



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

TESI DI LAUREA

Consumo Energetico ed Accessibilità di Dispositivi Android: Un Approccio per l'Ottimizzazione di Interfacce Grafiche

RELATORE

Prof. Fabio Palomba
Università degli Studi di Salerno

CANDIDATO

Vincenzo De Martino
Matricola: 0522500966

Tutto non è che cambiamento, non per evitare di essere ma per diventare ciò che non si è ancora.

-EPITTETO

Sommario

Lo studio condotto in questa tesi di laurea consiste nella realizzazione di interfacce grafiche ottimizzate per dispositivi Android attraverso le quali si riesce a ridurre la quantità di energia prodotta per poter visualizzare le interfacce grafiche degli smartphone. Al contempo, lo studio ha riguardato problemi legati all'accessibilità effettuando esperimenti a persone normovedenti, astigmatici e persone adulte, per verificare se gli utenti finali riuscissero a visualizzare correttamente le componenti interne delle nuove GUI ottimizzate. Lo studio è basato su l'articolo scientifico GEMMA (Multi-objective optiMization for Android apps) e da questo sono stati condotti nuovi approcci ed esperimenti per verificare e validare gli studi precedenti ed estenderli a nuove categorie di utenti.

Indice	ii
Elenco delle figure	iv
1 Introduzione	1
1.1 Motivazioni e Obiettivi	2
1.2 Risultati	4
1.3 Struttura della tesi	5
2 Consumi Energetici	7
2.1 Introduzione alla misurazione	7
2.2 Misurazione dei display	9
2.3 Stima dei consumi energetici	10
2.4 Fondamenti di Ottimizzazione di Interfacce Grafiche	13
2.4.1 Struttura di un algoritmo genetico	14
3 Re-Implementazione di una Tecnica Search-Based per l'Ottimizzazione di Interfacce Grafiche	17
3.1 Panoramica generale	17
3.2 Prima fase: ottenimento dei dati	17
3.2.1 Android Studio e UIAtuomatorView	18
3.2.2 Algoritmo BOCP e BOCC	20
3.3 Seconda fase: Architettura dell'algoritmo genetico realizzato	24

3.4 Terza fase: Ricostruzione delle nuove interfacce grafiche ottimizzate	30
4 Design dello Studio Empirico	32
4.1 OR1- Definire se il sistema realizzato riesce ad ottimizzare i consumi energetici delle nuove interfacce grafiche	32
4.2 OR2 - Definire se le interfacce prodotte riescono ad avere un adeguato contrasto	33
4.3 OR3 - Definire se le componenti delle interfacce riescono ad essere distinguibili da persone con problemi visivi	33
4.4 OR4 -Definire quale approccio di ottimizzazione multi-obiettivo è adatto per il sistema realizzato	33
4.5 Contesto d'uso	34
4.6 Tecnologie e Metodologie del contesto d'utilizzo	35
5 Risultati	40
6 Limitazioni dello Sviluppo	45
7 Conclusioni e Lavori Futuri	50

Elenco delle figure

1.1	Consumo in mA dei colori con luminosità massima su uno smartphone Pixel.	3
1.2	Risultati applicazione Learn Music Notes design originale vs ottimizzato.	6
2.1	Consumi in % di uno smartphone [1]	8
2.2	Percentuale di odori di codice che appaiono negli odori più consumati del 10%, del 20%, del 30% più alto, del 40% più alto e del 50% più alto [2]	9
2.3	Dispositivo di misurazione Power Monitor con uno smartphone di prova	10
2.4	Esempio di interfaccia grafica del dispositivo di Monsoon Inc. prodotto da TestDevLAb	11
2.5	Modelli di consumo (mA) per i colori primari dello schermo Galaxy S4 SUPER AMOLED nello spazio colore RGB standard.	12
2.6	Modelli di consumo attuali (mA) per i colori primari dello schermo Galaxy S4 SUPER AMOLED nello spazio colore RGB lineare.	13
2.7	Fasi di un algoritmo genetico	14
3.1	Pipeline del lavoro svolto	18
3.2	Esempio di utilizzo di Ui Automator View di un applicazione	19
3.3	Esempio di Torneo binario	28
3.4	Esempio di crossover ad un punto	29
3.5	Esempio di mutation bit flip	30
4.1	Esempio di domanda del questionario	38
5.1	Confronto dei tempi di esecuzione dei due algoritmi rispetto a 9 applicazioni.	40

5.2 Risultati applicazione Board Geek Game design: originale vs ottimizzato.	41
5.3 Risultati applicazione Google Keep design: originale vs ottimizzato.	41
5.4 Media in percentuale di ottimizzazione dei consumi in Watt di 9 applicazioni.	42
5.5 Mediana in percentuale di ottimizzazione dei consumi in Watt di 9 applicazioni.	43
5.6 Percentuale media di problemi di visualizzazione delle interfacce prodotte.	43
5.7 Percentuale di fascino delle interfacce	44
6.1 Limitazioni colori interfacce per le applicazioni Diabete Plus, Board Game Geek e The Weather Channel.	46
6.2 Variazione delle colorazioni RGB delle GUI.	48

CAPITOLO 1

Introduzione

Con la nascita degli smartphone e la relativa commercializzazione il numero di persone che possiedono uno smartphone è cresciuto in maniera esponenziale, uno studio condotto da Strategy Analytics¹ afferma che nel 2021 circa metà dell'intera popolazione mondiale ha uno smartphone . Ci sono stati enormi progressi nel campo ingegneristico per la realizzazione di smartphone con potenze di calcolo sempre maggiori. I processori per smartphone che dominano per potenza nel 2022 sono rispettivamente lo Snapdragon 8 gen 1 di Qualcom e il A15 Bionic di Apple con processi produttivi rispettivamente di 5nm e 4nm². Questa potenza di calcolo sempre maggiore però fa sì che si utilizzi una maggiore quantità di elettricità provocando una maggiore riduzione della quantità della batteria. Anche se le componenti che vengono utilizzate negli smartphone come i sensori, processori ecc... sono migliorate negli anni, ovvero diventando più precise e con un consumo minore di elettricità, la mancanza di progressi nella tecnologia delle batterie non è migliorata allo stesso modo, facendo sì che non si equilibrasse il consumo degli smartphone con le capacità delle batterie. Al momento, gli ingegneri non possono aumentare a sufficienza la quantità di energia creata dalle reazioni chimiche e l'unico modo per creare batterie più potenti sembra essere quello di ingrandirle di dimensione. Tuttavia, ciò non si sposa bene con l'evoluzione dei dispositivi mobili che

¹StrategyAnalytics:<https://news.strategyanalytics.com/press-releases/press-release-details/2021/Strategy-Analytics-Half-the-World-Owes-a-Smartphone/default.aspx>

²AnTuTu: <https://www.antutu.com/en/index.htm>

tendono ad avere meno spazio a disposizione per la batteria per poter ospitare componenti e tecnologie aggiuntive [3].

Le problematiche legate alle componenti hardware, che consumano una quantità eccessiva di elettricità, però non sono solo relative ai processori e alle schede video integrate, ma troviamo anche un grande consumo energetico dovuto ai sensori dei dispositivi mobile [3; 4], problemi legati agli hotspot [5] e problemi legati agli schermi dei display, che diventano sempre più realistici ma anche energivori [6]. Sempre più produttori di smartphone hanno iniziato a installare sui propri dispositivi schermi OLED (Organic Light Emitting Diode) per i quali il consumo di energia dipende dalle combinazioni di colori a livello di sub-pixel, fino ad arrivare ad installare gli attuali pannelli AMOLED (AM-OLED, Active Matrix OLED) con una qualità maggiore del display rendendo i colori e la risoluzione quasi realistiche ma con consumi energetici maggiori³, inizialmente introdotti solo sui prodotti top di gamma ma, successivamente, anche su altre categorie di prodotti. Ci sono diversi studi online per ridurre i consumi energetici dei dispositivi mobile [1], riguardanti sia l'hardware, come anche i lavori citati in precedenza, che il software [3], con problematiche riguardanti strutture dati inefficienti, trasmissione dei dati senza compressione ecc... Negli anni infatti sono nate sempre più applicazioni e software che permettono di diagnosticare gli eccessivi consumi della batteria che utilizzano delle tecniche per ridurre l'utilizzo di energia degli smartphone, oppure consigliano all'utente come preservare la durata della batteria [7]. Il playstore e l'app store oramai sono sommersi di applicazioni simili⁴⁵. Ritornando invece ai consumi legati ai display e ai colori, già Google nel 2018 ha dimostrato quanto incidono sul consumo di energia degli smartphone i colori visualizzati sui display OLED⁶; dalla Figura 1.1. possiamo infatti desumere che i colori più chiari, come il bianco, consumino una quantità di mA maggiore rispetto ai colori più scuri, come il nero.

1.1 Motivazioni e Obiettivi

Questi studi sui consumi energetici, i colori che verranno utilizzati saranno utili per studiare e ricreare le interfacce dei dispositivi Android. Questo studio riguarda lo sviluppo di un software per l'ottimizzazione dei consumi energetici degli smartphone Android, incentrato sulla realizzazione delle GUIs (Graphical User Interface) delle applicazioni mobile, che

³AMOLED: <https://it.wikipedia.org/w/index.php?title=AMOLED&oldid=121731071>

⁴AccuBattery: <https://accubatteryapp.com/>

⁵BatteryMonitor: <https://play.google.com/store/apps/details?id=com.glgjing.hulk>

⁶Android Dev Summit: <https://www.hdblog.it/2018/11/09/android-tema-scuro-risparmio-oled-google/>

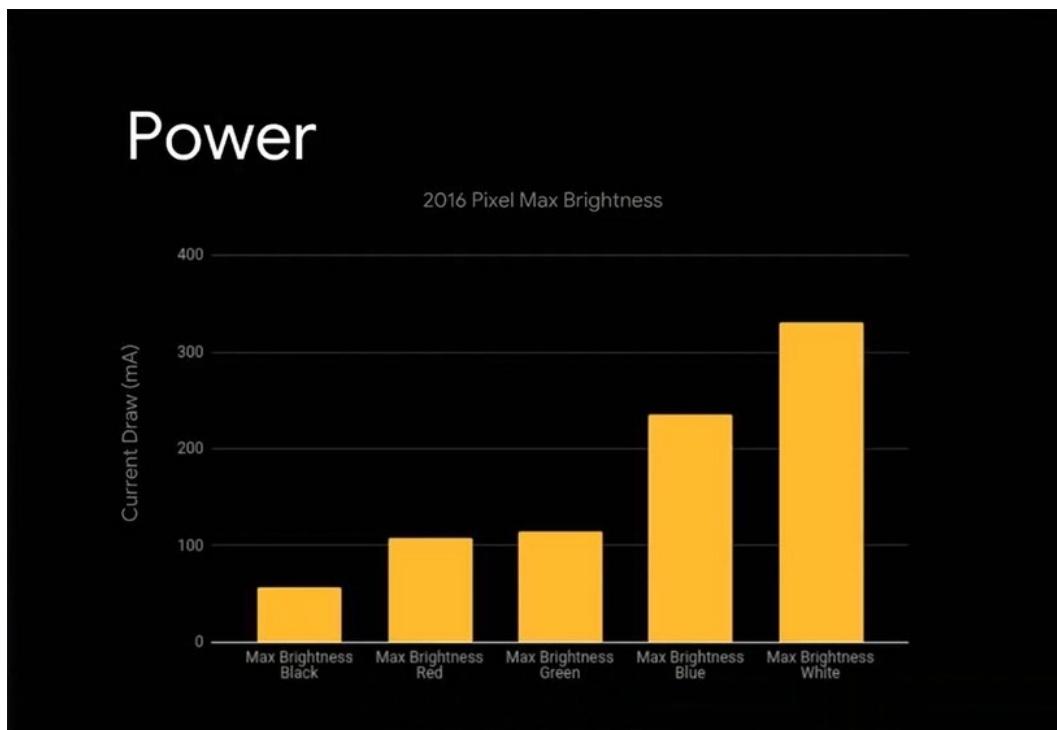


Figura 1.1: Consumo in mA dei colori con luminosità massima su uno smartphone Pixel.

permetterà di generare delle versioni delle stesse interfacce ma che consumano una quantità di energia inferiore. Gli obiettivi di questo studio saranno quindi (i) generare interfacce con un basso consumo energetico (ii) preservare il contrasto o addirittura creare un contrasto maggiore tra le componenti e (iii) verificare come le interfacce realizzate possano impattare la capacità visiva da parte di persone con problemi visivi. Questo studio riguarderà sia persone con alcuni problemi visivi oppure persone con una età più avanzata, poiché vengono meno considerate nei casi di studio. Gli sviluppatori con l’obiettivo di ridurre i consumi energetici tendono a realizzare interfacce troppo scure e senza contrasto, portando a problemi nella visualizzazione e non permettendo di distinguere correttamente le componenti con cui queste persone devono interagire nella vita di tutti i giorni. Sono diversi gli studi come nel documento di K. Ishihara et al. [8], nel quale gli autori hanno indagato il declino correlato all’età nella percezione del colore e le difficoltà nelle attività quotidiane causate da un ingiallimento della vista. L’esperimento di discriminazione del colore ha supportato l’ipotesi che alcune persone anziane non riescono a discriminare tra alcune combinazioni di colori, come giallo/bianco, blu/verde, blu scuro/nero e viola/rosso scuro. Nelle interviste che hanno presentato, gli anziani intervistati hanno menzionato difficoltà nella loro vita quotidiana derivanti dal loro errore di valutazione dei colori che potrebbe derivare dall’ingiallimento della vista legato all’età. Lo studio invece di C. Kaufman-Scarborough [9] parte direttamente

dalle richieste delle persone con problemi legati alla vista, che si lamentano dei problemi legati alle colorazioni delle componenti, come ad esempio la mancanza di un adeguato contrasto tra il testo e il background. Questo studio citato è legato principalmente al settore della pubblicità, affermando che ci sono diverse lacune per quanto riguarda il considerare le tematiche legate all'accessibilità e che ciò può far diventare i clienti insoddisfatti e provocare delle opportunità mancate legate all'acquisto dei prodotti che si sta pubblicizzando.

Il lavoro svolto per la tesi è iniziato con lo studio del paper e, successivamente, la replicazione delle tecnologie utilizzate, il principale approccio su cui è basato il documento è GEMMA (Multi-objective optiMization for Android apps) [10] ovvero un sistema per generare composizioni di colori che riducono il consumo energetico delle GUIs nelle app Android e che sono visivamente attraenti allo stesso tempo.

Le fasi di sviluppo sono diverse rispetto allo studio originale, ciò è dovuto alle diverse problematiche riscontrate durante la replicazione del documento, sia per la poca chiarezza del documento che ha richiesto tempo per poter riprodurre ciò che viene descritto, in diverse parti sono state necessarie delle nuove soluzioni per poter continuare lo sviluppo del sistema. Il progetto parte dalla scelta delle applicazioni e la relativa acquisizione delle schermate da voler analizzare, scelte le interfacce dalle applicazioni verranno estratti i colori dei pixel e le componenti e, una volta terminato questo processo di acquisizione, i dati verranno dati in pasto ad un algoritmo genetico multi-obiettivo che genererà una composizione di colori che verranno usati per poter ricostruire le nuove interfacce ottimizzate rispettando i tre obiettivi iniziali.

1.2 Risultati

Un esempio di esecuzione di questo approccio è la Figura 1.2 dove viene confrontata l'applicazione originale Learn Music Notes e i risultati prodotti dal sistema realizzato. Le interfacce prodotte da questo studio riescono ad ottimizzare i consumi energetici con una media di oltre il 70% di riduzione in termini di Watt. Invece per quanto riguarda l'accessibilità e la corretta visualizzazione delle interfacce le persone normovedenti riescono correttamente a visualizzare gli elementi delle interfacce, tranne per alcune applicazioni dove il testo non si riesce a leggere correttamente. Per le persone astigmatiche i problemi visivi legati al testo aumentano, presentando una percentuale maggiore di problemi legati al testo che non avendo un adeguato contrasto non permette la corretta visualizzazione. In generale però le

GUI ottimizzate hanno avuto un buon feedback dai partecipanti dal punto di vista estetico, soprattutto le interfacce che riescono ad aumentare il risparmio energetico.

