UNIVERSITÀ DEGLI STUDI DI PISA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA INFORMATICA

Tesi di Laurea

Implementazione di algoritmi genetici multiobiettivo distribuiti in ambiente Matlab®

Candidato: Alessandro Simi

Relatori: Prof. Beatrice Lazzerini

Prof. Francesco Marcelloni

Ing. Marco Cococcioni

Anno Accademico 2007/2008

Indice

Indice ... 3
Sommario ... 5
Capitolo 1 Introduzione ... 7
Capitolo 2 Gli algoritmi genetici ... 9
2.1 Il funzionamento ... 9
2.2 Il cromosoma ... 10
2.2.1 La codifica ... 10
2.2.2 Crossover ... 11
2.2.3 Mutazione ... 12
2.3 L'evoluzione ... 13
2.3.1 Fitness ... 13
2.3.2 Selezione ... 13
2.3.3 Rimpiazzamento ... 14
Capitolo 3 Algoritmi genetici multiobiettivo ... 15
3.1 Problemi multiobiettivo ... 15
3.1.1 Due approcci alla risoluzione ... 16
3.1.2 Algoritmi evoluzionistici ... 17
3.2 Soluzioni ottime ... 18
3.2.1 Il concetto di dominanza ... 19
3.2.2 L'insieme ottimo Pareto ... 20
3.3 Algoritmi ... 20
3.3.1 Metodo base ... 21
3.3.2 Metodo degli aggiornamenti continui ... 21
3.3.3 Metodo efficiente di Kung ... 22
3.4 Non-dominated sorting genetic algorithm II ... 23
3.4.1 Ordinamento a fronti non-dominati ... 23
3.4.2 Preservare la diversità ... 23
3.4.3 Ranking per la selezione ... 24
Capitolo 4 MOGA paralleli ... 26
4.1 Architetture di calcolo ... 26
4.1.1 Cluster di computer ... 27
4.2 La struttura dell'algoritmo ... 27
4.2.1 Calcolo della fitness ... 28
4.2.2 Ranking ... 28
4.2.3 Selezione ... 29
4.2.4 Ricombinazione ... 29
4.3 La parallelizzazione ... 29
4.3.1 Dividere la popolazione ... 29
Capitolo 5 Cono di separazione ... 31
5.1 Il funzionamento ... 31
5.1.1 La struttura principale ... 32
5.1.2 EVOPs ... 33
5.1.3 Gli estremi ... 33
5.1.4 La migrazione ... 33
5.2 Ranking ... 34
5.2.1 La dominanza ... 34

5.2.2 Il fronte ottimo ... 35
5.2.3 La distribuzione delle soluzioni ... 37
5.3 L'evoluzione dell'algoritmo ... 38
5.3.1 La popolazione dei processori ... 38
Capitolo 6 Microcono di separazione ... 40
6.1 Il funzionamento ... 40
6.1.1 La migrazione ... 41
6.2 Ranking ... 42
6.2.1 La distribuzione ... 42
6.2.2 La frammentazione ... 43
Capitolo 7 Ranking separation ... 45
7.1 La migrazione ... 45
7.2 I vincoli ... 46
7.3 Ranking ... 46
7.3.1 Microcone assignment ... 46
7.4 L'implementazione ... 48
7.4.1 La struttura principale ... 48
7.4.2 Microcone assignment ... 48
7.4.3 Migrazione casuale ... 53
7.5 Osservazioni ... 54
Capitolo 8 Separazione casuale ... 55
8.1 Modello a isole ... 55
8.1.1 Isole e migrazione ... 55
8.1.2 Problemi multiobiettivo ... 55
8.2 Random Separation ... 56
8.2.1 Il suo utilizzo ... 57
Capitolo 9 Confronto tra gli algoritmi distribuiti ... 58
9.1 Metriche ... 58
9.1.1 Distanza dal fronte ottimo ... 58
9.1.2 Distribuzione delle soluzioni ... 59
9.1.3 Fronte ottimo di Pareto ... 60
9.1.4 Migrazione ... 60
9.2 Problemi di test ... 60
9.2.1 Schaffer (SCH) ... 60
9.2.2 Fonseca e Fleming (FON) ... 63
9.2.3 Kursawe (KUR) ... 66
9.2.4 Zitzler, Deb e Thiele (ZDT) ... 67
9.3 Sintesi dei risultati ... 69
9.3.1 Cono di separazione ... 69
9.3.2 Microcono di separazione ... 70
9.3.3 Ranking separation ... 70
9.3.4 Separazione casuale ... 70
Capitolo 10 Conclusioni ... 71
Appendice A Crowding distance assigment ... 73
Appendice B Algoritmi genetici multiobiettivo con vincoli ... 74
B.1 Trattare i vincoli ... 74
B.1.1 Ignorare le soluzioni ... 74
B.1.2 Penalty function ... 74
B.1.3 Jiménez Verdegay Goméz-Skameta ... 75
B.1.4 Constrained Tournament ... 75
B.1.5 Ray Tai Seow ... 75
B.2 Considerazioni ... 75
Riferimenti bibliografici ... 77
Indice analitico ... 79

