ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

EleKtion: una Libreria Kotlin Multiplatform per la Democrazia Digitale in Nuovi Contesti

Tesi di laurea in:
Laboratorio Di Sistemi Software

Relatore
Prof. Danilo Pianini

Candidato
Jacopo Corina

III Sessione di Laurea
Anno Accademico 2022-2023

Sommario

Max 2000 characters, strict.

Ai miei genitori

Indice

Elenco delle figure

Lista degli estratti di codice

Capitolo 1

Introduzione

Capitolo 2

Democrazia digitale

Il termine *Democrazia*, derivante dal greco *governo del popolo*, indica un forma di governo nella quale il potere appartiene al popolo ed è esercitato attraverso forme dirette, che coinvolgono attivamente i cittadini, e attraverso forme rappresentantive, che prevedono un sistema di rappresentanza, la quale viene rinnovata ciclicamente attraverso apposite elezioni.

In una qualunque forma di *democrazia*, sono fondamentali le basi dei principi di elezioni libere in cui i cittadini sono considerati equi, partecipazione attiva dei cittadini alla vita politica, protezione dei diritti fondamentali e della libertà personale attraverso costituzioni [**?**].

Nelle epoche successive alle prime forme di democrazia "unanime" (per quanto il concetto di cittadinanza fosse assai diverso da quello attuale, ad esempio, erano escluse donne, schiavi e minori di 20 anni), complice anche la complessità delle materie da amministrare e il numero dei partecipanti fisici, la democrazia rappresentativa ha preso la maggior diffusione.

La modernizzazione tecnologica, accelerata di recente con l'avvento della pandemia da Covid-19 che ha evidenziato notevole problemi logistici, tra cui la gestione del distanziamento [**?**] e ha aperto un ventaglio di possibilità tali da permettere un ritorno ad un controllo più diretto da parte della collettività.

La *democrazia digitale* (o *e-democracy*) è definibile come "l'esercizio di processi

decisionali democratici attraverso l'uso di informazioni e tecnologie abilitanti alla comunicazione, in particolare Internet"[**?**].

Queste tecnologie, abilitanti a rompere i vincoli geo-fisici e territoriali, permettono di facilitare l'esercizio di ulteriori forme di democrazie, come quella partecipata, in cui i cittadini, piuttosto che sostituirsi ai rappresentanti, forniscono idee sullo sviluppo dell'indirizzo di governo permettendo così di creare un collettore di confronto ed analisi di situazioni differenti.

Tuttavia, è necessario considerare che un uso eccessivamente estensivo di questi strumenti può portare al rischi di coinvolgere persone che non hanno abbastanza competenze relativamente a temi delicati, ad esempio le politiche sociali.

Inoltre, nonostante la virtualizzazione della partecipazione permette alle fasce più disagiate di essere maggiormente coinvolte, introduce un motivo di scetticismo relativamente alla componente tecnologica in ambito di sicurezza informatica, usabilità, segretezza e manipolazione del voto [**?**].

Stabilire un legame di fiducia con queste tecnologie, normarne la verificabilità, stabilire principi di *accountability*, sono i fattore principali e quindi una precondizioni essenziali per la loro diffusione. Alcuni autori ([**?**]) sostengono che è impossibile avere leggitimità in quanto "la natura dei computer è tale che il loro funzionamento interno è segreto. Poiché le transazioni e i calcoli avvengono a livello elettronico, non è fisicamente possibile per gli esseri umani osservare esattamente ciò che un computer sta facendo."

L'applicazione più comune di queste tecnologie emergenti è legata al mondo della politica[**?**]: sono degni di nota casi come l'Estonia, che dal 2005 rende possibili via Internet le votazioni per elezioni e referendum, e l'Islanda che tramite un ambizioso progetto tenta di riscrivere la costituzione con l'apporto dei cittadini. In Italia possiamo rilevare l'applicativo PartecipaMi, organizzata per mettere in contatto la popolazione milanese con l'amministrazione comunale.[**?**]

Inoltre, queste soluzioni sono spesso adottate da partiti o movimenti politici per avviare iniziative e votazioni interne tra i membri, come nel caso del Partito Pirata tedesco in Germania (piattaforma LiquidFeedback), o il Movimento 5 Stelle

in Italia (piattaforma Rousseau). Lo scopo principale è quello di promuovere la discussione di proposte tra i membri di un organizzazione, le quali saranno portate in sede legislativa da un loro rappresentante[**?**].

Sistemi come LiquidFeedback, OpenDCN, Airesis, essendo strutturati come applicativi end-to-end, pongo molta enfasi sull'interazione tra utenti, la presentazione e validazione di proposte, l'eventuale presenza di deleghe, l'espressione del voto tramite una lista di preferenze, ma non permettono la variazione delle logiche di selezione del vincitore da applicare[**?**].

In altri, una fase di analisi iniziale poco trasversale ha portato a dar maggior risalto ad aspetti non funzionali, come la privacy, rispetto a quelli funzionali come il modello di democrazia da applicare, che per alcuni aspetti rimane implicito[**?**]. Ciò è dovuto anche al fatto che il maggior coinvolgimento si concentra principalmente alle fasi iniziali e finali del ciclo di votazione [**?**]. La mancanza di elementi formalizzanti, che descrivono dettagliatamente tutti gli step applicati nel processo, unita alle variazioni ed evoluzioni che vengono apportate nel tempo, portano alla divergenza di *logica del flusso* rispetto all'idea iniziale, che conseguentemente non risulterà chiara agli utilizzatori e potenzialmente potrebbe scoraggiare l'utilizzo della piattaforma[**?**].

Al di là dell'applicazione di e-democracy nel mondo della politica, che riscontra ostacoli più legati all'aspetto organizzativo e umano, che all'aspetto tecnico, vi sono esempi notabili anche nel campo della digitalizzazione pervasiva, ad esempio nell'ambito delle *smart cities*. Ricollegandosi al fenomeno dei cambiamenti climatici che stanno inesorabilmente continuando a proliferare, in certe zone del mondo l'acqua potabile sta diventando un bene sempre meno reperibile e se sempre più da salvaguardare. [**?**] pone il focus sull'adozione di sistemi di "IdroInformatica". Questo tipo di sistemi permettono di analizzare e monitorare, attraverso sensori e sistemi real-time `SCADA` (Supervisory Control And Data Acquisition), gli attuali flussi e pressioni nella rete di distribuzione ed effettuare predizioni sui futuri guasti. Questi strumenti di analisi possono fornire ai cittadini informazioni su come ottimizzare l'utilizzo delle risorse. L'unione di questi sistemi, ad una politica di

gestione elettronica e democratica che porti a prendere parte alle decisioni per i cambiamenti che coinvolgono il regime di gestione presente in una determinata area, permetterebbe una miglior localizzazione circa le soluzioni da intraprendere, ed ai cittadini una maggior consapevolezza, in quanto sono i principali *stakeholders* e a loro viene attribuito anche un principio di responsabilità. In altri Stati, come Canada e Paesi Bassi, la gestione dell'acqua è delegata ad un gestione locale, che promuove la collaborazione tra soggetti pubblici e privati.

Questi nuovi approcci alla gestione democratica degli interessi dei cittadini aprono nuovi scenari di trasparenza e fiducia reciproca, permettendo un maggior capacità espressione di espressione verso i rappresentanti ma anche da parte di chi è ora in grado di percepire, in ottica di miglioramento continuo, le esigenze del mondo "reale". L'intermediazione, applicata a questi nuovi contesti, continua a rimanere essenziale. Piuttosto che diventare un nuovo "medium" per imporre decisioni, risulta più efficace per favorire lo scambio costruttivo, mantenendo comunque le opportune autonomie riguardo alle decisioni finali, tutelando gli interessi collettivi e al contempo educando le persone a mantenere la libertà di pensiero[**?**].

Capitolo 3

Analisi

In questo capitolo verrà descritta l'analisi sugli aspetti che dovranno caratterizzare la libreria multipiattaforma `EleKtion`.

Verrà esposto un approfondimento sui tipi di voto ed alcuni algoritmi applicabili per ciascun tipo; successivamente sarà definito un elenco di requisiti che dovranno essere considerati nelle fasi successive e infine, sarà fornita una modellazione del dominio mediante diagramma UML.

3.1 Votazioni a singola preferenza

La tipologia di voto a singola preferenza impone al votante di effettuare una scelta tra un numero definito di concorrenti. Al termine della votazione, tipicamente si adotta il criterio della votazione maggioritaria: si contano i voti ottenuti da ciascun concorrente, e colui che ottiene il numero più elevato, risulta vincitore. A seconda del contesto preso in considerazione, è possibile avere più turni per diminuire di volta in volta il numero di concorrenti coinvolti. Se si arriva ad un pareggio, è possibile effettuare un nuovo ballottaggio oppure adottare criteri secondari, che variano in base al dominio trattato.

3.2 Votazioni a lista di preferenze

La tipologia di voto a lista di preferenze non impone al votante di effettuare un'unica scelta, bensì di fissare un ordinamento tra i concorrenti che parta dal meno preferito al più preferito o viceversa. Questa logica permette di conservare più contenuto informativo sulle preferenze rispetto alla singola scelta, e offre anche una panoramica di gradimento degli altri concorrenti. Questo tipo di voto è stato introdotto inizialmente dall'algoritmo di Condorcet ed adottato anche nelle sue varianti. Di seguito viene data una spiegazione sul funzionamento dell'algoritmo di Condorcet, e di un suo derivato, l'algoritmo di Schultze.

3.2.1 Algoritmo di Condorcet

L'algoritmo di Condorcet permette di estrarre un vincitore tra n concorrenti, applicando il criterio di maggioranza nei confronti tra le coppie di concorrenti. Si generano le possibili coppie, si confrontano e si valuta quale coppia ha ottenuto il maggior numero di preferenze. Nello scontro a coppie, un concorrente è vincitore se in un voto occupa una posizione preferenziale rispetto all'altro.

Di seguito viene riportato un esempio, supponendo di avere tre possibili concorrenti, A, B, C, e 60 differenti votanti, che ordinano i concorrenti dalla posizione 1 alla posizione 3 dove la posizione 1 indica la posizione di maggior preferenza. Si ottengono delle liste di preferenze come rappresentato in tabella 3.1, in cui è mostrata l'occorrenza di ciascuna lista.

Posizione 1	Posizione 2	Posizione 3	Numero di occorrenze della lista
A	C	B	23
B	C	A	19
C	B	A	16
C	A	B	2

Tabella 3.1: Tabella delle liste di preferenze, con l'occorrenza di ciascuna lista

Ora è possibile effettuare confronti tra le coppie, partendo ad esempio dalla coppia (B,C)

- Nella prima lista C è in una posizione favorevole rispetto a B, quindi C ottiene 23 punti
- Nella seconda lista B è in una posizione favorevole rispetto a C, quindi B ottiene 19 punti
- Nella terza lista C è in una posizione favorevole rispetto a B, quindi C ottiene 16 punti
- Nella quarta lista C è in una posizione favorevole rispetto a B, quindi C ottiene 2 punti

Si ha che il concorrente C batte il concorrente B con un punteggio di 41 a 19.

Facendo i confronti tra le coppie (A,B) e (A,C) si ottiene che il concorrente B batte il concorrente A con un punteggio di 35 a 25, e il concorrente C batte il concorrente A con un punteggio di 37 a 23. Il risultato finale è quindi il seguente:

- Posizione 1: C
- Posizione 2: B
- Posizione 3: A

In alcuni casi il metodo di Condorcet può non portare ad alcun vincitore, in quel caso si ha un ciclo, ad esempio, trovandosi in una situazione in cui C $>$ B, B $>$ A, A $>$ C.

Le varianti del metodo di Condorcet ottimizzano l'algoritmo per minimizzare il rischio di indeterminazione del vincitore. Può accedere, in Condorcet come nelle sue varianti, di ottenere dei pareggi, i quali avvengono quando due o più concorrenti pareggiano l'uno con l'altro ma battono tutti i restanti. La situazione di pareggio può essere gestita in vari modi, ad esempio, estraendo una sequenza

in modo casuale, valutando altri criteri come la preferenza nei singoli voti (most first choice) oppure mantenendo il set di pareggio senza effettuare alcuna azione.

Dato il risultato della votazione, è possibile definire l'utilità individuale come il grado di soddisfazione del votante. Il grado di soddisfazione sarà tanto più alto quanto più vicino è il risultato rispetto al voto espresso.

Concludendo, l'algoritmo porta ad un risultato nel quale si tende a massimizzare l'utilità per il maggior numero di votanti, ossia a massimizzare l'utilità complessiva.

3.2.2 Algoritmo di Schultze

L'algoritmo di Schultze, conosciuto anche come Schwartz Sequential Dropping (SSD), è una variante di Condorcet basata sulla sommatoria del conteggio delle preferenze tra i concorrenti.

Ad esempio, nel caso in cui si hanno 3 candidati, A, B, C, e questi vengono ordinati in una lista di preferenze dalla posizione 1 alla posizione 3 dove la posizione 1 indica la posizione di maggior preferenza, si può avere il seguente risultato:

- Posizione 1: A
- Posizione 2: B
- Posizione 3: C

In questa lista A ottiene 2 punti, B ottiene 1 punto mentre C nessuno.

Riprendendo la situazione presente in tabella 3.1 e moltiplicando i punti per il numero di occorrenze della lista, si ha un totale dei punti pari a 48 per la lista in oggetto. Proseguendo con l'algoritmo, la situazione finale è mostrata in tabella 3.2

concorrente	x 23	x 19	x 16	x 2	Sommatoria
A	2	0	0	1	48
B	0	2	1	0	54
C	1	1	2	2	78

Tabella 3.2: Tabella delle sommatorie ottenute con l'applicazione dell'algoritmo di Schultze

Prendendo in considerazione le sommatorie in ordine decrescente, otteniamo, il seguente risultato:

- Posizione 1: C
- Posizione 2: B
- Posizione 3: A

Il risultato di Schultze non coincide sempre con il risultato di Condorcet.

3.3 Requisiti

La libreria `EleKtion` fornirà la possibilità di creare ed effettuare votazioni. Essa non tratterà la parte preliminare ad una votazione, quale fase di proposta, discussione, eventuali pre-votazioni o quorum, ma porrà l'obiettivo sulla definizione e gestione dei concorrenti coinvolti, la tipologia di algoritmo utilizzato per ottenere il risultato finale ed i voti presenti nella fase di votazione. Al termine della stessa, sarà disponibile una classifica che riporta informazioni circa le specifiche impostate e le caratteristiche dei concorrenti.

Ciascuna votazione avrà associata una competizione con un metrica che caratterizza l'operato del concorrenti. Prendendo esempi di riferimento legati al mondo del *motor sport*, si potrebbe pensare ai punti totalizzati per ogni gara, come nel caso di *Formula 1 (F1)* o *Moto GP*.

Seppur in molti casi le regole non cambiano in modo drastico, ciascuno sport è influenzato da molti fattori: i concorrenti, gli ambienti di gara, i tifosi, e altri aspetti, che contribuiscono a rendere ogni campionato come un capitolo a sè stante.

Non per ultimo, ognuno di essi ha un proprio metodo di calcolo ed organizzazione delle competizioni. Soprattutto in caso di pareggi, è opportuno avere una logica per risolvere il conflitto e determinare un ordine specifico per le classifiche. Se si considerasse l'esito di ciascuna gara come un votazione, cambiando l'algoritmo utilizzato in un campionato, si otterrebbero esiti differenti, o ancora, se si cambiasse la logica di vittoria nelle fasi di un campionato, si potrebbe simulare come sarebbe stato l'esito finale con regole differenti.

3.3.1 Requisiti funzionali

- Definizione di un gestore di votazioni, intesi come contenitore delle stesse. Ciascuna votazione dovrà essere indipendente dalle altre. L'eventuale collegamento tra le stesse sarà demandato all'utilizzatore della libreria.

- Definizione dei concorrenti che rappresenteranno i candidati disponibili, delle relative caratteristiche e di eventuali punteggi, che potranno essere presenti a

seconda del dominio della votazione. Gli eventuali punteggi saranno collegati ad una specifica metrica di interesse.

- Definizione dell'algoritmo da utilizzare. L'algoritmo dovrà determinare la logica con cui saranno confrontati e selezionati i concorrenti. L'algoritmo potrà utilizzare la metrica disponibile per effettuare ulteriori valutazioni in occasione di pareggio o altre casistiche definibili dall'utilizzatore. L'algoritmo potrà essere parametrizzato, per impattare il proprio funzionamento.

- Definizione dei voti presenti nella votazione. La tipologia di voto dovrà essere dipendente dall'algoritmo utilizzato, e in ogni caso potrà essere dei seguenti tipi:

  1. Voto a singola preferenza
  2. Voto a lista di preferenze

  Dovrà essere data la possibilita si anonimizzare il votante, in modo tale che successivamente non sia possibile risalire al suo identificativo.

- Le fasi di definizione dovranno essere correlate ad opportuni controlli di congruità, al fine di elaborare la votazione in modo coerente.

- La classifica finale dovrà essere disponibile e visualizzabile dall'utilizzatore. Dovrà esser data evidenza di eventuali situazioni di pareggio, di quale algoritmo è stato utilizzato e delle caratteristiche dei concorrenti.

3.3.2 Requisiti non funzionali

- Dovrà essere possibile definire tutti gli elementi necessari attraverso l'uso di costrutti che agevolino la rapidità di creazione, evitando che l'utilizzatore effettui manualmente i controlli minimi ed obbligatori.

- La libreria dovrà essere disponibile su molteplici piattaforme, facilmente importabile in esse ed avere una documentazione a corredo.

3.4 Modello del dominio

`EleKtion` avrà a disposizione uno o più gestori di votazioni. I gestori sono indipendenti tra loro e così anche le votazioni all'interno degli stessi, fatto salvo l'uso coerente della tipologia di votazione e della metrica di punteggio, che rimane stabile in ogni gestore.