1.3 Struttura della tesi

Questa tesi è strutturata nel seguente modo: Iniziando con una sezione riguardante la misurazione dell'energy consumption Capitolo 2, nella quale andremo a vedere quali tipologie di misurazioni esistono in letteratura, entrando poi nel dettaglio su quali sono correlate su questo studio di ricerca e delle tecniche utilizzate per poter risolvere questi compiti attraverso l'uso dell'Intelligenza Artificiale. Una sezione incentrata sull'implementazione per l'ottimizzazione delle nuove interfacce grafiche, Capitolo 3, che permetterà di conoscere come poter sviluppare la pipeline del progetto realizzato, con relativi approfondimenti per ogni sottoparagrafo. Avremo poi una sezione riguardante il design dello studio empirico, Capitolo 4, che permetterà di scoprire gli obiettivi dello studio e dei principali iper-parametri utilizzati per la replicazione di questo studio. Una sezione riguardante i risultati dello studio 5 e una sezione con le principali limitazioni e problematiche legate allo sviluppo del sistema, Capitolo 6, e infine le conclusioni e i lavori futuri nel Capitolo 7

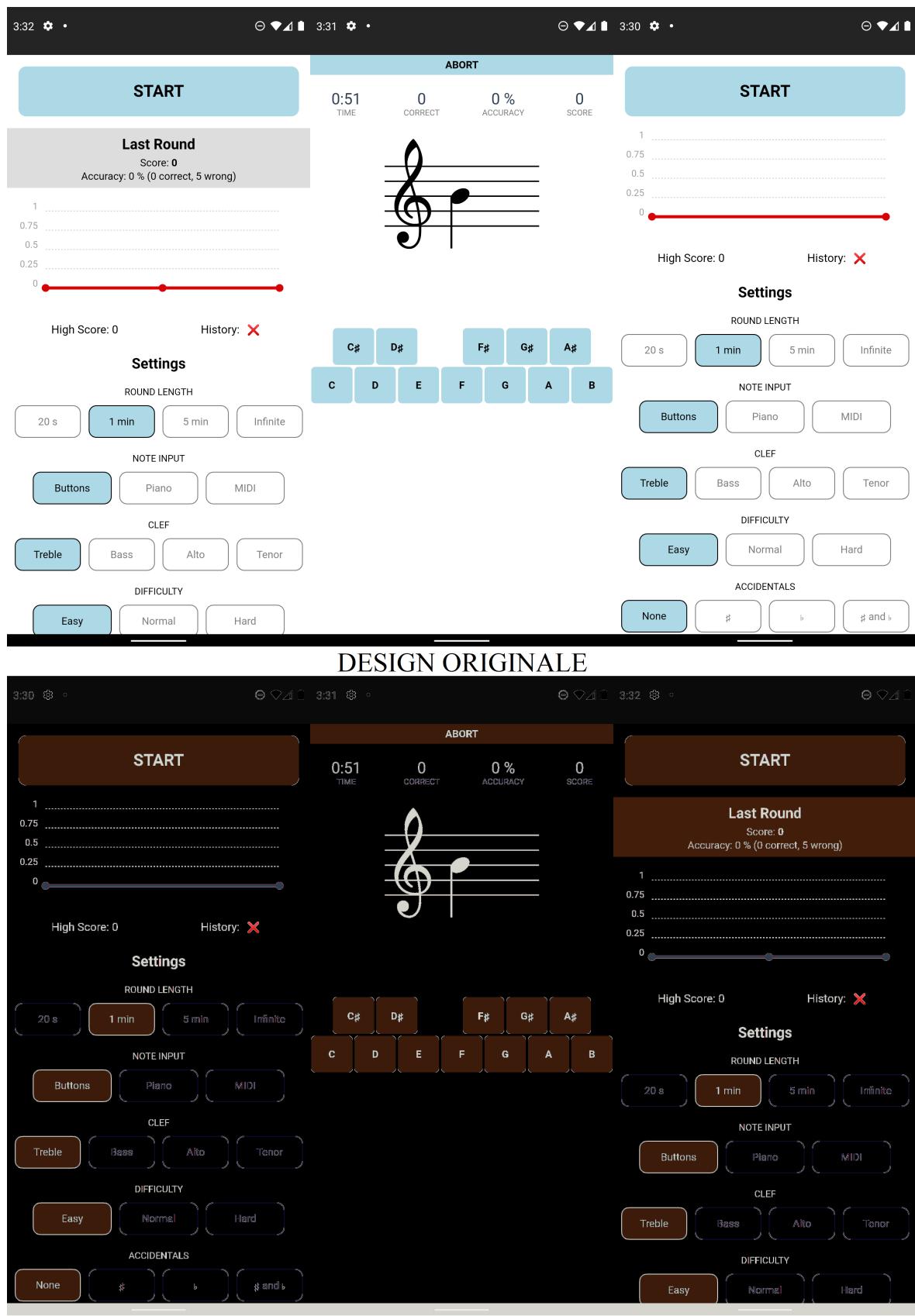


Figura 1.2: Risultati applicazione Learn Music Notes design originale vs ottimizzato.

CAPITOLO 2

Consumi Energetici

2.1 Introduzione alla misurazione

Gli studi sull'energy consumption hanno come obiettivo principale quello di monitorare, in un determinato contesto applicativo, la valutazione e la stima del consumo energetico di un determinato oggetto, nel nostro contesto degli smartphone includendo, eventualmente, l'analisi e la modellazione del consumo energetico. Le componenti analizzate sono, come citato già nell'introduzione, sensori, processori, display e quelle che possono essere le problematiche legate ad uno sviluppo errato nella progettazione del software, che porteranno ad usare componenti hardware inutilmente, consumando più velocemente la batteria. Ci sono diverse metodologie per misurare il consumo e sono legate sia all'ambito software che hardware e, di solito, la rilevazione viene effettuata da componenti hardware esterne come nei documenti di [2; 4; 10]. Terminata la misurazione e ottenuti i dati o i metadati dai sistemi di rilevamento si passerà agli studi legati all'ambito software, per poter analizzare nel dettaglio i consumi e fare le opportune ricerche. Alcune soluzioni come lo studio di Pramanik et al. [1] creano dei sistemi che permettono di simulare realisticamente il comportamento di runtime di un'applicazione, così da analizzare automaticamente l'utilizzo dei dati sensoriali di un'applicazione in diversi stati e segnalare informazioni utilizzabili per aiutare gli sviluppatori a individuare i problemi legati all'inefficienza energetica per indentificare le cause principali. Un esempio può essere la Figura 2.1, nella quale si evidenzia, dagli studi effettuati, quali sono le principali cause del consumo energetico di uno smartphone con le relative percentuali.

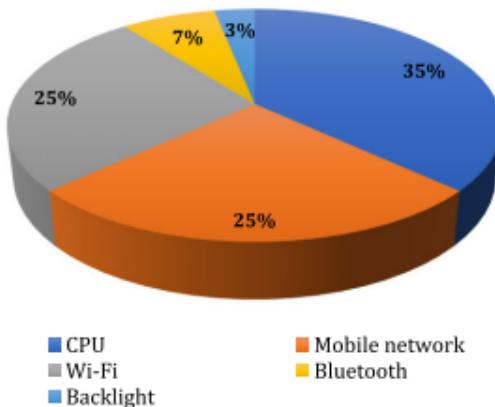


Figura 2.1: Consumi in % di uno smartphone [1]

Tuttavia, tutte queste soluzioni presentano vantaggi e svantaggi, come nel caso di Di Nucci et al. [2], nel quale gli autori hanno sottolineato che gli strumenti basati su hardware, per il loro studio, sono altamente precisi ma, allo stesso momento, queste componenti sono costose; quindi, questo tipo di rilevazione può risultare dispendiosa. Gli strumenti basati su modelli richiedono la calibrazione dei parametri necessari per creare correttamente un modello su un dispositivo hardware specifico. Gli approcci basati sul software, invece, sono più economici e più facili da usare rispetto agli strumenti basati sull'hardware, d'altro canto però i dati saranno meno precisi. Lo studio invece di Palomba et al. [11] riguarda la progettazione di software ecologici, questa tipologia di progettazione è in costante aumento dovuto al fatto che la capacità di calcolo dei dispositivi mobile è limitata dalla durata della batteria, creare smartphone con potenze di calcolo molto elevate ma che durano un paio di ore non riuscirà a coinvolgere un pubblico di scala globale. Gli autori si sono concentrati sui difetti di progettazione che, teoricamente, dovrebbero essere correlati ad attributi non funzionali del codice sorgente, come le prestazioni e il consumo di energia. I risultati del loro studio hanno evidenziato che i metodi interessati da quattro tipi di smell¹, ovvero Internal Setter, Leaking Thread, Member Ignoring Method e Slow Loop, consumano fino a 87 volte di più rispetto ai metodi interessati da altri odori di codice. Nella Figura 2.2 possiamo vedere diverse tipologie di odori e che alcuni tipi di odore del codice tendono a manifestarsi e co-presentarsi più frequentemente, indicando che potenzialmente hanno un effetto sul consumo di energia.

¹smell code, ovvero sintomi di scelte progettuali o implementative inadeguate, sul consumo energetico delle applicazioni mobili

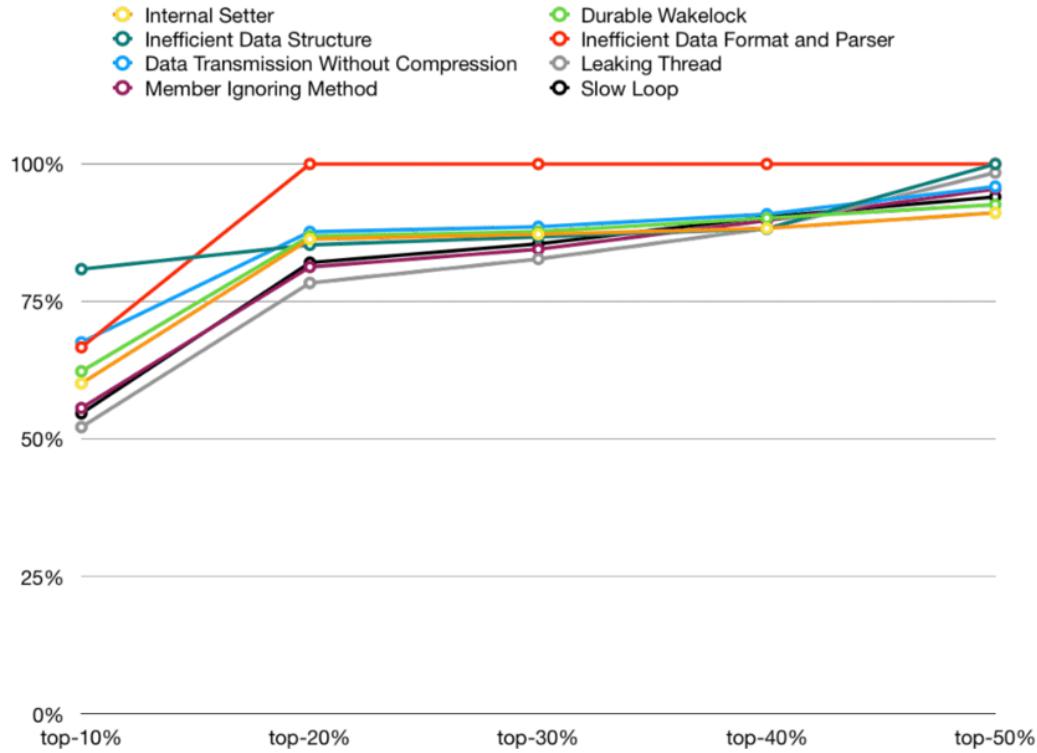


Figura 2.2: Percentuale di odori di codice che appaiono negli odori più consumati del 10%, del 20%, del 30% più alto, del 40% più alto e del 50% più alto [2]

2.2 Misurazione dei display

Passando alle misurazioni dei display mobile, l’obiettivo è quello di riuscire a capire il consumo dei colori dei pixel dello schermo e, una volta realizzato l’obiettivo, fare delle valutazioni per poter utilizzare le informazioni ottenute e generare dei modelli che consumino una quantità di energia inferiore. Diversi studi come [5; 10; 12; 13; 14] utilizzano nel loro studio il formato rgb per lavorare con i colori. In base alla specifica matrice di pixel dello schermo, ogni colore in un pixel viene visualizzato utilizzando una combinazione di sub-pixel rosso, verde e blu nel modello di colore RGB standard (ovvero sRGB). Nel caso degli schermi OLED, il consumo energetico dei sub-pixel dipende dal livello della componente colore. Il consumo di energia di un pixel OLED è una funzione lineare dei suoi valori RGB lineari, ovvero la decodifica gamma risulta dai suoi valori nello spazio RGB standard. Gli autori dei paper [6; 10] al posto di utilizzare misurazioni lato software, come PowerManagement² e PowerTutor³, con modelli che hanno generalmente un’adattabilità limitata alle molte variazioni delle configurazioni hardware disponibili e che incorrono in inevitabili errori di valutazione in

²<https://developer.android.com/about/versions/pie/power/>

³<http://ziyang.eecs.umich.edu/projects/powertutor/>



Figura 2.3: Dispositivo di misurazione Power Monitor con uno smartphone di prova

fase di esecuzione, hanno preferito utilizzare dispositivi hardware esterni. Invece di calcolare le informazioni sull'alimentazione dal software, ciò che è stato utilizzato è un monitor di alimentazione esterno, per misurare direttamente il comportamento della batteria in caso di interruzione del consumo di energia, con una serie di esperimenti di contrasto. Quello che hanno fatto [10; 15] è stato utilizzare un monitor di potenza per dispositivi mobili prodotto da Monsoon Inc. per la misurazione della potenza in tempo reale. Il dispositivo Figura 2.3 supporta una frequenza di campionamento fino a 5 kHz e può essere utilizzato per sostituire la batteria come alimentazione dello smartphone.

2.3 Stima dei consumi energetici

Il monitor di potenza è collegato direttamente allo smartphone e registra un istogramma di potenza Figura 2.4. Durante la valutazione dell'alimentazione dei moduli display, hanno disabilitato tutti i servizi di sistema non necessari, che potrebbero aver causato un significativo rumore di consumo energetico, comprese le comunicazioni di rete 4G e Wi-Fi, i servizi

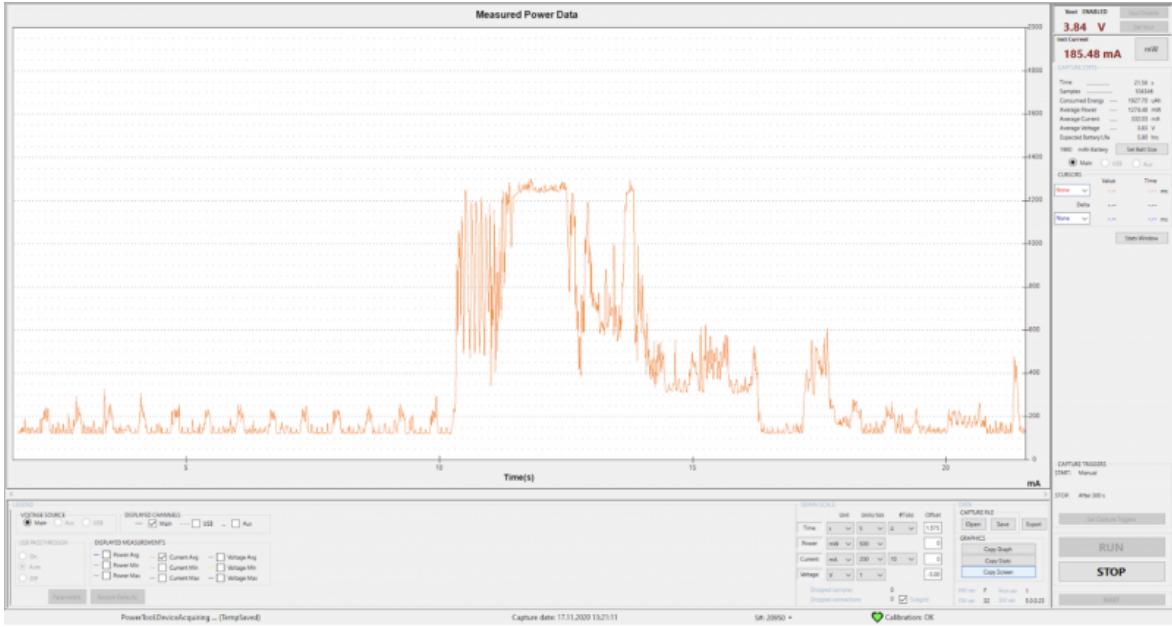


Figura 2.4: Esempio di interfaccia grafica del dispositivo di Monsoon Inc. prodotto da TestDevLab

in background e le applicazioni di ottimizzazione dell'alimentazione. Gli esperimenti di contrasto sono progettati per separare il consumo energetico del display da quello del sistema, ad esempio CPU e GPU. La maggior parte dei test di consumo energetico vengono completati mentre la luminosità dello schermo del dispositivo è impostata al massimo. Nel caso di Linares-Vásquez et al. [10], il documento che ha avuto un grande impatto per questo studio di ricerca, hanno costruito il modello di alimentazione per uno schermo SUPER AMOLED (1080×1920 pixel) in un Samsung Galaxy S4. Per definire ciascuna funzione di colore, hanno misurato la corrente assorbita dallo schermo impostando in due modi: la prima misurazione è stata fatta con tutti i pixel impostati sul colore nero, ovvero durante un periodo di inattività (periodo idle), per 30 secondi, la seconda invece viene eseguita a schermo intero, con tutti i pixel impostati sulla stessa componente di colore primario per ogni $livello \in [1, 255]$ per 30 secondi (periodo measurement). In totale quindi sono stati raccolti 765 misurazioni sia per i periodi di inattività che di misurazione, ovvero 255×3 . Per la misurazione, gli autori del paper, non hanno considerato l'effetto della luminosità del display poiché lo hanno impostato al valore massimo consentito dal display. Sia per il periodo di inattività che per quello di misurazione è stata calcolata la corrente media di un pixel dividendo il valore grezzo, misurato con il monitor di alimentazione, per il numero totale di pixel sullo schermo (1080×1920). Dopo aver raccolto tutti i valori, ciò che hanno fatto per tenere conto solo della corrente assorbita dai pixel dello schermo è sottrarre la corrente in $inattività_k$ dalla corrente nella $misurazione_k$ corrispondente. Infine, hanno rimosso il rumore utilizzando lo

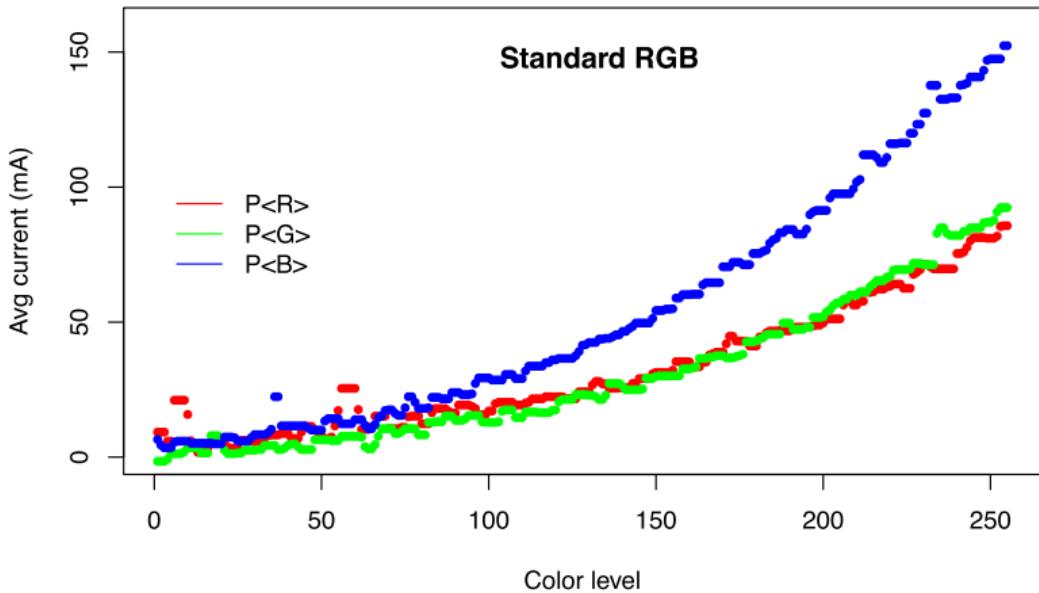


Figura 2.5: Modelli di consumo (mA) per i colori primari dello schermo Galaxy S4 SUPER AMOLED nello spazio colore RGB standard.

smoother di Tukey implementato in R [16]. Lo smoother di Tukey utilizza le mediane mobili per smussare una serie temporale, sostituendo un valore k-esimo nella serie temporale con la mediana di se stesso, del suo predecessore e del suo successore [16]; Le misurazioni dopo il livellamento sono presentate nella Figura 2.5.

