Alma Mater Studiorum · Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea in Ingegneria e scienze informatiche

MODELLI E PIATTAFORME PER LA DEMOCRAZIA DIGITALE: ANALISI E CONFRONTO

Relatore:
Chiar.mo Prof.
Andrea Omicini

Presentata da:
Thomas Trapanese

Sessione III
Anno Accademico 2016/2017

Alla mia compagna Gëzime,
per la sua inesauribile
fiducia in me...

A mio figlio Alessandro,
grazie a cui tutto è iniziato...

Introduzione

Democrazia: etimologicamente significa "governo del popolo", una forma di governo nella quale il potere decisionale spetta al popolo.

L'idea di democrazia non è univoca, nel corso della storia ha trovato diverse modalità di applicazione tutte caratterizzate dalla ricerca della forma ideale capace di dare l'effettivo potere decisionale al popolo.

Generalmente siamo portati ad associare l'idea di democrazia ad una forma di governo, è certamente vero che molti governi nazionali implementino una loro forma democratica ma è anche vero che il concetto di democrazia è molto più vasto e può essere applicato in qualunque contesto in cui sia richiesto di prendere una decisione su un argomento.

Negli ultimi anni grazie all'avvento di internet, il concetto di democrazia si è evoluto ed ha portato alla nascita dei concetti di e-democracy (una forma di democrazia diretta esercitata attraverso l'ausilio di strumenti digitali) e democrazia fluida (un modello ibrido tra la democrazia diretta e quella rappresentativa). Entrambi questi concetti sono stati concretizzati in software che mirano a fornire gli strumenti per implementare una democrazia.

L'idea di democrazia ha innumerevoli forme e implementazioni ma, è possibile estrarre delle caratteristiche comuni? in altre parole, è possibile individuare un modello per le varie forme di democrazia?

Nel corso di questa trattazione cercherò di rispondere a questa domanda individuando le componenti essenziali di un modello democratico e valutandone le principali implementazioni. Farò anche un analisi dei principali software per l'e-democracy che valuterò sia dal punto di vista tecnico che democratico (analizzando quale modello implementino).

Indice

Capitolo 1

Un modello per la democrazia

L'idea di democrazia può trovare infinite implementazioni pratiche, basti pensare a quante differenti forme di governo democratico esistono. Ogni implementazione democratica risponde a certe necessità ed è difficile immaginare un sistema che sia abbastanza flessibile da adattarsi ad ogni tipo di decisione, per ogni tipologia e numero di votanti. In questo capitolo cercherò di individuare un modello che permetta di identificare e classificare vari sistemi democratici. I sistemi analizzati successivamente si riferiscono tutti a sistemi on-line, ma le caratteristiche che individueremo hanno un carattere generale e indipendente dalla piattaforma analizzata.

Il processo democratico parte dal bisogno di effettuare una scelta tra più opzioni disponibili, quindi la prima domanda a cui rispondere nel cercare un modello è "chi presenta le opzioni di scelta?".

La democrazia prevede generalmente anche la possibilità di dibattere in qualche forma sulle opzioni disponibili, di confrontarsi con gli altri e di presentare le proprie idee agli altri membri della comunità.

Infine, ovviamente, esiste la votazione. Come vedremo le tipologie di votazione possibili sono moltissime e molto studiate storicamente. Il processo di votazione comincia con l'espressione del voto da parte dei votanti nei quali questi esprimo il loro parere su una certa tematica. I metodi di espressione sono diversi e determinano la quantità di informazioni che ogni votante può sottomettere per esprimere la propria preferenza. Una volta raccolte tutte le votazioni, l'estrazione del vincitore avviene mediante l'applicazione di un qualche algoritmo. La dimensione algoritmica della democrazia è quella che analizzeremo più affondo analizzando algoritmi ben noti e valutandone alcuni di recente concezione.

Da queste semplici considerazioni possiamo estrapolare 4 caratteristiche principali che caratterizzano un sistema democratico:

1. Modalità di presentazione delle proposte
2. Modalità di interazione
3. Modalità di voto
4. Algoritmo di selezione dei risultati

Di seguito mostrerò più nel dettaglio che cosa implichi ognuno di questi punti e quali siano le possibili alternative.

1.1 Presentazione delle proposte

A seconda dei sistemi democratici le proposte potranno essere presentate da diverse tipologie di persone.

Si possono pensare sistemi in cui qualunque membro della comunità possa presentare delle proposte. Questo approccio è certamente valido per sistemi composti da un limitato numero di utenti, diverso è invece il caso in cui le comunità sia costituite da una moltitudine di persone. Supponiamo di voler implementare un sistema per decidere come investire un certo budget disponibile, ogni utente potrà presentare la propria proposta su come investire le risorse disponibili. Un sistema con presentazione libera delle proposte, è applicabile fintanto che il numero totale di proposte presentate rimane limitato. Si tratta di un metodo sicuramente applicabile in un contesto familiare o in piccole comunità, potrebbe valere per piccole associazioni sportive, ma all'aumentare del numero di componenti della comunità possiamo aspettarci che aumentino proporzionalmente le proposte su come gestire il budget, per questo in molti sistemi democratici nasce l'esigenza di una qualche forma di rappresentanza.

Tutti i sistemi democratici di grandi dimensioni presentano una qualche forma di gerarchia, nella quale dei rappresentanti eletti secondo qualche criterio propongono (e generalmente votano) soluzioni a problemi di varia natura.

Un altro aspetto importante da considerare è che non sempre le proposte presentate vengono automaticamente accettate. Diversi sistemi implementano meccanismi di verifica della proposta atti a valutarne la correttezza formale o la conformità con certe regole. Non è quindi raro il caso in cui una proposta venga rifiutata ancor

prima di essere sottoposta ad un qualsivoglia tipo di valutazione da parte della comunità.

A seconda delle modalità di presentazione, un sistema risulterà essere più o meno strutturato. Un sistema non strutturato è un sistema a presentazione libera delle proposte, senza vincoli sul presentante. Viceversa un sistema fortemente strutturato presenterà delle gerarchie tra gli utenti e delle limitazioni sulle modalità di presentazione.

Il meccanismo di presentazione delle proposte, quali persone coinvolge, in quali modalità e quali sono i criteri di accettazione rappresentano senz'altro un aspetto importantissimo nella valutazione di un sistema democratico.

1.2 Interazione

In questa categoria vengono raggruppate tutte le modalità di interazione tra gli utenti riguardo l'argomento oggetto di decisione.

Prendiamo ad esempio un sistema parlamentare. Generalmente in un sistema di questo tipo qualche membro del parlamento sottopone una proposta di legge, gli altri membri potranno poi apportare e votare delle modifiche alla proposta presentata prima che questa sia messa in votazione.

Se invece pensiamo ad un sistema elettorale in cui la scelta è tra più candidati, le modalità di interazione può essere riferita alla comunicazione tra i candidati e i votanti. In questo caso potrebbero essere previsti dei proclami elettorali, oppure dei confronti tra candidati o addirittura confronti diretti tra i candidati e i cittadini.

L'interazione è essenziale e si intreccia con il concetto stesso di democrazia. Tutti i membri della comunità devono poter interagire in modo da esporre il loro punto di vista che potrà contribuire a migliorare la risposta al problema che si vuole affrontare.

Le possibilità di interazione tra gli utenti sono molteplici e difficilmente raggruppabili in categorie. Ogni sistema implementa un proprio meccanismo peculiare di gestione delle interazioni che lo caratterizza in maniera pressoché univoca.

1.3 Voto delle proposte

Le modalità attraverso le quali gli utenti esprimono la loro opinione possono portare a risultati diversi indipendentemente dal parere dei votanti.

I sistemi di voto possono essere classificati in tre macro-categorie:

1. a preferenza singola nei quali è richiesto di indicare una preferenza tra n possibili
2. a ordinamento, nei quali il votante può ordinare le n proposte in base alle sue preferenze.
3. sistemi di voto pesati, nei quali il votante assegna un punteggio ad ogni proposta

Bisogna inoltre sottolineato che alcuni sistemi potrebbero prevedere un voto in più fasi, ad esempio si potrebbe prevedere una prima votazione da cui selezionare solo le posposte che superino una certa soglia e poi una seconda votazione tra cui scegliere la proposta migliore.

La modalità con la quale ogni utente può esprimere la propria votazione determina la quantità di informazioni che vengono inviate al sistema. I sistemi a preferenza singola sono quelli che hanno il contenuto informativo minore; la quantità di informazioni inviate aumenta invece passando ad un sistema di votazioni ad ordinamento e poi ad un sistema di voto pesato.

1.4 Algoritmi di selezione dei risultati

L'algoritmo utilizzato per l'estrapolazione dei risultati di una votazione è determinante per il modello democratico in quando, a parità di altre condizioni, può portare a risultati anche molto differenti tra loro.

Quando si pensa ad un algoritmo di selezione dei risultati di una votazione viene subito in mente un sistema di voto maggioritario nel quale sostanzialmente, chi ottiene il maggior numero di voti vince. Questo sistema si basa sull'idea che ogni votante esprima la propria opinione scegliendo una tra più opzioni possibili, si pensi ad esempio alle elezioni politiche nelle quali in generale si esprime il proprio voto per un solo candidato.

Se però espandiamo la modalità di voto dando la possibilità di scegliere una lista di candidati, dal preferito al meno desiderabile si apre un nuovo spettro di possibilità

con molti possibili algoritmi tra cui scegliere. In questa nuova classe di algoritmi non conta più solo la prima scelta di ogni votante, ma anche la posizione relativa di ogni candidato nelle preferente individuali. Con questa modalità di selezione non necessariamente chi ottiene più voti come prima scelta risulta essere il vincitore.

Gli algoritmi di selezione del vincitore sono innumerevoli, di seguito illustro alcuni dei più famosi e notevoli approcci al problema.

1.4.1 Condorcet

Il metodo di Condorcet [22] permette di estrarre un vincitore tra n alternative usando il criterio di maggioranza applicato a confronti di coppia. Si procede effettuando un confronto tra ogni coppia di canditati e si valuta chi ha ottenuto il maggior numero di preferenze. Il risultato finale è dato dalla somma di tutti i risultati dei confronti a due dei candidati.

Facciamo un esempio pratico di applicazione del metodo di Condorcet. Supponiamo di avere una votazione tra 3 differenti proposte: A, B e C con 10 votanti che esprimono il loro voto attribuendo un ordinamento alle tre alternative.

	1°Scelta	2°Scelta	3°Scelta
Votante 1	A	C	B
Votante 2	B	C	A
Votante 3	A	B	C
Votante 4	C	B	A
Votante 5	C	A	B
Votante 6	C	A	B
Votante 7	A	B	C
Votante 8	C	A	B
Votante 9	B	A	C
Votante 10	C	A	B

Tabella 1.1: Esempio di esito di votazione a 3 candidati

Ora valutiamo un confronto diretto tra ogni coppia di candidati:

	Numero di votanti					
	2	4	1	1	1	1
1°Scelta	A	C	B	A	B	C
2°Scelta	B	A	C	C	A	B
3°Scelta	C	B	A	B	C	A

Tabella 1.2: Confronto di Condorcet

In tabella 1.2 troviamo il numero di votanti che hanno scelto ogni soluzione, ora possiamo effettuare un confronto tra ognuna delle alternative in ogni situazione di voto. Confrontiamo per ognuna delle scelte dei votanti le posizioni di A e di B:

1. 2 votanti hanno scelto la soluzione $A > B > C$ quindi questo porta 2 punti alla soluzione A rispetto alla soluzione B
2. la soluzione $C > A > B$ è stata scelta da 4 votanti, questo porta altri 4 punti alla soluzione A
3. la soluzione $B > C > A$ è scelta da 2 votanti, 1 punto alla soluzione B
4. la soluzione $A > C > B$ è scelta da 1 votante, 1 punto alla soluzione A
5. la soluzione $B > A > C$ è scelta da 1 votante, 1 punto alla soluzione B
6. la soluzione $C > B > A$ è scelta da 1 votante, 1 punto alla soluzione B

Al termine del confronto la soluzione A vince sulla soluzione B per 7 a 3

In maniera analoga eseguiamo il confronto tra l'opzione A e l'opzione C ottenendo che la scelta C è preferibile per 6 a 2 Infine eseguiamo il confronto tra C e B e otteniamo 6 4 Quindi secondo il metodo di Condorcet in questo caso il risultato della votazione è C > A > B

Bisogna notare come il metodo di Condorcet non porti sempre a un vincitore certo, infatti ci può trovare nel caso in cui $A > B$, $B > C$, $C > A$.

Il metodo di Condorcet nasce dall'esigenza di individuare un vincitore unico da un insieme di possibilità: non sempre questo è possibile. La difficoltà nell'individuare un unico vincitore ha portato all'introduzione di diverse varianti, alcune delle quali

verranno analizzate in seguito, che ottimizzano l'algoritmo per ridurre al minimo il caso di indeterminazione del vincitore.

Nel caso di una votazione possiamo definire l'utilità[1] individuale come il grado di soddisfazione dell'individuo in seguito al risultato della votazione. Il grado di soddisfazione sarà tanto più alto quanto più vicino è il risultato rispetto al voto espresso. Ora analizziamo l'algoritmo di Condorcet dal punto di vista dell'utilità. Possiamo affermare che l'algoritmo porta ad un sistema nel quale si tende a massimizzare l'utilità per il maggior numero di votanti, ovvero a massimizzare l'utilità complessiva.

Come misura dell'utilità possiamo definire una qualche metrica che misuri la distanza tra la votazione individuale e il risultato. Definita una metrica di questo tipo l'utilità individuale sarà inversamente proporzionale alla distanza calcolata (tanto minore è la distanza tra la scelta individuale e l'esito della votazione, tanto maggiore sarà la soddisfazione per l'esito)

Ad esempio possiamo calcolare la distanza tra la soluzione vincitrice e quelle dei vari votanti nell'esempio di tabelle 1.2. Per farlo ricorriamo a una misura di distanza tra le soluzioni detta distanza di Kemeny.

Applicando il calcolo otteniamo:

[1] In economia l'utilità è la misura della felicità o soddisfazione individuale [27]

	1°Scelta	2°Scelta	3°Scelta	Distanza Complessiva
Votante 1	A	C	B	26
Votante 2	B	C	A	34
Votante 3	A	B	C	30
Votante 4	C	B	A	30
Votante 5	C	A	B	22
Votante 6	C	A	B	22
Votante 7	A	B	C	30
Votante 8	C	A	B	22
Votante 9	B	A	C	38
Votante 10	C	A	B	22

Tabella 1.3: Distanza complessiva di ogni soluzione

La distanza per la soluzione scelta $C > A > B$ è 22 ovvero la minima distanza possibile e, quindi, quella che massimizza l'utilità complessiva.

In generale tutti i metodi di Condorcet esposti cercheranno di massimizzare l'utilità complessiva, ognuno con un proprio definizione di distanza.

1.4.2 Copeland

Una delle più note varianti al metodo di Condorcet è il metodo di Copeland [23]. In questo metodo il punteggio assegnato ad ogni candidato è dato dal numero di vittorie ottenute in ogni confronto di coppia meno il numero di sconfitte.

Riprendendo il caso in tabella 1.1 otteniamo

Confronto	Risultato	Vincitore
A vs B	7 vs 3	A
A vs C	4 vs 6	C
B vs C	4 vs 6	C

Tabella 1.4: Confronto a due tra i candidati

Ora applichiamo il il metodo di Copeland e otteniamo:

Confronto	Vittore	Sconfitte	Risultato
A	1	1	0
B	0	2	-2
C	2	0	2

Tabella 1.5: Confronto di Copeland

In questo caso il vincitore secondo Copeland coincide con il vincitore di Condorcet, esistono casi in cui pur non essendoci vincitori di Condorcet si ha un vincitore secondo Copeland.

1.4.3 Schulze

Il metodo di Schulze [26], anche conosciuto come Schwartz Sequential Dropping (SSD) è un'altra variante del metodo di Condorcet. Il Metodo Schulze si basa sul conteggio delle preferenze complessive tra i candidati ad esempio se un votante esprime il suo voto come $A > B > C$ la proposta A ottiene 2 punti ($A > B$ e $A > C$), la proposta B ne ottiene uno ($B > C$) mentre la proposta C non ne ottiene nessuno.

Riprendiamo sempre l'esempio in tabella 1.1, applicando Schulze otteniamo:

	2x	4x	1x	1x	1x	1x	Totale
A	2	1	0	2	1	0	11
B	1	0	2	0	2	1	7
C	0	2	1	1	0	2	12

Tabella 1.6: Confronto di Schulze

e otteniamo $C > A > B$

1.4.4 Tideman

Il metodo di Tideman [25] anche detto metodo delle Ranked Pairs è un'altra variante del metodo di Condorcet. Tideman tiene conto del numero di voti ottenuto da ogni candidato rispetto ad ogni altro (a differenze del metodo di Condorcet nel quale si considera solo il vincitore tra ogni confronto).

Per calcolare il vincitore secondo Tideman si procede ordinando i vincitori di ogni confronto da quello che ha ottenuto il margine maggiore a quello che ha ottenuto il margine minore. Riprendendo l'esempio di tabella 1.1 e i risultati di tabella 1.2

Confronto	Risultato	Vincitore
A vs B	7 vs 3	A (70%)
A vs C	4 vs 6	C (60%)
B vs C	4 vs 6	C (60%)

Tabella 1.7: Confronto di Tideman

Ora si procede scorrendo la lista dal vincitore con il margine più alto e si aggiungono vincoli alla soluzione man mano che si procede nella lista. In questo caso dal primo confronto otteniamo $A > B$, dal secondo $C > A$ e dal terzo $C > B$ il che porta alla soluzione unica $C > A > B$

Nel semplice caso analizzato non abbiamo trovato cicli, ipotizziamo che il risultato della nostra votazione abbia portato ad una situazione di questo tipo:

Confronto	Risultato	Vincitore
A vs B	8 vs 2	A (80%)
A vs C	4 vs 6	C (60%)
B vs C	7 vs 3	B (70%)

Tabella 1.8: Confronto di Tideman con cicli

In questa situazione usando il metodo di Condorcet ci troviamo a non poter eleggere un vincitore in quanto risulterebbe $A > B$, $B > C$, $C > A$ creando un ciclo. Secondo l'approccio di Tideman invece, la soluzione viene costruita partendo dal vincitore con maggior margine, se durante la costruzione si incontra un vincolo che crea un ciclo, questo viene ignorato. Quindi secondo Tideman l'ordinamento vincente nel caso di tabella 1.8 è: $A > B > C$ ignorando l'ultimo vincolo per margine di votazione (60%) $C > A$

1.4.5 Pareto

I metodi di selezione del vincitore sin qui analizzati rientrano tutti nei metodi di Condorcet e mirano a estrapolare un vincitore da un'insieme di scelte possibili usando una qualche metrica di confronto.

Si possono però pensare sistemi iterativi nei quali la votazione non debba necessariamente essere unica. Sistemi di questo tipo richiedono un meccanismo si selezione e di riduzione delle soluzioni disponibili ad ogni votazione. Un possibile approccio al problema è dato dall'utilizzo del fronte di pareto [16].

Il fronte di pareto rappresenta un sotto-insieme di soluzioni ottime tra tutte quelle possibili. Il calcolo delle soluzioni ottime di basa sulla definizione di una regola di "dominazione" che stabilisca, date due possibili soluzioni, se una è migliore dell'altra. Se abbiamo due proposte A e B, ed usando una qualche regola stabiliamo che la soluzione A è preferibile alla soluzione B allora diremo che A domina B. Il fronte di pareto sarà costituito da tutte le soluzioni non dominate da altre.

Supponiamo di avere stabilito una regola di dominazione basata su due criteri $f1$ ed $f2$, possiamo porre su un grafico bi-dimensionale tutte le proposte possibili ponendo nell'asse delle x il valore di $f1$ e nell'asse delle y il valore $f2$.

