliente come "utility-based", secondo la tassonomia proposta Russel-Norvig [5]; l'architettura di un agente "utility-based" è riportata nella figura 3.1.

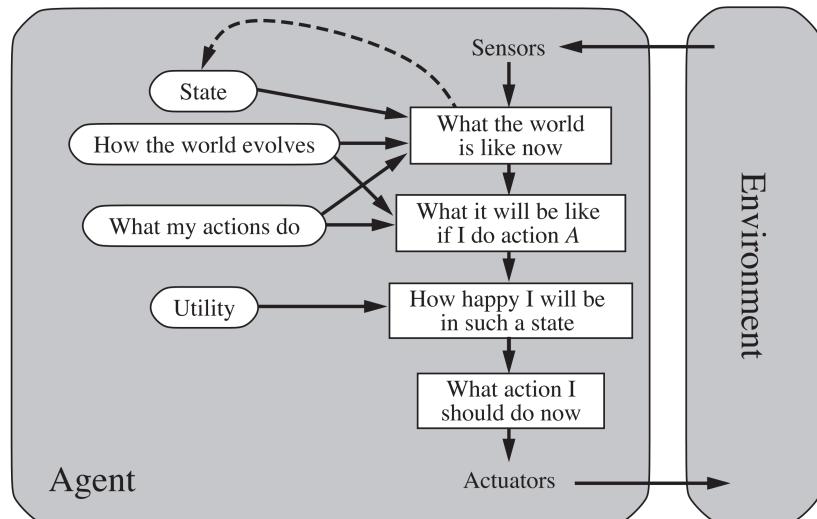


Figure 3.1: Agente di tipo utility-based.

Da questa immagine possiamo notare come ci sia una corrispondenza tra l'architettura proposta e quella dell'agente di tipo cliente. Infatti l'agente pianifica le sue azioni e valuta la bontà dell'azione di scegliere una coda per mezzo di una utility function.

3.3.1 Workflow

I macro-step che l'agente deve seguire per il raggiungimento del goal, che coincidono con gli stati associati all'agente durante la simulazione, sono:

1. **Attesa all'entrata del supermercato:** in questo stato il cliente si mette in attesa finché non è possibile entrare nel supermercato.
2. **Fase di shopping:** il cliente è entrato e può iniziare a "fare la spesa", con questo si intende raggiungere il numero di prodotti desiderato, infatti nella simulazione viene considerato il *basket size*, un'astrazione del carrello modellato attraverso un numero intero. Il basket size è il target da raggiungere nella fase di shopping, a ogni step il numero di prodotti aumenta di una certa quantità (la velocità di shopping è un parametro del modello descritto nella sezione 3.7).

La basket size viene generata stocasticamente secondo una distribuzione esponenziale di parametro $\lambda = 0.07361$. La scelta di questo parametro viene giustificata nella sezione 3.7.

3. **Scelta della coda:** in questo stato il cliente ha finito di fare la spesa e vuole mettersi in coda ad una cassa. Dal momento che sono disponibili più code e ognuna di queste avrà molto probabilmente altri clienti già in coda, il cliente deve scegliere in quale coda andare in base alla coda che lui considera migliore.

Formalmente viene definita una strategia di scelta della coda in base a una funzione: la *utility function*. Il cliente sceglie quindi la coda q^* che minimizza questa funzione:

$$q^* = \underset{q \in Q}{\operatorname{argmin}} f(q) \quad (3.1)$$

dove Q è l'insieme delle code e f varia a seconda del tipo di strategia che il cliente può utilizzare per la scelta della coda.

4. **Attesa in coda e jockeying:** una volta che il cliente ha scelto la coda deve aspettare il proprio turno per essere servito. Nel lavoro di Tomasz Antczak e altri [1] viene suggerito come sviluppo futuro lo studio del fenomeno di *jockeying*: con questa espressione si intende cambiare la coda che è stata scelta inizialmente in quanto un'altra risulta essere più conveniente, ad esempio perchè più scorrevole.

Ad ogni timestamp l'agente considera le due code adiacenti (parametro spiegato nella sezione 3.7 e scelto a priori per il modello) alla propria e per ognuna di queste ricalcola la coda migliore secondo la 3.1. A questo punto l'agente valuta se per lui è conveniente cambiare la coda o rimanere in quella dove è già presente. Si noti come per effettuare questo confronto è necessario utilizzare l'approccio della 3.1 sulla coda in cui l'agente è in quel momento, andando ad escludere i clienti che si sono inseriti in coda dopo di lui. Una volta fatta questa valutazione, il cliente cambia coda solo se il "guadagno" è maggiore di un certo threshold, anche questo parametro nel modello. A questo punto il cliente può cambiare coda con una certa probabilità, scelta a priori.

La scelta modellistica di effettuare jockeying stocasticamente e in funzione dell'effettivo guadagno di tempo è stata fatta per cercare di avere una rappresentazione il più fedele possibile al mondo reale. Infatti nella realtà un cliente al supermercato potrebbe scegliere di non cambiare la coda nel momento in cui ha un guadagno minimo in quanto questo richiede uno sforzo fisico che non tutti vogliono spendere nella realtà. Inoltre questa operazione richiede che gli agenti controllino continuamente le casse vicine, anche questo non è necessariamente vero in quanto dispendioso di energia. Infatti nel lavoro di Tomasz Antczak e altri [1] viene fatto notare come questo comportamento non sia molto frequente, tuttavia seppur essendo poco frequente noi lo consideriamo

un fattore importante da modellare, adottando le dovute precauzioni e sfruttando la stocasticità dell'azione.

5. **Attesa alla cassa:** in questa fase il cliente è arrivato alla cassa, essendo arrivato con lo stato precedente in prima posizione della coda. Da questo punto in poi verrà servito dall'agente di tipo cassa che avrà la responsabilità di processare il basket size del cliente. Dal momento che ci sono diversi tipi di casse, questa fase dipende dall'agente di tipo cassa e non dal cliente. Il cliente deve attendere la fine dell'elaborazione della spesa e quindi uscire dal negozio.

Il workflow dell'agente di tipo cliente viene astratto e riassunto da questa immagine:

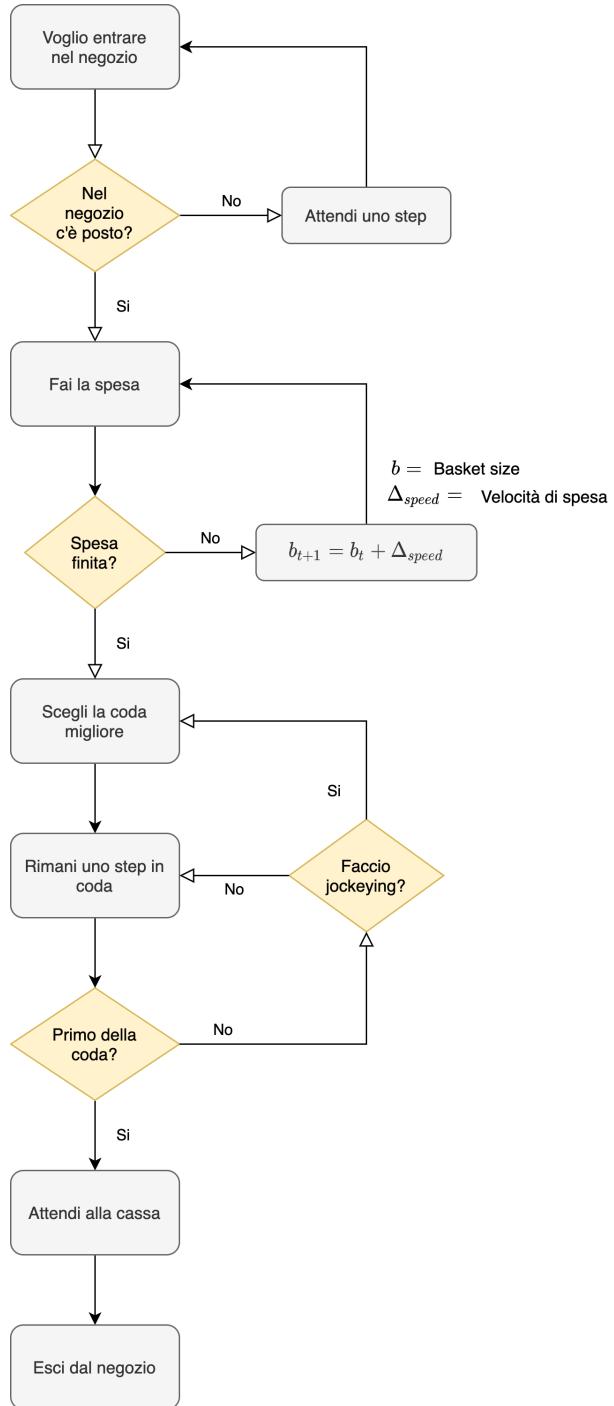


Figure 3.2: Workflow dell'agente di tipo customer.

3.4 Agente di tipo Cassa

Mentre i clienti sono gli agenti principali del sistema supermercato, che si muovono e svolgono le azioni interagendo con esso, le casse sono agenti statici nella griglia della simulazione; le casse non effettuano azioni per se stesse, ma attendono fino a che non ci sono clienti in coda per servirli, e permettergli di effettuare le transazioni di pagamento. Per servire i clienti,

la cassa necessita di una rappresentazione del proprio stato, che dipende dalla presenza o meno di clienti e dal loro basket size, di cui ha bisogno per effettuare il pagamento. L'unica decisione individuale che può prendere una cassa, e questo vale soltanto per le casse di tipo standard che si vedranno più avanti, è la propria attivazione o disattivazione in base al numero di clienti presenti nel negozio; l'essere attiva o disattiva fa parte dello stato interno della cassa.

Poichè la cassa non ha obiettivi ma solo il compito di attivarsi e servire gli eventuali clienti in coda, può essere classificata come "model-based reflex agent" secondo [5], descritto in figura 3.3.

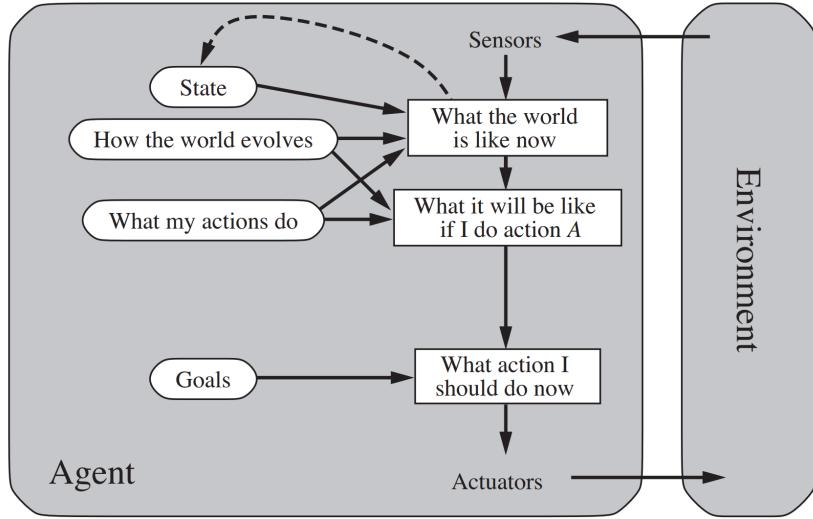


Figure 3.3: Agente di tipo model-based reflex.

3.4.1 Tipi di cassa

Gli agenti di tipo cassa hanno l'unica funzione di servire i clienti che si mettono in coda per pagare la spesa; sono di 4 tipi e differiscono per modalità di elaborazione della spesa del cliente e per velocità.

1. **Cassa standard:** questa cassa rappresenta la normale cassa che si trova solitamente in tutti i supermercati, all'interno della quale lavora un cassiere che si occupa di passare i prodotti del carrello del cliente sul nastro.

Il tempo di servizio di un cliente per questo tipo di cassa è formato da due tempi diversi: il **transaction-time**, che è il tempo vero e proprio durante il quale avviene la transazione, e il **break-time**, che è il tempo che passa tra il servizio di un cliente e di un altro ed include anche la fase in cui il cliente insacchetta i prodotti acquistati. Il transaction-time è stato stimato con una regressione di potenza e il break-time con una distribuzione Gamma. In particolare, per un cliente in coda $c_i \in q$, i due tempi sono calcolati in questo modo:

$$\text{transaction-time}_i = e^{a \log(\text{basket-size}(c_i)) + b} \quad (3.2)$$

$$\text{break-time}_i = \frac{\beta^\alpha \text{basket-size}(c_i)^{\alpha-1} e^{-\beta \text{estimate-basket-size}(c_i)}}{\Gamma(\alpha)} \quad (3.3)$$

Dove Γ è la funzione **gamma** e corrisponde a:

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx \quad \forall z \in \mathbb{C} \quad (3.4)$$

Queste stime sono state prese dall'articolo [1] e in particolare i parametri $a, b, \alpha, \beta \in \mathbb{R}$ sono:

$$a = 0.6984, \quad b = 2.1219, \quad \alpha = 3.074209, \quad \beta = \frac{1}{4.830613} \quad (3.5)$$

2. **Cassa self-service:** alla cassa self-service il cliente provvede da solo al passaggio dei prodotti del carrello allo scanner, per cui non è presente un cassiere. Si suppone che il cliente non sia veloce quanto un cassiere esperto a passare i prodotti sullo scanner, per questo la velocità di processing per questa cassa è stata divisa per un fattore 1.5, uno dei parametri descritti nella sezione 3.7. A causa della velocità ridotta, di solito nei supermercati è posto un limite di *basket size* per accedere alla cassa self-service; nel nostro modello questo limite è posto a 15, per cui se un cliente ha più di 15 elementi nel carrello e non va nel self-scan, deve per forza andare alla cassa standard.

Anche in questo caso il tempo di servizio è diviso in transaction-time e break-time, solo che, a differenza della cassa normale, il break-time risulta più lungo in quanto il cliente inizia ad insacchettare solamente dopo aver passato i prodotti allo scanner, invece nella cassa standard può iniziare già mentre il cassiere passa i prodotti. In questo caso, dunque, il transaction-time e il break-time sono stati stimati entrambi con una regressione di potenza:

$$\text{transaction-time}_i = e^{a \log(\text{basket-size}(c_i)) + b} \quad (3.6)$$

$$\text{break-time}_i = e^{c \log(\text{basket-size}(c_i)) + d} \quad (3.7)$$

I parametri in questo caso sono:

$$a = 0.6725, \quad b = 3.1223, \quad c = 0.2251, \quad d = 3.5167 \quad (3.8)$$

3. **Cassa self-scan:** la cassa self-scan si differenzia totalmente dalle prime due descritte, il cliente che voglia utilizzarla infatti, deve deciderlo già al momento dell'entrata nel supermercato, in quanto deve scannerizzare i prodotti man mano che fa la spesa e necessita dunque di una macchinetta scanner; una volta arrivato alla cassa self-scan, egli deve semplicemente effettuare il pagamento, saltando quindi le fasi di scanner e insacchettamento che caratterizzano le casse standard e self-service. La velocità di elaborazione della spesa in questo caso è praticamente nulla, in particolare un cliente che inizia il pagamento a una cassa self-scan, lo termina allo step successivo; il tempo di pausa per questa cassa è ugualmente nullo, dato che un cliente paga ed entra immediatamente nella fase di uscita dal negozio.

