POLITECNICO DI TORINO

Corso di Laurea
in Matematica per l'Ingegneria

Tesi di Laurea

Sport Scheduling Optimization

Relatori
prof. Paolo Brandimarte
firma dei relatori

Candidato
Davide Omento
firma del candidato

Anno Accademico 2022-2023

Sommario

La tesi si concentra sull'ottimizzazione dello scheduling sportivo, un settore rilevante per le leghe sportive e l'economia.
Dopo una breve introduzione ai problemi di ottimizzazione, vengono esaminati e tradotti in for- mule matematiche utilizzando l'ambiente $MATLAB$ i modelli matematici relativi alle situazioni da affrontare.
In particolare, si affrontano la generazione e l'ottimizzazione di calendari Round Robin(tutti contro tutti esattamente una volta) e le tecniche che si possono utilizzare a questo scopo. In seguito viene analizzato l'NBA Scheduling Problem, ovvero il problema che ha come obiettivo l'ottimizzazione del calendario nba. Viene esaminata la struttura matematica a cui si appoggia e vengono forniti dettagli sull'implementazione degli algoritmi utilizzati per ottenere una soluzione ottimale.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione generale

1.1 Introduzione allo scheduling sportivo

L'ambito sportivo è diventato un settore economico di ampia portata e nazioni e città fanno a gara per ospitare eventi di rilievo garantendosi notevoli benefici economici. Le competizioni sportive attirano milioni di spettatori da tutto il mondo che seguono le proprie squadre sia allo stadio che da casa tramite TV, radio e giornali.
Le leghe sportive professionistiche rappresentano una componente importante dell'economia e affrontano sfide complesse di ottimizzazione, come la massimizzazione dei ricavi e la razionalizzazione della logistica.
Una pianificazione accurata delle competizioni sportive è fondamentale per vari motivi, tra cui il coinvolgimento degli sponsor e i diritti televisivi. Un buon calendario assicura un'ottimizzazione degli aspetti logistici e promozionali, garantendo un impatto positivo sulle entrate e sulla visibilità.

Per avere un'idea delle cifre che le aziende televisive sono disposte a investire, si può osservare l'ammontare ottenuto tramite i diritti televisivi da ciascuna squadra della Premier League (la lega di calcio inglese) nella stagione 2021/2022. In totale, la lega sportiva ha ottenuto oltre due miliardi di sterline da questi diritti. Questi numeri offrono un'indicazione dei notevoli investimenti effettuati per acquisire i diritti di trasmissione delle competizioni sportive.

Premier League | 2021/22 Payments to Clubs

Club Name	Live	UK			International		Central Commercial	Total Payment
		Equal Share	Facility Fees	Merit Payment	Equal Share	Merit Payment		
Manchester City	28	31,809,969	24,436,525	33,779,160	48,885,768	7,365,240	6,814,232	**153,090,894**
Liverpool	29	31,809,969	25,274,331	32,090,202	48,885,768	6,996,978	6,814,232	**151,871,480**
Chelsea	24	31,809,969	21,085,299	30,401,244	48,885,768	6,628,716	6,814,232	**145,625,228**
Tottenham Hotspur	27	31,809,969	23,598,718	28,712,286	48,885,768	6,260,454	6,814,232	**146,081,427**
Arsenal	29	31,809,969	25,274,331	27,023,328	48,885,768	5,892,192	6,814,232	**145,699,820**
Manchester United	28	31,809,969	24,436,525	25,334,370	48,885,768	5,523,930	6,814,232	**142,804,794**
West Ham United	23	31,809,969	20,247,493	23,645,412	48,885,768	5,155,668	6,814,232	**136,558,542**
Leicester City	16	31,809,969	14,382,847	21,956,454	48,885,768	4,787,406	6,814,232	**128,636,676**
Brighton & Hove Albion	15	31,809,969	13,545,041	20,267,496	48,885,768	4,419,144	6,814,232	**125,741,650**
Wolverhampton Wanderers	16	31,809,969	14,382,847	18,578,538	48,885,768	4,050,882	6,814,232	**124,522,236**
Newcastle United	21	31,809,969	18,571,880	16,889,580	48,885,768	3,682,620	6,814,232	**126,654,049**
Crystal Palace	16	31,809,969	14,382,847	15,200,622	48,885,768	3,314,358	6,814,232	**120,407,796**
Brentford	16	31,809,969	14,382,847	13,511,664	48,885,768	2,946,096	6,814,232	**118,350,576**
Aston Villa	20	31,809,969	17,734,073	11,822,706	48,885,768	2,577,834	6,814,232	**119,644,582**
Southampton	12	31,809,969	11,031,621	10,133,748	48,885,768	2,209,572	6,814,232	**110,884,910**
Everton	22	31,809,969	19,409,686	8,444,790	48,885,768	1,841,310	6,814,232	**117,205,755**
Leeds United	22	31,809,969	19,409,686	6,755,832	48,885,768	1,473,048	6,814,232	**115,148,535**
Burnley	12	31,809,969	11,031,621	5,066,874	48,885,768	1,104,786	6,814,232	**104,713,250**
Watford	12	31,809,969	11,031,621	3,377,916	48,885,768	736,524	6,814,232	**102,656,030**
Norwich City	12	31,809,969	11,031,621	1,688,958	48,885,768	368,262	6,814,232	**100,598,810**
All figures in £		636,199,380	354,681,464	354,681,180	977,715,360	77,335,020	136,284,640	**2,536,897,044**

Figura 1.1: Profitto per la vendita dei diritti televisivi.

Il crescente interesse verso la pianificazione e la gestione nello sport da parte di studiosi provenienti da diverse discipline è alimentato dalla complessità intrinseca di tali sfide. Le varie tecniche di ottimizzazione, basate sulla ricerca operativa, la programmazione a vincoli, la teoria dei grafi e la matematica applicata, costituiscono strumenti preziosi per affrontare con successo i problemi di questo settore e contribuire allo sviluppo di soluzioni innovative e efficienti.

Capitolo 2

Problemi di ottimizzazione

2.1 Definizione

In matematica e in informatica, un problema di ottimizzazione consiste nel trovare la migliore soluzione fra tutte le soluzioni ammissibili.

Esso si può formalizzare in termini matematici e scrivere in questo modo:

$$\min \quad f(x) \tag{2.1}$$

$$\text{s.t.} \quad \begin{cases} g_i(x) \leq 0 \quad i = 1, ..., m \\ \\ f_j(x) = 0 \quad j = 1, ..., p \end{cases} \tag{2.2}$$

dove

- $f(x) : R^n \to R$ è la funzione obiettivo da minimizzare sulla variabile x.
- $g_i(x) \leq 0$ sono chiamati vincoli di disuguaglianza.
- $f_j(x) = 0$ sono chiamati vincoli di uguaglianza.

Non tutti i problemi di ottimizzazione richiedono necessariamente la minimizzazione di una funzione; in alcuni casi, è fondamentale massimizzare un valore. Pertanto, nel caso in cui il problema richieda di massimizzare il valore della funzione obiettivo, basta introdurre un segno negativo di fronte a tale funzione. Questa procedura permette di riformulare un problema di massimizzazione in un problema di minimizzazione, mantenendo intatta la struttura complessiva e gli obiettivi del problema.

Nel contesto della programmazione matematica, una **soluzione ammissibile** è un punto nello spazio delle variabili che rispetta le restrizioni del problema, ovvero soddisfa tutti i vincoli imposti dalle equazioni e dalle disuguaglianze. Ad esempio, in un problema di

programmazione lineare, una soluzione ammissibile è un punto che giace all'interno dell'insieme definito dalle restrizioni lineari.
In sintesi, una soluzione ammissibile è quella che rispetta tutte le condizioni del problema, consentendo di considerarla come una possibile candidata per la soluzione ottimale.