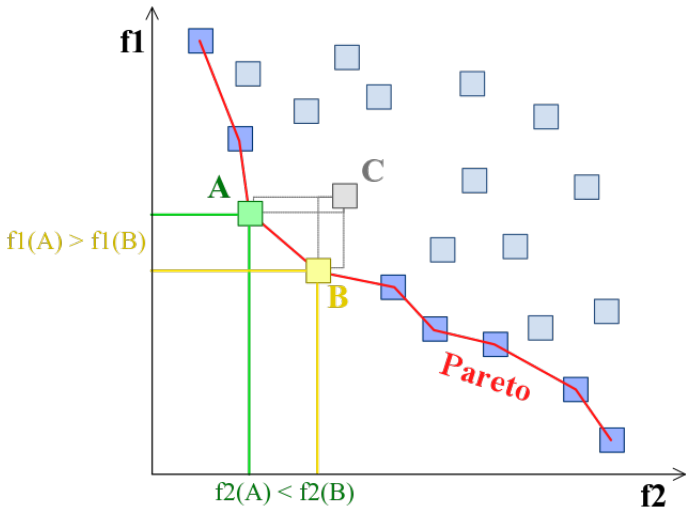


Figura 1.1: Fronte di pareto

Da notare che, in generale, il fronte di pareto sarà costituito da più di un elemento e che i criteri che compongono la regola di dominazione possono essere molteplici.

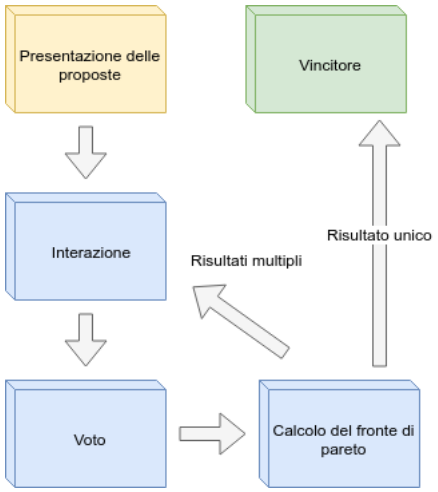


Figura 1.2: Esempio di sistema ricorsivo con fronte di pareto

Avendo più di una soluzione disponibile possiamo pensare ad un sistema iterativo (Figura 1.2) nel quale i risultati ottenuti dal fronte di pareto vengono poi rimessi in discussione e votati nuovamente fino a raggiungere una soluzione unica.

Il metodo di pareto non offre di per se un sistema di calcolo del vincitore, ma solo un approccio alla selezione. La scelta della regola di dominazione determina inte-

ramente il risultato. In generale gli altri metodi esposti possono essere ricondotti al calcolo mediante il fronte di pareto con la definizione di opportune regole di dominazione

1.4.6 Contucci

Il metodo di Contucci [4] applica la statistica per l'estrapolazione del risultato di una votazione. Si basa sul calcolo della distanza tra le varie soluzioni utilizzando una qualche metrica, la distanza più comunemente usata è la distanza di Kemeny ma il metodo esposto è indipendente dalla definizione di distanza utilizzata. In generale la soluzione ottima è considerata quella che minimizza la distanza tra le varie soluzioni, ovvero quella che minimizza la formula:

$$\mu(c) = \frac{1}{n} \sum_{v=1}^{n} d(r_v, c) \tag{1.1}$$

Dove:

- n è sono il numero di possibili soluzioni
- c è la soluzione presa in esame
- $\mu(c)$ rappresenta il valore calcolato per la soluzione
- r_v la v-esima alternativa

Questa applicazione è riconducibile ai vari metodi di Condorcet e derivati (utilizzando opportune definizioni di distanza). La particolarità del metodo di Contucci è che questo non estrae la soluzione come quella che minimizza la distanza complessiva ma cerca la soluzione che sia equidistante da tutte le altre. Questo approccio sovverte completamente il senso comune di votazione. Concettualmente non ci si focalizza più sul trovare la soluzione che accontenti il maggior numero di persone, ma su quella che ne scontenti di meno. In termini pratici si cerca di minimizzare il valore di $\sigma(c)$ nella formula:

$$\sigma(c) = \sqrt{\frac{1}{n} \sum_{v=1}^{n} [d(r_v, c) - \mu(c))]^2} \tag{1.2}$$

Tramite il metodo di Contucci quindi si cerca di minimizzare la varianza della distanza tra le diverse soluzioni. A questo punto è giusto notare come il vincolo di minimizzazione della varianza non sia sempre sufficiente ad individuare una soluzione univoca, infatti è possibile mostrare casi in cui soluzioni diametralmente opposte soddisfino ugualmente il vincolo di minimizzazione della distanza, in questi casi Contucci suggerisce di scegliere la soluzione vincitrice tra quelle che minimizzano contemporaneamente sia μ che σ

Distanza di Kemeny La distanza di Kemeny è una misura della differenza tra due possibili esiti di un votazione [24]. Siano X e Y due risultati ad una votazione con una lista ordinata di preferenze.

Allora la distanza di Kemeny tra le due soluzione è definita come:

$$dist(X,Y) = \sum_{\{x,y\}} d_{X,Y}(x,y) \tag{1.3}$$

dove x e y rappresentano tutte le combinazioni dei componenti delle soluzioni e $d_{X,Y}(x,y)$ è definita come:

$$d_{X,Y}(x,y) = \begin{cases} 0 \text{ se X e Y concordano su x e y} \\ 1 \text{ se X e Y sono in disaccordo su x e y} \end{cases}$$

Di base la distanza di Kemeny non tiene conto dell'eventualità di pareggi tra i candidati, per introdurre anche questa possibilità la definizione è stata estesa come segue [6]:

$$d_{X,Y}(x,y) = \begin{cases} 0 \text{ se X e Y concordano su x e y} \\ 1 \text{ se uno tra X e Y ha una preferenza tra i dei candidati e l'altro no} \\ 2 \text{ se X e Y sono strettamente in disaccordo su x e y} \end{cases}$$

Facciamo un semplice esempio di calcolo della distanza di Kemeny.

Prendiamo due soluzioni:

$$X : A > B = C > D$$
$$Y : B > C > A > D$$

Confronto	X	Y	$d_{X,Y}(x,y)$
A vs B	$A > B$	$B > A$	2
A vs C	$A > C$	$C > A$	2
A vs D	$A > D$	$A > D$	0
B vs C	$B = C$	$B > C$	1
B vs D	$B > D$	$B > D$	0
C vs D	$C > D$	$C > D$	0

Tabella 1.9: Esempio di calcolo della distanza di Kemeny

In questo caso la distanza di Kemeny tra le due soluzione è data da $2 + 2 + 0 + 1 + 0 + 0 = 5$.

Da notare che al momento non si conoscono algoritmi per il calcolo della distanza di Kemeny in maniera polinomiale e che è dimostrato che la classe del problema è NP-Hard [6].

1.5 Il modello individuato

In questo capitolo ho mostrato quali siano le principali caratteristiche di un modello democratico e alcuni dei valori che queste caratteristiche possono assumere.

Abbiamo visto l'importanza della modalità di presentazione delle proposte e come questa determini un sistema più o meno strutturato. Siamo poi passati all'individuazione delle modalità di interazione tra gli utenti e al ruolo essenziale che questa caratteristica assume in ogni sistema democratico.

Infine ci siamo concentrati sulle modalità di espressione del parere dei votanti. Prima abbiamo classificato il sistema voto in base alla quantità di informazione che viene scambiata tra gli utenti e il sistema, poi ci siamo concentrati sugli algoritmi di selezione dei risultati. Abbiamo visto che, in generale, i metodi derivati da Condorcet mirano a massimizzare l'utilità complessiva. Ho inoltre mostrato un approccio alternativo alla selezione del vincitore tramite l'analisi del fronte di pareto. Poi abbiamo visto il metodo proposto da Contucci il quale offre un approccio innovativo al problema di selezione del vincitore, scostandosi dai classici metodi di Condorcet.

Capitolo 2

Sistemi di e-democracy

Con il termine e-democracy si indica una forma di democrazia diretta che viene esercitata con l'ausilio di strumenti on-line per la cooperazione e la votazione [21].

Esistono numerosi sistemi per l'e-democracy la maggior parte dei quali è open-source. Nel corso di questo capitolo analizzerò alcuni dei sistemi principali e cercherò di estrarre il modello democratico che implementano, come individuato nel capitolo precedente. Nel corso dell'analisi effettuerò anche una analisi tecnica per quei sistemi che rendono disponibile il codice sorgente.

2.1 LiquidFeedback

LiquidFeedback (spesso abbreviato in LQFB) è un software open-source nato per supportare il processo democratico di decisione seguendo i dettami della democrazia liquida.

La democrazia liquida combina gli aspetti sia della democrazia diretta che di quella rappresentativa. LQFB implementa un sistema di deleghe che permette ad ogni utente di scegliere un proprio rappresentate per le votazioni, ogni delegato potrà a sua volta delegare un altro utente (Figura 2.1). Ogni rappresentate avrà un peso decisionale pari al numero complessivo di deleghe ricevute +1 (il proprio voto).

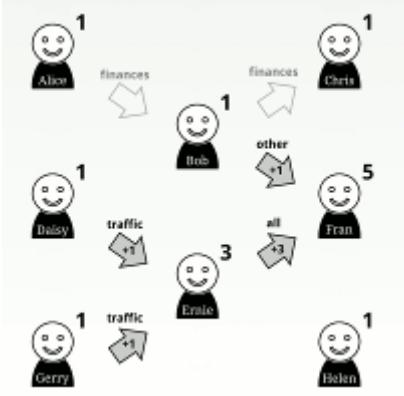


Figura 2.1: Democrazia liquida

2.1.1 Struttura del software

Il progetto è suddiviso in due componenti, LiquidFeedback Core e LiquidFeedback Front-End:

LiquidFeedback Core La componente core di LiquidFeedback è scritta in PL/-pgSQL e C. Il core contiene le strutture sql, le viste e le stored procedure necessarie per il corretto funzionamento del software.

LiquidFeedback Front-End La componente di front-end permette agli utenti di accedere a tutte le funzionalità del core. Il software è basato su WebMCP, un framework implementato dagli stessi sviluppatori di LiquidFeedback.

WebMCP è un framework web scritto in Lua e C. WebMCP abbandona il ben conosciuto pattern Model-View-Controller (Figura 2.2) e utilizza un approccio detto Model-View-Action (Figura 2.3).

Nel Model-View-Action, il database viene acceduto attraverso lo strato Model che è realizzato mediante un ORM (Object Relational Mapping). Le richieste http GET vengono gestite dalle View che processano la richiesta, interrogano il database e renderizzano il risultato. Le richieste http POST vengono gestite dalle componenti Action che possono scrivere su database e ridirezione ad una view (la view può essere differente in base all'esito dell'operazione)

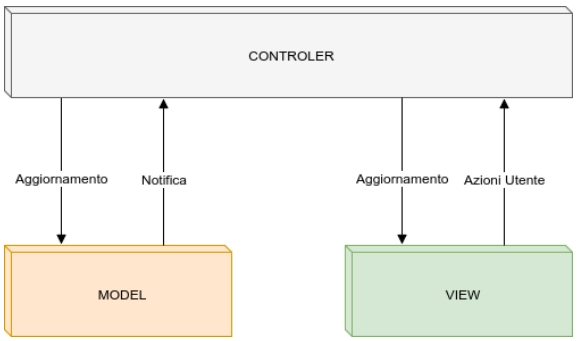


Figura 2.2: Pattern MVC

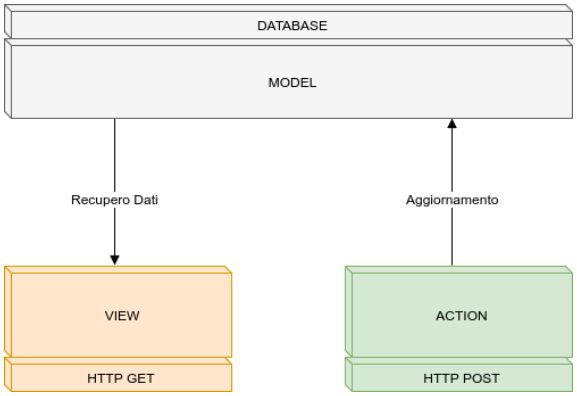


Figura 2.3: Pattern MVA

Al momento in cui scrivo esiste un altro progetto relativo a LiquidFeedback in fase di sviluppo, LiquidFeedback API.

LiquidFeedback API Lo scopo di LiquidFeedback API è quello di creare un'astrazione dalla base dati. Le API dovrebbero creare un'interfaccia comune per l'accesso alle funzionalità core, senza che chi consuma i servizi debba avere alcuna conoscenza sul come le funzionalità siano implementate. Il progetto andrebbe quindi inserito tra la componente core e la componente di front-end.

Guardando nel repository dei sorgenti risulta evidente che il progetto sia fermo da diverso tempo (ultima commit del 22-03-2013) e la documentazione indica che il progetto non è manutenuto [10]

2.1.2 Il modello democratico di LiquidFeedback

Cerchiamo ora di estrarre il modello democratico di LiquidFeedback. LiquidFeedback mette a disposizione una struttura che implementa il concetto di democrazia liquida applicato alla presentazione e approvazione di proposte. In pratica si basa sull'idea che ogni utente possa scegliere se esercitare il proprio voto direttamente o se delegare qualcuno a prendere decisioni in sua vece.

Il flusso di approvazione di LQFB si articola in 4 fasi (Figura 2.4):

1. Ammissione: Questa è la prima fase di presentazione di una proposta. Una volta inserita la proposta questa deve raggiungere un quorum di persone che la supportano per essere ammessa alla fase successiva.

2. Discussione: Se la proposta raggiunge il quorum questa viene ammessa alla fase di discussione, come suggerisce il nome questa fase permette agli utenti di commentare e suggerire migliorie alla proposta che potranno essere integrate dall'iniziatore (il presentatore della proposta) al fine di migliorarne la qualità.
3. Verifica: Giunti a questa fase la proposta non è più modificabile e viene richiesto il raggiungimento di un secondo quorum per passare alla fase di voto
4. Voto: Se anche il secondo quorum viene raggiunto si giunge alla fase di voto. Al termine della fase viene calcolato e reso disponibile il risultato della votazione.

La durata di ogni fase e i quorum sono stabiliti nelle "policy", insiemi di impostazioni create dall'amministratore e selezionabili al momento della presentazione della proposta.

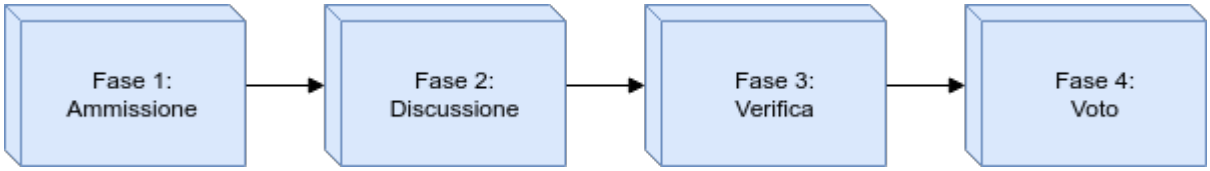


Figura 2.4: Flusso di approvazione di LQFB

LQFB presenta una struttura gerarchica a tre livelli organizzata in unità, aree e sezioni.

- Unità: possono essere settate in modo da dare agli utenti della piattaforma accesso selettivo. Questa particolarità è utile quando una stessa piattaforma LQFB gestisce diverse comunità. Nel caso si gestisca un'unica comunità questo livello può essere ignorato
- Aree: sono generalmente divise per tematiche (es. Economia, Ambiente, Tempo libero,...). Anche queste possono essere settate in modo selettivo, in modo da dare agli utenti opzioni di lavoro differenti.
- Sezioni: sono l'ultimo livello della gerarchia dove le iniziative sono raccolte per argomento.

Presentazione delle proposte Ogni membro abilitato può sottoporre delle proposte nell'opportuna sezione. La verifica della validità e dell'interesse delle proposte viene fatto dagli stessi utenti, la proposta presentata potrà essere messa

in discussione solo se raggiungere un minimo quorum di persone interessate (di default il 10% degli aventi diritto di voto).

Il raggiungimento del quorum tiene conto del peso di ogni voto, eventuali persone delegate avranno un peso pari al numero di deleghe ricevute.

La modalità di presentazione della proposta risulta quindi libera e aperta a tutti e la verifica della validità viene fatta dalla comunità stessa.

Interazione L'interazione tra gli utenti abbraccia tutto il processo decisionale di LiquidFeedback. Durante il processo di approvazione di una proposta è richiesta una forte interazione degli utenti con il sistema.

Inizialmente una nuova proposta presentata deve raggiungere un quorum di utenti che la sostengono per poter essere messa in discussione.

Superata l'approvazione preliminare inizia una fase di dibattito sulla proposta aperta a tutti gli utenti nella quale si possono proporre modifiche e/o soluzioni alternative.

Completata anche la fase di dibattito la soluzione viene ritenuta "stabile" e non risulta quindi modificabile. Prima della votazione gli utenti potranno comunque proporre soluzioni alternative a quella presentata.

Voto delle proposte LiquidFeedback prevede un sistema di voto preferenziale a ordinamento diviso in tre classi: favorevole, contrario, astenuto.

Ogni votante dovrà inserire ognuna delle proposte all'interno di una delle tre classi, all'interno della stessa classe potrà stabilire l'ordine di priorità di ognuna delle proposte.

Il voto delle proposte è sempre pubblico e visibile agli altri utenti votanti

Figura 2.5: Sistema di voto di LQFB

Algoritmo di selezione dei risultati L'algoritmo di selezione dei risultati implementa il ben noto algoritmo di Schulze, il calcolo viene effettuato tramite una Function all'interno del database Postgres.

Presentazione delle proposte	
• Presentazione libera	Sì
• Validazione proposta	Ad opera degli stessi utenti, la proposta deve raggiungere un quorum di approvazione
Interazione	
• Richiesta modifiche	Si, durante la fase di dibattito
• Presentazione alternative	Si, in ogni momento prima del voto
• Visibilità richieste	visibili a tutti gli utenti iscritti
Voto delle proposte	
• Modalità di espressione	Valutazione ad ordinamento divisi in tre gruppi: favorevole, contrario, astenuto
• Voto segreto	Non previsto in nessuna modalità
• Deleghe	Si, possibili anche deleghe transitive
Selezione	
• Algoritmo	Algoritmo di Schulze
• Vincitore unico	Sì

Tabella 2.1: Scheda riassuntiva del modello democratico di LiquidFeedback

2.2 OpenDCN

OpenDCN è un software open source sviluppato da Fondazione RCM - Rete Civica di Milano insieme al Laboratorio d'Informatica Civica dell'Università degli Studi di Milano; nasce dall'esigenza di estendere la partecipazione dei cittadini alla vita politica della città attraverso la rete [12].

Il software consente ai partecipanti di intervenire nelle discussioni informate, dare visibilità ad eventi ed iniziative e facilitare il raggiungimento di posizioni condivise nella formulazione di proposte e nella scelta tra differenti alternative.

Uno dei principali siti realizzati con openDCN è partecipaMi [13] che è nato con l'obiettivo di creare un luogo d'incontro tra la cittadinanza milanese e i loro amministratori.

2.2.1 Struttura del software

Il software è un'applicazione web realizzata in PHP mediante l'ausilio del framework CakePHP. L'applicazione è realizzata seguendo il pattern MVC e la base dati è interamente gestita tramite l'ORM integrato nel framework. La base dati di riferimento per il progetto è mysql.

2.2.2 Il modello democratico di OpenDCN

OpenDCN è un software composto da diverse componenti, quella che analizzeremo è la componente di brainstorming che realizza il sistema di decisione partecipativa della piattaforma.

Il flusso di presentazione delle proposte si articola in 4 fasi (Figura 2.6):

1. Avvio del dibattito: In questa fase gli utenti propongono i loro scenari (problemi che intendono affrontare).

2. Raccolta delle proposte: gli utenti presentano le loro proposte per la risoluzione dello scenario,

3. Valutazione delle idee: Le proposte sono visibili a tutti i membri della comunità che votano per le varie proposte.

4. Calcolo delle idee più apprezzate: conclusa la fase di votazione si procede alla selezione delle idee migliori.

Al termine dell'ultima fase potrebbe esserci più di una proposta selezionata, in questo caso il flusso di ripete a partire dalla fase di raccolta delle proposte (le proposte selezionate dall'ultima iterazione vengono riproposte automaticamente).