Sommario

Parallelizzare gli algoritmi genetici multiobiettivo è un sistema molto efficacie per aumentare la potenza di calcolo e rimediare all'elevata complessità di questi strumenti. Il calcolo delle soluzioni non-dominate rappresenta però un ostacolo in quanto l'operazione deve disporre dell'intera popolazione. Per risolvere questo inconveniente è stato proposto nel 2004 un algoritmo chiamato *cone separation* [2], che sfrutta alcune proprietà del fronte di Pareto per poter distribuire il calcolo della non-dominanza tra più processori. In questa tesi viene riesaminata questa tecnica in modo approfondito, evidenziando pregi e difetti. Dall'analisi di quest'ultimi, vengono proposte una serie di varianti per migliorare la ricerca del fronte di Pareto e successivamente, vengono riassunte in un unico algoritmo chiamato *ranking separation*. Le simulazioni presentate nell'ultima parte della tesi mostrano che entrambi gli algoritmi sono capaci di trovare le soluzioni ottime del problema. Questi risultati sono ottenuti però con un diverso utilizzo delle risorse, in particolare, con un differente numero di dati trasmetti tra i processori. Sotto questo aspetto l'algoritmo *ranking separation* dimostra di funzionare correttamente con un numero limitato e controllato di individui che migrano.

Capitolo 1
Introduzione

La complessità degli algoritmi genetici multiobiettivo è sempre stata considerata il loro principale punto critico. L'enorme quantità di dati che utilizzano e l'elevata potenza di calcolo richiesta hanno spinto lo sviluppo di algoritmi verso soluzioni sempre più performanti ed efficienti. Tuttavia, tutti questi sforzi non hanno mai preso in considerazione la parallelizzazione come un valido strumento per migliorare le prestazioni dell'algoritmo. Questo aspetto risulta ancora più strano se consideriamo le numerose tecniche di parallelizzazione presenti nel campo degli algoritmi a singolo obiettivo.

Un possibile motivo di questo disinteresse è causato dal calcolo delle soluzioni non-dominate, elemento fondamentale per tutti gli algoritmi genetici multiobiettivo. Questa operazione ostacola la parallelizzazione perché utilizza tutti gli individui della popolazione, infatti, per capire se una soluzione è dominata deve essere confrontata con tutte le altre. In questo modo però non è possibile dividere la popolazione in parti e distribuire il calcolo tra i processori.

Nel 2004 Kalyanmoy Deb, professore del IITK (*Indian Institute of Technology Kanpur*), ha pubblicato un articolo dove viene illustrata una tecnica per parallelizare gli algoritmi genetici multiobiettivo. Questa tecnica chiamata *cone separation* mostra un interessante approccio alla distribuzione del calcolo basata sulla divisione della popolazione. In poche parole Deb suddivide lo spazio degli obiettivi in aree a forma di coni e associa alla popolazione di ogni zona un diverso processore. Così facendo molte operazioni dell'algoritmo avvengono in parallelo e si ottiene un considerevole aumento delle prestazioni.

