UNIVERSITA' DEGLI STUDI DI PADOVA

Facoltà di Scienze Statistiche

Corso di laurea in Scienze Statistiche e Tecnologie Informatiche

Tesi di Laurea Triennale

Analisi preliminare per la costruzione di un pacchetto software per la simulazione dello spazio aereo di un aeroporto.

Relatore:

Giovanni Andreatta

Laureando:

Paolo Moras

matricola: 515300

Anno accademico 2007-2008

INDICE

INTRODUZIONE

La simulazione è una materia nata con lo scopo di studiare dei sistemi presenti nel mondo per valutarne i comportamenti in condizioni diverse da quelli in cui operano. Si basa sulla costruzione di opportuni modelli che imitino tali comportamenti in condizioni di laboratorio, ossia che siano controllabili, manipolabili e riproducibili. Il fine principale è la valutazione di stati diversi dallo standard senza il rischio di modificare il sistema stesso. Le fasi del processo di simulazione sono elencate di seguito:

1. Definizione degli obiettivi.
2. Analisi del sistema.
3. Raccolta e analisi dei dati.
4. Costruzione del modello.
5. Codifica del modello su calcolatore.
6. Validazione del modello.
7. Conduzione degli esperimenti.
8. Analisi dei risultati.

Questa tesi mira, in particolare, all’analisi preliminare per la costruzione di un pacchetto software per la simulazione della lato aereo di un aeroporto, cioè ad

un’analisi del sistema con lo scopo di realizzare opportuni modelli per ricostruire il comportamento di un aeroporto nel gestire la domanda di utilizzo da parte degli aeromobili e nell’ottimizzare l‘impiego delle sue funzioni (piste, taxiways, stand). Inoltre sarà presentato un programma in linguaggio C++, implementato con il fine di fornire un esempio pratico all’aspetto teorico. Nella fattispecie i dati impiegati per la simulazione si riferiranno all’Aeroporto Internazionale di Atene *“Elefthérios Venizélos“*, gentilmente forniti dal Dipartimento di Matematica Pura ed Applicata dell’Università degli Studi di Padova.

Nella realtà esistono già una moltitudine di software sviluppati con lo scopo di assistere ed aiutare i manager e gli operatori aeroportuali nella pianificazione, costruzione e gestione delle operazioni aeroportuali. Tali software si possono distinguere in funzione del tipo di studio utilizzato o delle metodologia adoperata dai modelli che lo compongono. Il livello di dettaglio dello studio effettuato si divide in macroscopico e microscopico. I modelli macroscopici forniscono risposte approssimate per la pianificazione e la gestione delle operazioni presentato una vasta scelta di alternative. Quelli microscopici invece sono concepiti per affrontare le questioni più dettagliatamente ed ha come obiettivo fornire una rappresentazione altamente fedele dei processi che si svolgono nell’aeroporto. Anche la metodologia si può distinguere in analitica e simulazione. I modelli analitici ricavano stime delle entità di interesse, le quali spesso sono capacità e ritardi; quelli di simulazione invece creano le istanze (aerei) che si muovono all’interno delle sezioni dell’aeroporto descritte dal modello stesso. I principali esempi per comprendere queste distinzioni possono

essere MACAD (Mantea Airfield Capacity And Delay Model). Il vantaggio del primo è il fornire uno strumento che permetta analisi di scenari e ipotesi su condizioni diverse e future, in modo rapido, affidabile e senza notevoli sforzi. Difatti accetta come input, oltre ai dati dell'aeroporto analizzato, l'orario schedulato completo dei voli in arrivo e in partenza, fornendo in output delle stime della capacità e dei ritardi associati ad ogni elemento dell'aeroscalo. Utilizza dei modelli di tipo macroscopici ed analitici. Il secondo invece provvede ad assistere nella costruzione e modifica dettagliata dei piani dell'aeroporto. Ha una funzionalità maggiore, in quanto effettua uno studio più ampia rispetto a MACAD, con alcuni svantaggi quali la grossa richiesta di tempo, sforzo e denaro. Inoltre non è di facile utilizzo e esige una preparazione specifica che può richiedere anche mesi di studio. Non è atto a fornire una risposta immediata alle esigenze degli operatori, ma è maggiormente utile per studiare più dettagliatamente le funzioni e gli elementi dell'aeroporto stesso. A differenza di MACAD utilizza dei modelli microscopici ed effettua una simulazione.

CAPITOLO 1: STRUTTURA DI UN AEROPORTO, OPERAZIONI EFFETTUATE DAGLI AEROMOBILI E FUNZIONI DI PROBABILITÀ

1.1: STRUTTURA DI UN AEROPORTO E PRINCIPALI LAYOUT

La struttura di un aeroporto influisce in modo preponderante all'efficienza delle operazioni, alla flessibilità e ad una possibile crescita futura del sistema. Difatti esistono una vasta serie di layout standard ideati nel corso degli anni grazie all'esperienza degli enti aeronautici civili nazionali ed internazionali, i quali promuovono da sempre il perseguimento di un'elevata soglia di sicurezza.

Nella progettazione bisogna tener conto del fatto che il lato aereo occupa più del 70% dell'area totale dedicata ed è influito soprattutto dal numero di piste, dal loro orientamento, dalla configurazione geometrica e dallo spazio dedicato ad una crescita futura. Errori che spesso si commettono possono essere la mancanza di possibilità per un espansione futura, la costruzione di troppe infrastrutture nella fase iniziale, la mancanza di integrazione ed coordinamento tra i vari componenti e l'assenza di stima degli effetti economici nella scelta del design. Quindi l'obiettivo finale è la costruzione in funzione delle reali esigenze attuali con la prospettiva di una crescita futura.

Un'altra importante componente è la copertura del vento. Infatti atterraggi e

partenze sono tipicamente condotte controvento, salvo alcuni particolari e limitati casi in cui è permesso effettuare operazioni con vento “in poppa” o laterale. Da qui deriva l’orientamento magnetico delle piste, deciso in funzione di studi sul comportamento del vento nell’area di costruzione dell’aeroporto. Tale orientamento magnetico deve essere concordato di modo che la copertura favorevole del vento sia di almeno del 95%; in altre parole la pista in questione non deve essere soggetta a restrizioni causate dal vento stesso per non più del 5% del suo utilizzo.

Una classificazione degli aeroporti è fornita dall’ICAO (International Civil Aviation Organization), la quale li codifica in funzione della dimensione degli aerei che possono ospitare e della lunghezza della pista. Un esempio è riportato nella tabella 1-1.

Codice ICAO Elemento 1		**Codice ICAO Elemento 2**	
Numero	**Lunghezza pista (RFL)**	**Lettera**	**Apertura Alare (WS)**
1	RFL < 800 m	**A**	WS < 15 m
2	800 m ≤ RFL < 1200 m	**B**	15 m ≤ WS < 24 m
3	1200 m ≤ RFL < 1800 m	**C**	24 m ≤ WS < 36 m
4	1800 m ≤ RFL	**D**	36 m ≤ WS < 52 m
		E	52 m ≤ WS < 65 m
		F	65 m ≤ WS < 80 m

Tabella 1-1 (fonte: *“Airport Systems”* R. de Neufville, A. Odoni)

Ad esempio un aeroporto di classe 4E potrebbe ospitare tutti i modelli di aeromobile che abbiano un’apertura alare al massimo di 65 m, che corrisponde a

tutti gli esistenti ad eccezione del nuovo Airbus A380, il quale per poter atterrare ha invece bisogno di un aeroscalo di classe superiore, 4F.

Di seguito sono riportati alcuni layout di aeroporti:

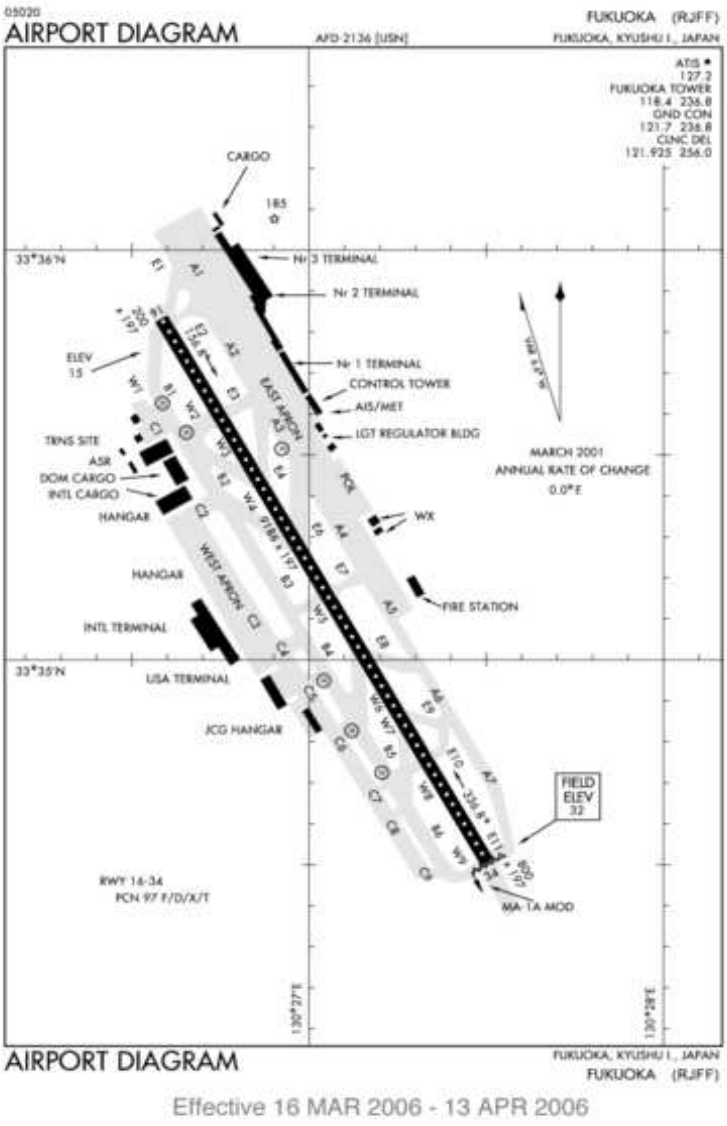

Figura 1-1

La Figura 1-1 rappresenta l'aeroporto di Fukuoka in Giappone, dove è presente un'unica pista. Si possono notare le molteplici taxiway ad uscita rapida, che permettono agli aeromobili atterrati di liberare la pista con maggior velocità. Le configurazioni di utilizzo sono limitate in quanto è presente un singolo servente

per gli atterraggi ed i decolli, con un unico orientamento.

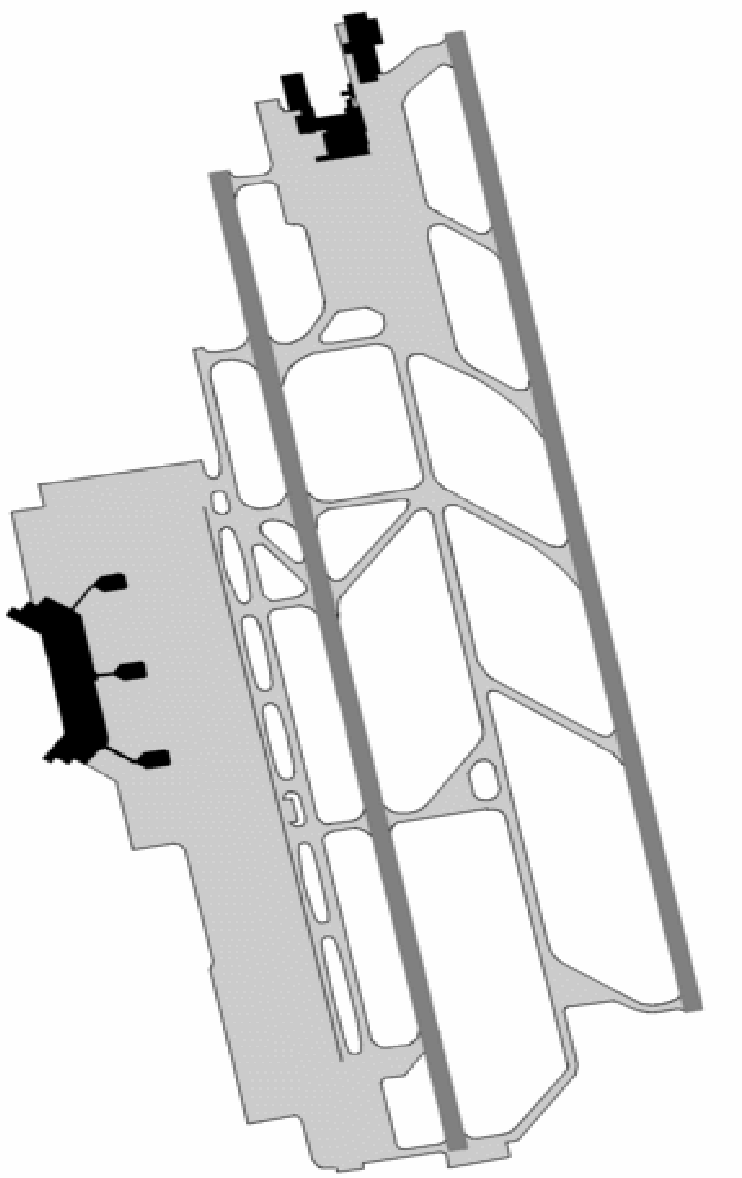

Figura 1-2

La Figura 1-2 mostra il layout dell'aeroporto di Milano Malpensa, il quale ha invece 2 piste parallele. In nero sono evidenziati i 2 terminal. Chiaramente in questo caso, rispetto al precedente, le configurazioni possono essere maggiori ed a seconda delle necessità adoperare entrambe le piste per operazioni miste di decollo e atterraggio oppure, come spesso accade, utilizzarne una per gli arrivi ed

una per le partenze, in modo da semplificare la gestione del traffico.