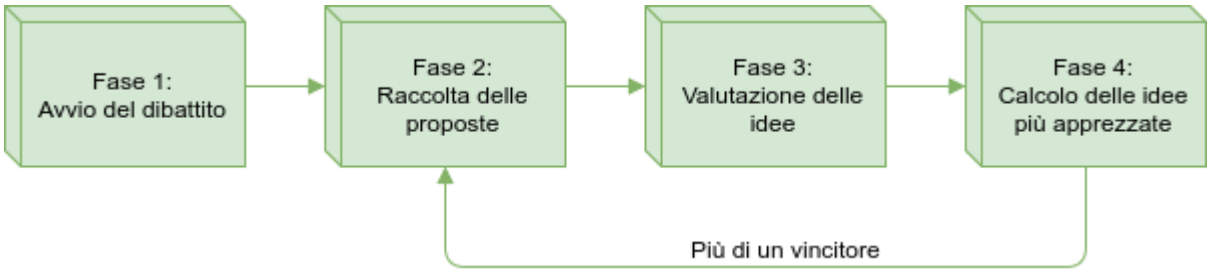


Figura 2.6: Flusso di approvazione di OpenDCN

Presentazione delle proposte La presentazione delle proposte si articola nelle prime due fasi del brainstorming. Durante la fase di avvio del dibattito, tutti i membri del sito possono sottoporre le loro problematiche alla comunità. Le problematiche sottoposte vengono "accettate" e quindi rese pubbliche agli altri membri del sito previa approvazione da parte dell'amministratore.

Una volta che la problematica è stata ammessa, inizia la fase di raccolta delle proposte. In questa fase tutti i membri del sito posso sottoporre delle proposte per la risoluzione del problema. Le proposte non saranno visibili da altri membri della comunità fino alla fase successiva. La conclusione della fase di presentazione delle proposte è stabilita dall'amministratore.

La presentazione delle proposte è quindi libera e chiunque può sottoporre la propria ma è comunque sempre sottoposta alla valutazione di un utente moderatore che si occupa di governare tutto il processo.

Interazione Nel brainstorming di OpenDCN non sono presenti meccanismi diretti di interazione tra gli utenti, l'interazione avviene sempre tra un utente e il sistema.

L'evoluzione delle proposte avviene grazie al meccanismo iterativo di presentazione, valutazione e selezione delle idee.

Voto delle proposte Il voto delle proposte può essere espresso tramite l'assegnazione di un voto (Figura 2.7). Il range di valori possibili viene stabilito quando si inserisce uno scenario, i valori possibili sono da 0 a 1; da 0 a 3; da 0 a 5;

Ogni utente potrà assegnare un voto ad ognuna delle proposte.

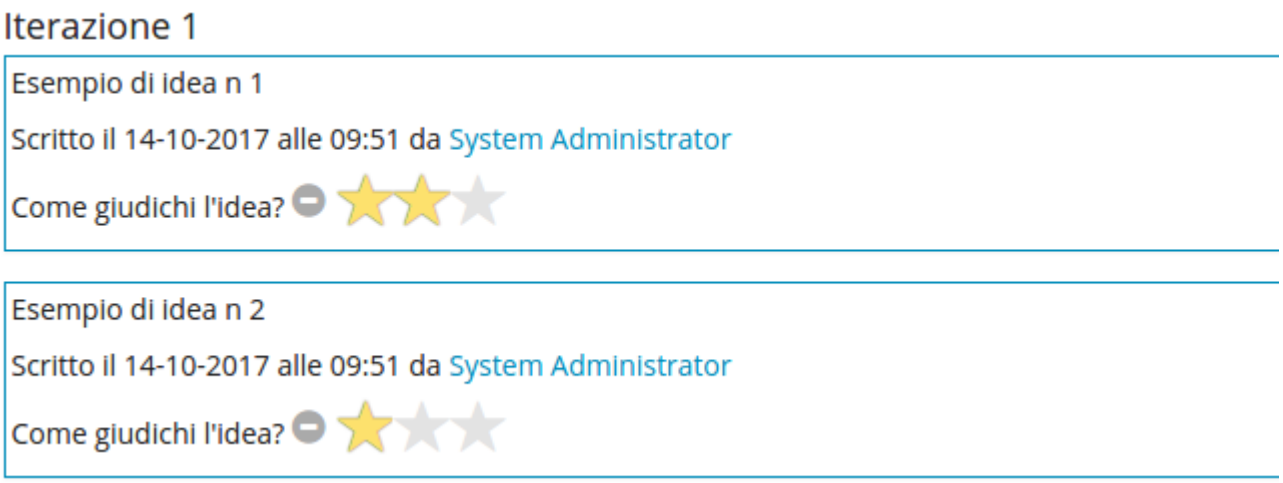


Figura 2.7: Sistema di voto di OpenDcn

Algoritmo di selezione dei risultati L'algoritmo di selezione delle proposte utilizza il fronte di pareto definendo la regola di dominazione come:

1. Ogni proposta ha associato un vettore V con le valutazioni dei partecipanti

2. Una proposta a domina una proposta b se $V_a \neq V_b$ e per ogni partecipante p $V_a(p) \neq V_b(p)$

In questo modo se un partecipante vota per una sola proposta questa verrà sicuramente selezionata.

Presentazione delle proposte	
• Presentazione libera	Sì per gli utenti iscritti. Le idee possono essere presentate ad ogni iterazione
• Validazione proposta	Ad opera dell'amministratore
Interazione	
• Richiesta modifiche	No
• Presentazione alternative	Sì nell fase di raccolta delle idee
• Visibilità richieste	Proposte visibili solo durante la fase di voto
Voto delle proposte	
• Modalità di espressione	Voto numerico per ognuna delle proposte, il range di voti viene stabilito durante la presentazione dello scenario (0-1, 0-3, 0-5)
• Voto segreto	No
• Deleghe	No
Selezione	
• Algoritmo	Selezione mediante fronte di pareto
• Vincitore unico	Generalmente no, in tal caso di procede con un nuova iterazione partendo dalla fase di presentazione delle proposte

Tabella 2.2: Scheda riassuntiva del modello democratico di OpenDCN

2.3 Airesis

Airesis di autodefinisce come "Social Network per l'E-Democracy", si tratta di un progetto open-source realizzato da un team italiano che mette a disposizione una serie di strumenti social e deliberativi per assistere un comunità democratica on-line.

2.3.1 Struttura del software

Il software è realizzato come un'applicazione web scritta in Ruby utilizzando il pattern MVC, poggia su un database PostgreSQL 9 per la memorizzazione dei dati applicativi.

Per la gestione dei task in background viene utilizzato Sidekiq [15], libreria per la gestione di task in background in ambiente ruby in combinazione con Redis [14], un in-memory database open source.

Airesis offre anche un potente motore di ricerca interno basato su SOLR [2], piattaforma di ricerca open source che sfrutta il noto motore di indicizzazione Lucene [1].

2.3.2 Il modello democratico di Airesis

Airesis è strutturato in modo fortemente orientato all'interazione, sono presenti vari strumenti social (blog, forum e calendario) per stimolare la partecipazione e il coinvolgimento dei membri della comunità.

Il portale è basato sul concetto di "gruppo", costituito da un sotto-insieme di membri del portale che condividono particolari interessi ed è legato ad una specifica area geografica. Ogni membro può far parte di più gruppi ai quali può richiedere l'iscrizione che deve essere approvata dall'amministratore.

All'interno di ogni gruppo gli utenti possono confrontarsi usando i vari strumenti messi a disposizione dalla piattaforma e deliberare in maniera assistita su diverse tipologie di proposte. L'iter di presentazione delle proposte si articola in 3 fasi (Figura 2.8):

1. Presentazione della proposta: gli utenti all'interno dei gruppi possono presentare delle problematiche con una o più soluzioni

2. Dibattito: tutti gli utenti possono discutere per elaborare al meglio la proposta prima della messa in votazione

3. Votazione: Dopo il dibattito le proposte vengono messe in votazione dove ogni utente può votare le soluzioni emerse nella fase di dibattito

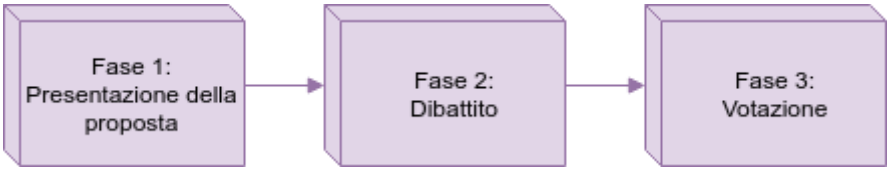


Figura 2.8: Flusso di approvazione di Airesis

Presentazione delle proposte La presentazione di proposte può venire da qualunque membro della comunità che ne diventa il portavoce. Al momento della presentazione vengono forniti diversi modelli di proposta tra cui scegliere: standard, regolamento, preventivo di spesa, comunicato, decisione tecnica, ecc... Ogni modello presenta una serie di campi da compilare che lo caratterizzano.

Oltre ai campi della proposta l'utente potrà scegliere diverse opzioni:

- Anonimato temporaneo: se attivata questa opzione nasconde i nomi degli utenti che partecipano alla discussione. L'idea di base è di fare in modo che i vari contributi siano valutati in maniera oggettiva evitando che asti personali tra i membri influenzino il giudizio delle proposte. La funzionalità è disattivatile in qualunque momento da parte del presentante.
- Visibile pubblicamente: questa opzione permette di stabilire se la proposta e i vari commenti degli utenti deve essere visibile pubblicamente.
- Voto segreto: determina se il voto degli utenti debba essere palese o segreto, in caso di voto palese al termine della votazione, ogni utente potrà vedere il voto degli altri

Infine il portavoce dovrà impostare le durate delle fasi di dibattito e di votazione per la proposta.

Durante la compilazione dei vari campi, il presentante dovrà indicare anche una o più soluzioni possibili, i campi e le proposte potranno essere modificati durante tutta la fase di dibattito. La fase di dibattito presenta un quorum che deve essere raggiunto per poter passare alla fase di voto che si basa su 3 parametri

- Partecipanti: numero minimo di persone che partecipano al dibattito.
- Tempo di discussione: periodo temporale in cui la proposta rimane nella fase di dibattito (a partire dalla presentazione della stessa).
- Maturità: percentuale di partecipanti che dichiarano che la proposta è pronta per essere votata.

Le proposte che al termine della fase di dibattito non raggiungono il quorum, vengono archiviate come "abbandonate" e rimangono visibili agli utenti.

Interazione I meccanismi di interazione tra gli utenti sono alla base di Airesis e questo si rispecchia anche nelle modalità di cooperazione per il miglioramento di una proposta.

Gli utenti possono lasciare commenti su ogni campo della proposta e su ognuna delle soluzioni presentate, potranno anche votare (favorevole, neutrale o contrario) ogni commento lasciato da ogni altro utente, in questo modo per il portavoce sarà più semplice individuare quali siano le modifiche più desiderate dagli utenti.

I commenti possono riguardare anche la proposta nella sua interezza, potrebbe ad esempio essere avanzata una nuova soluzione come alternativa a quelle presenti.

La decisione finale su come integrare ogni commento nella proposta originale spetta al portavoce che, in ogni momento, può modificarla ed eventualmente estenderla con la presentazione di soluzioni alternative.

Voto delle proposte Il voto avviene per un periodo di tempo determinato dal portavoce al termine della fase di dibattito.

La modalità di espressione del voto degli utenti cambia automaticamente in base al numero di soluzioni tra cui scegliere. Nel caso di una soluzione unica, la votazione ha carattere approvativo e il voto di ogni utente potrà essere espresso scegliendo tra tre possibilità: favorevole; neutrale; contrario;

Nel caso di più soluzioni disponibili ad ogni utente verrà chiesto di ordinarle in base alla loro desiderabilità (con la possibilità di equiparazione delle proposte).

Algoritmo di selezione dei risultati Nel caso di votazione approvativa non esiste un vero e proprio algoritmo di calcolo dei risultati ma ci si limita a raccogliere i voti di ognuno dei tre tipi e a mostrarli (la visualizzazione avviene con un grafico a torta).

Nel caso di più soluzioni disponibili viene usato l'algoritmo di Schultze per la determinazione della soluzione vincitrice.

Presentazione delle proposte	
• Presentazione libera	Si
• Validazione proposta	Validazione da parte degli utenti, richiesto il raggiungimento di un quorum
Interazione	
• Richiesta modifiche	Si, su ogni campo delle proposte o sulla proposta nella sua interezza
• Presentazione alternative	Si
• Visibilità richieste	Pubblica o anonima, configurabile al momento della presentazione; in ogni momento l'amministratore può rimuovere l'opzione di segretezza
Voto delle proposte	
• Modalità di espressione	Nel caso di soluzione singola l'utente esprime il voto come favorevole, contrario o neutrale. In caso di soluzioni multiple l'utente effettua una votazione ad ordinamento con possibilità di pareggi
• Voto segreto	Configurabile al momento della presentazione
• Deleghe	No
Selezione	
• Algoritmo	Algoritmo di Schulze
• Vincitore unico	Sì

Tabella 2.3: Scheda riassuntiva del modello democratico di Airesis

2.4 Comparazione dei modelli individuati

In questo capitolo ho presentato un'analisi dei 3 principali sistemi di delibera online attualmente esistenti ed ho estratto per ognuno di essi il modello democratico

che lo caratterizza come analizzato nel capitolo 1 (schede riassuntive 2.1 2.2 e 2.3).

I vari sistemi permettono a tutti gli utenti di presentare liberamente delle proposte (previa registrazione), alcuni richiedono una validazione da parte di un amministratore mentre in altri è stato studiato un sistema di validazione che poggia sulla comunità di utenti. Tra i software analizzati l'unico a presentare un sistema gerarchico è LiquidFeedback che implementa alla perfezione i concetti della democrazia liquida con un avanzato sistema di deleghe; questa sua peculiarità lo rende estremamente flessibile e gestibile anche con un numero molto elevato di utenti.

Un aspetto molto importate da tenere in considerazione sono le modalità di interazione tra gli utenti, questa è una delle condizioni più caratterizzanti di un sistema di e-democracy. Abbiamo visto modelli fortemente interattivi come Airesis e altri più centralizzati e con un'interazione più limitata come OpenDCN. Airesis è anche l'unica applicazione a consentire una qualche forma di anonimato, questo aspetto non viene preso in considerazione, forse volutamente, dagli altri sistemi.

Le modalità di espressione del voto sono simili tra le varie applicazioni: 2 sfruttano esplicitamente una modalità ad ordinamento con possibilità di pareggi, la terza, OpenDCN, sfrutta invece un sistema di votazione a punteggi che può facilmente essere ricondotto ad un sistema ad ordinamento (valutazioni ordinate da quella con più punti a quella con meno punti)

Per quanto riguarda la dimensione algoritmica, due dei sistemi analizzati sfruttano l'algoritmo di Schultze (1.4.3) per estrarre un vincitore unico, mentre OpenDCN sfrutta un approccio di selezione iterativa basata sull'analisi del fronte di pareto (1.4.5), in questo caso non necessariamente si avrà un unico vincitore. Da notare che nessuno dei sistemi analizzati permette di configurare in alcun modo l'algoritmo di calcolo dei risultati.

Per poter comparare i vari algoritmi, nel prossimo capitolo, procederò alla progettazione e realizzazione di un simulatore di sistemi di voto.

Capitolo 3

Analisi del simulatore

Fin'ora abbiamo individuato un modello per la democrazia e abbiamo visto come questo modello viene implementato dai principali sistemi di voto attualmente disponibili. Per poter procedere con la comparazione e con l'Individuazione di nuovi modelli democratici, nel corso di questo capitolo, progetterò e realizzerò un simulatore che permetta di implementare i modelli sin qui individuati e di valutarne di nuovi

Il modello presentato si divide in 4 caratteristiche:

1. Presentazione delle proposte
2. Interazione
3. Voto delle proposte
4. Algoritmi di selezione dei risultati

La prima fase, quella di presentazione delle proposte/candidature riguarda più che altro questioni amministrative e organizzative, quindi nella realizzazione del simulatore ometterò questa fase che deve essere considerata come un operazione propedeutica alla votazione.

La fase di interazione rappresenta un punto cruciale del nostro modello. Questa fase è difficilmente circoscrivibile entro confini definiti, non parliamo di un algoritmo o di una procedura che possano essere direttamente proiettati in una implementazione software. L'interazione tra gli utenti deve essere vista come un sistema complesso nel quale il numero di interazioni e le loro modalità non sono definibili a priori. Risulta quindi impossibile valutare tutte le possibili tipologie di interazione e i loro effetti ma è comunque possibile valutare il fenomeno in termini olistici.

Nell'analisi effettuata nella costruzione del modello e nelle implementazioni nei vari software ho mostrato come la fase di interazione porta al confronto i vari utenti su delle proposte, questo confronto può portare a due effetti, il primo è la modifica/integrazione delle proposte con nuove funzionalità; il secondo riguarda la modifica del voto conseguente a valutazioni più approfondite fatte sulle varie proposte.

Mentre l'effetto di modifica ed evoluzione di una proposta risulta difficilmente simulabile, l'effetto dello spostamento di voti è un fenomeno che può essere valutato complessivamente e quindi trasposto all'interno del nostro simulatore.

In un sistema equamente distribuito nel quale ci siano un egual numero di sostenitori per ogni proposta e ogni sostenitore riesca a "convincere" un egual percentuale di persone a votare per la propria favorita, l'effetto complessivo sarebbe nullo. In realtà le capacità comunicative dei sostenitori, l'esposizione mediatica ed innumerevoli altri fattori, possono spostare voti a favore di una proposta piuttosto che un'altra.

Nel simulatore che andrò a implementare partirò quindi da questo presupposto: a parità di idee di voto dei votanti la presenza di interazioni di varia natura a sostengono di una proposta piuttosto che un'altra potrà alterare con una certa probabilità l'esito della votazione. A questo punto le domande a cui rispondere sono:

- Come mappare le intenzioni di voto?
- Come modificare le intenzioni di voto a seguito della fase di interazione?

Iniziamo con il valutare il meccanismo di rappresentazione delle intenzioni di voto. Come abbiamo visto nel primo capitolo il meccanismo più comune di indicazione della preferenza è quello ad ordinamento, ad esempio dati 4 candidati A, B, C e D un voto potrebbe essere espresso come:

$$D > C > A = B$$

Questa rappresentazione del voto ci dà una chiara indicazione sull'ordine delle preferenze di ogni candidato, non fornisce però alcuna indicazione sulla "distanza" tra la prima e la seconda preferenza o tra la seconda e la terza. Per studiare un algoritmo di modifica del voto abbiamo bisogno di una valutazione che permetta di quantificare la distanza tra ogni candidato. Supponiamo quindi di dare un valore numerico alle preferenze per ogni candidato ad esempio presi i 4 candidati precedenti assegniamo una valutazione da 0 a 5, in questo caso avremo una rappresentazione delle intenzioni di voto di questo tipo:

$$A \longrightarrow 1 \qquad B \longrightarrow 1 \qquad C \longrightarrow 4 \qquad D \longrightarrow 5$$

ora, con una questa rappresentazione, siamo in grado di valutare l'effettiva distanza tra ogni candidato. Valutando questo esempio possiamo tranquillamente affermare che debba essere più "semplice" modificare l'opinione del votante per convincerlo che la proposta C sia la più valida piuttosto che la proposta A o B.

Proviamo ora ad ipotizzare un algoritmo che gestisca l'influenza verso una preferenza piuttosto che un'altra. L'influenza verso una certa proposta dovrà avvenire in probabilità. Assumiamo che solo una parte dei votanti sia disposta a modificare la propria opinione e, anche qualora lo facesse, difficilmente questo cambio sarà radicale, sarà molto più probabile che la prima scelta venga sostituita dalla seconda piuttosto che dall'ultima.

Avendo espresso le intenzioni di voto in termini numerici possiamo assumere che l'influenza verso una certa proposta non faccia altro che modificare il punteggio assegnato alla proposta di una certa quantità e con una certa probabilità. Nell'esempio precedente se aggiungiamo un influenza verso il candidato A la probabilità che il voto passi dall'attuale 1 a 2 dovrà essere maggiore rispetto alla probabilità di passaggio a un voto 3 o 4.