2.2 Classificazione

2.2.1 Ottimizzazione continua e intera

- L'**ottimizzazione continua** si riferisce ai problemi in cui le variabili di decisione possono assumere qualsiasi valore all'interno di un intervallo continuo. In altre parole, non ci sono restrizioni che vincolano le variabili ad essere valori discreti. Le sfide di ottimizzazione continua sono spesso risolte utilizzando calcolo differenziale e integrale.
  Ad esempio, nell'ottimizzazione continua, si potrebbe cercare di massimizzare o minimizzare una funzione obiettivo soggetta a vincoli che coinvolgono equazioni o disuguaglianze continue. Una situazione tipica potrebbe essere quella di ottimizzare i profitti di una produzione data una serie di vincoli sulla disponibilità delle risorse.

- L'**ottimizzazione intera**, d'altra parte, riguarda i problemi in cui almeno alcune delle variabili di decisione devono assumere valori interi anziché continui. Questo aggiunge una sfida significativa, poiché la presenza di variabili intere rende tutto più complesso da risolvere. Gli algoritmi che funzionano bene nell'ottimizzazione continua potrebbero non essere efficaci in questo contesto.

  Nell'ottimizzazione intera, i vincoli possono richiedere che le variabili abbiano valori interi (0, 1, 2, ecc.), il che rende l'esplorazione dello spazio delle soluzioni più complicata. Ad esempio, potresti dover assegnare determinate risorse a diversi progetti, dove ogni risorsa può essere assegnata solo in quantità intere.

2.2.2 Ottimizzazione lineare e non lineare

- L'**ottimizzazione lineare** riguarda la ricerca della soluzione migliore in un sistema in cui sia la funzione obiettivo che i vincoli sono lineari. Ovvero la funzione obiettivo e i vincoli possono essere espressi come combinazioni lineari delle variabili coinvolte.

La forma generale di un problema di ottimizzazione lineare è:

$$\min \quad c_1x_1 + c_2x_2 + ... + c_nx_n \tag{2.3}$$

$$\text{s.t.} \quad \begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b_2 \\ ... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m \\ x_1, x_2, ..., x_n \geq 0 \end{cases} \tag{2.4}$$

Dove:

- $x_1, x_2, ..., x_n$ sono le variabili di decisione che devono essere determinate per ottenere la soluzione ottimale.
- $c_1, c_2, ..., c_n$ sono i coefficienti della funzione obiettivo da massimizzare o minimizzare.
- a_{ij} sono i coefficienti dei vincoli lineari.
- b_i rappresenta i valori limite per ciascun vincolo.

- **L'ottimizzazione non lineare** riguarda il caso in cui la funzione obiettivo o i vincoli coinvolgono relazioni non lineari tra le variabili. Cioè, almeno uno dei termini della funzione obiettivo o dei vincoli è una funzione non lineare delle variabili.

  I problemi di ottimizzazione non lineare sono in generale molto più complessi da risolvere rispetto a quelli lineari, poiché le tecniche di risoluzione richiedono spesso iterazioni numeriche, algoritmi di ricerca e possono coinvolgere minimi o massimi locali anziché globali. Alcuni esempi includono l'ottimizzazione di funzioni quadrate, esponenziali, trigonometriche e altre funzioni non lineari.

  Esistono diversi metodi per affrontare l'ottimizzazione non lineare, come i metodi di Newton, i metodi di gradiente, i metodi di ricerca diretta, tra gli altri. La scelta del metodo dipenderà dalla natura specifica del problema e dai requisiti di prestazioni.

2.2.3 Ottimizzazione lineare mista intera

Un modello **MILP** (**Mixed-Integer Linear Programming**) è un metodo di ottimizzazione che combina elementi di programmazione lineare (LP) e programmazione intera

(IP). In un problema MILP, alcune delle variabili decisionali sono vincolate a essere intere, mentre altre possono assumere valori continui. L'obiettivo è minimizzare o massimizzare una funzione obiettivo lineare, soggetta a un insieme di vincoli lineari.

La programmazione lineare (LP) coinvolge variabili continue e richiede di ottimizzare una funzione obiettivo lineare soggetta a vincoli lineari. D'altra parte, nella programmazione intera (IP), alcune o tutte le variabili decisionali devono essere vincolate a valori interi. Spesso le variabili decisionali intere assumono valori 0 e 1.

Nel modello in questione, alcune variabili sono ammesse a essere continue, mentre altre devono essere intere. Questo permette di affrontare problemi in cui alcune decisioni possono essere prese in modo frazionario (continuo), mentre altre devono essere decisioni discrete (interi), il che può rappresentare situazioni reali in cui alcune variabili hanno significati discreti o "conteggi" naturali (come il numero di oggetti da selezionare, il numero di volte da eseguire un'azione, ecc.).

I modelli MILP si presentano in molti contesti, come la pianificazione, la logistica, l'allocazione delle risorse, la produzione e altri settori in cui è necessario prendere decisioni ottimali che coinvolgono sia variabili continue che variabili intere.

2.3 Algoritmi di risoluzione

2.3.1 Greedy algorithm

Gli algoritmi avidi (o "greedy algorithms" in inglese) sono una classe di algoritmi che cercano di fornire una soluzione facendo scelte localmente ottimali in ogni passo, nella speranza che tali scelte portino a una soluzione globalmente ottimale. In altre parole, un algoritmo avido cerca di massimizzare o minimizzare una certa funzione obiettivo in ogni passo, senza considerare le conseguenze future delle scelte fatte.

Funzionamento:

- **Inizializzazione**: l'algoritmo avido inizia con una soluzione vuota o parziale e una lista di opzioni da cui scegliere.
- **Scelta locale ottima**: in ogni passo, l'algoritmo seleziona l'opzione migliore in base a un criterio definito. Questo criterio può essere il massimo valore, il minimo costo, la massima efficienza, ecc. L'importante è che la scelta sia ottimale solo nel contesto dell'attuale passo, senza considerare gli effetti a lungo termine.
- **Aggiornamento della soluzione**: una volta fatta la scelta, l'opzione selezionata viene aggiunta alla soluzione parziale. In alcuni casi, questa scelta può anche escludere alcune opzioni future.

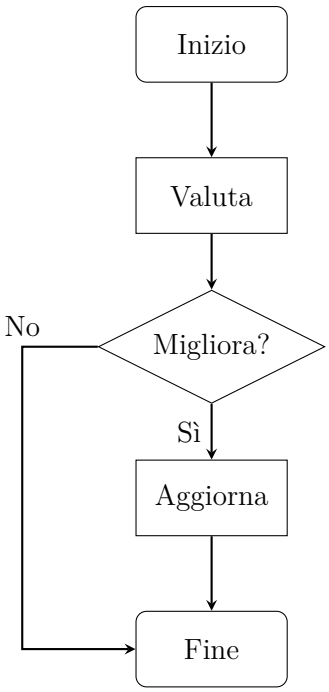
- **Ripetizione**: i passi 2 e 3 vengono ripetuti finché non si raggiunge una soluzione completa o accettabile.
- **Verifica**: infine, è importante verificare se la soluzione ottenuta soddisfa i requisiti del problema e se è effettivamente ottimale secondo il criterio definito. In alcuni casi, potrebbe essere necessario apportare modifiche alla soluzione o tornare a esaminare le scelte fatte nei passi precedenti.

Gli algoritmi avidi possono essere molto efficienti e semplici da implementare, ma è essenziale notare che non sempre portano a una soluzione ottimale per tutti i tipi di problemi. In alcuni casi, un'ottimizzazione locale potrebbe portare a una soluzione globale non ottimale, o addirittura ad una soluzione che non rispetta i vincoli imposti inizialmente. Pertanto, è fondamentale valutare attentamente il problema e le sue caratteristiche prima di decidere se un algoritmo avido è l'approccio giusto da seguire.

Un esempio comune di algoritmo avido è l'algoritmo di selezione di attività, in cui si cerca di selezionare un insieme massimale di attività compatibili da svolgere, ognuna con un certo valore e una certa durata. In ogni passo, si seleziona l'attività che offre il massimo valore rispetto alla sua durata attuale, senza considerare il futuro. Questo tipo di approccio è particolarmente efficace quando le scelte successive non influenzano retroattivamente le scelte precedenti.