Chiaramente per evitare che vengano rubati prodotti, i supermercati decidono di effettuare dei controlli random sui clienti che scelgono di usare le casse self-scan. A questo scopo è stata implementata la cassa di tipo riservato.

4. **Cassa riservata:** questa cassa è identica nel funzionamento a una cassa standard, ciò che cambia è che è riservata alla rilettura della spesa dei clienti self-scan, che avviene in maniera random. Nel momento in cui un cliente si approssima ad una cassa self-scan, viene estratto un numero che determina la **rilettura parziale** o la **rilettura totale** della sua spesa; la rilettura parziale consiste nell'estrazione di 10 prodotti dal carrello

e nella verifica che essi appartengano alla lista contenuta nello scanner del cliente, invece la rilettura totale consiste appunto nella rilettura completa della spesa di esso. Le probabilità di rilettura sono parametri del modello ed attualmente corrispondono allo 2% per la rilettura parziale e allo 1% per la rilettura totale.

Il funzionamento e i tempi di servizio sono esattamente gli stessi della cassa standard, considerando un basket size completo per la rilettura totale e un basket size di soli 10 elementi per la rilettura parziale.

Tutti i tipi di cassa attraversano 3 fasi: fase di attesa di un nuovo cliente, fase di elaborazione della spesa del cliente, fase di completamento della transazione.

Code Nel modello, per quanto riguarda le sole casse standard, vengono considerati i casi di:

- **Code parallele:** per questa disposizione ad ogni cassa è associata una coda. Nel momento in cui la cassa è libera il cliente in cima alla coda viene spostato alla cassa per procedere con la fase di elaborazione della spesa.
- **Coda condivisa:** nota anche come *N-fork* in [3], più casse condividono la stessa coda, nel momento che una qualsiasi delle casse risulta libera il cliente in posizione utile procede alla cassa. Utilizzando questa disposizione non viene ammesso jockeying.

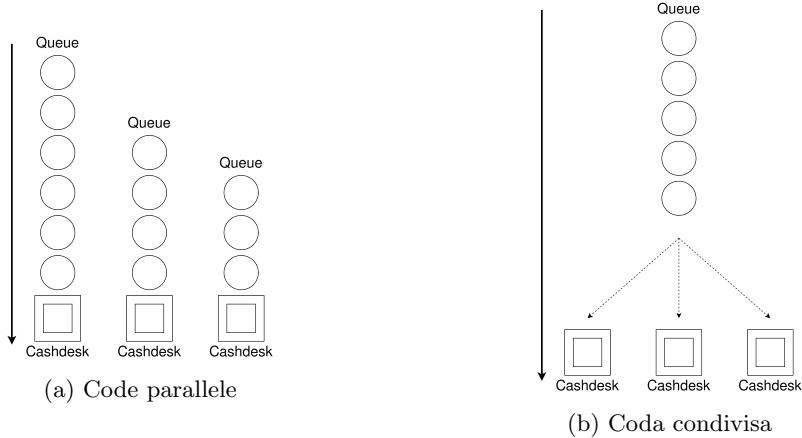


Figure 3.4: Tipi di code

3.4.2 Workflow

Cassa standard All'inizio della simulazione, e in generale quando la cassa ha terminato la transazione di un cliente o quando non ci sono clienti in coda, la cassa si trova nella **fase di attesa di un nuovo cliente**. L'agente cassa chiede alla coda (sia nel caso di code parallele sia nel caso di N-fork) a cui è associata il primo cliente in fila, se ce n'è, quindi compie 4 passi:

1. Cambia lo stato al cliente che sta servendo, impostandolo in fase di attesa alla cassa
2. Muove il cliente di fianco alla propria postazione, spostandolo quindi dalla coda
3. Cambia il proprio stato impostandolo in fase di elaborazione
4. Fa avanzare tutti i clienti che erano in coda dopo il cliente che viene servito

Nella **fase di elaborazione della spesa del cliente**, la cassa simula il passaggio dei prodotti sullo scanner, diminuendo il *basket size* del cliente ad ogni passo di una quantità

che dipende dalla velocità di processing, parametro del modello. Quando la transazione è completata, ovvero quando il *basket size* del cliente è nullo, fa uscire il cliente dal negozio e cambia il proprio stato, entrando nella **fase di completamento della transazione**.

In questa ultima fase la cassa ha terminato il servizio di un cliente, quindi si rimette nel primo stato, in fase di attesa di un nuovo cliente.

Il workflow della cassa standard viene riassunto nell'immagine 3.5.

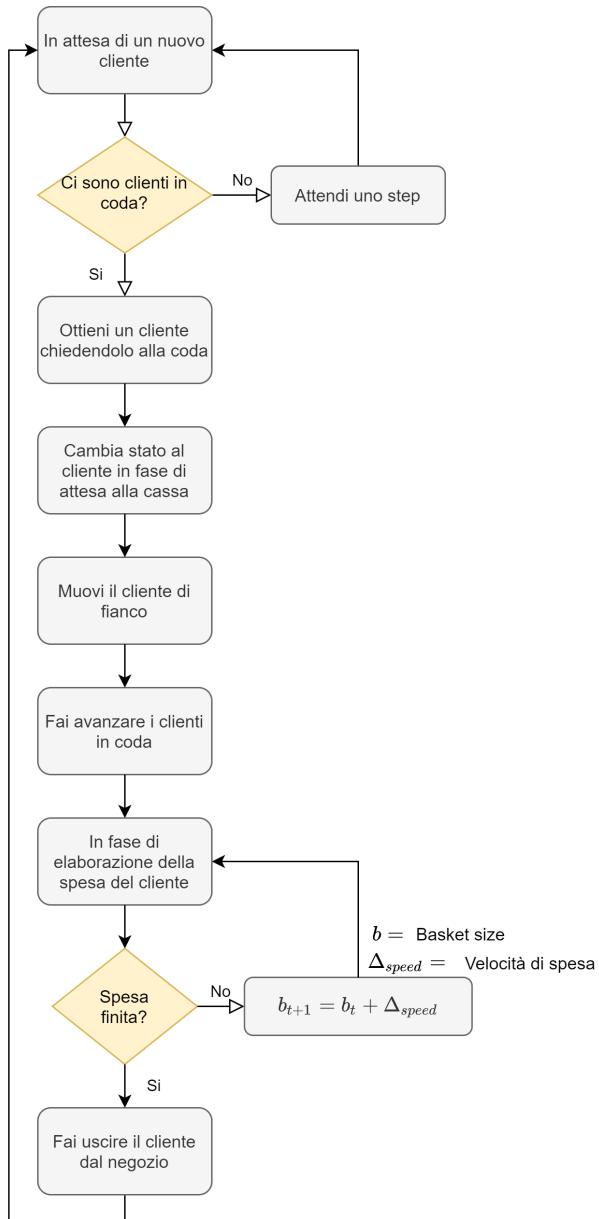


Figure 3.5: Workflow dell'agente di tipo cassa standard.

Cassa self-service Il workflow della cassa self-service è uguale in tutto a quello della cassa standard, le uniche differenze sono che esiste una coda condivisa ogni 8 casse self-service (che però funziona allo stesso modo della coda della cassa standard) e che la velocità di elaborazione della spesa è più piccola di 1.5, un parametro del modello.

Cassa self-scan Le casse self-scan hanno sempre una coda condivisa, come si può notare in tutti i negozi che hanno adottato questo tipo di cassa. Nella **fase di attesa di un nuovo cliente**, se ci sono clienti nella coda condivisa, la cassa self-scan prende il primo cliente della coda; estrae quindi un numero e, con il 2% di probabilità lo manda alla cassa riservata per una **rilettura parziale**, con il 1% di probabilità lo manda alla cassa riservata per una **rilettura totale** e invece con il 97% di probabilità gli fa fare il pagamento normale. Nel caso di rilettura parziale o totale, la cassa sposta il cliente alla cassa riservata, fa avanzare i clienti in coda e rimane nella fase di attesa di un nuovo cliente, pronta per prendere il nuovo cliente della coda.

Nel caso non debba avvenire una rilettura la cassa compie queste 4 azioni:

- Cambia lo stato al cliente che sta servendo, impostandolo in fase di attesa alla cassa
- Muove il cliente di fianco alla propria postazione, spostandolo quindi dalla coda
- Cambia il proprio stato impostandolo in fase di elaborazione
- Fa avanzare tutti i clienti che erano in coda dopo il cliente che viene servito

Nella **fase di elaborazione della spesa del cliente**, la cassa effettua la vera e propria transazione, la differenza con la cassa standard è la velocità di elaborazione della spesa, che in questo caso è uguale alla *basket size* del cliente; il cliente infatti deve soltanto pagare, in quanto ha già scannerizzato i prodotti durante la sua spesa.

Nell'ultima fase la cassa ha terminato il servizio di un cliente, quindi si rimette nel primo stato, in fase di attesa di un nuovo cliente.

Il workflow della cassa self-scan viene riassunto nell'immagine 3.6.

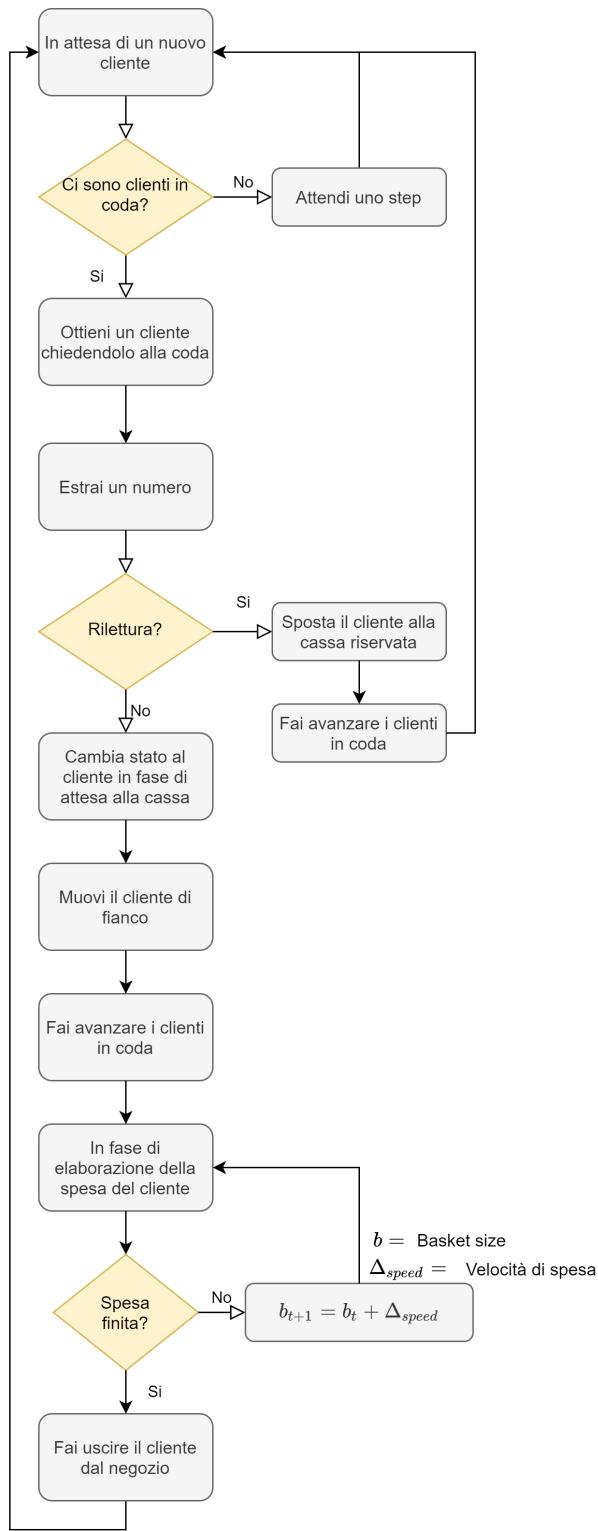


Figure 3.6: Workflow dell'agente di tipo cassa self-scan.

Cassa riservata Nella **fase di attesa di un nuovo cliente** la cassa riservata è pronta a servire i clienti in coda, però prima controlla che ci siano in coda clienti a cui è stata

assegnata una rilettura parziale della spesa, dandogli la precedenza in quanto la rilettura parziale è più veloce di quella totale.

Scelto quindi il cliente, questa cassa passa per le stesse fasi della cassa standard.

Il workflow della cassa riservata viene riassunto nell'immagine 3.7.

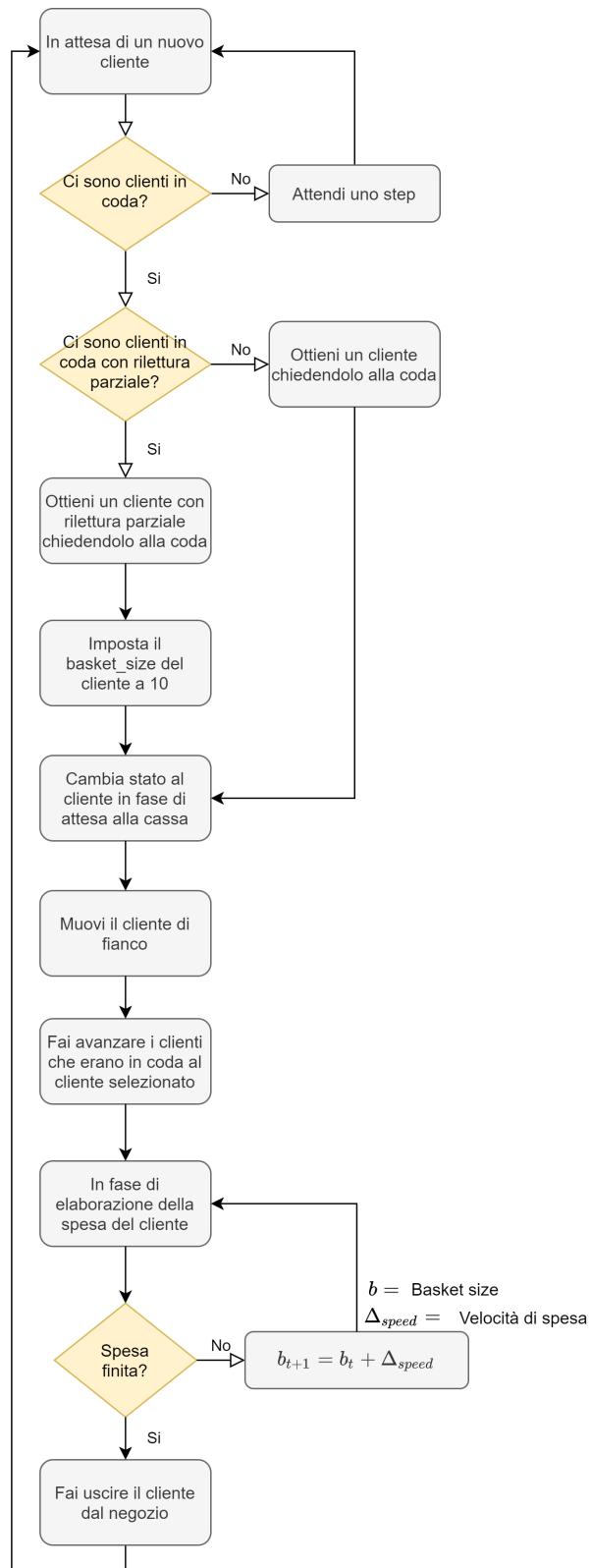


Figure 3.7: Workflow dell'agente di tipo cassa riservata.