L'algoritmo *cone separation* individua nella forma del fronte di Pareto delle caratteristiche che permettano di distribuire il calcolo della non-dominanza. Su queste caratteristiche si basa la divisione dello spazio degli obiettivi in coni e la parallelizzazione dei calcoli. Inoltre l'algoritmo utilizza il meccanismo della migrazione per garantire che gli individui di un processore rimangano all'interno della propria area.

In questa tesi vengono analizzati a fondo tutti gli aspetti che riguardano l'algoritmo *cone separation*, evidenziando sia i pregi che i difetti. Partendo da quest'ultimi viene proposta una variante chiamata *microcone separation* che migliora la ricerca delle soluzioni ottime in presenza di un fronte di Pareto discontinuo. Questa variante e l'originale sono caratterizzate però dello stesso difetto: un eccessiva comunicazione tra i processori causata della migrazione degli individui. Per risolvere questo inconveniente viene proposto un ulteriore algoritmo chiamato *ranking separation*.

Questo algoritmo è profondamente diverso dai precedenti perché utilizza l'operatore di ranking al posto della migrazione e considera lo spazio degli obiettivi diviso in aree ammissibili (dentro i coni) e non ammissibili (fuori dai coni). Ispirandosi ad una tecnica con vincoli *soft constrained* gli individui sono ordinati in base alla distanza da queste aree, cercando di privilegiare le soluzioni ammissibili. In conclusione l'operatore di ranking viene modificato per classificare le soluzioni secondo la vicinanza del fronte, dei coni e seguendo la distribuzione della popolazione.

Lo scambio degli individui tra i processori con l'algoritmo *ranking separation* è praticamente nullo. Da una parte questo rappresenta un vantaggio perché diminuiscono i tempi di trasmissione, dall'altra viene meno un possibile effetto di cooperazione tra i processori. Per rimediare a questa mancanza viene reintrodotto l'operatore di migrazione

per condividere i risultati raggiunti da ogni processore e rendere più veloce la ricerca del fronte.

La tesi è organizzata in dieci capitoli. Nei primi quattro capitoli vengono introdotti i concetti base degli algoritmi genetici classici, multiobiettivo e distribuiti. Nel capitolo 5 si analizza l'algoritmo *cone separation* e nel successivo viene presentata una sua prima variante chiamata *microcone separation*. Nel capitolo 7 sono riassunte tutte le considerazioni fatte precedentemente per creare uno nuovo algoritmo chiamato *ranking separation*, mentre nel capitolo 8 viene approfondito il concetto del modello a isole riferito agli algoritmi genetici. Infine negl'ultimi due capitoli sono riportati i risultati delle simulazioni e le conclusioni della tesi.

Capitolo 2
Gli algoritmi genetici

Gli algoritmi genetici sono strumenti per risolvere problemi di ricerca e di ottimizzazione che seguono un procedimento euristico ispirato alla genetica e al principio della selezione naturale di Charles Darwin.

Gli algoritmi genetici utilizzano un insieme di soluzioni che si evolve in intervalli di tempo chiamati generazioni. L'evoluzione è guidata dalla ricerca della soluzione ottima per mezzo del confronto, infatti, ad ogni soluzione dell'insieme o individuo della popolazione viene associato un valore chiamato *fitness* che indica la qualità rispetto al problema. Per mezzo di questo valore si possono confrontare e selezionare gli individui migliori della generazione attuale e utilizzarli per creare la popolazione di quella successiva.

Con questo procedimento, ispirato al principio della selezione naturale, si cerca di ricavare un insieme di individui sempre migliore che, con l'avanzare delle generazioni, contenga la soluzione ottima del problema o una sua approssimazione.

Spesso ci si riferisce agli individui con il nome di cromosomi per via della loro struttura e delle operazioni con cui si possono modificare. Infatti ogni soluzione viene descritta da un insieme di caratteristiche del tutto simili ai geni, sulle quali vengono utilizzati gli stessi operatori di mutazione e crossover presenti nella genetica per creare nuove soluzioni.