2.3.2 Hill Climbing Algorithm

In analisi numerica, l'*Hill Climbing* è una tecnica di ottimizzazione matematica che rientra nella categoria delle ricerche locali. Questo algoritmo iterativo parte da una soluzione iniziale qualsiasi di un problema e cerca poi di migliorarla apportando cambiamenti piccoli e graduali. Se un cambiamento porta a una soluzione migliore, si effettua un altro cambiamento alla nuova soluzione e si continua così fino a quando non si riesce più a individuare miglioramenti ulteriori oppure fino a quando si raggiunge un certo numero di modifiche apportate.



Un esempio concreto è l'applicazione dell'hill *climbing* al problema del commesso viaggiatore. Spesso, si riesce a trovare una soluzione iniziale che visita tutte le città, ma questa soluzione sarà probabilmente molto peggiore rispetto a quella ottimale. L'algoritmo parte da questa soluzione iniziale e apporta piccole migliorie, ad esempio scambiando l'ordine in cui vengono visitate due città. Nel corso del tempo, si arriverà molto probabilmente a individuare un percorso notevolmente più breve.

Tuttavia, è importante notare che l'*Hill Climbing* è efficace per trovare soluzioni ottimali solo in presenza di problemi di forma convessa. Per problemi più complessi, l'algoritmo potrebbe raggiungere solo ottimi locali, cioè soluzioni che non possono essere ulteriormente migliorate da configurazioni adiacenti. Questi ottimi locali potrebbero non

rappresentare necessariamente la soluzione migliore possibile nell'intero spazio di soluzioni possibili.

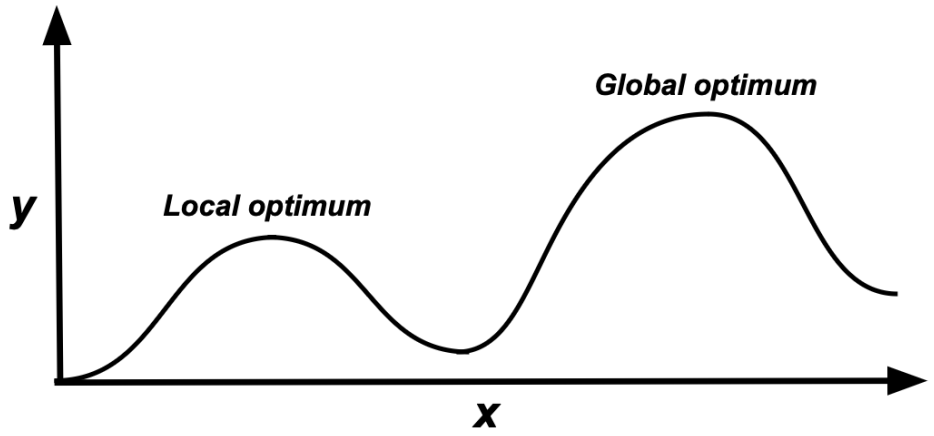


Figura 2.1: Funzione con ottimo locale e globale

Molte volte, algoritmi più avanzati possono offrire risultati migliori. Tuttavia, in alcune situazioni, soprattutto quando il tempo disponibile per la ricerca è limitato, ad esempio nei sistemi in tempo reale, questo algoritmo può produrre risultati altrettanto validi. Inoltre è importante sottolineare che è un algoritmo "anytime", ovvero può restituire una soluzione valida anche se viene interrotto prima di completare tutte le iterazioni previste.

Capitolo 3

Calendari Round Robin

3.1 Introduzione

I campionati Round Robin, noti anche come tornei all'italiana, costituiscono un fondamentale formato di competizione sportiva in cui ciascuna squadra si scontra con tutte le altre partecipanti. Il nome "Round Robin" deriva dall'antico termine inglese che indicava una pratica di scambio di racconti tra amici, che corrisponde all'idea di tutti che partecipano e condividono le proprie storie.

Questo schema di competizione rappresenta un passo molto importante nella nascita dei problemi di scheduling sportivo. Il Round Robin è stato infatti un approccio iniziale per garantire una competizione equa, permettendo a ogni squadra di sfidare ogni avversario e valutare appieno le abilità di ciascuna nel corso del torneo.

Questo formato non solo offre una distribuzione uniforme delle sfide, ma crea anche una base per determinare la squadra vincitrice senza l'eliminazione diretta. Quasi tutte le leghe sportive europee si appoggiano a tale organizzazione di calendario.

In maniera più dettagliata, considerando un intero positivo n, un torneo Round Robin per $2n$ squadre richiede $2n-1$ turni di gioco. In questo tipo di torneo, ogni squadra si scontra con tutte le altre esattamente una volta e partecipa a una partita in ogni turno. Nel caso di un campionato con un numero dispari di squadre, è possibile preservare l'equilibrio del torneo Round Robin aggiungendo una *squadra fittizia di riposo*. Questa squadra immaginaria non partecipa a nessuna partita reale, ma è introdotta per garantire che il calendario possa essere strutturato in modo coerente. Infatti ad ogni match che coinvolge tale *squadra di riposo* consisterà in una giornata di pausa della squadra avversaria. Ciò assicura che ogni squadra abbia lo stesso numero di partite effettive e che il torneo possa seguire il formato Round Robin in modo uniforme.

Esiste un calendario Round Robin per ogni numero pari di squadre?

Si. E' stato dimostrato da **Thomas Kirkman** nel 1847 che esiste un calendario all'italiana per ogni campionato con numero di squadre pari a $2n$. La dimostrazione si avvale di numerose definizioni e teoremi rigurdanti la teoria dei grafi.
La ricerca operativa e l'ottimizzazione entrano in gioco dal momento che un aumento del numero di squadre porta ad un aumento notevole delle combinazioni possibili di incontri. Ad esempio potrebbero emergere situazione in cui alcune squadre giocano più partite di altre o altre in cui ci potrebbero essere conflitti di programmazione tra gli incontri.
Inoltre, è vero che l'esistenza di un calendario round robin per qualsiasi numero pari di squadre è stata dimostrata, ma ciò non significa che la costruzione di tali calendari sia sempre intuitiva e automatica. Sono dunque necessari algoritmi appositi per garantire che ogni squadra incontri tutte le altre in modo bilanciato.

3.2 Generazione

Esistono numerosi metodi che consentono di generare una *schedule* ammissibile per un calendario Round Robin, analizziamo in seguito i più semplici.

3.2.1 Circle Method

L'idea è di disporre le prime $2n-1$ squadre su un cerchio e mettere l'ultima squadra al centro. Poi scegliere una delle squadre disposte sul cerchio, tracciare un segmento che la connette alla squadra centrale e far giocare le due squadre, inoltre si sfidano le squadre che si trovano sui segmenti ortogonali a quello tracciato. In questo modo ottengo uno slot e proseguendo allo stesso modo per ogni squadra sul bordo ottengo una *schedule* ammissibile.

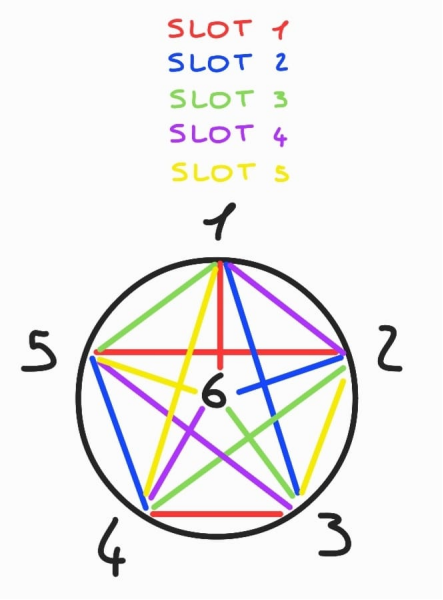


Figura 3.1: Circle Method

Più in dettaglio, posso costruire gli slot in modo che, per i che va da 1 a $2n-1$, nello slot i-esimo giocano:

- i vs $2n$
- a vs b se $a + b \equiv 2i(mod(2n-1))$

Per esempio, per un campionato a 6 squadre il Circle Method assegna le partite in questo modo:

Tabella 3.1: Circle Method

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
(1 vs 6)	(2 vs 6)	(3 vs 6)	(4 vs 6)	(5 vs 6)
(2 vs 5)	(1 vs 3)	(2 vs 4)	(3 vs 5)	(1 vs 4)
(3 vs 4)	(4 vs 5)	(1 vs 5)	(1 vs 2)	(2 vs 3)

Questo metodo fornisce una *schedule* ammissibile per qualsiasi numero pari di squadre.