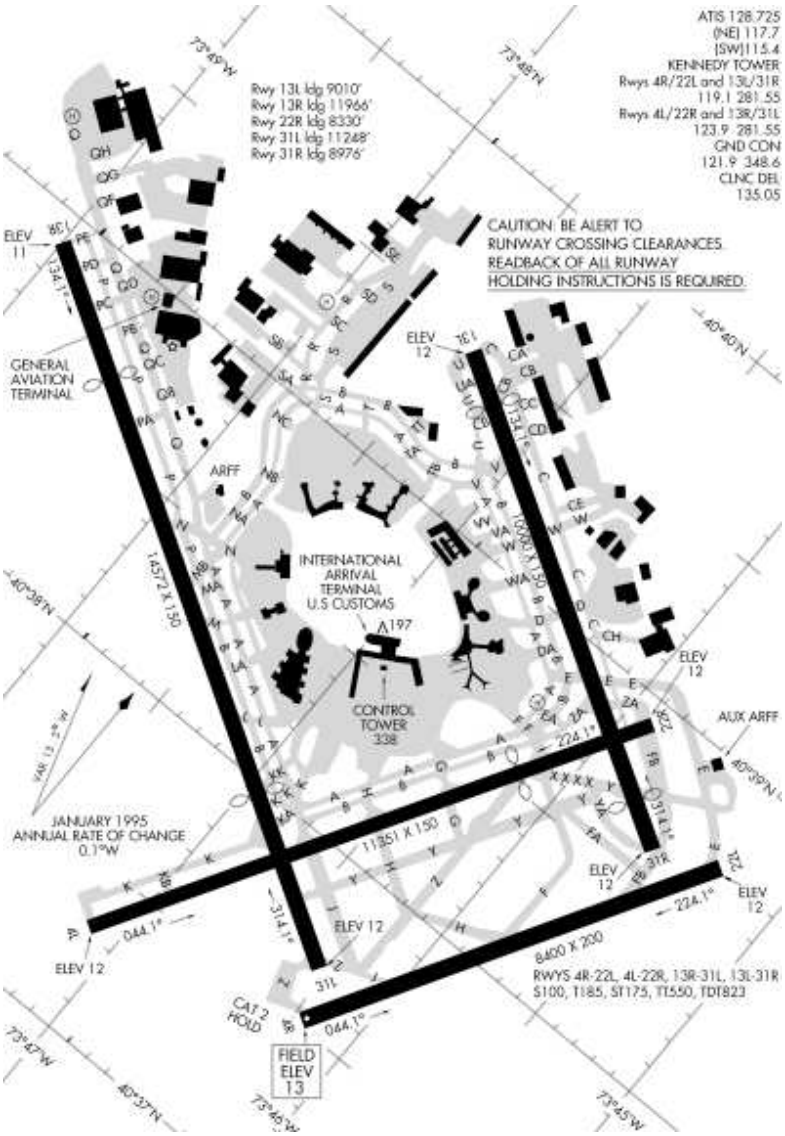

Figura 1-3

Le difficoltà maggiori nel gestire i casi in cui ci siano due o più piste intersecanti, come nel caso dell'aeroporto J.F.K. di New York riportato in Figura 1-3, sono nel gestire il coordinamento tra le varie parti, in modo da garantire la massima sicurezza. Il vantaggio deriva dal fatto che in caso di vento laterale ad una pista tanto da dover diminuire la sua capacità, si ha comunque la possibilità di completo impiego dell'altra e viceversa.

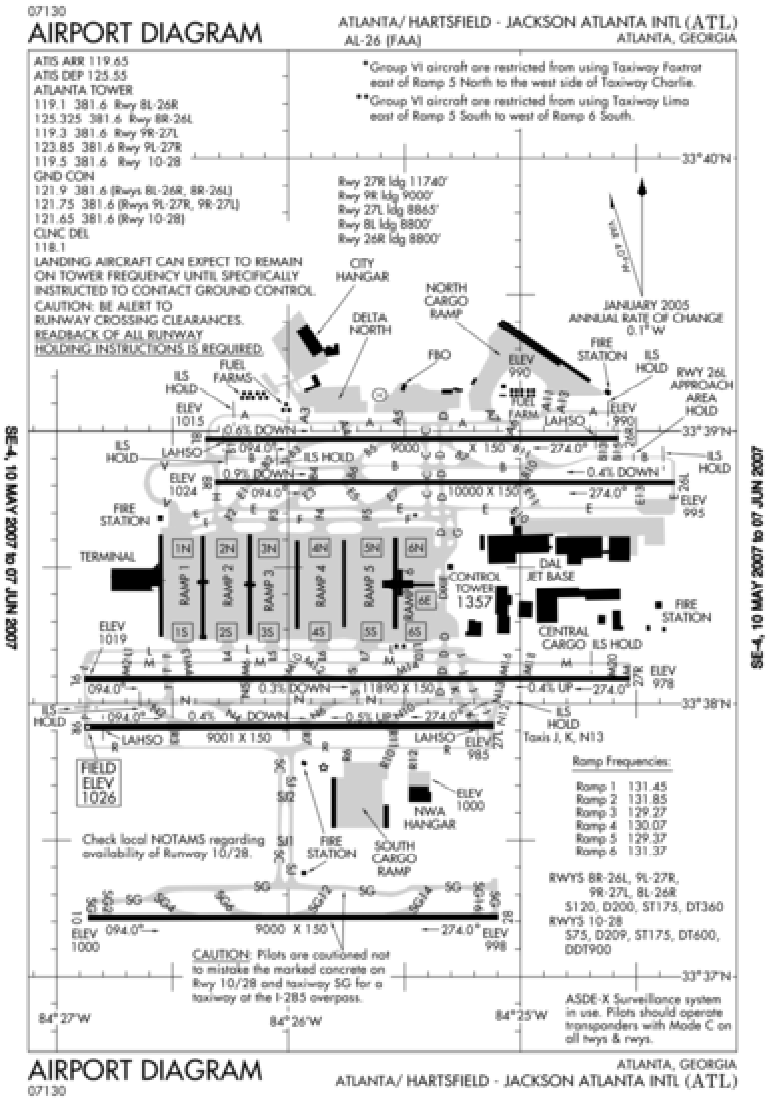

Figura 1-4

L'aeroporto di Atlanta, il cui layout è riportato in Figura 1-4, invece rappresenta uno di quei tipici scali americani in cui sono presenti 2 coppie di piste parallele a nord ed a sud dei terminal. È un esempio di una nuova generazione con una vasta capacità aerea ed un traffico di passeggeri di oltre 50 milioni di unità all'anno. La distanza tra le coppie di piste è sufficiente per operare indipendentemente.

Un'altro esempio di aeroscali con 4 o più piste è quello di Dallas-Fort Worth, il cui layout è riportato in Figura 1-5.

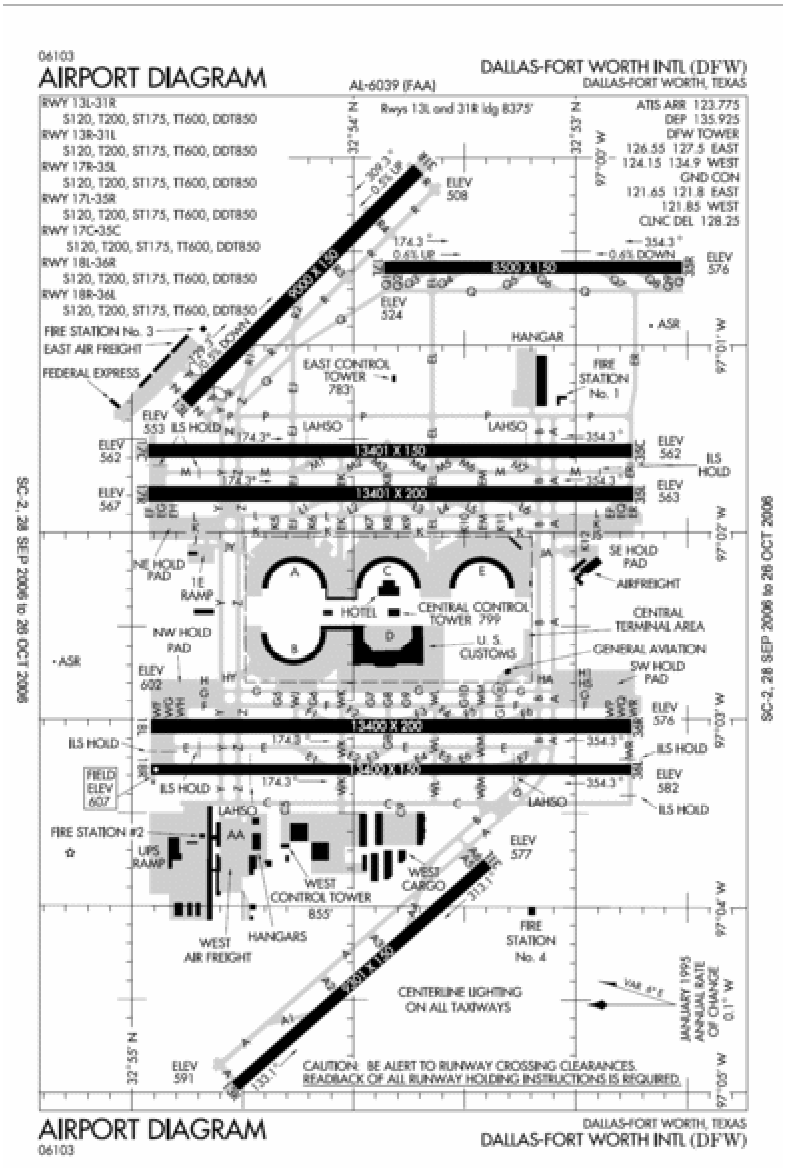

Figura 1-5

1.2: OPERAZIONI EFFETTUATE DAGLI AEROMOBILI

All'interno di ogni aeroscalo quando un aeromobile arriva effettua un percorso determinato, cioè compie una serie di azioni schematiche all'interno dell'aeroporto. Queste azioni si possono sintetizzare in un elenco:

- Arrivo, ossia il momento in cui l'aereo passa in carico al controllo a terra dell'aeroporto;

- Entrata in coda per atterrare ed atterraggio; in alternativa eventuale dirottamento verso un altro scalo;

- Entrata nella taxiway d'uscita dalla pista e direzionamento verso lo stand assegnato;

- Operazioni effettuate nello stand: sbarco dei passeggeri, scarico bagagli, servizi vari (pulizia, rifornimento, controllo funzionalità, etc..), imbarco passeggeri e carico bagagli. Nel caso l'aereo non riparta immediatamente, direzionamento verso i parcheggi o gli hangar;

- Entrata nella taxiway d'entrata in pista, ed in coda per partire;

- Partenza.

Ad ognuna di queste operazioni corrisponde un determinato tempo di esecuzione che sarà generato da una funzione di probabilità. Le famiglie probabilistiche e le motivazioni che portano alla loro scelta per la creazione del modello di simulazione saranno oggetto del seguente paragrafo.

1.3: FUNZIONI DI PROBABILITÀ E LORO APPLICAZIONI

La costruzione di un modello che rappresenti la realtà implica il fatto che dovranno esserci delle stime dei fenomeni da studiare. Difatti in fase concettuale si devono riconoscere quali siano le situazioni critiche nel sistema sotto studio, ossia identificare i fenomeni che abbiano una componente stocastica, cioè dovuti al caso. Chiaramente si dovranno analizzare i dati a disposizione ed individuare quale distribuzione probabilistica rappresenti meglio la situazione sotto osservazione per poter poi procedere alla generazione di numeri casuali che saranno in seguito utilizzati come parametri per la simulazione.

Per generare dei numeri casuali da delle qualsiasi variabili aleatorie bisogna ricavare l'inversa della funzione di ripartizione di tale famiglia aleatoria, la funzione quantile. Tale funzione data una determinata probabilità calcola il corrispondente il quantile corrispondente. Ad esempio nel caso più semplice della variabile Uniforme compresa nell'intervallo fra a e b, si procede nel seguente modo:

- Funzione di densità della v. c. Uniforme[a,b] (Figura 1-6):

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{for } x < a \text{ or } x > b, \end{cases}$$

Figura 1-6

- Funzione di ripartizione della v. c. Uniforme[a,b] (Figura 1-7):

$$F(x) = \begin{cases} 0 & \text{for } x < a \\ \frac{x-a}{b-a} & \text{for } a \leq x < b \\ 1 & \text{for } x \geq b \end{cases}$$

Figura 1-7

- Funzione inversa, o funzione quantile della v. c. Uniforme[a,b]:

$$\boldsymbol{x = F^{-1}(u) = a + (b - a) \cdot u}$$ dove *u* appartiene a IR[0,1]

Nel modello in studio saranno utilizzate solamente 2 funzioni aleatorie, entrambe appartenenti alla famiglia delle variabili continue:

- Triangolare(a,b,c) con *a = valore minimo,*

b = valore più probabile

c = valore massimo.

Funzione di densità (Figura 1-8):

$$f(x|a,b,c) = \begin{cases} \frac{2(x-a)}{(b-a)(c-a)} & \text{per } a \leq x \leq c \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{per } c < x \leq b \end{cases}$$

Figura 1-8

Posti $d = (b - a) / (c - a)$

$r = u$ dove u appartiene a IR[0,1]

la funzione quantile risulterà uguale a:

$$\boldsymbol{X = a + (c - a) \cdot \{ \text{RADQ}(r \cdot d)} \quad \textit{se } r < d$$

$$\boldsymbol{1 - \text{RADQ}[(1 - d) \cdot (1 - r)]} \quad \textit{se } r \geq d$$

dove RADQ rappresenta la radice quadrata.