Non è contemplata l'ipotesi di più turni nella stessa votazione, quindi in tal caso dovranno essere gestiti come votazioni separate.

Una votazione avrà associata una determinata competizione, alla quale è associato l'elenco dei concorrenti.

Ciascun concorrente può avere una serie di punteggi, e la tipologia di punteggio è strettamente collegato alla tipologia di competizione.

La votazione, inoltre, ha associato l'elenco dei voti, i quali hanno riferimento sia del votato che del votante, in caso di votazione non anonima. È importante che un voto, per essere considerato valido, sia riferito ad un candidato esistente e sia della tipologia ammessa dalla votazione.

Infine, ci può essere un solo algoritmo collegato alla votazione, il quale può contenere dei parametri per effettuare variazioni nella configurazione per il suo funzionamento. La votazione avrà un' unica classifica finale, la quale può essere consultata.

In figura 3.1 viene rappresentato il diagramma UML con le interfacce che modellano il dominio descritto in questo capitolo.

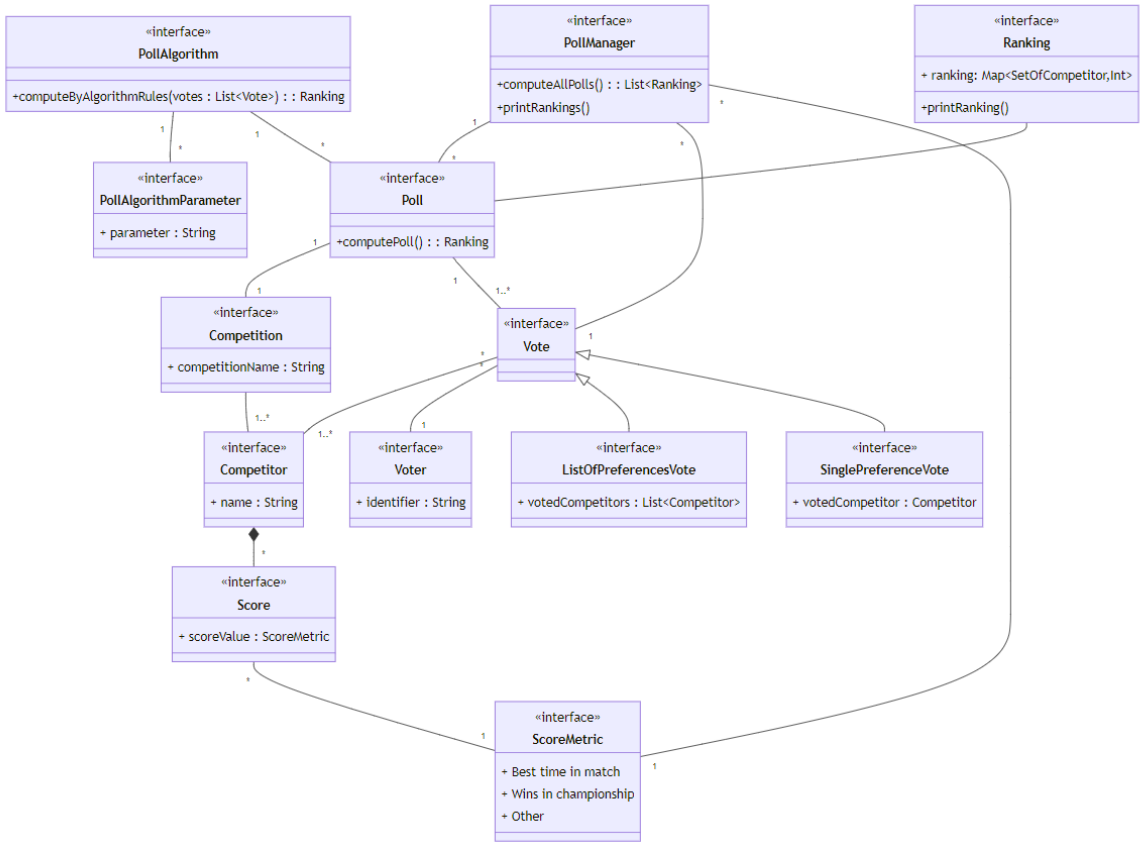

Figura 3.1: Diagramma UML del modello di dominio di `EleKtion`

Capitolo 4

Design

In questo capitolo, partendo dalla modellazione del dominio descritta nel capitolo 3.4, sarà sviluppato un approfondimento sulla definizione delle metriche di punteggio e sulla porzione impattata dall'introduzione di DSL, volto al miglioramento dell'utilizzabilità della libreria `EleKtion`. Per ciascuna funzionalità verrà evidenzata un'apposita descrizione e un diagramma UML che esplichi come saranno organizzate le interfacce e le classi astratte.

Questi diagrammi sono realizzati con il presupposto di poter adottare l'utilizzo dei seguenti costrutti:

- Tipi `generici` [**?**];
- `Overloading` degli `operatori` [**?**];
- `Higher order functions` (in particolare `function literal with receiver`) [**?**];
- `Companion object` [**?**].

Si rimanda alle fonti per l'approfondimento specifico in `Kotlin`, la sintassi utilizzata nei diagrammi è generica ed astrae dal linguaggio specifico.

4.1 Domain Specific Language per la definizione del gestore di votazioni

Questo DSL è pensato per facilitare la creazione del gestore di votazioni. Un gestore di votazioni viene vincolato ad adottare uno specifico tipo `S`, sottotipo di `ScoreMetric`, ed uno specifico tipo di voto `V`, sottotipo di `Vote`.

Una metrica è definibile come il criterio tale per cui un concorrente può essere messo a confronto con un altro, sulla base della sua prestazione nella competizione. Pertanto, è rappresentabile come una comparazione tra valori. Dall'interfaccia `ScoreMetric` possono essere definiti numerosi sottotipi utili alla competizione trattata, ad esempio il punteggio di un concorrente può essere legato ai punti assegnati in una gara, alle vittorie ottenute in campionato, oppure a criteri legati al tempo. Per facilitarne la creazione è utile definire una funzione di supporto contenuta in un `companion object`, come mostrato in figura 4.1.

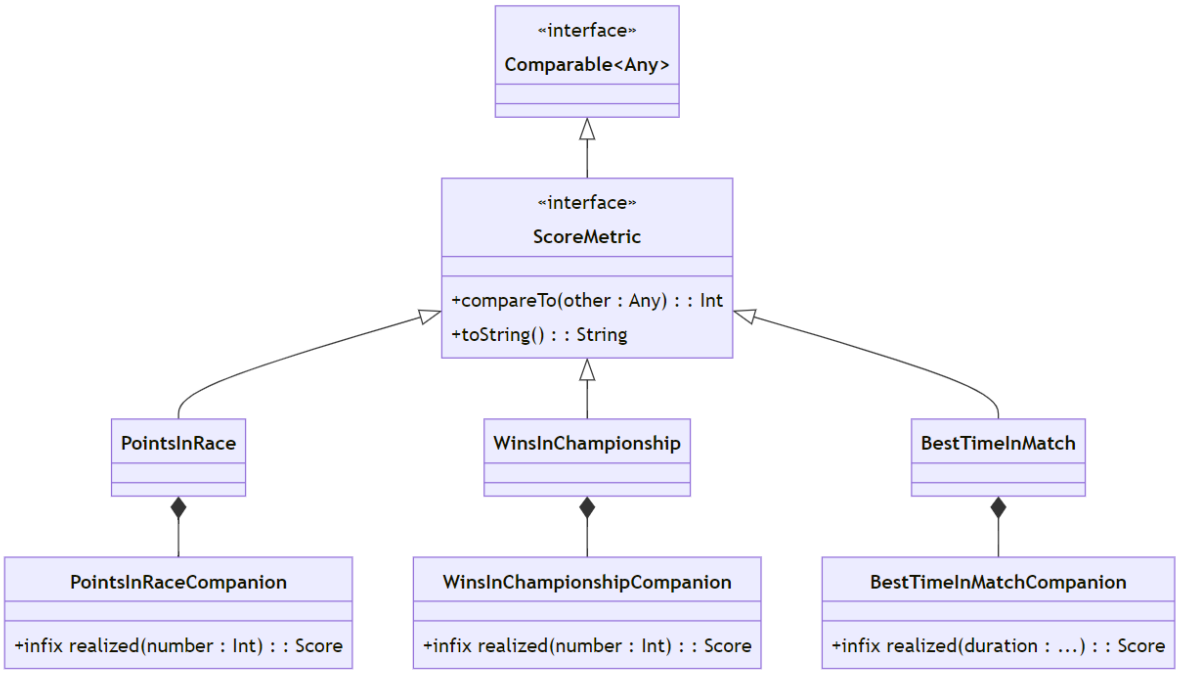

Figura 4.1: Diagramma UML dell'organizzazione di una metrica

I tipi `S` e `V` vincolano i seguenti componenti che verranno trattati, pertanto non è possibile adottare diversi metriche di punteggio tra i concorrenti, o diversi tipologie di voto, tra le varie votazioni presenti nel medesimo gestore.

Il punto di ingresso dell'intero flusso è rappresentato dalla funzione `initializedAs`, al suo interno è possibile inizializzare una votazione attraverso la funzione `poll`, e l'utilizzo dell'operatore unario `+`, per l'aggiunta di quest'ultima alla lista di votazioni. In figura 4.2 viene mostrato il diagramma UML delle interfacce e della classe astratta.

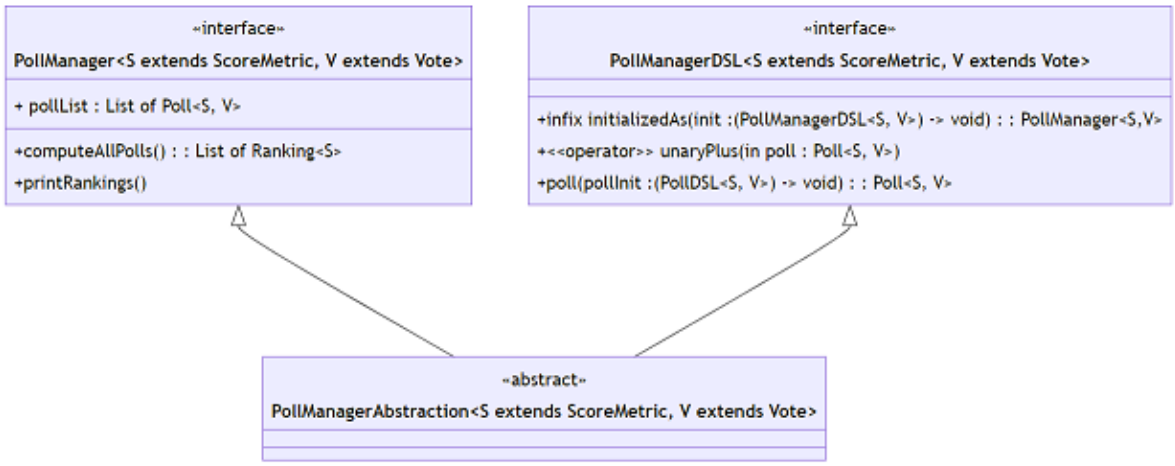

Figura 4.2: Diagramma UML del DSL per la definizione del gestore di votazioni

4.2 Domain Specific Language per la definizione della votazione

Questo DSL è pensato per facilitare la creazione di una votazione. In una votazione è obbligatorio scegliere l'algoritmo con cui sarà elaborato l'esito, l'algoritmo può ricevere parametri utili al uso funzionamento, attraverso l'utilizzo dell' operatore unario `+` presente in `PollAlgorithmDSL`. In base al tipo di voto (vedi capitolo 3.3.1) è utile definire un costrutto per agevolarne la creazione. Nel caso in cui il voto sia a singola preferenza, è necessario specificare l'identificativo del votato e l'identificativo del votante (vedi funzione `votedBy` in `SPVoteAlgorithmDSL`, figura 4.3). Nel caso in cui quest'ultimo non sia richiesto, corrisponderà ad un identificativo casuale (vedi funzione `asAnonymousVote` in `SPVoteAlgorithmDSL`, figura 4.3). Nel caso in cui il voto sia a lista di preferenze, per costruire la lista è necessario specificare l'elenco degli identificativi dei votati, utilizzando la funzione `then` in concatenazione (vedi funzione `then` in `LOPVoteAlgorithmDSL`, figura 4.3)). Allo stesso modo del voto a singola preferenza, è prevista la possibilità di anonimizzare il votante. Come visibile in figura 4.3, sono previsti metodi per semplificare l'inizializzazione dei possibili algoritmi:

- `majorityVotesAlgorithm`: rappresenta la creazione di un algoritmo a maggioranza, in cui il vincitore è colui che ricevuto più voti;
- `majorityVotesHScoreAlgorithm`: rappresenta la creazione di un algoritmo a maggioranza, in cui il vincitore è colui che ricevuto più voti. Se si verificano pareggi, viene selezionato il concorrente che ha il punteggio maggiore, secondo la metrica di punteggio definita;
- `majorityVotesHScoreAlgorithm`: rappresenta la creazione di un algoritmo a maggioranza, in cui il vincitore è colui che ricevuto più voti. Se si verificano pareggi, viene selezionato il concorrente che ha il punteggio minore, secondo la metrica di punteggio definita;
- `condorcetAlgorithm`: rappresenta la creazione dell'algoritmo di Condorcet;

- `schultzeAlgorithm`: rappresenta la creazione dell'algoritmo di Schultze.

In figura 4.4 vengono mostrati gli altri metodi utili alla creazione di una votazione: La funzione `competition`, utile alla creazione della competizione, richiede che venga definito un nome alla competizione e un inizializzatore in cui saranno definiti i concorrenti. Quest'ultimo, così come l'algoritmo desiderato, sono effettivamente assegnati alla votazione attraverso l'uso dell'operatore unario `-`. Infine è possibile aggiungere un elenco di voti, preferibilmente creati utilizzando i metodi descritti nella prima parte di questo sotto capitolo. Ciascun voto richiede di essere effettivamente aggiunto alla lista di voti attraverso l'uso dell' operatore unario `+`.

Per motivi di spazio, nelle figure 4.3 e 4.4 sono stati utilizzati gli acronimi `SP` e `LOP`, al posto di, rispettivamente, `SinglePreference` e `ListOfPreferences`.

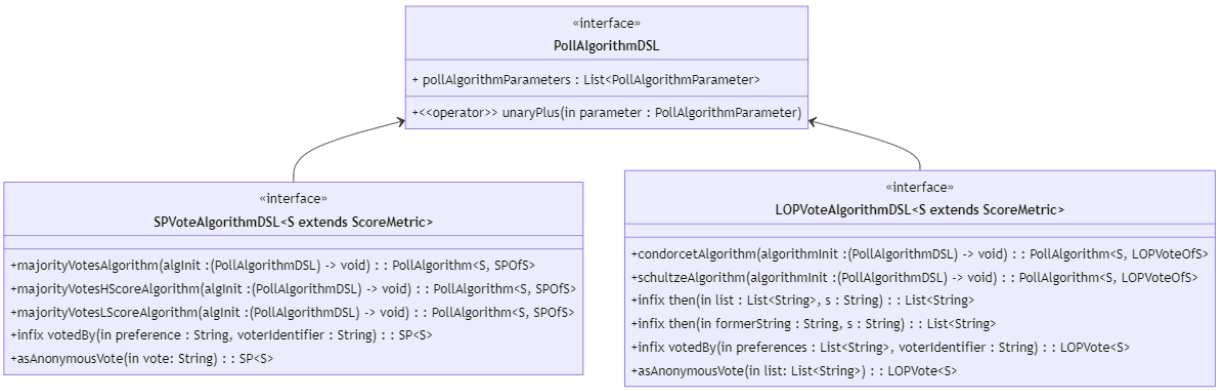

Figura 4.3: Diagramma UML del DSL per la porzione di definizione dell'algoritmo

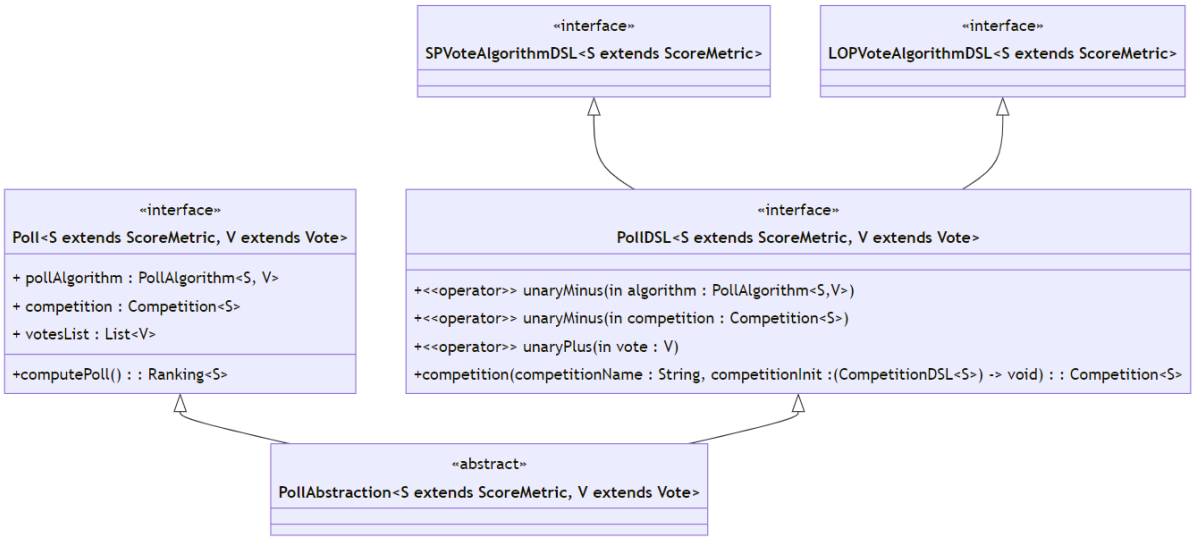

Figura 4.4: Diagramma UML del DSL per la porzione di definizione della votazione

4.3 Domain Specific Language per la definizione della competizione

Questo DSL è pensato per facilitare la creazione ed il popolamento di una competizione. La funzione `competitor`, utile alla creazione del concorrente, richiede che venga definito l'identificativo del concorrente e un inizializzatore in cui saranno definiti i punteggi realizzati. Il concorrente istanziato, viene aggiunto alla lista dei concorrenti grazie all'uso dell'operatore unario `+`.