Per quanto detto fin'ora possiamo estrarre le caratteristiche essenziali dell'algoritmo di influenza come:

- L'algoritmo viene applicato con una certa probabilità derivante dall'effetto cumulativo delle interazioni del sistema
- L'algoritmo altera le intenzioni di voto migliorando la valutazione data del candidato sostenuto dall'influenzante
- L'entità della modifica del voto dovrà essere tanto meno probabile quanto più è grande

Per l'algoritmo di influenza si possono pensare diverse funzioni che rispettino i requisiti, nel nostro caso ho scelto di implementare la funzione di influenza come una funzione di probabilità gaussiana. L'applicazione della funzione di influenza porterà a una variazione del valore determinato da un valore casuale sulla gaussiana positiva centrata in 0, la larghezza della gaussiana sarà parametrizzabile e rappresenterà il meccanismo con cui determinare l'ammontare dell'effetto dell'influenza.

per parametrizzare la gaussiana verrà richiesto in input il dato relativo al valore di una σ, per le proprietà della gaussiana il 68,3% delle variazioni sarà $< \sigma$, il 95,5% sarà entro 2σ e il 99,7% entro 3σ

Passiamo ora all'espressione del voto. Come abbiamo indicato nel capitolo 1 sono possibili 3 modalità di espressione del voto che si differenziano per la quantità di informazioni inviate dal votante al sistema:

1. Voto di preferenza
2. Voto ad ordinamento
3. Voto pesato

Nel simulatore voglio supportare tutti e tre i tipi di votazione, inoltre dobbiamo considerare che gli algoritmi di voto potrebbero richiedere uno specifico tipo voto, dovremo quindi identificare un meccanismo per mappare ogni tipologia in ogni altra.

Iniziamo analizzando il voto pesato: in questo caso avendo espresso le intenzioni di voto come un valore numerico, la trasposizione al voto pesato è automatica. Per quanto riguarda il voto ad ordinamento possiamo procedere semplicemente ordinando i candidati da quello con la votazione più alta a quello con la votazione minore. Infine per il voto a preferenza sarà sufficiente scegliere il candidato con il voto maggiore. Su quest'ultimo tipo di voto è necessaria una puntualizzazione: non avendo posto alcuna limitazione all'espressione di voto sui vari candidati, è possibile che si verifichi il caso di un pareggio nella prima posizione, questa condizione non è accettabile per il tipo di voto a preferenza e quindi il voto sarà considerato nullo.

Per quanto riguarda gli algoritmi, implementerò tutti quelli elencati nel capitolo 1 oltre ad un semplice algoritmo maggioritario. I vari algoritmi prevedono un certo tipo di informazione in input, ad esempio quelli derivati da Condorcet richiedono che il voto sia espresso in forma di ordinamento mentre l'algoritmo maggioritario richiede un'unica preferenza, per questo vanno previsti dei meccanismi di conversione da un tipo di voto ad un altro.

Iniziamo valutando gli algoritmi che richiedono voti in ordine di preferenza, nel caso il voto sia stato espresso in maniera pesata sarà sufficiente ordinare le preferenze da quella col punteggio maggiore a quella col punteggio minore. Nel caso il voto sia stato espresso con un preferenza singola possiamo trasformare le informazioni in maniera ordinata ponendo la preferenza al primo posto e gli altri candidati al secondo posto.

Ancor più semplice è il caso di algoritmi che richiedono la presenza di una preferenza unica (l'algoritmo maggioritario), in questo caso basterà selezionare il favorito della votazione: per la votazione ordinata prenderemo il primo elemento; partendo

da una votazione pesata prenderemo il candidato con il voto maggiore. Nel caso di un pareggio il voto sarà considerato nullo.

Oltre alle considerazioni fatti fatte sinora, sarebbe interessante introdurre due concetti aggiuntivi:

- La topologia del sistema elettorale
- Le votazione in più turni

Con topologia del sistema elettorale intendo l'organizzazione, generalmente territoriale con la quale vengono raccolte le votazioni. L'introduzione di questo concetto può risultare conveniente quando valutiamo l'effetto delle interazioni, una particolare circoscrizione elettorale potrebbe essere influenzata in maniera differente dalle altre.

Aggiungiamo anche il concetto di votazione a più turni, potremmo stabilire che dal primo turno vengano estratti due candidati che poi andranno al ballottaggio. Il meccanismo dei turni non deve necessariamente vincolare la scelta dell'algoritmo del secondo turno a quello del primo. Per la gestione dei turni assumerò che il voto espresso al primo turno rimanga invariato, anche se limitato ai soli candidati rimasti, anche nel secondo turno (e teoricamente terzo ecc...). Esempio: Il voto al primo turno $A > B > C > D$ se filtrato per i candidati B e D diventerà $B > D$ al secondo.

Ricapitoliamo i requisiti individuati:

- Agenti in grado di influenzare il voto, da qui in poi mi riferirò a queste entità come "influencer"
- Diversi sistemi di espressione del voto
- Implementazione di vari algoritmi
- Gestione di votazioni in più turni
- Gestione di una topologia del sistema elettorale

Capitolo 4

Realizzazione del simulatore

Vista la struttura del simulatore, questo si presta molto bene ad una implementazione distribuita. Per la realizzazione ho scelto di usare TuCSoN con il quale ho già lavorato in passato e che permette di organizzare il sistema di voto separando nettamente le responsabilità tra i vari agenti in gioco.

Dati i requisiti individuati la configurazione di una simulazione dovrà contenere:

- Espressioni di voto: abbiamo precedentemente indicato che i voti andranno gestiti con l'attribuzione di un punteggio ai vari candidati
- Per gestire la topologia della rete introduciamo il concetto di seggio, inteso come unità di raccolta dei voti. La configurazione dovrà indicare la distribuzione dei voti tra i vari seggi.
- Per la gestione dei turni andranno definiti: il numero di turni e il numero di candidati estratti alla fine di ogni turno
- Per ogni turno andranno definiti il tipo di votazione (Preferenza, Ordinata o Pesata) e l'algoritmo da utilizzare, con l'eventuale configurazione, per l'estrazione dei vincitori di quel turno
- Per quanto riguarda gli influencer, deve essere indicato: il candidato di riferimento; la probabilità (σ della gaussiana); il seggio/i seggi a cui si applica

Tutte queste configurazioni verranno suddivise in diversi file per massimizzare la modularità del sistema.

Configurazione voti

La configurazione dei voti prevede due tipi di file:

- un meta file con informazioni generali sulla votazione.
- n file, uno per ogni seggio contente l'elenco dei voti.

Inoltre andranno definiti:

- un file contenente la configurazione dei turni.
- un file contenente la configurazione degli influencer.

Di seguito elenco in dettaglio un esempio delle varie configurazioni del simulatore:

```
{
  "candidate" : [ "A", "B", "C", "D", "E" ],
  "pollingStations" : {
    "0" : 563,
    "1" : 823,
    "2" : 920,
    "3" : 688,
    "4" : 701
  },
  "totalVote" : 3695,
  "maxValue" : 10
}
```

Listato 4.1: Meta file della votazione

Il file json nel listato 4.1 contiene l'indicazione dei candidati, del numero totale dei voti e dei voti per ogni seggio.

```
A:4;B:8;C:9;D:6;E:9
A:3;B:0;C:9;D:9;E:5
A:4;B:6;C:7;D:4;E:2
A:4;B:9;C:4;D:4;E:9
A:8;B:3;C:2;D:4;E:3
A:9;B:2;C:3;D:5;E:0
```

```
A:0;B:4;C:7;D:0;E:7
A:0;B:4;C:5;D:6;E:3
```

Listato 4.2: File della votazione

Il file csv nel listato 4.2 contiene l'elenco dei voti espressi. Per ogni candidato viene indicato un punteggio che, come specificato in precedenza, definisce la desiderabilità del candidato da parte del votante.

```
{
  "turn" : [ {
    "results" : 3,
    "voteType" : "Wighted"
  }, {
    "results" : 2,
    "voteType" : "Ordered"
  }, {
    "results" : 1,
    "voteType" : "SinglePreference"
  } ],
  "algConfs" : [ {
    "name": "Borda",
    "pollingStationAlgorithm" : "Borda",
    "extraConfig" : { }
  }, {
    "name": "Condorcet",
    "pollingStationAlgorithm" : "Condorcet",
    "extraConfig" : { }
  }, {
    "name": "Copeland",
    "pollingStationAlgorithm" : "Copeland",
    "extraConfig" : { }
  }, {
    "name": "Contucci 1",
```

```
26      "pollingStationAlgorithm" : "Contucci",
27      "extraConfig" : {
28        "AvoidTie" : "true",
29        "FrontierSelectionCriteria" : "MinimizeMu"
30      }
31    }, {
32      "name": "Contucci 2",
33      "pollingStationAlgorithm" : "Contucci",
34      "extraConfig" : {
35        "AvoidTie" : "true",
36        "FrontierSelectionCriteria" : "EqualizeMuAndSigma"
37      }
38    } ]
39  }
```

Listato 4.3: Configurazione dei turni

Il file del listato 4.3 definisce i turni egli algoritmi utilizzati, nello specifico vengono indicati:

- Gli algoritmi utilizzati
- Le configurazioni aggiuntive per i vari algoritmi
- Il tipo di votazione per ogni turno
- Il numero di vincitori alla fine di ogni turno

```
1  [ {
2    "pollingStationProb" : {
3      "1" : 1,
4      "2" : 0.5,
5      "4" : 2
6    },
7    "candidate" : "B"
8  } ]
```

Listato 4.4: Configurazione degli influencer

Il file di configurazione nel listato 4.4 contiene la definizione di vari influencer per ognuno di essi và indicato il candidato di riferimento (il candidato verso il quale verranno spostati dei voti) e le probabilità di applicazione per ogni seggio.

Il progetto del simulatore è suddiviso in diversi moduli, l'organizzazione della build è gestita tramite Maven [11]. I progetti che compongono il simulatore sono:

- voting-system-simulator-builder: progetto padre che contiene solo i riferimenti ai moduli figli
- voting-algorithm: contiene l'implementazione di tutti gli algoritmi di calcolo dei risultati di voto
- voting-system-simulator-core: il simulatore vero e proprio
- voting-system-simulator-gui: una semplice GUI che permette di interagire con il simulatore
- voting-launcher: lanciatore da console delle simulazioni
- voting-web-gui: Gui web per il lancio e la navigazione dei dati
- voting-test-generator: questo è un progetto di utility che è stato usato in fase di sviluppo per generare dei test case

4.1 Implementazione degli algoritmi

L'implementazione dei vari algoritmi è contenuta nel progetto *voting-algorithm* che contiene inoltre tutti i dto, le enumerazione e le principali interfacce che rappresentano le varie entità del sistema.

Tutti gli algoritmi implementano *Algorithm* (4.5), un'interfaccia con un generics che andrà implementato con il tipo di voto richiesto. I valori possibili per T sono: $OrderedVote$ $WeighedVote$ $SinglePreferenceVote$, ognuna di queste interfacce rappresenta uno dei tipi di voto precedentemente esposti. Poiché abbiamo stabilito precedentemente che ogni voto, comunque espresso, dovrà poter essere passato come parametro ad ognuno degli algoritmi implementati, ogni implementazione concreta implementa tutte e tre le interfacce che, per semplicità, sono state raccolte nell'interfaccia $Vote$ in figura (Figura 4.1) il diagramma uml che mostra la struttura dei voti e delle loro implementazioni concrete.

```
public interface Algorithm<T extends BaseVote> {

    void addVote(T v);

    void addVotes(List<T> votes);

    Winner getWinner() throws InvalidWinnerException;

    Winner getWinner(int n) throws InvalidWinnerException;

    int getCount();

    int getInvalid();
}
```

Listato 4.5: Interfaccia Algorithm

```
public interface OrderedVote extends Serializable {
    /**
     * @return Ritorna l'ordinamento dei candidati
     * dal preferito al meno desiderabile
     */
    List<VoteItem> getOrder();
}
```

Listato 4.6: Interfacce dei voto a ordinamento

```
public interface WeighedVote extends Serializable {
    /**
     * @return Ritorna una mappa contenente
     * per ogni candidato il voto assegnatogli
     */
    Map<String, Integer> getVotes();
}
```

Listato 4.7: Interfacce dei voto pesato

```
public interface SinglePreferenceVote extends Serializable {
    /**
     * @return Il candidato selezionato
     */
    String getPreference();
```

```
}
```

Listato 4.8: Interfacce voto a preferenza singola

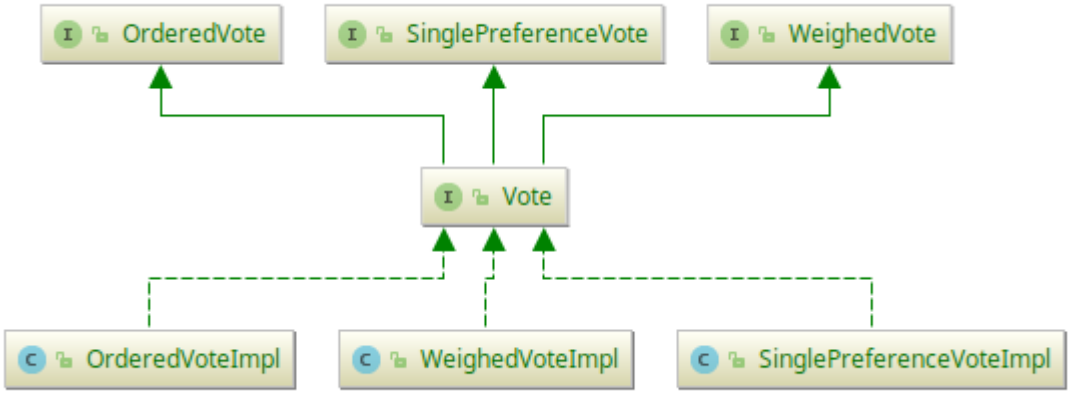


Figura 4.1: Uml dei tipi di voto

Ometto per brevità di riportare nel dettaglio tutte le implementazioni che sono comunque consultabili dal repository git del progetto.

Gli algoritmi implementati sono quelli illustrati nella sezione 1.4, ogni algoritmo accetta un certo tipo di voto tra quelli appena esposti

4.1.1 Condorcet

L'algoritmo di Condorcet si base su voti di tipo ordinamento.

Il calcolo dell'algoritmo di Condorcet si basa sulla costruzione di una matrice contenente i risultati di tutti i confronti di coppia come differenza tra il numero di vittorie e il numero di sconfitte. La matrice ha un numero di righe/colonne pari al numero di candidati. La costruzione della matrice avviene dinamicamente all'aggiunta di ogni nuova votazione, in questo modo lo stato interno della classe non deve memorizzare tutti i voti riportati.

Come esempio di costruzione della matrice, ipotizziamo una votazione con tre candidati A, B e C. Se aggiungiamo un primo voto $A > B > C$ la matrice sarà:

$$\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$

Se ora aggiungiamo un primo voto $C > B > A$ la matrice diventa:

$$\begin{bmatrix} 0 & 2 & 0 \\ -2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

In questo caso abbiamo che A batte B 2 volte (e di conseguenza B viene battuto due volte da A), tutti gli altri confronti sono in pareggio

Per quanto riguarda il calcolo del vincitore, la soluzione viene costruita per passi successivi: come prima cosa valuta, usando la matrice dei confronti, quale candidato non è mai battuto da nessuno, poi procede cercando quei candidati che sono battuti solo dal primo classificato, poi quelli che sono battiti dal primo e dal secondo classificato e così via fino a costruire la soluzione completa. Ovviamente, come abbiamo visto nel capitolo 1, nel calcolo del vincitore di Condorcet bisogna prevedere l'esistenza di cicli, quando un ciclo viene rilevato la computazione viene arrestata e ci si limita a fornire una soluzione parziale.

```
public class AlgorithmCondorcet extends BaseOrderedAlgorithm {
    final int[][] matrix;

    public AlgorithmCondorcet(List<String> candidates) {
        super(candidates);
        matrix = new int[candidates.size()][candidates.size()];
    }

    @Override
    protected void addVoteInternal(OrderedVote v) {
        for (int c1 = 0; c1 < candidates.size(); c1++) {
            for (int c2 = c1 + 1; c2 < candidates.size(); c2++) {
                int ind1 = AlgorithmUtils.indexOf(candidates.get(c1), v);
                int ind2 = AlgorithmUtils.indexOf(candidates.get(c2), v);

                int compare = Integer.compare(ind2, ind1);
                matrix[c1][c2] += compare;
                matrix[c2][c1] += -compare;
            }
        }
    }

```

```
    @Override
24  public Winner getWinner() {
        List<VoteItem> votes = new ArrayList<>();
26      List<Integer> exclude = new ArrayList<>();

28      while (exclude.size() < matrix.length) {
            List<Integer> w = findMainWinner(exclude);
30          if (w.isEmpty()) return new Winner(true, votes);
            votes.add(new VoteItem(w.stream().map(candidates::get).collect(Collectors.toList(
32          exclude = votes.stream()
                    .flatMap(voteItem -> voteItem.getCandidates().stream())
34                  .map(candidates::indexOf)
                    .collect(Collectors.toList());
36      }

38      return new Winner(votes);
    }
40
    private List<Integer> findMainWinner(List<Integer> exclude) {
42      int numCandidate = matrix.length;
        ext:
44      for (int c1 = 0; c1 < numCandidate; c1++) {
            List<Integer> ret = new ArrayList<>();
46          if (exclude.contains(c1)) continue;
            for (int c2 = 0; c2 < numCandidate; c2++) {
48              if (exclude.contains(c2)) continue;
                if (matrix[c1][c2] < 0) continue ext;
50              if (matrix[c1][c2] == 0) ret.add(c2);
            }
52          return ret;
        }
54      return Collections.emptyList();
    }
56
}
```

Listato 4.9: Implementazione dell'algoritmo di Condorcet

4.1.2 Copeland

L'algoritmo di Copeland è stato implementato come un'estensione di quello di Condorcet, come la classe padre richiede voti di tipo ordinamento.

La costruzione della matrice avviene esattamente come nell'algoritmo di Condorcet, la differenza tra i due risiede nel calcolo del vincitore. Per Copeland vengono calcolate tutte le vittorie di ogni candidato nei confronti di coppia e vengono sottratte tutte le sconfitte.

Per come è stata realizzata la matrice, possiamo calcolare il numero di vittorie del candidato i come il numero di celle con valore > 0 nella riga i-esima meno il numero di celle con valore < 0 nella riga i-esima Una volta effettuato il calcolo, la sequenza vincitrice si ottiene semplicemente ordinando dal candidato col maggior numero di vittorie a quello con meno vittorie.

```
1  public class AlgorithmCopeland extends AlgorithmCondorcet {

3      public AlgorithmCopeland(List<String> candidates) {
           super(candidates);
5      }

7      @Override
       public Winner getWinner() {
9          Map<String, Integer> vote = new HashMap<>(matrix.length);

11         for (int i = 0; i < matrix.length; i++) {
               int val = Arrays.stream(matrix[i]).map(Integer::signum).sum();
13             vote.put(candidates.get(i), val);
           }
15
           Map<Integer, List<String>> revMap = vote.entrySet().stream()
17             .map(e -> new AbstractMap.SimpleEntry<>(e.getValue(), e.getKey()))
               .collect(
19                 groupingBy(Map.Entry::getKey, mapping(Map.Entry::getValue, toList()))
               );
21         List<VoteItem> item = revMap.entrySet().stream()
               .sorted(Comparator.comparing(Map.Entry::getKey, Comparator.reverseOrder()))
23             .map(Map.Entry::getValue)
               .map(VoteItem::new)
25             .collect(toList());
           return new Winner(item);
27     }
```

```
29 }
```

Listato 4.10: Implementazione dell'algoritmo di Copeland

4.1.3 Tideman

Anche l'algoritmo di Tideman è implementato come estensione di quello di Condorcet.