- Normale (μ,σ^2) con μ = *media* e σ^2 = *varianza.*

Funzione di densità (Figura 1-9):

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad \text{con} \quad -\infty < x < \infty$$

Figura 1-9

Funzione di ripartizione (Figura 1-10):

$$F(x) = \int_{-\infty}^{x} f(u)du$$

Figura 1-10

Funzione quantile (Figura 1-11):

$$F^{-1}(u) = \inf \{x \mid F(x) = u, 0 < u < 1\}$$

Figura 1-11

Prendendo il caso in studio si riportano di seguito gli eventi non deterministici e la funzione di probabilità associata ad essi, con la motivazione che ha spinto a questa scelta:

- Arrivo: la comparsa dell'aeromobile sugli schermi radar dell'aeroporto è stato modellato con una distribuzione triangolare. Tale scelta è stata fatta in quanto descrive nel miglior modo la distribuzione degli arrivi, poiché si può dedicare un'area maggiore di probabilità ai ritardi, che sono in maggioranza rispetto agli anticipi, ma prediligere un valore modale più vicino all'orario prestabilito.

- Incertezza della posizione dell'aeromobile: si distribuisce come una funzione Normale di media 0 e deviazione standard espressa in miglia nautiche**.**

- Velocità dell'aeromobile: modellata come una normale con media e deviazione standard in nodi (miglia nautiche orarie).

- Separazioni tra 2 consecutivi aeromobili nel percorso finale: rappresenta l'effettivo tempo di interarrivo, cioè il tempo che separa due arrivi

consecutivi, rispettando le distanze minime imposte dai controllori di volo. E' modellato come una variabile aleatoria normale con media e scarto espressi in minuti.

- Occupazione della pista: espressa con una funzione normale con media e scarto quadratico medio valutati in minuti.

- Occupazione delle taxiway e degli stand: come nel precedente caso si modellano con una normale di media e deviazione standard in minuti.

- Ritardo tra comunicazione e partenza del velivolo: rappresenta il tempo che intercorre tra l'effettivo permesso per il decollo dato dal controllore di volo e l'inizio della manovra in pista.

La scelta delle rispettive variabili aleatorie è stata suggerita dalle direttive presenti sia sul manuale di utilizzo di MACAD che sull'articolo "*A decision support system for airport strategic planning*" riportato nella bibliografia. Solamente la decisione di utilizzare la funzione triangolare non è stata suggerita da alcun libro o manuale.

CAPITOLO 2: SIMULAZIONE DEL LATO AEREO DI UN AEROPORTO, NEI CASI DI UNA PISTA E DUE PISTE PARALLELE INDIPENDENTI

2.1: ASPETTI GENERALI

Per costruire il programma di simulazione si devono innanzitutto comprendere 2 aspetti: quali siano le entità e quali i serventi. Nella fattispecie di un aeroporto le entità da considerare saranno i singoli aeromobili, mentre le componenti serventi sono le pista e gli stand. Le code quindi andranno poste prima di ogni tipo di servente. Dopodiché bisogna costruire lo schema all'interno del quale si svolgono le operazioni che le singole entità devono eseguire; tale schema è gia stato descritto nel paragrafo 1.2. Infine si procede alla formulazione del modello vero e proprio, ponendo l'attenzione nell'individuare le componenti stocastiche che richiederanno la generazione di numeri casuali, e le componenti deterministiche (cfr. paragrafo 1.3,). Una volta stilato il modello si prosegue nell'implementazione computazionale e alla successiva validazione e analisi degli output. Nel caso di un aeroporto i dati di input possono essere molteplici: oltre agli aspetti tecnici dell'aeroporto stesso (numero di piste, numero di stand, lunghezza piste, etc..), i più interessanti riguardano la gestione dei tempi di arrivo delle singole entità. Difatti si possono generare gli arrivi sia in modo

deterministico, fornendo un orario schedulato a cui è possibile aggiungere dei valori casuali che rappresentano gli eventuali anticipi o ritardi (molto più preciso), oppure in modo stocastico utilizzando il processo degli arrivi di Poisson (utile per avere un'idea molto generale). La sostanziale differenza è evidenziata dal fatto che nel primo caso il tempo di interarrivo è il risultato di un calcolo basato su dati effettivi, nel secondo è fornito in input come parametro per la creazione delle entità.

2.2: SIMULAZIONE DI UN AEROPORTO CON UNA PISTA PER GLI ATTERRAGGI ED I DECOLLI

Un aeroporto con una singola pista servente sia gli arrivi che i decolli è il sistema prevalente nel territorio italiano. Solitamente la struttura è semplice ed è composta da una pista e da un numero di stand adeguato alla richiesta di movimenti. I casi di congestione del traffico aereo in entrata ed uscita inoltre sono casi limite, dettati solitamente dalle condizioni, dato il non elevato numero di voli giornaliero.

L'aspetto più importante è calcolare il tempo di interarrivo, cioè il tempo minimo che intercorre tra due movimenti consecutivi nel pieno rispetto delle distanze minime imposte dai controllori di volo. Per esplorare tale argomento prendiamo in esame 4 differenti situazioni di utilizzo della pista:

1) Solo arrivi.

2) Inserire una partenza tra due arrivi.

3) Alternare arrivi e partenze in modo sequenziale.

4) Solo partenze.

1) Assumiamo che la velocità nel percorrere il percorso finale sia stimata con una variabile casuale Normale, con medie e deviazione standard diverse per ogni tipo di aereo. Inoltre anche il tempo di occupazione della pista sia trattato con una Normale con medie e scarti quadratici medi in funzione del tipo. Il modello calcolerà μ, che rappresenta la distanza temporale tra due successivi arrivi all'inizio del percorso finale. Questo valore è calcolato in 2 passi: prima si computa il tempo che separa due arrivi successivi senza violare la distanza minima in aria durante la fase di atterraggio (μ_1); dopo si ricalcola il tempo tra 2 arrivi tenendo però conto del fatto che non possono occupare simultaneamente la pista (μ_2). Il valore più restrittivo risulterà μ=max(μ_1, μ_2). Ora ponendo V_i la velocità media in nodi, δVi la deviazione standard della velocità stessa, D la lunghezza del percorso finale, S_{ij} la distanza minima in miglia nautiche tra un aereo di i e uno di tipo j, Ra_i l'occupazione media della pista in minuti, δRa_i la deviazione standard per un aereomobile di i, e indicando con L il precedente e con F il successivo, per calcolare μ_1 *e* μ_2 bisogna dividere in due diverse situazioni: il caso in cui la velo-

cità del seguente è maggiore di quella del leader ed il caso opposto nel quale la velocità del seguente è minore del precedente.

- $V_S \geq V_L$:

  $\mu_1 \geq (D/V_L) - [(D-S_{FL})/V_F] + [(1.65 \cdot \sigma_1)/V_F]$

  $\sigma_1^2 = [(D^2 \cdot V_F^2)/V_L^2]\,[(\sigma_{VF}^2/V_F^2) + (\sigma_{VL}^2/V_L^2)] + \mu_1^2 \cdot V_F^2 \cdot (\sigma_{VF}^2/V_F^2)$

  e σ_{VL}^2 e σ_{VF}^2 sono le varianze delle velocità dei due aerei.

  $\mu_2 \geq (D/V_L) - (D/V_F) + Ra_L + (2.215 \cdot \sigma_2)$

  $\sigma_2^2 = \sigma_{RaL}^2 + [(D^2/V_F^2) \cdot (\sigma_{VF}^2)/V_F^2)] + [(D^2/V_L^2) + (\sigma_{VL}^2/V_L^2)]$

  e σ_{RaL}^2 la varianza dell'occupazione della pista dell'aereo leader.

- $V_S < V_L$:

  $\mu_1 \geq (S_{FL}/V_L) + [(1.65 \cdot \sigma_1)/V_L]$

  $\sigma_1^2 = \mu_1^2 \cdot \sigma_{VL}^2$ e σ_{VL}^2 la varianza delle velocità del leader.

  $\mu_2 \geq (D/V_L) - (D/V_F) + Ra_L + (2.215 \cdot \sigma_2)$

  $\sigma_2^2 = \sigma_{RaL}^2 + [(D^2/V_F^2) \cdot (\sigma_{VF}^2)/V_F^2)] + [(D^2/V_L^2) + (\sigma_{VL}^2/V_L^2)]$

  e σ_{RaL}^2 la varianza dell'occupazione della pista dell'aereo leader.

Calcolato quindi $\mu = \max(\mu_1, \mu_2)$, il tempo che trascorre tra due arrivi consecutivi è dato dalla seguente equazione:

$$IAT_{FL} = [D/(V_F + \delta V_F)] - [D/(V_L + \delta V_L)] + \mu$$

e si distribuisce come una variabile casuale Normale con parametri:

$$E(IAT_{ij}) = (D/V_F) - (D/V_L) + \mu$$

$$\sigma_{IATij}^2 = [(D^2/V_F^2)\cdot(\sigma_{VF}^2)/V_F^2)] + [(D^2/V_L^2)\cdot(\sigma_{VL}^2/V_L^2)]$$

2) L'inserimento di una partenza tra due arrivi consecutivi è possibile nel solo caso in cui la distanza lo permetta e senza costringere ad aumentare tale distanza. Assumiamo che il tempo di occupazione della pista per un aereo in partenza sia modellato con una variabile casuale normale di media Rd_i e deviazione standard δRd_i. Inoltre sia G_{ij} le distanze minime tra ogni coppia di tipi di aerei in partenza espresse in minuti e c il ritardo di comunicazione stimato con una Normale con media $E(c)$ e con scarto quadratico medio σ_c. La probabilità di inserire una partenza di tipo k tra un arrivo di tipo i (seguente) e j (leader) è:

$$Pd_{ikj} = 1- C[0, E(IAT_{ij}) - E(Raj) - E(c) - \max[E(Rd_k),E(TFA_i)],$$
$$RADQ(\sigma_{IATij}^2 + \sigma_{Raj}^2 + \sigma_c^2 + \sigma_x^2)]$$

e con σ_x^2 che è ugualea σ_{Rdk}^2 o σ_{TFAi}^2 dipende dal vincolo costrittivo

dove TFA_i è il tempo che un aereo in arrivo di tipo i impiega a raggiungere la soglia della pista quando si trova alla distanza minima per concedere una partenza,

e $C(x,\mu,\sigma)$è la funzione di ripartizione della Normale. *TFA* si distribuisce come una variabile aleatoria Normale e si calcola nel seguente modo:

$$TFA_F = [SD/(V_F + \delta V_F)]$$

$$E(TFA) = SD/V_F$$

$$\sigma_{TFA}^2 = [(SD^2 \cdot \sigma_{VF}^2)/ V_F^4]$$

3) A differenza del punto precedente, alternare atterraggi e decolli in modo sequenziale potrebbe portare ad un allargamento del tempo di interarrivo quando è necessario per concedere una partenza rispettando le distanze minime imposte. La computazione è uguale al punto uno, salvo il calcolo di μ_2 che sarà invece la somma del tempo di occupazione della pista dell'aereo leader, dei possibili ritardi di comunicazione e del valore più restrittivo tra il tempo di occupazione della pista dell'aereo in partenza e il tempo che un aereo in arrivo impiega a raggiungere la soglia della pista quando si trova alla distanza minima per concedere una partenza:

$$\mu_2 \geq (D/V_L) - (D/V_F) + [Ra_L + E(c) + \max(Rd_d, E(TFA_d)] + (2.215 \cdot \sigma_2)$$

$$\sigma_2^2 = (\sigma_{RaL}^2 + \sigma_c^2 + \sigma_x^2) + [(D^2/V_F^2) \cdot (\sigma_{VF}^2)/V_F^2)] + [(D^2/V_L^2) + (\sigma_{VL}^2/V_L^2)]$$

e con σ_x^2 che è ugualea σ_{Rdd}^2 o σ_{TFAd}^2 dipende dal vincolo costrittivo

4) Il caso in cui la pista serva solo partenze è il più semplice in quanto si deve tener conto solamente delle distanze minime imposte tra due consecutive partenze di aerei di tipo *i* e *j* e il ritardo di comunicazione tra torre di controllo e

pilota, entrambi espressi in minuti. Quindi assumendo che i ritardi *c* siano stimati con una Normale di media *E(c)* e di scarto quadratico medio σ_c i tempi tra due consecutive partenze saranno dati dalla loro somma: $G_{ij} + c$.

2.3: SIMULAZIONE NEL CASO DI UN AEROPORTO CON 2 PISTE PARALLELE

Nel caso di due piste parellele operanti si deve distinguere in due casi: se possono operare indipendentemente o meno. Inoltre esistono diverse configurazioni di utilizzo: possono ospitare gli arrivi in una e le partenze nell'altra, quindi operando separate, oppure accettare entrambe arrivi o partenze simultaneamente.

Il fatto che si possano giudicare indipendenti o meno deriva dalla distanza che intercorre tra il centro delle due piste. Se tale parametro supera il vincolo fissato per operare indipendentemente allora si comportano come fossero due piste singole. Se invece tale parametro non soddisfa il vincolo allora operano come fossero una pista singola. Solitamente si riducono a due casi: o la distanza è tale da permettere la totale indipendenza, oppure è in misura da consentire di operare autonomamente tranne per il caso di arrivi contemporanei. Un esempio di quanto spiegato è mostrato nella Tabella 2-1. Si riferisce alle separazioni richieste tra i movimenti di aeromobili sulle due piste parallele negli Stati Uniti dalla F.A.A. (Federal Aviation Administration).