In figura 4.5 viene mostrato il diagramma UML delle interfacce e della classe astratta.

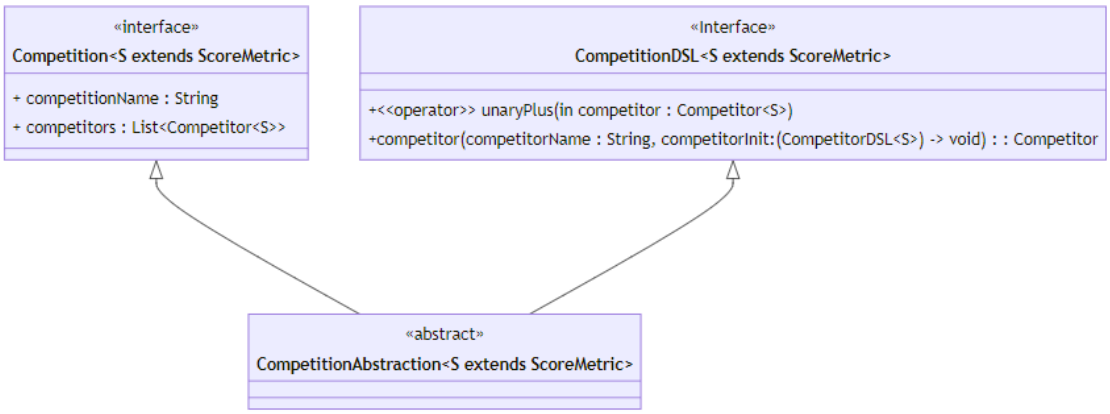

Figura 4.5: Diagramma UML del DSL per la definizione della competizione

4.4 Domain Specific Language per la definizione del concorrente

Questo DSL è ideato per facilitare la creazione di un concorrente. Ciascun concorrente può avere un elenco di punteggi, e ciascuno di essi può essere assegnato al concorrente (vedi capitolo 4.3) attraverso l'uso dell'operatore unario +. .

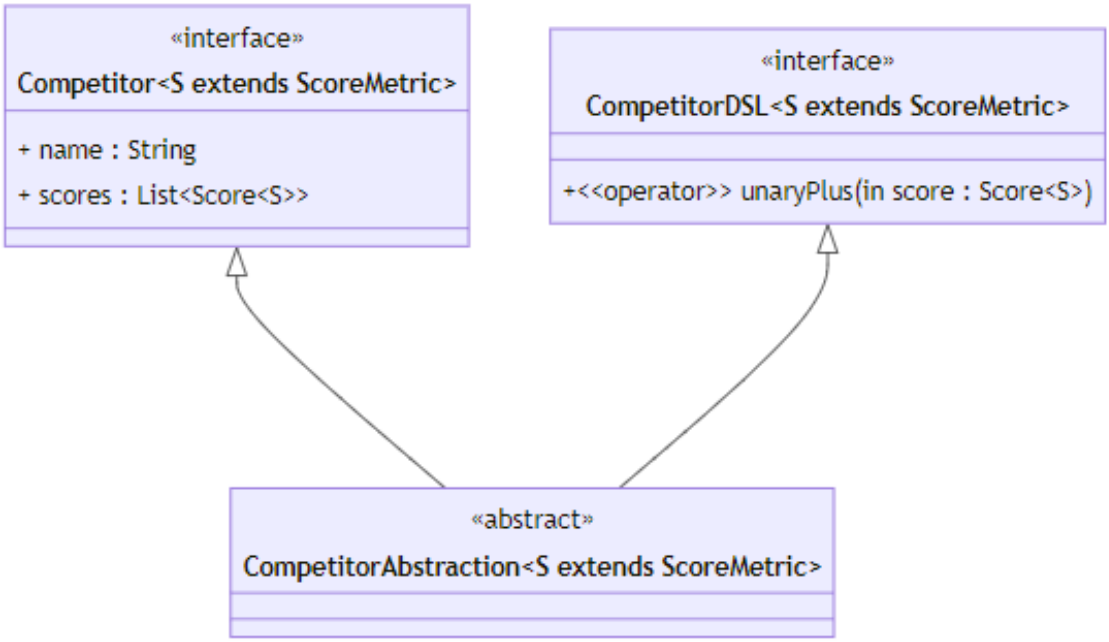

Figura 4.6: Diagramma UML del DSL per la definizione della concorrente

Capitolo 5

Implementazione

5.1 Build automation e versionamento degli artefatti

L'adozione di un sistema di build automation è un fattore chiave per la produzione efficace ed efficiente di software di qualità.

Come principio base, è importante che ciascuna modifica apportata al codice sia testata a livello unitario e abbia documentazione correlata, che sia l'aggiunta di una funzionalità piuttosto che la variazione di un campo, a titolo meramente esemplificativo.

Inoltre, queste modifiche devono essere ulteriormente verificate attraverso test di integrazione, in modo da garantire la piena compatibilità tra i componenti.

Le verifiche possono essere effettuate manualmente dallo sviluppatore, ma per ridurre il rischio di errori è possibile adottare un sistema di automazione, che attraverso una serie di task predefiniti e ripetibili, verifichi automaticamente le varie fasi, gestendo anche le dipendenze tra i task stessi. Così facendo, si può subito notare se tutto è andato a buon fine oppure se è necessario effettuare ulteriori eventi correttivi.

Una volta che il flusso di compilazione e verifica viene terminato con successo, è possibile procedere alla generazione di uno o più artefatti e della relativa

documentazione. Questi prodotti possono essere sottoposti ad un operazione di versionamento, in modo da distinguere le evoluzioni del software e semplificarne la distribuzione e la tracciabilità.

In questo progetto sono stati adottati `Kotlin` come linguaggio di programmazione e `Gradle` come build automator.

Poichè gli artefatti che vengono generati sono dipendenti dall'architettura della macchina e dal sistema operativo sui quali avviene il processo di build, è opportuno eseguire tale processo su piattorme dedicate come Github. Github fornisce ambienti standard e personalizzabili in base a direttive, gestendo il processo di Continuous Integration (CI)/Continuous Delivery (CD) che è stato descritto in questo capitolo. Infine, per favorire la distribuzione e la compatibilità verso molteplici sistemi, è utile adottare strumenti come `Kotlin Multiplatform`, che semplificano la generazione dell'output finale adattandolo alle specifiche piattaforme.

5.1.1 Kotlin Multiplatform

Kotlin Multiplatform è un tool che favorisce il riuso di codice tra molteplici piattaforme. Grazie ad esso, riutilizzando la logica applicativa scritta in `Kotlin`, è possibile produrre artefatti compatibili in molteplici piattaforme, dette anche *target*, come `JVM`, `JS`, `Android`, `iOS` e piattaforme native come `Linux`. I *target* sono disposti secondo una gerarchia predefinita, comunque modificabile in caso di necessità, e sono previsti *target* intermedi. Per ogni *target* viene definito un *source set*, ossia un insieme di file sorgente, che definisce anche le dipendenze e le compatibilità. Il *source set* predefinito è `commonMain`, e contiene il codice comune a tutti i *target*. Al suo interno è possibile utilizzare funzioni e dipendenze che siano compilabili verso tutte le piattaforme dichiarate, mentre quelle che sono legate ad uno specifico linguaggio o architettura vanno definite ed utilizzate all'interno degli opportuni *source sets*, come `jvmMain` e `jsMain`. Un esempio è visibile in figura 5.2, in cui i *target* intermedi sono colorati in azzurro: `apple` conterrà codice compatibile solamente con i *target* che lo estendono, oltre a quello ereditato dai livelli superiori.

Pertanto, è ammesso che `macos` utilizzi funzioni che non siano disponibili in `tvos` ma entrambi accedono al codice messo a disposizione da `apple`.

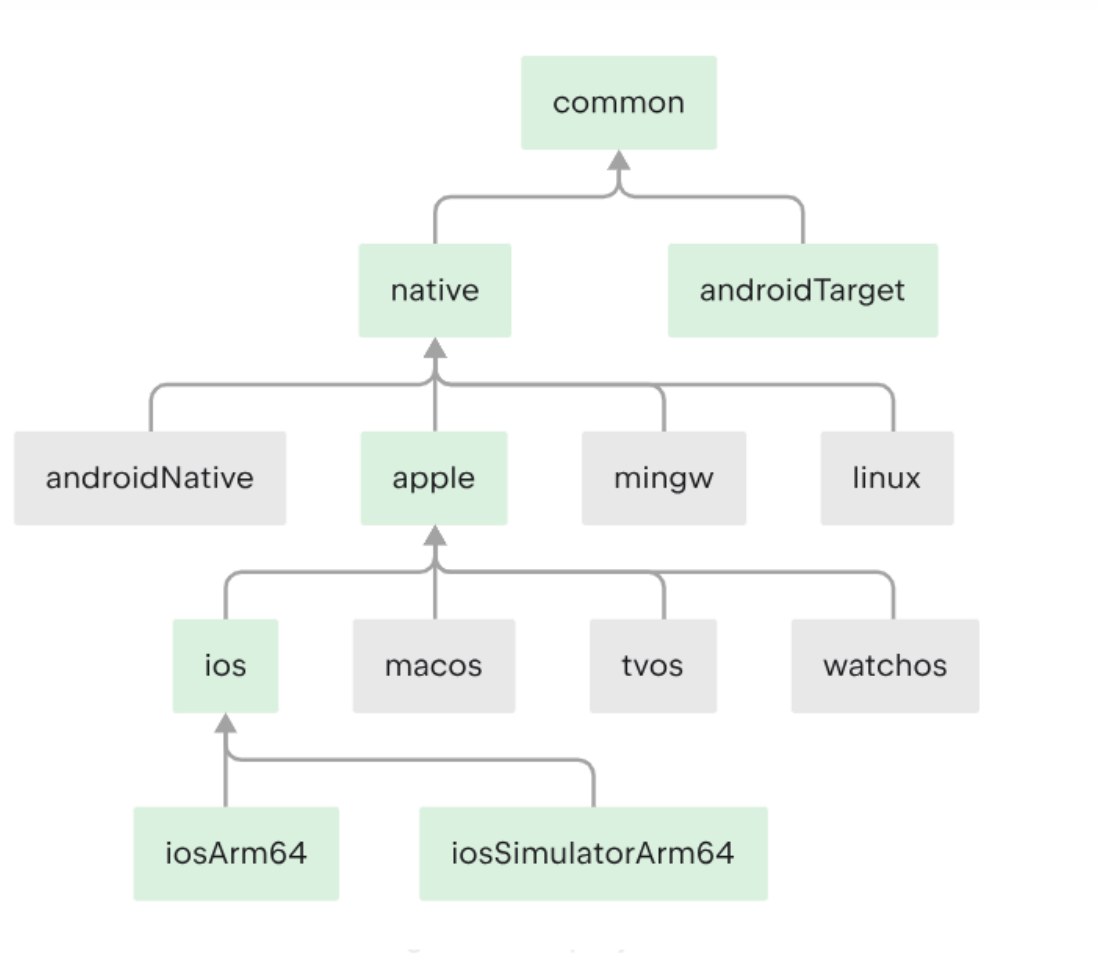

Figura 5.1: Esempio di una possibile gerarchia di *target*. Fonte: [**?**]

Inoltre, questo meccanismo permette di applicare un *template method* architetturale, grazie al quale è possibile dichiarare nel *source set* `commonMain` funzioni agnostiche tramite la direttiva `expected`, senza definirne il contenuto, che sarà valorizzato in un successivo momento. In fase di compilazione, nelle versioni platform-specific, saranno valorizzate tramite funzioni che adottano la direttiva `actual`, che posso richiamare funzionalità altrimenti non adottabili in `commonMain`.

Una logica analoga è applicabile anche alla sezione dei test: è disponibile un *source set* di base, detto `commonTest`, e altri eventuali che contengono codice compatibile solo in quella determinata piattaforma. In figura **??** viene mostrato il flusso e gli output ottenuti utilizzando i *target* `JVM` e `JS`.

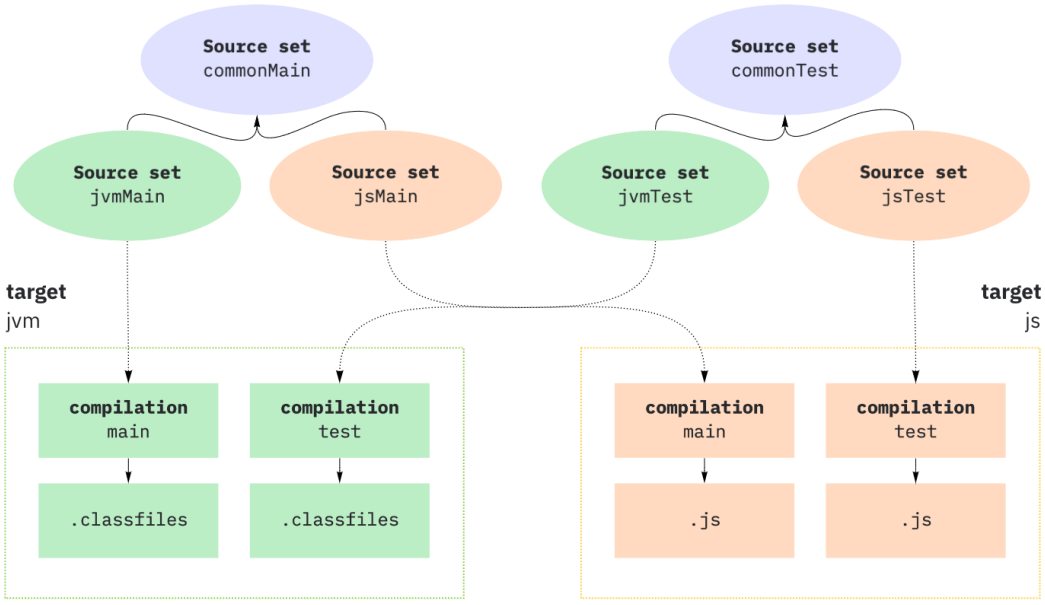

Figura 5.2: Esempio di output ottenibili in un progetto multipiattaforma. Fonte: [?]

5.1.2 GitHub Actions

GitHub Actions (GH Actions) è una piattaforma di CI/CD basata su repository GitHub (GH), ai quali possono essere agganciati degli elementi detti workflow.

Un `workflow`, definibile attraverso la sintassi `YAML`, rappresenta una sequenza di azioni che possono essere eseguite all'occorrenza di un evento, ad esempio la creazione di un `tag` nel repository, una `richiesta pull`, un'esecuzione manuale richiesta dal proprietario del repository, ecc...

In ogni `workflow` va specificato il nome dello stesso e l'insieme di eventi che ne scatena l'esecuzione. Di seguito a ciò, è possibile definire un insieme di `job`.

Ciascun `job` è legato ad uno specifico ambiente di esecuzione: va definito che tipo di sistema operativo deve essere adottato (`Linux`, `Windows`, `macOS`) e se al termine dello stesso debba essere restituito un output, che sarà accessibile da altri `job`. È possibile specificare delle variabili in una `matrice`, in modo eseguire parallelamente molteplici istanze di uno stesso `job` in base alle combinazioni delle variabili; a titolo esemplificativo, eseguire la compilazione di codice utilizzando la stessa versione del compilatore ma su 3 sistemi operativi differenti.

Ciascun `job` è caratterizzato da un proprio flusso indipendente, di conseguenza per orchestare l'intero processo è opportuno specificare condizioni basate sulle variabili di output oppure sull'indicazione del completamento di uno o più i `jobs` desiderati.

All'interno del `job` sono definibili uno o più `step`: questi vengono eseguiti in maniera sequenziale e sono impostabili per eseguire una sequenza arbitraria di comandi `bash` oppure per eseguire una `action` pubblica. Quest'ultima rappresenta un ottimo modo di favorire il riuso e la condivisione di codice nella community GH, in quanto è possibile, ad esempio, utilizzare un workflow complesso come se fosse un singolo step (`composite action`) oppure lanciare l'esecuzione di un container senza imporre *effort* riguardo alla gestione delle dipendenze e delle configurazioni (`docker container action`).

5.1.3 GitHub Pages

GitHub Pages (GH Pages) è un servizio di hosting disponibile per ogni account ed organizzazione in GH, attraverso il quale è possibile pubblicare un sito web statico, che viene costruito e pubblica attraverso un apposito `workflow` predefinito. Nel caso di un iniziativa progettuale come una libreria, può essere utilizzato dare informazioni su come scaricare ed installare il software oppure per pubblicare la documentazione.

5.1.4 Maven Central, GitHub Packages e NPM

Maven Central è tra i repository centralizzati più popolari per distribuire e riutilizzare artefatti basati su Java Virtual Machine (JVM). Un repository è definibile come "la directory che memorizza tutti i progetti e le librerie JAR, i plugin o qualsiasi altro artefatto specifico del progetto che può essere facilmente utilizzato da Maven"[**?**].