2.1 Il funzionamento

La struttura base di un algoritmo genetico prevede un funzionamento ciclico che simula il processo evolutivo di una popolazione. Ogni ciclo rappresenta una generazione e al suo interno vengono svolte le operazioni per generare una nuova popolazione formata da individui sempre migliori.

Il primo passo di questo algoritmo è creare una popolazione casuale dalla quale partire (figura 2.1a). Mentre le fasi successive si ripetono ad ogni generazione e sono associate al principio della selezione naturale (figura 2.1b) o alla genetica (figura 2.1c).

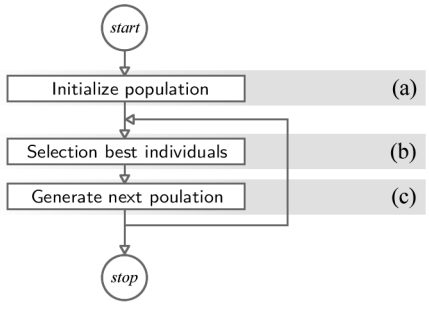


Figura 2.1 – Struttura generale di un algoritmo genetico.

In particolare le prime fasi si occupano di selezionare gli individui migliori della popolazione, mentre le altre di generare nuovi individui. Quest'ultime operazioni sono possibili unendo o modificando le caratteristiche che identificano un individuo.

Dal punto di vista della genetica, i nuovi cromosomi si ottengono ricombinando il loro patrimonio genetico ovvero andando a modificare i geni con gli operatori di mutazione ed di crossover.

Per ogni combinazione di geni è possibile calcolare un valore chiamato *fitness* che indica la capacità con cui il cromosoma o la soluzione sia in grado di risolvere il problema. Nella selezione naturale questo valore misura l'adattamento dell'individuo nell'ambiente. Per cui una fitness migliore è legata ad una maggiore probabilità di sopravvivenza, mentre all'interno dell'algoritmo genetico ad una maggiore probabilità di selezione.

2.2 Il cromosoma

I cromosomi sono alla base del funzionamento dell'algoritmo genetico e rappresentano le possibili soluzioni del problema. I dati in ingresso al problema vengono codificati in geni in modo da poter utilizzare gli operatori di crossover e mutazione per ricombinare il loro patrimonio genetico e generare nuove soluzioni. In generale, il funzionamento di questi operatori dipende dal tipo di codifica che viene utilizzata, per cui è importante scegliere in quale modo viene rappresentato il problema.

Per meglio capire il funzionamento degli algoritmi genetici facciamo riferimento ad problema d'esempio. Abbiamo due punti A e B su una superficie e vogliamo trovare il percorso più breve che collega i due punti percorrendo dalla superficie (figura 2.2, linea tratteggiata).

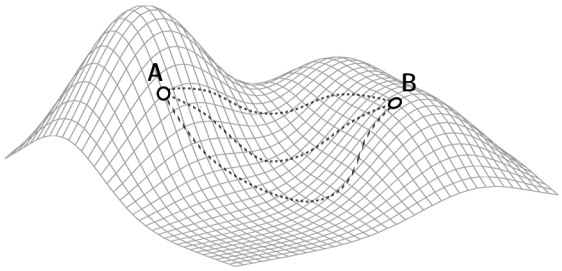


Figura 2.2 – Esempio di problema risolvibile utilizzando gli algoritmi genetici.

Chiaramente esistono infiniti percorsi che uniscono i due punti per cui ci limitiamo a considerare solo quelli che si ottengono intersecando la superficie con i piani passanti per i punti A e B (figura 2.3).

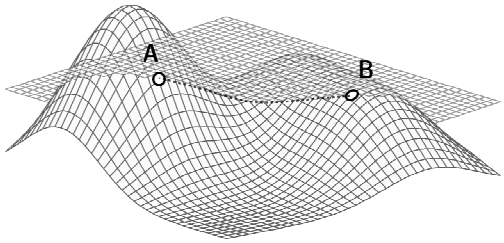


Figura 2.3 – Percorso tra i due punti ottenuto con un piano di intersezione.