Distanza tra il centro delle due piste	***Arrivo/ Arrivo***	***Partenza/ Partenza***	***Arrivo/ Partenza***	***Partenza/ Arrivo***
Fino a 760 m	Come una pista singola	Come una pista singola	L'aereo è atterrato	L'aereo è decollato
760 m – 1300 m	1,5 miglia nautiche	Indipendenti	Indipendenti	Indipendenti
Più il 1300 m	Indipendenti	Indipendenti	Indipendenti	Indipendenti

Tabella 2-1 (fonte: "*Airport Systems*" R. de Neufville, A. Odoni)

Il caso in cui entrambe le piste sono utilizzate per un arrivo e distano tra loro tra 760 m e 1300 m, deve esserci una separazione tra i due aerei di almeno 1,5 miglia nautiche misurata diagonalmente e rappresenta la distanza diretta tra i due aeromobili.

Inoltre si possono considerare ulteriori due punti. Il primo indica che quando entrambe le piste sono utilizzate per una partenza, i movimenti sono indipendenti solo se gli aerei che decollano simultaneamente seguano percorsi divergenti per arrivare in quota. Nel caso in cui non lo siano si utilizzano le separazioni come nel caso di pista singola. Il secondo punto invece riguarda l'allineamento delle della soglia delle piste. La Tabella 2-1 assume come presupposto che l'inizio di entrambe le piste sia allineato. Se invece è sfalsato, si crea una distanza tra l'inizio di una e la proiezione dell'inizio dell'altra, chiamata O*ffset.* Nello specifico lo spazio tra i centri delle piste si riducono di 30 m ogni 150 m di *Offset* sino ad un minimo di 360 m. Ad esempio quando è presente un *Offset* di 300 m, una distanza tra i centri delle piste di 700 m equivale ad una di 760 m quando un *Offset* non è presente.

CAPITOLO 3: IMPLEMENTAZIONE DI UN PACCHETTO SOFTWARE: IL CASO DELL'AEROPORTO INTERNATIONALE *"ELEFTHÉRIOS VENIZÉLOS"* DI ATENE

L'aeroporto internazionale di Atene è di recente costruzione in quanto è entrato in servizio il 29 marzo del 2001. È l'unico aeroscalo civile della città, in quanto ha rimpiazzato il vecchio aeroporto internazionale *"Ellinikon"*, ed inoltre è il maggiore hub della compagnia statale di bandiera ellenica Olympic Airlines ed è base anche per Aegean Airlines. In figura 4-1 è riportato il layout dell'aeroscalo.

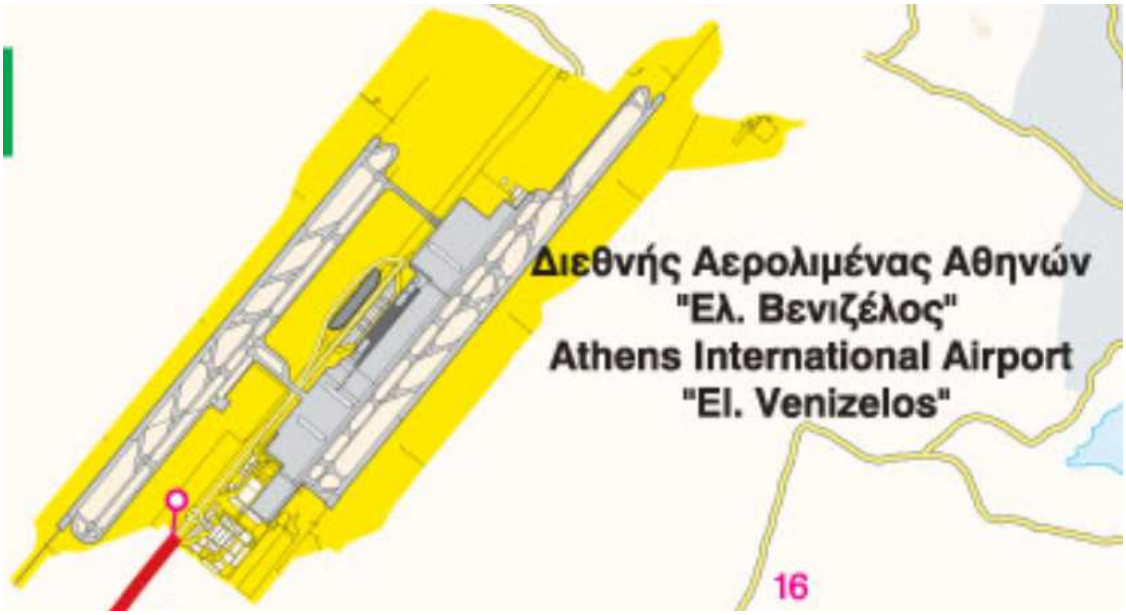

Figura 3-1 (fonte: *www.aia.gr*)

3.1: INFORMAZIONI GENERALI

L'aeroporto è situato tra le cittadine di Markopoulo, Koropi, Spata e Loutsa a circa 20 km ad est in linea d'aria dal centro di Atene (30 km in strada). È dedicato a *Elefthérios Venizélos*, importante politico di origine cretese che fu Primo Ministro della Grecia e diede un sostanziale contributo alla ribellione dell'isola di Creta contro l'occupazione Ottomana nel 1896.

Attualmente l'aeroscalo ha due terminal, uno principale ed uno satellite, accessibile da quello principale grazie ad una galleria percorribile a piedi, e due piste parallele della lunghezza di circa 4 km con orientamento magnetico 03-21. È stato costruito grazie ad una collaborazione pubblico-privata; lo Stato Greco detiene circa il 55% delle azioni, mentre le restanti sono divise tra privati. È stato inoltre premiato con *"European Airport of the Year 2004"* (Aeroporto Europeo dell'anno 2004), dato da *Institute of Trasport Managemente (ITM*) per gli innovativi schemi imprenditoriali e il successo delle operazioni e dei risultai, e con il premio *"Skytrax"* per il miglior aeroporto dell'Europa meridionale negli anni 2005 e 2006.

L'aeroporto è stato progettato per poter subire delle espansioni future in modo da accomodare il futuro incremento del traffico aereo. Il piano di crescita è stato diviso in sei fasi: la prima, nonché la corrente, permette di accogliere un transito di quasi 16 milioni di passeggeri all'anno; l'ultima, ossia la sesta,

permetterà di accoglierne più di 50 milioni all'anno.

Per una maggiore sicurezza di quanti operano o transitano al proprio interno è stato equipaggiato con due sistemi robotici, *"Hercules"* e *"Ulysses"*, in grado di maneggiare dispositivi sospetti, e di identificare e rimuovere senza pericoli possibili esplosivi. Il primo è stato donato dalla Fondazione *Stavros Niarchos*; è un robot capace di raccogliere e trasportare dei possibili ordigni per il disinnesco. Ha una forma sferica del diametro di 120 cm, è corazzato ed è dotato di due braccia robotiche. Il secondo invece ha un costo di circa 94.000 € ed è stato donato da *Soukos Robots ABEE*. È stato costruito col fine di servire da integrazione di *Hercules*, permettendogli l'ingresso in luoghi più impervi quali bagni, autobus o aerei. *Ulysses* è un robot leggero ma molto efficiente, munito di un sistema di assorbimento degli urti, che consente anche il movimento su superfici irregolari.

L'aeroporto ha avuto, nel 2006, un aumento di traffico di passeggeri del 5,6% rispetto al 2005 arrivando a 15.079.662. I grafici in Figura 4-2 mostrano la distinzione della crescita per tipologia di volo, nazionale o internazionale, e mensilmente. Si può notare come solo verso la fine dell'anno si sia manifestato lo sviluppo che ha alzato la media annuale, mentre nei primi tre mesi solamente i voli nazionali erano in effettivo aumento.

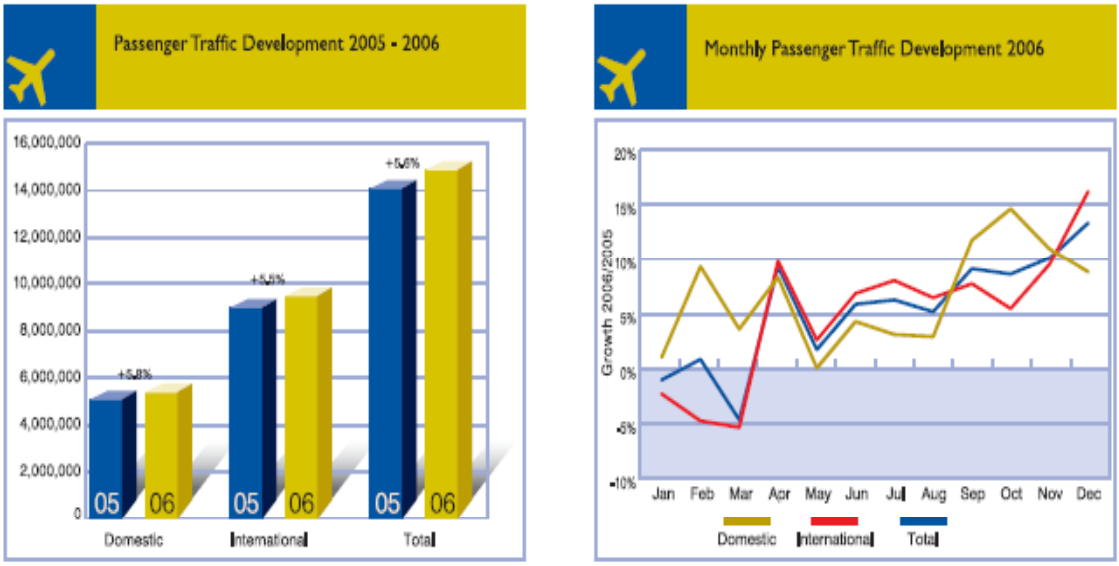

Figura 3-2 (fonte: *www.aia.gr*)

Nei grafici in Figura 4-3 si evidenzia invece lo sviluppo del traffico aereo, evidenziando l'aumento di movimenti degli aeromobili. La crescita è dell'ordine del 5,5% rispetto al 2005 arrivando ad un totale di 191.000 movimenti all'anno.

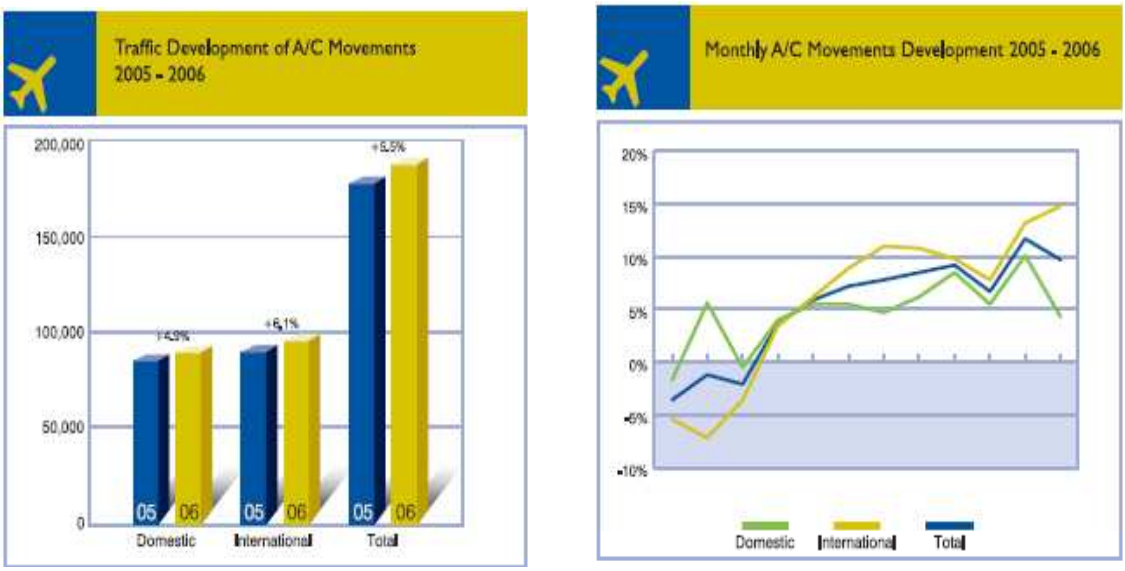

Figura 3-3 (fonte: *www.aia.gr*)

Nell'aeroporto sono attive 60 compagnie che gestiscono i voli di linea, 78 che organizzano voli charter e 12 che operano con voli cargo. Le destinazioni attive sono in totale 138, suddivise in: 128 charter, 33 nazionali, 77 internazionali, divise tra 28 in Europa Occidentale, 20 in Europa Orientale, 4 in Nord America, 9 in Medio Oriente, 4 in Asia Centrale, 2 nel Sud-Est Asiatico, 3 in Africa Settentrionale e 1 in Africa Meridionale.

I grafici Figura 4-4 riportano la percentuale di aerei con un ritardo superiore ai 15 minuti e il ritardo medio per volo sempre espresso in minuti; entrambi i grafici sono divisi per arrivi e partenze e in anni. Si nota subito come sia maggiore la percentuale di ritardi in partenza, 32,2%, rispetto a 24% dei ritardi in arrivo. Significa che almeno un aereo su tre decolla dopo almeno 15 minuti dall'orario schedulato e uno su quattro atterra con 15 minuti di ritardo. Inoltre dal 2005 al 2006 non si sono verificate variazioni degne di attenzione. Invece si può osservare come i minuti medi di ritardo per aereo siano diminuiti da un anno all'altro: per gli arrivi si è passati da una media di 48 a 42 e per le partenze da 42 a 38. In pratica c'è una porzione maggiore di aerei in ritardo in fase di partenza ma che accumulano un minor minutaggio medio rispetto alla minore frequenza di aerei in arrivo che hanno però un maggior dilazione media. Questo fatto si nota meglio nei grafici Figura 4-5. La frequenza di voli che hanno un ritardo superiore ai 15 minuti è, per tutto il corso dell'anno, maggiore per le partenze. A fianco invece si osserva come la situazione si capovolga: gli arrivi accumulano un minutaggio medio di ritardo maggiore rispetto ai decolli.