3.2.2 Greedy Method

Il Greedy Method si basa sugli algoritmi *Greedy*. E' possibile implementare una moltitudine di algoritmi seguendo questo approccio, il problema è che non sempre portano ad una soluzione compatibile con quella richiesta.
Per esempio, un'idea potrebbe essere la seguente:

- Ordinare le partite in modo lessicale secondo le regole:
  - la partita (i, j) prima della partita (i, k) se $j < k$
  - la partita (i, j) prima della partita (k, l) se $i < k$
- Ordinare gli slot in modo ciclico.
- Assegnare la partita (1,2) allo slot 1.
- Seguendo l'ordine ottenuto in precedenza, assegnare le partite al primo slot libero in cui nessuna delle due squadre abbia già in programma un match.

Per un campionato a 6 squadre assegno dunque le partite in questo modo:

Tabella 3.2: Greedy Method(6 teams)

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
(1 vs 2)	(1 vs 3)	(1 vs 4)	(1 vs 5)	(1 vs 6)
(3 vs 5)	(4 vs 5)	(2 vs 3)	(2 vs 4)	(2 vs 5)
(4 vs 6)	(2 vs 6)	(5 vs 6)	(3 vs 6)	(3 vs 4)

Se aumento il numero di squadre si ottiene una *schedule* ammissibile?

No, infatti se $n = 8$ l'algoritmo genera una programmazione non completa, non riuscendo a collocare alcune le partite negli slot rimasti:

Tabella 3.3: Greedy Method(8 teams)

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7
(1 vs 2)	(1 vs 3)	(1 vs 4)	(1 vs 5)	(1 vs 6)	(1 vs 7)	(1 vs 8)
(4 vs 6)	(4 vs 7)	(2 vs 3)	(2 vs 4)	(2 vs 5)	(2 vs 6)	(2 vs 7)
(5 vs 8)	(2 vs 8)	(5 vs 6)	(6 vs 7)	(3 vs 4)	(3 vs 5)	(3 vs 6)
(3 vs 7)	?	(7 vs 8)	(3 vs 8)	?	(4 vs 8)	(4 vs 5)

Rimangono da programmare i match (5 vs 7) e (6 vs 8), ma si può notare che non è possibile inserirli in nessuno dei due posti ancora disponibili.

In conclusione, mentre gli algoritmi greedy possono fornire soluzioni rapide in alcuni casi, è evidente che la costruzione di calendari Round Robin per un numero più alto di squadre richiede un approccio più sofisticato.

3.3 Ottimizzazione

Precedentemente, abbiamo esaminato e valutato due approcci per la creazione di un calendario Round Robin, ma essi si sono concentrati esclusivamente sulla generazione di uno schema che rispetti i vinvoli della programmazione Round Robin non tenendo conto di ulteriori fattori.

Supponiamo adesso di considerare un'ulteriore variabile: ad ogni partita è associato un costo che varia in base al round in cui viene disputata. L'obiettivo principale è quello di ottenere una programmazione delle partite che non solo rispetti i vincoli di gioco specifici, ma che dimostri anche una notevole ottimizzazione dei costi totali. In altre parole, cerchiamo di minimizzare il costo complessivo delle partite distribuendo strategicamente gli incontri nei vari round, tenendo conto dei costi variabili associati a ciascuna partita. Tale approccio non solo garantisce un calendario di gioco ammissibile, ma cerca anche di raggiungere un compromesso tra le esigenze di gioco e la gestione ottimale delle risorse finanziarie.

3.3.1 Modello matematico

Supponiamo valgano le seguenti ipotesi:

- $T = \{1,2,...,n\}$ è l'insieme delle n squadre(n pari), indicizzate da i, $1 \leq i \leq n$.
- $R = \{1,2,...,n-1\}$ è l'insieme dei turni di gioco, indicizzati da r, $1 \leq r \leq n-1$.
- $M = \{1,2,...,(n/2)*(n-1)\}$ è l'insieme delle partite, indicizzate da m, $1 \leq m \leq (n/2)*(n-1)$.

- $M_i = \{1,2,...,n-1\}$ è l'insieme delle partite che deve disputare la squadra i.
- $c_{m,r}$ è il costo della partita m se disputata nel turno r.

Introduciamo poi una variabile decisionaria $x_{m,r}$ con $m \in M, r \in R$ per valutare se la partita m viene giocata nel turno r ($x_{m,r} = 1$) oppure no ($x_{m,r} = 0$).

Il problema può dunque essere modellizzato come:

$$\min \quad \sum_{m \in M} \sum_{r \in R} c_{m,r} x_{m,r} \tag{3.1}$$

$$\text{s.t.} \quad \begin{cases} \sum_{r \in R} x_{m,r} = 1 \quad m \in M \\ \sum_{m \in M_i} x_{m,r} = 1 \quad i \in T, r \in R \\ x_{m,r} \in \{0,1\} \quad m \in M, r \in R \end{cases} \tag{3.2}$$

Vincoli:

- $\sum_{r \in R} x_{m,r} = 1 \quad m \in M$: ogni squadra affronta tutte le altre squadre esattamente una volta.
- $\sum_{m \in M_i} x_{m,r} = 1 \quad i \in T, r \in R$: ogni squadra gioca esattamente una volta per turno.
- $x_{m,r} \in \{0,1\} \quad m \in M, r \in R$: la variabile decisionaria $x_{m,r}$ è binaria.

Se una partita m non si può assolutamente programmare per il round r, allora è sufficiente assegnare a $c_{m,r}$ un valore spropositatamente alto in modo che nell'ottimizzare il calendario tale opzione verrà scartata.

3.3.2 Implementazione

Dopo aver acquisito l'input del numero n di squadre, si calcola il numero di partite $M = n/2 * (n-1)$ e il numero di round $R = n-1$. Successivamente, si generano tutte le possibili coppie di squadre(cioè le partite da disputare), ciascuna rappresentata da un vettore bidimensionale contenente i numeri delle due squadre coinvolte. Queste coppie sono state memorizzate all'interno di una cella chiamata *partite*. La posizione di ciascuna partita è l'id della partita stessa

Per poter imporre il secondo vincolo si definisce una matrice $M_{squadre}$ di dimensione $n \times n-1$ in cui per ogni riga $i-esima$ fornisce l'id delle partite in cui la squadra i è coinvolta. Per esempio, se nella prima riga ci sono i numeri 1, 2 e 3, ciò significa che la squadra numero 1 è coinvolta nelle prime tre partite della cella *partite*.

I costi $c_{m,r}$ sono stati generati casualmente grazie al comando *randi()* di $MATLAB$ e inseriti in una matrice C di dimensione $n/2 * (n-1) \times (n-1)$ in modo che nella posizione (m, r) ci sia il costo che ha la partita m se giocata il round r.

La variabile decisionale $x_{m,r}$ è rappresentata da una matrice X di dimensione $n/2 * (n-1) \times (n-1)$ il cui elemento (m, r) vale 1 se la partita m viene giocata il turno r, 0 altrimenti.

Per stabilire il primo vincolo, è sufficiente richiedere che la somma degli elementi su ciascuna riga della matrice X sia pari a 1. Questo vincolo assicura che ogni partita venga disputata esattamente una volta, garantendo così che ogni squadra incontri tutte le altre esattamente una volta.

Per quanto riguarda il secondo vincolo è necessario, come anticipato, l'utilizzo della matrice $X_{squadre}$. Infatti bisogna imporre che per ogni team i,$1 \leq i \leq n$, la somma dei valori su ogni colonna di X, considerando solo quelli che sono situati sulle righe che corrispondono ai valori contenuti nella $i - esima$ riga della matrice $X_{squadre}$, sia pari a 1.
Per esempio, supponendo che ci siano $n = 4$ squadre, allora la cella *partite* sarà:
$partite = \{[1,2], [1,3], [1,4], [2,3], [2,4], [3,4]\}$. La matrice $X_{squadre}$ sarà invece:

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 6 \\ 2 & 4 & 6 \\ 3 & 5 & 6 \end{bmatrix}$$

Il secondo vincolo dunque, per la squadra 1, obbliga ad avere, per ogni colonna di X, la somma dei valori delle righe 1, 2 e 3 pari a 1.
Per la squadra due vale la stessa cosa ma si deve considerare le righe 1, 4 e 6. Questo deve valere per ogni squadra.