Il sistema di gestione delle dipendenze di Maven è in grado di risolvere automaticamente le dipendenze transitive, semplificando la gestione manuale delle dipendenze per gli sviluppatori. Gli artefatti caricati su Maven Central, tra cui librerie di codice compilato, documentazione ed elenco di dipendenze, sono immutabili e non possono essere modificati né eliminati. Le dipendenze vengono identificate in modo univoco con una tripletta `groupId:artifactId:version`, dove `groupId` identifica un gruppo di artefatti, `artifactId` si riferisce al nome della libreria e `version` identifica univocamente ogni rilascio della libreria. Ad esempio, la tripletta 'org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.0' identifica la versione 1.6.0 di un serializzatore json per `Kotlin`, gestito da `JetBrains`.

Tutte le versioni di una libreria rilasciate su Maven Central, una volta pubblicate sono sempre disponibili nel repository. Gli utenti della libreria sono liberi di decidere quale versione utilizzare e per quanto tempo e a questi ultimi è delegata la responsabilità di aggiornare le proprie dipendenze per adeguarsi a problemi di sicurezza o evoluzioni dell'API. Nel lungo termine, queste decisioni determinano le librerie e le versioni più popolari nell'intero ecosistema software. Si osserva che per la maggior parte delle librerie, più di una versione è attivamente utilizzata in un dato momento [**?**].

Dallo studio empirico condotto da [**?**], è emerso che "circa il 40% delle librerie ha due o più versioni attivamente utilizzate, mentre quasi il 4% non ha mai avuto alcun utente su Maven Central. Inoltre, si è scoperto che più del 90% delle versioni più popolari non sono le ultime versioni rilasciate, e sia le versioni attive che significativamente popolari sono distribuite lungo la storia delle versioni della libreria".

Un' alternativa all'uso di Maven Central può risiedere in GH Packages. Quest'ultimo pone una gestione semplificata per la pubblicazione degli artefatti, in quanto strettamente connesso all'uso di repository GH, il che permette di avere un controllo degli accessi efficace come quello disponibile per i repo. Al pari di Maven Central viene mantenuto il concetto di artefatti immutabili, e in più viene dato il supporto per l'archiviazione di pacchetti in numerosi linguaggi, tra cui `C#`.

Un altro famoso gestore di pacchetti è Node Package Manager (NPM), focalizzato sulle diffusione di librerie `Javascript` e `Node.js`, si differenzia in particolare dai precedenti in quanto non adotta il principio di immutabilità. In questo caso, la modifica o eliminazione di una versione di un pacchetto può avere impatti notevoli sugli utilizzatori, portando a potenziali disastri mondiali [**?**].

5.2 Casi realizzati per i differenti tipi di algoritmi

Nel capitolo 3.4 e nel capitolo 4 è stata presentata l'organizzazione del dominio da implementare, assieme ad un approfondimento sulle entità che compongono il linguaggio DSL che è stato realizzando appositamente per questa libreria. In questo capitolo verranno mostrati dei casi d'uso realizzati per verificare il corretto funzionamento della libreria, spiegando anche come vengono gestiti eventuali conflitti a seconda della logica scelta.

5.2.1 Votazioni a singola preferenza

Per gestire le votazioni a singola preferenza è stato creato inizialmente il caso base per la votazione a maggioranza, nella quale il vincitore è il concorrente che riceve più voti. In alcuni casi è possibile avere parità tra i voti, pertanto ci sono più vincitori nella stessa posizione. Successivamente è stata estesa la gestione dei pareggi, dando la possibilità all'utilizzatore della libreria di definire una metrica per la competizione ed un elenco di punteggi realizzati da ciascun concorrente, basati su quella

metrica. Nel caso in cui si ha un pareggio di voti è possibile decidere di effettuare un ulteriore sotto-confronto basato sui punteggi, per selezionare chi risulta vincitore sulla base del punteggio più basso o più alto realizzato nella competizione. Questo sotto-confronto può portare ad avere dei vincitori a pari merito, oppure di piazzarli in posizioni differenti, proprio per la differenza dei punteggi. La gestione quindi non garantisce l'assenza totale di pareggi, che può verificarsi anche a livello di punteggio. Inoltre, è possibile avere un parametro che indica la capacità di effettuare molteplici voti per votante , ma di negargli comunque l'opportunità di votare più volte lo stesso candidato . Le classi realizzate per gestire tutte queste casistiche sono `MajorityVotesAlgorithm.kt`, `MajorityVotesThenHighestScoreAlgorithm` e `MajorityVotesThenLowestScoreAlgorithm`. Nell'estratto di codice 5.1 vengono mostrati i controlli effettuati prima di compiere la scelta del vincitore.

Estratto di codice 5.1: Controlli effettuati prima della votazione a singola preferenza

```
1   if (candidates.groupingBy { it.name }.eachCount().any { it.value > 1 }) {
2           error("Candidate already declared")
3       }
4
5       require(votes.any())
6
7       if (votes.map { it.votedCompetitor }.any { it !in candidates }) {
8           error("Voted candidate doesn't exist as object")
9       }
10
11      when (pollAlgorithmParameters.count { it == ConstantParameter.
            AllowMultipleVoteInPollParameter }) {
12          0 -> {
13              if (votes.groupingBy { it.voter.identifier }.eachCount().any { it.
                    value > 1 }) {
14                  error("Each voter can vote only once")
15              }
16          }
17          1 -> {
18              // multiple vote allowed
19              if (votes.groupingBy {
20                      Pair(
21                          it.votedCompetitor.name,
22                          it.voter.identifier,
```

```
                    )
                }.eachCount().any { it.value > 1 }
            ) {
                error("Each voter can vote just once for each competitor")
            }
        }
        else -> error("Parameter can't be repeated more than once")
    }
```

Successivamente, viene generata la classifica, che contiene l'insieme dei vincitori con il relativo numero di voti. Per ciascun vincitore, viene riportato il punteggio maggiore o quello minore. Nel caso in cui si utilizzi l'algoritmo a maggioranza senza sotto-confronto, vengono riportati tutti i punteggi realizzati, inoltre per quest'ultimo non è obbligatorio associare dei punteggi ai giocatori.

Nell'estratto di codice 5.2 vengono mostrati alcuni esempi realizzati: nella definizione di ogni competizione è essenziale dichiarare l'elenco di tutti i possibili candidati in modo da poter verificare che non ci siano voti associati ad un candidato non esistente. Questo controllo è effettuato anche dal DSL nella fase di definizione dei voti, e quindi in precedenza all'invocazione dell'algoritmo, che replica questo controllo in quanto è possibile utilizzarlo indipendentemente dall'adozione di DSL. Si definisce poi l'algoritmo da utilizzare e l'elenco di voti. I voti possono essere anonimizzati, utilizzando come identificativo una stringa `UUID`.

Estratto di codice 5.2: Esempi di votazioni a singola preferenza, attraverso l'utilizzo di DSL

```
var election = PollManagerInstance<BestTimeInMatch, SinglePreferenceVote<
    BestTimeInMatch>>() initializedAs {
        +poll {
            -competition("Race 1") {
                +competitor("competitor1") {
                }

                +competitor("competitor2") {
                }
            }

            -majorityVotesAlgorithm {
                +ConstantParameter.AllowMultipleVoteInPollParameter
```

```
13                }
14                +("competitor1" votedBy "voter1")
15                +("competitor2" votedBy "voter2")
16                +("competitor1" votedBy "voter2")
17            } // competitor1 wins
18
19            +poll {
20                -competition("Race 2") {
21                    +competitor("competitor1") {
22                        +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                              HOURS)))
23                        +(BestTimeInMatch realized (4.toDuration(DurationUnit.
                              HOURS)))
24                    }
25
26                    +competitor("competitor2") {
27                        +(BestTimeInMatch realized (2.toDuration(DurationUnit.
                              HOURS)))
28                    }
29                }
30                -majorityVotesAlgorithm {
31                    +ConstantParameter.AllowMultipleVoteInPollParameter
32                }
33
34                +("competitor1" votedBy "voter1")
35                +("competitor2" votedBy "voter3")
36                +("competitor2" votedBy "voter2")
37                +("competitor1" votedBy "voter2")
38            } // competitor1 and competitor2 have tie in votes
39        }
40        election.printRankings()
41
42    //    ** Ranking #1 **
43    //    Competition name is Race 1
44    //    Used algorithm is MajorityVotesAlgorithm
45    //    Placement #1 -> [competitor1] with 2 votes
46    //      competitor1 scores list :
47    //            - no score available
48
49    //    ** Ranking #2 **
50    //    Competition name is Race 2
51    //    Used algorithm is MajorityVotesAlgorithm
52    //    Placement #1 -> [competitor1 | competitor2] with 2 votes
53    //      competitor1 scores list :
54    //            - Score type is BestTimeInMatch with time = 1h -
55    //            - Score type is BestTimeInMatch with time = 4h -
```

```
56      //      competitor2 scores list :
57      //            - Score type is BestTimeInMatch with time = 2h -
58
59
60
61      election =
62          PollManagerInstance<BestTimeInMatch, SinglePreferenceVote<BestTimeInMatch
                >>() initializedAs {
63              +poll {
64                  -competition("Race 1") {
65                      +competitor("competitor1") {
66                          +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                                HOURS)))
67                          +(BestTimeInMatch realized (2.toDuration(DurationUnit.
                                HOURS)))
68                      }
69
70                      +competitor("competitor2") {
71                          +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                                HOURS)))
72                      }
73                  }
74                  -majorityVotesHScoreAlgorithm {
75                      +ConstantParameter.AllowMultipleVoteInPollParameter
76                  }
77
78                  +("competitor2" votedBy "voter1")
79                  +("competitor1" votedBy "voter1")
80                  +("competitor1" votedBy "voter2")
81              } // competitor1 has more votes but select highest score anyway
82
83              +poll {
84                  -competition("Race 2") {
85                      +competitor("competitor1") {
86                          // T = 0.5
87                          +(BestTimeInMatch realized (T.toDuration(DurationUnit.
                                HOURS)))
88                          +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                                HOURS)))
89                          +(BestTimeInMatch realized (2.toDuration(DurationUnit.
                                HOURS)))
90                      }
91
92                      +competitor("competitor2") {
93                          +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                                HOURS)))
```

```
94                      }
95                  }
96                  -majorityVotesHScoreAlgorithm {
97                      +ConstantParameter.AllowMultipleVoteInPollParameter
98                  }
99
100                 +("competitor2" votedBy "voter1")
101                 +("competitor1" votedBy "voter1")
102                 +("competitor2" votedBy "voter2")
103                 +("competitor1" votedBy "voter2")
104             } // same votes, competitor1 has the highest score
105
106             +poll {
107                 -competition("Race 3") {
108                     +competitor("competitor1") {
109                         +(BestTimeInMatch realized (T.toDuration(DurationUnit.
                                HOURS)))
110                         +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                                HOURS)))
111                         +(BestTimeInMatch realized (2.toDuration(DurationUnit.
                                HOURS)))
112                     }
113
114                     +competitor("competitor2") {
115                         +(BestTimeInMatch realized (2.toDuration(DurationUnit.
                                HOURS)))
116                     }
117                 }
118                 -majorityVotesHScoreAlgorithm {
119                     +ConstantParameter.AllowMultipleVoteInPollParameter
120                 }
121
122                 +("competitor2" votedBy "voter1")
123                 +("competitor1" votedBy "voter1")
124                 +("competitor2" votedBy "voter2")
125                 +("competitor1" votedBy "voter2")
126             } // same votes, same highest score
127         }
128
129         election.printRankings()
130
131
132         // ** Ranking #1 **
133         // Competition name is Race 1
134         // Used algorithm is MajorityVotesThenHighestScoreAlgorithm
135         // Placement #1 -> [competitor1] with 2 votes
```

```
136         //      competitor1 scores list :
137         //          - Score type is BestTimeInMatch with time = 2h -
138
139         // ** Ranking #2 **
140         // Competition name is Race 2
141         // Used algorithm is MajorityVotesThenHighestScoreAlgorithm
142         //  Placement #1 -> [competitor1] with 2 votes
143         //      competitor1 scores list :
144         //          - Score type is BestTimeInMatch with time = 2h -
145         //  Placement #2 -> [competitor2] with 2 votes
146         //      competitor2 scores list :
147         //          - Score type is BestTimeInMatch with time = 1h -
148
149         // ** Ranking #3 **
150         // Competition name is Race 3
151         // Used algorithm is MajorityVotesThenHighestScoreAlgorithm
152
153         // Placement #1 -> [competitor2 | competitor1] with 2 votes
154         //      competitor2 scores list :
155         //          - Score type is BestTimeInMatch with time = 2h -
156         //      competitor1 scores list :
157         //          - Score type is BestTimeInMatch with time = 2h -
158
159
160         election =
161         PollManagerInstance<BestTimeInMatch, SinglePreferenceVote<BestTimeInMatch
                >>() initializedAs {
162             +poll {
163                 -competition("Race 1") {
164                     +competitor("competitor1") {
165                         +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                                HOURS)))
166                         +(BestTimeInMatch realized (2.toDuration(DurationUnit.
                                HOURS)))
167                     }
168
169                     +competitor("competitor2") {
170                         +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                                HOURS)))
171                     }
172                 }
173                 -majorityVotesLScoreAlgorithm {
174                     +ConstantParameter.AllowMultipleVoteInPollParameter
175                 }
176
177                 +("competitor2" votedBy "voter1")
```

```
178                 +("competitor2" votedBy "voter2")
179                 +("competitor1" votedBy "voter2")
180             } // same score, competitor2 has more votes
181
182             +poll {
183                 -competition("Race 2") {
184                     +competitor("competitor1") {
185                         +(BestTimeInMatch realized (T.toDuration(DurationUnit.
                                HOURS)))
186                         +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                                HOURS)))
187                         +(BestTimeInMatch realized (2.toDuration(DurationUnit.
                                HOURS)))
188                     }
189
190                     +competitor("competitor2") {
191                         +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                                HOURS)))
192                     }
193                 }
194                 -majorityVotesLScoreAlgorithm {
195                     +ConstantParameter.AllowMultipleVoteInPollParameter
196                 }
197
198                 +("competitor2" votedBy "voter1")
199                 +("competitor1" votedBy "voter1")
200                 +("competitor2" votedBy "voter2")
201                 +("competitor1" votedBy "voter2")
202             } // same votes, competitor1 has the lowest score
203
204             +poll {
205                 -competition("Race 3") {
206                     +competitor("competitor1") {
207                         +(BestTimeInMatch realized (T.toDuration(DurationUnit.
                                HOURS)))
208                         +(BestTimeInMatch realized (1.toDuration(DurationUnit.
                                HOURS)))
209                         +(BestTimeInMatch realized (2.toDuration(DurationUnit.
                                HOURS)))
210                     }
211
212                     +competitor("competitor2") {
213                         +(BestTimeInMatch realized (T.toDuration(DurationUnit.
                                HOURS)))
214                     }
215                 }
```

```
216                 -majorityVotesLScoreAlgorithm {
217                     +ConstantParameter.AllowMultipleVoteInPollParameter
218                 }
219 
220                 +("competitor2" votedBy "voter1")
221                 +("competitor1" votedBy "voter1")
222                 +("competitor2" votedBy "voter2")
223                 +("competitor1" votedBy "voter2")
224             } // same votes, same lowest score
225         }
226 
227     election.printRankings()
228 
229     // ** Ranking #1 **
230     // Competition name is Race 1
231     // Used algorithm is MajorityVotesThenLowestScoreAlgorithm
232     // Placement #1 -> [competitor2] with 2 votes
233     //      competitor2 scores list :
234     //          - Score type is BestTimeInMatch with time = 1h -
235 
236     // ** Ranking #2 **
237     // Competition name is Race 2
238     // Used algorithm is MajorityVotesThenLowestScoreAlgorithm
239     // Placement #1 -> [competitor1] with 2 votes
240     //      competitor1 scores list :
241     //          - Score type is BestTimeInMatch with time = 6m -
242     // Placement #2 -> [competitor2] with 2 votes
243     //      competitor2 scores list :
244     //          - Score type is BestTimeInMatch with time = 1h -
245 
246     //** Ranking #3 **
247     //Competition name is Race 3
248     //Used algorithm is MajorityVotesThenLowestScoreAlgorithm
249 
250     //Placement #1 -> [competitor2 | competitor1] with 2 votes
251     //      competitor2 scores list :
252     //          - Score type is BestTimeInMatch with time = 6m -
253     //      competitor1 scores list :
254     //          - Score type is BestTimeInMatch with time = 6m -
```

Sono stati realizzati test a supporto dell'utilizzo del DSL e degli algoritmi, in particolare è stato verificato che, nell'ambito di una votazione, venga restituito una eccezione nei casi in cui:

1. Nell'ambito degli algoritmi, in una determinata votazione

- un candidato sia presente più di una volta nell'insieme dei candidati;
- non siano presenti voti;
- un votante effettui più di un voto, a meno di passare all'algoritmo l'apposito parametro;
- il parametro, quando presente, sia dichiarato più di una volta;
- un votante voti due volte lo stesso candidato, seppur in presenza del parametro;
- un votante voti un candidato non esistente;
- la classifica generata non contenga alcun elemento;
- la classifica generata restituisca i risultati ordinati in modo non decrescente di voti;
- gli algoritmi che utilizzano il sotto-criterio rilevino che ci sono concorrenti senza alcun punteggio.

2. Nell'ambito del DSL

- non siano presenti voti;
- si dichiari lo stesso candidato più di una volta nell'insieme di candidati;
- si dichiari un voto per un candidato non esistente

5.2.2 Votazioni a lista di preferenze

Per gestire la votazioni a lista di preferenze è stato necessario procedere all'implementazione dell'algoritmo di Condorcet e dell'algoritmo di Schultze. Come è stato descritto in 3.2, quest'ultimo è una variante del primo, pertanto condividono buona parte della logica.

Inizialmente usando una matrice di dimensione $n\ x\ n$, dove n è il numero dei candidati, ed esaminando l'indice in cui si posizionano i candidati in un voto, si evidenzia chi dei due abbiamo ottenuto un punto. Si ottiene quindi una matrice

che indica quante volte sono stati vinti gli scontri di coppia tra due determinati candidati.