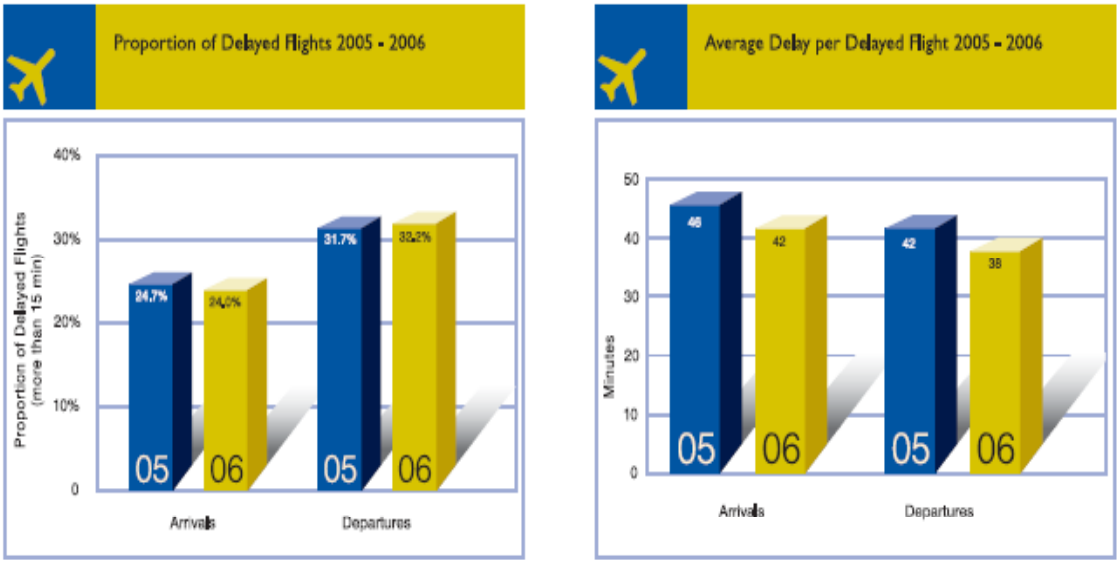

Figura 3-4 (fonte: *www.aia.gr*)

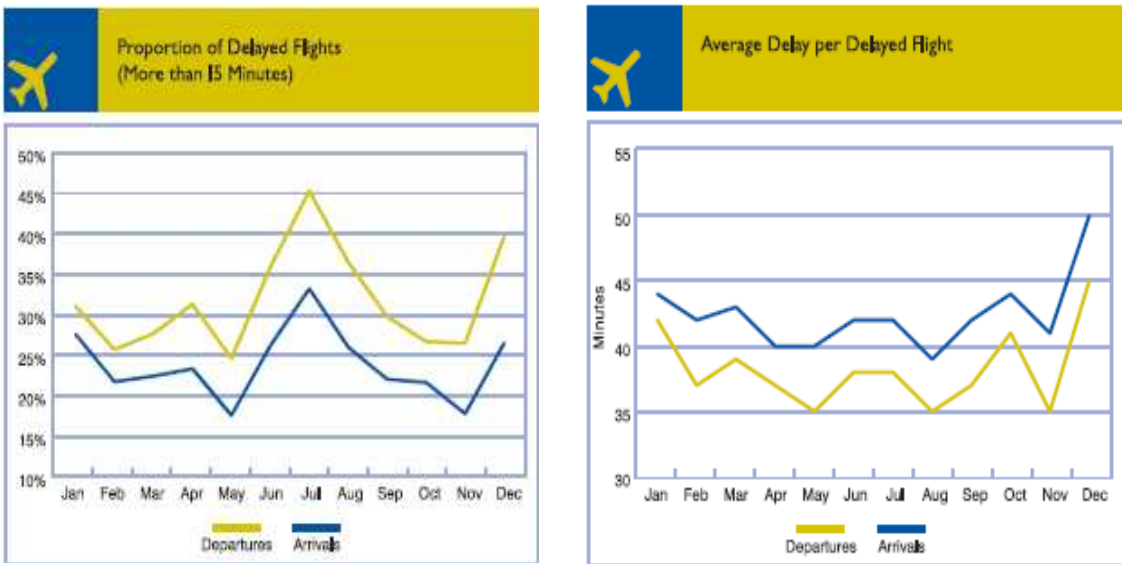

Figura 3-5 (fonte: *www.aia.gr*)

3.2: ANALISI DEI DATI

I dati su cui si basa la successiva simulazione sono stati gentilmente forniti dal Dipartimento di Matematica Pura ed Applicata dell'Università di Padova. Si tratta dello stesso input che viene fornito al programma MACAD e consta di diversi file di testo rappresentanti le diverse informazioni tecniche su aerei ed aeroporto e di un file di testo con un orario giornaliero schedulato. Tra i dati a disposizione si hanno parametri stocastici quali i tempi medi di occupazione di pista e stand con le relative deviazioni standard, i tempi di interarrivo, le velocità medie e gli scarti quadratici medi, l'incertezza della posizione, il ritardo di comunicazione, il tempo medio di preparazione dello stand, oppure deterministici come la lunghezza del percorso finale, il numero di stand o le distanze minime tra ogni coppia di aerei. Inoltre è già presente il tempo di interarrivo nel caso di una pista dedicata solo agli arrivi. Chiaramente sono tutti dati che si riferiscono all'aeroporto di Atene.

Gli aeromobili sono suddivisi in 5 categorie, riportate sotto insieme alla loro frequenza all'interno dei voli della giornata:

Tipo Aereo	**Numero Voli**	**Frequenza Relativa**
Small	17	0,0637
Medium	68	0,2547
Large	110	0,4120

Wide	49	0,1835
Jumbo	23	0,0861

Tabella 3- 1

Inoltre anche gli stand sono suddivisi in funzione dei tipi di aereo che possono servire, come mostrato in Tabella 3-2. Chiaramente uno stand che può servire un aeromobile di tipo Wide, può ospitare anche i tipi più piccoli.

Tipo Aereo	**Numero di stand**
Small	13
Medium	40
Large	12
Wide	8
Jumbo	36

Tabella 3- 2

3.3: MODELLO UTILIZZATO

L’aeroporto di Atene è dotato di due piste parallele, distanti tra loro abbastanza da poter operare indipendentemente. Il modello utilizzato sarà lo stesso descritto nel paragrafo 2.3, con una pista dedicata agli arrivi e l’altra dedicata alle partenze. Per gli stand e le taxiway invece è utilizzato un modello a

tempi discreti. Si computa in 4 passi, i primi 2 si eseguono prima dell'avanzare del clock eventi:

1. Si aggiunge il tempo di taxiway a tutti gli aerei.
2. Si allocano gli stand agli aerei che erano parcheggiati dal giorno prima (overnight).

I seguenti 2 passi invece permettono l'avanzare del timer degli eventi:

3. Arrivo in coda di un aereo: se esiste uno stand libero lo alloca, se non esiste lo inserisci in coda.
4. Si libera uno stand. Se è presente un aereo in coda che può utilizzare tale parcheggio lo assegna allo stand.

3.4: DESCRIZIONE DEL PROGRAMMA

L'implementazione del pacchetto software è stata articolata in 6 moduli: un modulo principale, in cui è presente il corpo del programma; uno per gestire le operazioni degli aerei, uno per gestire le azioni dell'aeroporto; uno per gestire la coda; uno dove sono implementate le funzioni di probabilità e infine uno per

gestire la lista. Il lavoro è stato svolto con l'ausilio del linguaggio di programmazione *C++* ed in particolare con la metodologia *Object Oriented*.

Di seguito sono esaminati i moduli singolarmente:

3.4.1: IMPLEMENTAZIONE DELLA LISTA

La lista rappresenta una delle strutture dati più usate nelle applicazioni. Assume la veste di una sequenza ordinata di dati di dimensione indefinita in grado di crescere e diminuire a seconda delle esigenze espresse in fase di applicazione. Nella fattispecie sarà utilizzato un vettore di liste, dove ogni posizione di tale vettore rappresenta un momento temporale del timer degli eventi. Una possibile implementazione di una lista realizzata mediante la concatenazione di nodi è la seguente:

```
// lista.h

typedef int T_eleml;


struct Nodo
{
   T_eleml elem;
   Nodo *succ;
};

typedef Nodo* Posizione;

class Lista
{
   Posizione inizio;
   Posizione ultimo;
public:
```

```
   Lista();       //costruttore
   ~Lista();      //distruttore
   void Init();
   int Vuota()
   {
      return (inizio->succ==NULL);
   }
   Posizione Primo();
   Posizione Fine();
   Posizione Successiva(Posizione);
   Posizione Precedente(Posizione);
   void Cancella(Posizione);
   void Inserisci(const T_eleml &, Posizione);
   T_eleml& Valore(Posizione);
};
```

```
// lista.cpp

Lista::Lista()
{
   inizio=new Nodo;
   inizio->succ=NULL;
   ultimo=inizio;
};
Lista::~Lista()
{
   Posizione psuc;
   for(Posizione p=inizio;p!=NULL;p=psuc)
   {
      psuc=p->succ;
      delete p;
   }
};

void Lista::Init()
{
   inizio=new Nodo;
   inizio->succ=NULL;
   ultimo=inizio;
};
Posizione Lista::Primo()
{
   return inizio;
};
Posizione Lista::Fine()
{
   return ultimo;
};
Posizione Lista::Successiva(Posizione p)
```

```
{
   if (p->succ!=NULL)
      return (p->succ);
   else
   {
      cout << "Errore, il successivo non esiste" << endl;
      exit(NULL);
   }
};
Posizione Lista::Precedente(Posizione p)
{
   Posizione pos;
   for (pos=inizio;(pos->succ!=p) && (pos->succ!=NULL); pos=pos->succ);
   if (pos->succ!=NULL)
      return (pos);
   else
   {
      cout << "Errore, il precedente non esiste" << endl;
      exit(NULL);
   };
};
void Lista::Cancella(Posizione p)
{
   Posizione t=p->succ;
   if (t==ultimo)
      ultimo=p;
   p->succ=p->succ->succ;
   delete t;
   return;
};
void Lista::Inserisci(const T_eleml& el, Posizione p)
{
   Posizione t=new Nodo;
   t->elem=el;
   t->succ=p->succ;
   p->succ=t;
   if (p==ultimo)
      ultimo=t;
   return;
};
T_eleml& Lista::Valore(Posizione p)
{
   return p->succ->elem;
};
```

3.4.2: IMPLEMENTAZIONE DELLA CODA

Anche la coda rappresenta una delle strutture dati più utilizzata nelle applicazioni. L'ordine di inserimento e di recupero dei dati segue il principio F.I.F.O. (First In First Out), ossia il principio "democratico" in cui gli elementi recuperati e/o cancellati rispettano l'ordine con cui sono stati inseriti. In una coda sono considerate ambedue le estremità: una viene usata nell'inserimento, l'altra per il recupero o la cancellazione dei dati nella struttura, pertanto a differenza della lista non esiste il concetto di posizione (un elemento o è in testa o in coda). Nella fattispecie è stata implementata una coda a struttura circolare:

```
// coda.h

typedef int T_elemc;



class Coda
{
   T_elemc *elem;
   int dim;
   int fronte, retro, n_elem;
public:
   Coda(int maxdim=20);
   ~Coda();
   int Vuota();
   int Piena();
   int Cancella();
   int Inserisci(const T_elemc&);
   T_elemc& Valore();
};
```

```
// coda.cpp

Coda::Coda(int maxdim)
```

```
{
   dim=maxdim;
   fronte=0;
   retro=0;
   n_elem=0;
   elem=new T_elemc[dim];
};
Coda::~Coda()
{
   delete [] elem;
};
int Coda::Vuota()
{
   return(n_elem==0);
};
int Coda::Piena()
{
   return(n_elem==dim);
};
int Coda::Cancella()
{
   if (Vuota())
   {
      return 1;
   }
   fronte=++fronte % dim;
   n_elem-=1;
   return 0;
};
int Coda::Inserisci(const T_elemc& el)
{
   if (Piena())
   {
      return 1;
   }
   elem[retro]=el;
   retro=++retro % dim;
   n_elem+=1;
   return 0;
};
T_elemc& Coda::Valore()
{
   if (Vuota())
   {
      exit(1);
   }
   return elem[fronte];
};
```

3.4.3: IMPLEMENTAZIONE DELLE FUNZIONI PROBABILISTICHE

Come già visto nel paragrafo 1.3 le uniche due funzioni quantile implementate sono state della distribuzione Triangolare e della distribuzione Normale.

```
// probabilita.h

float triang(float, float, float);
float inv_norm(float, float);
```

```
// probabilita.cpp

float RandG(float, float);

float triang(float a, float b, float c)
{
   float d,p;
   Randomize();
   d=(b-a)/(c-a);
   p=rand();
   return (a+(c-a)) * ((p<d) ? sqrt(p*d) : (1-sqrt((1-d)*(1-p))));
};

float inv_norm(float mean, float stdev)
{
   Randomize();

   return RandG(mean, stdev);
};
```

RandG è una funzione di C++ che dati media e deviazione standard ritorna il quantile corrispondente ad una probabilità che gestisce casualmente al suo interno.