3.5 Ambiente

L'ambiente in cui gli agenti vivono è il supermercato, implementato dalla classe **Supermarket** che si occupa di inizializzare la griglia che verrà poi mostrata in fase di simulazione sull'interfaccia, inizializzare le casse, i clienti e mediare le interazioni con gli agenti.

Per inizializzare la griglia, l'ambiente viene diviso in zone: zona d'entrata, zona di shopping, zona casse normali, zona casse self-service, zona casse self-scan. Questa divisione permette una gestione più semplice dello spazio e dei movimenti degli agenti. Ogni zona ha come parametri la dimensione o il numero di casse che deve contenere, questi parametri vengono inizializzati a priori e gestiti dall'entry point del programma, il file **main**, come si vedrà nella prossima sezione.

Ogni zona è responsabile della propria costruzione, ovvero del proprio collocamento nella griglia dell'interfaccia in base alle proprie dimensioni ed eventualmente del posizionamento delle casse che contiene. Inoltre ogni zona è responsabile dei movimenti dei clienti: se un cliente vuole muoversi da una zona all'altra oppure mettersi in coda nelle casse di una zona, è la zona di destinazione che fornisce il metodo per posizionarsi correttamente in essa.

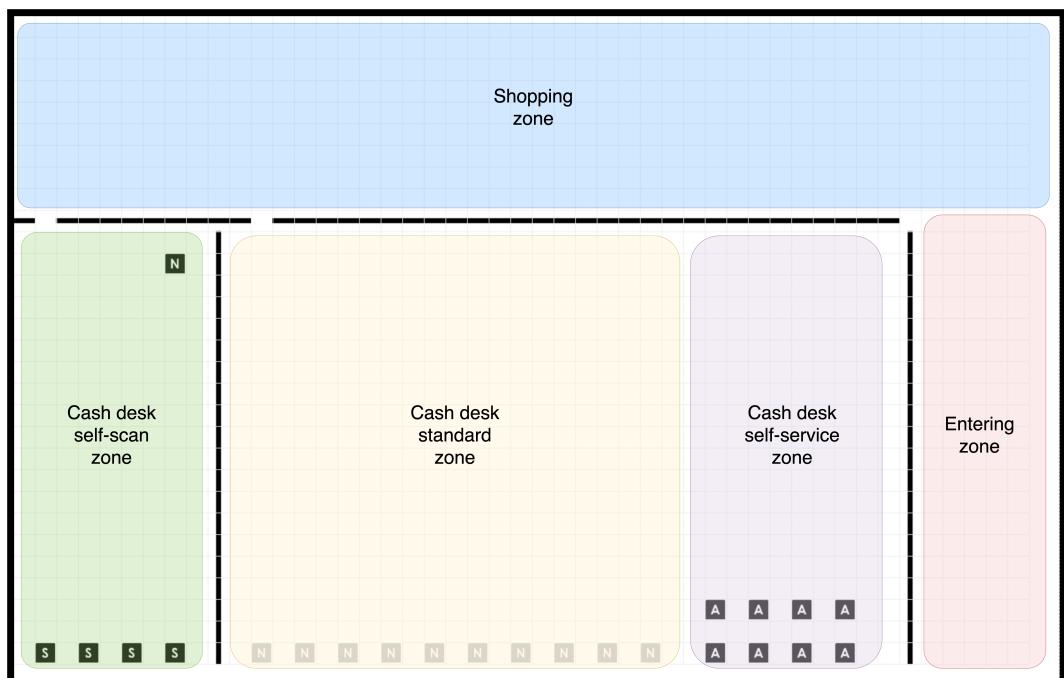


Figure 3.8: Struttura del supermercato diviso in zone.

Ad ogni step della simulazione, il modello crea dei clienti secondo la distribuzione data (si veda la sezione 3.7) e li posiziona nella **Entering Zone** del supermercato in coordinate random, dunque si occupa della attivazione e disattivazione delle casse standard in base al numero di clienti presenti nel negozio in quello step, secondo dei parametri che si vedranno nella prossima sezione. Quindi il modello chiama gli scheduler degli agenti, clienti e casse, e fa eseguire i loro step.

Parlare
di quanto
dura uno
step e
quanto
dura una
simulazione

3.5.1 Interazione tra agenti

I due tipi di agenti che compongono il modello interagiscono tra loro nella fase di attesa in coda da parte del cliente e quindi pagamento; l'interazione è necessaria perché i clienti

non sono in grado di autogestire le code e pagare, e questo fatto giustifica la presenza degli agenti Cassa, pur semplici che siano (insieme al fatto che le casse si attivano o disattivano in base al numero di clienti nel negozio, come si vedrà più avanti).

L'interazione tra clienti e casse è di tipo **collaborativo semplice**, in quanto gli obiettivi di entrambi sono compatibili (effettuare il pagamento e far uscire i clienti dal negozio), le risorse sufficienti e le abilità insufficienti (i clienti non sanno fare da soli il pagamento e le casse non possono fare la spesa ai clienti). Le casse e i clienti collaborano e comunicano per raggiungere l'obiettivo.

Gli agenti sono a conoscenza di dove si trovano nell'ambiente, ma non vedono in modo diretto gli altri agenti e la distanza che li separa; se un agente di tipo Cliente prende una decisione su quale coda seguire, ha bisogno di chiedere al supermercato dove si trova questa coda. Come descritto sopra, il supermercato è diviso in zone, ogni zona è responsabile di comunicare agli agenti la propria posizione e le casse che contiene, quindi fornisce i metodi per muoversi in essa. Il mezzo attraverso cui gli agenti comunicano è dunque l'ambiente stesso.

3.5.2 Caratteristiche dell'ambiente

L'ambiente supermercato è per gli agenti totalmente **accessibile**, infatti i clienti sanno dove si trovano tutte le casse e le casse conoscono ad ogni istante il numero di clienti presenti nel negozio.

L'ambiente è **deterministico**, se gli agenti compiono un'azione (ad esempio se un cliente si muove verso una coda), si conosce a priori l'effetto dell'azione; l'evoluzione dell'ambiente è non deterministica, in quanto sono stati introdotti elementi aleatori nelle decisioni degli agenti.

Un'altra caratteristica dell'ambiente è l'**episodicità**, infatti le esperienze degli agenti, sia casse che clienti, sono divise in fasi il cui esito dipende soltanto dal presente e non dalla serie di azioni svolte in precedenza.

Il supermercato è un ambiente **statico** perché non ha altri processi che operano oltre le casse e i clienti, che ci vivono dentro ma non lo modificano; non esistono interferenze nelle azioni degli agenti.

Infine l'ambiente è **discreto**, esistono un numero finito di azioni da effettuare, ma anche dal punto di vista fisico esso è rappresentato da una griglia di celle finite.

3.6 Considerazioni sul modello

Il modello è stato sviluppato in modo da essere flessibile per permettere con semplicità future espansioni e cambiamenti. Gli aspetti più dinamici del modello, ovvero il comportamento degli agenti, le varie modalità di scelta di una coda e la capacità di un agente Cliente di effettuare *jockeying* sono stati realizzati lasciando all'utilizzatore piena possibilità di modifica, ampliamento o di non utilizzo. A livello di architettura del software questo si traduce nell'implementazione dei pattern:

- **Strategy**: usato per definire diverse modalità di scelta della coda e diversi comportamenti di *jockeying*. Nuovi comportamenti possono essere aggiunti rispettando un'interfaccia standard. Permette se necessario cambiamenti di comportamento a run time.
- **State**: usato per definire tramite una macchina a stati finiti i comportamenti degli agenti Cliente e Cassa. Nuovi stati possono essere aggiunti con facilità e collegati ad altri stati tramite transizioni. La divisione in stati rende semplice ragionare sul comportamento di un agente.

3.7 Parametri del modello

Per rendere il modello più generale possibile e in modo da adattarlo a diverse configurazioni con lo scopo di simulare scenari what-if, vengono introdotti diversi parametri per l'inizializzazione e il comportamento della simulazione.

Controllare
questo
paragrafo
pls

3.7.1 Configurazione delle casse e struttura del supermercato

Tra i parametri più importanti e con maggiore impatto per il modello troviamo sicuramente la **configurazione del supermercato**. Un primo fattore che caratterizza la struttura del supermercato è la dimensione delle diverse zone: *shopping zone*, *cash-desk zone* e *entering zone*.

Ancora più influente e determinante per il comportamento del modello è il numero e tipo di casse che compongono il supermercato. Infatti una volta scelta la struttura e le dimensioni delle diverse zone, è necessario indicare nella *cash-desk zone* il numero di casse per ogni tipo, quindi specificare la quantità di:

- Casse self-scan (+ 1 riservata obbligatoria)
- Casse standard
- Casse self-service

Inoltre è necessario specificare un'ulteriore parametro che definisce se le casse di tipo standard hanno delle code parallele o una coda condivisa tra tutte; questo non avviene per le casse self-scan e self-service, in quanto nel primo caso c'è sempre una coda condivisa unica, e nel secondo c'è sempre una coda ogni 8 casse.

3.7.2 Parametri di jockeying

Il comportamento di jockeying è influenzato da alcuni parametri, questi sono:

- **Numero di code adiacenti considerate:** la coda in cui il customer è presente in quel momento svolge il ruolo di coda pivot, questo parametro indica il numero di code adiacenti alla coda pivot che vengono prese in considerazione per valutare l'eventualità di fare jockeying.
- **Threshold di jockeying:** questo parametro ha lo scopo di modellare il fatto che un cliente effettua jockeying solo nel caso in cui per lui risulta essere conveniente non meno di un certo livello; questo threshold indica quale deve essere il minimo guadagno per effettuare jockeying. Questo approccio basato su una soglia di minimo guadagno è stato presentato nel lavoro di Koenigsberg [2].
- **Probabilità di jockeying:** nel lavoro di Tomasz Antczak e altri [1], viene spiegato come il fenomeno di jockeying sia piuttosto raro. Il modello proposto tiene conto di questo aspetto introducendo una variabile aleatoria bernoulliana che ha lo scopo di consentire o negare il jockeying, nel caso in cui il jockeying risulti essere conveniente basandosi sul threshold. Con questo approccio si vuole rendere meno frequente il jockeying.

3.7.3 Basket size

Ogni cliente all'interno del supermercato vuole comprare un diverso numero di prodotti. Questa quantità viene assegnata a priori per tutti i clienti e supponiamo non cambi durante la fase di spesa; a questo scopo è necessario assegnare il basket size a ogni cliente secondo una certa distribuzione. Nel lavoro di Antczak e altri [1] vengono messi a disposizione dei dati relativi al numero di prodotti acquistati da parte dei clienti in tre supermercati diversi

nell'arco temporale di 13 giorni. Questi dati sono stati analizzati con lo scopo di introdurre la distribuzione che genera il basket size di ogni cliente.

Come prima cosa è stata presa in considerazione la frequenza delle dimensioni del basket size per ogni negozio al variare del tempo. Disegnando in un grafico queste informazioni si ottiene l'immagine 3.9.

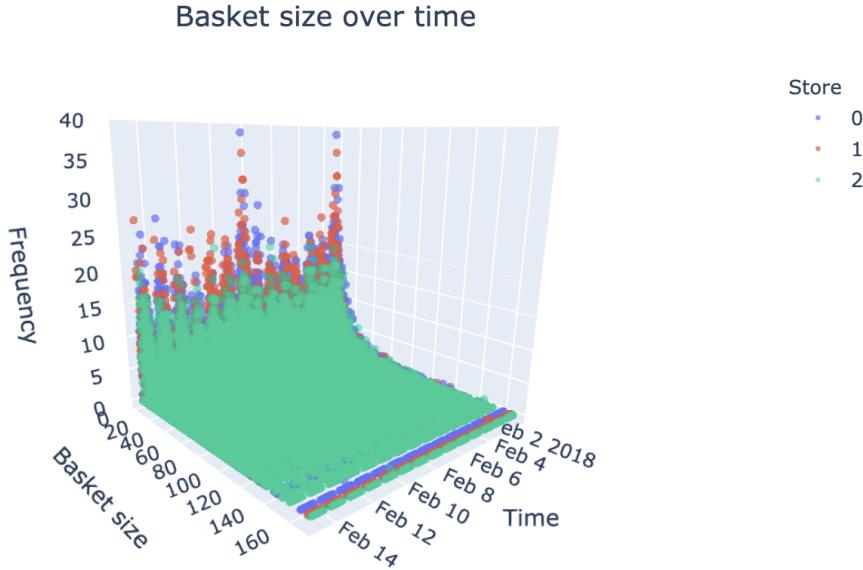


Figure 3.9: Frequenza della dimensione del basket size nei supermercati al variare del tempo.

Da questa immagine emerge chiaramente come la maggior parte delle persone compra pochi articoli e solo pochi clienti fanno una grossa spesa. Inoltre il comportamento tra i 3 supermercati è analogo a meno del numero di clienti totali del supermercato. Per quanto riguarda i diversi giorni considerati nell'arco di tempo messo a disposizione, non sembra che questi influiscano sulla dimensione del basket size, pertanto assumiamo per semplicità che non esista una correlazione tra i giorni della settimana e la dimensione del basket size.

Con questa assunzione si procede aggregando i dati sulla dimensione del tempo e calcolando la media, inoltre questi dati vengono normalizzati per renderli *scale-free* e quindi indipendenti dal numero di clienti per ogni supermercato. Dopo queste operazioni si ottiene il grafico 3.10.