Successivamente, l'algoritmo di Condorcet effettua tante iterazioni quanti sono i concorrenti rimasti da piazzare in graduatoria, ed in ogni iterazione si esamina quali siano i candidati che hanno battuto tutti gli altri nello scontro a coppie. Anche l'algoritmo di Schultze effettua tante iterazioni quanti sono i concorrenti rimasti da piazzare in graduatoria, ed in ogni iterazione si determinano quali sono i candidati che hanno ottenuto il maggior numero di vittorie negli scontri a coppie. Pertanto, è stata ammessa la possibilità di ottenere dei pareggi a parità di posizionamento.

Anche per questa tipologia di votazione è possibile avere un parametro che indica la capacità di effettuare molteplici voti per votante , ma di negargli comunque l'opportunità di votare più volte lo stesso candidato.

Le classi realizzate per gestire le casistiche sono `CondorcetAlgorithm.kt` e `SchultzeAlgorithm.kt`. Nell'estratto di codice 5.3 vengono mostrati i controlli effettuati prima di eseguire l'algoritmo di votazione.

Estratto di codice 5.3: Controlli effettuati prima della votazione a lista di preferenze

```
1    if (candidates.groupingBy { it.name }.eachCount().any { it.value > 1 }) {
2            error("Candidate already declared")
3        }
4
5        require(votes.any())
6
7        when (pollAlgorithmParameters.count { it == ConstantParameter.
             AllowMultipleVoteInPollParameter }) {
8            0 -> {
9                if (votes.groupingBy { it.voter.identifier }.eachCount().any { it.
                     value > 1 }) {
10                   error("Each voter can vote only once")
11               }
12           }
13           1 -> {
14               // multiple vote allowed
15               if (votes.groupingBy {
16                       Pair(it.votedCompetitors.map { c -> c.name }, it.voter.
                             identifier)
17                   }.eachCount().any { it.value > 1 }
```

```
18                ) {
19                    error("Each voter can vote just once for each list of
                          preferences")
20                }
21            }
22            else -> error("Parameter can't be repeated more than once")
23        }
24
25        votes.map { it.votedCompetitors }.forEach {
26            val setOfCompetitors = it.toSet()
27
28            if (setOfCompetitors != candidates) { // mismatch between sets
29                if ((setOfCompetitors - candidates).isNotEmpty()) {
30                    error("A list of preferences contains one o more not allowed
                          candidate")
31                }
32                if ((candidates - setOfCompetitors).isNotEmpty()) {
33                    error("Every allowed candidate must be present in every list
                          of preferences")
34                }
35            }
36
37            val groupCount = it.groupingBy { comp -> comp }.eachCount()
38            // every candidate can be present only once in the list of competitors
39            if (groupCount.any { count -> count.value > 1 }) {
40                error(
41                    "Every allowed candidate can be present only once in the list
                          of competitors",
42                )
43            }
44        }
```

Infine viene generata la classifica contenente l'elenco dei vincitori, posti in ordine decrescente secondo la logica dell'algoritmo scelto. A parità di posizione è possibile trovare più di un candidato e non è previsto riportare il numero di voti, poichè è scollegato dalla logica algoritmica.

Nell'estratto di codice **??** vengono mostrati alcuni esempi realizzati: nella definizione di ogni competizione è essenziale dichiarare l'elenco di tutti i possibili candidati in modo da poter verificare che non ci siano voti associati ad un candidato non esistente. Questo controllo è effettuato anche dal DSL nella fase di definizione dei voti, e quindi in precedenza all'invocazione dell'algoritmo, che replica questo

controllo in quanto è possibile utilizzarlo indipendentemente dall'adozione di DSL. Si definisce poi l'algoritmo da utilizzare e l'elenco di voti. I voti possono essere anonimizzati, come nel caso delle votazioni a preferenza singola.

Estratto di codice 5.4: Esempi di votazioni a lista di preferenze, attraverso l'utilizzo di DSL

```
1   var counter = 1
2   val election =
3       PollManagerInstance<BestTimeInMatch, ListOfPreferencesVote<BestTimeInMatch
        >>() initializedAs {
4           +poll {
5               -competition("Sport match") {
6                   +competitor("competitorB") {
7                   }
8
9                   +competitor("competitorA") {
10                  }
11
12                  +competitor("competitorC") {
13                  }
14              }
15              -condorcetAlgorithm {}
16
17              +("competitorA" then "competitorB" then "competitorC" votedBy "
                    anonym" + counter++)
18              +("competitorA" then "competitorB" then "competitorC" votedBy "
                    anonym" + counter++)
19              +("competitorC" then "competitorA" then "competitorB" votedBy "
                    anonym" + counter++)
20              +("competitorC" then "competitorA" then "competitorB" votedBy "
                    anonym" + counter++)
21              +("competitorC" then "competitorA" then "competitorB" votedBy "
                    anonym" + counter++)
22              +("competitorC" then "competitorA" then "competitorB" votedBy "
                    anonym" + counter++)
23              +("competitorB" then "competitorC" then "competitorA" votedBy "
                    anonym" + counter++)
24              +("competitorA" then "competitorC" then "competitorB" votedBy "
                    anonym" + counter++)
25              +("competitorB" then "competitorA" then "competitorC" votedBy "
                    anonym" + counter++)
26              +("competitorC" then "competitorB" then "competitorA" votedBy "
                    anonym" + counter++)
27          } // competitorC, competitorA, competitorB
```

```
28          }
29      election.printRankings()
30          // ** Ranking #1 **
31          // Competition name is Sport match
32          // Used algorithm is CondorcetAlgorithm
33          // Placement #1 -> [competitorC]
34          //      competitorC scores list :
35          //          - no score available
36          // Placement #2 -> [competitorA]
37          //      competitorA scores list :
38          //          - no score available
39          // Placement #3 -> [competitorB]
40          //      competitorB scores list :
41          //          - no score available
42
43      counter = 1
44      election =
45          PollManagerInstance<BestTimeInMatch, ListOfPreferencesVote<BestTimeInMatch
                >>() initializedAs {
46              +poll {
47                  -competition("Sport match") {
48                      +competitor("competitorB") {
49                      }
50
51                      +competitor("competitorA") {
52                      }
53
54                      +competitor("competitorC") {
55                      }
56                  }
57                  -schultzeAlgorithm {}
58
59                  +("competitorA" then "competitorB" then "competitorC" votedBy "
                        anonym" + counter++)
60                  +("competitorA" then "competitorB" then "competitorC" votedBy "
                        anonym" + counter++)
61                  +("competitorC" then "competitorA" then "competitorB" votedBy "
                        anonym" + counter++)
62                  +("competitorC" then "competitorA" then "competitorB" votedBy "
                        anonym" + counter++)
63                  +("competitorC" then "competitorA" then "competitorB" votedBy "
                        anonym" + counter++)
64                  +("competitorC" then "competitorA" then "competitorB" votedBy "
                        anonym" + counter++)
65                  +("competitorB" then "competitorC" then "competitorA" votedBy "
                        anonym" + counter++)
```

```
66                +("competitorA" then "competitorC" then "competitorB" votedBy "
                      anonym" + counter++)
67                +("competitorB" then "competitorA" then "competitorC" votedBy "
                      anonym" + counter++)
68                +("competitorC" then "competitorB" then "competitorA" votedBy "
                      anonym" + counter++)
69            } // competitorC, competitorA, competitorB
70        }
71
72        election.printRankings()
73
74        // ** Ranking #1 **
75        // Competition name is Sport match
76        // Used algorithm is SchultzeAlgorithm
77        // Placement #1 -> [competitorC]
78        //      competitorC scores list :
79        //         - no score available
80        // Placement #2 -> [competitorA]
81        //      competitorA scores list :
82        //         - no score available
83        // Placement #3 -> [competitorB]
84        //      competitorB scores list :
85        //         - no score available
```

Sono stati realizzati test a supporto dell'utilizzo del DSL e degli algoritmi, in particolare è stato verificato che, nell'ambito di una votazione, venga restituito una eccezione nei casi in cui:

1. Nell'ambito degli algoritmi, in una determinata votazione

   - un candidato sia presente più di una volta nell'insieme dei candidati;
   - un votante effettui più di un voto, a meno di passare all'algoritmo l'apposito parametro;
   - il parametro, quando presente, sia dichiarato più di una volta;
   - un votante voti due volte la stessa lista, seppur in presenza del parametro;
   - un votante voti una lista in cui appaia un candidato non esistente;
   - un votante voti una lista in cui sia stato omesso almeno un candidato;

- un votante voti una lista in cui un candidato appaia più volte;
- non siano presenti voti;
- la classifica generata non contenga alcun elemento.

2. Nell'ambito del DSL

- si dichiari un candidato più di una volta nell'insieme di candidati;
- si dichiari un voto per un candidato non esistente;
- si dichiari un voto in cui un candidato appaia più volte;
- non siano presenti voti.

Utilizzando il DSL, è possibile dichiarare voti in cui non appaiono tutti i possibili candidati e in tal caso, questi saranno piazzati in fondo alla lista, ossia come più sfavoriti.

5.3 Generazione multipiattaforma e pubblicazione degli artefatti

Il progetto è stato realizzando partendo da un template [**?**] disponibile su GH, che fornisce la struttura di avvio per un progetto volto alla generazione finale di artefatti per molteplici piattaforme.

I target sono tutti compilati in `Java` versione 1.8, successivamente vengono anche convertiti in `Javascript` per la generazione della libreria in ambito Web, e viene generata la versione `native`, compatibile in linguaggio nativo con i seguenti sistemi:

- linux x64
- linux Arm64
- mingw x64

- macos X64
- macos Arm64
- ios Arm64
- ios X64
- iosSimulator Arm64
- watchos Arm32
- watchos X64
- watchos Simulator Arm64
- tvos Arm64
- tvos X64
- tvos Simulator Arm64

Il processo di compilazione viene eseguito grazie ai workflow presenti nel repository GH: il primo workflow, definito `dispatcher`, viene avviato al momento del ricevimento di un comando *push* o della creazione di una *richiesta pull*, e richiama in cascata i seguenti workflow

1. `build-and-deploy`: questa azione si occupa di predisporre l'ambiente di pubblicazione, compilare i sorgenti per le piattaforme, eseguire i test nelle piattaforme, effettuare controlli di `code linting` e `bug detection`, ed infine pubblicare gli artefatti generati e finalizzare l'ambiente di pubblicazione;

2. `publish-docs`: questa azione è dipendente da build-and-deploy, in quanto viene eseguita successivamente ad essa e solo nel caso in cui sia terminata con esito positivo. Si occupa di generare la documentazione degli artefatti, interpretando gli appositi commenti scritti nel codice sorgente, e caricarla nel branch `gh-pages` del repository GH. Un workflow di default elabora il

contenuto di questo branch e lo mette a disposizione tramite un webserver, accessibile all'indirizzo https://corinz97.github.io/EleKtion/ .

5.3.1 Generazione e pubblicazione della libreria

Il workflow `build-and-deploy` si occupa inizialmente di richiamare un'apposita `action` [**?**], che calcola la versione da associare alla libreria per il rilascio, ed inizializzare un repository di stage su Maven Central. e memorizzare l'id generato dal servizio. Ciò è fatto mediante l'uso di una `action` preposta alla compilazione e pubblicazione di progetti basati su Gradle [**?**].

Successivamente viene lanciata il task `gradle build` utilizzando una matrice di tre sistemi operativi `windows-2022`, `macos-12`, `ubuntu-22.04`. Ciò permette di compilare il codice sorgente, eseguire i test, effettuare i controlli di `code linting` e `bug detection` e generare gli artefatti in ciascun ambiente. Questi sono generati basandosi sui target definiti nella lista al paragrafo 5.3; ciascun ambiente di compilazione, a seconda del sistema operativo in esecuzione, genera un *subset* di artefatti e li carica sul repository di stage.

Infine, se la libreria deve essere pubblicata nel branch `master` del repository GH, e la action [**?**] abbia calcolato che la versione è candidabile al rilascio, viene chiuso e finalizzato il repository di stage su Maven Central, Altrimenti questo viene chiuso e scartato.

La finalizzazione del repository porta all'effettiva pubblicazione su Maven Central e si prosegue a pubblicare gli artefatti anche su NPM e GH Packages, i quali non seguono l'iter di creazione del repository di stage ma operano con una pubblicazione diretta.

Solo in quest'ultimo caso di successo si prosegue ad invocare il workflow `publish-docs`

5.3.2 Generazione e pubblicazione della documentazione

spiegare la creazione di un nuovo workflow, e collegato a quelli esistenti

Capitolo 6

Sperimentazione con i dati di Formula 1

La sperimentazione della libreria `EleKtion` è stata estesa ad un caso di studio reale, applicando gli algoritmi di votazione ai risultati delle gare di *F1*.

I dati sono accessibili attraverso un API REST pubblica chiamata Ergast API [**?**], disponibile per scopi non-commerciali, che mette a disposizione dati inerenti ai mondiali di *F1* a partire dall'anno 1950. Per ognuno di questi sono a disposizione numerose informazioni come l'elenco delle gare ufficiali (Gran Premi), le gare di qualifica, le classifiche generali dei piloti, lo storico dei piloti, le classifiche generali dei costruttori e molto altro.

Allo scopo di questa tesi, sono risultate utili le informazioni riguardo ai risultati delle gare ufficiali e le classifiche generali dei piloti al termine di ogni gara, entrambe legate ad uno specifico anno.

Attaverso appositi endpoint, è stato possibile ottenere i dati necessari per predisporre le informazioni necessarie ad `EleKtion`, utilizzando gli esiti delle varie gare come voti e applicando di conseguenza gli algoritmi a lista di preferenze. L'elemento più favorito nel voto è rappresentato dal pilota che si è classificato primo in gara, mentre l'elemento più sfavorito è il pilota classificatosi ultimo in gara.

Nei seguenti sottocapitoli vengono mostrati i passaggi effettuati per elaborare i

dati di un mondiale, in particolare vengono generate, come output delle votazioni, le classifiche parziali al termine di ogni gara in modo da avere uno storico, che verrà confrontato con lo storico delle classifiche reali. Verrà infine svolto un calcolo utile ad indicare la differenza, in termini qualitativi, confrontando i dati ottenuti.

6.1 Elaborazione dati di un mondiale *F1*

Si è provveduto ad ottenere l'elenco delle gare svoltesi nel corso del mondiale AAAA (anno del mondiale), disponibili all'endpoint di Ergast API https://ergast.com/api/f1/AAAA.json.

In questo mondiale sono state effettuate in totale k gare, per ognuna di esse sono stati scaricati i relativi dati, filtrandoli per tenere in considerazione solamente il nominativo del pilota e la posizione di arrivo. Eventuali ritiri a causa di guasti o altri motivi non sono gestiti, ciascun pilota in ogni caso ha attribuita una posizione di arrivo.

I dati sono associati ad una mappa `Map<String, List<Pair<String, Int>>>`, nella quale le chiavi rappresentano il nome della gara (es. Australian-Grand-Prix-1-AAAA, Malaysian-Grand-Prix-2-AAAA, ecc...) mentre i valori rappresentano liste, di tuple contenenti il nominativo del pilota e la posizione di arrivo.

Come mostrato nell'estratto di codice 6.1, si organizzano tante votazioni indipendenti quante sono le gare presenti nel mondiale: viene definito l'insieme dei nominativi presenti nel mondiale, in modo da avere il censimento di tutti coloro che hanno partecipato almeno in una gara.Si definisce l'algoritmo da utilizzare, e in questo caso di studio sono ammissibili l'algoritmo di Condorcet o l'algoritmo di Schultze. Successivamente si determina l'elenco dei voti, in cui i votati sono costituiti dall'elenco dei nominativi dei piloti secondo l'ordine di arrivo reale (dal primo all'ultimo), e come votante viene utilizzato il nome del circuito della gara. Se uno o più piloti sono assenti in quella gara ma presenti in altre, tramite un confronto con l'insieme dei candidati vengono identificati e posti "virtualmente" ad ultimi in modo da avere la lunghezza della lista di preferenze uniforme al numero

dei candidati. Infine, questa votazione viene inserita nell'elenco delle votazioni da elaborare.

Per la seconda votazione viene considerata anche la seconda gara, e mantenendo stabile l'insieme dei candidati e l'algoritmo selezionato, vengono considerati come voti l'elenco dei nominativi (secondo l'ordine di arrivo reale) della prima gara e l'elenco dei nominativi (secondo l'ordine di arrivo reale) della seconda gara. Proseguendo nelle iterazioni, si arriva ad avere k votazioni indipendenti, in cui l'ultima conterrà come voti gli esiti dalla prima alla k-esima gara.

La generazione delle classifiche parziali fornisce una serie di risultati in cui il primo consiste nella classifica provvisoria al termine della prima gara, il secondo nella classifica provvisoria al termine della seconda gara, fino ad arrivare alla classifica finale.