Una volta ottenuti i consumi ciò che si avrà sarà la non linearità nello standard sRGB come nei documenti [6; 10; 14]. Successivamente da sRGB verrà trasformata il livello di intensità in formato RGB lineare, che semplifica la costruzione del modello senza perdita di informazioni, ottenendo una relazione lineare tra consumo energetico e intensità dei pixel quando viene visualizzata una interfaccia specifica. In Figura 2.6 vengono mostrate le misure di corrente in mA (milliampere); la corrente è proporzionale alla potenza (Watt), perché la tensione è costante durante le misurazioni. Quindi, non sarà necessario tracciare i risultati in termini di Watt. Ritornando quindi ai singoli colori dei pixel all'interno di un'immagine, dato un profilo $P_{\langle color \rangle}(livello)$ che restituisce il consumo di un colore, possiamo calcolare il consumo di un pixel nella posizione $\langle x, y \rangle$ con $color_{x,y}$ e le componenti $R_{x,y}$ (Rosso) $G_{x,y}$ (Verde) e $B_{x,y}$ (Blu) attraverso l'espressione 2.3.1.

$$P(color_{x,y}) = P_{\langle R \rangle}(R_{x,y}) + P_{\langle G \rangle}(G_{x,y}) + P_{\langle B \rangle}(B_{x,y}) \quad (2.3.1)$$

Una volta che abbiamo realizzato l'espressione per calcolare il consumo di un unico colore di un singolo pixel in posizione x,y possiamo calcolare la potenza totale consumata dallo

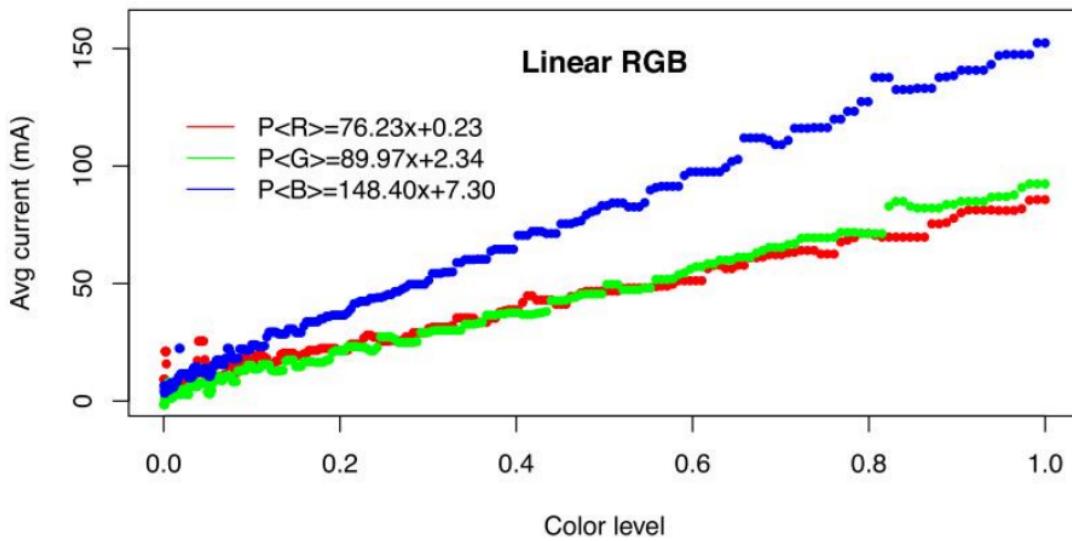


Figura 2.6: Modelli di consumo attuali (mA) per i colori primari dello schermo Galaxy S4 SUPER AMOLED nello spazio colore RGB lineare.

schermo effettuando la sommatoria di tutti i consumi dei colori dei pixel all'interno di una interfaccia con l'espressione 2.3.2.

$$TP(GUI) = \sum_{x=1}^X \sum_{y=1}^Y P(color_{x,y}) \quad (2.3.2)$$

Il profilo di potenza delle componenti di colore rappresentate dalle funzioni $P_{(color)}$ è specifico dello schermo. Nel lavoro iniziale non è stato possibile effettuare le misurazioni attraverso gli strumenti utilizzati da [6; 10; 14], sia per motivi economici che per motivi legati ai tempi per poter effettuare queste misurazioni, che avrebbero richiesto una certa quantità di tempo, ma ciò che è stato fatto è utilizzare le espressioni 2.3.1 e 2.3.2 per poter analizzare i colori delle interfacce grafiche, assegnando i valori della Figura 2.6.

2.4 Fondamenti di Ottimizzazione di Interfacce Grafiche

Un algoritmo genetico è un algoritmo euristico utilizzato per tentare di risolvere problemi di ottimizzazione per i quali non si conoscono altri algoritmi efficienti di complessità lineare o polinomiale. L'aggettivo "genetico", ispirato al principio della selezione naturale ed evoluzione biologica teorizzato nel 1859 da Charles Darwin, deriva dal fatto che, al pari del modello evolutivo darwiniano che trova spiegazioni nella branca della biologia detta genetica, gli algoritmi genetici attuano dei meccanismi concettualmente simili a quelli dei processi biochimici scoperti da questa scienza [17]. Gli algoritmi genetici consistono in al-

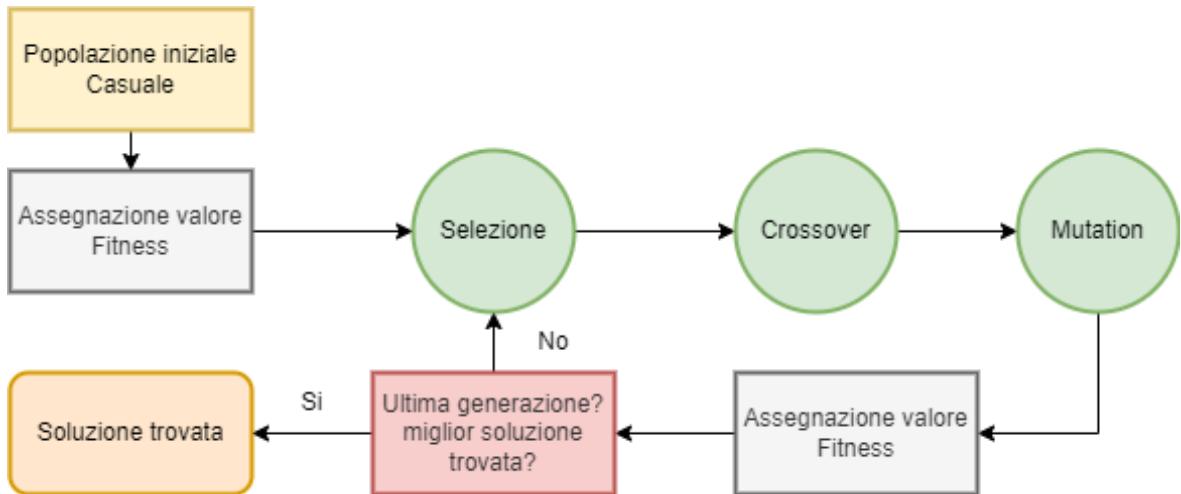


Figura 2.7: Fasi di un algoritmo genetico

goritmi che permettono di valutare diverse soluzioni di partenza (come se fossero diversi individui biologici) e che ricombinandole (analogamente alla riproduzione biologica sessuata) ed introducendo elementi di disordine (analogamente alle mutazioni genetiche casuali) producono nuove soluzioni (nuovi individui) che vengono valutate scegliendo le migliori (selezione ambientale) nel tentativo di convergere verso soluzioni "di ottimo". Ognuna di queste fasi di ricombinazione e selezione si può chiamare generazione come quelle degli esseri viventi. Nonostante questo utilizzo nell'ambito dell'ottimizzazione, data la natura intrinsecamente casuale dell'algoritmo genetico, non vi è modo di sapere a priori se sarà effettivamente in grado di trovare una soluzione accettabile al problema considerato. Se si otterrà un soddisfacente risultato, non sarà detto che si capisca perché abbia funzionato, in quanto non è stato progettato da nessuno ma da una procedura casuale⁴.

2.4.1 Struttura di un algoritmo genetico

Ci sono diverse caratteristiche in un algoritmo genetico, abbiamo (i) un cromosoma, ovvero una delle soluzioni ad un problema considerato. (ii) Un gene, ovvero una parte del cromosoma. (iii) Una popolazione, di individui, che viene creata in modo casuale e che rappresenta un insieme di soluzioni del problema considerato. Ogni individuo può essere pensato come una soluzione candidata per qualche problema di interesse. Le variazioni tra gli individui nella popolazione fanno sì che alcuni individui abbiano un valore di fitness maggiore degli altri (ad esempio, migliori soluzioni ai problemi). Queste differenze vengono utilizzate per influenzare la selezione di una nuova serie di soluzioni candidate nella fase

⁴ https://it.wikipedia.org/w/index.php?title=Algoritmo_genetico&oldid=115946309

successiva, denominata selezione. Durante la (iv) selezione, viene creata una nuova popolazione producendo copie di individui di maggior successo ed eliminando quelli di minor successo. Tuttavia, le copie non sono esatte. Esiste una probabilità di (v) mutazione (per esempio cambiando un gene della popolazione), (vi) crossover (scambio di sotto elementi corrispondenti tra due individui) o altre modifiche alla popolazione durante l'operazione di copia. Trasformando il precedente insieme di buoni individui in uno nuovo, le operazioni di mutazione e crossover generano un nuovo insieme di individui, o campioni, che idealmente hanno una probabilità superiore alla media di produrre individui buoni [18; 19; 20]. Quando questo ciclo di valutazione, selezione e operazioni genetiche viene ripetuto per diverse generazioni, l'idoneità generale della popolazione generalmente migliora e gli individui nella popolazione rappresentano "soluzioni" migliorate a qualsiasi problema sia stato posto nella funzione di idoneità, un esempio per rappresentare questo processo è la Figura 2.7.

Quando però abbiamo bisogno di riuscire ad ottenere diverse funzioni di fitness da ottimizzare entra in gioco l'ottimizzazione multi-obiettivo. E' un'area di ricerca molto importante perché molti problemi pratici di ingegneria sono problemi di ottimizzazione multi-obiettivo [10; 21; 22; 23; 24], quindi non avremo più una singola funzione da minimizzare ma saranno diverse. Il problema di ottimizzazione multi-obiettivo può essere generalmente espresso come segue 2.4.1.

$$\begin{cases} V - \min & f(x) = [f_1(x), f_2(x), \dots, f_p(x)]^T, \\ s.t & x \in X \\ & X \subseteq R^m \end{cases} \quad (2.4.1)$$

Dove V-min rappresenta la minimizzazione del vettore, che consiste nel minimizzare simultaneamente ogni sotto obiettivo nell'obiettivo del vettore $f(x) = [f_1(x), f_2(x), \dots, f_p(x)]^T$. L'ottimizzazione simultanea di una funzione obiettivo multipla, possibilmente in competizione, si discosta dall'ottimizzazione di funzione singola in quanto ammette raramente una singola soluzione perfetta, ma piuttosto un insieme di soluzioni alternative, chiamate insiemi di soluzioni non dominate o insieme di Pareto. I tre concetti seguenti sono strettamente correlati: (i) Data una situazione iniziale, un miglioramento paretiano è una nuova situazione in cui alcuni agenti guadagneranno e nessun agente perderà. (ii) Una situazione è chiamata dominata paretiana se esiste un possibile miglioramento paretiano. (iii) Una situazione è chiamata Pareto ottima o Pareto efficiente se nessun cambiamento può portare a una maggiore soddisfazione per un agente senza che un altro agente perda o, equivalentemente, se non c'è spazio per un ulteriore miglioramento paretiano. L'obiettivo principale delle tecniche di

ottimizzazione multi-obiettivo è trovare una o più soluzioni accettabili nell'insieme di Pareto e il **fronte di Pareto** è l'insieme di tutte le situazioni Pareto efficienti⁵. Queste informazioni saranno necessarie per poter costruire l'architettura di questo sistema, realizzando le funzioni di fitness che saranno, appunto, gli obiettivi di ricerca dello studio, ovvero l'ottimizzazione del consumo energetico e il contrasto tra le componenti per una corretta visualizzazione delle componenti.

⁵https://en.wikipedia.org/w/index.php?title=Pareto_efficiency&oldid=1089288372

CAPITOLO 3

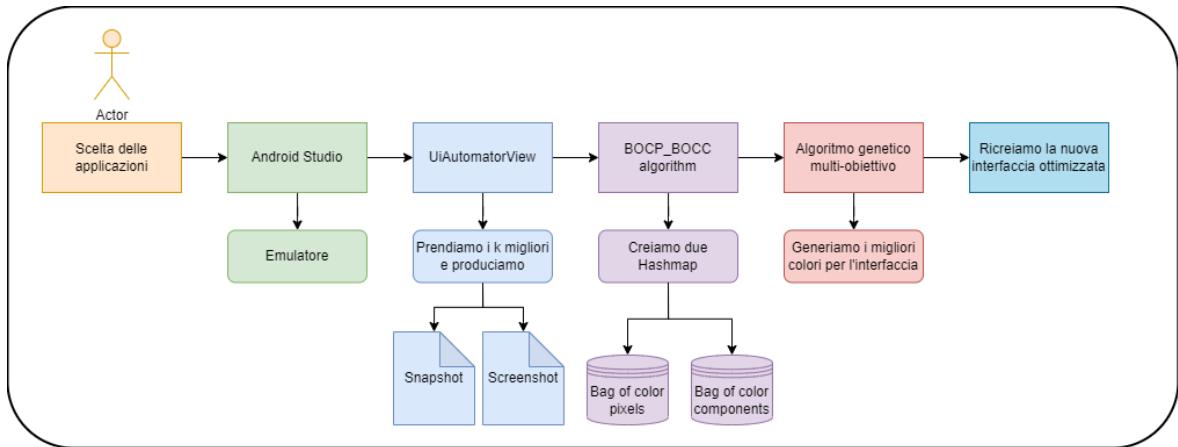
Re-Implementazione di una Tecnica Search-Based per l'Ottimizzazione di Interfacce Grafiche

3.1 Panoramica generale

In questo capitolo verrà descritta la pipeline del lavoro svolto per la realizzazione del software che consentirà di sviluppare nuove interfacce grafiche con un minore consumo energetico. Sono principalmente tre le fasi di sviluppo di questo software; Una prima fase nella quale verranno scelte le applicazioni, con le relative acquisizioni delle k migliori interfacce, da queste verranno poi estratte le componenti e i colori dei pixel delle immagini. Una seconda fase che riguarderà gli algoritmi di Intelligenza Artificiale, più nello specifico verranno utilizzati degli algoritmi genetici multi-obiettivo per produrre i colori che permettano di ridurre la quantità di energia. La terza e ultima fase riguarderà la ricostruzione delle interfacce dai risultati ottenuti dagli algoritmi implementati, Figura 3.1.

3.2 Prima fase: ottenimento dei dati

La prima cosa da fare per lo studio dell'ottimizzazione delle interfacce è individuare le applicazioni da voler ottimizzare. Il numero di applicazioni non influenzera la qualità del processo ma servirà principalmente a validare lo studio e verificare la generalizzazione del sistema.

**Figura 3.1:** Pipeline del lavoro svolto

3.2.1 Android Studio e UIAutomatorView

Scelte le applicazioni iniziali e scaricate sul proprio dispositivo Android sarà necessario utilizzare Android Studio¹ e sarà necessario utilizzare un dispositivo Android per poter utilizzare UI Automator View². Possedere o meno un dispositivo fisico Android non sarà però indispensabile, Android Studio permette di poter emulare diversi dispositivi e le dimensioni del dispositivo non incideranno sullo studio. Collegato il dispositivo ad Android Studio, che sarà quindi un emulatore o un dispositivo fisico, bisognerà scaricare le applicazioni scelte e sarà necessario utilizzare il tool UI Automator View che, tramite un'interfaccia grafica, permetterà di acquisire per ogni interfaccia i relativi screenshot e snapshot³. Le API di UI Automator consentono di interagire con gli elementi visibili su un dispositivo, indipendentemente da quale Activity è selezionata, e quindi, consente di eseguire operazioni come l'apertura del menu Impostazioni o l'avvio dell'app di un dispositivo considerato, come può essere un dispositivo fisico o un emulatore. Il test può cercare un componente dell'interfaccia utente utilizzando comodi descrittori come il testo visualizzato in quel componente o la sua descrizione del contenuto. Poiché il path di UI Automator è cambiato con i continui aggiornamenti di Android Studio vi inserisco il cammino con la versione di Android Studio che è stato utilizzato per questo esperimento, il path è: "C:\Users\AppData\Local\Android\Sdk\tools\bin", invece la versione: "Android Studio Chipmunk 2021.2.1".