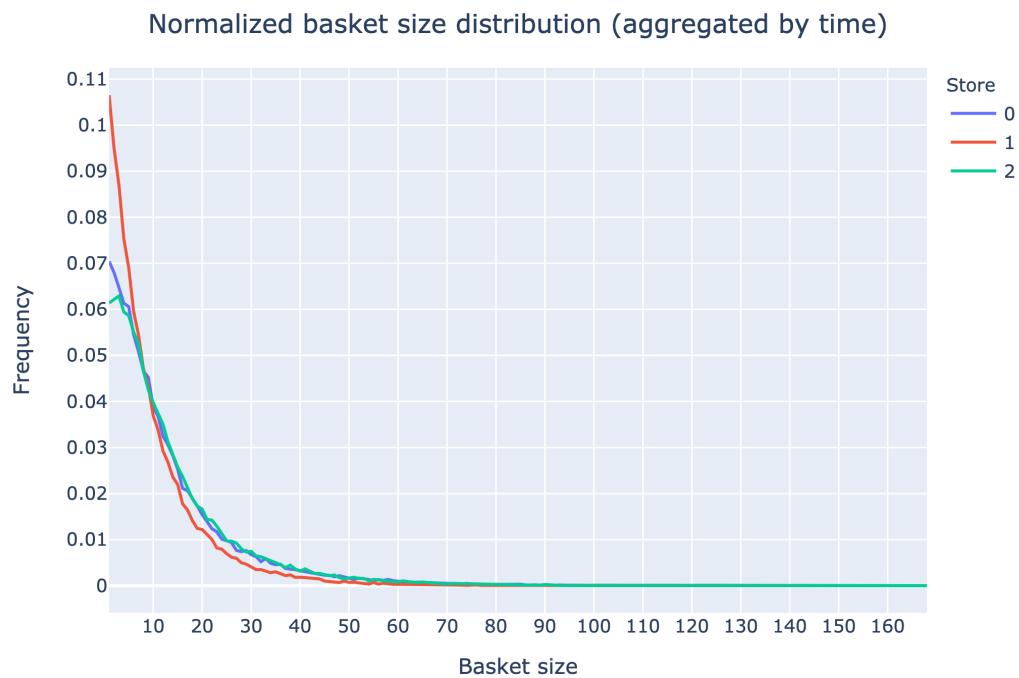


Figure 3.10: Frequenza della dimensione del basket size nei supermercati aggregato sul tempo.

Notiamo come le distribuzioni nei 3 supermercati siano in generale simili, per questa ragione vengono mediate le 3 distribuzioni ottenendo questa la distribuzione finale 3.11.

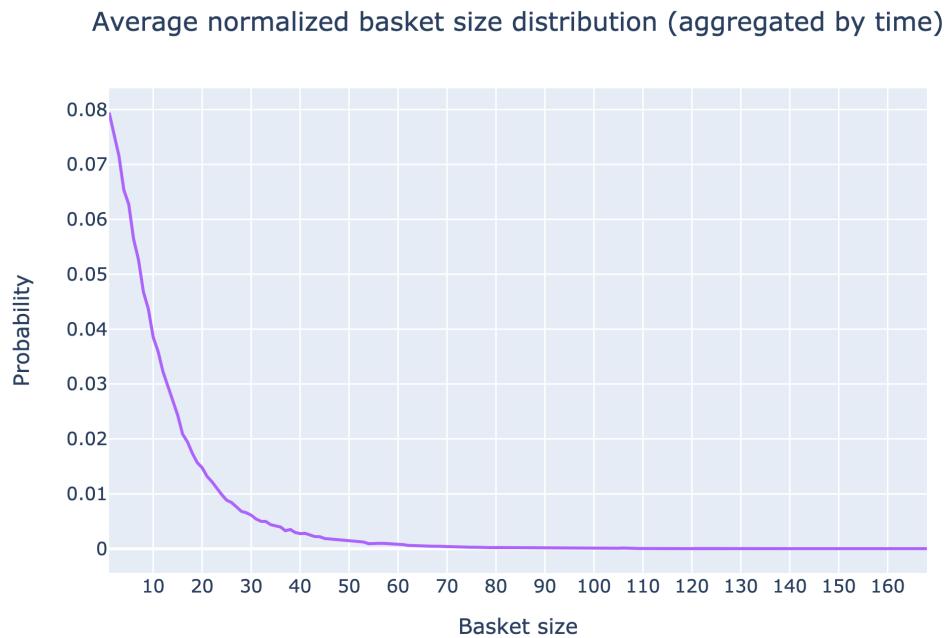


Figure 3.11: Media delle distribuzioni di frequenza del basket size aggregato sul tempo.

Da questa ultima immagine emerge chiaramente come la probabilità del basket size segua un andamento esponenziale, pertanto a partire dai dati originali si vuole ricavare il parametro λ che governa la distribuzione esponenziale sottostante. A questo scopo si trasformano i dati calcolando il logaritmo naturale delle ascisse e tracciando la retta di regressione lineare. Il risultato ottenuto viene mostrato nell'immagine 3.12.

Log-transformation normalized basket size distribution (aggregated by time)

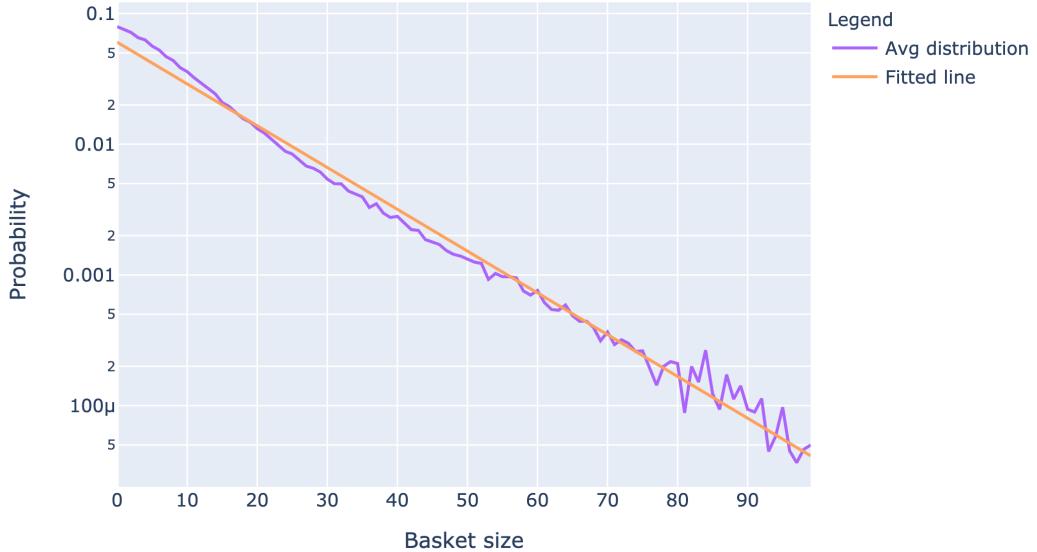


Figure 3.12: Retta di regressione dopo la trasformazione logaritmica dei dati.

La retta di regressione in output ha i seguenti parametri:

- $m = -0.0736184654254411$
- $q = -2.806857867915933$

A partire da questi parametri è facile calcolare il parametro λ della distribuzione esponenziale di partenza, infatti:

Sia X la variabile aleatoria che rappresenta la dimensione del basket size, e assumendo che $X \sim Exp(\lambda)$, dalla funzione di densità di probabilità si ha che:

$$f(x; \lambda) = \lambda e^{\lambda x}, \quad \text{se } x > 0$$

Per mezzo di una log-trasformazione si ottiene:

$$\begin{aligned}
 & \ln(\lambda e^{\lambda x}) && \text{(dalla funzione di densità di probabilità)} \\
 & = \ln(\lambda) + \ln(e^{-\lambda x}) && \\
 & = \ln(\lambda) - \lambda x \ln(e) && \\
 & = \ln(\lambda) - \lambda x &&
 \end{aligned} \tag{3.9}$$

La retta di regressione ottenuta a parire dai dati dopo la log-trasformazione è della forma:

$$\begin{aligned}
 y &= mx + q \\
 m &= -0.0736184654254411; q = -2.806857867915933
 \end{aligned}$$

Da cui (sfruttando l'equazione 3.9):

$$m = -\lambda \implies \lambda = -m = 0.0736184654254411$$

$$q = \ln(\lambda)$$

Pertanto si ricava dai dati che la distribuzione esponenziale teorica sottostante $Exp(\lambda)$ ha $\lambda = 0.0736184654254411$.

Nella figura 3.13 viene mostrata la distribuzione reale dei dati e la distribuzione esponenziale, la quale verrà poi effettivamente utilizzata per generare il basket size di ogni cliente:

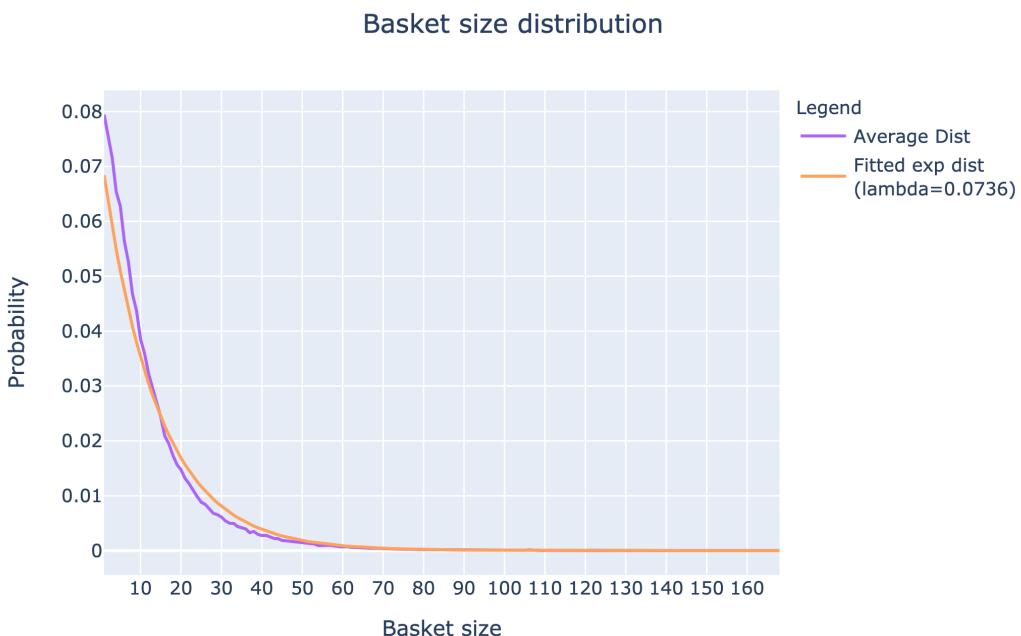


Figure 3.13: Distribuzione basket size reale a confronto con distribuzione esponenziale teorica.

3.7.4 Parametri di tempo

Lo scorrere del tempo all'interno del modello viene simulato per mezzo di step. Per questo motivo è necessario scegliere uno step del modello a quanti secondi corrisponde. Per le simulazioni effettuate nel nostro lavoro è stato scelto che **uno step corrisponde a 30 secondi**.

Come visto nel workflow dell'agente di tipo cliente, riassunto nell'immagine 3.2, una volta che egli entra nel supermercato è necessario simulare il processo di spesa del cliente, con questo si intende l'arco di tempo che l'agente impiega all'interno del supermercato per prendere i prodotti e inserirli nel carrello (che in questo lavoro non viene direttamente modellato ma simulato). Il tempo impiegato da parte del cliente per prendere tutti i prodotti desiderati, quantità che corrisponde al suo basket size, dipende dalla velocità di acquisto del cliente. Questa velocità è un ulteriore parametro del modello, che al momento è pari a

Inserire
a quanti
prodotti
corrisponde
la velocità e
quindi cosa
si vuole
simulare (es
2 prodotti)

Dopo la fase di spesa iniziale il cliente sceglie una coda per venire successivamente servito in cassa. Una volta in cassa, se questa è di tipo cassa standard o cassa self-service, introdotte nella sezione 3.4.1, allora è necessario processare tutto il basket size; quindi il cliente esce dal supermercato. Il tempo di elaborazione della spesa è governato dai parametri $a, b, \alpha, \beta \in \mathbb{R}$ introdotti nell'articolo [1] e diversi per i due tipi di casse. Come questi parametri influiscono il tempo di processamento del cliente una volta in cassa è spiegato nella sezione 3.4.1.

Spiegare la velocità di processamento per le casse normali

3.7.5 Distribuzione dei clienti in entrata nel supermercato

Nel lavoro di Antczak e altri [1], oltre ai dati relativi al basket size di ogni cliente, vengono messi a disposizione i dati degli arrivi dei clienti per ogni supermercato analizzato. I dati raccolti fanno riferimento a 13 giorni: dal "1 febbraio 2018" al "14 febbraio 2018" e si considera il supermercato aperto per 14 ore (dalle 8:00 alle 22:00).

La distribuzione dei clienti in ingresso rappresenta l'attività all'interno del supermercato viene mostrata in figura 3.14

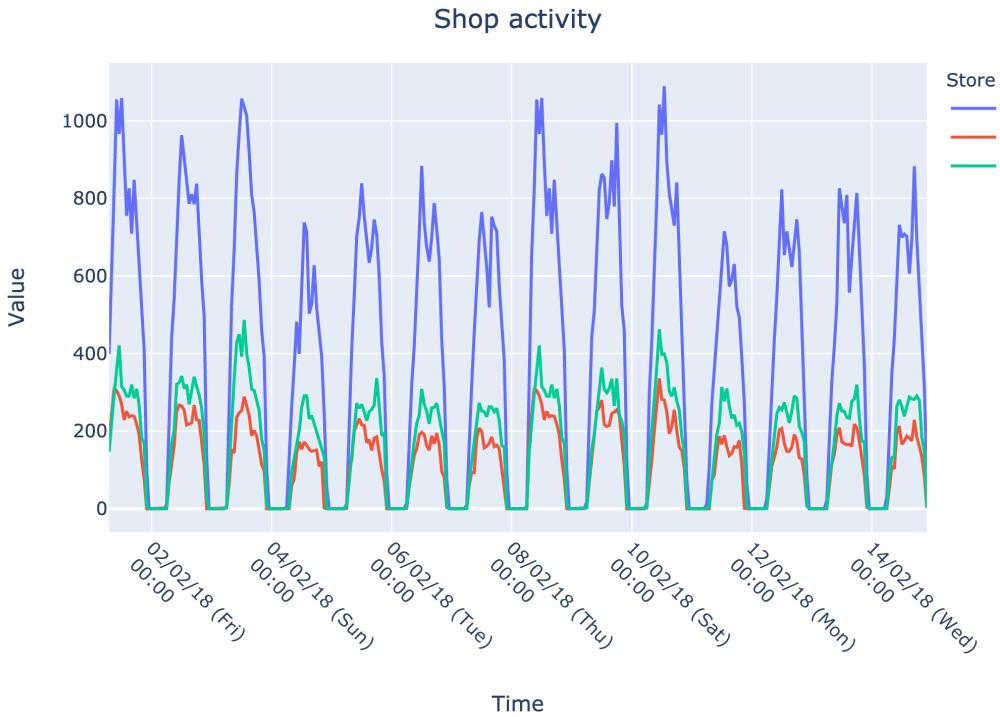


Figure 3.14: Distribuzione dei clienti in ingresso per ogni supermercato.