Come spiegato nel capitolo 1 l'algoritmo di Tideman costruisce la soluzione per passi successivi partendo dai confronti con margine maggiore. I vari confronti vengono quindi ordinati usando la matrice dei confronti, da quelli con il valore nella cella maggiore a quelli con il valore minore. Una volta ordinati i vari vincoli vengono via via aggiunti alla soluzione. Il calcolo del vincitore di Tideman viene realizzato con l'ausilio della libreria JGrapht [8] che fornisce algoritmi e classi di supporto per la gestione dei grafi. L'aggiunta di vincoli viene gestita come la costruzione di un grafo direzionato, all'aggiunta di ogni nuovo vincolo si verifica se sono presenti cicli, in quel caso il nuovo vincolo vine ignorato. Da notare che l'algoritmo di Tideman non gestisce la presenza di pareggi nella soluzione, nel caso in cui più vincoli abbiamo la stessa priorità la scelta di quale applicare è casuale.

```
1  public class AlgorithmTideman extends AlgorithmCondorcet {

3      public AlgorithmTideman(List<String> candidates) {
           super(candidates);
5      }

7      @Override
       /*
9        L'algoritmo di estrazione del vincitore costruisce una lista con tutti i confronti
         di coppia e il loro esito.
11       La lista viene realizzata basantosi sulal mappa dei confronti

13       Una volta ottenuta la lista dei confronti questa viene ordinata dal confronto
         con margine maggiore a quella con margine minore.
15       Infini la costruzione della soluzione e demandata alla classe TidemanSolutionBuilder
         che permette di aggiungere i vari confronti ed evita la nascita di cicli
17
         In caso di pareggio viene valutato solo il primo dei confronti
19      */
```

```
    public Winner getWinner() {
        List<Vs> comparison = new ArrayList<>(matrix.length * matrix.length / 2);
        for (int c1 = 0; c1 < matrix.length; c1++) {
            for (int c2 = 0; c2 < matrix.length; c2++) {
                if (c1 == c2) continue;
                String first = candidates.get(c1);
                String second = candidates.get(c2);
                comparison.add(new Vs(first, second, matrix[c1][c2]));
            }
        }

        TidemanSolutionBuilder sb = new TidemanSolutionBuilder(candidates);

        comparison.stream()
            .sorted((o1, o2) -> Integer.compare(o2.point, o1.point))
            .forEach(i -> sb.add(i.firstCandidate, i.secondCandidate));

        return sb.result();
    }

    /**
     * Classe di utility per gestire i confronti
     */
    private class Vs {
        private final String firstCandidate;
        private final String secondCandidate;
        private final int point;

        Vs(String firstCandidate, String secondCandidate, int point) {
            this.firstCandidate = firstCandidate;
            this.secondCandidate = secondCandidate;
            this.point = point;
        }
    }

}
```

Listato 4.11: Implementazione dell'algoritmo di Tideman

4.1.4 Schultze

L'implementazione dell'algoritmo di Schultze usa un approccio differente rispetto agli altri metodi di Condorcet. In questo cosa all'aggiunta di un nuovo voto non viene popolata una matrice ma una semplice mappa contenente il numero complessivo di vittorie per ogni candidato. Per estrarre la soluzione si ordinano semplicemente i candidati da quello che a ottenuto più vittorie a quello che ne ha ottenute di meno.

```
1  public class AlgorithmSchultze extends BaseOrderedAlgorithm {
       private final Map<String, Integer> vote = new HashMap<>();
3
       public AlgorithmSchultze(List<String> candidates) {
5          super(candidates);
       }
7
       private Map<Integer, Integer> maps = new LinkedHashMap<>();
9
       @Override
11     protected void addVoteInternal(OrderedVote v) {
           for (String s : getCandidates()) {
13             vote.merge(s, countWinning(v, s), (i1, i2) -> i1 + i2);
               int p = AlgorithmUtils.indexOf("A", v);
15             maps.merge(p, 1, (a, b) -> a + b);
           }
17     }

19     @Override
       /*
21       L'algoritmo di scelta del vincitore si ottiene ordinando
         i candidati da quello che ha vinto piu' confronti
23       di coppia in maniera discendente
         Una volta ottenuto l'ordinamento si costruisce a soluzione
25      */
       public Winner getWinner() {
27         Map<Integer, List<String>> revMap = vote.entrySet().stream()
                   .map(e ->
29                     new AbstractMap.SimpleEntry<>(e.getValue(), e.getKey())
                   )
31                 .collect(
                       groupingBy(
33                         Map.Entry::getKey,
```

```
                        mapping(Map.Entry::getValue, toList())
35                  )
                );
37
        List<VoteItem> item = revMap.entrySet().stream()
                .sorted(Comparator.comparing(
                            Map.Entry::getKey,
41                          Comparator.reverseOrder())
                        )
43              .map(Map.Entry::getValue)
                .map(VoteItem::new)
45              .collect(toList());
        return new Winner(item);
47  }

49  /**
     * Metodo che calcola il numero di vittorie di un candidato
51   * nei confronti di coppia in un certo voto
     *
53   * @param v         Voto
     * @param candidate Candidato
55   * @return Numero di vittorie
     */
57  public static int countWinning(OrderedVote v, String candidate) {
        int ind = AlgorithmUtils.indexOf(candidate, v);
59      if (ind + 1 >= v.getOrder().size()) return 0;

61      return v.getOrder().subList(ind + 1, v.getOrder().size())
            .stream()
63          .mapToInt(voteItem -> voteItem.getCandidates().size()).sum();
    }
65  }
```

Listato 4.12: Implementazione dell'algoritmo di Schultze

4.1.5 Contucci

L'algoritmo di Contucci merita qualche precisazione rispetto a quanto illustrato nel Capitolo 1. Abbiamo visto come Contucci individui le soluzioni vincitrici tra quelle che minimizzano sia la varianza della distanza sia la mu.

Prima di procedere ad una implementazione dell'algoritmo dobbiamo stabilire come valutare quali soluzione considerare nell'insieme delle soluzioni ottime e como filtrare tra queste la soluzione vincitrice.

Per la scelta delle soluzioni ottime ho sfruttato il calcolo del fronte di pareto nel quale la regola di dominazione tra due soluzioni A e B è definita come:

$$\begin{cases} se \mu(A) < \mu(B) \quad e \quad \sigma(A) < \sigma(B) \longrightarrow A \quad domina \quad B (A \succ B) \\ se \mu(A) > \mu(B) \quad e \quad \sigma(A) > \sigma(B) \longrightarrow B \quad domina \quad A (B \succ A) \end{cases}$$

Una volta definite le soluzioni ottime, per l'individuazione della soluzione vincitrice ho individuato 3 possibilità:

1. Minimizzare μ: dato il sotto-insieme delle soluzioni ottime individuate dal fronte di pareto, selezioniamo quella che minimizza μ
2. Minimizzare σ: dato il sotto-insieme delle soluzioni ottime individuate dal fronte di pareto, selezioniamo quella che minimizza σ, on caso di pareggio, si sceglie la soluzione che minimizza anche μ
3. Minimizzare d: dato il sotto-insieme delle soluzioni ottime individuate dal fronte di pareto, per ognuna di esse calcolo la d come: $d = \sqrt{\mu^2 + \sigma^2}$ e seleziono la soluzione che minimizza il valore di d

Per quanto riguarda l'implementazione l'algoritmo per il calcolo di μ e σ è necessario avere a disposizione tutti i voti forniti, per ridurre la dimensione in memoria della lista, viene salvata una mappa contente la cardinalità di ogni votazione.

L'algoritmo può essere configurato con due parametri che permetto di definire:

- La modalità di scelta della soluzione tra quelle del fronte di pareto.
- Se includere o meno soluzioni con pareggi tra quelle valutate.

Per il calcolo delle sotto-soluzioni ottime l'algoritmo sfrutta una classe di supporto che si occupa di estrarre il fronte di pareto delle soluzioni fornite.

```
1 public class AlgorithmContucci extends BaseOrderedAlgorithm {
    private static final Logger log = LogManager.getLogger(AlgorithmContucci.class);
3   private final Map<OrderedVote, Integer> votes = new ConcurrentHashMap<>();
    private final AlgorithmContucciSettings settings;
5   private final List<OrderedVote> solutionSpace;

7   public AlgorithmContucci(List<String> candidates,
                             AlgorithmContucciSettings settings) {
```

```
9           super(candidates);
            this.settings = settings;
11          if (settings.isAvoidTie()) {
                solutionSpace = AlgorithmUtils.generateAllPermutationWithoutTie(candidates);
13          } else {
                solutionSpace = AlgorithmUtils.generateAllPermutation(candidates);
15          }
        }
17
        @Override
19      /*
            L'algoritmo di contucci richiede che per il calcolo
21          della soluzione si abbiano tutti i voti forniti,
            per ridurre la dimensione in memoria della lista,
23          salvo il voto con la relativa cardinalita'
         */
25      protected void addVoteInternal(OrderedVote vote) throws InvalidVoteException {
            OrderedVote newVote = AlgorithmUtils.normalize(candidates, vote);
27          votes.compute(newVote, (v, c) -> c != null ? c + 1 : 1);
        }
29
        @Override
31      public Winner getWinner() throws InvalidWinnerException {
            ParetoFrontier<ExtVote> pareto = new ParetoFrontier<>(
33                  (a, b) -> a.mu < b.mu && a.sigma < b.sigma
                );
35
            for (OrderedVote s : solutionSpace) {
37              double mu = AlgorithmUtils.mu(candidates, s, votes);
                double sig = AlgorithmUtils.sigma(mu, candidates, s, votes);
39              ExtVote tmp = new ExtVote(s, mu, sig);
                log.debug(tmp);
41              pareto.addItem(tmp);
            }
43
            List<ExtVote> frontier = pareto.getFrontier();
45          List<ExtVote> solutions;
            switch (settings.getFrontierSelectionCriteria()) {
47              case MinimizeMu:
                    double minMu = frontier.stream()
49                      .mapToDouble(a -> a.mu).min().orElse(0);
```

```
                solutions = frontier.stream()
51                  .filter(a -> a.mu == minMu)
                    .collect(Collectors.toList());
53                  break;
            case MinimizeSigma:
55                  double minSigma = frontier.stream()
                        .mapToDouble(a -> a.sigma)
57                      .min().orElse(0);
                    solutions = frontier.stream()
59                      .filter(a -> a.sigma == minSigma)
                        .collect(Collectors.toList());
61
                    double minMuSigma = solutions.stream()
63                      .mapToDouble(a -> a.mu).min()
                        .orElse(0);
65                  solutions = solutions.stream()
                        .filter(a -> a.mu == minMuSigma)
67                      .collect(Collectors.toList());
                    break;
69          case EqualizeMuAndSigma:
            default:
71                  Map<Double, List<ExtVote>> tmp = frontier.stream()
                        .collect(
73                          Collectors.groupingBy(
                                a -> Math.sqrt(Math.pow(a.mu, 2) + Math.pow(a.sigma, 2)
75                          )
                        )
77                  );
                    double minDist = tmp.keySet().stream().mapToDouble(a -> a).min().orElse(0);
79                  solutions = tmp.get(minDist);
                    break;
81      }

83
        if (solutions == null || solutions.size() == 0) {
85          String err = "Wrong solution, unable to compute a winner";
            log.error(err);
87          throw new InvalidWinnerException(err);
        }
89      if (solutions.size() > 1) {
            String err = "More than one solution, unable to compute a winner";
```

```
91              log.error(err);
                throw new InvalidWinnerException(err);
93          }
            return new Winner(solutions.get(0).vote.getOrder());
95      }

97      private static class ExtVote {
            private final OrderedVote vote;
99          private final double mu;
            private final double sigma;
101
            ExtVote(OrderedVote vote, double mu, double sigma) {
103             this.vote = vote;
                this.mu = mu;
105             this.sigma = sigma;
            }
107
            @Override
109         public String toString() {
                return vote + ";" + mu + ";" + sigma;
111         }
        }
113 }
```

Listato 4.13: Implementazione dell'algoritmo di Contucci

4.1.6 Maggioritario

Questo è l'unico tra gli algoritmi implementati a richiedere un tipo di voto a preferenza singola. L'algoritmo è piuttosto banale e si limita a calcolare il candidato che ha ottenuto più preferenze.

```
1
public class AlgorithmMajority extends BaseAlgorithm<SinglePreferenceVote> {
3       private final Map<String, Integer> results = new HashMap<>();

5       public AlgorithmMajority(List<String> candidates) {
            super(candidates);
7       }
```

```
9       @Override
        protected void addVoteInternal(SinglePreferenceVote vote)
11                                      throws InvalidVoteException {
            if (vote.getPreference() == null) return;
13              throw new InvalidVoteException("Invalid null vote");
            if (!candidates.contains(vote.getPreference()))
15              throw new InvalidVoteException("Invalid vote for candidate "
                        + vote.getPreference());
17          results.compute(vote.getPreference(), (k, v) -> v == null ? 1 : v + 1);
        }
19
        @Override
21      public Winner getWinner() {
            Map<Integer, List<String>> r = results.entrySet().stream()
23              .collect(Collectors.groupingBy(
                    Map.Entry::getValue
25                  Collectors.mapping(Map.Entry::getKey, Collectors.toList())
                )
27          );

29          Winner w = new Winner();
            w.setOrder(new ArrayList<>());
31          r.entrySet().stream()
                    .sorted((o1, o2) -> Integer.compare(o2.getKey(), o1.getKey()))
33                  .map(Map.Entry::getValue)
                    .forEach(list -> w.getOrder().add(new VoteItem(list)));
35          return w;
        }
37  }
```

Listato 4.14: Implementazione dell'algoritmo Maggioritario

4.1.7 Borda

L'algoritmo di Borda implementato rappresenta una variante di quello mostrato nel capitolo 1. Nella versione implementata Borda accetta un tipo di voto pesato, ogni votante può quindi assegnare un punteggio ad ogni candidato.

Il vincitore è il candidato che ha ottenuto il maggior punteggio.

```
1   public class AlgorithmBorda extends BaseAlgorithm<WeighedVote> {
```

```
    private final Map<String, Integer> results = new ConcurrentHashMap<>();

    public AlgorithmBorda(List<String> candidates) {
        super(candidates);
    }

    @Override
    protected void addVoteInternal(WeighedVote vote) throws InvalidVoteException {
        if (vote.getVotes() == null)
            throw new InvalidVoteException("Invalid null vote");
        vote.getVotes().forEach(
            (k, v) -> results.compute(k, (rk, rv) -> rv == null ? v : rv + v)
        );
    }

    @Override
    public Winner getWinner() {
        Map<Integer, List<String>> r = results.entrySet().stream(
                    .collect(
                        Collectors.groupingBy(
                            Map.Entry::getValue,
                            Collectors.mapping(
                                Map.Entry::getKey,
                                Collectors.toList()
                            )
                        )
                    );

        Winner w = new Winner();
        w.setOrder(new ArrayList<>());
        r.entrySet().stream()
                .sorted((o1, o2) -> Integer.compare(o2.getKey(), o1.getKey()))
                .map(Map.Entry::getValue)
                .forEach(list -> w.getOrder().add(new VoteItem(list)));
        return w;
    }
}
```

Listato 4.15: Implementazione dell'algoritmo di Borda

4.2 Simulatore

Il simulatore è realizzato all'interno del progetto voting-simulator-core che contiene tutta la logica applicativa. Il simulatore è stato realizzato utilizzando un approccio distribuito basato su TuCSoN [19] e Jade [7].

Grazie all'utilizzo del servizio per Jade "T4j" [20], gli agenti JADE hanno la possibilità di comunicare tramite un tuple centre, nel quale avverrà la maggior parte della coordinazione della simulazione.

La simulazione è stata progettata per permettere la distribuzione dei vari agenti, su nodi distribuiti. Avviando il programma con l'opzione *-remote {porta}*, creiamo un nodo della simulazione in attesa di ricevere richieste sulla porta passata come parametro.

Se avviato senza l'opzione remote sarà possibile passare come parametro al simulatore gli host a cui connettersi per creare la rete distribuita di calcolo. I vari agenti (nello specifico i vari seggi) saranno distribuiti tra i vari nodi.

Nel progetto trovano posto 3 tipi di agenti:

4.2.1 SimulationLoaderAgent

Questo agente si occupa di inizializzare tutte le componenti del simulatore e di avviare la simulazione. In particolare ha il compito di gestire la topologia degli agent container e la distribuzione dei vari agenti sulle diverse macchine. L'agente, una volta avviate tutte le componenti del sistema, si occupa di pubblicare la tupla di start nel tuple center che sancisce l'avvio della simulazione.

4.2.2 PolingStationAgent

Gli agenti di tipo PolingStationAgent rappresentano i seggi elettorali. Una volta avviato, ogni agente rimane in attessa della tupla di inizio turno; quando viene avviato un turno la polling station si occupa di inviare i voti sotto forma di tuple nel tuple center.

I voti vengono letti da file nel formato base indicato nel listato 4.2, una volta letti viene applicata l'azione degli influencer (se configurati), infine il voto viene convertito nel tipo richiesto e pubblicato nel tuple center.

Una volta letti e pubblicati tutti i voti, il seggio rimane in attessa del turno successivo.

4.2.3 ResultAgent

Il ResultAgent si occupa di raccogliere i voti dal tuple center e di passarli all'algoritmo della simulazione. Inoltre ha il compito di pubblicare la tupla di fine turno e i risultati attraverso il gestore di eventi [1]

L'algoritmo di applicazione degli influencer merita una puntualizzazione:

Algoritmo Influencer L'applicazione dell'effetto degli influencer è stata realizzata mediante l'utilizzo della classe *Random*. Come illustrato precedentemente l'implementazione si basa sul calcolo di un delta da applicare alla preferenza del candidato oggetto dell'influencer, il calcolo avviene mediante il metodo *nextGaussian*() della classe *Random* di seguito un estratto della classe VoteUtils con i metodi che si occupano dell'applicazione del calcolo.

```
/**
 * Applica la regola di influenza al voto
 *
 * @param wrapper   Wrapper del voto a cui applicare l'influenza
 * @param candidate Candidato per il quale applicare l'influenza
 * @param maxValue  Valore massimo della preferenza
 * @param std       parametro di definizione del livello di influenza
 */
public static void applyInfluence(VoteWrapper wrapper, String candidate,
                                  int maxValue, Double std) {
    try {
        for (Map.Entry<String, Integer> e : wrapper.getCandidatePoint().entrySet()) {
            if (!candidate.contains(e.getKey())) continue;
            Integer newVal = applyProb(e.getValue(), maxValue, std);
            if (Objects.equals(newVal, e.getValue())) continue;
            log.debug("Modified vote for candidate {} from {} to {}",
                    e.getKey(), e.getValue(), newVal);
            e.setValue(newVal);
            return;
```

[1] Per la gestione delle comunicazioni è stato implementato un semplice gestore di eventi, per maggiori dettagli rimando direttamente alla consultazione del codice sorgente.

```
20          }
        } catch (Exception e) {
22          log.error(e, e);
        }
24  }
```

Listato 4.16: Influencer

4.2.4 Sistema ad eventi

La componente core del simulatore è pensata come un modulo software da richiamare da altre applicazioni, per questa sua natura è utile pensare ad un meccanismo di comunicazione verso l'esterno che esuli dalla specifica implementazione. Data la natura distribuita del sistema ho deciso di implementare un sistema di eventi. La mia implementazione si basa principalmente su 3 entità:

Event L'interfaccia Event rappresenta un evento, non presenta alcun metodo e viene usata solo come raggruppamento identificativo degli eventi, riporto il banale codice per completezza.

```
interface Event {
2   }
```

Listato 4.17: Interfaccia evento

EventHandler Questa interfaccia permette di definire un gestore da agganciare ad un particolare evento, L'Implementazione dell'azione da eseguire sarà all'interno del metogo $run(T)$ dove T è l'evento che si intende intercettare . Il metodo $getType()$ permette all'EventDispatcher di instradare correttamente i vari eventi al loro gestore.

```
public interface EventHandler<T extends Event> {
2       Class<T> getType();
        void run(T event);
4   }
```

Listato 4.18: Interfaccia event handler

EventDispatcher Questa classe implementa un singleton a cui andranno agganciati i vari event handler configurati.

```
public class EventDispatcher {
    private static EventDispatcher singleton;
    private final Map<Class<? extends Event>, List<EventHandler>> eventsBus;

    private EventDispatcher() {
        eventsBus = new WeakHashMap<>();
    }

    public static EventDispatcher get() {
        if (singleton == null) {
            singleton = new EventDispatcher();
        }
        return singleton;
    }

    public <T extends Event> void addEventHandler(EventHandler<T> handler) {
        Class<T> eventType = handler.getType();
        eventsBus.computeIfAbsent(eventType, k -> new ArrayList<>());
        eventsBus.get(eventType).add(handler);
    }

    public <T extends Event> void fireEvent(T event) {
        List<EventHandler> handlers = eventsBus.get(event.getClass());
        if (handlers != null) {
            for (EventHandler<T> h : handlers) {
                h.run(event);
            }
        }
    }

}
```

Listato 4.19: Classe di gestione degli eventi

Gli eventi configurati per la simulazione sono:

- SimulationStateChangeEvent: evento lanciato ad ogni cambio di stato della simulazione, contiene una enum che indica lo stato corrente (Running, Stop)

- TurnStartEvent: evento lanciato all'inizio di ogni turno

- TurnEndEvent: evento lanciato alla fine di ogni turno, contiene anche informazioni statistiche relative al turno appena concluso

- ResultUpdateEvent: evento lanciato periodicamente contenente i risultati parziali per il turno in corso

- StatsEvent: evento contenente le statiche di voto delle varie polling station

Tutti gli eventi vengono lanciati dal ResultAgent, essendo in un sistema distribuito il ResultAgent deve essere sempre lanciato nella macchina di avvio della simulazione, in caso contrario non sarebbe possibile agganciarsi agli eventi nel dispatcher; Questo vincolo topologico è gestito all'interno del SimulatorAgent.