¹<https://developer.android.com/studio?gclid=Cj0KCQjwr4eYBhDrARIsANPywChQd7AJC6hoLPRK17GuIVwcB&gclsrc=aw.ds>

²<https://developer.android.com/training/testing/other-components/ui-automator>

³Uno snapshot, letteralmente una "istantanea", è generalmente la cattura di stato di un oggetto in un determinato momento nel tempo

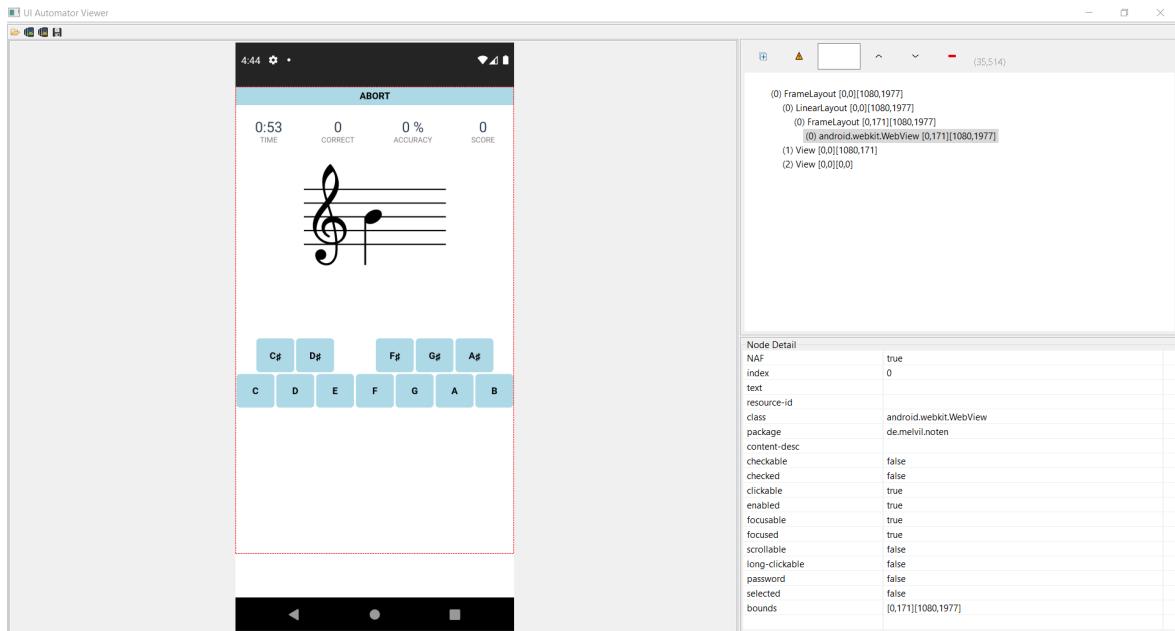


Figura 3.2: Esempio di utilizzo di Ui Automator View di un applicazione

Ritornando di nuovo alle caratteristiche di questo tool, questo software permette di acquisire le interfacce che vengono visualizzate, restituendo in output uno screenshot e uno snapshot come in Figura 3.2, lo screenshot è un semplice file .png contenente, appunto, solo l’immagine della schermata visualizzata nel momento in cui viene effettuata l’acquisizione. Lo snapshot invece è un file .xml contenente tutta la gerarchia dell’interfaccia e le relative componenti, ogni componente è caratterizzata da diciassette o diciotto caratteristiche, ciò dipende da come è stata implementata l’interfaccia dagli sviluppatori dell’app, un esempio del tool lo troviamo nella Figura 3.2, nella parte in basso a destra possiamo notare che, ad ogni caratteristica è associato un valore. L’acquisizione delle schermate però non sarà come nei documenti [13; 25] che acquisiscono un’unica schermata alla volta, generando quindi un’inconsistenza delle tavole di colori delle nuove interfacce che vengono realizzate in output. Creando quindi interfacce, di una stessa applicazione, con colori diversi che non sono coerenti tra di loro. I metodi di acquisizione, inoltre, possono essere svolti sia online che offline, ovvero online quando l’applicazione è mostrata a schermo oppure offline quando utilizziamo gli screenshot acquisiti attraverso il codice sul lato server [13]. Gli screenshot e gli snapshot delle interfacce possono essere catturate sia "a mano", ossia selezionando l’interfaccia e cliccando il pulsante che permette di ottenere screenshot/snapshot sia automatizzando il processo, tramite le API di Google è possibile eseguire lato software queste catture, esplorando l’applicazione e prendendo solo le k interfacce utilizzando un algoritmo di esplorazione in profondità per poter scegliere le interfacce che vengono visitate con un numero maggiore di volte. Per

rendere lo studio coerente il valore di k dovrebbe essere uguale per tutte le applicazioni. UI Automator View ha un grande potenziale, infatti, Patil et al. [26] hanno realizzato uno strumento chiamato "Enhanced UI Automator Viewer", simile a Ui Automator, ma più potente, per ispezionare le componenti dell'interfaccia utente delle applicazioni mobile e che utilizza una funzione per calcolare il contrasto del colore per identificare la migliore combinazione di colori per il primo piano e il background, così da rendere le applicazioni Android accessibili agli utenti con problemi di vista e daltonismo.

3.2.2 Algoritmo BOCP e BOCC

Ottenute le k interfacce, con i relativi snapshot e screenshot, andremo ad implementare l'algoritmo BOCP_BOCC, un algoritmo che permette di estrarre da un numero di interfacce k, due HashMap, uno di BOCP (ovvero bag of color pixels) e uno di BOCC (ovvero bag of color components). Il BOCP è una struttura hash-map per tutti i pixel nella GUI, in cui una chiave è un colore quantizzato (*colore**) e il valore corrispondente (per una chiave) è un elenco di pixel assegnati a quel colore e la relativa interfaccia. BOCC è una hash-map, in cui la chiave è un colore quantizzato (*colore**) e il valore corrispondente è un elenco contenente tutti gli elementi delle componenti associate ai pixel in BOCP[*colore**] e la relativa interfaccia. Rispetto all'implementazione iniziale dello studio di Linares-Vásquez et al. verrà salvato anche il valore h insieme ai pixel e alle componenti che rappresenteranno l'indice delle interfacce negli HashMap BOCP/BOCC.

L'algoritmo BOCP_BOCC è strutturato nel seguente modo: in input vengono inseriti le variabili *k* per estrarre tre diversi colori da ciascun componente della GUI, il rapporto di contrasto *r* e la variabile *GUIS* che rappresenta un insieme di screenshot e snapshot.

Per ogni interfaccia vengono create due variabili, S e C, la prima contiene l'immagine dell'interfaccia e la seconda l'elenco delle componenti dello snapshot. Per ogni componente all'interno della lista di componenti della variabile C, ciò che faremo sarà effettuare un controllo sulla tipologia di componente, se la classe della componente è un ImageView o un ImageButton allora non faremo niente. Questa scelta è stata presa per evitare la trasformazione/distorsione di immagini/loghi che appartengono alla GUI o anche a terze parti. Alterare i colori di un'immagine cambierebbe completamente la sua natura e l'aspetto. Sebbene ciò sarebbe stato del tutto possibile secondo questa tipologia di approccio, è stato ritenuto che queste modifiche non rendessero realistiche le interfacce quando vengono ricreate dall'output fornito agli sviluppatori. Le componenti del ciclo vengono scelte in maniera decrescente, l'algoritmo prende i pixel appartenenti a ciascun componente attraversando la gerarchia

Algorithm 1: Estrazione di BOCP e BOCC da un insieme di GUI in un'app nativa

Android

Data: $k, r, GUIs$

Result: $BOCP, BOCC$

```

1 begin
2    $BOCP = \emptyset, BOCC = \emptyset, pixelsImage = \emptyset;$ 
3   foreach  $GUI \in GUIs$  do
4      $S = getUISnapshot(GUI);$ 
5      $C = getComponentsFromSnapshot(S);$ 
6     foreach  $c \in C$  do
7       if  $c \neq image$  then
8          $pixels_c = getPixels(S, c, pixelsImage);$ 
9         if  $pixel_c \neq \emptyset$  then
10            $histo_c = sortedHistogram(pixels_c);$ 
11            $medoids_c = getMedoids(histo_c, k, r);$  // Algoritmo 2;
12           foreach  $pixel \in pixel_c$  do
13              $color = getColor(pixel);$ 
14              $color^* = getClosest(color, medoids);$ 
15              $add(BOCP[color^*], n_{gui}, pixel);$ 
16              $add(BOCC[color^*], n_{gui}, pixel);$ 
17           end
18         end
19       end
20     end
21   end
22 end

```

Algorithm 2: Rilevamento colore dei medoidi (getMethoid viene chiamato nell'algoritmo 1)

Data: $hist_c, k, r$

Result: $medoids$

```

1 begin
2   |    $medoids = \emptyset;$ 
3   |    $medoids[1] = hist_c[1];$ 
4   |    $index = 2, medoidIndex = 1;$ 
5   |   while  $|medoids| \leq k$  and  $index \leq |hist_c|$  do
6   |     |    $color = hist_c[index];$ 
7   |     |   if  $Lum(color, medoids[medoidIndex]) > r$  then           // 3.2.3
8   |       |        $medoidIndex ++;$ 
9   |       |        $medoids[medoidIndex] = color;$ 
10  |     |   end
11  |     |    $index ++;$ 
12  |   end
13 end

```

della GUI in maniera bottom-up, ovvero il livello più profondo viene elaborato per primo. Un pixel sullo schermo è assegnato ad un solo componente e ad un solo colore e la priorità per questa assegnazione si basa sulla profondità della gerarchia della GUI. Queste euristiche consentono di assegnare pixel in base allo z-index nella GUI; se sono presenti componenti all'interno di un contenitore, i pixel vengono prima assegnati alle componenti; quindi, i pixel rimanenti vengono assegnati al contenitore. La funzione `getPixels(S,c,PixelImage)` ritorna un DataFrame contenente come colonne ("x", "y", "colore-RGB"). Nella funzione `getPixels(S,c)`, per rispettare che tutti i pixel vengono presi una sola volta è stato creato un `HashMap`, se il pixel in posizione (x,y) è stato già scelto non succede nulla e si passa alle componenti successive, altrimenti viene memorizzata la nuova coordinata (x,y) e l'algoritmo prosegue.

Una volta ottenuti tutti colori dei pixel delle componenti da questi verrà creato un istogramma di colori, che verrà ordinato in base ai colori più frequenti. Dopo aver ottenuto i colori dei pixel, i colori delle componenti e dopo aver creato l'istogramma andremo a calcolare i medoid⁴ dei colori delle componenti, definito nell'Algoritmo 2. I k colori più

⁴I Medoid sono oggetti rappresentativi di un set di dati o di un cluster all'interno di un dataset la cui somma delle differenze rispetto a tutti gli oggetti nel cluster è minima

frequenti per ciascun componente vengono rilevati nell'istogramma dei colori di ciascun componente evitando i gradienti e le ombre dei caratteri che vengono automaticamente iniettati da Android. Per questo, l'algoritmo attraversa l'istogramma del colore ordinato, dal maggiore al minor numero di volte in cui troviamo il colore, alla ricerca di blocchi nel rapporto di contrasto superiori a una soglia r . Per quanto riguarda il calcolo del rapporto di contrasto tra due colori a e b , è stato utilizzato il rapporto di contrasto basato sulla luminanza (anche detta, luminanza relativa) [27] $\text{Lum}(a,b)$ che spiega meglio le differenze tra i colori del testo e dello sfondo. Le equazioni da 3.2.1 a 3.2.3 descrivono la procedura per calcolare il rapporto di contrasto basato sulla luminanza tra due colori a e b ($\text{Lum}(a,b)$), calcolando la luminanza relativa L di ciascun colore 3.2.2; La luminanza relativa si basa sui livelli di colore RGB di un determinato colore, con un valore di livello in $[1, 255]$. Pertanto, il primo colore nell'istogramma è il medoid (ovvero il colore più frequente in un cluster) per il primo cluster (ovvero il più frequente nell'istogramma). Un nuovo cluster (ovvero testo) viene rilevato quando il rapporto di contrasto è superiore a r . Questa procedura viene applicata fino a quando non vengono rilevati k medoid. In questo lavoro il medoid è il colore più frequente in un cluster anziché la media, perché la media dei colori in un cluster potrebbe modificare le caratteristiche dei colori definiti dagli sviluppatori nel progetto originale.

$$T(level) = \begin{cases} \frac{level}{12.92} & level \leq 0.03928 \\ \left(\frac{level+0.055}{1.055}\right)^{2.4} & otherwise, \end{cases} \quad (3.2.1)$$

$$L(a) = 0.216 \cdot T\left(\frac{red_a}{255}\right) + 0.7152 \cdot T\left(\frac{green_a}{255}\right) + 0.0722 \cdot T\left(\frac{blue_a}{255}\right). \quad (3.2.2)$$

$$\text{Lum}(a,b) = \begin{cases} \frac{L(a)+0.05}{L(b)+0.05} & L(a) > L(b) \\ \frac{L(b)+0.05}{L(a)+0.05} & otherwise, \end{cases} \quad (3.2.3)$$

Infine, i colori dei componenti della GUI vengono discretizzati (cioè viene effettuata la quantizzazione dei colori) utilizzando i k colori dei medoid negli istogrammi corrispondenti. Ciò significa che il colore originale di ciascun pixel di una componente della GUI viene sostituito con il colore quantizzato più vicino (cioè, color medoid). Per memorizzare i colori quantizzati per ogni componente della GUI, vengono utilizzate le strutture BOCP e BOCC.

Terminato l'algoritmo le due strutture HashMap verranno salvate in file .pickle per poterle utilizzare durante lo sviluppo degli algoritmi di intelligenza artificiale.

3.3 Seconda fase: Architettura dell'algoritmo genetico realizzato

Ora andremo ad analizzare nel dettaglio la struttura utilizzata e le singole operazioni per generare l'algoritmo genetico multi-obiettivo. I passi sono i seguenti e sono rappresentati come nella Figura 2.7. Per prima cosa deve essere definito il problema da dover risolvere, definendo le variabili in input, le fitness function e i valori di vincolo. La gestione dei vincoli è essenziale per risolvere un problema di ottimizzazione.

Problema: Il problema prende in input gli HashMap BOCC e BOCP, una lista contenente il tempo di permanenza per ogni interfaccia in secondi e il path della cartella contenente screenshot e snapshot, una lista di tuple contenente i colori iniziali (ovvero le chiavi di bocp/bocc) e una tavola di 512 colori.

Per la creazione della tavolozza di colori ciò che verrà fatto sarà prendere (i) m colori da BOCP iniziale (quindi prenderemo tutte le chiavi facendole diventare i primi m colori), (ii) un colore bianco (iii) un colore nero (iv) una tavolozza con un armonia di colori equidistanti con $(512-m-2)/3$ colori, (v) una tavolozza monocromatica di $(512-m-2)/3$ colori con differente saturazione e luminosità (vi) e una tavolozza di armonia equidistante con saturazione e luminosità casuali per ciascuno dei colori nell'armonia di (iv). Quindi in totale sommando tutti i punti descritti verrà generata una tavolozza con 512 colori in formato rgb. Il tono nello spazio HSB/HSV varia da 0° a 359° e ogni valore intero corrisponde a un colore (ad esempio, 0° è rosso, 120° è verde e 240° è blu), per derivare colori equidistanti dividendo lo spazio della tonalità in un numero di colori n è stato calcolato $360/|\text{color}|$, dove $|\text{color}|$ rappresenta il numero dei colori. Ad esempio, un'armonia equidistante di 20 colori avrà una differenza di tonalità di $(360/20)$ tra ciascun colore. Tutti questi colori però saranno convertiti sempre in formato RGB e questo formato, ovvero la tupla (RGB), sarà l'unico formato utilizzato durante tutta la fase di sviluppo per rappresentare i colori. Definito il problema verrà implementata una funzione di valutazione che permetterà di calcolare le tre fitness function e il valore del vincolo. Questa funzione, inoltre, verrà chiamata ogni volta che la popolazione cambierà.

La **Prima fitness function** deve essere minimizzata e calcola la quantità di energia consumata dall'interfaccia, poiché l'implementazione iniziale è molto onerosa dal punto di vista computazionale è stata definita una variante più leggera. La precedente implementazione iterava tutte le coordinate (x,y) nell'HashMap di BOCP cercando l'indice delle coordinate inserite, effettuando un grand numero di operazioni, soprattutto quando gli elementi si trovavano nelle ultime posizioni dell'HashMap per poter eseguire completamente il calcolo

del consumo totale.

$$ECF = \sum_{h=1}^{nh} p_h \cdot \sum_{i=1}^{ni} (PowerPixel(Solution_i) \cdot CounterIndex(BOCP_h)_i) \quad (3.3.1)$$

L'equazione 3.3.1 calcola il consumo totale in Joule; per ogni interfaccia moltiplica il tempo di permanenza in secondi (p_h) per il consumo PowerPixel di tutte le soluzioni Solution per il numero di elementi in lista CounterIndex(BOCP). Se non si calcola il tempo di permanenza per ogni interfaccia si può creare una lista contenente solo 1, in base al numero di interfacce e il consumo sarà in Watt. CounterIndex(BOCP) conta, per una interfaccia h, il numero di pixel per ogni chiave in BOCP. Per ogni soluzione (ossia per ogni gene del cromosoma) verrà moltiplicato il valore del consumo PowerPixel dell'indice di Solution con il relativo numero di elementi nell'indice di CounterIndex e il valore risultante verrà moltiplicato per il tempo di permanenza (p_h). Il valore restituito da PowerPixel è descritto nell'espressione 2.3.2

La **Seconda fitness function** deve essere massimizzata e riguarderà il calcolo del contrasto tra le componenti. Nel documento originale non viene descritta completamente come strutturare correttamente il calcolo del valore di contrasto. Inoltre, poiché non viene spiegato correttamente il calcolo del **vincolo del contrasto** tra le componenti è stata realizzata una nuova soluzione, calcolando contemporaneamente sia la seconda fitness function che il constraint value in un'unica soluzione. L'algoritmo 3 inizia creando un array contenente come

Algorithm 3: Seconda Fitness function e constraint value

Data: $solution, bocc, snapshot, CnTh$

Result: $contrast$

```

1 begin
2   |    $contrast = [0, 0];$ 
3   |    $screen = getSnapshot(path);$ 
4   |   foreach  $interface \in screen$  do
5   |     |    $tcon = TCon(solution, bocc, interface, nInterface, CnTh) // Algoritmo 4;$ 
6   |     |    $contrast[1] += tcon[1];$ 
7   |     |    $contrast[2] += tcon[2];$ 
8   |   end
9 end

```

primo valore la fitness function e il secondo il valore del vincolo. Prende il path di tutti gli snapshot e usa l'algoritmo TCON 4 per calcolare il contrasto. TCon prende tutte le componenti all'interno dello snapshot inserito e attraverso la funzione getAdiacentComponent

Algorithm 4: algoritmo TCon

Data: *solution, bocc, snapshot, nInterface, CnTh***Result:** *totcontrast*

```

1 begin
2   totcontrast = [0, 0];
3   foreach component ∈ snapshot do
4     listComponent = getAdiacentComponent(snapshot, component);
5     if listComponent ≠ ∅ then
6       foreach child ∈ listComponent do
7         firstComp = BCIndexWithH(bocc, component, nInterface);
8         secondComp = BCIndexWithH(bocc, child, nInterface);
9         contrast =
10        Con(solution[firstComp], solution[secondComp]); // Equazione 3.3.2
11        totContrast[1] += contrast;
12        if contrast < CnTh ∧ (firstComp ≠ secondComp) then
13          totContrast[2] += 1;
14        end
15      end
16    end
17  end

```

restituisce il padre diretto e i figli diretti.

$$Con(a, b) = \left| \frac{299 \cdot red_a + 587 \cdot green_a + 114 \cdot blue_a}{1000} - \frac{299 \cdot red_b + 587 \cdot green_b + 114 \cdot blue_b}{1000} \right| \quad (3.3.2)$$

Per trovare questa gerarchia sfrutto i file snapshot.xml, per ottenere il padre diretto itero tutte radici e controllo se esiste come figlio la componente selezionata, inserendola in una lista, per ottenere tutti i figli diretto è stato creato un HashMap contenente come chiave un padre e come valori i figli, così da ottenere in maniera diretta i figli. Per ogni componente della gerarchia ottenuta e per la componente scelta, viene usata la funzione BCIndexWithH che consente di ottenere l'indice della posizione della componente all'interno dell'HashMap BOCC. Ottenuti i due indici, verrà usata l'equazione 3.3.2 per confrontare il contrasto tra le componenti e lo sfondo per ottenere il valore di totContrast in posizione 0, ovvero la posizione che calcola la seconda fitness function. Se il contrasto è minore di CnTh allora verrà aumentato il valore di

totContrast in posizione 1, ovvero il valore del vincolo.