La distribuzione dei clienti in ingresso dipende dal numero di clienti totali nell'arco dell'intera giornata. Per rendere la distribuzione indipendente da questo fattore, vengono nominalizzate le distribuzioni per il numero totale di clienti. Il risultato viene mostrato in figura 3.15

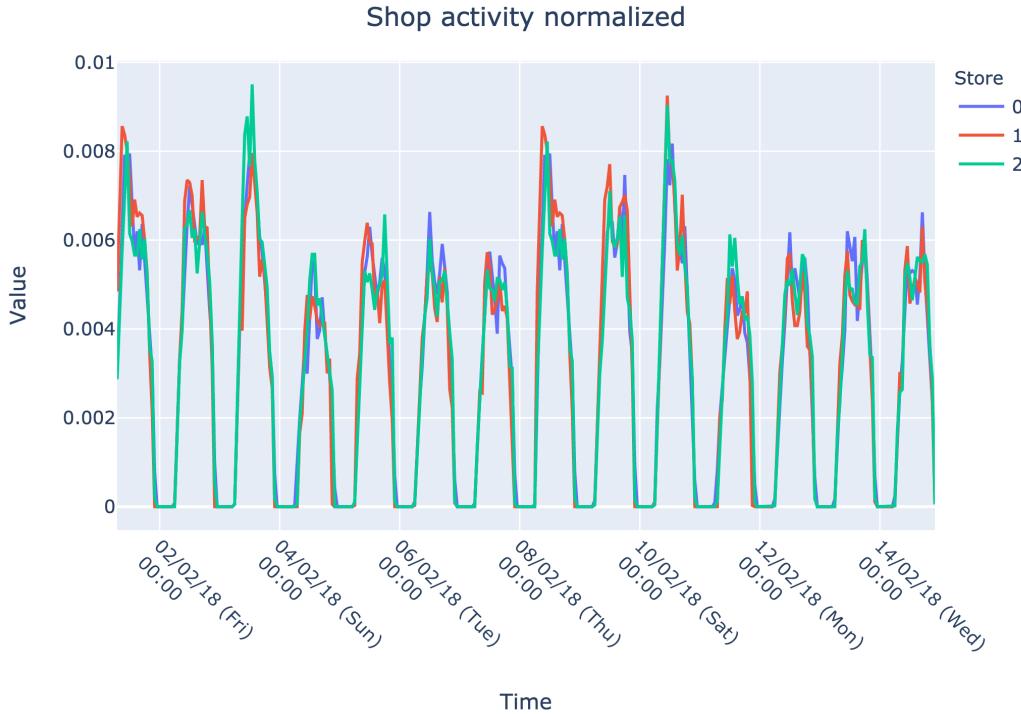


Figure 3.15: Distribuzione normalizzata dei clienti in ingresso per ogni supermercato.

Dalla figura 3.15 si nota come le distribuzioni nei supermercati, normalizzando per i clienti totali in ingresso nell’arco della giornata, è molto simile e quindi assumiamo che possa valere per un supermercato in generale.

Si procede aggregando gli ingressi al variare delle diverse giorante e calcolando la media. Dopo questa operazione si ottengono le tre distribuzioni dei diversi supermercati, più una ottenuta dalla media di queste tre. Il risultato viene mostrato in figura 3.16, dove notiamo come la linea viola corrisponde alla media delle tre distribuzioni, questa distribuzione verrà usata come parametro per specificare gli ingressi durante le simulazioni al variare del tempo.



Figure 3.16: Distribuzione dei clienti in ingressi aggregata per diversi giorni.

Un ulteriore fattore da considerare è che nei dati a nostra disposizione gli ingressi dei clienti sono aggregati per ora, tuttavia durante la simulazione è necessario fare entrare i clienti gradualmente ad ogni ora, ovvero gestire gli ingressi ad ogni step del modello. A questo scopo abbiamo considerato come distribuzione di riferimento quella ottenuta mediante le tre distribuzioni aggregate per i diversi giorni (linea viola nell'immagine 3.16). Dal momento che ad uno step del modello corrisponde a 30 secondi reali (spiegato qui 3.7.4), un'ora è composta da 120 steps. Con questa osservazione è possibile calcolare il valore atteso di clienti in entrata per ogni step al variare delle ore.

Il valore atteso di clienti in ingresso per ogni step al variare delle ore viene mostrato nella figura 3.17.



Figure 3.17: Valore atteso di clienti in ingresso per ogni step al variare delle ore.

Tuttavia emerge chiaramente come questo valore atteso non sia un valore intero in quanto ottenuto dalla divisione tra numero di clienti totali in ingresso in un'ora e numero di step per un'ora. Inoltre anche nel caso in cui il valore atteso dei clienti in ingresso per una determinata ora fosse un numero intero, non è realistico che per ogni step di una determinata ora gli ingressi siano identici. Per questo motivo si procede aggiungendo del rumore ad ogni step dove il nuovo valore che indica il numero di ingressi per step è frutto della realizzazione della variabile aleatoria X distribuita secondo una normale con media pari al valore atteso del numero di clienti per step e varianza pari a 0.4. In questo modo si vuole simulare il fatto che ad ogni step possano entrare meno clienti di quelli attesi e viceversa per altri step, pur ottenendo come numero di ingressi totali il valore atteso per quella specifica ora.

Come esempio prendiamo in considerazione le ore 12. In questo orario in media entrano 231 clienti all'interno del supermercato, vale a dire 1.931 ingressi per step. Si procede quindi campionando 120 valori (numero di step in un'ora) distribuiti secondo una normale con media 1.931 e varianza 0.4. A parire dal valore continuo x , si procede discretizzando il numero di clienti in ingresso per ogni step utilizzando la funzione così definita:

$$f(x) = \begin{cases} 0 & \text{se } x \leq 0 \\ \lfloor x + 0.5 \rfloor & \text{se } x > 0 \end{cases} \quad (3.10)$$

Nella figura 3.18 viene mostrato il risultato dell'operazione di campionamento e discretizzazione. Alle ore 12 il numero totale di clienti in ingresso nel supermercato è 231, si può notare dalla figura che dopo il campionamento e la discretizzazione il numero generato di clienti in ingresso è 232, che si avvicina molto al valore reale.

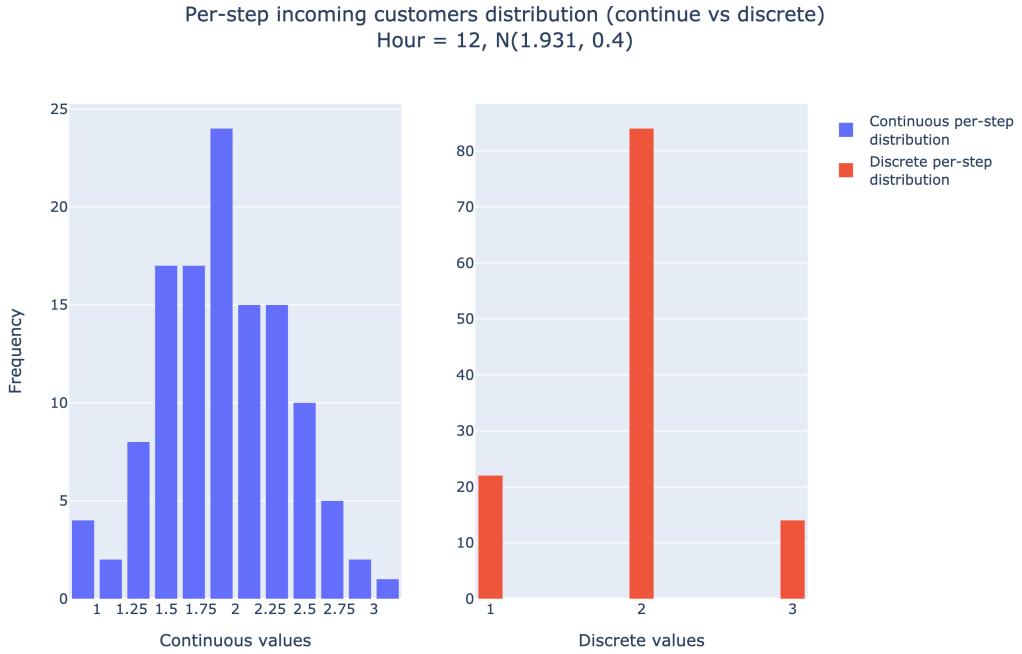


Figure 3.18: Numero di clienti in ingresso per ogni step (distribuzione continua vs discreta).

Al fine di testare questo metodo che consente di generare il numero di clienti in ingresso per ogni step, vengono eseguiti 30 campionamenti diversi per ogni ora al fine di poter confrontare il risultato con la distribuzione reale. Il confronto tra queste due distribuzioni viene mostrato in figura 3.19. Viene inoltre considerato l'errore relativo tra la distribuzione reale e quella generata al variare delle ore. L'errore relativo mostrato in figura 3.20 è ottenuto calcolando:

$$\text{Errore} = \frac{|X_{reale} - X_{generato}|}{X_{reale}}$$

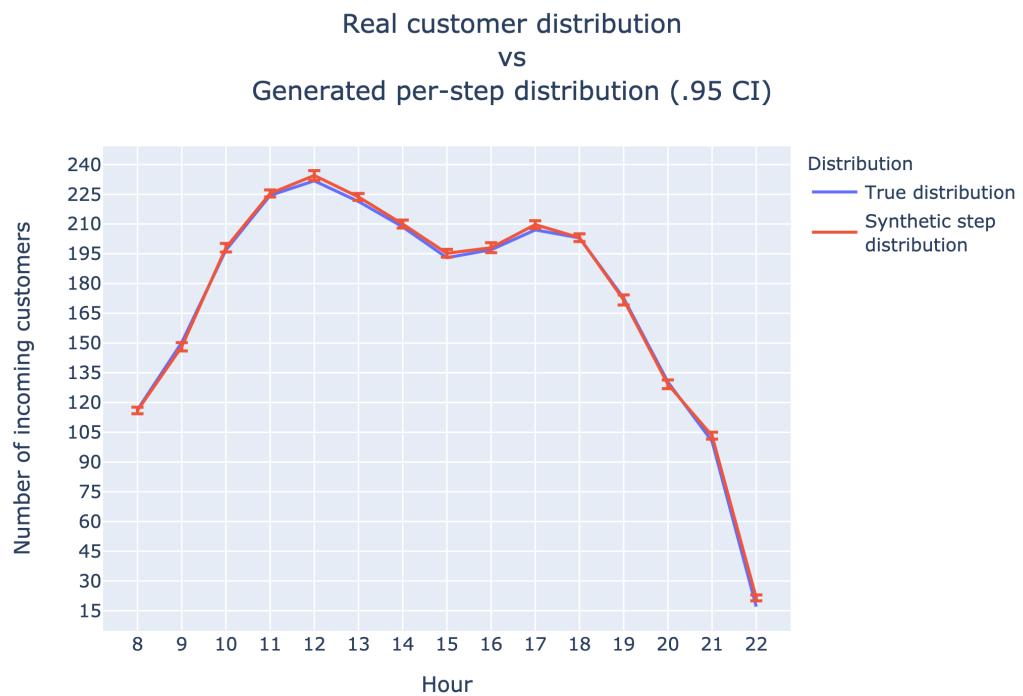


Figure 3.19: Distribuzione reale vs distribuzione generata.

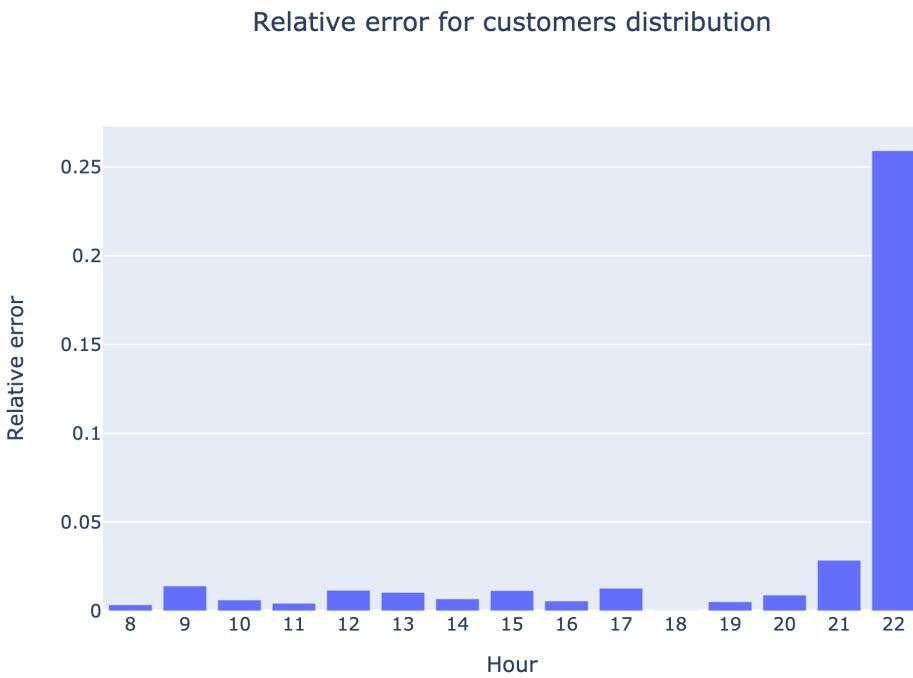


Figure 3.20: Errore tra distribuzione reale e distribuzione generata.

In questo capitolo è stato descritto abbondantemente il modello di supermercato implementato; in particolare si è parlato del workflow e delle capacità degli agenti, oltre che alla flessibilità del modello grazie ai diversi parametri che è necessario specificare per ogni simulazione. Questo ultimo aspetto è un'arma a doppio taglio in quanto molto parametri garantiscono generalizzazione tuttavia portano ad un'esplosione combinatoria delle configurazioni possibili e quindi di difficile tuning in fase di validazione.

Nel capitolo successivo vedremo le strategie di scelta della coda e di jockeying, parte importante della simulazione in quanto l'obiettivo finale del progetto è indagare su quale sia la strategia che minimizza il tempo d'attesa alle casse.

Chapter 4

Implementazione dei comportamenti e delle strategie

In questo capitolo verranno analizzate le strategie di scelta della coda da parte del cliente. In particolare queste strategie determinano la coda che il cliente deciderà di seguire durante la *fase di scelta della coda* e durante la *fase di attesa in coda e jockeying*.

Come introdotto all'inizio della relazione, obiettivo di questo progetto è analizzare le varie configurazioni di casse all'interno del supermercato, al variare del tipo di casse, della quantità di clienti presenti nel negozio e della strategia di scelta della coda dei clienti, pertanto è necessario disporre di strategie che sfruttano calcoli basati su variabili diverse: le variabili in gioco saranno il numero di elementi nel carrello, il numero di persone e il tempo medio di attesa.