3.4.4: IMPLEMENTAZIONE DELLE FUNZIONI DELL'AEROPORTO

La classe aeroporto contiene tutti i parametri per gestire le funzioni degli aeromobili al proprio interno. Le funzioni implementate ritornano i tempi di esecuzione dei serventi (pista, taxiway e stand) e memorizzano se e quanti ne vengono occupati. Per gli stand si è adottata la politica di occupazione del tipo corrispondente o di tipo maggiore.

```
// Aeroporto.h
class Aeroporto
{
   int pista[2][2];           //piste serventi
   float IAT_media[5][5];      //minuti di tempo di interarrivo in colonna i successivi e in
riga i precedenti
   float IAT_stdev[5][5];
   float min_dist_p[5][5];     //minuti in colonna i successivi e in riga i precedenti
   float vel_media[5];         //nodi
   float vel_stdev[5];         //nodi
   float lung_perc_fin;        //miglia nautiche
   float occpista_media[5][2]; //minuti
   float occpista_stdev[5][2]; //minuti
   float rit_comun_stdev;      //minuti
   float err_pos;              //miglia nautiche
   float taxi_media[2];        //minuti
   float taxi_stdev[2];        //minuti
   int stand[5][2];            //stand totali per tipo di aereo e di volo
   int standserv[5][2];        //stand serventi per tipo di aereo e di volo
   float occstand_media[5];    //minuti
   float occstand_stdev[5];    //minuti
   float prepstand_media;      //minuti
   float prepstand_stdev;      //minuti


public:

   Aeroporto();

   int Pista_libera(int);  //PISTA
   void Occupa_pista(int, int);
   void Libera_pista(int);
   int Tipo_occupa(int);
```

```
   float Velmedia(int);
   float Velstdev(int);
   float Errorepos();
   float Lungpercfin();
   float Occpistam(int,int);
   float Occpistasd(int,int);
   float Ritcomun();

   long int Dist_mina(int, int); //DISTANZE MINIME
   long int Dist_minp(int, int);


   float Occtaxim(int);  //TAXIWAY
   float Occtaxisd(int);


   int Stand_libero(int, int);    //STAND
   int Occupa_stand(int, int);
   void Libera_stand(int, int);
   float Occstandm(int);
   float Occstandsd(int);
   int Tempo_prepstand();
};
```

```
// Aeroporto.cpp

Aeroporto::Aeroporto()
{
};


int Aeroporto::Pista_libera(int pos)  //PISTA
{
   return (pista[pos][0]==0);
};
void Aeroporto::Occupa_pista(int pos, int tipo)
{
   pista[pos][0]=1;
   pista[pos][1]=tipo;
   return;
};
void Aeroporto::Libera_pista(int pos)
{
   pista[pos][0]=0;
   pista[pos][1]=-1;
   return;
};
int Aeroporto::Tipo_occupa(int pos)
{
```

```
   return (pista[pos][1]);
};
float Aeroporto::Velmedia(int tipoa)
{
   return (vel_media[tipoa]);
};
float Aeroporto::Velstdev(int tipoa)
{
   return (vel_stdev[tipoa]);
};
float Aeroporto::Errorepos()
{
   return err_pos;
};
float Aeroporto::Lungpercfin()
{
   return lung_perc_fin;
};
float Aeroporto::Occpistam(int tipoa, int pos)
{
   return occpista_media[tipoa][pos];
};
float Aeroporto::Occpistasd(int tipoa ,int pos)
{
   return occpista_stdev[tipoa][pos];
};
float Aeroporto::Ritcomun()
{
   return rit_comun_stdev;
};


long int Aeroporto::Dist_mina(int tipop, int tipos)  //DISTANZE MINIME
{
   return INT(inv_norm(IAT_media[tipos][tipop],IAT_stdev[tipos][tipop])*60,0);
};
long int Aeroporto::Dist_minp(int tipop, int tipos)
{
   return INT(min_dist_p[tipos][tipop]*60,0);
};

float Aeroporto::Occtaxim(int pos)  //TAXIWAY
{
   return taxi_media[pos];
};
float Aeroporto::Occtaxisd(int pos)
{
   return taxi_stdev[pos];
};
```

```
int Aeroporto::Stand_libero(int tipoa, int tipov)    //STAND
{
   int lib=0,t;
   t=tipoa;
   switch (t)
   {
      case 1:  if (standserv[t][tipov]<stand[t][tipov])
               {
                 lib=1;
                 break;
               }
               else
                 t+=1;
      case 2:  if (standserv[t][tipov]<stand[t][tipov])
               {
                 lib=1;
                 break;
               }
               else
                 t+=1;
      case 3:  if (standserv[t][tipov]<stand[t][tipov])
               {
                 lib=1;
                 break;
               }
               else
                 t+=1;
      case 4:  if (standserv[t][tipov]<stand[t][tipov])
               {
                 lib=1;
                 break;
               }
               else
                 t+=1;
      case 5:  if (standserv[t][tipov]<stand[t][tipov])
               {
                 lib=1;
                 break;
               }
               else
                 t+=1;
   };
   return lib;
};
int Aeroporto::Occupa_stand(int tipoa, int tipov)
{
   int tipo, t=tipoa;
   switch (t)
   {
      case 1:  if (standserv[t][tipov]<stand[t][tipov])
```

```
            {
              standserv[t][tipov]+=1;
              tipo=t;
              break;
            }
            else
              t+=1;
      case 2:  if (standserv[t][tipov]<stand[t][tipov])
            {
              standserv[t][tipov]+=1;
              tipo=t;
              break;
            }
            else
              t+=1;
      case 3:  if (standserv[t][tipov]<stand[t][tipov])
            {
              standserv[t][tipov]+=1;
              tipo=t;
              break;
            }
            else
              t+=1;
      case 4:  if (standserv[t][tipov]<stand[t][tipov])
            {
              standserv[t][tipov]+=1;
              tipo=t;
              break;
            }
            else
              t+=1;
      case 5:  if (standserv[t][tipov]<stand[t][tipov])
            {
              standserv[t][tipov]+=1;
              tipo=t;
              break;
            }
            else
              t+=1;
   };
   return tipo;
};
void Aeroporto::Libera_stand(int tipoa, int tipov)
{
   standserv[tipoa][tipov]-=1;
   return;
};
float Aeroporto::Occstandm(int tipoa)
{
   return occstand_media[tipoa];
```

```
};
float Aeroporto::Occstandsd(int tipoa)
{
   return occstand_stdev[tipoa];
};
int Aeroporto::Tempo_prepstand()
{
   float tempo;
   tempo=INT(inv_norm(prepstand_media,prepstand_stdev)*60,0);
   return tempo;
};
```

3.4.5: IMPLEMENTAZIONE DELLE FUNZIONI DELL'AEREO

La classe Aereo contiene tutte le informazioni riguardanti le singole entità, quali il numero volo, il tipo di aereo, l'orario di arrivo e partenza previsto ed effettivo, il tipo di stand occupato; contiene inoltre le funzioni che effettuano la generazione dei numeri casuali dalle funzioni di probabilità per calcolare i tempi di utilizzo delle varie componenti dello schema del modello.

```
// Aereo.h

class Aereo
{
   long int time;
   long int ora_preva;
   long int ora_effa;
   long int ora_prevp;
   long int ora_effp;
   int tipo_aereo;
   int tipo_volo;
   int tipo_stand;
   char *compagnia;
   char *num_volo;
public:
   Aereo();
```

```
   void Init(long int, long int, int, int, const char *, const char *);

   char* Ritornanumv();
   int Ritornatipoa();
   int Ritornatipov();
   int Ritornatipos();
   long int Ora_prev(int);
   long int Ora_eff(int);
   void Instipostand(int);
   void Insoraeff(long int,int);

   long int Arrivo(float inf=-20, float sup=30, float med=-5);
   long int Tempo_percorsofinale(float, float, float, float);
   long int Tempo_pista(float, float);
   long int Tempo_taxia(float, float);
   long int Tempo_stand(float, float);
   long int Tempo_taxip(float, float);
   long int Ritardo_comunicazione(float);
};
```

```
Aereo::Aereo()
{
   time=0;
   ora_preva=-1;
   ora_prevp=-1;
   ora_effa=-1;
   ora_effp=-1;
   tipo_aereo=-1;
   tipo_volo=-1;
   tipo_stand=-1;
};
void Aereo::Init(long int oraa, long int orap, int tipoa, int tipov, const char *comp, const
char *n_vol)
{
   int lung;
   time=0;
   ora_preva=oraa;
   ora_prevp=orap;
   tipo_aereo=tipoa;
   tipo_volo=tipov;
   lung=strlen(comp);
   compagnia=new char[lung+1];
   strcpy(compagnia,comp);
   lung=strlen(n_vol);
   num_volo=new char[lung+1];
   strcpy(num_volo,n_vol);
};
```

```
char* Aereo::Ritornanumv()
{
   return num_volo;
};
int Aereo::Ritornatipoa()
{
   return (tipo_aereo);
};
int Aereo::Ritornatipov()
{
   return (tipo_volo);
};
int Aereo::Ritornatipos()
{
   return (tipo_stand);
};
long int Aereo::Ora_prev(int pos)
{
   if (pos==0)
      return ora_preva;
   else
      return ora_prevp;
};



long int Aereo::Ora_eff(int pos)
{
   if (pos==0)
      return ora_effa;
   else
      return ora_effp;
};
void Aereo::Instipostand(int tipo)
{
   tipo_stand=tipo;
   return;
};
void Aereo::Insoraeff(long int ora, int pos)
{
   if (pos==0)
      ora_effa=ora;
   else
      ora_effp=ora;
   return;
};
```

```
long int Aereo::Arrivo(float inf, float sup, float med)
{
   time=ora_preva+INT(triang(inf,med,sup)*60,0);
   return time;
};
long int Aereo::Tempo_percorsofinale(float lung_percorso, float mvel, float sdvel, float
errore)
{
   float vel,err_pos;
   long int tempo;
   vel=inv_norm(mvel, sdvel);
   err_pos=inv_norm(0,errore);
   tempo=INT(((lung_percorso+err_pos)/vel)*3600,0);
   time+=tempo;
   return tempo;
};
long int Aereo::Tempo_pista(float mtp, float sdtp)
{
   long int tempo;
   tempo=INT(inv_norm(mtp,sdtp)*60,0);
   time+=tempo;
   return tempo;
};
```

```
long int Aereo::Tempo_taxia(float mta, float sdta)
{
   long int tempo;
   tempo=INT(inv_norm(mta,sdta)*60,0);
   time+=tempo;
   return tempo;
};
long int Aereo::Tempo_stand(float ms, float sds)
{
   long int tempo;
   tempo=INT(inv_norm(ms,sds)*60,0);
   time+=tempo;
   return tempo;
};
long int Aereo::Tempo_taxip(float mtp, float sdtp)
{
   long int tempo;
   tempo=INT(inv_norm(mtp,sdtp),0);
   time+=tempo;
   return tempo;
};
long int Aereo::Ritardo_comunicazione(float rit)
{
```

```
   long int tempo=INT(inv_norm(0,rit),0);
   time+=tempo;
   return tempo;
};
```

3.4.6: IMPLEMENTAZIONE DEL PROGRAMMA PRINCIPALE

Il programma principale consta di diverse fasi di implementazione, ognuna riguardante una funzione delle entità all'interno del modello.

Il primo passo riguarda il calcolo degli orari di arrivo degli aeromobili, ossia il momento in cui si presentano sugli schermi dei controllori di volo. Tale arrivo è computato su tutte le entità presenti ad eccetto di quelle che già stazionano nel sistema, i cosiddetti *Overnight*.

Il ciclo è effettuato su tutte le entità *aerei[i]*, che è un vettore di oggetti di tipo Aereo. Si controlla che l'ora di arrivo prevista sia diversa da *-1* e solo in quel caso inserire l'evento arrivo nel clock della pista, poiché l'elemento *-1* significa che l'aereo è un *Overnight*.

```
   arrivo=-1;
   for (i=0;i<n_aerei;i++) //CALCOLO L'ARRIVO
   {
      if (aerei[i].Ora_prev(0)!=-1) //-1 per i voli solo in partenza
      {
         arrivo=aerei[i].Arrivo();
         inizio[i]=-1;
         clockp[arrivo].Inserisci(i,clockp[arrivo].Fine());
      }
   }
```