I dati delle classifiche reali sono ottenuti contattando l'endpoint https://ergast.com/api/f1/AAAA/GP/driverStandings.json, dove AAAA è l'anno del mondiale e GP è il numero della gara

Estratto di codice 6.1: Generazione delle classifiche parziali utilizzando `EleKtion`

```
1  val raceResults : Map<String, List<Pair<String, Int>>>() = ... //get data from
       Ergast API
2
3  val validCompetitors = raceResults.flatMap { it.value }.groupBy({ it.first },
       { it.second })
4
5  val allCompetitorNames = validCompetitors.keys.fold(setOf<String>()) { s,
       element -> s + element }
6
7  var election =
8          PollManagerInstance<Nothing, ListOfPreferencesVote<Nothing>>()
               initializedAs {
9              var currentGPs: Map<String, List<Pair<String, Int>>> = mapOf()
10             var index = 1
11             for (raceResult in raceResults) {
12                 // add incrementally a new GP, for each poll
13                 currentGPs = currentGPs + raceResult.toPair()
14                 +poll {
15                     -competition("F1 Pilots - Partial Ranking - GPs #1 to #${
                           index++}") {
16                         allCompetitorNames.forEach {
```

```
17                              +competitor(it) {}
18                          }
19                      }
20                      -chosenLOPAlgorithm{} //-condorcetAlgorithm{} or -
                            schultzeAlgorithm
21
22                      // incremental set of votes
23                      currentGPs.entries.forEach { (runningField, competitors)
                            ->
24                          +(
25                              competitors.fold(listOf<String>()) { l, element ->
                                    l then element.first }
26                                  votedBy runningField
27                              )
28                      }
29                  }
30              }
31          }
32          var rankings = election.computeAllPolls()
```

Per confrontare gli esiti ottenuti impiegando i differenti algoritmi e i dati reali estratti dall'API, è necessario organizzarli secondo matrici r x k, dove ogni riga r identifica un pilota, e ogni colonna k identifica una gara. Nell'intersezione è presente la posizione nella classifica provvisoria (reale o generata) ottenuta dal r-esimo pilota al termine della k-esima gara.

Per effettuare un confronto qualitativo tra matrici è fondamentale organizzarle in modo tale che le righe e le colonne siano ordinate allo stesso modo. Nel caso delle righe è possibile adottare un ordinamento lessicografico o l'ordinamento presente in una delle matrici, che viene preso come riferimento per le altre. Nella sperimentazione effettuata, sono state calcolate per prime le classifiche che impiegano l'algoritmo di Condorcet, poi la sequenza dei piloti nella classifica finale è stato preso come riferimento di ordinamento. Le classifiche ottenute con l'algoritmo di Schultze sono state organizzate secondo lo stesso riferimento e così anche le classifiche reali. Si rimanda agli estratti di codice 6.2, 6.3, 6.4 per approfondire l'implementazione di questi criteri.

Esistono numerose metriche di confronto tra matrici [**?**], in questo caso si è optato per la *norma di Frobenius*, rappresentata dalla formula $\sqrt{\sum_{i,j} abs(a_{i,j})^2}$,

dove $a_{i,j}$ è l'elemento di una matrice A ricavata dalla differenza tra le matrici trattate.

Estratto di codice 6.2: Ordinamento propedeutico al confronto (Condorcet)

```
//Condorcet
//flatten, in order to extract ties from sets
val flattenedOrdersInRankings = rankings.map { it.ranking.flatMap { it.key.map
    { it.name } } }
// take final standing
val condorcetFlattenedOrderInFinalRanking = flattenedOrdersInRankings.last()

var m: Map<String, List<Int>> = mapOf()
for (competitor in condorcetFlattenedOrderInFinalRanking) {
    var listOfPositionsPerCompetitor: List<Int> = listOf()
    for (ranking in rankings.map { it.ranking.keys.toList() }) {
        var index = ranking.indexOfFirst { it.map { it.name }.contains(
            competitor) }
        listOfPositionsPerCompetitor = listOfPositionsPerCompetitor + ++index
    }
    m = m + (competitor to listOfPositionsPerCompetitor)
}
```

Estratto di codice 6.3: Ordinamento propedeutico al confronto (Schultze)

```
//Schultze
//reuse same order
val schultzeFlattenedOrderInFinalRanking =
    condorcetFlattenedOrderInFinalRanking
var m: Map<String, List<Int>> = mapOf()
for (competitor in schultzeFlattenedOrderInFinalRanking) {
    var listOfPositionsPerCompetitor: List<Int> = listOf()
    for (ranking in rankings.map { it.ranking.keys.toList() }) {
        var index = ranking.indexOfFirst { it.map { it.name }.contains(
            competitor) }

        listOfPositionsPerCompetitor = listOfPositionsPerCompetitor + ++index
    }
    m = m + (competitor to listOfPositionsPerCompetitor)
}
```

Estratto di codice 6.4: Ordinamento propedeutico al confronto (Dati reali)

```
//Real
//reuse same order
```

```
3    val realWorldFlattenedOrderInFinalRanking =
         condorcetFlattenedOrderInFinalRanking
4     var m: Map<String, List<Int>> = mapOf()
5     for (competitor in realWorldFlattenedOrderInFinalRanking) {
6         var listOfPositionsPerCompetitor: List<Int> = listOf()
7
8         for (ranking in raceResults.map { it.value }) {
9             val index = ranking.firstOrNull { it.first == competitor }?.second
10             ?: ranking.maxOf { it.second } //managed "virtually" present
                   competitors
11
12            listOfPositionsPerCompetitor = listOfPositionsPerCompetitor + index
13        }
14        m = m + (competitor to listOfPositionsPerCompetitor)
15    }
```

6.2 Elaborazione del mondiale *F1* 2023

Si è provveduto alla generazione delle classifiche provvisorie utilizzando i dati del mondiale *F1* 2023.

Si riporta in tabella 6.1 la classifica finale reale e quelle finali usando l'algoritmo di Condorcet e l'algoritmo di Schultze

Posizione	Reali	Condorcet	Schultze
1.	Max-Verstappen	Max-Verstappen (C1.)	Max-Verstappen
2.	Sergio-Pérez	Sergio-Pérez (C2.)	Sergio-Pérez
3.	Lewis-Hamilton	Lewis-Hamilton (C3.)	Lewis-Hamilton
4.	Fernando-Alonso	Fernando-Alonso (C4.) Carlos-Sainz (C5.) Charles-Leclerc (C6.)	Fernando-Alonso
5.	Charles-Leclerc	Lando-Norris (C7.)	Carlos-Sainz
6.	Lando-Norris	George-Russell (C8.)	Lando-Norris
7.	Carlos-Sainz	Lance-Stroll (C9.) Oscar-Piastri (C10.)	George-Russell
8.	George-Russell	Pierre-Gasly (C11.)	Charles-Leclerc
9.	Oscar-Piastri	Alexander-Albon (C12.) Esteban-Ocon (C13.)	Pierre-Gasly
10.	Lance-Stroll	Yuki-Tsunoda (C14.)	Oscar-Piastri
11.	Pierre-Gasly	Valtteri-Bottas (C15.) Nico-Hülkenberg (C16.)	Lance-Stroll
12.	Esteban-Ocon	Guanyu-Zhou (C17.)	Esteban-Ocon
13.	Alexander-Albon	Kevin-Magnussen (C18.)	Alexander-Albon
14.	Yuki-Tsunoda	Logan-Sargeant (C19.)	Yuki-Tsunoda
15.	Valtteri-Bottas	Nyck-de Vries (C20.)	Valtteri-Bottas
16.	Nico-Hülkenberg	Daniel-Ricciardo (C21.)	Guanyu-Zhou
17.	Daniel-Ricciardo	Liam-Lawson (C22.)	Nico-Hülkenberg
18.	Guanyu-Zhou	Assente	Kevin-Magnussen
19.	Kevin-Magnussen	Assente	Logan-Sargeant
20.	Liam-Lawson	Assente	Daniel-Ricciardo
21.	Logan-Sargeant	Assente	Nyck-de Vries
22.	Nyck-de Vries	Assente	Liam-Lawson

Tabella 6.1: Classifica finale reale e classifiche finali ottenute con l'algoritmo di Condorcet e l'algoritmo di Schultze, relativamente al mondiale *F1* 2023. Nella colonna relativa all'algoritmo di Condorcet viene fornito un acronimo ad ogni concorrente.

Nelle tabelle 6.2, 6.3 e 6.4 vengono mostrate le matrici complete con il posizionamento dei piloti in ogni classifica provvisoria.

Per motivi prettamente grafici, a ciascun concorrente è stato assegnato un acronimo, consultabile nella terza colonna della tabella 6.1 Infine, per ciascuna tabella è stato generato il relativo grafico, nel quale le linee rappresentano i concorrenti, l'asse delle ascisse rappresenta il numero sequenziale della gara e l'asse delle ordinate è associato al posizionamento in classifica al termine della relativa gara. Per la tabella 6.2 il grafico di riferimento è in figura 6.1, mentre per le tabelle 6.3 e 6.4 i grafici sono rispettivamente in figura 6.2 ed in figura 6.3.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18	G19	G20	G21	G22
C1.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
C2.	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
C3.	5	5	4	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3
C4.	3	3	3	3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	5	4	5	4
C5.	4	4	5	5	5	6	6	5	5	5	6	7	5	5	5	5	5	5	4	6	4	7
C6.	19	8	10	6	7	7	7	7	6	7	7	5	6	6	6	6	6	7	7	7	7	5
C7.	17	20	8	9	9	11	11	11	10	9	8	8	8	8	8	7	7	6	6	5	6	6
C8.	7	6	7	7	6	5	5	6	7	6	5	6	7	7	7	8	8	8	8	8	8	8
C9.	6	7	6	8	8	8	8	8	8	8	9	9	9	9	9	10	10	11	11	10	10	10
C10.	20	19	13	11	14	13	13	14	14	11	11	11	12	12	11	9	9	9	9	9	9	9
C11.	9	11	14	14	10	10	10	10	11	12	12	12	10	10	10	11	11	10	10	11	11	11
C12.	10	13	18	17	18	18	18	12	13	13	13	13	13	13	13	13	13	13	13	13	13	13
C13.	18	10	12	13	12	9	9	9	9	10	10	10	11	11	12	12	12	12	12	12	12	12
C14.	11	14	16	16	16	16	16	17	17	17	17	17	17	17	17	17	17	16	16	14	14	14
C15.	8	9	11	12	13	14	14	15	15	15	15	15	15	15	15	15	14	14	14	15	15	15
C16.	15	15	9	10	11	12	12	13	12	14	14	14	14	14	14	14	15	15	15	16	16	16
C17.	16	17	15	15	15	15	15	16	16	16	16	16	16	16	16	16	16	17	18	18	18	18
C18.	13	12	17	18	17	17	17	18	18	18	18	18	18	18	18	18	18	18	19	19	19	19
C19.	12	16	19	19	19	20	20	20	19	19	19	19	19	19	20	20	20	20	21	21	21	21
C20.	14	18	20	20	20	19	19	19	20	20	20	20	20	21	21	21	21	21	22	22	22	22
C21.	20	20	20	20	20	20	20	20	20	20	21	21	21	22	22	22	22	22	17	17	17	17
C22.	20	20	20	20	20	20	20	20	20	20	21	21	22	20	19	19	19	19	20	20	20	20

Tabella 6.2: Tabella relativa ai valori reali del Mondiale *F1* 2023, in cui per ogni gara (G*) è riportata la posizione del concorrente (C*) in classifica generale, al termine della gara stessa. I valori sono ricavati da Ergast API [**?**]

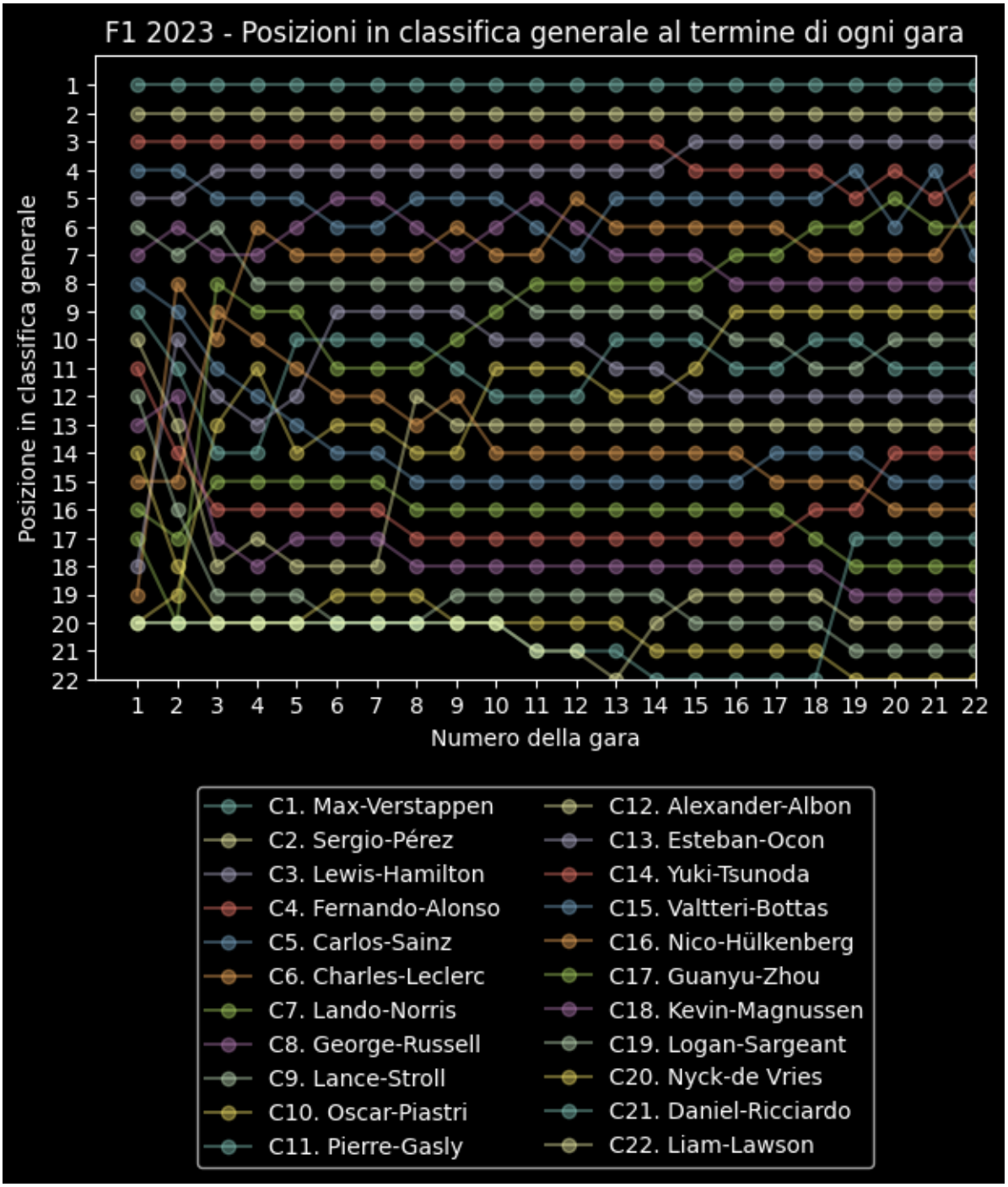

Figura 6.1: Rappresentazione grafica della tabella 6.2

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18	G19	G20	G21	G22
C1.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
C2.	2	1	2	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
C3.	5	3	4	3	5	4	4	3	4	3	4	4	4	4	4	3	4	4	4	3	4	3
C4.	3	2	3	2	3	3	3	2	3	2	3	3	3	3	3	3	3	3	3	3	3	4
C5.	4	3	5	3	4	5	6	4	5	5	7	7	7	6	5	4	5	5	5	4	5	4
C6.	19	10	14	11	6	7	7	6	7	5	6	6	6	6	7	5	5	7	5	6	5	4
C7.	17	11	13	7	11	10	11	10	11	8	10	9	8	7	8	6	6	6	6	5	6	5
C8.	7	4	7	4	6	6	5	5	6	4	5	5	5	5	6	5	7	6	7	5	7	6
C9.	6	11	6	4	6	9	7	7	8	6	8	8	8	7	9	9	9	10	9	9	8	7
C10.	20	11	14	8	15	12	12	10	13	10	11	12	9	10	11	8	8	9	8	8	9	7
C11.	9	5	8	6	7	8	8	8	9	7	9	10	8	8	10	7	8	8	8	7	8	8
C12.	10	11	14	11	13	13	16	12	14	11	11	13	9	9	11	8	9	10	9	9	10	9
C13.	18	10	11	9	9	9	9	8	10	9	11	11	9	9	11	9	9	11	9	10	11	9
C14.	11	6	8	5	8	9	10	9	12	12	13	13	11	12	13	11	11	13	11	11	13	10
C15.	8	10	11	9	12	11	13	11	13	11	12	13	10	11	12	10	10	12	10	12	12	11
C16.	15	8	9	7	10	11	11	12	15	13	14	14	12	13	14	12	12	14	12	12	14	11
C17.	16	9	10	10	13	12	14	13	15	13	15	14	13	14	15	13	13	15	13	13	15	12
C18.	13	7	12	9	9	12	13	13	15	15	16	15	14	15	16	14	14	16	14	14	16	13
C19.	12	10	13	11	16	14	17	15	17	15	18	16	15	16	17	15	15	17	15	15	17	14
C20.	14	9	12	11	14	13	15	14	16	14	17	16	16	17	18	16	16	18	16	16	18	15
C21.	21	12	15	12	17	15	18	16	18	16	19	17	17	18	19	17	17	19	17	17	19	16
C22.	22	13	16	13	18	16	19	17	19	17	20	18	18	19	20	18	18	20	18	18	20	17

Tabella 6.3: Tabella relativa al Mondiale *F1* 2023, in cui per ogni gara (G*) è riportata la posizione del concorrente (C*) in classifica generale, al termine della gara stessa. I valori sono ricavati dall'applicazione dell'algoritmo di Condorcet al termine di ogni gara, considerando anche le precedenti. Ogni gara rappresenta un voto, composto dagli ordini di arrivo.