Se un cliente entra nel supermercato con l'intenzione di usare le casse self-scan, chiaramente non sarà dotato di strategie di scelta della coda e di jockeying, dal momento che le casse self-scan hanno sempre un'unica coda condivisa. Anche nel caso in cui le casse standard abbiano un'unica coda condivisa, come capita in molti negozi, il cliente non avrà una strategia di scelta della coda o di jockeying. Al contrario invece, dopo la fase di shopping il cliente dovrà scegliere la coda tra le code disponibili delle casse standard e delle casse self-service; una volta che esso si accoda in una cassa standard, può decidere di cambiare coda e quindi effettuare il jockeying, se ritiene, con la propria strategia, che nelle casse a lui più vicine ci sia un tempo minore di attesa.

Uno degli scopi del progetto è indagare sulla configurazione di casse migliori, quindi ad ogni simulazione tutti i clienti avranno la stessa strategia di scelta della coda e di jockeying per creare una netta distinzione tra simulazioni diverse.

4.1 Fase di scelta della coda

Alla fine della *fase di shopping*, il cliente che non vuole andare alla cassa self-scan, deve scegliere quale coda seguire tra quelle disponibili, ovvero tra le code associate a casse aperte.

Una strategia è una scelta della coda q che minimizza una certa funzione $f(q)$, come riportato nel capitolo precedente all'equazione 3.1.

Al fine di definire le strategie, è importante notare che la funzione $\text{basket-size}(c_i)$ non deve essere deterministica, come già introdotto nel capitolo precedente: il cliente fa una stima del numero di elementi nel carrello degli altri clienti, non ne è però certo, per questo questa funzione ha una probabilità di errore che rende la stima più veritiera, e si può ridefinire:

$$\text{estimate-basket-size}(c_i) = \text{basket-size}(c_i) + e_i \quad (4.1)$$

dove e_i è l'errore commesso nella stima e può essere sia positivo che negativo. L'errore dipende dalla grandezza del carrello, in particolare più elementi ci sono nel carrello più l'errore aumenta; intuitivamente in questo modello si è deciso di far variare l'errore in modo logaritmico rispetto alla grandezza del carrello, in base alla formula:

$$e_i = \frac{\text{basket-size}(c_i)}{d} \quad (4.2)$$

dove $d \in \mathbb{R}$, $d \geq 1$ e se $d = 1$ non viene commesso alcun errore sulla stima. La quantità $\text{estimate-basket-size}$ diventa dunque una variabile aleatoria normale con media la basket size reale e deviazione standard l'errore commesso sulla stima.

Le 4 strategie di scelta della coda prese in considerazione per il modello sono:

1. **Minimo numero di elementi:** viene scelta la coda con il minimo numero di elementi nei carrelli di tutte le persone in attesa. La funzione da minimizzare è quindi per una coda $q \in Q$ dove Q è l'insieme di tutte le code disponibili:

$$f(q) = \sum_{i=1}^N \text{estimate-basket-size}(c_i) \quad (4.3)$$

dove $c_i \in q$ è un cliente in coda, $\text{estimate-basket-size}(c_i)$ è il numero di elementi che ha nel suo carrello e $N = |q|$ è il numero di clienti in coda in q .

2. **Minimo numero di persone:** viene scelta la coda con il minimo numero di persone accodate. La funzione da minimizzare per $q \in Q$ è:

$$f(q) = |q| \quad (4.4)$$

3. **Minimo tempo d'attesa in base al tempo di servizio medio:** viene scelta la coda con il tempo d'attesa minimo, calcolato in base al tempo di servizio medio. Il *tempo di servizio totale di una coda* è calcolato come somma del tempo di servizio per ogni cliente della coda. Il tempo di servizio per un cliente comprende il **tempo di transazione** e il **tempo di pausa** tra un cliente e un altro, e variano a seconda del tipo di cassa, come illustrato nella sezione 3.4. Le formule per il calcolo dei tempi per la cassa standard 3.2 e 3.3, comprendendo la stima della grandezza del carrello, diventano dunque:

$$\text{transaction-time}_i = e^{alog(\text{estimate-basket-size}(c_i)) + b} \quad (4.5)$$

$$\text{break-time}_i = \frac{\beta^\alpha \text{estimate-basket-size}(c_i)^{\alpha-1} e^{-\beta \text{estimate-basket-size}(c_i)}}{\Gamma(\alpha)} \quad (4.6)$$

e per la cassa self-service, le 3.6 e 3.7 diventano:

$$\text{transaction-time}_i = e^{alog(\text{estimate-basket-size}(c_i)) + b} \quad (4.7)$$

$$\text{break-time}_i = e^{clog(\text{estimate-basket-size}(c_i)) + d} \quad (4.8)$$

I tempi di servizio per ogni cliente di una coda sono quindi sommati per calcolare il tempo servizio totale per quella coda. Il tempo di servizio totale di una coda q_j , $j = 1, \dots, M$, dove M è il numero totale di code del supermercato, è pertanto:

$$\text{total-service-time}(q_j) = \sum_{i=1}^N (\text{transaction-time}_i + \text{break-time}_i) \quad (4.9)$$

Il tempo di servizio medio per le code è la somma dei tempi totali di servizio divisa per il numero di code. Viene scelta la coda con il tempo totale minimo, mettendo insieme le equazioni 4.5, 4.6, 4.7, 4.8 e 4.9 si ottiene la funzione da minimizzare:

$$f(q) = |q| * \frac{1}{M} \sum_{j=1}^M (\text{total-service-time}(q_j)) \quad (4.10)$$

4. **Minimo tempo d'attesa in base alla *power regression*:** viene scelta la coda con il tempo d'attesa minimo, calcolato in base al tempo di transazione e il tempo di pausa medi per quella coda. Il tempo di servizio totale per una coda è calcolato anche qui in base alla 4.9. La funzione da minimizzare è:

$$f(q) = |q| * \text{total-service-time}(q) \quad (4.11)$$

4.2 Fase di attesa in coda e jockeying

Quando il cliente è nella *fase di attesa in coda* ha la possibilità di cambiare la propria scelta se nota che nelle code adiacenti il tempo di attesa è minore che nella propria; le code adiacenti implicate nel calcolo sono quelle che hanno distanza minore o uguale del **parametro di adiacenza**, descritto nel capitolo precedente, rispetto a dove il cliente è già accodato. Una coda è distante 1 da un'altra coda se fisicamente nella griglia della simulazione esse compaiono una di fianco all'altra. Nella pratica la coda in cui il cliente è presente in un determinato momento svolge la funzione di coda pivot e le altre code prese in considerazione per fare jockeying sono quelle adiacenti alla coda pivot.

Le strategie seguite nel jockeying sono due, e coincidono con le prime due strategie di scelta della coda, descritte nella sezione precedente. Anche in questo caso il cliente stima le grandezze dei carrelli degli altri clienti in base alle 4.1 e 4.2. Entrambe le strategie necessitano di un **threshold**, che è un parametro del modello; dopo aver trovato la coda adiacente con tempo di attesa minore, viene calcolato un "guadagno" di fare jockeying, e se questo guadagno è più alto del threshold, allora il cliente cambia coda, altrimenti no; questo parametro serve a simulare il fatto che le persone in coda tendono sempre a preferire la propria posizione corrente, e a cambiare coda solo quando conviene tanto. Inoltre abbiamo pensato che non tutte le persone al supermercato vogliono fare jockeying, ci sono alcuni che neanche lo considerano e non guardano le casse adiacenti per valutarne il tempo medio d'attesa, pertanto abbiamo impostato un parametro che rappresenta la probabilità di fare jockeying, in modo che prima di scegliere si estragga un numero casuale e si decida. Le strategie possibili sono:

1. **Minimo numero di elementi:** viene scelta la coda con il minimo numero di elementi nei carrelli di tutte le persone in attesa. Il "guadagno" nel cambiare coda è calcolato come:

$$g = \#\text{elementi nei carrelli nella coda pivot} - \min_{q \in Q_{\text{adj}}} \#\text{elementi nei carrelli} \quad (4.12)$$

dove Q_{adj} è l'insieme delle code adiacenti alla coda corrente. Se questo valore è più alto del threshold, significa che il guadagno è abbastanza alto e il cliente cambia coda.

2. **Minimo numero di persone:** viene scelta la coda con il minimo numero di persone accodate. Il "guadagno" nel cambiare coda è calcolato come:

$$g = i - \min_{q \in Q_{\text{adj}}} |q| \quad (4.13)$$

dove $0 < i \leq |q_{\text{current}}|$ è la posizione del cliente nella coda corrente.

Nella figura 4.1 è riportato il grafico che rappresenta il comportamento del cliente in fase di jockeying.

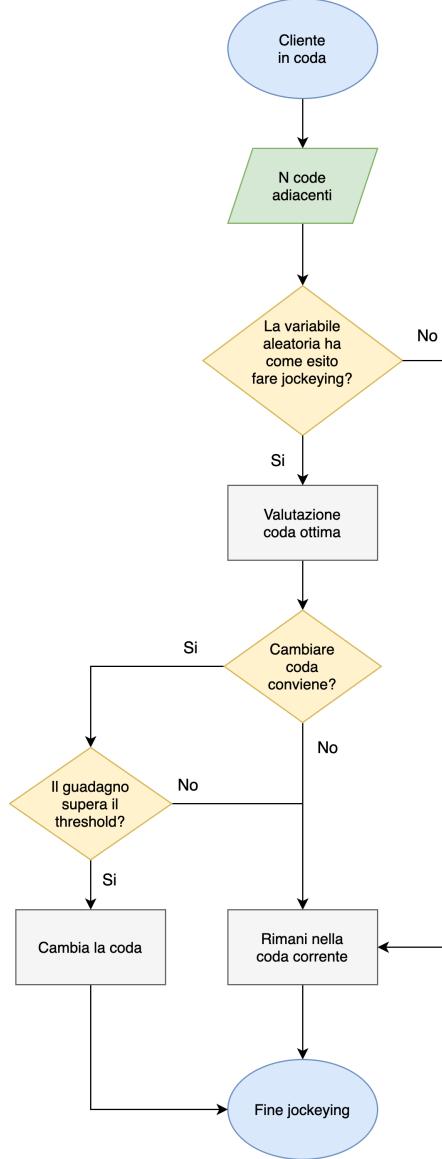


Figure 4.1: Workflow jockeying.

In questo capitolo sono state descritte le strategie utilizzate nelle fasi di scelta della coda e di jockeying da parte del cliente; le strategie di scelta della coda come si è visto sono 4 e sono state descritte in ordine di complessità; le strategie per fare jockeying sono invece 2 e si basano solamente sul numero di elementi nel carrello e sul numero di clienti in coda.

Probabilmente non tutte le strategie sono realistiche e utilizzate concretamente dalle persone in coda al supermercato, l'obiettivo non è indagare sulla scelta effettuata dagli umani in coda, ma capire quale sia la strategia migliore che può essere implementata in un vero supermercato ad esempio con un display che suggerisce quale sia la coda migliore in un dato istante. Nel prossimo capitolo finalmente vedremo i risultati del nostro modello, in particolare metteremo a confronto la nostra simulazione con quella dell'articolo [1] allo scopo di calibrare e validare il modello; introdurremo quindi man mano le espansioni che abbiamo apportato, lo scopo rimarrà sempre quello di diminuire i tempi d'attesa dei clienti e migliorare la loro esperienza.

Chapter 5

Simulazione e analisi dei risultati

In questo capitolo vedremo i risultati delle simulazioni eseguite sul nostro modello. Introdurremo una misura che ci aiuterà a disegnare dei grafici per mettere a confronto simulazioni fatte con strategie di scelta della coda diverse e anche con parametri diversi.

Al fine di analizzare i risultati è stata introdotta la definizione di densità di clienti per ogni step di simulazione; questa misura ci permette di misurare l'affluenza dei clienti nel supermercato e come questa viene gestita dalle casse che hanno il potere di attivarsi o disattivarsi in base alla quantità di clienti presenti. La densità ci permetterà di disegnare i grafici fondamentali che mostrano per ogni simulazione come le casse smaltiscono l'afflusso di clienti in entrata, soprattutto nei periodi critici (in cui il supermercato raggiunge una quantità di clienti che "mettono alla prova" le casse).

La **densità** di clienti per step corrisponde al numero di clienti medio per ogni cassa; la densità allo step i è:

$$\text{density}_i = \frac{\# \text{ customers in the supermarket}}{\# \text{ cashdesks}} \quad (5.1)$$

Si terrà conto della densità totale, della densità per le casse standard o self-service e della densità per le casse self-scan.

5.1 Simulazione con scopo di validazione

Stato dell'arte Al fine di validare il modello, come primo esperimento è stata condotta una simulazione con gli stessi parametri usati nell'articolo [1]. Nella loro simulazione il supermercato contiene 20 casse standard e 6 casse self-service; si noti che nel lavoro citato, le casse si attivano o disattivano in base ai dati raccolti dai ricercatori nei supermercati, nel nostro modello invece le casse si attivano in base al numero di clienti presenti nel supermercato, in base a un parametro che è possibile cambiare nel momento in cui si raccolgono i dati. Vengono messe a confronto le strategie diverse di scelta della coda da parte dei clienti, arrivando alla conclusione che la strategia che porta al minor tempo d'attesa è quella chiamata "minimo tempo d'attesa in base alla *power regression*" nel capitolo 4; tuttavia i risultati portano a dire che questo scenario non è quello ottimale per tutti i supermercati negli orari di punta.

Nelle prossime simulazioni verrà testato il jockeying, in particolare si metteranno a confronto le diverse strategie di scelta della coda e di jockeying per stabilire se questo porti a un miglioramento dell'esperienza del cliente nel supermercato. Questi esperimenti verranno poi messi a confronto con una simulazione in cui c'è soltanto la coda condivisa per le casse standard, pertanto i clienti non avranno bisogno di scegliere la coda né di fare jockeying: in base a quanto detto nell'articolo [3], questo tipo di coda dovrebbe portare a tempi di attesa minori. Verranno quindi introdotte nella simulazione le casse self-scan con rilettura randomica, si prevede che queste portino a una diminuzione considerevole dei tempi d'attesa in coda, in quanto viene a mancare la fase di scanner da parte dei cassieri. Infine verrà effettuata una simulazione con gli elementi probabilistici già descritti nel corso dei capitoli 3 e 4, che rendono l'esperimento non deterministico e a nostro riguardo più realistico.

Simulazione con gli stessi parametri Come validazione sono state effettuate delle simulazioni con gli stessi parametri usati nell'articolo [1]: 20 casse standard con code parallele, 6 casse self-service, no jockeying; è stata ripetuta la stessa simulazione 4 volte però con le 4 strategie diverse di scelta della coda.