Il secondo passo computa invece il momento in cui gli aeromobili toccano terra. Si sviluppa su di un ciclo che analizza tutti gli eventi del clock in tutti i momenti dell'unità di misura, che in questo caso è il secondo. Il clock è strutturato come un vettore di oggetti lista, di dimensione 86400, cioè i secondi che formano il giorno. Ogni volta che entra nel ciclo effettua le seguenti operazioni: il puntatore *pos* punta sul primo elemento della lista; controlla se è l'uscita dalla pista di un aereo ed eventualmente libera il servente; controlla se ci sono elementi in coda ed eventualmente calcola il momento di atterraggio e lo inserisce nel clock degli eventi oppure computa e memorizza per quella data entità l'eventuale momento in cui è possibile iniziare il percorso di approccio finale. Dopodichè cicla su eventuali elementi della lista, che rappresenta gli arrivi e ripete le operazioni effettuate sugli elementi in coda.

```
iat=0;
tipoaereo=-1;
pos=NULL;
for(t=0;t<86400;t++) //ARRIVI
{
   pos=clockp[t].Primo(); //punto sul primo della lista di eventi
   if (pos->succ!=NULL)
   {
      if (clockp[t].Valore(pos)==-1) //controllo se è l'uscita dalla pista
      {
         atene.Libera_pista(0);
         pos=clockp[t].Successiva(pos);
      }
   }
   if (!C_arrivi.Vuota()) //controllo se ci sono elementi in coda
   {
      idxaereo=C_arrivi.Valore();  //mi restituisce l'indice dell'aereo
      tipoaereo=aerei[idxaereo].Ritornatipoa();
      if (inizio[idxaereo]==-1)  //controllo se non ha iniziato il percorso finale
      {
         iat=atene.Dist_mina(tipoprec,tipoaereo);  //calcolo il tempo di interrarrivo tra
```

```
il precedente ed il successivo
            if (inizio[tipoprec]+iat<=t)  //controllo se può occupare il percorso finale
            {
               inizio[idxaereo]=t;
               tipoprec=tipoaereo;
            }
         }
         if (atene.Pista_libera(0)) //controllo se la pista è libera
         {
            t_percfin=aerei[idxaereo].Tempo_percorsofinale(atene.Lungpercfin(),
atene.Velmedia(tipoaereo), atene.Velstdev(tipoaereo), atene.Errorepos()); //calcolo il
tempo per fare il percorso finale
            occ_pista=aerei[idxaereo].Tempo_pista(atene.Occpistam(tipoaereo,0),
atene.Occpistasd(tipoaereo,0)); //calcolo il tempo in pista
            atterraggio=inizio[idxaereo]+t_percfin; //momento in cui tocca terra
            aerei[idxaereo].Insoraeff(atterraggio,0); //inserisco l'ora effettiva di arrivo
            fineoccp=atterraggio+occ_pista; //momento in cui libera la pista
            clockt[fineoccp].Inserisci(idxaereo,clockt[fineoccp].Fine());
            clockp[fineoccp].Inserisci(-1,clockp[fineoccp].Primo()); //inserisco il
momento in cui si libera la pista
            atene.Occupa_pista(0,aerei[idxaereo].Ritornatipoa()); //occupo la pista
            check=C_arrivi.Cancella(); //cancello dalla coda
            if (check)
            {
               cout << "Errore al punto 2" << endl;
               return 1;
            }
         }
      }
      while(pos->succ!=NULL)
      {
         idxaereo=clockp[t].Valore(pos);  //mi restituisce l'indice dell'aereo
         if (inizio[idxaereo]==-1)  //controllo se non ha iniziato il percorso finale
         {
            if (tipoprec!=-1)
            {
               iat=atene.Dist_mina(tipoprec,tipoaereo);  //calcolo il tempo di interrarrivo
tra il precedente ed il successivo
               if (inizio[tipoprec]+iat<=t) //controllo se può occupare il percorso finale
               {
                  inizio[idxaereo]=t;
                  tipoprec=tipoaereo;
               }
            }
            else
            {
               tipoprec=tipoaereo;
               inizio[idxaereo]=t;
            }
         }
```

```
        if (atene.Pista_libera(0)) //se la pista è libera atterra
        {
           t_percfin=aerei[idxaereo].Tempo_percorsofinale(atene.Lungpercfin(),
atene.Velmedia(tipoaereo), atene.Velstdev(tipoaereo), atene.Errorepos()); //calcolo il
tempo per fare il percorso finale
           occ_pista=aerei[idxaereo].Tempo_pista(atene.Occpistam(tipoaereo,0),
atene.Occpistasd(tipoaereo,0)); //calcolo il tempo in pista
           atterraggio=inizio[idxaereo]+t_percfin; //momento in cui tocca terra
           aerei[idxaereo].Insoraeff(atterraggio,0); //inserisco l'ora effettiva di arrivo
           fineoccp=atterraggio+occ_pista; //momento in cui libera la pista
           clockt[fineoccp].Inserisci(idxaereo,clockt[fineoccp].Fine());
           clockp[fineoccp].Inserisci(-1,clockp[fineoccp].Primo()); //inserisco il
momento in cui si libera la pista
           atene.Occupa_pista(0,aerei[idxaereo].Ritornatipoa()); //occupo la pista
        }
        else  //altrimenti inserisci in coda
        {
           check=C_arrivi.Inserisci(idxaereo);  //lo inserisco nella coda
           if (check)
           {
              cout << "Errore al punto 1";
              return 1;
           }
        }
        pos=clockp[t].Successiva(pos);
     }
  }
```

Il terzo passo computa il tempo di esecuzione delle taxiway di uscita dalla pista. Come prima il ciclo si basa sul clock degli eventi e si sviluppa nello stesso modo del passo precedente. L'unica modifica è nel fatto che occupa gli stand per gli aerei *Overnight*.

```
taxi=0;
  idxaereo=-1;
  pos=NULL;
  for(t=0;t<86400;t++) //TEMPO TAXIWAY ARRIVI
  {
     pos=clockt[t].Primo();
     tipos=0;
     while ((pos->succ)!=NULL)
     {
        idxaereo=clockt[t].Valore(pos);
        taxi=t+aerei[idxaereo].Tempo_taxia(atene.Occtaxim(0),atene.Occtaxisd(0));
```

```
        clocks[taxi].Inserisci(taxi,clocks[taxi].Fine());   //aggiungo il tempo delle
taxiway
        if (aerei[idxaereo].Ora_prev(0)==-1)
        {
           tipos=atene.Occupa_stand(aerei[idxaereo].Ritornatipoa(),
aerei[idxaereo].Ritornatipov()); //occupo gli stand per gli overnight già presenti
           aerei[idxaereo].Instipostand(tipos);
        }
        pos=clockt[t].Successiva(pos);
     }
  }
```

Il quarto passo va a calcolare la permanenza negli stand. Si sviluppa anch'esso con un ciclo sugli elementi del clock. Consta dei seguenti passi: *pos* punta sul primo elemento della lista; cicla finché ci sono stand da liberare; controlla se ci sono elementi in coda ed eventualmente alloca tali elementi a possibili stand liberi, memorizzando quale servente è stato assegnato ad ogni entità; infine cicla su tutte le posizioni della lista che rappresentano l'arrivo di un aereo da servire. Chiaramente il tempo totale di servizio sarà computato il maggiore tra il tempo schedulato di partenza e l'esecuzione di tutte le funzioni di handling effettuate nello stand. Invece il servente rimarrà occupato per un tempo superiore rappresentato dalla necessità di prepararlo ad accogliere un nuovo aereo.

```
  pos=NULL;
  idxaereo=-1;
  tipos=-1;
  turnaround=0;
  partenza=0;
  check=0;
  t_preps=0;
  t_tots=0;
  for(t=0;t<86400;t++) //TEMPO STAND
  {
```

```
      pos=clocks[t].Primo(); //punto sul primo della lista di eventi
      if (pos->succ!=NULL)   //controllo se ci sono stand liberi
      {
         idxaereo=clocks[t].Valore(pos);
         l=1;
         while (idxaereo<0 && l==1)   //ciclo finché ci sono stand da liberare
         {
            idxaereo=abs(idxaereo);
            atene.Libera_stand(aerei[idxaereo].Ritornatipos(),
aerei[idxaereo].Ritornatipov());
            if (pos->succ!=NULL)
            {
               pos=clocks[t].Successiva(pos);
               idxaereo=clocks[i].Valore(pos);
            }
            else
               l=0;
         }
         pos=clocks[t].Precedente(pos);
      }
      c=1;
      while (!C_stand.Vuota() && c==1) //ciclo su tutti gli elementi della coda
      {
         idxaereo=C_stand.Valore();
         if (atene.Stand_libero(aerei[idxaereo].Ritornatipoa(),
aerei[idxaereo].Ritornatipov())) //controllo se ci sono stand liberi
         {  //occupo gli eventuali stand liberi con gli elementi in coda
            tipos=atene.Occupa_stand(aerei[idxaereo].Ritornatipoa(),
aerei[idxaereo].Ritornatipov()); //occupa lo stand
            aerei[idxaereo].Instipostand(tipos);   //memorizzio il tipo di stand occupato
            if (aerei[idxaereo].Ora_prev(1)!=-1) //controllo che non stiano lì la notte
            {
               turnaround=t+aerei[idxaereo].Tempo_stand(atene.Occstandm(
aerei[idxaereo].Ritornatipoa()),
               atene.Occstandsd(aerei[idxaereo].Ritornatipoa()));//dai il tempo dello stand
               partenza=t+aerei[idxaereo].Ora_prev(1);
               check=C_stand.Cancella();
               if (check)
               {
                  cout << "Errore al punto 4";
                  return 1;
               }
               t_preps=atene.Tempo_prepstand();
               if (turnaround<=partenza)  //confronto quale tempo è maggiore
               {
                  t_tots=partenza+t_preps;
                  clocks[t_tots].Inserisci(-idxaereo,clocks[t_tots].Primo());
                  clockp[partenza].Inserisci(idxaereo,clockp[t_tots].Primo());
               }
               else
```

```
                {
                    t_tots=turnaround+t_preps;
                    clocks[t_tots].Inserisci(-idxaereo,clocks[t_tots].Primo());
                    clockp[turnaround].Inserisci(idxaereo,clockp[t_tots].Primo());
                }
            }
        }
        else
            c=0;
    }
    while(pos->succ!=NULL)
    {
        idxaereo=clocks[t].Valore(pos);
        if (atene.Stand_libero(aerei[idxaereo].Ritornatipoa(),
aerei[idxaereo].Ritornatipov())) //controllo se ci sono stand liberi
        {
            tipos=atene.Occupa_stand(aerei[idxaereo].Ritornatipoa(),
aerei[idxaereo].Ritornatipov()); //occupa lo stand
            aerei[idxaereo].Instipostand(tipos);   //memorizzio il tipo di stand occupato
            if (aerei[idxaereo].Ora_prev(1)!=-1) //controllo che non stiano lì la notte
            {
                turnaround=t+aerei[idxaereo].Tempo_stand(atene.Occstandm(
aerei[idxaereo].Ritornatipoa()),atene.Occstandsd(aerei[idxaereo].Ritornatipoa()));//dai il
tempo dello stand
                partenza=t+aerei[idxaereo].Ora_prev(1);
                t_preps=atene.Tempo_prepstand();
                if (turnaround<=partenza)  //confronto quale tempo è maggiore
                {
                    t_tots=partenza+t_preps;
                    clocks[t_tots].Inserisci(-idxaereo,clocks[t_tots].Primo());
                    clockp[partenza].Inserisci(idxaereo,clockp[t_tots].Primo());
                }
                else
                {
                    t_tots=turnaround+t_preps;
                    clocks[t_tots].Inserisci(-idxaereo,clocks[t_tots].Primo());
                    clockp[turnaround].Inserisci(idxaereo,clockp[t_tots].Primo());
                }
            }
        }
        else //altrimenti inserisci in coda
        {
            check=C_stand.Inserisci(idxaereo);
            if (check)
            {
                cout << "Errore al punto 3";
                return 1;
            }
        }
        pos=clockt[t].Successiva(pos);
```

```
    }
  }
```

Al quinto passo si aggiunge il tempo delle taxiway per entrare in pista.

```
  taxi=0;
  idxaereo=-1;
  pos=NULL;
  for(t=0;t<86400;t++) //TEMPO TAXIWAY PARTENZE
  {
    pos=clockt[t].Primo();
    taxi=0;
    while ((pos->succ)!=NULL)
    {
      idxaereo=clockt[t].Valore(pos);
      taxi=t+aerei[idxaereo].Tempo_taxip(atene.Occtaxim(1),atene.Occtaxisd(1));
      clockp[taxi].Inserisci(taxi,clockp[taxi].Fine());   //aggiungo il tempo delle
taxiway
      pos=clockt[t].Successiva(pos);
    }
  }
```

Infine il sesto ed ultimo passo computa i momenti delle partenze. Il ciclo è sempre sugli eventi del clock e effettua i seguenti passi: *pos* punta sul primo elemento della lista; controlla se esiste un evento che libera la pista ed in tal caso oltre a disimpegnare il servente si memorizza il momento in cui è liberata e il tipo di aereo che la occupava; controlla se sono presenti elementi in coda e se è possibile farli decollare nel pieno rispetto delle distanze minime imposte; infine cicla su tutti gli elementi della lista e decide se permettere la partenza o inserirli in coda, sempre in funzione della distanza minima in minuti con l'entità che ha appena liberato il servente.

```
   pos=NULL;
   partito_tempo=0;
   partito_tipo=-1;
   idxaereo=-1;
   tipoaereo=-1;
   distminima=0;
   tocc_pista=0;
   decollo=0;
   check=0;
   for(t=0;t<86400;t++) //PARTENZE
   {
      pos=clockp[t].Primo(); //punto sul primo della lista di eventi
      if (pos->succ!=NULL)
      {
         if (clockp[t].Valore(pos)==-1) //controllo se è l'uscita dalla pista
         {
            partito_tempo=t;
            partito_tipo=atene.Tipo_occupa(1);
            atene.Libera_pista(0);
            pos=clockp[t].Successiva(pos);
         }
      }
      if (!C_partenze.Vuota()) //controllo se ci sono elementi in coda
      {
         if (atene.Pista_libera(0)) //controllo se la pista è libera per eventuali aerei in coda
         {
            idxaereo=C_partenze.Valore(); //mi restituisce l'indice dell'aereo
            tipoaereo=aerei[idxaereo].Ritornatipoa();
            distminima=atene.Dist_minp(partito_tipo,tipoaereo);
            if ((t-partito_tempo)>=distminima) //controllo se può decollare rispettando le
distanze
            {
               if (atene.Ritcomun()!=0)
                  ritcom=aerei[idxaereo].Ritardo_comunicazione(atene.Ritcomun());
               else
                  ritcom=0;
               tocc_pista=aerei[idxaereo].Tempo_pista(atene.Occpistam(tipoaereo,1),
atene.Occpistasd(tipoaereo,1)); //calcolo il tempo in pista
               decollo=t+ritcom+tocc_pista;   //li aggiungo al momento in cui il decollo
               aerei[idxaereo].Insoraeff(decollo,1);
               clockp[decollo].Inserisci(-1,clockp[decollo].Primo());  //inserisco il
momento in cui si libera la pista
               atene.Occupa_pista(1,aerei[idxaereo].Ritornatipoa()); //occupo la pista da
un aereo del tipo indicato
               check=C_partenze.Cancella();  //cancello dalla coda
               if (check)
               {
                  cout << "Errore al punto 5" << endl;
```

```
                return 1;
            }
        }
    }
}
while(pos->succ!=NULL) //inizio il ciclo per esplorare la lista
{
    idxaereo=clockp[t].Valore(pos);  //mi restituisce l'indice dell'aereo
    if (partito_tipo!=-1)
    {
        if (atene.Pista_libera(0)) //controllo se la pista è libera
        {
            tipoaereo=aerei[idxaereo].Ritornatipoa();
            distminima=atene.Dist_minp(partito_tipo,tipoaereo);
            if ((t-partito_tempo)>=distminima)  //controllo se può decollare rispettando
le distanze
            {
                if (atene.Ritcomun()!=0)
                ritcom=aerei[idxaereo].Ritardo_comunicazione(atene.Ritcomun());
            else
                ritcom=0;
                tocc_pista=aerei[idxaereo].Tempo_pista(atene.Occpistam(tipoaereo,1),
atene.Occpistasd(tipoaereo,1)); //calcolo il tempo in pista
                decollo=t+ritcom+tocc_pista;   //li aggiungo al momento in cui il decollo
                aerei[idxaereo].Insoraeff(decollo,1);
                clockp[decollo].Inserisci(-1,clockp[decollo].Primo());  //inserisco il
momento in cui si libera la pista
                atene.Occupa_pista(1,aerei[idxaereo].Ritornatipoa()); //occupo la pista
da un aereo del tipo indicato
            }
            else //altrimenti inserisco in coda
            {
                check=C_partenze.Inserisci(idxaereo);  //lo inserisco nella coda
                if (check)
                {
                    cout << "Errore al punto 7";
                    return 1;
                }
            }
        }
        else //altrimenti inserisco in coda
        {
            check=C_partenze.Inserisci(idxaereo);  //lo inserisco nella coda
            if (check)
            {
                cout << "Errore al punto 8";
                return 1;
            }
        }
    }
```

```
        else
        {
           tipoaereo=aerei[idxaereo].Ritornatipoa();
           tocc_pista=aerei[idxaereo].Tempo_pista(atene.Occpistam(tipoaereo,1),
atene.Occpistasd(tipoaereo,1)); //calcolo il tempo in pista
           decollo=t+tocc_pista;   //li aggiungo al momento in cui il decollo
           aerei[idxaereo].Insoraeff(decollo,1);
           clockp[decollo].Inserisci(-1,clockp[decollo].Primo());  //inserisco il momento
in cui si libera la pista
           atene.Occupa_pista(1,aerei[idxaereo].Ritornatipoa()); //occupo la pista da un
aereo del tipo indicato
        }
        pos=clockp[t].Successiva(pos);
     }
  }
```