La realizzazione distribuita sfruttando gli agenti mi ha permesso di separare nettamente le competenze tra gli agenti: le polling station si occupano della pubblicazione dei voti e non hanno alcuna conoscenza dell'algoritmo che verrà applicato mentre il result agent si occupa esclusivamente di applicare l'algoritmo, ignorando completamente la topologia dei seggi o l'esistenza degli influencer.

4.3 GUI Swing

Per semplificare la fase di sviluppo e permettesse di verificate il funzionamento della gestione ad eventi, ho deciso di implementare una semplice interfaccia swing che mi permettesse di testare diverse configurazioni di lancio, non scenderò nei dettagli implementativi dell'interfaccia che risultano abbastanza banali e per la cui consultazione rimando al codice sorgente.

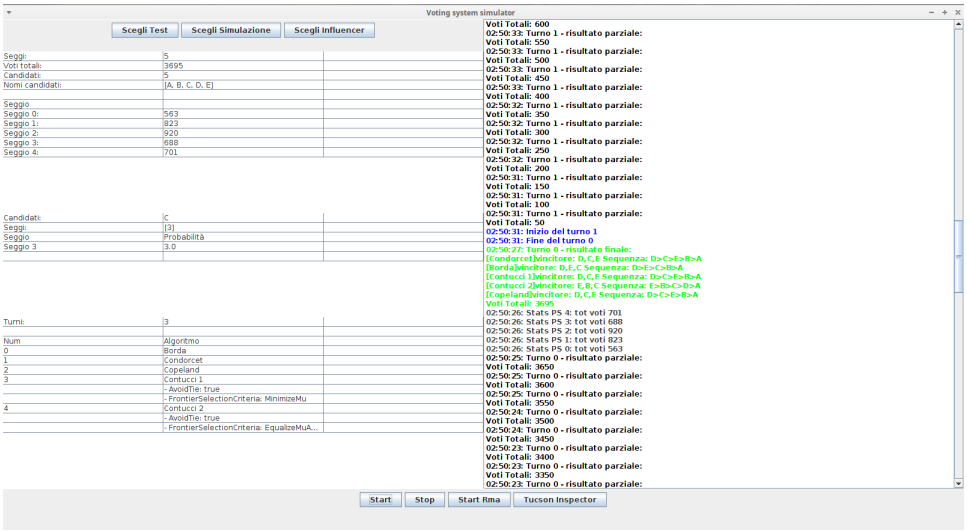


Figura 4.2: Interfaccia di test SWING

L'interfaccia permette di caricare i file di configurazione che vengono poi mostrati nella parte sinistra della GUI. Sulla destra viene visualizzato il log delle operazione gestito interamente grazie al sistema di eventi.

4.4 Il Lanciatore

Durante lo sviluppo del simulatore mi sono dovuto confrontare con un problema legato alla ripetibilità dei test. Per alcune simulazione avevo la necessità di eseguire lo stesso test più volte per verificare la consistenza dei risultati ottenuti, il sistema distribuito basato su TuCSon e Jade non consentiva per svariati motivi di terminare l'intero sistema (tuple center e jade container) e di riavviare la simulazione, questo portava inevitabilmente a delle inconsistenze nel sistema dopo un certo numero di simulazioni. Per ovviare a questo problema ho deciso di aggiungere un modulo software che faccia da strato di separazione tra la componente core e la GUI (GUI web che illustrerò nel paragrafo successivo). Il lanciatore verrà avviato in un JVM separata, questo mi permetterà di ripetere le simulazione senza mai incorrere in rischi di interferenza tra i vari agenti.

Il lanciatore è stato pensato per raccogliere i dati elaborati e immagazzinarli per successive analisi. I dati raccogli dagli eventi lanciati dal simulatore sono immagazzinati all'interno di un database Postgresql. La gestione del salvataggio delle

informazioni su db è stata realizzata grazie al framework JPA [9] nella sua implementazione eclipse-link [5], in figura (Figura 4.3) il diagramma ER della base dati.

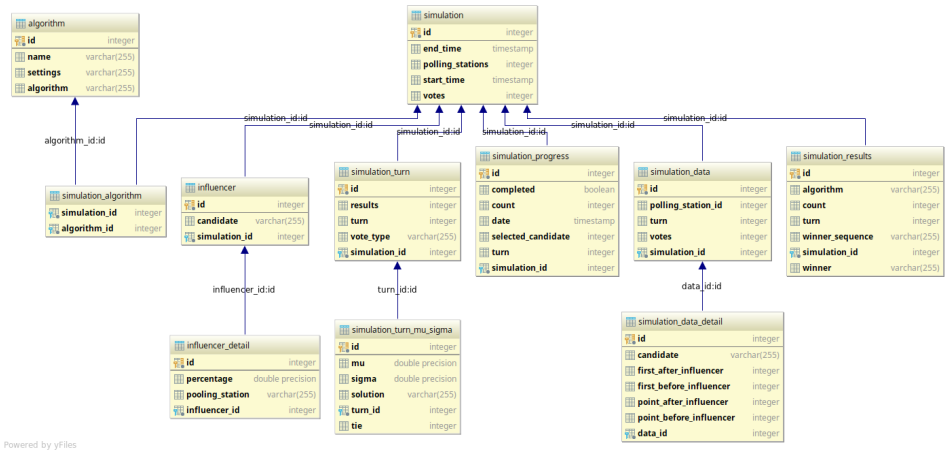


Figura 4.3: Diagramma ER della base dati implementata

Il lanciatore presenta due modalità di avvio, una tramite file i file di configurazione visti precedentemente (listati 4.1, 4.2, 4.3, 5.1) l'altra tramite la configurazione della simulazione nel database. In particolare questa seconda modalità verrà utilizzata per lo sviluppo della webgui.

4.5 La GUI web

Come artefatto finale della simulazione ho realizzato un'interfaccia web per il lancio delle simulazioni e la consultazione dei dati.

Il simulatore web è basato sul framework SpringBoot [17] usando Thymeleaf [18] come template engine per la realizzazione delle pagine web. Il progetto è composto da due sezioni, la sezione di lancio della simulazione e la sezione di consultazione dei dati consultabili attraverso il menù laterale. L'applicazione è stata predisposta per poter essere fruita sia da pc che da dispositivi mobile sfruttando le potenzialità responsive offerte dal framework css Bootstrap [3].

4.5.1 Launcher

Il lancio della simulazione permette di configurare i vari parametri del modello, l'interfaccia è suddivisa in diverse card ognuna delle quali permette di configurare determinati parametri di lancio. Le informazioni configurabili sono:

- Scelta di un test set (Figura 4.4), per ogni test disponibile vengono mostrati: il numero di seggi, il numero complessivo di voti, il numero di candidati e il valore massimo di preferenza espresso per ogni candidato

- Scelta degli algoritmi (Figura 4.5), è possibili indicare uno o più algoritmi da utilizzare per effettuare la simulazione, in caso di doppio turni multipli, la selezione dei candidati vincitori si baserà sul primo algoritmo selezionato

- Configurazione dei turni (Figura 4.6): si possono selezionare fino a tre turni distinti, per ognuno è possible definire il tipo di voto (preferenza singola, ordinamento, pesato) e il numero di vincitori selezionati.

- Configurazione degli influencer (Figura 4.7): una volta selezionato il test set è possibile configurare un arbitrario numero di influencer indicando il candidato che supportano e il livello di influenza per ogni seggio.

Una volta premuto il tasto avvia un popup avviserà che la simulazione è stata lanciata.

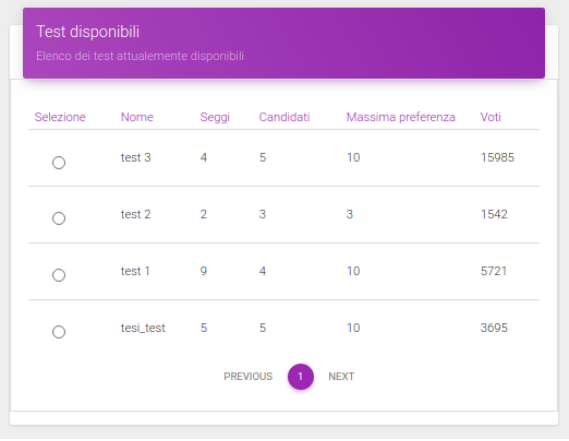


Figura 4.4: Scelta del test

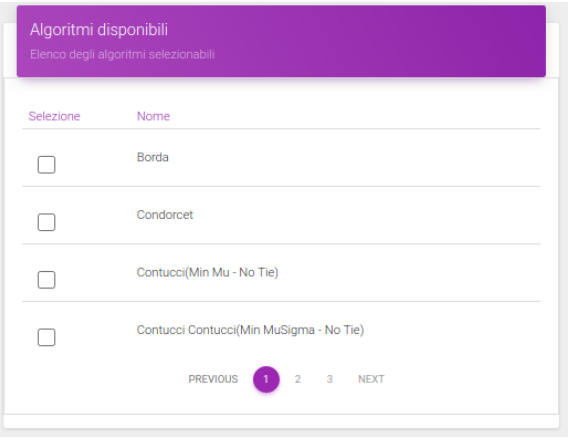


Figura 4.5: Selezione degli algoritmi

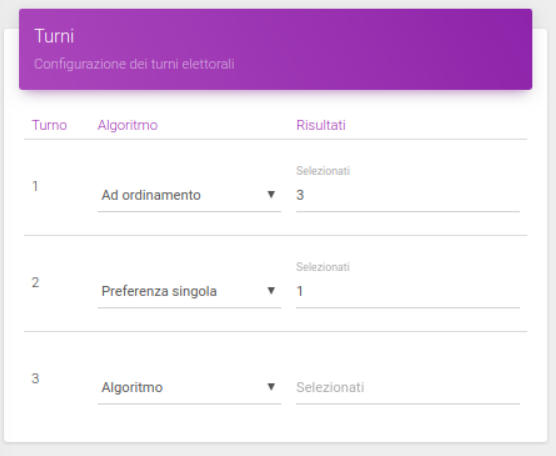


Figura 4.6: Configurazione dei turni

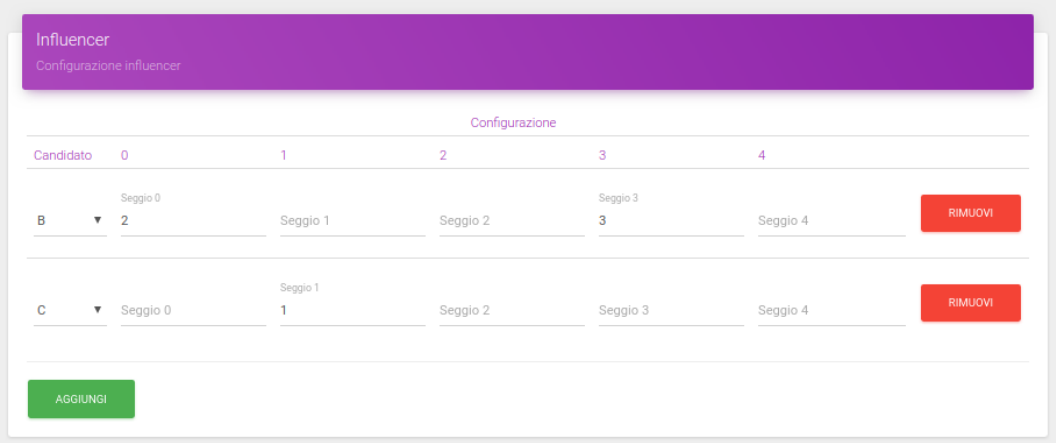


Figura 4.7: Configurazione degli influencer

4.5.2 Simulazioni

Nella sezione "Simulazioni" è presente una tabella con l'elenco delle simulazioni lanciate (Figura 4.8). Cliccando sul link "dettagli" si passerà alla pagina di dettaglio per la simulazione scelta.

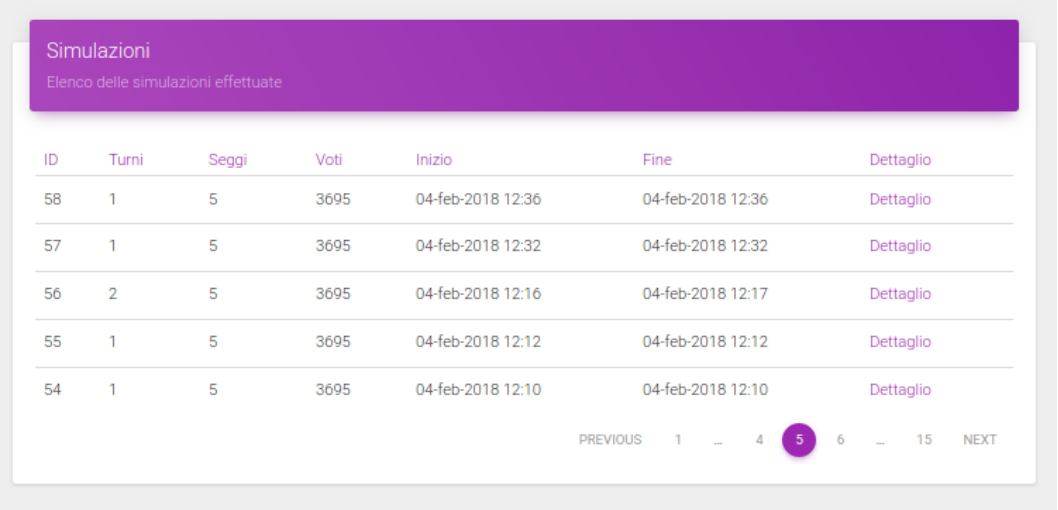


Figura 4.8: Tabella delle simulazioni

La pagina di dettaglio mostra informazioni avanzate sull'esito della simulazione e sul set di dati. Nella sezione in altro della finestra sono mostrate informazioni di carattere generale sulla simulazione: numero di candidati, numero di seggi, numero di turni e voti totali La sezione centrale (Figura 4.9) permette di navigare i dati per i vari seggi. Nel primo riquadro sulla sinistra sono mostrati in tabella il numero di voti per ogni seggio e il totale, cliccando su una qualunque delle righe della tabella i dati ne secondo e terzo riquadro verranno aggiorni mostrando i dati relativi al seggio selezionato o al totale. Il riquadro centrale mostra il numero di voti ottenuti come prima preferenza per ognuno dei candidati, da notare la presenza di due serie nel grafico una mostra i dati prima dell'applicazione dell'azione degli influencer (bianca), l'altra mostra il risultato dopo l'applicazione degli influencer (rossa). Cliccando sui pulsanti presenti sotto il grafico è possible selezionare il turno da mostrare. Il terzo riquadro ha un comportamento analogo al secondo ma quì vengono mostrate le preferenze cumulative per i vari candidati.

Figura 4.9: Sezione di analisi dati dei seggi

La terza sezione, in fondo alla pagina (Figura 4.10), permette di navigare i risultati. Nel riquadro a sinistra si trova la tabella contenente il risultato per ogni algoritmo. Per ogni turno vengono indicati i candidati vincitori, tra parentesi viene inoltre indicata la sequenza vincitrice. Nel riquadro a sinistra troviamo lo spazio di tutte e soluzioni possibili, il grafico riporta sull'asse delle ascisse la μ e quello delle ordinate σ come calcolate secondo l'algoritmo di contucci 1.4.6. Cliccando sulle righe della tabella degli algoritmi viene evidenziato in rosso il risultato all'interno del grafico. Anche in questo grafico è possibile navigare i turni tramite i relativi pulsanti. Infine è possibile filtrare solo i risultati senza pareggi al fine di migliorarne la leggibilità

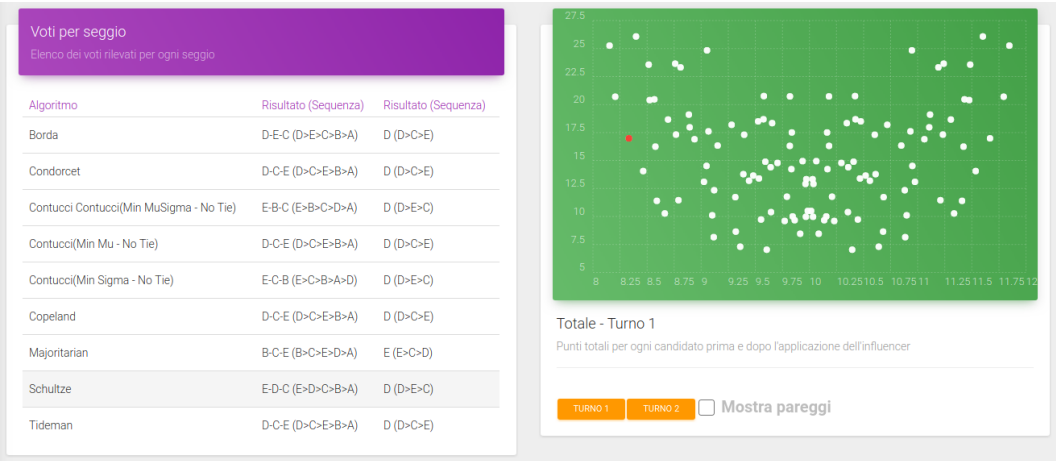


Figura 4.10: Sezione di analisi dei risultati della votazione

Capitolo 5

Comparazioni

Ora che abbiamo realizzato il simulatore, possiamo procedere con la realizzazione di alcune simulazioni.

Per rendere riproducibili in maniera semplice i risultati ottenuti ho creato un nuovo progetto: voting-test-generator, il cui compito è quello di estrapolare i dati delle simulazioni elencate in questo capitolo.

Ho creato un test case su cui fare alcune valutazioni. Il caso di test si riferisce ad un'ipotetica votazione atta a stabilire il luogo migliore in cui costruire un ospedale. I possibili cantieri si trovano in 5 città: A; B; C; D; E;

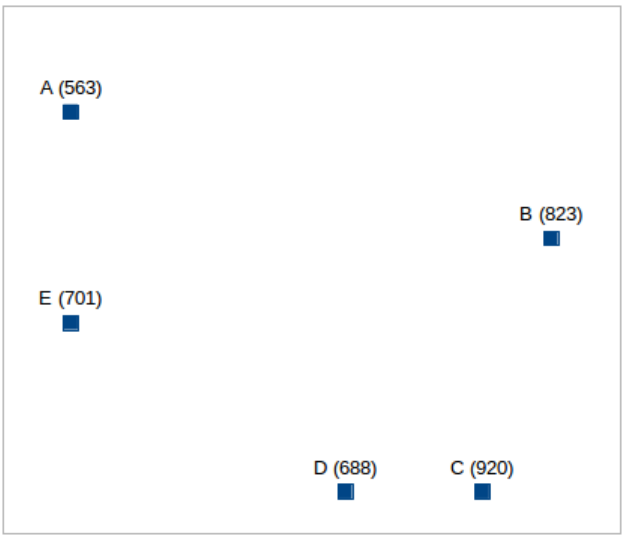
Nella realizzazione del test case ho assunto che ogni città abbia un numero differente di cittadini votanti e che le preferenze dei cittadini siano determinate dalla vicinanza del luogo di costruzione alla propria cittadina.

In figura 5.1 la distribuzione delle cittadine utilizzate per la realizzazione del caso di test.

Nella tabella 5.1 sono indicate le distanze relative tra le varie città.