Total customers - Ground truth vs simulations for validation (Normalized)

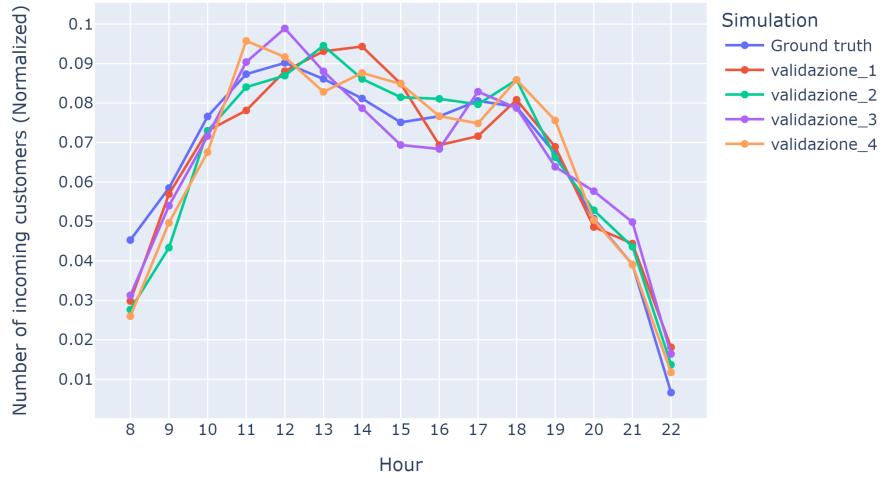


Figure 5.1: Numero di clienti totali al variare dell'ora

Il grafico ?? funge da validazione: mettiamo a confronto il numero totale di clienti (normalizzati) presenti nel negozio ad ogni ora per ogni simulazione e per la simulazione dell'articolo. Si nota come, a parità di parametri, il nostro modello e il modello di [1] supportino allo stesso modo il numero di clienti presenti nel negozio, facendoli defluire in modo paragonabile.



Figure 5.2: Tempo medio d'attesa al variare dell'ora

Nell'immagine ?? si può vedere la variazione del tempo medio di attesa in coda dei clienti nel corso della giornata lavorativa che va dalle 8:00 alle 22:00. Le simulazioni variano per strategia di scelta della coda da parte dei clienti, si nota quindi che, in assenza di jockeying, la strategia 3 risulta quella che durante l'arco della giornata mantiene il tempo medio più basso, anche se a fine giornata tutte e 4 le strategie raggiungono una media paragonabile.

Nelle prossime simulazioni verranno aggiunte delle caratteristiche che pensiamo possano migliorare il tempo medio d'attesa dei clienti, che metteremo a confronto con questa simulazione la quale rappresenta il nostro punto di partenza.

5.2 Simulazione con strategie *jockey*

Introduciamo dunque le 2 strategie di jockey descritte nel capitolo 4. In queste simulazioni sono presenti 20 casse normali con code parallele, 6 casse self-service, 4 strategie di scelta della coda e 2 strategie di jockeying, per un totale di 8 simulazioni.



Figure 5.3: Tempo medio d'attesa al variare dell'ora

Nell'immagine ?? si mettono a confronto 2 strategie di scelta della coda: minimo numero di elementi, minimo numero di persone in coda. Le due strategie sono affiancate da nessun jockeying, jockey per minimo numero di elementi e jockey per minimo numero di persone in coda.

Si osserva che i risultati migliori si raggiungono quando si fa jockey e inoltre la strategia di scelta della coda è la stessa del jockey, con una diminuzione massima del tempo medio di attesa di circa 2 timestamp (1 minuto) e della media finale di 1 timestamp (30 secondi) rispetto alle simulazioni senza jockey. Si può dire pertanto che mischiare le strategie, ad esempio scegliendo la coda per numero di persone e facendo jockey per numero di elementi nei carrelli, peggiori le prestazioni del sistema rispetto al tempo di attesa dei clienti. Questo è in linea con quanto si osserva nell'articolo [6], che afferma che a parità di parametri un sistema con jockeying è più efficiente di uno senza.

KDE (Kernel Density Estimates) - jockey and parallel queues vs N-fork queue

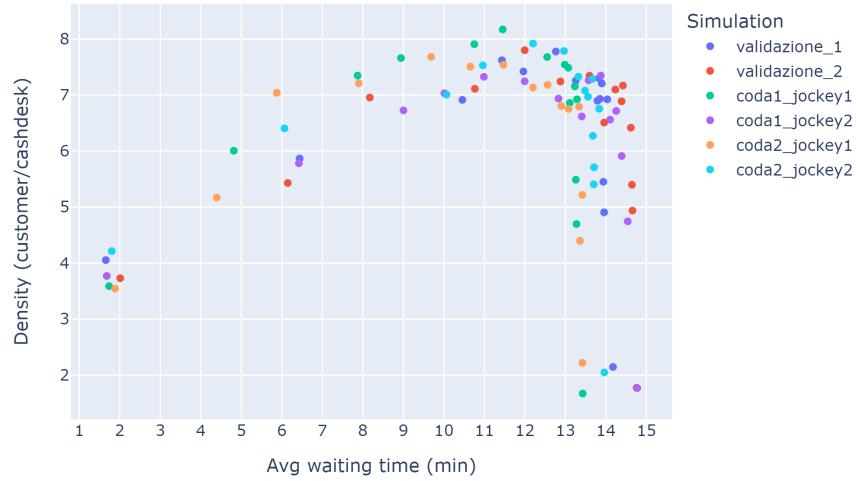


Figure 5.4: KDE - densità al variare del tempo medio d'attesa

Nell'immagine ?? è riportato un grafico KDE (Kernel Density Estimates), che mostra la densità di clienti nel negozio (come definita a inizio capitolo) al variare del tempo medio d'attesa. Questo grafico mostra che con l'aumentare della quantità di clienti nel negozio aumenta anche il tempo d'attesa in coda, come ci si aspetterebbe; si nota che viene raggiunto un "punto critico" in cui sia la densità che il tempo medio d'attesa sono molto alti, superato il quale il sistema reagisce facendo defluire più rapidamente i clienti, il cui tempo d'attesa medio crolla. Questo accade sia perché le casse vengono aperte dato che il numero di clienti aumenta e anche perché raggiunta una certa ora il negozio chiude e non entrano più clienti.



Figure 5.5: Tempo medio d'attesa al variare dell'ora

Nell'immagine ?? vengono messe a confronto 2 strategie di scelta della coda: minimo tempo d'attesa in base al tempo di servizio medio, minimo tempo d'attesa in base alla power regression. Le due strategie sono affiancate da nessun jockeying, jockey per minimo numero di elementi e jockey per minimo numero di persone in coda.

In questo caso non vale quanto detto sopra, le strategie di scelta della coda e di jockey non possono essere le stesse e non si può dire che affiancandole si ottengono risultati migliori; questo potrebbe rappresentare un ulteriore studio da fare per allargare il nostro modello.

Si nota come la strategia di scelta della coda che mantiene il tempo d'attesa medio minore durante tutta la giornata sia la power regression affiancata da jockey per minimo numero di persone in coda; questa simulazione raggiunge un tempo medio d'attesa minore finale di circa 1.5 timestamp rispetto alla simulazione peggiore. Questa però è comunque molto vicina al tempo medio della strategia di scelta della coda per minimo tempo di servizio medio e senza jockeying, anche se quest'ultima tende a peggiorare nell'ora finale.

Average waiting time - best simulations with jockey

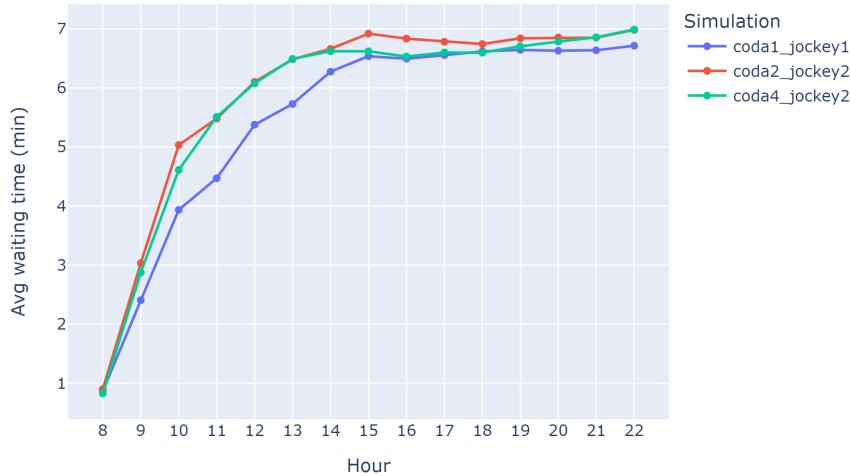


Figure 5.6: Tempo medio d'attesa al variare dell'ora

Nel grafico ?? mettiamo a confronto il tempo medio d'attesa delle migliori strategie viste fin ora: il tempo medio finale è circa lo stesso per tutte e tre, ma si nota che il grafico della prima strategia rimane sotto gli altri nell'arco di tutta la giornata.

5.3 Simulazione con coda condivisa

Con le prossime simulazioni indagheremo sulla diminuzione del tempo medio d'attesa nel caso di coda condivisa (N-fork): metteremo a confronto la strategia migliore trovata fin ora (scelta della coda e jockey per minimo numero di elementi) con il caso di una coda condivisa per tutte le casse.

Come già specificato, nei casi esaminati fin ora le casse si aprivano e chiudevano in maniera automatica in base al numero di clienti presenti nel negozio e che necessitano di essere serviti; nel caso di coda condivisa questa funzione non è stata implementata, e potrebbe rappresentare un'estensione del nostro modello. Per questo motivo abbiamo deciso di diminuire il numero di casse nella simulazione: in base a osservazioni fatte in fase sperimentale abbiamo notato che il numero di casse aperte massimo in un dato momento è 8, chiaramente rispetto alla distribuzione dei clienti in entrata utilizzata; per questo motivo abbiamo voluto indagare i casi di 5, 6, 7 e 8 casse standard presenti nel negozio (e 6 casse self-service come fatto in precedenza).

Average waiting time - jockey and parallel queues vs N-fork queue

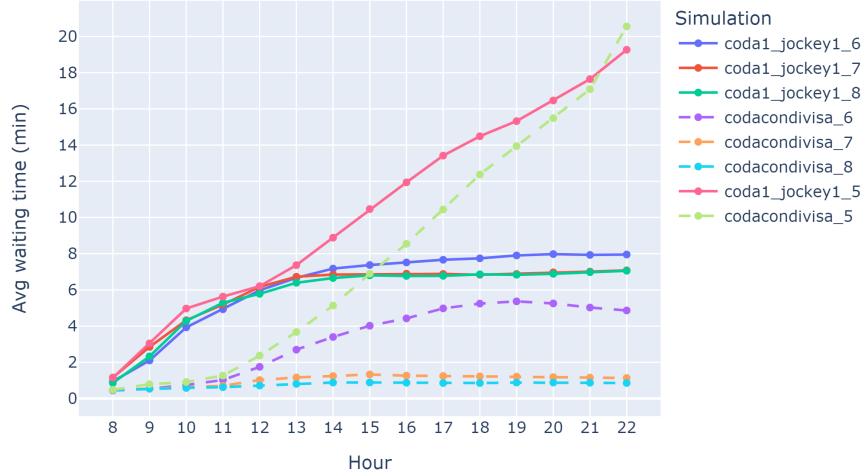


Figure 5.7: Tempo medio d'attesa al variare dell'ora

Nel grafico ?? riportiamo i tempi medi d'attesa delle simulazioni con code parallele e jockey rappresentandoli con delle linee, e i tempi medi d'attesa delle simulazioni con coda condivisa rappresentandoli con dei trattini. Si nota immediatamente che nel caso di 5 casse il tempo medio d'attesa sale vertiginosamente a circa 20 minuti sia per le code parallele che per la coda condivisa: questo è imputabile alla distribuzione dei clienti in entrata, il supermercato con 5 casse è troppo piccolo per gestire questa quantità.

Una cosa interessante da notare invece è la divisione netta nel grafico tra le simulazioni con code parallele e quelle con coda condivisa: l'utilizzo della coda condivisa abbassa i tempi medi d'attesa durante tutta la giornata, anche rispetto al caso in cui i clienti fanno jockey. Questi risultati sono in linea con l'articolo [3], in cui si mette in luce il fatto che l'utilizzo di una coda condivisa migliori le prestazioni del negozio.

KDE (Kernel Density Estimates) - jockey and parallel queues vs N-fork queue

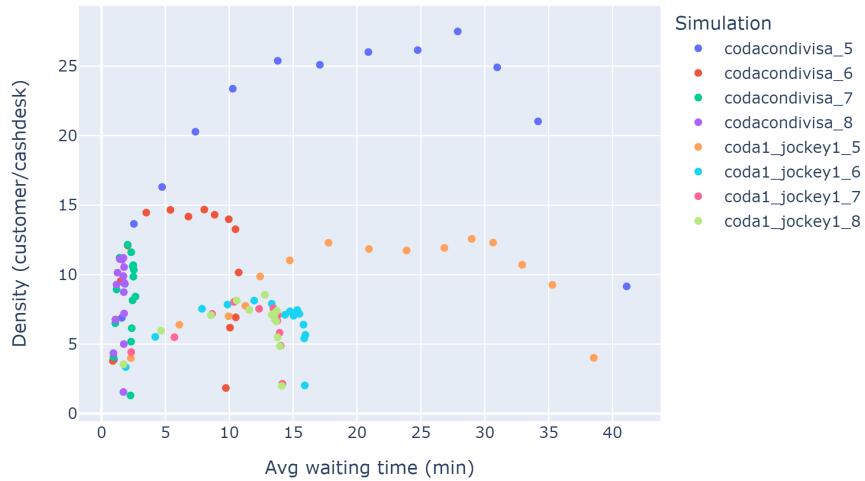


Figure 5.8: KDE - densità al variare del tempo medio d'attesa

Nell'immagine ?? è riportato il grafico KDE delle stesse simulazioni del grafico precedente. Si nota ancora come il negozio con 5 casse non abbia delle buone prestazioni rispetto alla quantità di clienti entranti, sia nel caso di code parallele che nel caso di coda condivisa. Possiamo osservare però che nel caso delle code condivise si arriva a toccare livelli di densità di clienti più alti, questo accade per un limite del modello, infatti se la coda arriva alla sua capienza massima, i clienti attendono che si liberi per mettersi in coda, creando così più traffico nella zona di shopping, come si può vedere nell'immagine ??; dopo aver superato il momento critico, il grafico ci mostra che il sistema che utilizza l'N-fork è in grado di ritornare meglio a livelli di densità minori del sistema con code parallele, il tutto mantenendo un tempo medio d'attesa basso.