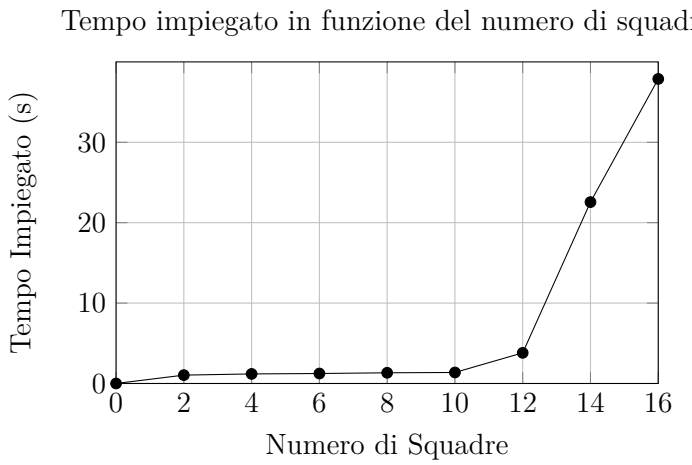
3.3.3 Risultati e confronti

Il problema appena presentato è stato risolto grazie al "Optimization Toolbox" di base fornito da $MATLAB$. Il solver specifico utilizzato all'interno del "Optimization Toolbox" è 'intlinprog', che è progettato per risolvere problemi di programmazione lineare intera mista.

Il codice produce risultati ottimali in tempi ragionevoli per un numero di squadre inferiore a 10. Tuttavia, si verifica una leggera diminuzione delle prestazioni quando si lavora con campionati di dimensioni comprese tra 12 e 16 squadre. Per numeri di squadre superiori a 16, il programma diventa troppo lento e talvolta non riesce a trovare una

soluzione.
In sintesi, il codice è efficiente per tornei di piccole dimensioni, ma tende a diventare meno efficiente man mano che il numero di squadre aumenta. Potrebbe essere utile esplorare ulteriori strategie di ottimizzazione o considerare l'uso di algoritmi più avanzati per affrontare il problema in modo efficiente anche per competizioni di dimensioni maggiori.

Osserviamo un grafico che rappresenta il tempo impiegato dal programma per fornire una soluzione in funzione del numero di squadre n richiesto:



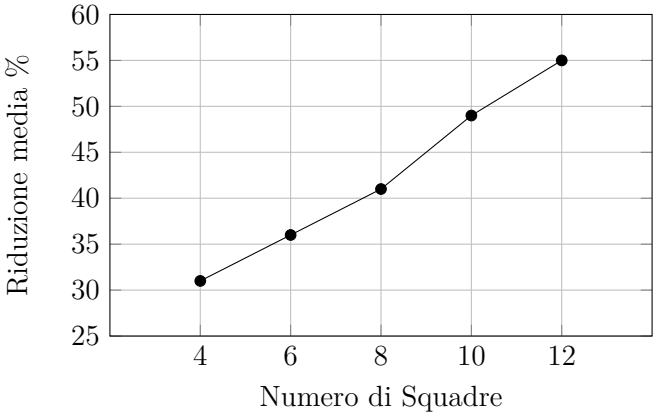
Valutiamo grazie a questa tabella quanto tale approccio ci consente di risparmiare rispetto al calendario generato dal *Circle Method*:

Tabella 3.4: Calendario non ottimizzato vs calendario ottimizzato

Numero di squadre	Costo medio non ottimizzato	Costo medio ottimizzato	Riduzione media (%)
4	32,83	22,66	31
6	85,66	55,16	36
8	141.5	83.5	41
10	250,66	126,66	49
12	355,66	158,5	55

È evidente che in tutte le situazioni si ottiene una notevole riduzione del costo totale del campionato, e questa riduzione aumenta all'aumentare del numero di squadre coinvolte. E' confermato che quanto più numeroso è il gruppo di squadre partecipanti, tanto maggiore è l'importanza di ottimizzare il calendario per evitare spese eccessive.

Riduzione percentuale del costo in base al numero di squadre



Capitolo 4

Calendario NBA

4.1 Introduzione

Molti campionati sportivi non si avvalgono dei calendari Round Robin, preferendo invece adottare formati che sono frutto di una ponderata valutazione di vari fattori. Questi fattori includono le peculiarità dell'organizzazione sportiva, le preferenze e le esigenze dei partecipanti e del pubblico, nonché fattori di carattere pratico e logistico.

Fondata nel 1946, la *National Basketball Association* (NBA) rappresenta la massima espressione del basket professionistico negli Stati Uniti. La sua reputazione di essere la migliore lega di basket al mondo è incontestabile, tanto che costituisce il sogno più ambizioso di ogni atleta professionista.

La NBA è composta da 30 squadre suddivise in due *conference* formate da tre *division* con cinque squadre ciascuna.

Tabella 4.1: Conference, Division e Squadre NBA

Conference	**Divisione**	**Squadre NBA**
Eastern	Atlantic	Boston Celtics Brooklyn Nets New York Knicks Philadelphia 76ers Toronto Raptors
	Central	Chicago Bulls Cleveland Cavaliers Detroit Pistons Indiana Pacers Milwaukee Bucks
	Southeast	Atlanta Hawks Charlotte Hornets Miami Heat Orlando Magic Washington Wizards
Western	Northwest	Denver Nuggets Minnesota Timberwolves Oklahoma City Thunder Portland Trail Blazers Utah Jazz
	Pacific	Golden State Warriors LA Clippers Los Angeles Lakers Phoenix Suns Sacramento Kings
	Southwest	Dallas Mavericks Houston Rockets Memphis Grizzlies New Orleans Pelicans San Antonio Spurs

L'ottimizzazione del calendario NBA viene chiamata *NBA Scheduling Problem* e si può indicare con la sigla **NBASP**.

4.2 Vincoli e obiettivo

Il campionato della stagione regolare NBA consiste in 1230 partite in circa 170 giorni con una media di oltre 7 partite ogni giorno. Ogni squadra gioca 82 partite in totale(41 in casa e 41 in trasferta), andando a sfidare:

- 4 volte le squadre della stessa *division* (16 partite)
- 4 volte sei delle squadre delle altre due *division* della stessa *conference* (24 partite)
- 3 volte le quattro rimanenti della stessa *conference* (12 partite)
- 2 volte le squadre dell'altra *conference* (30 partite)

Al termine della stagione regolare si passa alle fasi finali, i playoff, in cui viene determinata la squadra campione attraverso un torneo ad eliminazione diretta che vede partecipare le migliori 16 squadre del campionato concluso. Da un punto di vista di ottimizzazione le fasi finali non ci interessano in quanto si tratta di partite che non si conoscono a priori.
Il programma della stagione regolare, al contrario, viene elaborato prima dell'inizio della stagione e deve rispettare una serie significativa di restrizioni. Tenendo presente che ogni squadra deve giocare 82 partite in meno di 170 giorni è importante proteggere gli atleti dagli infortuni cercando di ridurre il loro sforzo, per questo sono stati imposti i vincoli per cui ogni squadra:

- gioca al massimo una partita al giorno.
- gioca al massimo due partite in due giorni consecutivi(back to back)
- gioca al massimo cinque partite in otto giorni consecutivi.
- gioca al massimo diciotto back to back(due partite in due giorni consecutivi) durante la stagione.

Inoltre, è vantaggioso sia dal punto di vista dei costi che per preservare la salute mentale e fisica degli atleti limitare il più possibile le frequenti trasferte che implicano lunghi viaggi e potenziali effetti del jet lag.
L'obiettivo è dunque quello di ottenere una pianificazione accettabile, ossia in linea con i vincoli imposti, mentre si cerca di ridurre al minimo la somma dei viaggi di tutte le squadre coinvolte.