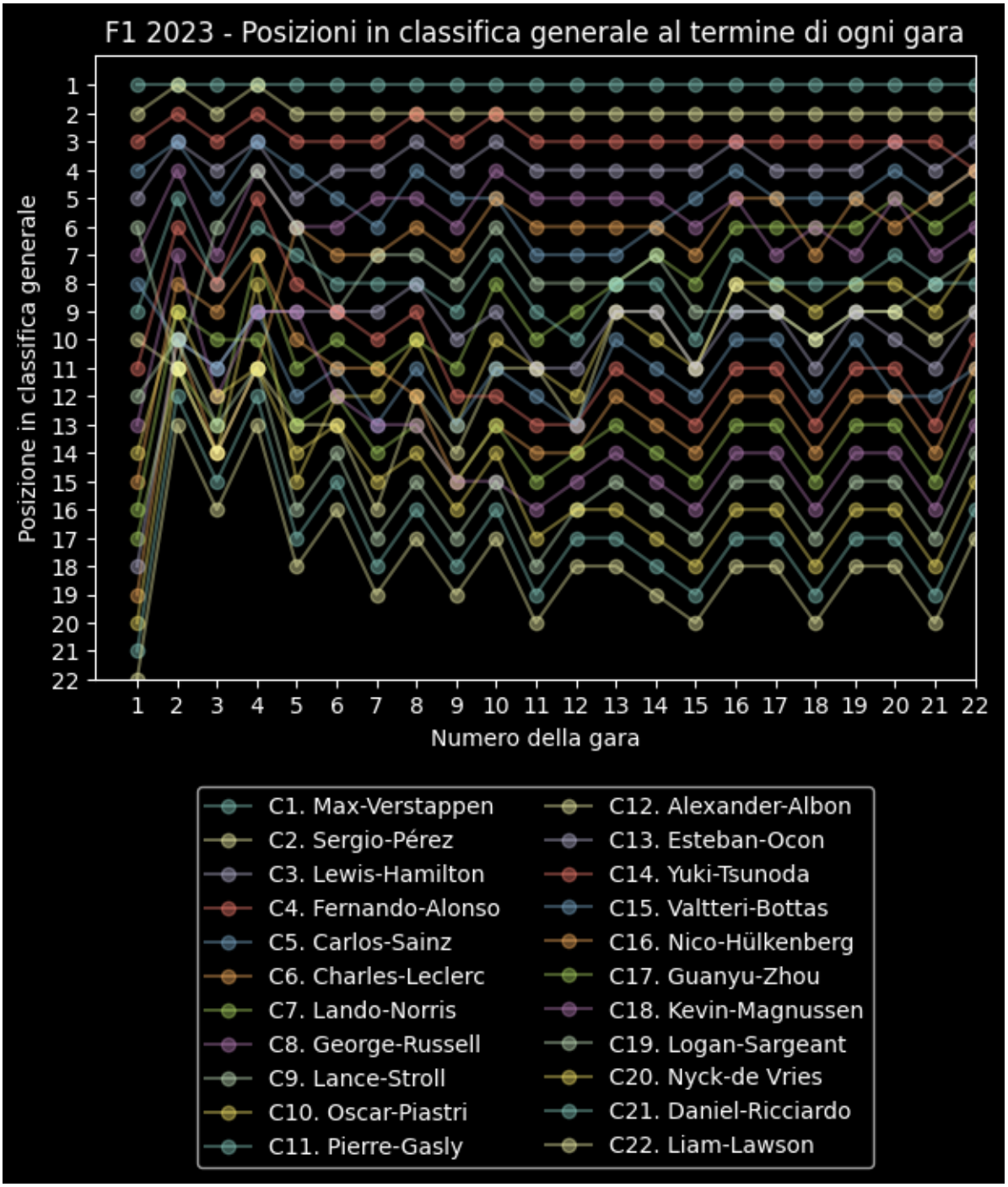

Figura 6.2: Rappresentazione grafica della tabella 6.3

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18	G19	G20	G21	G22
C1.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
C2.	2	1	2	2	2	3	4	4	3	4	3	2	2	2	2	3	4	2	3	3	2	2
C3.	5	3	4	4	4	4	3	3	4	3	2	3	3	3	3	2	2	4	2	2	3	3
C4.	3	2	3	3	3	2	2	2	2	2	2	4	2	3	4	4	3	3	4	4	4	4
C5.	4	3	5	5	5	5	5	5	5	5	4	6	4	4	5	5	5	5	5	5	5	5
C6.	19	8	16	9	9	8	8	7	7	7	6	7	6	6	6	6	6	8	8	8	8	8
C7.	17	12	13	9	12	12	11	12	11	10	7	8	7	7	8	8	8	6	6	6	6	6
C8.	7	4	6	6	6	6	6	6	6	6	5	5	5	5	7	7	7	7	7	7	7	7
C9.	6	8	7	6	7	11	9	10	9	8	8	9	8	8	10	11	11	11	11	11	11	11
C10.	20	13	14	12	14	15	13	13	14	12	10	12	11	11	11	10	9	10	10	10	10	10
C11.	9	5	8	8	8	7	7	8	8	9	9	10	9	9	9	9	10	9	9	9	9	9
C12.	10	11	17	15	15	17	16	15	13	14	12	14	12	10	12	12	13	12	12	13	13	13
C13.	18	8	13	13	11	9	9	9	10	11	11	11	10	11	13	13	12	13	13	12	12	12
C14.	11	6	9	7	8	10	10	11	12	13	13	13	13	13	15	14	16	15	14	14	14	14
C15.	8	8	11	13	13	13	14	14	14	15	14	15	14	12	14	14	14	14	14	15	15	15
C16.	15	9	10	10	12	15	14	16	15	17	16	17	16	15	17	16	17	17	16	17	17	17
C17.	16	11	12	14	14	16	12	15	14	16	15	16	15	14	16	15	15	16	15	16	16	16
C18.	13	7	13	11	10	14	15	17	16	18	17	18	17	16	18	17	18	18	17	18	18	18
C19.	12	10	15	15	16	19	18	19	18	20	18	19	18	17	19	18	19	19	18	19	19	19
C20.	14	10	14	16	17	18	17	18	17	19	18	20	19	18	20	19	20	20	19	20	20	21
C21.	21	14	18	17	18	20	19	20	19	21	19	21	20	19	22	21	22	22	21	21	21	20
C22.	22	15	19	18	19	21	20	21	20	22	20	22	21	20	21	20	21	21	20	22	22	22

Tabella 6.4: Tabella relativa al Mondiale *F1* 2023, in cui per ogni gara (G*) è riportata la posizione del concorrente (C*) in classifica generale, al termine della gara stessa. I valori sono ricavati dall'applicazione dell'algoritmo di Schultze al termine di ogni gara, considerando anche le precedenti. Ogni gara rappresenta un voto, composto dagli ordini di arrivo.

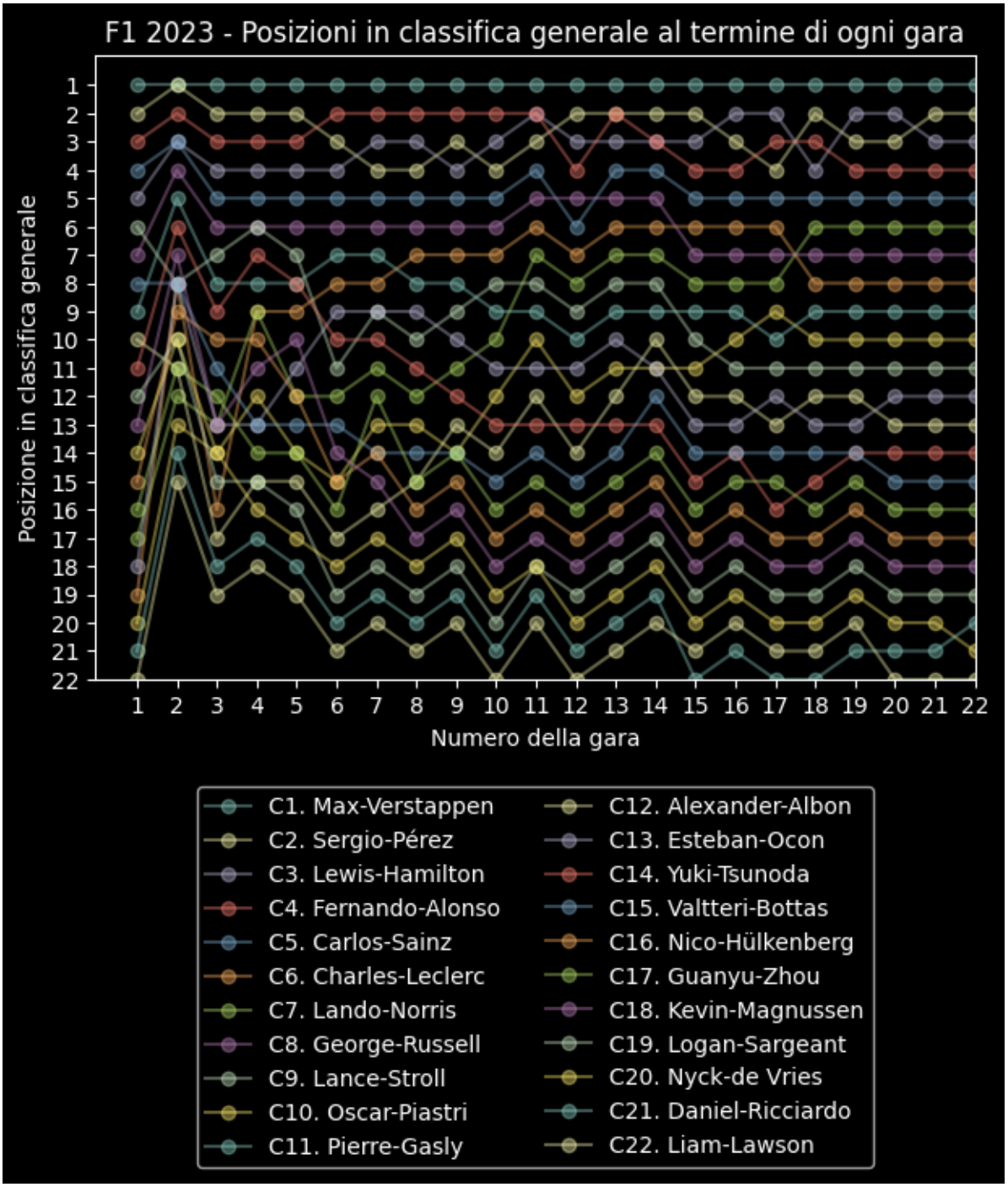

Figura 6.3: Rappresentazione grafica della tabella 6.4

Si può notare che entrambi gli algoritmi impiegati tendono ad una soluzione iniziale che porta i concorrenti ad essere molto simili come posizionamento nelle prime fasi del campionato, causando numerosi pareggi ma proseguendo con le gare questi tendono a stabilizzarsi per via del numero incrementale di voti. Tuttavia, i risultati evidenziati da Condorcet nell'ultima classifica manifestano numerosi pareggi, mentre Schultze riesce a discriminare in maniera più accurata gli esiti.

In tabella 6.5 vengono mostrati i valori di distanza ottenuti applicando la *norma di Frobenius* alla differenza delle matrici rappresentate nelle prime due colonne. Si può notare che l'insieme delle classifiche generate sono significativamente diverse dall'insieme delle classifiche reali, mentre se ci si focalizza sulle classifiche finali la distanza è minore ma i valori generati da Condorcet continuano a distinguersi maggiormente, soprattutto a causa dei pareggi, che nella classifica finale reale non sono presenti.

Elemento 1	Elemento 2	Valore distanza euclidea
Matrice "reale"	Matrice Condorcet	68.23
Matrice "reale"	Matrice Schultze	44.72
Classifica finale "reale"	Classifica finale Condorcet	14.35
Classifica finale "reale"	Classifica finale Schultze	8.31

Tabella 6.5: Campionato *F1* 2023 - Valori di distanza ottenuti applicando la *norma di Frobenius* alla differenza delle matrici rappresentate nelle prime due colonne.

6.3 Elaborazione del mondiale *F1* 2008

È stata effettuata l'elaborazione del mondiale *F1* 2008, che rispetto al mondiale 2023 ha visto una maggior competizione tra i concorrenti per le prime posizioni. Si è provveduto a generare le classifiche provvisorie replicando il flusso rappresentato in .

Si riporta in tabella 6.6 la classifica finale reale ed i risultati ottenuti nelle classifiche finali, usando l'algoritmo di Condorcet e l'algoritmo di Schultze.

Posizione	Reali	Condorcet	Schultze
1.	Lewis-Hamilton	Felipe-Massa (C1.)	Lewis-Hamilton
2.	Felipe-Massa	Lewis-Hamilton (C2.)	Robert-Kubica
3.	Kimi-Räikkönen	Kimi-Räikkönen (C3.) Robert-Kubica (C4.)	Felipe-Massa
4.	Robert-Kubica	Nick-Heidfeld (C5.)	Nick-Heidfeld
5.	Fernando-Alonso	Heikki-Kovalainen (C6.)	Kimi-Räikkönen
6.	Nick-Heidfeld	Fernando-Alonso (C7.)	Fernando-Alonso
7.	Heikki-Kovalainen	Jarno-Trulli (C8.)	Heikki-Kovalainen
8.	Sebastian-Vettel	Timo-Glock (C9.) Mark-Webber (C10.)	Jarno-Trulli
9.	Jarno-Trulli	Nico-Rosberg (C11.) Sebastian-Vettel (C12.)	Mark-Webber
10.	Timo-Glock	David-Coulthard (C13.)	Timo-Glock
11.	Mark-Webber	Kazuki-Nakajima (C14.) Rubens-Barrichello (C15.)	Nico-Rosberg
12.	Nelson-Piquet Jr.	Sébastien-Bourdais (C16.) Nelson-Piquet Jr. (C17.) Jenson-Button (C18.)	Sebastian-Vettel
13.	Nico-Rosberg	Giancarlo-Fisichella (C19.)	Kazuki-Nakajima
14.	Rubens-Barrichello	Adrian-Sutil (C20.)	Nelson-Piquet Jr.
15.	Kazuki-Nakajima	Takuma-Sato (C21.)	David-Coulthard
16.	David-Coulthard	Anthony-Davidson (C22.)	Jenson-Button
17.	Sébastien-Bourdais	Assente	Rubens-Barrichello
18.	Jenson-Button	Assente	Sébastien-Bourdais
19.	Giancarlo-Fisichella	Assente	Giancarlo-Fisichella
20.	Adrian-Sutil	Assente	Adrian-Sutil
21.	Takuma-Sato	Assente	Takuma-Sato
22.	Anthony-Davidson	Assente	Anthony-Davidson

Tabella 6.6: Classifica finale reale e classifiche finali ottenute con l'algoritmo di Condorcet e l'algoritmo di Schultze, relativamente al mondiale *F1* 2008

Nelle tabelle 6.7, 6.8 e 6.9 vengono mostrate le matrici complete con il posizionamento dei piloti in ogni classifica provvisoria.

Per motivi prettamente grafici, a ciascun concorrente è stato assegnato un acronimo, consultabile nella terza colonna della tabella 6.6 Infine, per ciascuna tabella è stato generato il relativo grafico, nel quale le linee rappresentano i concorrenti, l'asse delle ascisse rappresenta il numero sequenziale della gara e l'asse delle ordinate è associato al posizionamento in classifica al termine della relativa gara. Per la tabella 6.7 il grafico di riferimento è in figura 6.4, mentre per le tabelle 6.8 e 6.9 i grafici sono rispettivamente in figura 6.5 ed in figura 6.6.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18
C1.	8	18	6	4	2	3	3	1	2	2	3	2	2	2	2	2	2	2
C2.	1	1	3	2	3	1	2	4	1	1	1	1	1	1	1	1	1	1
C3.	8	2	1	1	1	2	4	3	3	3	2	3	4	4	4	4	4	3
C4.	8	5	4	3	4	4	1	2	4	4	4	4	3	3	3	3	3	4
C5.	2	3	2	5	5	5	5	5	5	5	5	6	5	5	5	5	5	6
C6.	5	4	5	6	6	6	6	6	6	6	6	5	6	6	6	6	7	7
C7.	4	7	9	10	8	8	9	9	9	9	8	8	8	7	7	7	6	5
C8.	8	8	7	7	9	9	8	7	7	7	7	7	7	8	9	9	9	9
C9.	8	18	14	14	15	17	13	13	14	16	10	10	10	11	10	11	10	10
C10.	8	11	10	8	7	7	7	8	8	8	9	9	9	10	11	10	11	11
C11.	3	6	8	9	10	10	10	10	11	12	13	13	14	14	12	13	13	13
C12.	8	18	21	21	22	12	14	14	15	15	16	14	12	9	8	8	8	8
C13.	8	12	13	15	14	16	12	12	13	14	15	16	16	16	16	16	16	16
C14.	6	9	11	11	11	11	11	11	12	13	14	15	15	15	15	15	15	15
C15.	8	16	16	17	17	14	15	15	10	10	12	12	13	13	14	14	14	14
C16.	7	10	12	13	13	15	17	18	17	18	18	18	17	17	17	17	17	17
C17.	8	14	17	18	18	19	19	17	18	11	11	11	11	12	13	12	12	12
C18.	8	13	15	12	12	13	16	16	16	17	17	17	18	18	18	18	18	18
C19.	8	15	18	16	16	18	18	19	19	19	19	19	19	19	19	19	19	19
C20.	8	18	21	21	21	22	22	22	22	21	21	21	20	20	20	20	20	20
C21.	8	18	20	19	19	20	20	20	20	20	20	20	21	21	21	21	21	21
C22.	8	17	19	20	20	21	21	21	21	22	22	22	22	22	22	22	22	22

Tabella 6.7: Tabella relativa al Mondiale *F1* 2008, in cui per ogni gara (G*) è riportata la posizione del concorrente (C*) in classifica generale, al termine della gara stessa. I valori sono ricavati da Ergast API [**?**]