	A	B	C	D	E
A	0	14	22	20	10
B	14	0	16	17	16
C	22	16	0	1	14
D	20	17	4	0	8
E	10	16	14	8	0
Tot	66	63	56	49	48

Tabella 5.1: Distanze relative tra le città



Città	Votanti
A	563
B	823
C	920
D	688
E	701
Totale	3695

Figura 5.1 & Tabella 5.2: Distribuzione votanti tra le varie città

Nella nostra simulazione, ogni cittadina rappresenterà un seggio elettorale, il numero di voti per ogni seggio sarà ovviamente determinato dal numero di abitanti.

Nelle figure sottostanti è possibile vedere la distribuzione di voti per la prima preferenza (5.2) e la distribuzione cumulativa di punti tra i vari votanti nei differenti seggi (5.3). In caso di pareggio al primo posto, per il primo set di grafici, sono stati calcolati tutte le prime prefrenze (es nel voto A=B=C>E>D A, B e C sono stati contati come prima preferenza)

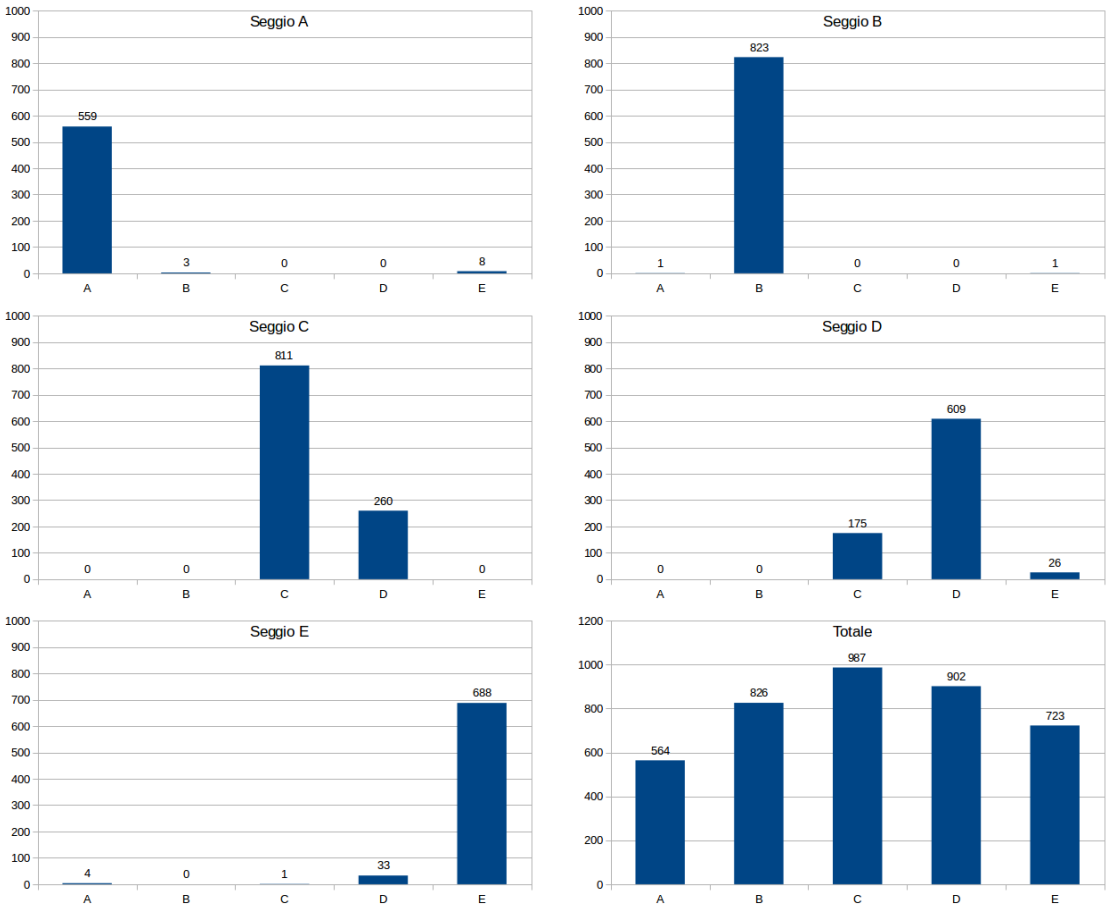


Figura 5.2: Distribuzione dei voti per la prima preferenza nei vari seggi

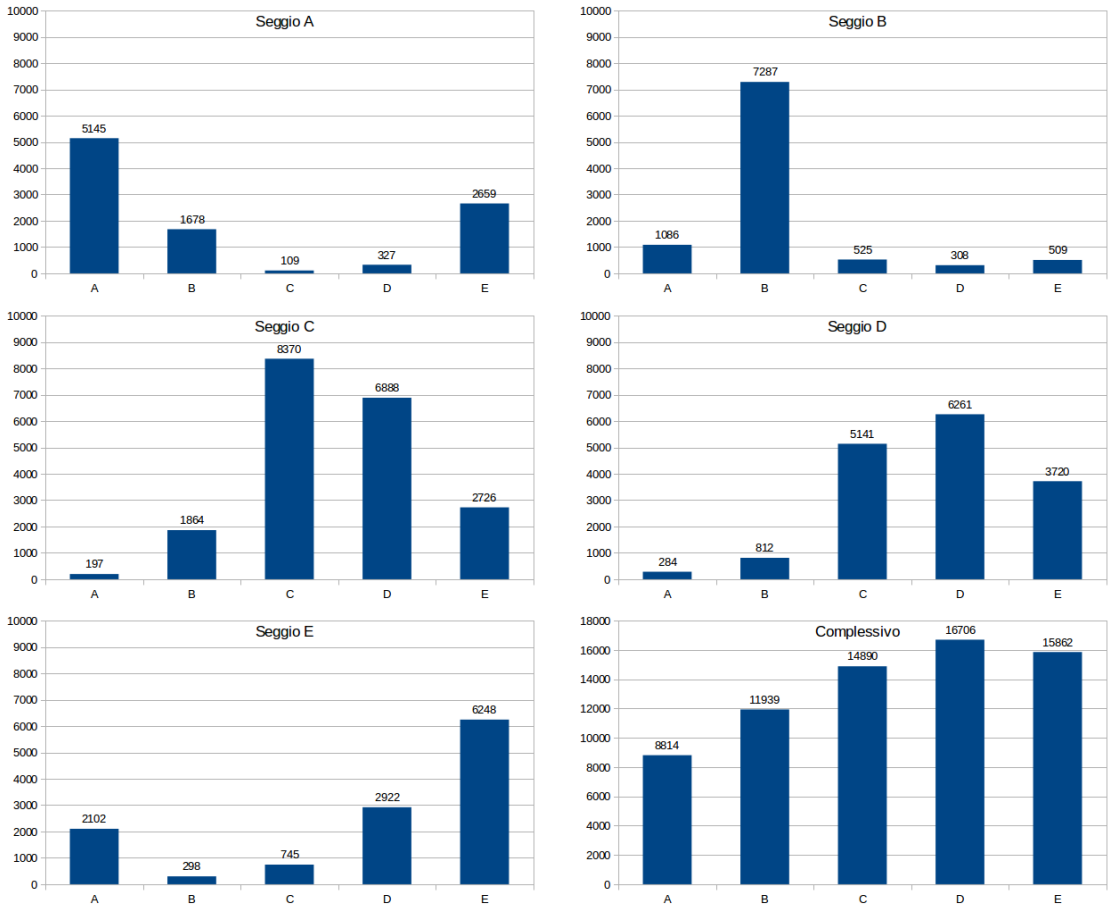


Figura 5.3: Distribuzione dei punti di preferenza nei vari seggi

i voti sono distribuiti in base alla distanza dal possibile luogo di costruzione, infatti nelle città più lontane quasi la totalità dei voti ricade sul candidato che ospita il seggio, per città più vicine (C-D) la distribuzione tra i due candidati è più equa.

Può essere interessante riportare alcuni dati sullo spazioe delle soluzioni: in tabella 5.3 tutte le soluzioni senza pareggi con il calcolo dei relativi μ e σ, in grassetto sono evidenziate le soluzioni che rientrano nel fronte di pareto che è visibile in figura 5.4

Soluzione	σ	μ	Soluzione	σ	μ	Soluzione	σ	μ	Soluzione	σ	μ
A>B>C>D>E	16.98	11.67	A>C>D>B>E	8.15	10.88	D>C>B>A>E	18.68	9.58	B>D>C>A>E	14.40	10.36
A>B>C>E>D	20.70	11.79	A>C>D>E>B	9.60	10.23	D>C>B>E>A	23.32	8.81	B>D>C>E>A	14.89	9.59
A>B>E>C>D	25.26	11.85	A>C>E>D>B	10.39	10.36	**D>C>E>B>A**	25.26	8.15	B>D>E>C>A	10.39	9.64
A>E>B>C>D	23.32	11.19	A>E>C>D>B	14.89	10.41	**D>E>C>B>A**	20.70	8.21	B>E>D>C>A	9.60	9.77
E>A>B>C>D	18.68	10.42	E>A>C>D>B	14.40	9.64	**E>D>C>B>A**	16.98	8.33	**E>B>D>C>A**	8.15	9.12
E>A>B>D>C	20.73	10.18	E>A>C>B>D	14.23	10.17	E>C>D>B>A	16.24	8.58	**E>D>B>C>A**	11.40	8.59
A>E>B>D>C	24.84	10.95	A>E>C>B>D	17.60	10.93	C>E>D>B>A	20.40	8.52	**D>E>B>C>A**	14.06	8.46
A>B>E>D>C	26.08	11.60	A>C>E>B>D	12.34	10.88	C>D>E>B>A	26.08	8.40	D>B>E>C>A	12.34	9.12
A>B>D>E>C	20.40	11.48	A>C>B>E>D	14.06	11.54	C>D>B>E>A	24.84	9.05	D>B>C>E>A	17.60	9.07
A>B>D>C>E	16.24	11.42	A>C>B>D>E	11.40	11.41	C>D>B>A>E	20.73	9.82	D>B>C>A>E	14.23	9.83
A>D>B>C>E	10.11	10.90	C>A>B>D>E	11.45	11.21	C>B>D>A>E	18.34	10.34	D>B>A>C>E	9.97	10.03
A>D>B>E>C	14.52	10.95	C>A>B>E>D	10.27	11.34	C>B>D>E>A	20.75	9.58	D>B>A>E>C	8.46	10.09
A>D>E>B>C	14.78	10.29	C>A>E>B>D	8.65	10.68	C>B>E>D>A	14.78	9.71	D>B>E>A>C	8.65	9.32
A>E>D>B>C	20.75	10.42	C>E>A>B>D	8.46	9.91	C>E>B>D>A	14.52	9.05	**D>E>B>A>C**	10.27	8.66
E>A>D>B>C	18.34	9.66	E>C>A>B>D	9.97	9.97	**E>C>B>D>A**	10.11	9.10	E>D>B>A>C	11.45	8.79
E>D>A>B>C	16.33	9.15	E>C>A>D>B	13.19	9.44	**E>C>B>A>D**	7.03	9.61	E>B>D>A>C	11.73	9.32
D>E>A>B>C	13.09	9.03	C>E>A>D>B	13.78	9.39	C>E>B>A>D	9.73	9.55	B>E>D>A>C	13.33	9.97
D>A>E>B>C	11.77	9.79	C>A>E>D>B	10.01	10.16	C>B>E>A>D	11.77	10.21	B>D>E>A>C	10.01	9.84
D>A>B>E>C	9.73	10.45	C>A>D>E>B	13.33	10.03	C>B>A>E>D	13.09	10.97	B>D>A>E>C	13.78	10.61
D>A>B>C>E	7.03	10.39	C>A>D>B>E	11.73	10.68	C>B>A>D>E	16.33	10.85	B>D>A>C>E	13.19	10.56
D>A>C>B>E	9.67	10.14	C>D>A>B>E	16.30	10.18	B>C>A>D>E	17.96	11.11	B>A>D>C>E	16.90	11.06
D>A>C>E>B	13.65	9.48	C>D>A>E>B	18.50	9.53	B>C>A>E>D	17.31	11.23	B>A>D>E>C	19.10	11.11
D>A>E>C>B	13.40	9.53	C>D>E>A>B	23.65	8.76	B>C>E>A>D	13.40	10.47	B>A>E>D>C	23.65	11.24
D>E>A>C>B	17.31	8.77	C>E>D>A>B	19.10	8.89	B>E>C>A>D	13.65	10.52	B>E>A>D>C	18.50	10.47
E>D>A>C>B	17.96	8.89	E>C>D>A>B	16.90	8.94	E>B>C>A>D	9.67	9.86	E>B>A>D>C	16.30	9.82
E>A>D>C>B	17.30	9.40	E>D>C>A>B	18.66	8.69	**E>B>C>D>A**	7.30	9.36	E>B>A>C>D	14.94	10.06
A>E>D>C>B	17.50	10.16	D>E>C>A>B	20.46	8.57	B>E>C>D>A	10.50	10.02	B>E>A>C>D	18.19	10.72
A>D>E>C>B	12.91	10.04	D>C>E>A>B	23.56	8.52	B>C>E>D>A	12.91	9.96	B>A>E>C>D	23.56	11.48
A>D>C>E>B	10.50	9.98	D>C>A>E>B	18.19	9.28	B>C>D>E>A	17.50	9.84	B>A>C>E>D	20.46	11.43
A>D>C>B>E	7.30	10.64	D>C>A>B>E	14.94	9.94	B>C>D>A>E	17.30	10.60	B>A>C>D>E	18.66	11.31

Tabella 5.3: Elenco di soluzioni senza pareggi con relativa μ e σ

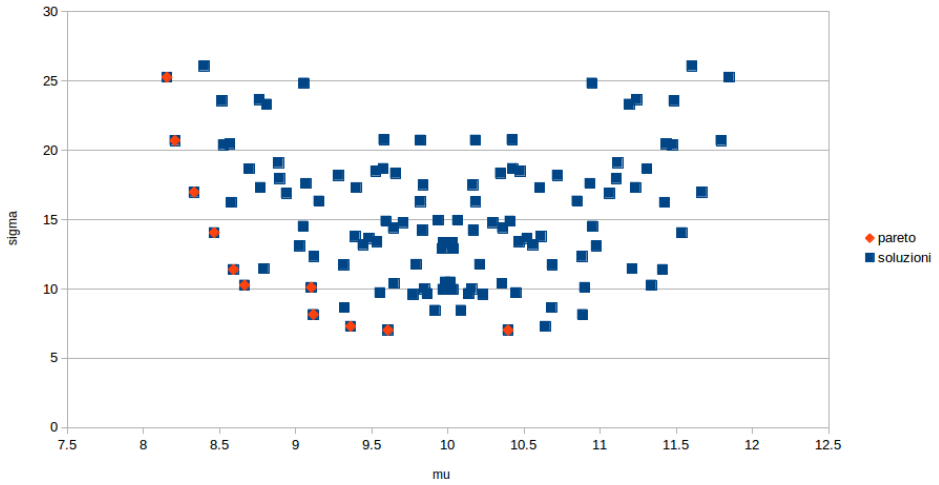


Figura 5.4: Grafico della distribuzione di soluzioni per mu e sigma

5.1 Votazione singola

ora proviamo a valutare il risultato di una votazione in assenza di influencer e con varie modalità di espressione di voto:

Algoritmo	Preferenza singola	Voto ad ordinamento	Voto pesato
Maggioritario	B>C>E>D>A	B>C>E>D>A	B>C>E>D>A
Condorcet	B>C>E>D>A	D>C>E>B>A	D>C>E>B>A
Schultze	B>C>E>D>A	E>D>C>B>A	E>D>C>B>A
Tideman	B>C>E>D>A	D>C>E>B>A	D>C>E>B>A
Copeland	B>C>E>D>A	D>C>E>B>A	D>C>E>B>A
Contucci[1]	B>C>E>D>A	D>C>E>B>A	D>C>E>B>A
Contucci[2]	D>C>B>E>A	E>C>B>A>D	E>C>B>A>D
Contucci[3]	D>C>B>E>A	E>B>C>D>A	E>B>C>D>A
Borda	B>C>E>D>A	D>E>C>B>A	D>E>C>B>A

[1] Minimize μ, escludi pareggi.

[2] Minimize σ, escludi pareggi.

[3] Equalize, escludi pareggi.

Tabella 5.4: Risultati con le varie modalità di espressione di voto

Partendo dai risultati in tabella 5.4 possiamo fare alcune considerazioni:

Intanto vediamo immediatamente come la quantità di informazione passata dai votanti all'algoritmo influenzi anche notevolmente il risultato. Daii risultati ottenuti dai vari algoritmi possiamo vedere come nel caso di preferenza singola il vincitore è quasi sempre B ma, se aumentiamo la quantità di informazione permettendo ai votanti di esprimersi con un voto ad ordinamento, B non solo non è più il vincitore, ma diventa addirittura la penultima scelta (Ovviamente l'aumento della quantità di informazione non ha alcun effetto sull'algoritmo maggioritario che semplicemente ignora le informazioni aggiuntive).

Guardando alla tabella 5.1, vediamo che B è il seggio con il maggior numero di votanti, i vari algoritmi tendono, come prevedibile, a far vincere B nelle votazioni a preferenza singola [1].

Passando alla votazione ad ordinamento, le soluzioni E e D tendono a essere le preferite dai vari algoritmi, queste sono anche le soluzioni che minimizzano la distanza complessiva (Tabella 5.1).

[1]Ricordo che nel caso di un voto a preferenza singola venga passato ad un algoritmo ad ordinamento, questo viene trasformato ponendo la preferenza come prima scelta, e tutte le alternative come seconda scelta a parimerito

5.2 Doppio turno

Proviamo ora a introdurre il **doppio turno** di votazione. Con doppio turno intendo un sistema in cui la votazione è suddivisa in due fasi, i vincitori selezionati dalla prima fase di voto vengono sottoposti ad una nuova votazione dalla quale viene estratto il vincitore. Riprendendo l'esempio precedente, assumiamo di selezionare i primi due candidati dal risultato del primo turno e rimettiamoli in votazione.

Algoritmo	Preferenza singola		Voto ad ordinamento		Voto pesato	
	Turno 1	Turno 2	Turno 1	Turno 2	Turno 1	Turno 2
Maggioritario	B>C	C>B	B>C	C>B	B>C	C>B
Condorcet	B>C	C>B	D>C	D>C	D>C	D>C
Schultze	B>C	C>B	E>D	D>E	E>D	D>E
Tideman	B>C	C>B	D>C	D>C	D>C	D>C
Copeland	B>C	C>B	D>C	D>C	D>C	D>C
Contucci[1]	B>C	C>B	D>C	D>C	D>C	D>C
Contucci[2]	D>C	D>C	E>C	C>E	E>C	C>E
Contucci[3]	D>C	D>C	E>B	E>B	E>B	E>B
Borda	B>C	C>B	D>E	D>E	D>E	D>E

[1] Minimize μ, escludi pareggi.

[2] Minimize σ, escludi pareggi.

[3] Equalize, escludi pareggi.

Tabella 5.5: Risultati secondo turno di votazione

I risultati del secondo turno sono riportati in tabella 5.5. Possiamo vedere come la presenza del secondo turno di votazione possa in alcuni casi sovvertire completamente il risultato. La differenza più evidente si ha nell'utilizzo dell'algoritmo maggioritario, in questo caso al primo turno gli abitanti della cittadina C distribuiscono i voti anche sulla vicina D, questo porta alla vittoria la cittadina B che, pur avendo un minor numero di abitanti rispetto a C, ha i voti più focalizzati essendo mediamente distante dalle altre cittadine. Quando però si passa al secondo

turno e la cittadina D viene esclusa dalle votazioni, ecco che C diventa la nuova vincitrice in quanto nella maggior parte dei voti della cittadina C, quando D era al primo posto, C era naturalmente la seconda scelta.