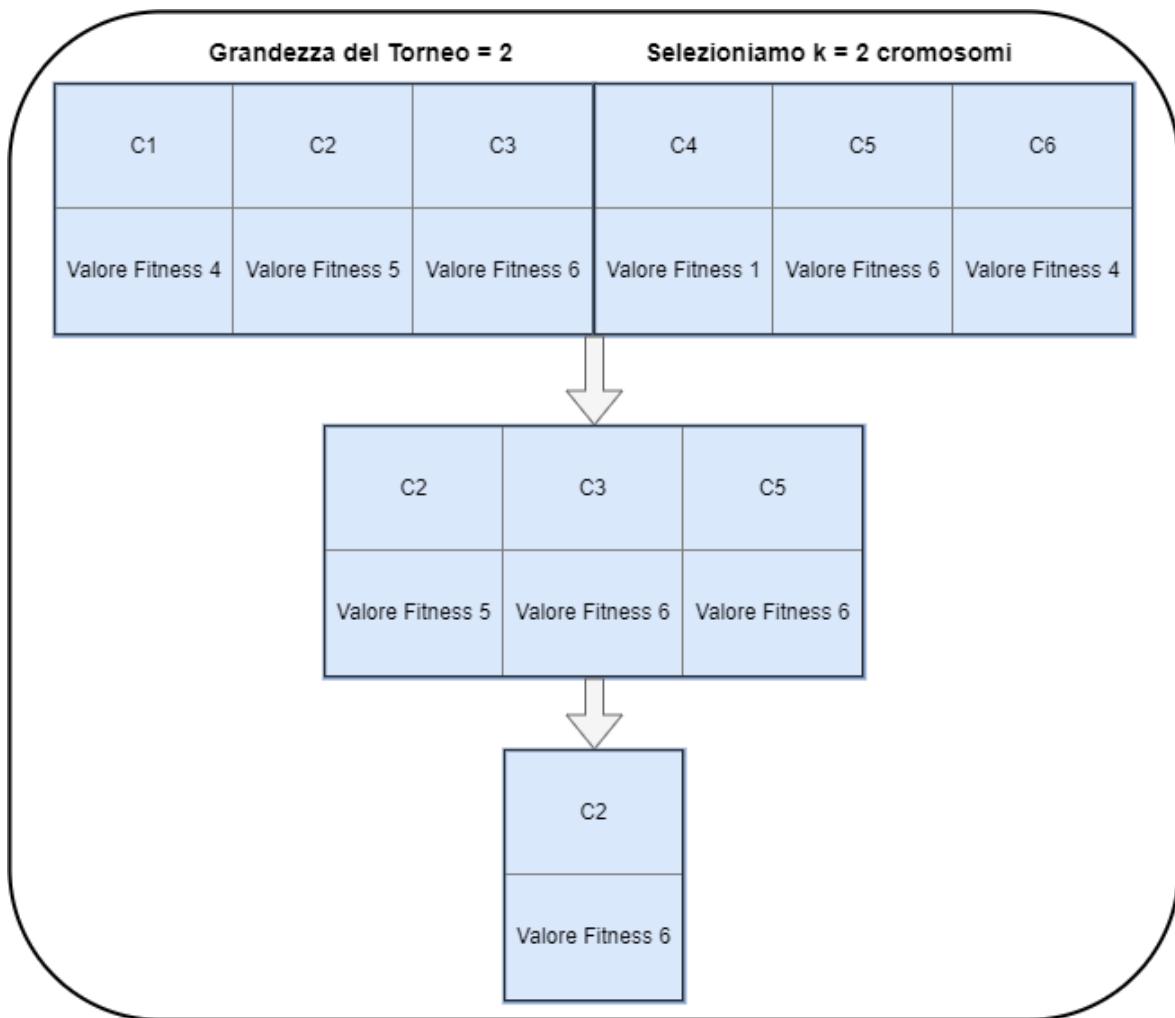
La **Terza fitness function** deve essere minimizzata e serve per realizzare interfacce con colori simili a quelli sviluppati nativamente. Per questo motivo viene utilizzata la distanza tra ogni colore $S[i]$ e il colore $Or[j]$ più vicino nella tavola di colori iniziale di BOCC. Per calcolare la distanza tra due colori a e b, viene utilizzata la distanza Euclidea [28]. Per ogni colore nella soluzione, ovvero ogni gene nel cromosoma, viene calcolato un fattore di penalità α 3.3.3 che, se il colore della soluzione $S[i]$ è diverso dal colore con la distanza più vicina in BOCC allora restituirà 1, altrimenti 2. DF restituisce la somma dei costi delle penalità moltiplicate per la distanza Euclidea tra le soluzioni $S[i]$ e le soluzioni più vicine nella tavola di colori iniziale.

$$\alpha(color) = \begin{cases} 1 & \text{if } color \neq Or^*(color) \\ 2 & \text{otherwise,} \end{cases} \quad (3.3.3)$$

$$DF(S) = \sum_{i=1}^n \alpha(S[i]) \cdot d(S[i], Or^*(S[i])). \quad (3.3.4)$$

La prima fase di questo algoritmo genetico è la **Generazione della popolazione iniziale**, ciò che verrà fatto sarà avere una rappresentazione contenente un gene per ogni chiave nel BOCP, quindi ogni gene definirà i colori di tutti i pixel assegnati ad una determinata chiave. Ogni individuo nella popolazione iniziale ha lo stesso numero di geni, ovvero ha lo stesso numero di colori nelle chiavi date da BOCP. Per assegnare i valori dei colori ai geni della popolazione iniziale vengono presi dei colori generati dalla tavola di colori creata precedentemente in maniera casuale. In Python è stata realizzata una lista di tuple (R,G,B) pari al numero di chiavi di BOCP.

Una volta generata la popolazione iniziale di individui casuali andremo ad effettuare l'operatore di selezione, la scelta è stata quella di utilizzare una **Selezione di tornei binari**. La selezione del torneo è una strategia di selezione utilizzata per selezionare i candidati più adatti dell'attuale generazione in un algoritmo genetico. Questi candidati selezionati vengono quindi trasmessi alla generazione successiva. Solo il candidato più idoneo tra i candidati selezionati viene scelto e viene trasmesso alla generazione successiva. In questo modo si svolgono molti di questi tornei e abbiamo la nostra selezione finale di candidati che passano alla generazione successiva, un esempio è la Figura 3.3. Nella quale vengono presi due individui a caso nel torneo di selezione e tra questi avvengono diverse condizioni per la decisione del miglior individuo, come descritto da Deb K. et al. [29]: (i) la prima riguarda scegliere tra i due individui quello con il valore del vincolo uguale a zero, se entrambi non violano nessun vincolo viene scelto l'individuo con fitness migliore.

**Figura 3.3:** Esempio di Torneo binario

(ii) La seconda invece se nessuno dei due individui non rispetta la condizione (i) allora verrà scelto l'individuo con il valore del vincolo più basso, infine, (iii) se entrambi hanno lo stesso numero di vincoli verrà scelto il valore di fitness più alto. I vincitori del torneo verranno scelti per l'operatore di crossover spiegato in precedenza. Ciò che viene applicato è la pressione di selezione, una misura probabilistica della probabilità di un cromosoma di partecipazione al torneo in base alla dimensione del pool di selezione dei partecipanti, è facilmente modificabile modificando la dimensione del torneo, il motivo è che se la dimensione del torneo è maggiore, gli individui deboli hanno minori possibilità di essere selezionati, perché, se viene selezionato un individuo debole per partecipare a un torneo, c'è una maggiore probabilità che anche un individuo più forte sia in quel torneo⁵. Quindi il metodo di selezione del torneo può essere descritto in pseudocodice 5:

⁵https://en.wikipedia.org/w/index.php?title=Tournament_selection&oldid=1068060239

Algorithm 5: Torneo di selezione

1. scegli k individui (ovvero la dimensione del torneo) dalla popolazione a caso
2. scegli il miglior individuo del torneo con probabilità p
3. scegli il secondo miglior individuo con probabilità $p \cdot (1-p)$
4. scegli il terzo miglior individuo con probabilità $p \cdot ((1-p)^2)$
5. e così via

Terminata quindi la fase di selezione e prendendo gli individui con fitness migliore, passeremo all'operatore di crossover. Il metodo che viene utilizzato è il **Crossover ad un punto**, Figura 3.4. Il crossover ad un punto che utilizziamo viene applicato con una determinata probabilità p_{cross} . Presi due individui, ovvero due genitori, ciò che si farà sarà scegliere un punto casuale e tagliarli, dopodiché le parti di sinistra rimarranno uguali; invece, le parti di destra verranno scambiate ed il risultato finale sarà la realizzazione di due figli, quindi, ciò che si avrà sarà una nuova generazione di figli dati i genitori, che verranno dati all'operatore di mutation.

Infine, l'ultimo operatore che viene descritto è l'operatore di mutation, più nello specifico viene applicato il **Bit-flip mutation** [30] che, con probabilità p_{mut} cambia i colori di un gene con un colore diverso preso in maniera casuale dalla tavolozza di colori creata all'inizio, possiamo vedere un esempio in Figura 3.5.

Al termine dell'operatore di mutation in questo progetto di ricerca gli individui della popolazione che sono uguali verranno eliminati.

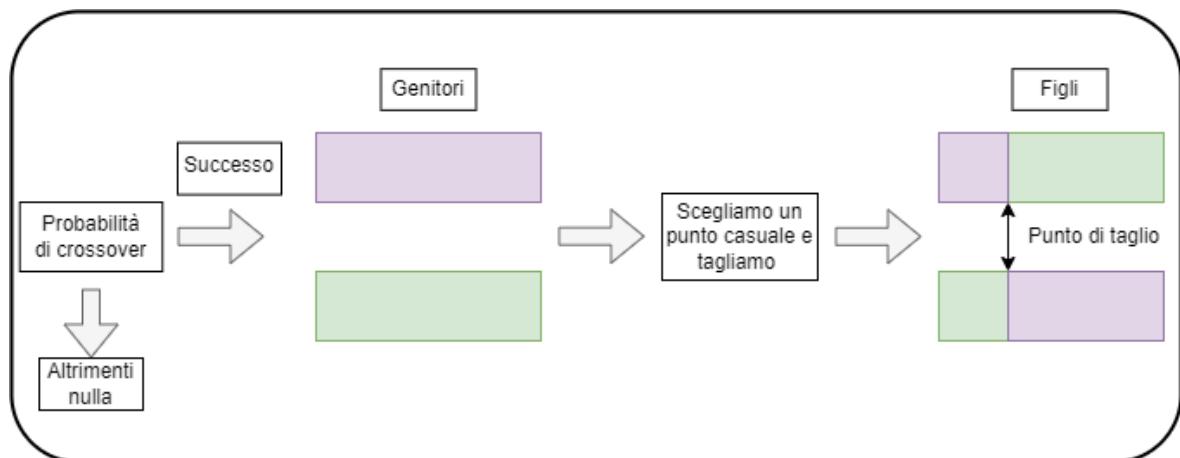
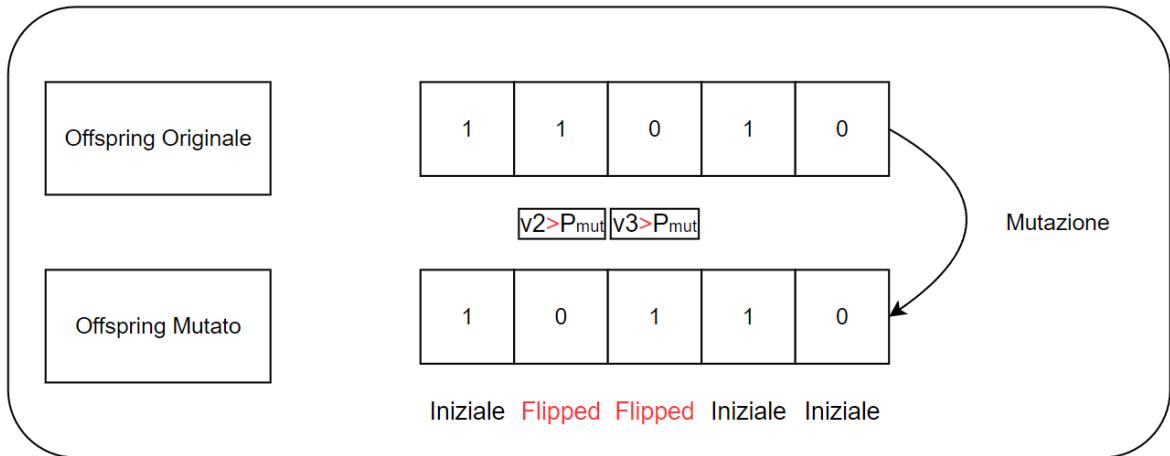


Figura 3.4: Esempio di crossover ad un punto

**Figura 3.5:** Esempio di mutation bit flip

L'eliminazione dei duplicati aiuta a garantire che un accoppiamento produca una prole che non esiste già nella popolazione attuale. In tal caso, viene attivato un altro processo di accoppiamento fino a quando non esistono individui unici.

3.4 Terza fase: Ricostruzione delle nuove interfacce grafiche ottimizzate

In questa terza e ultima fase di sviluppo il lavoro è incentrato sulla ricostruzione delle interfacce grafiche attraverso l'uso dei risultati dalle esecuzioni dell'algoritmo genetico multiobiettivo. Per la realizzazione delle nuove interfacce grafiche, una volta ottenute le soluzioni è stato implementato un algoritmo che sostituisce i colori del design originale e crea una nuova interfaccia con le soluzioni ottenute. La sostituzione dei colori viene fatta in base all'indice delle soluzioni e all'indice dei colori del design originale. Ad esempio, se nelle coordinate (x,y) (esempio, (5,6)) esiste il colore Azzurro (in formato RGB(0, 127, 255)) situato nella posizione 3 di BOCP (ovvero, BOCP[3]), verrà presa la soluzione ottenuta dall'algoritmo nella stessa posizione (ovvero, S[3]). Nella ricostruzione delle immagini verrà fatto un controllo sulle componenti e in caso esistono degli attributi "Image", a queste non verranno alterati i colori e verranno preservate, questa scelta è stata svolta per non alterare la l'aspetto dell'applicazione. Lo pseudo-codice per poter ricostruire le interfacce grafiche è spiegato nell'algoritmo 6. La funzione `getView(snapshot)` prende tutte le componenti costituite da "Image" e, iterando per tutti i pixel dell'immagine, se non esiste un Image nella posizione (x,y) allora viene restituito il corrispondente indice nella lista dei vecchi colori (ovvero, BOCP) con lo stesso colore e tramite questo indice verrà sostituito il colore del pixel dell'immagine (x,y) con il nuovo

colore avente lo stesso indice.

Algorithm 6: algoritmo BuildImages

Data: $image, snapshot, oldColors, newColors$

Result: $newInterface$

```
1 begin
2   listView = getView(snapshot);
3   foreach  $x \in width(image)$  do
4     foreach  $y \in height(image)$  do
5       if checkImage(listView,  $x, y$ ) == False then
6         indexColor = getIndexColor(image,  $x, y, oldColors$ );
7         newInterface = setNewColors(image, indexColor, newColors);
8       end
9       else
10        newInterface = setOldColors(image,  $x, y$ );
11      end
12    end
13  end
14 end
```

CAPITOLO 4

Design dello Studio Empirico

Sulla base del contesto di studio svolto sono stati proposti i seguenti obiettivi di ricerca:

- OR 1. Definire se il sistema realizzato riesce ad ottimizzare i consumi energetici delle nuove interfacce grafiche.
- OR 2. Definire se le interfacce prodotte riescono ad avere un adeguato contrasto
- OR 3. Definire se le componenti delle interfacce riescono ad essere distinguibili da persone con problemi visivi.
- OR 4. Definire quale approccio di ottimizzazione multi-obiettivo è adatto per il sistema realizzato

4.1 OR1- Definire se il sistema realizzato riesce ad ottimizzare i consumi energetici delle nuove interfacce grafiche

Una volta realizzate le nuove interfacce si dovrà verificare se tale ottimizzazione produce risultati soddisfacenti. Per riuscire a rispondere a questo obiettivo verranno misurati i consumi energetici delle interfacce originali e quelle prodotte dall'algoritmo realizzato per verificare se il sistema proposto riesce a ridurre la quantità di energia consumata dal display. Per poter rispondere a questo obiettivo di ricerca, una volta realizzate le nuove interfacce grafiche, verrà calcolato il consumo energetico di tutte le interfacce attraverso la formula 2.3.2,

le interfacce verranno raggruppate per tipologie e confrontate rispetto al design originale per valutare se le nuove interfacce ottimizzate riescono a consumare una quantità inferiore di energia.