In questo modo è possibile associare ad ogni piano e al percorso un valore φ che indica l'angolo del piano. Tutti i possibili angoli assunti costituiscono l'insieme delle soluzioni del problema. Lo scopo dell'algoritmo genetico è quello di trovare l'angolo φ per cui la distanza tra i due punti è minima.

2.2.1 La codifica

Le soluzioni del problema si trovano all'interno dello spazio delle soluzioni. Un punto in questo spazio costituisce un cromosoma e in genere si rappresenta con una stringa. La stringa può essere composta da valori binari oppure da valori reali: nel primo caso la codifica prende il nome di binaria, mentre nel secondo caso reale.

Nelle esempio mostrato in figura (figura 2.2) viene proposto un problema con uno spazio delle soluzioni costituito da un intervallo dei numeri reali. Una possibile rappresentazione utilizzando una codifica binaria si ha considerando il valore dell'angolo espresso in base 2, mentre con la codifica reale si ottiene con il valore stesso (figura 2.4).

Figura 2.4 – Codifica binaria e reale.

Il problema del percorso minimo ha uno spazio delle soluzioni ad una sola dimensione, per questo il cromosoma con codifica reale non è un vettore, ma uno scalare. In generale gli algoritmi genetici possono risolvere problemi a più dimensioni per cui le soluzioni sono stringe con più componenti.

Si nota inoltre che la posizione delle componenti del cromosoma non è casuale, ma è determinata dalla codifica, per cui assume un preciso significato.

2.2.2 Crossover

I nuovi cromosomi vengono generati da altri cromosomi trasmettendo il loro patrimonio genetico in modo che le nuove soluzioni siano simili ai genitori. Questo aspetto è fondamentale per la convergenza dell'algoritmo, infatti, ogni nuova popolazione viene costruita con gli individui migliori della precedente per propagare nelle generazioni il patrimonio genetico migliore. Per cui è chiaro che, se i figli non fossero simili ai genitori (nessun patrimonio genetico in comune), non avremo alcuna garanzia sulla convergenza.

Il crossover o crossing-over si occupa proprio di questo aspetto, ricombinare il materiale genetico di due o più cromosomi e viene descritto come lo scambio di porzioni omologhe di materiale genetico. Questa definizione sottolinea l'importanza di scambiare parti del cromosoma che occupano la stessa posizione (omologhe) perché hanno lo stesso significato.

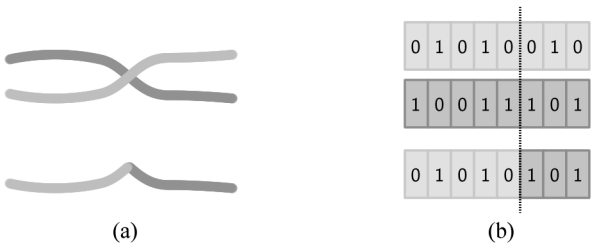


Figura 2.5 – Crossover nella biologia molecolare e crossover binario ad un punto.

L'operatore di crossover si può facilmente estendere ai cromosomi con codifica binaria combinando gli elementi di due cromosomi per generare un terzo. Il crossover binario si chiama *ad un punto* quando il cromosoma figlio di lunghezza n è composto dai primi k elementi del primo padre e dagli ultimi n-k elementi del secondo (figura 2.5b). Nel crossover a due punti si usa la stessa tecnica anche se i genitori vengono divisi in tre parti. Infine il crossover a n punti si ottiene costruendo un cromosoma dove l'i-esimo elemento è preso con la stessa probabilità dal primo o dal secondo padre.

Differente è l'approccio della codifica reale, in quanto non è riconducibile al crossover della biologia molecolare (figura 2.5a). Per cui l'operatore viene costruito con altre tecniche, sempre basate sul concetto di unire le caratteristiche di due individui per creare un terzo.