Per quanto riguarda gli altri algoritmi non troviamo particolari sorprese tranne che con l'algoritmo di Schultze (con espressione del voto ad ordinamento) dove possiamo riprendere la considerazione precedente, al primo turno vince la cittadina E in quanto i voti della cittadina D sono distribuiti tra se stessa e la vicina C, nel momento in cui C venga esclusa dal confronto il risultato dell'algoritmo di Schultze si capovolge e D diviene la favorita

5.3 Influencer

Proviamo ora a valutare l'effetto della presenza di una interazione che porti a movimentare delle preferenze verso un candidato. Come abbiamo visto negli esempi precedenti e dalla distribuzione delle preferenze, le città C e D sono quelle che hanno un maggior interscambio di preferenze, proviamo quindi ad accentuare questo scambio, ipotizzando magari una campagna elettorale nella città D a favore della candidatura di C. Per effettuare una simulazione di questo tipo definiamo un influencer in questo modo:

```
[{
  "pollingStationProb" : {
    "3" : 4.0
  },
  "candidate" : "C"
}]
```

Listato 5.1: Influencer della simulazione

Ovviamente l'applicazione ai vari voti dell'effetto di influenza avviene in probabilità, eseguo quindi la simulazione per 100 volte. In tabella 5.6 riporto il vincitore

per ogni simulazione con i vari algoritmi e le varie espressioni di voto.

	Preferenza		Ordinamento		Pesato	
Algoritmo	Originale	Risultati	Originale	Risultati	Originale	Risultati
Maggioritario	B	C(100/100)	B	C(100/100)	B	C(100/100)
Condorcet	B	C(100/100)	D	D(63/100) C(33/100) C=D(4/100)	D	D(62/100) C(35/100) C=D(3/100)
Schultze	B	C(100/100)	E	E(100/100)	E	E(100/100)
Tideman	B	C(100/100)	D	D(63/100) C(37/100)	D	D(62/100) C(38/100)
Copeland	B	C(100/100)	D	D(63/100) C(33/100) C=D(4/100)	D	D(62/100) C(35/100) C=D(3/100)
Contucci[1]	B	C(100/100)	D	D(63/100) C(33/100) UND(4/100)	D	D(62/100) C(35/100) UND(3/100)
Contucci[2]	D	A(100/100)	E	E(100/100)	E	E(100/100)
Contucci[3]	D	A(100/100)	E	E(100/100)	E	E(100/100)
Borda	B	C(100/100)	D	D(100/100)	D	D(100/100)

[1] Minimize μ, escludi pareggi.

[2] Minimize σ, escludi pareggi.

[3] Equalize, escludi pareggi.

Tabella 5.6: Risultati applicazione influencer

L'azione di influenza è stata applicata al seggio D per migliorare la percezione nei confronti del candidato C, come si può intuire dai grafici 5.2 e 5.3 generalmente nel seggio D, C era la seconda scelta, l'azione dell'influencer ha fatto si che molti voti promuovessero C alla prima posizione. Gli effetti maggiori di questa variazioni si hanno nel caso di votazione a preferenza singola dove questa variazione di voti porta C alla vittoria nella totalità dei casi simulati (fatta eccezione per le varianti di Contucci 2 e 3). Per quanto riguarda le votazioni a ordinamento o pesate, notiamo una maggior indeterminazione nella scelta del vincitore derivante da quali dei voti sono stati influenzati.

Da notare che gli algoritmi di Borda, Schultze e le varianti di Contucci 2 e 3 non sembrano essere minimamente influenzati dall'azione dell'influencer. L'algoritmo di Schultze si basa sul numero complessivo di vittorie, poiché il risultato della votazione, senza l'applicazione di nessun tipo di influenza, vedeva E come vincitore, il fatto di aver invertito in alcune votazioni C e D nelle prime posizioni,

non comporta un aumento di vittorie per il candidato C tale da impattare l'algoritmo. Discorso simile vale per l'algoritmo di Borda, come abbiamo detto C era già la seconda scelta della maggior parte dei voti nel seggio D, aver aumentato ulteriormente i voti a favore di C non ha avuto un impatto visibile sull'algoritmo.

Proviamo a ripetere la simulazione appena effettuata per diversi valori dell'std dell'influencer, per semplicità eseguiamo la simulazione solo per il tipo di voto pesato, in tabella 5.8 i risultati ottenuti dove il valore di ogni cella indica quante volte nei test eseguiti (100 per ogni valore di std) il vincitore è cambiato rispetto alla situazione senza influencer.

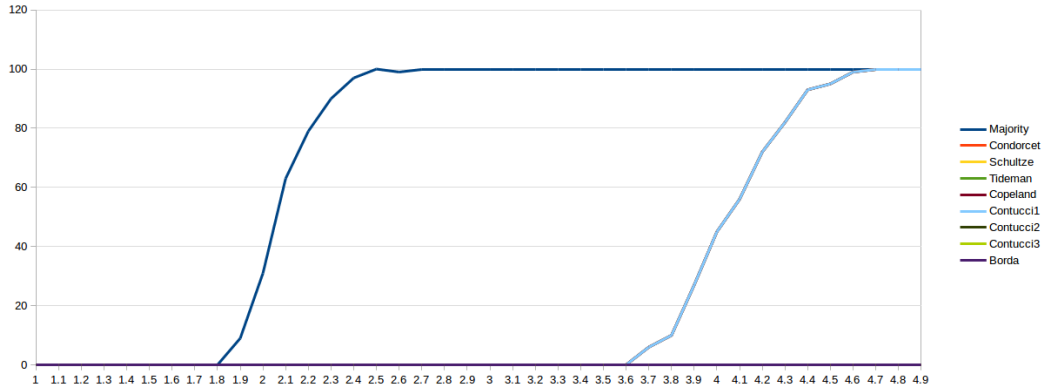


Figura 5.5: Cambiamenti del vincitore su 100 esecuzioni con std diverse

Possiamo vedere come basti un valore di std piuttosto basso per modificare radicalmente il risultato con un algoritmo maggioritario, gli altri algoritmi presentano due comportamenti distinti: alcuni hanno tutti lo stesso andamento all'aumentare std, mentre per altri (Schultze, Contucci2/3 e Borda) l'aumentare di std non provoca nessuna variazione nei risultati. Il fatto che alcuni algoritmi non siano influenzati dal valore di std deriva dal fatto che abbiamo applicato l'azione dell'influencer solo ad un seggio (D) che, per questi algoritmi, non è sufficiente a provocare variazioni significative nei risultati.

	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9
Maggioritario	0	0	0	0	0	0	0	0	0	9	31	63	79	90	97	100	99	100	100	100
Condorcet	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Schultze	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Tideman	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Copeland	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Contucci1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Contucci2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Contucci3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Borda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	3.0	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	4.0	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9
Maggioritario	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Condorcet	0	0	0	0	0	0	0	6	10	27	45	56	72	82	93	95	99	100	100	100
Schultze	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Tideman	0	0	0	0	0	0	0	6	10	27	45	56	72	82	93	95	99	100	100	100
Copeland	0	0	0	0	0	0	0	6	10	27	45	56	72	82	93	95	99	100	100	100
Contucci1	0	0	0	0	0	0	0	6	10	27	45	56	72	82	93	95	99	100	100	100
Contucci2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Contucci3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Borda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] Minimize μ, escludi pareggi.

[2] Minimize σ, escludi pareggi.

[3] Equalize, escludi pareggi.

Tabella 5.7: Cambiamenti del vincitore su 100 esecuzioni con std diverse

Proviamo ora a valutare come si comportano gli algoritmo applicando un influenza a favore del candidato A che per quasi tutti risulta, in assenza di influencer, l'ultima scelta.

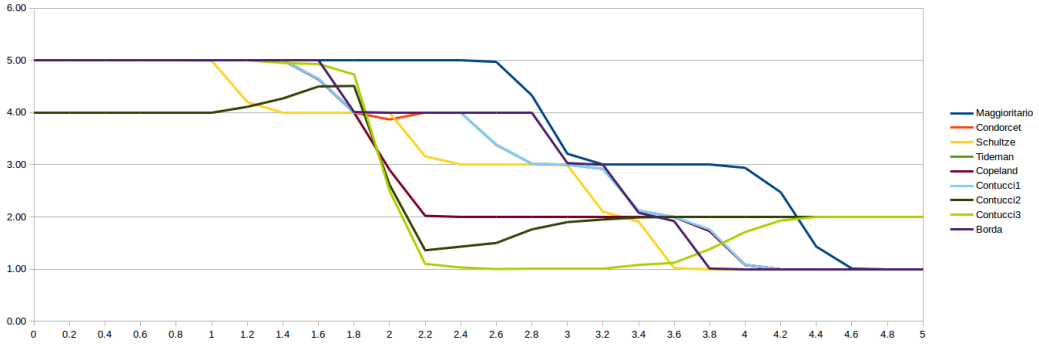


Figura 5.6: Cambiamenti nella posizione del candidato A medi su 100 esecuzioni con std diverse

Possiamo vedere come per tutti il cambiamento nella posizione di A sia graduale. Risula che tra tutti gli algoritmi, quello maggiroitario, è quello con il cambiamento più rapido all'approssimarsi di un certo valore di std, questo è dovuto al fatto che l'algoritmo tiene conto solo della prima preferenza di ogni voto e quindi per avere vraizioni apprezzabili è necessarrio che l'azione di influenza su un voto porti A alla prima posizione.

	0.00	0.20	0.40	0.60	0.80	1.00	1.20	1.40	1.60	1.80	2.00	2.20	2.40
Maggioritario	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
Condorcet	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	4.64	4.00	3.87	4.00	-
Schultze	5.00	5.00	5.00	5.00	5.00	4.99	4.20	4.00	4.00	4.00	4.00	3.16	3.00
Tideman	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	4.64	4.00	4.00	4.00	3.99
Copeland	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	4.64	4.00	2.90	2.02	2.00
Contucci1	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	4.65	4.00	4.00	4.00	3.99
Contucci2	4.00	4.00	4.00	4.00	4.00	4.00	4.11	4.27	4.50	4.51	2.61	1.36	1.43
Contucci3	5.00	5.00	5.00	5.00	5.00	5.00	5.00	4.95	4.93	4.73	2.50	1.10	1.03
Borda	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	4.01	4.00	4.00	4.00

	2.60	2.80	3.00	3.20	3.40	3.60	3.80	4.00	4.20	4.40	4.60	4.80	5.00
Maggioritario	4.97	4.33	3.21	3.00	3.00	3.00	3.00	2.94	2.47	1.43	1.01	1.00	1.00
Condorcet	-	-	-	-	2.00	2.00	1.73	1.08	1.00	1.00	1.00	1.00	1.00
Schultze	3.00	3.0	2.99	2.10	1.91	1.02	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Tideman	3.38	3.02	3.0	2.92	2.12	2.00	1.73	1.08	1.00	1.00	1.00	1.00	1.00
Copeland	2.00	2.0	2.0	2.00	2.00	2.00	1.73	1.08	1.00	1.00	1.00	1.00	1.00
Contucci1	3.39	3.02	3.0	2.92	2.12	2.00	1.76	1.08	1.00	1.00	1.00	1.00	1.00
Contucci2	1.50	1.76	1.9	1.95	1.99	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Contucci3	1.00	1.01	1.01	1.01	1.08	1.12	1.38	1.71	1.93	2.00	2.00	2.00	2.00
Borda	4.00	4.0	3.03	3.00	2.08	1.92	1.01	1.00	1.00	1.00	1.00	1.00	1.00

[1] Minimize μ, escludi pareggi.

[2] Minimize σ, escludi pareggi.

[3] Equalize, escludi pareggi.

Tabella 5.8: Cambiamenti nella posizione del candidato A medi su 100 esecuzioni con std diverse

Conclusioni

Nel costo di questa trattazione ho affrontato il tema, sempre più attuale, dei modelli democratici nell'era digitale.

Ho iniziato nel capitolo 1 analizzando il concetto di modello democratico, ho cercato di affrontare il problema in termini generali senza focalizzarmi su una specifica applicazione. La prima componente individuata è la modalità di presentazione delle proposte/candidature (sez. 1.1) che risponde alle domande: chi si può candidare? e con quali modalità? quali sono i vincoli perché una proposta sia considerata valida?

La seconda componente individuata è data dalle modalità di interazione (sez. 1.2) che portano alla modifica delle opinioni generali rispetto alle candidature, quì rientrano: dibattiti, discussione e proposte di modifica.

Sono poi passato alla componente più "tecnica" di un modello democratico, la modalità di espressione di voto (sez. 1.3) che determina la quantità di informazione passata dall'elettore al sistema, per la quale ho individuato 3 modalità: preferenza singola, voto ad ordinamento e voto pesato. Abbiamo anche visto la possibilità di suddividere una votazione in più turni con selezione progressiva dei candidati vincitori.

Alla fine del capitolo ho illustrato il funzionamento dei principali algoritmi per la determinazione del vincitore portando per ognuno dei semplici esempi.

Una volta individuati i caratteri fondanti di un modello democratico, nel capitolo 2, ho analizzato i principali sistemi di decisione partecipativa presenti on-line: LiquidFeedback, OpenDCN, Airesis.

Per ognuno di questi sistemi ho affrontato un'analisi tecnica del software individuandone le principali caratteristiche applicative: struttura del software, pattern utilizzati nello sviluppo, linguaggi e framework utilizzati; Ho inoltre individuato il modello democratico che caratterizza ogni software analizzato e riportato i risultati nelle tabelle 2.1, 2.2 e 2.3.

Per poter studiare in maniera indipendente le componenti e il modo in cui ogni parte del modello contribuisce al risultato complessivo, si è resa necessaria l'implementazione di un simulatore che permettesse di analizzare e comparare diversi modelli tra loro.

Per la realizzazione del simulatore ho usato un approccio incrementale. Nell'analisi del simulatore nel capitolo 3 ho definito la struttura del software, ho scelto quali componenti del modello implementare e ho analizzato nel dettaglio ogni elemento.

Nel capitolo 4 ho illustrato dettagliatamente da quali moduli è composto il progetto in ordine di implementazione. Ho iniziato modellando il sistema di voto e implementando i vari algoritmi (realizzando per ognuno di essi degli esaustivi test di unità atti a comprovarne la correttezza); Sono poi passato alla realizzazione del sistema di raccolta dei voti, per il quale ho optato per un approccio distribuito basato sull'utilizzo di TuCSoN e JADE; ho implementato il meccanismo di interazione descritto nell'analisi modellandolo come risultato complessivo di una moltitudini di interazioni il cui effetto è quello di spostare voti verso una candidatura piuttosto che un'altra.

Al fine di agevolare l'interazione con il simulatore ho implementato un sistema di eventi che ho testato realizzando una semplice interfaccia java swing utilizzata durante lo sviluppo

Una volta completato lo sviluppo della componente core del simulatore e verificato il corretto funzionamento del sistema di comunicazione ad eventi, si è presentato il problema di come navigare i dati estrapolati dalle simulazione. Per risolvere questo problema ho deciso di realizzare un sistema di salvataggio dei dati su database (certamente più consultabile dei file di log) e ho sviluppato una comoda interfaccia web che permettesse la visualizzazione e navigazione dei dati in maniera semplice e intuitiva.

Infine, forte del simulatore realizzato, nel capitolo 5 ho studiato un caso di test. Prima di procedere con le simulazioni ho analizzato lungo varie dimensioni il set di dati utilizzato (numero di seggi/votanti, distribuzione delle preferenze per ogni seggio ecc..). Ho effettuato diverse simulazioni variando ogni volta alcuni parametri del modello e analizzando gli impatti che queste modifiche hanno avuto sul risultato finale.

Dalla mia comparazione del modello sono emersi evidenti gli effetti dovuti alla variazione dei vari parametri. Ho innanzi tutto mostrato come si comportano i vari algoritmi al variare della quantità di informazione fornita, dalla tabella 5.4 è emersa evidente la differenza che si ha passando da un sistema a preferenza singola, a sistemi più evoluti come quello ad ordinamento o pesato.

Abbiamo poi visto simulazioni con il doppio turno, anche in questo erano presenti le differenze legate alla quantità di informazione ma è anche emerso come l'eliminazione di alcuni candidati possa totalmente sovvertire l'esito della votazione (tabella 5.5).

Infine ho analizzato la componente dell'interazione dalla quale è emerso che anche un piccolo spostamento di voti verso un candidato può portare a variazioni anche molto grandi del risultato. Ho mostrato come si comportano i vari algoritmi all'aumentare dell'effetto cumulativo dell'interazione, è risultato che gli algoritmi che gestiscono una maggior quantità di informazione subiscono modifiche più graduali, mentre in algoritmi come il maggioritario, che valutano solo la prima preferenza, le variazioni sono molto più repentine in prossimità di valori critici di influenza (Figura 5.6).

Nella realizzazione di questo elaborato ho cercato di mostrare le varie sfaccettature che caratterizzano un modello democratico e gli effetti che le varie componenti hanno sul risultato complessivo in un caso realistico. Non pretendo di aver esaurito l'argomento, per ognuno dei punti che ho individuato e trattato sarebbe possibile scrivere interi libri, quello che ho cercato di fare è mostrare che un modello esiste e che non tutte le implementazione democratiche sono uguali; a modelli democratici diversi, corrispondono diversi risultati elettorali anche a parità di intenzioni di voto. Quando parliamo di un sistema democratico dobbiamo dunque chiederci: di quale modello democratico stiamo parlando?.

Bibliografia

[1] Apache. Apache lucene. https://lucene.apache.org/. (Accessed on 10/07/2017). 30

[2] Apache. Apache solr. http://lucene.apache.org/solr/. (Accessed on 10/07/2017). 30

[3] Bootstrap · the most popular html, css, and js library in the world. https://getbootstrap.com/. (Accessed on 02/10/2018). 67

[4] P. Contucci, E. Panizzi, F. Ricci-Tersenghi, and A. Sîrbu. A new dimension for democracy: egalitarianism in the rank aggregation problem, 06 2014. 15

[5] Eclipselink. http://www.eclipse.org/eclipselink/. (Accessed on 02/10/2018). 67

[6] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of kemeny elections. *Theoretical Computer Science*, 349(3):382–391, 2005. 16, 17

[7] Jade. http://jade.tilab.com/. (Accessed on 12/24/2017). 61

[8] Welcome to jgrapht - a free java graph library. http://jgrapht.org/. (Accessed on 12/16/2017). 51

[9] Java persistence api - wikipedia. https://en.wikipedia.org/wiki/Java_Persistence_API. (Accessed on 02/10/2018). 67

[10] Liquidfeedback. http://dev.liquidfeedback.org/trac/lf/. (Accessed on 10/08/2017). 21

[11] Maven – welcome to apache maven. https://maven.apache.org/. (Accessed on 12/26/2017). 45

[12] Opendcn - free e-democracy. http://www.opendcn.org/. (Accessed on 10/08/2017). 25

[13] Home | e-participation ed eventi a milano e sua area metropolitana: cittadini e amministratori assieme per una citta' partecipata. http://www.partecipami.it/. (Accessed on 10/08/2017). 26

[14] Redis. Redis. https://redis.io/. (Accessed on 10/07/2017). 30

[15] mperham/sidekiq: Simple, efficient background processing for ruby. https://github.com/mperham/sidekiq. (Accessed on 10/07/2017). 30

[16] P. Speroni di Fenizio and D. Paterson. Don't vote, evolve!, 08 2010. 13

[17] Spring boot. https://projects.spring.io/spring-boot/. (Accessed on 02/10/2018). 67

[18] Thymeleaf. http://www.thymeleaf.org/. (Accessed on 02/10/2018). 67

[19] Tucson on apice. http://apice.unibo.it/xwiki/bin/view/TuCSoN/. (Accessed on 12/24/2017). 61

[20] Tucson on apice. http://apice.unibo.it/xwiki/bin/view/TuCSoN/4JADE. (Accessed on 12/24/2017). 61

[21] Wikipedia e-democracy. https://it.wikipedia.org/wiki/E-democracy. Accessed: 2017-09-18. 19

[22] Wikipedia. Condorcet method - wikipedia. https://en.wikipedia.org/wiki/Condorcet_method. (Accessed on 09/23/2017). 7

[23] Wikipedia. Copeland's method - wikipedia. https://en.wikipedia.org/wiki/Copeland%27s_method. (Accessed on 09/23/2017). 10

[24] Wikipedia. Kemeny–young method - wikipedia. https://en.wikipedia.org/wiki/Kemeny%E2%80%93Young_method. (Accessed on 09/24/2017). 16

[25] Wikipedia. Ranked pairs - wikipedia. https://en.wikipedia.org/wiki/Ranked_pairs. (Accessed on 09/23/2017). 12

[26] Wikipedia. Schulze method - wikipedia. https://en.wikipedia.org/wiki/Schulze_method. (Accessed on 09/23/2017). 11

[27] Wikipedia. Utility - wikipedia. https://en.wikipedia.org/wiki/Utility. (Accessed on 09/30/2017). 9