4.2 OR2 - Definire se le interfacce prodotte riescono ad avere un adeguato contrasto

Tra gli obiettivi di ricerca viene definito se le interfacce prodotte riescono ad avere un adeguato contrasto tra le componenti e lo sfondo, rendere le componenti e lo sfondo dello stesso colore non permetterà la corretta visualizzazione delle informazioni contenute in un’interfaccia e per questo verranno condotti dei test con degli utenti per verificare se le componenti sono distinguibili nelle interfacce.

4.3 OR3 - Definire se le componenti delle interfacce riescono ad essere distinguibili da persone con problemi visivi

Poichè in OR3 si verifica se gli utenti riescono a visualizzare correttamente le componenti all’interno delle nuove interfacce, questo studio di ricerca vuole estendere questa domanda anche alle persone con problemi visivi. Per questo verranno considerate nello studio anche questa tipologia di utenti chiedendo le tipologie di problemi e analizzando separatamente anche le loro impressioni. La tipologia di problema visivo considerato sarà l’astigmatismo.

4.4 OR4 -Definire quale approccio di ottimizzazione multi-obiettivo è adatto per il sistema realizzato

Definire quale tra i due algoritmi scelti è adatto per l’algoritmo implementato è uno degli obiettivi di ricerca di questa ricerca. Algoritmi diversi funzioneranno meglio o peggio su diversi tipi di problemi, di conseguenza, verificare quale tra questi riesce a ottimizzare questo problema non è una scelta banale. Gli algoritmi scelti sono infatti tutti algoritmi di ricerca multi-obiettivo e sono stati applicati gli algoritmi NSGA-II: Non-dominated Sorting Genetic Algorithm [31] e SMS-EMOA: Multiobjective selection based on dominated hypervolume [32]. Per rispondere e raggiungere questo obiettivo saranno necessari i risultati ottenuti dai precedente obiettivi di ricerca, così da capire attraverso metodologie scientifiche ed

esperimenti con gli utenti quale algoritmo è adatto e ottimale per raggiungere gli obiettivi prefissati.

4.5 Contesto d'uso

In questo studio è stato utilizzato un emulatore integrato in Android Studio per poter acquisire poi le schermate, il dispositivo utilizzato per lo svolgimento dello studio è un Pixel 4 con risoluzione 1080 x 2280. L'utilizzo di un emulatore è stato necessario a causa dell'assenza di un dispositivo Android per l'acquisizione delle applicazioni e per la cattura delle relative interfacce. Il dispositivo emulato utilizza la versione di Android 11.0 con API 30. La scelta del dispositivo non inciderà sull'acquisizione delle applicazioni e delle interfacce dello studio.

Per questo studio sono stati selezionate undici applicazioni dal Playstore, le applicazioni differiscono tra loro per categoria e colorazioni. Come descritto nel capitolo precedente, il numero di applicazioni non influirà sulla qualità dell'output prodotto dal sistema. Le applicazioni scelte sono le seguenti: Board Game Geek¹ che permette di avere le informazioni sul giochi da tavolo e le principali tendenze del momento, Daily Money² che permette di gestire i propri conti, Deadline³ utilizzato come promemoria, DiabetePlus⁴ un'applicazione che aiuta le persone diabetiche, Groupon⁵ che permette di risparmiare su diverse tipologie di prodotti, Keep⁶ per annotare e utilizzare promemoria anche attraverso un assistente virtuale, Learn Music Notes⁷ per imparare spartiti musicali, Safe Security⁸ per pulire la memoria dello smartphone e avere una maggiore sicurezza, Tasks⁹ che permette di annotare i task che dobbiamo svolgere, The Weather Channel¹⁰ per avere notizie sulle previsioni del meteo, Tic Tac Toe¹¹ un popolarissimo gioco di carta e matita sviluppato per smartphone.

¹<https://play.google.com/store/apps/details?id=com.ebytestudio.mybgg&gl=IT>

²<https://play.google.com/store/apps/details?id=com.bottleworks.dailymoney>

³<https://play.google.com/store/apps/details?id=website.igor.dyatlov.deadline&gl=IT>

⁴<https://play.google.com/store/apps/details?id=com.squaremed.diabetesplus.type1>

⁵<https://play.google.com/store/apps/details?id=com.groupon>

⁶<https://play.google.com/store/apps/details?id=com.google.android.keep>

⁷<https://play.google.com/store/apps/details?id=de.melvil.noten&gl=IT>

⁸<https://play.google.com/store/apps/details?id=com.qihoo.security&gl=IT>

⁹<https://play.google.com/store/apps/details?id=org.dmf.tasks>

¹⁰<https://play.google.com/store/apps/details?id=com.weather.Weather>

¹¹<https://play.google.com/store/apps/details?id=com.earthblood.tictactoe>

4.6 Tecnologie e Metodologie del contesto d'utilizzo

Per affrontare lo studio con una diversa prospettiva e per verificare la possibilità di poter replicare il documento di partenza è stato utilizzato il linguaggio Python. Nello studio originale è stato utilizzato il linguaggio JAVA con il framework JMetal. Invece in questo studio, per l'implementazione della seconda e terza fase del capitolo 3 è stato utilizzato come linguaggio Python 3.8. La seconda fase dello sviluppo è stata svolta tramite Google Colaboratory¹² che permette di sviluppare online codice in Python e utilizzare Google Drive in maniera semplice e veloce. La terza fase invece è stata svolta utilizzando il framework Pymoo versione 0.6 [33] per realizzare l'algoritmo genetico multi-obiettivo e come macchine per l'esecuzione i server dipartimentali dell'Università degli Studi di Salerno a causa della grande quantità di calcolo richiesta per l'esecuzione e la quantità di tempo richiesta per svolgere l'algoritmo per ogni applicazione. Per poter sviluppare e replicare il sistema allo stesso modo saranno necessarie le seguenti impostazioni:

Per la fase riguardante l'algoritmo BOCP_BOCC è necessario impostare il valore di $k=3$, per estrarre tre diversi colori da ciascun componente della GUI: sfondo, testo (o secondo colore principale) e il terzo colore per il bordo. Il parametro $r=1.6$ poiché è un limite inferiore comune per il contrasto tra i colori nei bordi e lo sfondo. GUIS invece è una lista di stringhe dove ogni stringa rappresenta una interfaccia, poiché snapshot e screenshot hanno lo stesso nome ma diversa tipologia, ovvero, file .png per gli screenshot e .xml per gli snapshot, sarà necessario aggiungere la rispettiva tipologia per denominare i due file durante l'esecuzione con i rispettivi formati. L'esecuzione di questo algoritmo è stato automatizzato per tutte le applicazioni scaricate, per ogni applicazione verranno salvati due file .pickle chiamati con il nome dell'applicazione e che terminano con "_BOCP" e "_BOCC" contenenti i rispettivi Hash-Map. Questo processo è stato automatizzato solo per poter ottenere in un'unica volta tutti i file, ma è possibile svolgere questa operazione per un'unica applicazione. Ottenuto l'HashMap di BOCC da questo verrà effettuata una fase di filtraggio, nella quale verranno eliminate le tuple uguali costituite da (colore, numeroInterfaccia, componente) per ridurre il numero di operazioni e la quantità di elementi della variabile, riuscendo a ridurre significativamente il tempo di esecuzione.

Invece, per la fase riguardante lo sviluppo dell'algoritmo genetico multi-obiettivo sarà necessario impostare diversi parametri. Partendo dalla definizione del Problema, la classe del problema è stata definita attraverso *ElementwiseProblem*, con il quale si può prendere un

¹²<https://research.google.com/colaboratory/faq.html>

cromosoma per volta, consentendo così la parallelizzazione del problema e permettendo di elaborare le fitness function e il vincolo in maniera separata dagli altri individui della popolazione. Questa scelta progettuale è dovuta al fatto che in questo studio interessa calcolare una soluzione per volta e non un insieme di soluzioni. I parametri invece impostati per l'esecuzione sono: $pop_size=50$ e $n_evaluation=10000$. Per il crossover ad un punto implementato, la probabilità che venga effettuata la sostituzione avviene con una probabilità $p_{cross}=0.9$. Per l'operatore di mutation bit-flip invece la probabilità di sostituzione di un colore all'interno di una soluzione avverrà con probabilità $p_{mut} = 1/|BOCP|$, dove $|BOCP|$ rappresenta il numero di chiavi di BOCP.

Definito quindi il problema e gli operatori andremo a gestire il processo per poter svolgere correttamente l'esecuzione. Una volta istanziato il problema definiremo l'algoritmo da utilizzare, esistono diversi algoritmi multi-obiettivo e in questo contesto sono stati utilizzati NSGA-II e SMS-EMOA. La scelta di questi due algoritmi è stata presa perché entrambi sono algoritmi multi-obiettivo e perché entrambi considerano durante l'esecuzione i vincoli del problema. L'oggetto algoritmo NSGA-II/SMS-EMOA prende in input la grandezza della popolazione, gli operatori e la funzione di eliminazione, in Pymoo poi verrà richiamata l'API **minimize** che avrà come obiettivo di minimizzare le fitness function e i valori dei vincoli e tracerà il fronte di Pareto. Inoltre, minimize crea una copia completa dell'oggetto algoritmo prima dell'esecuzione. Ciò garantisce che due esecuzioni indipendenti con lo stesso algoritmo e lo stesso seme casuale abbiano gli stessi risultati senza effetti collaterali. Il seme utilizzato per questo algoritmo è 1. La seconda fitness function, poiché "minimize" minimizza le soluzioni, ottenuto il valore dall'esecuzione dovrà essere reso negativo. Per velocizzare il processo di ottimizzazione e consentire un maggior processo di parallelizzazione verrà utilizzato l'oggetto starmap permettendo così la programmazione multithreading. Gli individui della popolazione vengono processati su diversi thread, con un miglioramento generale sui tempi di esecuzione. Il numero di thread per la macchina utilizzata è $n_thread=10$.

Per l'archiviazione dei dati sono stati realizzati diversi file .csv per poter analizzare i risultati ottenuti dall'algoritmo genetico, per poter ricostruire le interfacce grafiche e per usi statistici. Da un lato abbiamo un file .csv contenente tutte le principali informazioni dell'algoritmo multi-obiettivo utilizzato (considerando NSGAII e SMS-EMOA) costituito dalle tuple `(app_name,n_thread,n_interface,pop_size,time_execution,n_evals,hist_f,hist_cv,hist_cv_avg)`. La colonna **app_name** rappresenta il nome dell'applicazione, **n_thread** il numero di thread istanziati per poter parallelizzare e velocizzare l'esecuzione del sistema, **n_interface** il numero di interfacce dell'applicazione, **pop_size** il numero di individui creati nel problema,

time_execution il tempo di esecuzione per poter eseguire tutte le operazioni, **n_evals** il numero di valutazioni delle funzioni, **hist_f** i valori dello spazio oggettivo in ogni generazione, **hist_cv** la violazione dei vincoli in ogni generazione, **hist_cv_avg** la violazione media del vincolo nell'intera popolazione. In Pymoo si può utilizzare save_history per archiviare solo le informazioni necessarie e usarle in seguito per sviluppare grafici e gli ultimi valori con la sigla "hist" sono stati collezionati proprio grazie a questa istruzione.

Per ogni interfaccia è stato creato un file .csv contenente tutte le principali informazioni dell'esecuzione dell'algoritmo rispetto all'applicazione analizzata con le tuple (id, app_name, solution, fitness_function), **id** per identificare la tupla, **app_name** per il nome dell'applicazione, **solution** che rappresenta la lista di colori ottenuta dall'algoritmo genetico, la lista è composta da un insieme di tuple (RGB) e **fitness_function** che rappresenta la tupla contenente i valori delle tre fitness function. Tutte le soluzioni con il valore del vincolo maggiore di 0, se esistono, non verranno inserite in questo file .csv, per consentire alle colorazioni che non consentono un adeguato contrasto di non essere analizzate. Questo file serve principalmente per la terza fase descritta nel capitolo precedente, per poter ricostruire le interfacce grafiche con le nuove soluzioni ottenute dall'algoritmo genetico rispetto alle fitness function delle soluzioni. Per rispondere ad **OR1**, infatti, è stata utile la realizzazione di questo file per poter realizzare quattro tipologie di misurazione, per ogni interfaccia sono state considerate le soluzioni con la mediana e il minor consumo energetico (ovvero, il valore della prima fitness function), il valore massimo del contrasto delle componenti (ovvero, il valore della seconda fitness function) e la soluzione che si avvicina maggiormente al design originale (ovvero, il valore della terza fitness function). Quindi, ricostruite le interfacce verranno calcolate queste cinque misure tramite le quali verrà creato un ultimo file che conterrà le tuple (app_name,max_contrast,median_cons,min_cons,min_original,original_cons) con i valori totali in Watt di tutte le interfacce con la stessa tipologia. I dati ottenuti da questo ultimo file permetteranno di analizzare i consumi andando a definire se le interfacce realizzate dall'algoritmo genetico riescono effettivamente a realizzare delle interfacce grafiche ottimizzate e ad ottenere interfacce grafiche con un ridotto consumo energetico.

Per rispondere invece ad **OR2** e **OR3** sono stati realizzati dei questionari utenti tramite Google Forms¹³ che consente di realizzare in maniera semplice e immediata form da distribuire agli utenti. Per l'esperimento sono state scelte 16 partecipanti in totale, 8 persone normovedenti e 8 persone astigmatiche, l'esperimento sarà controllato e verrà svolto attraverso l'uso del proprio smartphone.

¹³<https://www.google.it/intl/it/forms/about/>

7. Seleziona gli elementi che NON riesci a vedere correttamente *



Seleziona tutte le voci applicabili.

- Pulsanti
- Testo
- Immagini
- Sfondo
- Non ci sono elementi che non visualizzo correttamente

Figura 4.1: Esempio di domanda del questionario

Il questionario è diviso in tre parti, una fase di pre-esperimento nella quale vengono collezionate le principali informazioni sugli utenti come la loro età, le ore di utilizzo giornaliero dello smartphone, quale sistema operativo usano abitualmente, se soffrono di astigmatismo e quali aspetti preferiscono in un'interfaccia ovvero: (i) una interfaccia che aumenta la durata della batteria (ii) una interfaccia che sia esteticamente bella e (iii) una interfaccia che abbia un adeguato contrasto per distinguere le componenti. La seconda fase invece riguarda l'esperimento vero e proprio, nella quale gli utenti devono selezionare quali elementi in un'interfaccia non riescono a visualizzare correttamente e se le interfacce prodotte dall'algoritmo sono piaciute. Per l'esperimento sono state selezionate sei applicazioni e per ognuna di esse è stata scelta una interfaccia con le colorazioni differenti in base ai parametri spiegati

in precedenza (ovvero, consumo minimo, massimo contrasto ecc.). Se gli utenti soffrono di astigmatismo verrà consigliato loro di non usare gli occhiali così da capire quanto incide realmente la colorazione sull'interfaccia ottimizzata. Ad ogni interfaccia, quindi, verrà chiesto ai partecipanti di selezionare le componenti che non riescono a visualizzare correttamente come nella Figura 4.1 e in caso non ci sono problemi nella visualizzazione delle componenti dovranno selezionare la voce apposita. Dopo aver visualizzato e espresso le problematiche relative alla visualizzazione assegneranno un punteggio su una scala da 1 (per niente) e 5 (tanto) sull'estetica delle interfacce ottimizzate visualizzate e successivamente verrà mostrata l'interfaccia originale e verrà chiesto loro quale preferisco esteticamente tra il design originale e le interfacce ottimizzate. Infine, il post-esperimento nel quale gli utenti daranno un feedback finale sugli esperimenti svolti, inserendo se in generale preferiscono le interfacce prodotte o quelle originali, inserendo un punteggio per l'estetica generale delle interfacce e uno spazio libero per eventuali commenti sull'esperimento. Per rispondere ad **OR4** verranno confrontati i punteggi ottenuti dai partecipanti dell'esperimento sull'estetica delle interfacce prodotte dai due algoritmi e sulla quantità di componenti che non si riescono a visualizzare correttamente.

CAPITOLO 5

Risultati

L'esecuzione dell'algoritmo genetico multi-obiettivo ha portato diversi risultati interessanti, da un lato è emersa la capacità di questo sistema di riuscire a realizzare interfacce ottimizzate per consumi energetici e con un adeguato contrasto, dall'altro invece è emersa la quantità di calcolo e tempo richiesto per poter ottenere dei risultati da questo software.

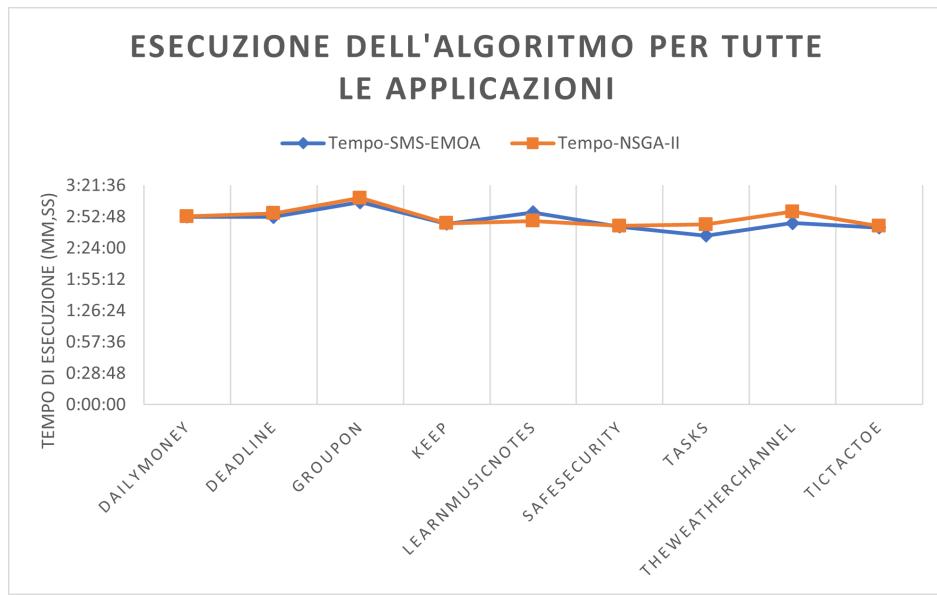


Figura 5.1: Confronto dei tempi di esecuzione dei due algoritmi rispetto a 9 applicazioni.