4.3 Modello matematico dell'NBASP

Il modello matematico dell'NBASP assume le seguenti ipotesi:

- $T = \{1,2,...,n\}$ è l'insieme delle n squadre, indicizzate da i, $1 \leq i \leq n$.
- $\Lambda_i = (\lambda_{i,1}, \lambda_{i,2})$ sono le coordinate dello stadio della squadra i.

- $\delta_{i,i'}$ è la distanza in kilmoetri delle arene della squadra i e della squadra i'.
- $S = \{1,2,...,p\}$ è l'insieme dei giorni disponibili per giocare, indicizzati da j, $1 \leq j \leq p$.
- $w_j \in \{1,2,...,7\}$ indica il giorno della settimana, 1 per lunedì, 2 per martedì, ..., 7 per domenica, a cui corrisponde il giorno j del calendario. Le partite giocate il giorno j, con $w_j \in \{5,6,7\}$ sono *weekend games* e l'interesse di ogni squadra è quello di disputarne il maggior numero possibile. Giocare durante i fine settimana infatti attira un pubblico più numeroso sia nei palazzetti che davanti alla televisione, portando a una maggiore esposizione e guadagni per la lega e le squadre.
- $S' = \{s_1, s_2, ..., s_q\}$ con $s'_j \in S$ sono i giorni in cui non è in programma nessuna partita, come per esempio la Vigilia di Natale.
- m è il numero di partite giocate da ogni squadra.
- $u = 1/2 * m * n$ è il numero totale di partite giocate in stagione.
- $G = \{1,2,...,u\}$ è l'insieme delle partite, indicizzate da k, $1 \leq k \leq u$.
- r_k e h_k sono rispettivamente la squadra in casa e la squadra in trasferta coinvolte nella partita k

Ad oggi la NBA ha $n = 30$ squadre che giocano $m = 82$ partite, per un totale di $u = 1/2 * m * n = 1230$ match.
Risolvere il problema significa assegnare ad ogni partita $k \in G$ un giorno $d_k \in S - S'$ in modo che se ci sono più partite lo stesso giorno, allora tutte le squadre che giocano in quel giorno sono diverse.
Sia ora $X_i = \{x_{i,1}, x_{i,2}, ..., x_{i,p}\}$ la *schedule* ottenuta per la squadra i, dove $x_{i,j} \in \{0\} \cup G$. L'elemento $x_{i,j}$ indica la partita giocata nel giorno j se $x_{i,j} \in G$, altrimenti indica che non c'è nessuna partita in calendario per la squadra i nel giorno j. Inoltre se $x_{i,j}, x_{i,j'} \cup G$ significa che il team i gioca nei giorni j e j', e le partite devono essere diverse, cioè $x_{i,j} \neq x_{i,j'}$.
Quando il giorno della partita d_k è stato determinato per ogni match k, allora si può ottenere la *schedule* per le squadre h_k e r_k, cioè $x_{r_k,d_k} = k$ e $x_{h_k,d_k} = k$.

Per il calcolo del viaggio complessivo di ciascuna squadra, bisogna considerare che ognuna di esse parte e torna al proprio stadio all'inizio e alla fine del campionato. Si può ottenere la distanza l_i percorsa dal team i in base alla sua *pianificazione* X_i mediante la seguente operazione in pseudocodice:

Algorithm 1: CalcoloViaggio(X_i)

```
l_i = 0, i' = i
for j = 1 : p do
    k = x_{i,j};
    if k ≠ 0 then
        i'' = h_k
        l_i = l_i + δ_{i',i''}
        i' = i''
l_i = l_i + δ_{i',i}
return l_i;
```

Grazie all'operazione $count(a)$, che conta quante volte l'argomento a è vero, si possono formalizzare i vincoli in forma matematica:

- *massimo due partite in tre giorni:*
  $\pi_i = count_{j=1,2,...,p-2}(x_{i,j}, x_{i,j+1}, x_{i,j+2} \neq 0) = 0 \quad \forall i = 1, ..., n.$
- *massimo cinque partite in otto giorni:*
  $\rho_i = count_{j=1,2,...,p-7}(count_{j'=j,j+1,...,j+7}(x_{i,j'} \neq 0) > 5) = 0 \quad \forall i = 1, ..., n.$
- *massimo diciotto back to back:*
  $\sigma_i = count_{j=1,2,...,p-1}(x_{i,j}, x_{i,j+1} \neq 0) \leq 18 \quad \forall i = 1, ..., n.$

In definitiva, il problema di ottimizzazione può essere formulato come:

$$\min \quad L = \sum_{i=1}^{n} l_i \tag{4.1}$$

$$\text{s.t.} \quad \begin{cases} \pi_i = 0 \quad \forall i = 1, ..., n \\ \rho_i = 0 \quad \forall i = 1, ..., n \\ \sigma_i \leq 18 \quad \forall i = 1, ..., n \end{cases} \tag{4.2}$$

4.4 Struttura dati e implementazione

Dopo aver esaminato dettagliatamente il modello matematico nella sezione precedente, ci concentreremo ora sull'aspetto pratico dell'implementazione. Vediamo adesso come le equazioni e i concetti teorici sono stati trasformati in un'applicazione concreta utilizzando $MATLAB$.

Per prima cosa le squadre sono state organizzate seguendo l'ordine alfabetico e sono state associate a indici basati sulla loro posizione in questa sequenza:

- **1:** Atlanta Hawks,
- **2:** Boston Celtics,
- ...
- **30:** Washington Wizards

Sono stati inizializzati poi il numero di squadre $n = 30$, il numero di giorni disponibili $p = 165$ e il numero di partite totali $u = 1230$.

Le distanze $\delta_{i',i''}$ tra lo stadio della squadra i' e quello della squadra i'' sono state inserite in una matrice di dimensione $n \times n$, dove $\delta_{i',i''}$ è posizionato nella cella (i', i'').

Per gestire le partite giocate ogni giorno, è stato definito un dizionario chiamato $match_{days}$. In questo dizionario, le chiavi rappresentano i giorni di gioco, mentre i valori associati sono matrici in cui ogni riga rappresenta una partita giocata in quel giorno. Ad esempio, se associamo alla chiave $key = 1$ la matrice:

$$\begin{bmatrix} 17 & 3 \\ 14 & 10 \end{bmatrix}$$

questo indica che il primo giorno vengono disputate le partite {team 17 vs team 3} e {team 14 vs team 10}, rispettivamente giocate in casa del team 17 e del team 14.
Alle chiavi che rappresentano i giorni senza alcuna partita in programma corrisponde un vettore bidimensionale di zeri.

A partire dal dizionario $match_{days}$ viene costruita la cella *schedule* di dimensione $n \times p$. Questa cella contiene in posizione (i, j) il vettore bidimensionale che rappresenta la partita disputata dalla squadra i il giorno j. Se la squadra i non ha in programma partite il giorno j allora la cella conterrà un vettore bidimensionale di zeri.
Ad esempio, supponiamo che la prima riga della cella sia {[0,0], [1,7], [0,0], [6,1], ...}. Ciò significa che il team 1 è a riposo il giorno 1 e il giorno 3, gioca in casa contro il team 7 il secondo giorno e sfida fuori casa il team 6 il quarto giorno ecc.

Il percorso di ciascuna squadra è determinato attraverso l'esecuzione dell'algoritmo illustrato nell'Algoritmo 1, implementato in ambiente $MATLAB$. I risultati vengono successivamente raccolti e memorizzati all'interno del vettore *travel*, il quale ha dimensione $1 \times n$. In questo vettore, l'elemento in posizione i rappresenta la distanza complessiva percorsa dalla squadra i durante l'intera stagione.

Grazie a questa struttura dati risulta semplice verificare che i vincoli siano rispettati:

- *massimo due partite in tre giorni*:
  è sufficiente scorrere la cella *schedule* e verificare che ci sia almeno un elemento che coincida col vettore $[0, 0]$ per ogni tre elementi consecutivi.