3.5: ANALISI DEI RISULTATI

I risultati raggiunti sono i seguenti: i voli con ritardi maggiori ai 15 minuti sono mostrati nel Grafico 3-1.

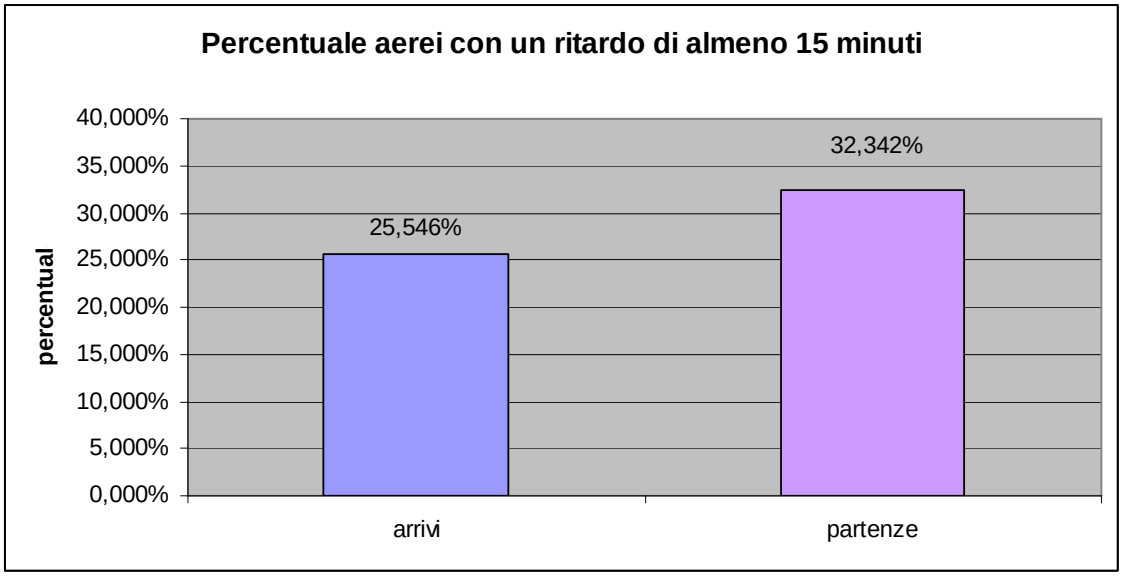

Grafico 3- 1

In media gli arrivi sono proporzionalmente inferiori alle partenze di circa il 7%.

Di seguito nel Grafico 3-2 sono evidenziati i ritardi medi in minuti per tipo di movimento. Gli arrivi hanno mediamente più minuti di ritardo, ma sull'ordine di quasi uno.

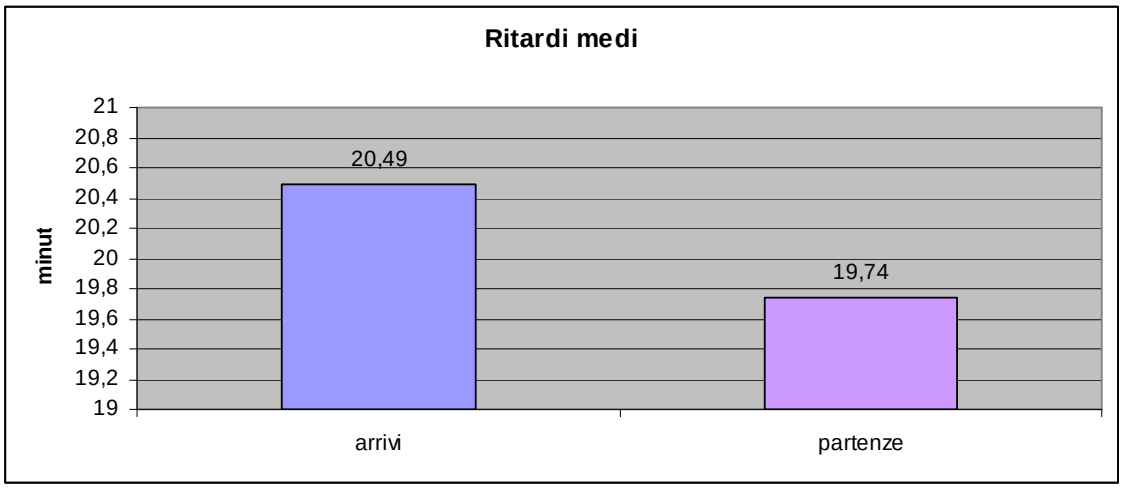

Grafico 3- 2

Nel Grafico 3-3 sono presentati i ritardi medi in minuti, divisi tra le ore del giorno di arrivi e partenze. Si nota come la linea dei ritardi delle partenze segua quelli degli arrivi quasi simmetricamente, a dimostrazione di come i ritardi di uno si riflettano sulle altre. Inoltre per comprendere meglio questo grafico è presentato anche, in Grafico 3-4, il numero di voli, sempre separati in atterraggi e decolli, diviso tra le ore del giorno. Si nota come in corrispondenza dell'aumento della domanda di utilizzo della pista dedicata agli arrivi aumentino anche i ritardi medi, e di come questo fenomeno, sommato alla richiesta di uso della pista dedicata alle partenze, si ripeta anche per i decolli. Entrambe le linee dei ritardi seguono quelle dei movimenti per ora, anche se ovviamente con un po' di

ridondanza dovuta al fatto che la congestione viene spalmata maggiormente nei momenti successivi all'effettivo avvenimento.

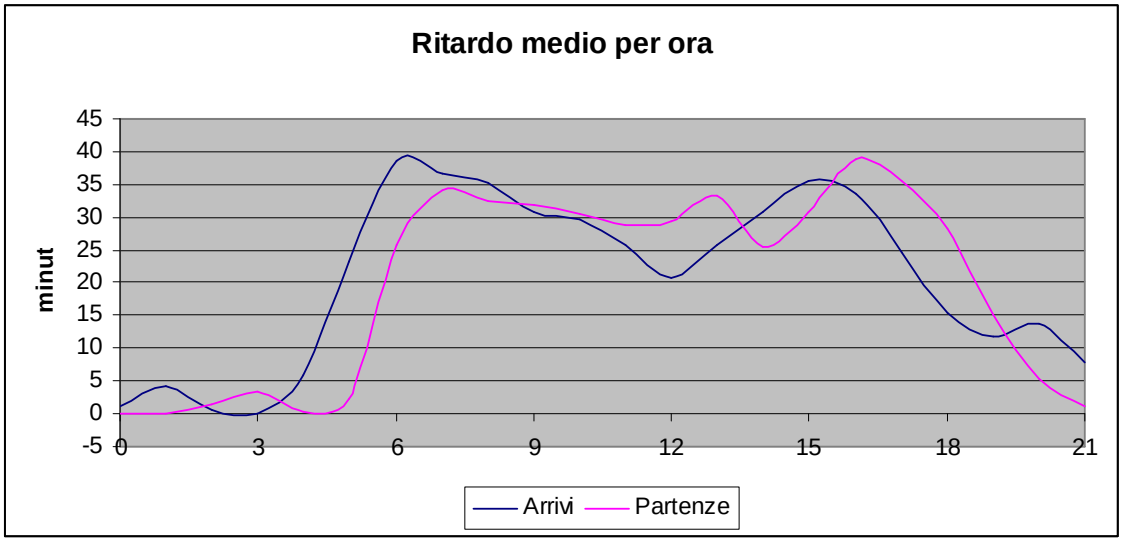

Grafico 3- 3

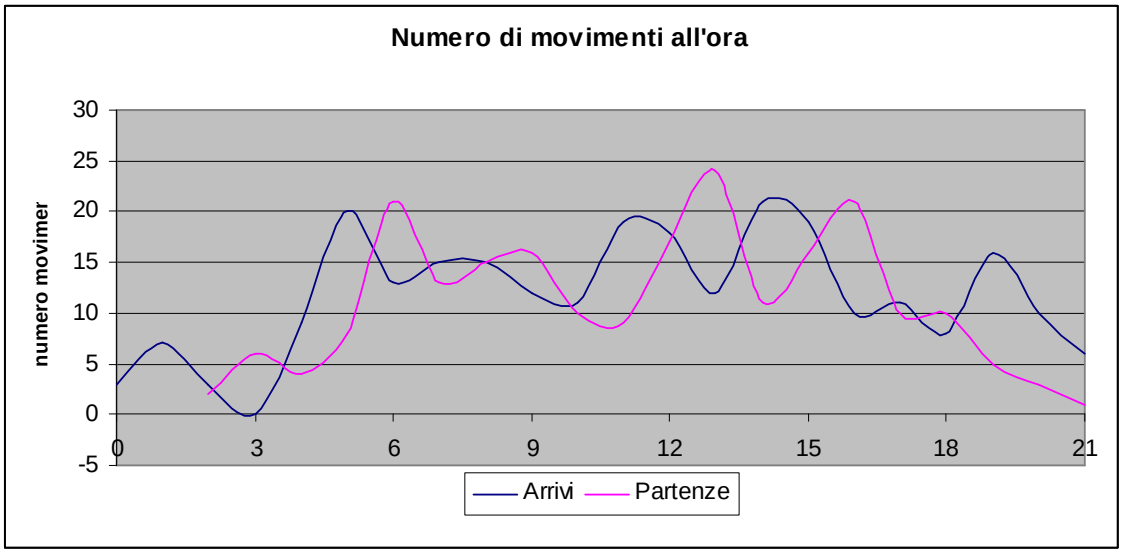

Grafico 3- 4

CONCLUSIONI

La costruzione di un solido pacchetto software per simulare il lato aereo di un aeroporto necessità di approfonditi studi sulla realtà da analizzare. Il caso qui esposto ha presentato notevoli semplificazioni che in un'opera professionale non sarebbero ammesse. I modelli usati sono comunque più che validi, poiché sono gli stessi sui quali si basa MACAD, anche se per perseguire un altro fine, cioè un'analisi analitica. Inoltre ha avuto lo scopo di fornire un'idea generica di come si comporta in condizioni standard un aeroporto, nella fattispecie quello di Atene, e di come la gestione del traffico aereo presenti notevoli problemi da affrontare. Sicuramente i pacchetti professionali forniscono risposte più dettagliate e fedeli alla realtà, ma l'oggetto di questa tesi era presentare dei casi di simulazione ed apportare un esempio pratico che ovviamente si avvicinasse maggiormente ai fenomeni reali. Difatti il lavoro si è soffermato maggiormente sulla presentazione di modelli e sull'esempio che sulla descrizione dei risultati finali, in quanto non ci si è potuti assicurare che rappresentassero al meglio la realtà, data la mancanza di altri output con cui potersi confrontare.

Spero però che il lavoro descritto e svolto sia comunque di aiuto a comprendere il funzionamento di un aeroporto a chi non si è mai avvicinato a questi argomenti.

BIBLIOGRAFIA

LIBRI:

Neufville, Richard de, Odoni, Amedeo R. *Airport System,* McGraw-Hill, New York, 2003.

Crivellari, Franco, *Elementi di programmazione con il C++,* FrancoAngeli, Milano, 1996.

Crivellari, Franco, *Programmazione ad oggetti e tipi di dati astratti con il C++,* FrancoAngeli, Milano, 1996.

ARTICOLI:

Stamatopoulos, Miltos A., Zografos Kostantinos G., Odoni Amedeo R., *A decision support system for airport strategic planning,* Elsevier Ltd., 2003.

MANUALI DI UTILIZZO:

MACAD User Manual.

The MACAD Decision Support System.

SITI INTERNET CONSULTATI:

Athens International Airport, disponibile su www.aia.gr.