A causa di diverse limitazioni che verranno spiegate nel prossimo capitolo, non sono state considerate tutte le applicazioni di partenza per valutare il consumo energetico e il contrasto tra le componenti, per questo le applicazioni che hanno risposto agli obiettivi



Figura 5.2: Risultati applicazione Board Geek Game design: originale vs ottimizzato.

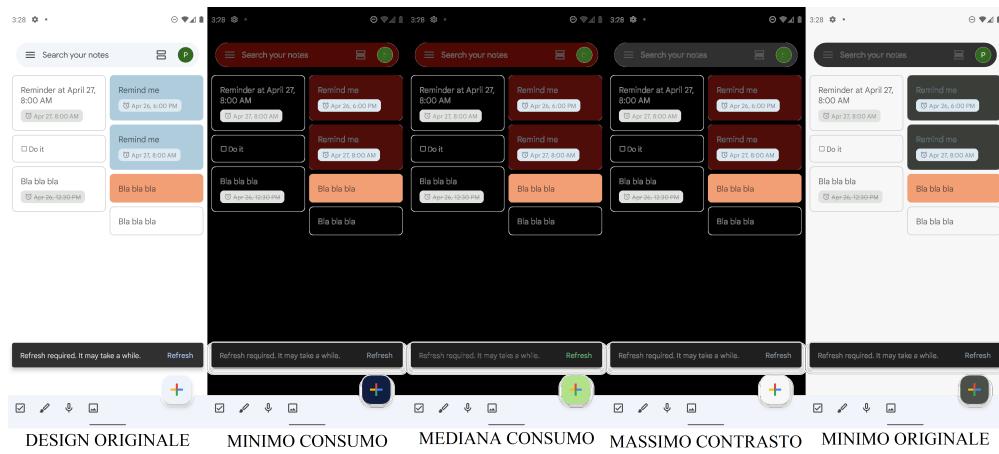


Figura 5.3: Risultati applicazione Google Keep design: originale vs ottimizzato.

di ricerca sono diventate nove. Dai risultati ottenuti, considerando gli iper-parametri del capitolo precedente, il tempo per poter eseguire i due algoritmi è pressoché simile Figura 5.1, in totale per eseguire SMS-EMOA su tutte le applicazione ci vogliono circa 25 ore e 23 minuti, per NSGA-II circa 25 ore e 46 minuti, con una media rispettivamente di 2 ore e 49 minuti e 2 ore e 51 minuti.

Per rispondere ad **OR1** sono stati confrontati i consumi energetici dei design originali rispetto a quattro differenti tipologie dall'insieme di soluzioni prodotte dall'algoritmo genetico realizzato, come la realizzazione dell'interfaccia con un consumo inferiore, la mediana del consumo, il massimo contrasto tra le componenti e il design più vicino a quello originale. Calcolando quindi il consumo di queste nuove interfacce e confrontandole con il consumo dell'interfaccia originale ci sono stati ottimi risultati sulla percentuale di risparmio energetico che si può ottenere utilizzando questo algoritmo. Alcuni esempi di Immagini prodotte sono le Figure 5.2 e 5.3.

In Figura 5.4 invece sono evidenziate le medie di ogni tipologia rispetto al design originale, in generale la quantità media di risparmio supera il 70% di consumo energetico in Watt, per quanto riguarda la mediana in Figura 5.5 il valore ottenuto medio supera l'80%. Il risparmio sul consumo energetico per tutte e quattro le tipologie, sia per NSGA-II che SMS-EMOA, permette di diminuire di oltre la metà i consumi in Watt dei display che impattano le batterie dei dispositivi mobile Android.

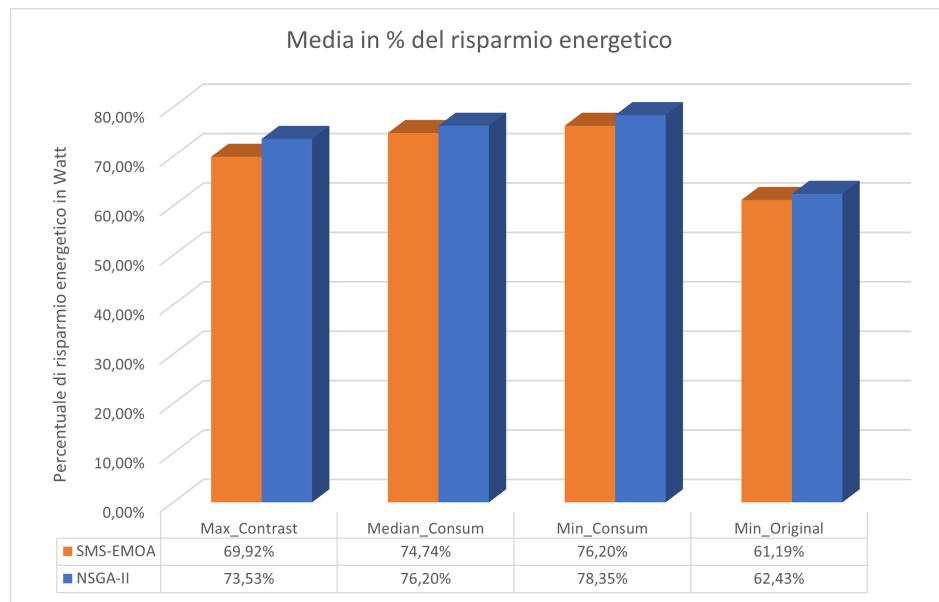


Figura 5.4: Media in percentuale di ottimizzazione dei consumi in Watt di 9 applicazioni.

Per rispondere ad **OR4**, confrontando le interfacce ottimizzate realizzate dalle due tipologie di algoritmi NSGA-II e SMS-EMOA, i risultati ottenuti sia per la media che per la mediana sono molto simili tra di loro, con una differenza media tra le medie di circa il 2%. In linea generale, sia per la media che per la mediana l'algoritmo NSGA-II riesce a superare quasi sempre la percentuale di risparmio energetico delle interfacce prodotte da SMS-EMOA ma i risultati ottenuti non sono statisticamente significativi per poter notare grandi differenze nei risultati.

L'esperimento è stato svolto da sedici persone, cinque di queste hanno un'età compresa tra i 50 e i 60, le restanti un'età compresa tra i 20 e i 33, con una media di 32.1 e deviazione standard di 14.3 e in media è durato circa 10 minuti. Dai risultati del pre-esperimento è emerso che su sedici persone, dieci di queste usano quotidianamente uno smartphone Android e tutte le persone over 50 ne usano uno. Ulteriormente il 56.3% preferirebbe avere interfacce grafiche che aumentano la durata della batteria dei propri smartphone, il 25% una interfaccia che sia esteticamente bella e il 18.8% interfacce con un adeguato contrasto tra le componenti. Per

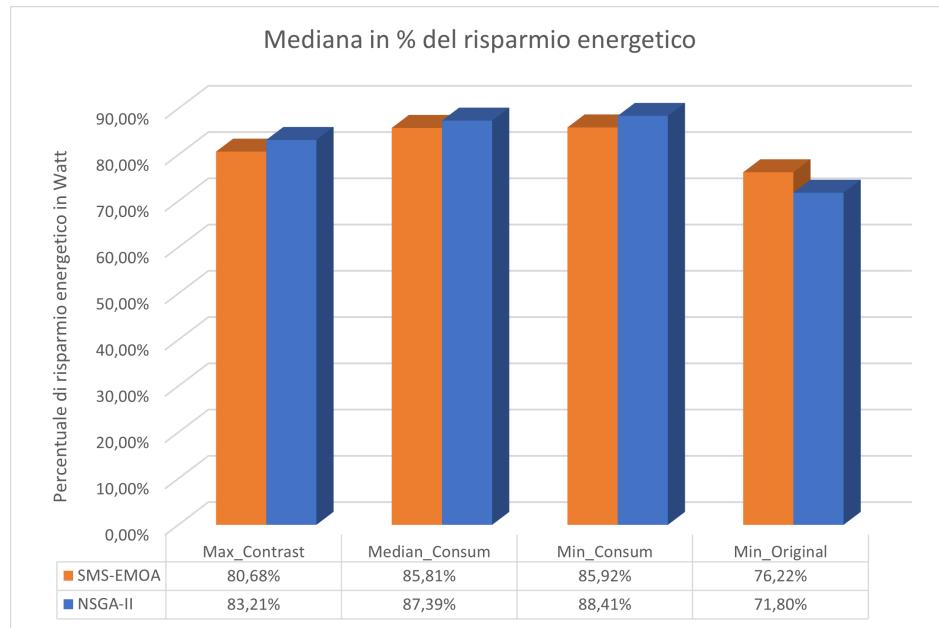


Figura 5.5: Mediana in percentuale di ottimizzazione dei consumi in Watt di 9 applicazioni.

rispondere ad **OR2**, i risultati per persone normovedenti hanno evidenziato che le interfacce riescono ad essere visualizzate correttamente in quasi tutte le applicazioni, Figura 5.6, la media di problemi per questa tipologia di partecipanti 35% ed è dovuta principalmente al testo che non viene visualizzato correttamente, producendo delle frasi o numeri che i partecipanti non sono riusciti a leggere subito oppure sono riusciti a leggere aumentando la dimensione dell’immagine.

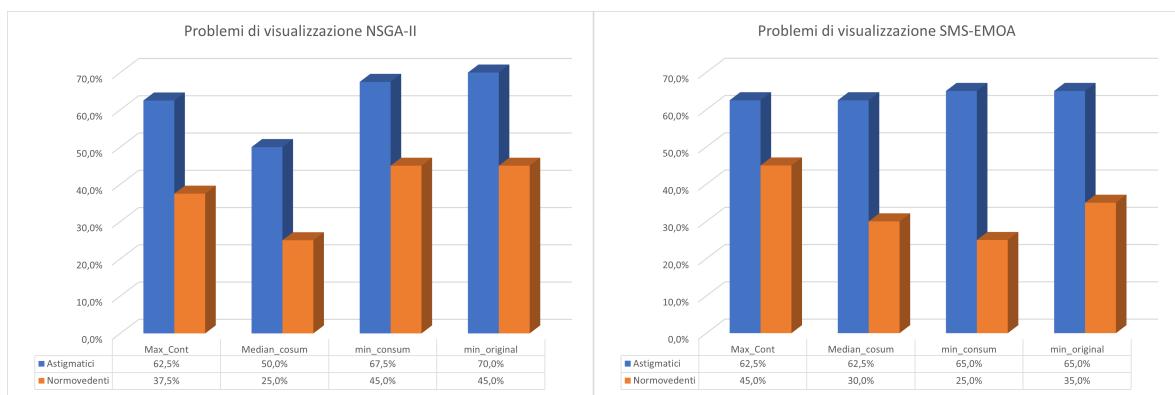


Figura 5.6: Percentuale media di problemi di visualizzazione delle interfacce prodotte.

Invece per quanto riguarda **OR3**, per le persone astigmatiche la quantità di problemi è aumentata, con una media di problemi di visualizzazione del 63,1% dovuta principalmente al testo e alla relativa bassa quantità di contrasto e in una minima parte ai pulsanti che non sono stati riconosciuti correttamente.

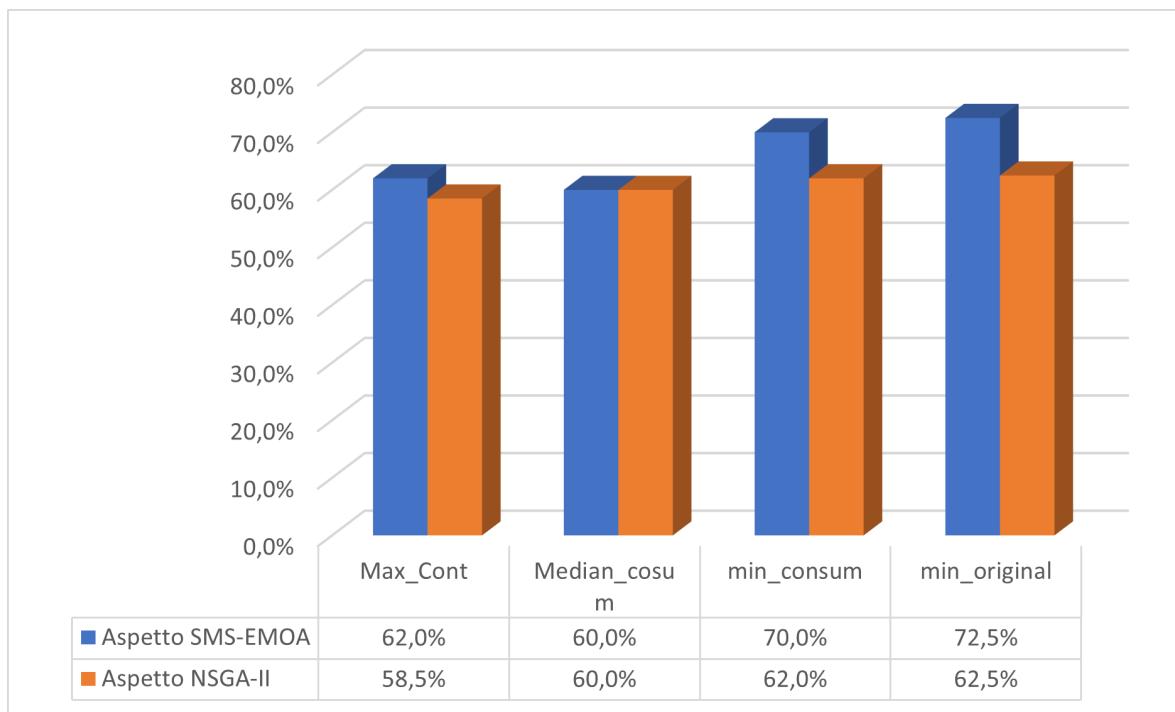


Figura 5.7: Percentuale di fascino delle interfacce

Confrontando invece i problemi delle interfacce prodotte dalle due tipologie di algoritmi **OR4**, la media per normovedenti di SMS-EMOA è leggermente migliore (media=33.8%) rispetto a NSGA-II (media=35.9%) invece per le persone astigmatiche questi dati cambiano, SMS-EMOA 65% e NSGA-II 63.1%. Considerando che il principale obiettivo è ridurre la quantità di energia consumata dalle interfacce grafiche le interfacce prodotte SMS-EMOA riescono ad avere colorazioni con un miglior contrasto per persone normovedenti invece NSGA-II solo nelle interfacce della tipologia riguardante la mediana del consumo.

Dal punto di vista estetico invece, pur riscontrando un gran numero di problemi legati alla visualizzazione del testo in Figura 5.7 si possono notare le percentuali per ogni tipologia di interfaccia prodotte dai due algoritmi. L'algoritmo SMS-EMOA produce interfacce più gradevoli alla vista degli utenti, anche con le interfacce che riescono a far risparmiare una quantità maggiore di energia. Dal post-esperimento invece esattamente la metà delle persone preferirebbe le interfacce ottimizzate rispetto al design originale e dai feedback generali lasciati dai partecipanti l'unica problematica è il testo che non riescono a visualizzare correttamente.

CAPITOLO 6

Limitazioni dello Sviluppo

Nello sviluppo di questo sistema sono state diverse le problematiche e le limitazioni riscontrate per poter riuscire nella realizzazione di questo progetto e sono legate sia all'hardware che al software. Per il calcolo del consumo energetico delle nuove interfacce grafiche, è stata utilizzata l'equazione 3.2.1 poichè, non disponendo di un dispositivo fisico Android non è stato possibile analizzare il reale consumo energetico di questa tipologia di schermo. A causa poi della differenza tra il consumo stimato lato software e l'effettivo consumo delle interfacce installate in un dispositivo, sarebbe stato ottimale utilizzare uno strumento che permettesse di ottenere i dati relativi ai consumi energetici che fossero il più attendibili possibili. Un esempio è lo studio di D.Nucci et al [2] nel quale, come citato nel capitolo precedente, spiega le differenze tra le acquisizioni software e hardware, con la possibilità di avere una maggior precisione nell'acquisizione dei consumi tramite dispositivi hardware. Le applicazioni utilizzate per questo studio sono le stesse utilizzate nello studio originale, con la differenza che le applicazioni che venivano usate nel 2015 sono state aggiornate e, di conseguenza, hanno cambiato apparenza e contenuti. Questi cambiamenti hanno fatto sì che sorgessero delle problematiche legate alla gerarchia delle interfacce e ai colori utilizzati nello sfondo. Nel colorare le interfacce con lo stesso colore ma con sfumature differenti ha portato a problemi come nella Figura 6.1 dove si possono notare quanto incidano queste differenze quando verranno ricostruite le nuove interfacce.