- *massimo cinque partite in otto giorni*:
  è sufficiente scorrere la cella *schedule* e verificare che ci siano almeno tre elementi coincidenti con il vettore $[0, 0]$ per ogni otto elementi consecutivi.

- *massimo diciotto back to back:*
  è sufficiente scorrere la cella *schedule* e contare quante volte sono presenti due elementi consecutivi diversi da $[0, 0]$, tale valore deve essere minore o uguale a 18.

A questo punto, l'obiettivo è creare un dizionario $match_{days}$, che rappresenta l'intero calendario della stagione sportiva, cercando di minimizzare la somma dei valori presenti nel vettore *travel*. Nel fare questo è importante verificare che i vincoli sopra elencati vengano rispettati.

4.5 Applicazione dell'Hill Climbing Algorithm

Come già discusso, l'Hill Climbing Algorithm consiste nell'apportare delle modifiche al calendario, garantendo sempre la sua validità, e successivamente valutare se tali modifiche hanno condotto ad un miglioramento o meno.
Di seguito esaminiamo come questo principio può essere applicato nel nostro contesto per raggiungere soluzioni ottimali proponendo due approcci diversi.

4.5.1 Spostamento del giorno di una partita

Ripetere un numero prestabilito di volte i seguenti passaggi:

1. Partire da una soluzione iniziale.

2. Estrarre a sorte un giorno day_1 tra quelli in cui c'è almeno una partita in programma.

3. Estarre a sorte una partita *game* tra quelle che devono essere disputate quel giorno. La squadra in casa sarà $team_1$, quella in trasferta $team_2$.

4. Estrarre a sorte un giorno day_2 tra quelli in cui $team_1$ e $team_2$ sono entrambi a riposo(non sono validi quelli in cui non c'è nessuna partita in programma).

5. Spostare la partita *game* dal giorno day_1 al giorno day_2 andando a modificare la cella *schedule*.

6. Verificare che i vincoli sul calendario siano rispettati, se non lo sono ripristinare la soluzione iniziale e ritornare al punto 1). Altrimenti proseguire al prossimo punto.

7. Calcolare la nuova distanza percorsa complessiva tramite l'algoritmo 1.

8. Se la nuova distanza è inferiore a quella della soluzione iniziale, procedere all'iterazione successiva; in caso contrario, ripristinare la soluzione precedente.

Seguendo questo procedimento, siamo in grado di esplorare un ampio numero di possibilità nella pianificazione delle partite. Ciò grazie allo scambio dei giorni in cui le partite estratte a sorte sono programmate, consentendo di ottenere una buona soluzione. Tuttavia, ciò comporta un'elevata complessità computazionale poiché è molto complicato trovare giorni disponibili per le squadre senza compromettere la validità della programmazione. Di conseguenza, su un gran numero di iterazioni, solo una frazione ristretta porterà a una pianificazione valida e inoltre un numero ancora più limitato di iterazioni potrebbe effettivamente ridurre la distanza complessiva percorsa dalle squadre

4.5.2 Scambio tra squadra in casa e squadra in trasferta

Ripetere un numero prestabilito di volte i seguenti passaggi:

1. Partire da una soluzione iniziale.
2. Estrarre a sorte un giorno *day* tra quelli in cui c'è almeno una partita in programma.
3. Estarre a sorte una partita $game_1$ tra quelle che devono essere disputate quel giorno.
4. Cercare nella cella *schedule* una partita $game_2$ che vede disputare le stesse squadre di $game_1$, ma in posizione invertita.
5. Scambiare la squadra in casa e quella in trasferta sia in $game_1$ che in $game_2$.
6. Modificare il dizionario $match_{days}$ e la cella *schedule* in seguito a tale modifica.
7. Calcolare la nuova distanza percorsa complessiva tramite l'algoritmo 1.
8. Se la nuova distanza è inferiore a quella della soluzione iniziale, procedere all'iterazione successiva; in caso contrario, ripristinare la soluzione precedente.

Vengono modificate entrambe le partite $game_1$ e $game_2$ in modo da mantenere fisso il numero di partite che ogni squadra gioca in casa(41) e in trasferta(41).
Questo approccio è interessante perché permette di ottenere una schedule che non va a modificare le date stabilite dalla soluzione iniziale. Invece di spostare le partite, si modificano semplicemente le squadre che giocano in casa e in trasferta. Questo può essere un vantaggio significativo, specialmente se le dirette televisive o altri impegni correlati alle partite sono già stati pianificati. Tuttavia, è importante notare che questo metodo esplora una piccola parte delle possibili soluzioni e quasi certamente non fornirà una delle migliori pianificazioni possibili.

4.6 Risultati e confronti

Per verificare l'efficienza degli algoritmi proposti sono stati acquisiti e utilizzati come dati di partenza le schedule delle stagioni NBA più recenti. I dati sono stati scaricati in formato excel dal sito Basketball-Reference.com e includono le date, gli orari e le squadre coinvolte nelle partite pianificate per le stagioni 2021-2022, 2022-2023, e 2023-2024.

Di seguito, presentiamo alcune informazioni riguardanti le soluzioni ottenute. Ciò permette di valutare l'efficacia dei due diversi approcci.

4.6.1 Spostamento del giorno di una partita

La seguente tabella mostra i risultati ottenuti a partire da tre diverse soluzioni iniziali.

Tabella 4.2: Risultati ottenuti con 100000 iterazioni

Stagione NBA	Distanza complessiva iniziale (km)	Distanza complessiva ottimizzata (km)	Riduzione percentuale (%)	Tempo di esecuzione(s)
2021-2022	2793744	2559400	8.39	29.81
2022-2023	2658720	2402056	9.65	38.04
2023-2024	2611448	2362464	9.53	35.12

Il tempo di esecuzione si aggira intorno ai 30 secondi e ci permette di ottenere ottimi risultati. Si può notare che in tutti e tre i casi la riduzione percentuale della distanza complessiva percorsa si avvicina al 10%.

Considerando l'importanza del numero di iterazioni, è legittimo chiedersi se sia possibile migliorare ulteriormente i risultati regolando la quantità di iterazioni impiegate. Per valutare questa possibilità, presentiamo ora i risultati ottenuti con differenti quantità di iterazioni.

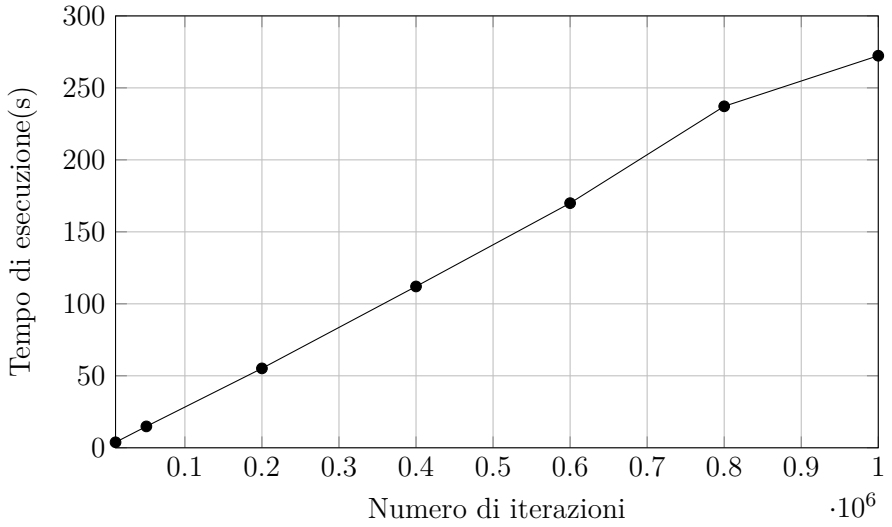
Tabella 4.3: Risultati ottenuti con 10000 iterazioni

Stagione NBA	Distanza complessiva iniziale (km)	Distanza complessiva ottimizzata (km)	Riduzione percentuale (%)	Tempo di esecuzione(s)
2021-2022	2793744	2736792	2.04	3.42
2022-2023	2658720	2595904	2.36	3.59
2023-2024	2611448	2565816	1.75	3.30