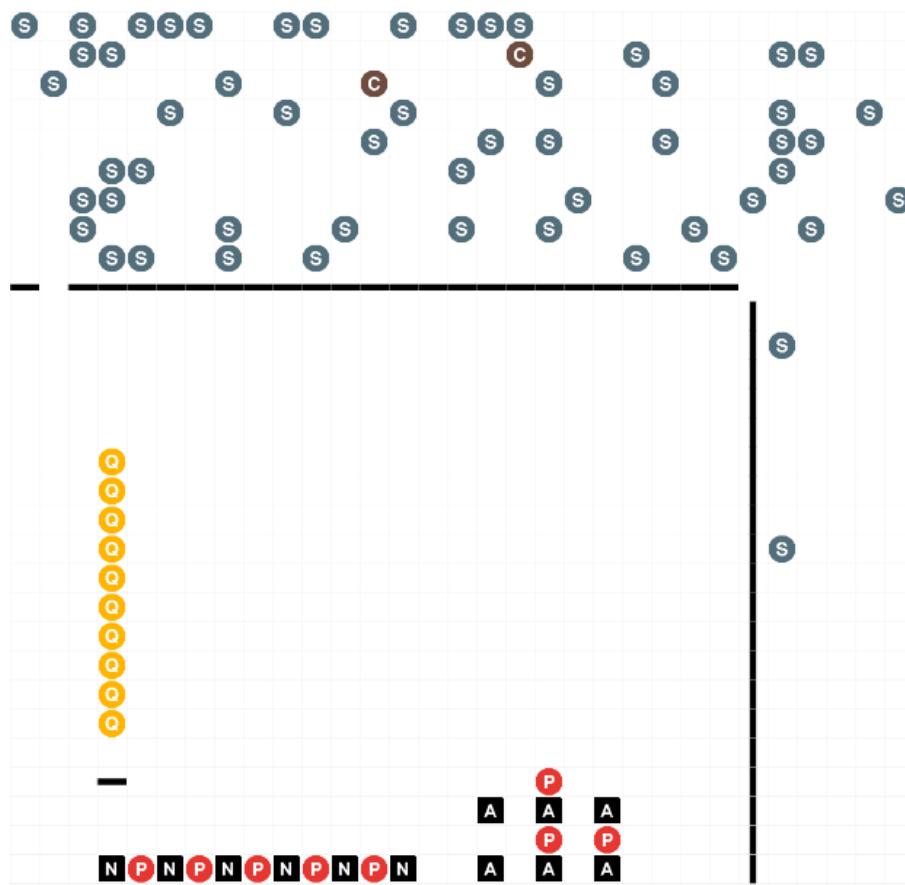


Figure 5.9: Screenshot della simulazione con coda quasi al limite

5.4 Simulazione con casse self-scan

In questa simulazione vedremo il caso in cui nel supermercato sono presenti solamente casse self-scan, in particolare 7; in questa simulazione chiaramente tutti i clienti si recheranno alla cassa self-scan, la probabilità di una rilettura parziale è del 2% e una rilettura totale è dell'1%.

Average waiting time - standard cashdesks vs self-scan cashdesks

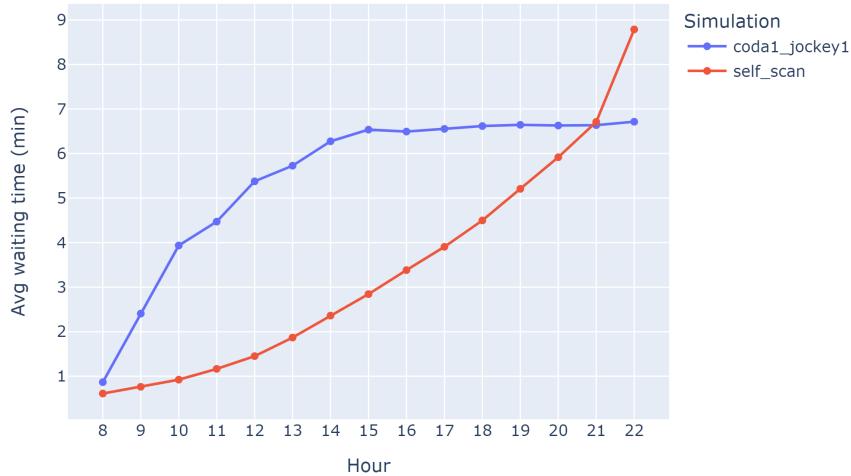


Figure 5.10: Tempo medio d'attesa al variare dell'ora

Si nota immediatamente dal grafico ?? che il tempo medio d'attesa della simulazione con self-scan va ad aumentare sempre più; questo accade a causa delle rilettture, infatti i clienti che non ricevono una rilettura escono immediatamente dal negozio, mentre gli altri restano in coda ad una sola cassa, che funge da collo di bottiglia. Aumentando la probabilità di rilettura, infatti, la simulazione fallisce perché la coda alla cassa riservata diventa troppo lunga.

Rispetto a questa simulazione bisognerebbe avere dati più completi, ad esempio sapere con che probabilità vengono fatte le rilettture nei supermercati o con che distribuzione entrano in negozio i clienti self-scan; questa è una possibile estensione del nostro lavoro.

KDE (Kernel Density Estimates) - standard cashdesks vs self-scan cashdesks

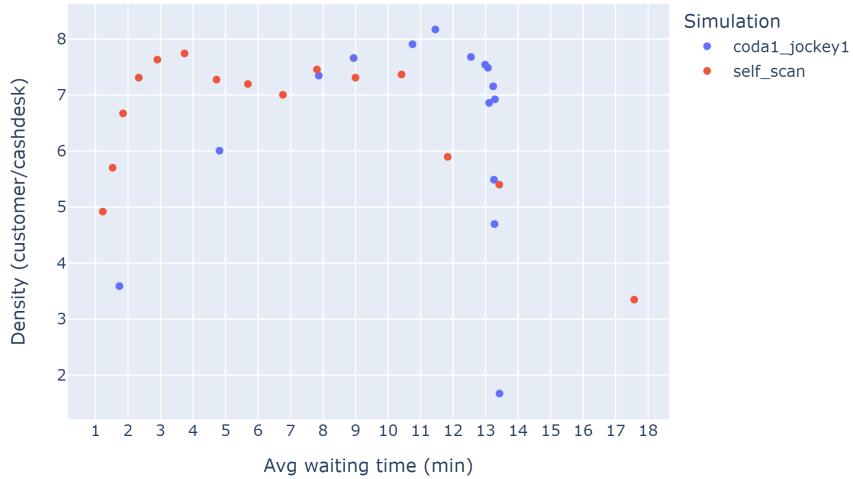


Figure 5.11: KDE - densità al variare del tempo medio d'attesa

L'immagine ?? mostra il grafico KDE della simulazione con sole casse self-scan messa a confronto con la simulazione con sole casse standard e jockey. Si nota che nella simulazione con casse standard c'è un momento di massima densità che viene gestito dal sistema portando a una diminuzione dei tempi medi d'attesa; nella simulazione con casse self-scan questa gestione della densità non avviene altrettanto bene, infatti dopo il punto critico di densità maggiore, questa diminuisce ma il tempo medio aumenta e questo è imputabile ancora una volta alle troppe persone che attendono di ricevere una rilettura della spesa.

5.5 Simulazione non deterministica

L'introduzione di parametri randomici rappresenta la nostra volontà di rendere più verosimile il modello e non un tentativo di miglioramento dei tempi medi d'attesa dei clienti.

Abbiamo voluto testare il modello completo, considerando 20 casse standard con code parallele, 6 casse self-service, 5 casse self-scan; la probabilità che un cliente sia self-scan (nel momento in cui entra nel negozio) è del 50%. Introduciamo anche l'errore di stima del basket-size da parte di un cliente: nel momento in cui un cliente deve calcolare il numero di elementi nel carrello di altri clienti commette un errore con 0.1 di deviazione standard.

Le simulazioni sono 8, combiniamo le strategie di scelta della coda (4) con quelle di jockey (2).

Average waiting time - probabilistic vs deterministic

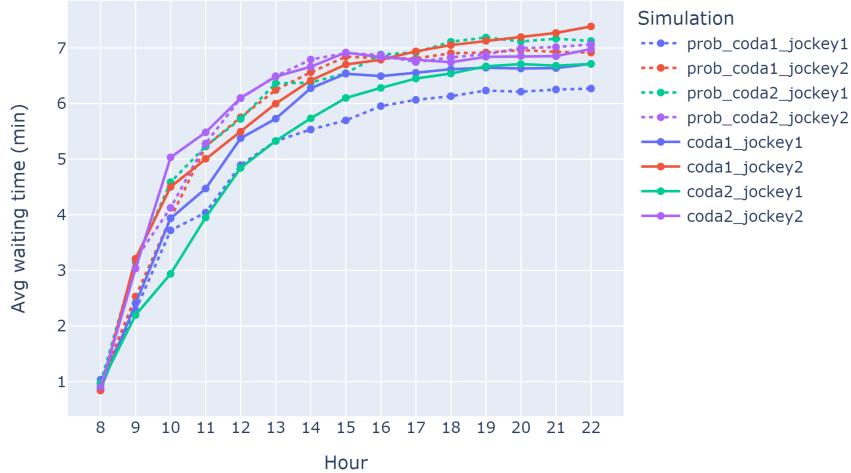


Figure 5.12: Tempo medio d'attesa al variare dell'ora

Notiamo dal grafico ?? che il tempo medio d'attesa delle simulazioni con e senza elementi randomici (raggruppati a coppie per colore, una tratteggiata e una no) è assolutamente paragonabile e questo ci mostra che gli elementi che abbiamo aggiunto, giustificati dal buon senso, non scombinano totalmente il modello ma lo rendono semplicemente più imprevedibile, riuscendo a simulare più scenari possibili.

Dopo aver condotto queste analisi possiamo concludere che il jockey da parte dei clienti porta a una diminuzione dei tempi d'attesa in coda e inoltre la configurazione delle casse a N-fork è in grado di gestire meglio il punto critico di densità di clienti rispetto alla configurazione a code parallele; nel prossimo capitolo verranno fatte le conclusioni e illustrati i possibili sviluppi futuri.

Chapter 6

Conclusioni

In questa relazione è stato illustrato un modello di supermercato basato su agenti, che ha lo scopo di simulare il comportamento dei clienti nel negozio in fase di scelta della coda e attesa di pagamento. Un supermercato è un sistema complesso composto da diversi attori che hanno ognuno un obiettivo, per questo abbiamo deciso di simularlo con un modello ad agenti ed analizzare diverse metriche allo scopo di indagare sulle configurazioni di casse e le strategie migliori che portano a una diminuzione del tempo di attesa dei clienti.

A seguito dell'emergenza pandemica di COVID-19 si è dimostrato necessario uno studio sui flussi di persone negli spazi pubblici e nei luoghi di affollamento la cui frequentazione è necessaria per tutte le persone, per questo uno studio sulla diminuzione del tempo d'attesa in coda può servire per minimizzare i contatti tra le persone e quindi il diffondersi di virus e malattie.

6.1 Sviluppi futuri

Un primo sviluppo futuro naturale che può estendere il modello è l'introduzione dell'elemento spaziale: i clienti nel nostro supermercato si muovono istantaneamente da una cella a un'altra della griglia che rappresenta il negozio e nel calcolo dei tempi d'attesa in coda non tengono conto della distanza dalle casse. Come fatto nell'articolo [3], si potrebbe considerare una configurazione di casse a D-fork, anzichè a N-fork, in cui appunto i clienti tengono conto della distanza dalla coda scelta. La considerazione dello spazio e delle distanze porterebbe anche a una modellazione più precisa dei tempi di movimento di tutti i clienti che, se integrata con una zona shopping più verosimile, potrebbe simulare il supermercato in maniera completa (anche adottando vere piantine dei negozi per fare le simulazioni). In ottica di crisi pandemica, questa estensione aiuterebbe a condurre uno studio sul distanziamento obbligatorio in tempi di virus e influenze, oltre che sulla diminuzione dei tempi passati all'interno del negozio.

Un'altra estensione che è stata prevista sin dal primo momento di programmazione del modello è l'introduzione di nuove strategie di scelta della coda e di jockey: avendo adottato il pattern Strategy è molto semplice definire e utilizzare nuove strategie nelle simulazioni. Come detto nel capitolo 5 l'aggiunta delle strategie di jockey basate sul tempo di servizio medio e sulla power regression porterebbero a considerazioni più approfondite sulla miglior coppia di strategie di scelta della coda e di jockey.

Avendo parametrizzato molti valori delle simulazioni, sarebbe utile avere a disposizione più dati sui supermercati italiani, ad esempio per quanto riguarda le casse self-scan: in fase di analisi dei risultati abbiamo osservato come i dati a nostra disposizione non fossero adatti per la simulazione con sole casse self-scan; sarebbe quindi opportuno avere dati sulla percentuale

di persone che utilizzano questo tipo di casse e sull'algoritmo usato dai supermercati per estrarre le riletture della spesa parziali o totali.

Una criticità del nostro modello è emersa durante la simulazione con casse standard e coda condivisa: in caso di densità alta di clienti le code tendono a riempirsi al massimo e i clienti che vogliono mettersi in coda sono costretti ad attendere in zona shopping (per evitare che una coda sia talmente lunga da sfondare il muro del supermercato). Questa criticità ha portato a un aumento della densità di clienti nella simulazione in oggetto, ma potenzialmente potrebbe accadere per tutte le simulazioni se fatte con una distribuzione di clienti in entrata molto alta. Sicuramente è bene prevedere una capienza massima del supermercato, superata la quale non devono più entrare clienti altrimenti si rischiano tempi d'attesa enormi, però se la coda è condivisa tra le casse è necessaria un'area molto più grande per accodarsi, come avviene in molti negozi al giorno d'oggi.

Bibliography

- [1] T. Antczak, R. Weron, and J. Zabawa, “Data-driven simulation modeling of the checkout process in supermarkets: Insights for decision support in retail operations,” *IEEE Access*, vol. PP, pp. 1–1, 12 2020.
- [2] E. Koenigsberg, “On jockeying in queues,” *Management Science*, vol. 12, no. 5, pp. 412–436, 1966.
- [3] D. Yanagisawa, Y. Suma, Y. Tanaka, A. Tomoeda, K. Ohtsuka, and K. Nishinari, “Methods for improving efficiency of queuing systems,” in *Pedestrian and Evacuation Dynamics*, pp. 297–306, Springer, 2011.
- [4] J. Kazil, D. Masad, and A. Crooks, “Utilizing python for agent-based modeling: The mesa framework,” in *Social, Cultural, and Behavioral Modeling* (R. Thomson, H. Bisgin, C. Dancy, A. Hyder, and M. Hussain, eds.), (Cham), pp. 308–317, Springer International Publishing, 2020.
- [5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. USA: Prentice Hall Press, 3rd ed., 2009.
- [6] S. H. Xu and Y. Q. Zhao, “Dynamic routing and jockeying controls in a two-station queueing system,” *Advances in applied probability*, vol. 28, no. 4, pp. 1201–1226, 1996.