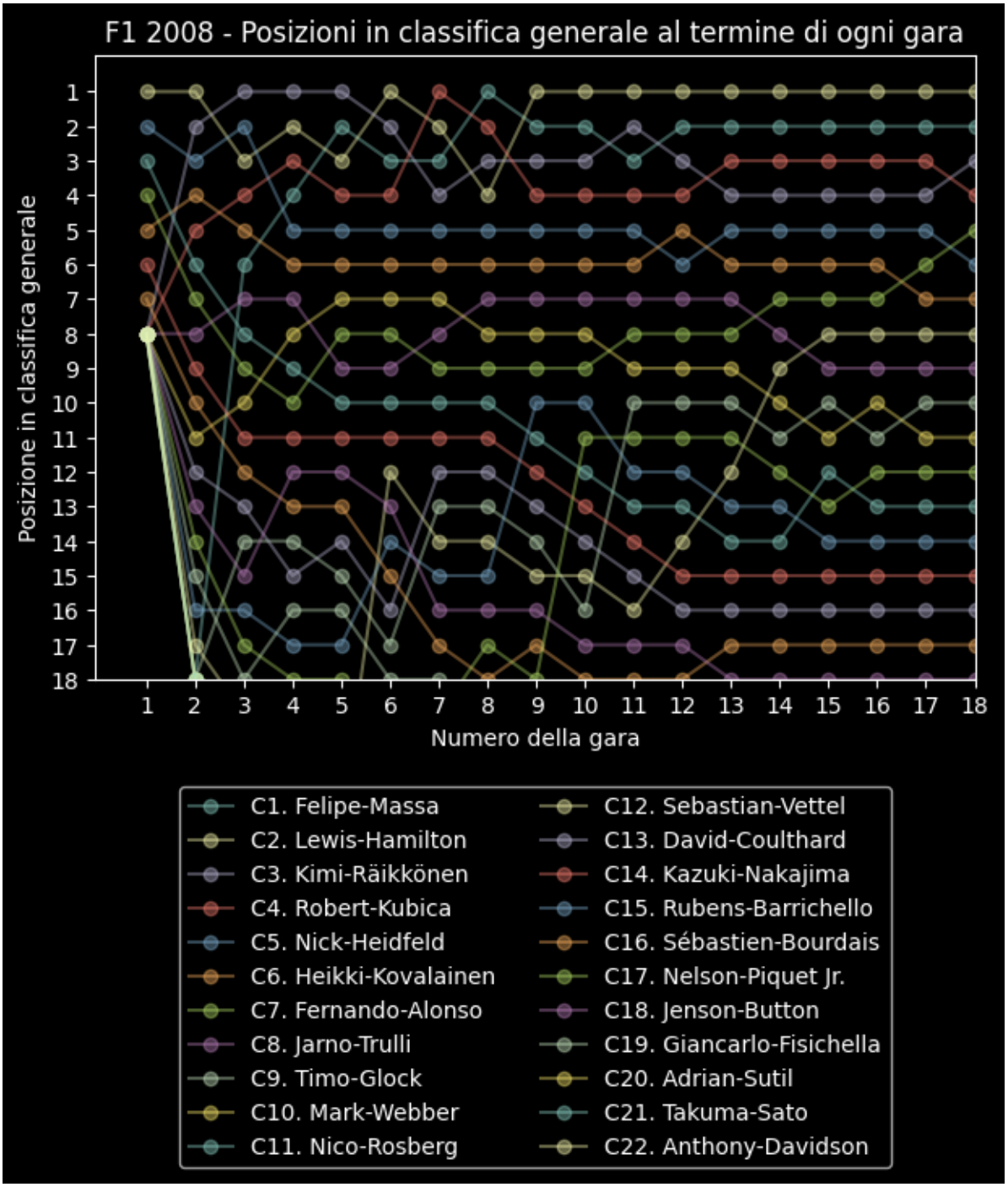

Figura 6.4: Rappresentazione grafica della tabella 6.7

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18
C1.	13	12	8	6	2	2	1	1	1	1	1	1	1	1	1	1	1	1
C2.	1	1	4	3	3	1	2	3	3	2	1	2	2	2	2	1	2	2
C3.	8	4	1	1	1	1	1	1	2	2	1	3	3	5	5	4	3	3
C4.	9	5	2	2	4	2	3	2	4	3	2	3	3	3	3	2	3	3
C5.	2	2	3	3	5	3	4	3	5	4	3	4	4	4	3	3	4	4
C6.	5	3	3	4	9	5	5	4	6	5	4	4	5	4	4	3	5	5
C7.	4	5	6	5	8	6	9	6	9	8	7	7	8	7	6	5	6	6
C8.	15	7	4	4	6	6	6	4	7	6	5	5	6	6	7	6	7	7
C9.	10	13	7	7	11	8	11	8	11	10	9	9	9	8	9	8	9	8
C10.	17	9	5	4	7	4	7	5	8	7	6	6	7	7	8	7	8	8
C11.	3	6	5	6	7	7	8	7	10	9	8	8	10	9	10	9	10	9
C12.	20	13	16	13	19	14	19	14	16	13	13	13	13	11	12	10	10	9
C13.	14	8	9	8	12	9	12	7	12	10	10	10	11	10	11	11	11	10
C14.	6	9	7	7	11	8	11	10	11	10	11	12	12	11	12	12	12	11
C15.	22	13	12	9	13	10	13	8	13	10	11	11	12	13	14	14	12	11
C16.	7	13	9	10	14	13	16	12	15	13	13	13	13	13	14	14	13	12
C17.	12	8	10	11	16	11	14	11	15	12	12	12	13	13	15	14	10	12
C18.	18	10	15	11	10	8	10	9	14	11	11	11	12	12	13	13	14	12
C19.	21	12	11	8	13	12	15	12	16	14	14	14	14	14	16	15	15	13
C20.	16	13	14	12	18	13	17	13	16	15	15	15	15	15	17	16	16	14
C21.	11	11	13	10	15	12	16	13	17	16	16	16	16	16	18	17	17	15
C22.	19	12	13	11	17	13	18	14	18	17	17	17	17	17	19	18	18	16

Tabella 6.8: Tabella relativa al Mondiale *F1* 2008, in cui per ogni gara (G*) è riportata la posizione del concorrente (C*) in classifica generale, al termine della gara stessa. I valori sono ricavati dall'applicazione dell'algoritmo di Condorcet al termine di ogni gara, considerando anche le precedenti. Ogni gara rappresenta un voto, composto dagli ordini di arrivo.

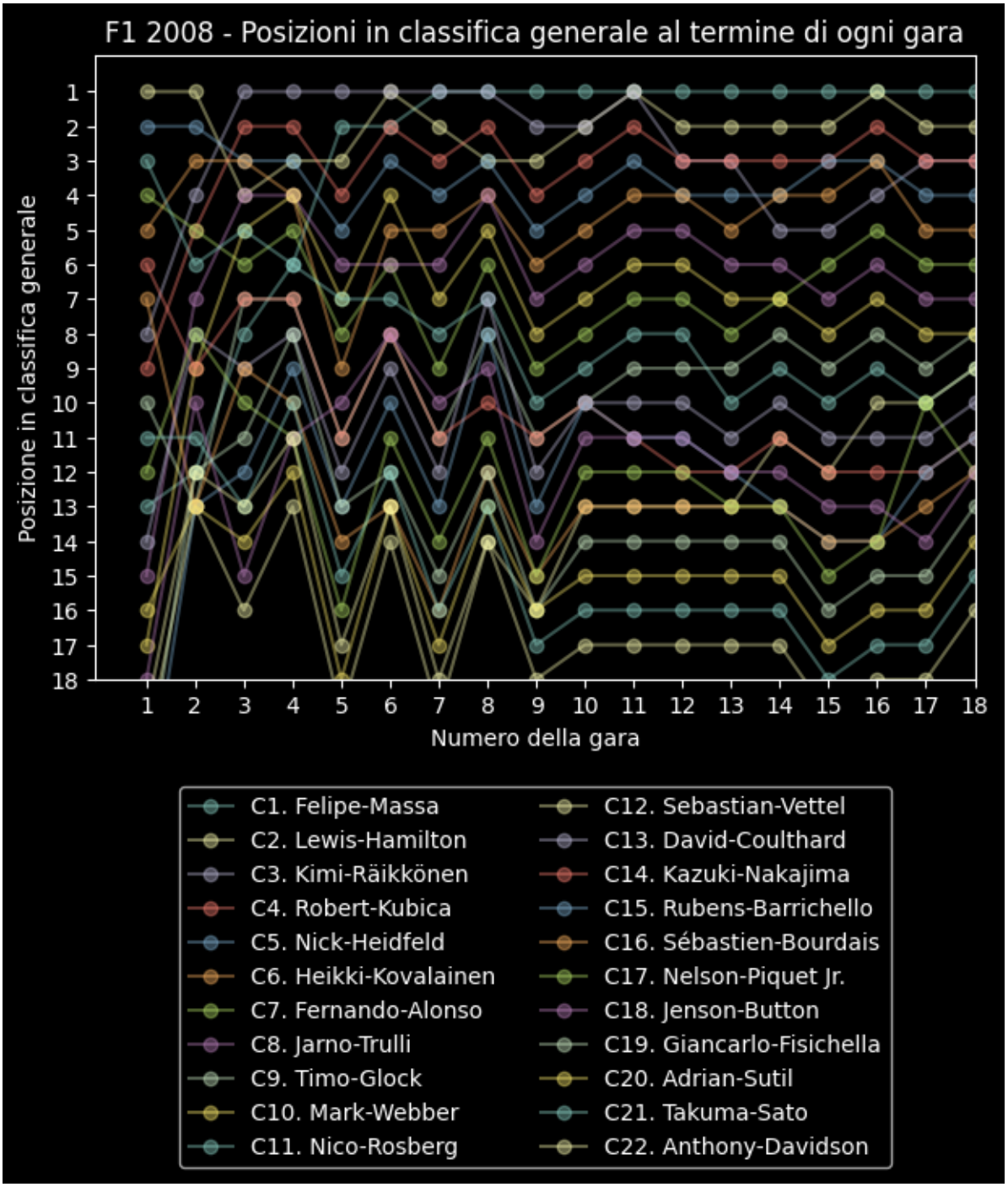

Figura 6.5: Rappresentazione grafica della tabella 6.8

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18
C1.	13	14	9	7	5	3	3	3	5	4	6	5	3	3	4	4	4	3
C2.	1	1	5	4	3	2	3	4	3	3	3	2	1	2	1	2	1	1
C3.	8	3	1	1	1	1	2	2	2	2	1	3	6	6	6	6	5	5
C4.	9	4	4	2	2	1	1	1	1	1	2	1	2	1	2	1	2	2
C5.	2	2	2	3	4	4	2	5	4	4	4	6	4	4	3	3	3	4
C6.	5	2	3	5	6	6	4	6	6	5	5	4	5	5	5	5	7	7
C7.	4	5	6	9	7	7	6	8	9	7	8	9	8	8	7	7	6	6
C8.	15	7	7	6	7	8	5	7	7	6	7	7	7	7	8	8	8	8
C9.	10	13	11	12	11	11	8	10	11	10	11	11	10	10	10	11	10	10
C10.	17	9	8	8	7	5	4	7	8	7	9	8	9	9	9	9	9	9
C11.	3	6	7	10	8	9	7	9	10	8	10	10	11	11	11	10	11	11
C12.	20	19	20	20	19	19	14	16	18	15	17	16	17	14	13	12	12	12
C13.	14	8	12	13	9	13	9	11	14	12	13	13	13	13	13	14	15	15
C14.	6	8	10	11	10	10	10	12	13	9	12	12	12	12	12	13	13	13
C15.	22	17	16	16	14	14	12	13	12	11	14	14	15	16	15	16	17	17
C16.	7	12	14	17	16	18	15	18	17	16	18	17	18	18	16	17	18	18
C17.	12	8	13	17	15	16	14	15	16	13	15	13	14	15	14	14	14	14
C18.	18	11	17	14	12	12	11	14	15	14	16	15	16	17	14	15	16	16
C19.	21	15	15	14	13	15	13	17	17	17	19	18	19	19	17	18	19	19
C20.	16	18	19	19	18	20	17	20	20	18	20	19	20	20	18	19	20	20
C21.	11	10	14	15	15	17	16	19	19	19	21	20	21	21	19	20	21	21
C22.	19	16	18	18	17	21	18	21	21	20	22	21	22	22	20	21	22	22

Tabella 6.9: Tabella relativa al Mondiale *F1* 2008, in cui per ogni gara (G*) è riportata la posizione del concorrente (C*) in classifica generale, al termine della gara stessa. I valori sono ricavati dall'applicazione dell'algoritmo di Schultze al termine di ogni gara, considerando anche le precedenti. Ogni gara rappresenta un voto, composto dagli ordini di arrivo.

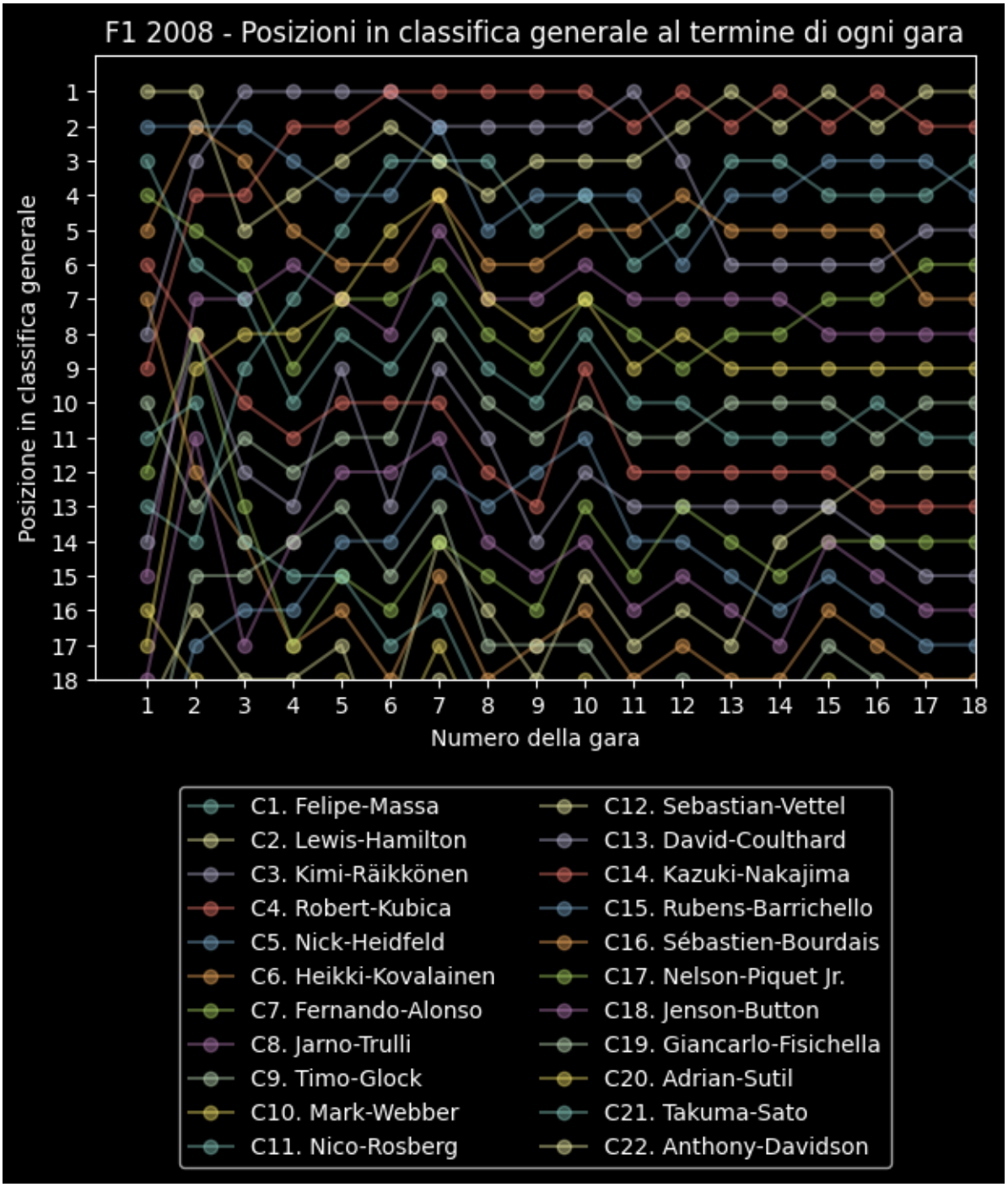

Figura 6.6: Rappresentazione grafica della tabella 6.9

Gli algoritmi impiegati hanno permesso di elaborare una differente distribuzione della prima classifica provvisoria, rispetto al caso reale in cui molti dei piloti erano finiti fuori gara per incidenti o problemi di motore.

Al pari del caso inerente al campionato 2023, si continua ad evidenziare una fase iniziale con numerosi pareggi per poi distinguerli successivamente, tuttavia i risultati di Condorcet mostrano limiti in merito alla capacità di discriminazione di un vincitore unico per ogni posizione.

In tabella 6.10 vengono mostrati i valori di distanza ottenuti applicando la *norma di Frobenius* alla distanza tra le matrici rappresentate nelle prime due colonne. Rispetto al caso del campionato 2023 si può notare un differenza più marcata in termini di distanze, sia nel caso di Schultze che in Condorcet. Quest'ultimo mostra una maggior differenza sia nel caso in cui viene confrontato l'insieme delle classifiche provvisorie, sia nel caso della classifica finale.

Elemento 1	Elemento 2	Valore distanza euclidea
Matrice "reale"	Matrice Condorcet	77.19
Matrice "reale"	Matrice Schultze	52.53
Classifica finale "reale"	Classifica finale Condorcet	24.6
Classifica finale "reale"	Classifica finale Schultze	12.45

Tabella 6.10: Campionato *F1* 2008 - Valori di distanza ottenuti applicando la *norma di Frobenius* alla differenza delle matrici rappresentate nelle prime due colonne.

Capitolo 7

Conclusioni