Tabella 4.4: Risultati ottenuti con 1000000 iterazioni

Stagione NBA	Distanza complessiva iniziale (km)	Distanza complessiva ottimizzata (km)	Riduzione percentuale (%)	Tempo di esecuzione(s)
2021-2022	2793744	2135392	23.57	321.14
2022-2023	2658720	2032024	23.57	261.60
2023-2024	2611448	2116816	18.94	248.37

Basandoci su questi risultati si può notare che il tempo di esecuzione cresce quasi linearmente al crescere del numero di iterazioni. Ciò è confermato anche da questo grafico:



E' necessario ora fare alcune valutazioni riguardo la scelta del numero di iterazioni. Prima però bisogna specificare che ciò dipende soprattutto dal proprio obbiettivo. Se per esempio volessimo trovare la miglior *schedule* possibile, e siamo disposti a dedicare anche diversi giorni a questo scopo, allora sicuramente bisognerà fare altre considerazioni.
Ai fini della tesi, però, possiamo immaginare di voler applicare l'algoritmo più volte, e dunque l'obbiettivo è ottenere un buon compromesso tra tempo di esecuzione e riduzione percentuale della distanza percorsa. Osservando i risultati e considerando la crescita lineare del tempo di esecuzione si può concludere che per un numero di iterazioni compreso tra 100000 e 1000000 si ottengono esiti positivi. Escludiamo numeri di iterazione minori di 100000 perchè, sepppur in un tempo molto breve, forniscono una percentuale troppo bassa di riduzione delle distanze percorse.

4.6.2 Scambio tra squadra in casa e squadra in trasferta

Presentiamo anche per questo approccio informazioni sui dati ottenuti a partire dalle tre diverse soluzioni iniziali con diversi numeri di iterazione.

Tabella 4.5: Risultati ottenuti con 1000 iterazioni

Stagione NBA	Distanza complessiva iniziale (km)	Distanza complessiva ottimizzata (km)	Riduzione percentuale (%)	Tempo di esecuzione(s)
2021-2022	2793744	2597472	7.03	0.40
2022-2023	2658720	2519704	5.23	0.40
2023-2024	2611448	2455096	5.99	0.39

Tabella 4.6: Risultati ottenuti con 5000 iterazioni

Stagione NBA	Distanza complessiva iniziale (km)	Distanza complessiva ottimizzata (km)	Riduzione percentuale (%)	Tempo di esecuzione(s)
2021-2022	2793744	2474896	11.41	1.41
2022-2023	2658720	2398064	9.80	1.45
2023-2024	2611448	2335880	10.55	1.39

Tabella 4.7: Risultati ottenuti con 10000 iterazioni

Stagione NBA	Distanza complessiva iniziale (km)	Distanza complessiva ottimizzata (km)	Riduzione percentuale (%)	Tempo di esecuzione(s)
2021-2022	2793744	2423440	13.25	2.98
2022-2023	2658720	2365768	11.02	2.68
2023-2024	2611448	2321776	11.09	2.72

Evidentemente, si può notare un notevole successo con un numero limitato di iterazioni, ma quando si aumenta significativamente quest'ultimo, i miglioramenti nella soluzione diventano meno sostanziosi o addirittura assenti. Questo fenomeno era prevedibile, poiché l'algoritmo esplora solo una piccola porzione delle possibili soluzioni e, una volta raggiunto un risultato che sembra ottimo tra quelle opzioni esplorate, tende a bloccarsi, senza garantire ulteriori miglioramenti nella pianificazione.

Si può osservare nella seguente tabella che anche aumentando a 100000 il numero di iterazioni si ottiene una riduzione sostanzialmente uguale a quella ottenuta con 10000.

Tabella 4.8: Risultati ottenuti con 100000 iterazioni

Stagione NBA	Distanza complessiva iniziale (km)	Distanza complessiva ottimizzata (km)	Riduzione percentuale (%)	Tempo di esecuzione(s)
2021-2022	2793744	2421072	13.34	32.11
2022-2023	2658720	2365192	11.04	27.29
2023-2024	2611448	2314720	11.36	35.46

4.6.3 Confronti

Ora procediamo a un confronto diretto tra i due algoritmi, esaminando vari aspetti significativi:

- **Qualità delle soluzioni:** Il primo algoritmo seppur più lento, tende a fornire soluzioni ottimizzate con una riduzione percentuale della distanza complessiva superiore rispetto a quella ottenuta dal secondo algoritmo.
- **Robustezza:** Il primo algoritmo è più resistente, infatti continua a migliorare la soluzione all'aumentare delle iterazioni. Il secondo invece fornisce risultati migliori su numeri bassi di iterazioni, ma tende a stabilizzarsi dopo un certo punto.
- **Tempo di esecuzione:** Entrambi gli algoritmi dimostrano di avere velocità computazionali molto simili quando vengono eseguiti con lo stesso numero di iterazioni. La distinzione principale tra di essi si evidenzia nella qualità delle soluzioni ottenute.

Si può concludere che se l'obiettivo principale è ottenere una risposta immediata senza apportare modifiche significative al calendario, il secondo algoritmo è la scelta più solida perchè è in grado di fornire soluzioni di buona qualità in tempi rapidi.
D'altra parte, se la priorità è massimizzare la qualità della *schedule*, sacrificando eventualmente una maggiore rapidità nell'ottenimento dei risultati, il primo algoritmo risulta essere la scelta preferibile, dimostrando una maggiore capacità di ottenere soluzioni ottimali o molto vicine all'ottimalità.

Capitolo 5

Conclusioni

5.1 Calendari Round Robin

Le considerazioni presentate sui calendari Roun Robin dimostrano che il *Circle Method* è il più affidabile tra i due algoritmi proposti. Questo perchè fornisce sempre una soluzione valida, anche all'aumentare del numero di squadre, al contrario del *Greedy Method*.

Per quanto riguarda la loro ottimizzazione, è importante sottolineare che esistono molteplici modelli matematici a cui ci si può appoggiare, ognuno mirato a raggiungere specifici obiettivi. Se per esempio il nostro obbiettivo fosse stato quello di distribuire in turni diversi i *big match*, ovvero le partite che teoricamente portano maggiore ricavo, in modo da garntire una spettacolarità costante di giornata in giornata, sicuramente avremmo utilizzato altre modalità di implementazione.

Il modello presentato fornisce ottimi risultati che dimostrano come l'adozione di tali approcci nell'organizzazione di tornei e campionati sia fondamentale, contribuendo a promuovere una gestione più sostenibile delle risorse finanziarie nel mondo dello sport.

5.2 Calendario NBA

Le modalità di ottimizzazione proposte per l'ottimizzazione del calendario NBA hanno conseguito risultati notevoli, riducendo la distanza totale percorsa di oltre il 20%. Considerando che stiamo parlando di oltre 2 milioni di chilometri, questo rappresenta un significativo miglioramento.

Tuttavia, è importante notare che l'algoritmo che effettua spostamenti dei giorni delle partite avrebbe potuto ottenere una soluzione ancora migliore se avessimo avuto la possibilità di aumentare ulteriormente il numero di iterazioni. Purtroppo, ciò non è stato fattibile a causa del tempo di esecuzione troppo elevato.

Va sottolineato che il problema affrontato è stato semplificato rispetto alla complessità del problema reale. L'NBASP non riguarda solamente i vincoli sulle partite consecutive e la riduzione della distanza totale percorsa, ma considera anche numerosi altri fattori. Questi includono la distribuzione equa delle partite nei weekend per ogni squadra, la disponibilità degli stadi e la massimizzazione del profitto.

Feng-Cheng Yang. *NBA Sports Game Scheduling Problem and GA-based Solver*. 2017

Zhiying Yao, Hongyi Ding. *The NBA Scheduling Problem*. 2017

Michael Trick. *Tutorial on Scheduling Sports Tournaments*. 2004

Jasper van Doornmalen, Christopher Hojny*, Roel Lambers, and Frits C.R. Spieksma. *Integer Programming Models for Round Robin Tournaments*

P. Brandimarte. *Ottimizzazione per la ricerca operativa*. CLUT, 2022.

Wikipedia. Problemi di ottimizzazione. Wikipedia, l'enciclopedia libera, 2020.