Queste problematiche sono state riscontrate nelle applicazioni Diabete Plus, Board Game Geek, The Weather Channel e Tik Tak Toe, nelle quali gli sviluppatori per gestire i colori delle

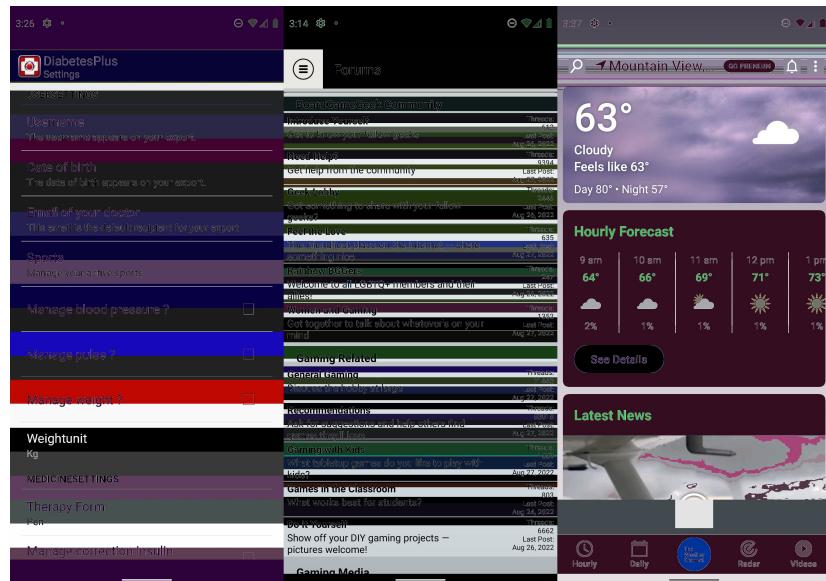


Figura 6.1: Limitazioni colori interfacce per le applicazioni Diabete Plus, Board Game Geek e The Weather Channel.

interfacce hanno optato di utilizzare lo stesso colore ma con piccolissime variazioni nel formato RGB, impercettibili a livello umano come nella Figura 6.2 nella quale le variazioni delle colorazioni sono impossibili da riconoscere ma, posizionandoci sulle diverse coordinate (x,y) nell'immagine viene evidenziato che il colore differisce in base alla verticalità dell'immagine. Di conseguenza ci saranno problemi nella realizzazione delle nuove interfacce quando verrà utilizzato l'algoritmo che permette di ricostruire le immagini, a causa del valore del vincolo dell'algoritmo e della seconda fitness function che richiedono un minimo valore di contrasto tra i colori e il fatto che vada massimizzato il contrasto tra le componenti. Per risolvere questo problema si dovrebbe vedere se aumentando il numero di valutazioni o generazioni dell'algoritmo genetico multi-obiettivo si riesce a generare un'insieme di soluzioni che riescono ad evitare questo problema, oppure realizzare una nuova soluzione che permetta di ottenere una misura di somiglianza e decidere quale colore della soluzione inserire per tutti i colori con una determinata soglia di somiglianza. Aumentando però il numero di valutazioni o generazioni aumenterebbe di conseguenza il tempo richiesto per poter ottenere una soluzione e non è neanche detto che si riesca ad ottenere sicuramente una soluzione, considerando soprattutto gli obiettivi e i vincoli che deve soddisfare l'algoritmo. Provando con un numero di valutazioni uguale a 10000 comunque questo problema non è stato risolto. Invece, considerando come soluzione l'individuazione di una soglia di somiglianza si deve stare attenti a non creare insiemi che incidono sui colori delle interfacce poiché si potrebbero creare interfacce che non hanno contrasto, che non riducono la quantità di energia emessa o

realizzare soluzioni che alterano i colori ottenuti come soluzione.

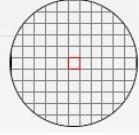
Descrivendo invece il lato software, nel documento di M. Linares-Vásquez et al. [10] viene descritto l'utilizzo del formato TYPE_INT_ARGB ma tutti i calcoli e le formule utilizzati per lo studio vengono effettuati sempre attraverso il formato RGB e per questo è stato adottato per tutto lo studio il formato RGB. Nell'algoritmo BOCP_BOCC 1 è stato necessario utilizzare una struttura dati che mantenesse traccia di tutti i pixel già utilizzati in precedenza, poichè prendere diverse volte lo stesso pixel nell'immagine non ha senso, aumenterebbe la dimensione degli HashMap e il tempo richiesto per poter acquisire le interfacce, soprattutto associandolo a diverse componenti o colori alterando di conseguenza le acquisizioni dei dati dagli screenshot e dagli snapshot. Questa integrazione, quindi, è stata necessaria sia per non alterare gli HashMap che verranno prodotti in output ma anche per ridurre il calcolo richiesto per l'acquisizione dei colori dei pixel e delle componenti nelle schermate. Ulteriormente è stato necessario ottimizzare la prima e la seconda fitness function a causa della quantità di calcolo e tempo richiesto per l'esecuzione rispetto alla descrizione nello pseudo-codice degli algoritmi originali. Per ottimizzare la prima fitness function è stato inserito in BOCP_BOCC una variabile h che permette di identificare una interfaccia, questa decisione è stata presa per non avere problemi di inconsistenza nell'assegnare i valori ad ogni interfaccia h . Ogni volta che durante l'esecuzione del sistema avviene l'uso delle interfacce, i nomi delle interfacce verranno ordinati in ordine crescente, così il valore di h corrisponderà sempre alla stessa interfaccia evitando problemi. Queste modifiche riusciranno a far analizzare e calcolare correttamente le tre fitness function e il valore del vincolo durante la fase di sviluppo dell'algoritmo genetico multi-obiettivo. Con questa scelta progettuale si riuscirà a svolgere con una velocità nettamente superiore il valore della prima fitness function. Con un miglioramento di oltre il 90% sui tempi di esecuzione. Nella seconda fitness function, invece, poichè per le stesse problematiche di calcolo e tempo descritto in precedenza, è stata ridotta la taglia dell'HashMap BOCC, una volta che viene calcolato questo HashMap verrà usato un ulteriore HashMap che servirà solo per evitare che una determinata componente di una determinata interfaccia non ricompaia due volte in BOCC. Per ogni colore in BOCC viene controllata se una componente (c, h) è stata inserita nell'HashMap momentaneo $hash_mom$, se non è stato ancora inserito, verrà inserito nel nuovo BOCC con la stessa chiave in cui è stato prelevato, altrimenti viene scartato. Questa soluzione riesce a ridurre drasticamente la quantità di calcolo richiesta e riuscirà a preservare la funzione $BCIndexWithH$ poichè questa funzione ritorna il primo indice in cui si trova la componente scelta, di conseguenza verrà preso il primo valore con le caratteristiche scelte senza nessun danno all'acquisizione delle componenti ritornando esattamente

Date of birth
The date of birth appears on your export.

Email of your doctor
This email is the default recipient for your export

Sports
Manage your active sports

Manage blood pressure ?

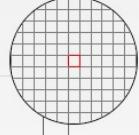


Date of birth
The date of birth appears on your export.

Email of your doctor
This email is the default recipient for your export

Sports
Manage your active sports

Manage blood pressure ?

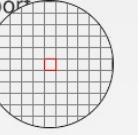


Date of birth
The date of birth appears on your export.

Email of your doctor
This email is the default recipient for your export

Sports
Manage your active sports

Manage blood pressure ?

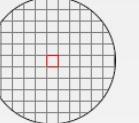


Date of birth
The date of birth appears on your export.

Email of your doctor
This email is the default recipient for your export

Sports
Manage your active sports

Manage blood pressure ?



Colors
Show more

●	○	●
HEX: #f3f3f3	<input type="button" value="Copy"/>	
RGB: rgba(243,243,243,255)	<input type="button" value="Copy"/>	

Colors
Show more

●	○	●
HEX: #f2f2f2	<input type="button" value="Copy"/>	
RGB: rgba(242,242,242,255)	<input type="button" value="Copy"/>	

Use Your Image
Show more

●	○	●
HEX: #f1f1f1	<input type="button" value="Copy"/>	
RGB: rgba(241,241,241,255)	<input type="button" value="Copy"/>	

Use Your Image
Show more

●	○	●
HEX: #f0f0f0	<input type="button" value="Copy"/>	
RGB: rgba(240,240,240,255)	<input type="button" value="Copy"/>	

Figura 6.2: Variazione delle colorazioni RGB delle GUI.

l'indice in cui si trova. Il calcolo del valore del vincolo invece è risultato problematico, ci sono diverse variabili che entrano in gioco: le scelte progettuali degli sviluppatori per costruire l'interfaccia grafica, lo snapshot ottenuto da UIAutomatorView che può essere cambiato nel tempo e l'errato calcolo per poter valutare ogni volta che esiste una condizione di vincolo. A causa di una di queste problematiche, se dovessimo seguire lo sviluppo del documento originale non si riuscirebbe ad ottenere un valore di vincolo uguale a 0, per poter risolvere questo problema è stata inserita una ulteriore condizione, oltre al valore di contrasto minore di una soglia CnTh, nella quale se una componente e un suo diretto figlio/genitore ha lo stesso indice in BOCC allora non aumenterà il conteggio del valore del vincolo. Questa scelta progettuale sembra funzionare e si riescono ad ottenere comunque interfacce ottimizzate sui consumi energetici, riuscendo a ridurre la quantità di energia prodotta dagli schermi degli smartphone con un adeguato contrasto tra le componenti.

Quando si parla poi di processi decisionali e di software che permettono di svolgere compiti legati all'intelligenza artificiale si sa che si va incontro a tempi di apprendimento che richiedono diverse ore, in questo contesto di sviluppo l'esecuzione di algoritmi genetici multi-obiettivo richiede una grande quantità di risorse di calcolo e di tempi per poter riuscire ad allenare il sistema e produrre risultati ottimali. Il sistema pur riuscendo ad essere eseguito su diverse applicazioni impiega una grande quantità di tempo e per questo il numero di valutazioni per ogni applicazione rispetto allo studio iniziale è stato ridotto di 1/10, impiegando per NSGA-II una media per ogni applicazione di circa 2 ore e 49 minuti e per SMS-EMOA di 2 ore e 51 minuti. La grande quantità di tempo può essere additata sia al linguaggio Python che permette di scrivere una quantità inferiore di codice rispetto al linguaggio Java ma ha tempi di esecuzione maggiori sia al framework Pymoo utilizzato, poichè il framework è in una versione non ancora completa (ovvero, 0.6) e probabilmente ancora non ottimizzata correttamente che al calcolo dell'algoritmo stesso in base agli iperparametri inseriti in input. La necessità di usare i server dipartimentali dell'Università degli Studi di Fisciano (SA) è stata fondamentale per poter eseguire tutte le applicazioni prese in considerazioni in questo studio a causa del tempo richiesto dall'algoritmo genetico multi-obiettivo, per poter eseguire tutte le applicazioni (in totale 11) ci vogliono quasi 26 ore, una quantità di tempo e calcolo onerosa considerando le ottimizzazioni sulle fitness function, la riduzione della taglia di BOCC, la diminuzione del numero di valutazione e soprattutto utilizzando la programmazione multithread per parallelizzare il calcolo degli individui su diversi thread.

CAPITOLO 7

Conclusioni e Lavori Futuri

Questa tesi di ricerca ha fornito un approccio differente per l'ottimizzazione delle interfacce grafiche dei dispositivi Android per riuscire a migliorare i consumi energetici e al contempo preservare il contrasto tra le componenti per riuscire a distinguere correttamente gli elementi all'interno delle interfacce. Il sistema realizzato riesce a ridurre il consumo energetico delle interfacce grafiche e al contempo sono stati considerati anche problemi relativi all'accessibilità riguardanti la visione delle componenti per riconoscere all'interno delle interfacce grafiche elementi come pulsanti, testo, immagini e sfondo. Per sviluppi futuri verranno proposte differenti modalità di acquisizione dei consumi dei dispositivi per avere una precisione maggiore sui dati, si darà maggiore attenzione al testo all'interno delle interfacce poichè è emerso che la maggior parte degli utenti ha avuto problemi a distinguere il colore del testo rispetto alle componenti adiacenti. Per questo problema infatti si decideranno nuove modalità per calcolare il vincolo sul testo per gestire maggiormente il contrasto tra testo e sfondo. Un ulteriore lavoro riguarderà gestire le leggere variazioni di colore RGB che sono impercettibili ad occhio umano ma che incideranno sull'esecuzione dell'algoritmo realizzato, non riuscendo a realizzare correttamente delle interfacce ottimizzate. Infine, verranno verificati diversi iper-parametri per verificare se con differenti valori verranno riprodotte interfacce che riescono a migliorare i risultati ottenuti in questa tesi.

Bibliografia

- [1] P. K. D. Pramanik, N. Sinhababu, B. Mukherjee, S. Padmanaban, A. Maity, B. K. Upadhyaya, J. B. Holm-Nielsen, and P. Choudhury, "Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage," *IEEE Access*, vol. 7, pp. 182113–182172, 2019. (Citato alle pagine iv, 2, 7 e 8)
- [2] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Software-based energy profiling of android apps: Simple, efficient and reliable?," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 103–114, 2017. (Citato alle pagine iv, 7, 8, 9 e 45)
- [3] G. P. Perrucci, F. H. P. Fitzek, and J. Widmer, "Survey on energy consumption entities on the smartphone platform," in *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*, pp. 1–6, 2011. (Citato a pagina 2)
- [4] Y. Liu, C. Xu, and S. Cheung, "Where has my battery gone? finding sensor related energy black holes in smartphone applications," in *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 2–10, 2013. (Citato alle pagine 2 e 7)
- [5] M. Wan, Y. Jin, D. Li, and W. G. J. Halfond, "Detecting display energy hotspots in android apps," in *8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015*, pp. 1–10, IEEE, 2015. (Citato alle pagine 2 e 9)
- [6] X. Chen, Y. Chen, Z. Ma, and F. C. A. Fernandes, "How is energy consumed in smartphone display applications?," in *Proceedings of the 14th Workshop on Mobile Computing Systems*

- and Applications*, HotMobile '13, (New York, NY, USA), Association for Computing Machinery, 2013. (Citato alle pagine 2, 9, 12 e 13)
- [7] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating android applications' cpu energy usage via bytecode profiling," in *Proceedings of the First International Workshop on Green and Sustainable Software*, GREENS '12, p. 1–7, IEEE Press, 2012. (Citato a pagina 2)
- [8] K. Ishihara, S. Ishihara, M. Nagamachi, S. Hiramatsu, and H. Osaki, "Age-related decline in color perception and difficulties with daily activities—measurement, questionnaire, optical and computer-graphics simulation studies," *International Journal of Industrial Ergonomics*, vol. 28, no. 3-4, pp. 153–163, 2001. (Citato a pagina 3)
- [9] C. Kaufman-Scarborough, "Accessible advertising for visually-disabled persons: the case of color-deficient consumers," *Journal of Consumer Marketing*, 2001. (Citato a pagina 3)
- [10] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Po-shyvanyk, "Multi-objective optimization of energy consumption of guis in android apps," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, sep 2018. (Citato alle pagine 4, 7, 9, 10, 11, 12, 13, 15 e 47)
- [11] F. Palomba, D. Di Nucci, A. Panichella, A. Zaidman, and A. Lucia, "On the impact of code smells on the energy consumption of mobile applications," *Information and Software Technology*, vol. 105, 09 2018. (Citato a pagina 8)
- [12] M. Dong and L. Zhong, "Chameleon: A color-adaptive web browser for mobile oled displays," *IEEE Transactions on Mobile Computing*, vol. 11, pp. 724–738, 2012. (Citato a pagina 9)
- [13] D. Li, A. H. Tran, and W. G. J. Halfond, "Making web applications more energy efficient for oled smartphones," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, (New York, NY, USA), p. 527–538, Association for Computing Machinery, 2014. (Citato alle pagine 9 e 19)
- [14] M. Dong and L. Zhong, "Power modeling and optimization for oled displays," *IEEE Transactions on Mobile Computing*, vol. 11, pp. 1587–1599, 2012. (Citato alle pagine 9, 12 e 13)
- [15] M. Varvello, K. Katevas, M. Plesa, H. Haddadi, and B. Livshits, "High-accuracy energy measurements on mobile devices," (Citato a pagina 10)

- [16] J. W. Tukey, "Exploratory data analysis," 1977. (Citato a pagina 12)
- [17] B. Charlesworth and D. Charlesworth, "Darwin and genetics," *Genetics*, vol. 183, no. 3, pp. 757–766, 2009. (Citato a pagina 13)
- [18] S. Forrest, "Genetic algorithms," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 77–80, 1996. (Citato a pagina 15)
- [19] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–73, 1992. (Citato a pagina 15)
- [20] O. Kramer, "Genetic algorithms," in *Genetic algorithm essentials*, pp. 11–19, Springer, 2017. (Citato a pagina 15)
- [21] Y. Gao, L. Shi, and P. Yao, "Study on multi-objective genetic algorithm," in *Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No.00EX393)*, vol. 1, pp. 646–650 vol.1, 2000. (Citato a pagina 15)
- [22] H. R. Cheshmehgaz, H. Haron, and M. R. Maybodi, "Cellular-based population to enhance genetic algorithm for assignment problems," *Am J Intell Syst*, vol. 1, no. 1, pp. 1–5, 2011. (Citato a pagina 15)
- [23] I. Nouiri, "Multi-objective tool to optimize the water resources management using genetic algorithm and the pareto optimality concept," *Water resources management*, vol. 28, no. 10, pp. 2885–2901, 2014. (Citato a pagina 15)
- [24] Y. Wang, H. Cheng, C. Wang, Z. Hu, L. Yao, Z. Ma, and Z. Zhu, "Pareto optimality-based multi-objective transmission planning considering transmission congestion," *Electric Power Systems Research*, vol. 78, no. 9, pp. 1619–1626, 2008. (Citato a pagina 15)
- [25] M. Wan, Y. Jin, D. Li, J. Gui, S. Mahajan, and W. G. Halfond, "Detecting display energy hotspots in android apps," *Software Testing, Verification and Reliability*, vol. 27, no. 6, p. e1635, 2017. (Citato a pagina 19)
- [26] N. Patil, D. Bhole, and P. Shete, "Enhanced ui automator viewer with improved android accessibility evaluation features," in *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)*, pp. 977–983, 2016. (Citato a pagina 20)
- [27] W3C, "G18: Ensuring that a contrast ratio exists between text (and images of text) and background behind the text." <https://www.w3.org/TR/WCAG20-TECHS/G18.html>. (Citato a pagina 23)

- [28] G. Sharma and R. Bala, *Digital color imaging handbook*. CRC press, 2017. (Citato a pagina 27)
- [29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002. (Citato a pagina 27)
- [30] F. Chicano, A. M. Sutton, L. D. Whitley, and E. Alba, "Fitness probability distribution of bit-flip mutation," *Evolutionary Computation*, vol. 23, no. 2, pp. 217–248, 2015. (Citato a pagina 29)
- [31] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014. (Citato a pagina 33)
- [32] N. Beume, B. Naujoks, and M. Emmerich, "Sms-emoa: Multiobjective selection based on dominated hypervolume," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, 2007. (Citato a pagina 33)
- [33] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020. (Citato a pagina 35)