L'operatore di crossover viene applicato a due cromosomi a codifica reale la cui struttura è un vettore di reali di lunghezza n. Questo tipo di codifica rientra nei modelli di rappresentazione *one gene-one variable* [9], dove esiste una corrispondenza uno a uno tra i geni del cromosoma e le variabili dello spazio delle soluzioni. Consideriamo due individui C_1 e C_2 con n componenti:

$$C_1 = \left(c_1^1, c_2^1, ..., c_n^1\right) \qquad C_2 = \left(c_1^2, c_2^2, ..., c_n^2\right) \tag{2.1}$$

Il crossover più semplice è chiamato *flat crossover* [14] dove l'*i*-esima componente del cromosoma figlio viene scelta casualmente nell'intervallo $[c_1,c_2]$, determinato dalle *i*-esime componenti dei vettori C_1 e C_2 (figura 2.6a).

Nei capitoli successivi viene utilizzata un'estensione del *flat crossover* chiamata *BLX-α crossover*, il cui funzionamento è lo stesso ad eccezione della dimensione dell'intervallo I che assume una grandezza legata al parametro positivo α.

$$I = \left[c_{\min} - l \cdot \alpha, c_{\max} + l \cdot \alpha \right] \qquad c_{\min} = \min(c_i^1, c_i^2),\ c_{\max} = \max(c_i^1, c_i^2),\ l = c_{\max} - c_{\min} \tag{2.2}$$

L'*i*-esima componente del cromosoma figlio è un valore scelto nell'intervallo I con probabilità uniforme. Se il parametro α assume valore nullo ritroviamo il *flat crossover* (figura 2.6a), mentre con valori maggiori di zero l'intervallo aumenta a destra e a sinistra dei punti c_1 e c_2 (figura 2.6b e c).

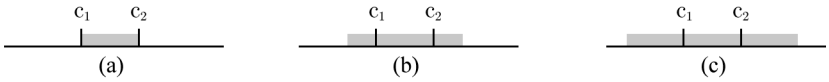


Figura 2.6 – BLX-α Crossover con parametro variabile (0.0, 0.5, 1.0).

Un aspetto che deve essere considerato nella scelta dell'operatore di crossover è la sua capacità di esplorare lo spazio delle soluzioni. Infatti deve bilanciare due comportamenti della ricerca chiamati *exploitation* e *exploration* che corrispondono alla ricerca locale e globale rispettivamente. L'operatore *flat crossover* equivalente a $BLX_{0.0}$ favorisce troppo l'*exploitation* (figura 2.6a), mentre il $BLX_{1.0}$ l'*exploration* (figura 2.6c). Un giusto compromesso tra i due comportamenti è il $BLX_{0.5}$ che assicura sia la convergenza che una buona capacità di esplorare lo spazio di ricerca (figura 2.6b).

2.2.3 Mutazione

L'operatore di mutazione introduce dei cambiamenti nel patrimonio genetico per generare nuovi cromosomi. Rispetto al crossover, che combina le caratteristiche di due cromosomi, la mutazione reintroduce delle aspetti genetici che erano andati persi modificando direttamente una parte di un singlo cromosoma. La mutazione di individui con una codifica binaria cambia il valore di uno o più componenti della stringa di bit (figura 2.7a).

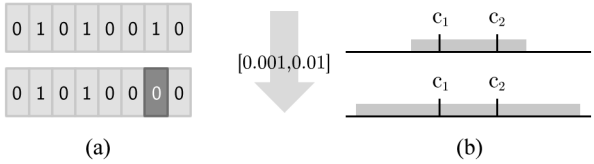


Figura 2.7 – Esempi di mutazione binaria e BLX.

L'effetto di questo operatore è di modificare profondamente il cromosoma, in modo che l'individuo mutato esplori zone dello spazio delle soluzioni non ancora osservate. Questo strumento viene introdotto per evitare la convergenza verso ottimi locali, favorendo così l'*exploration* dello spazio e la ricerca globale. D'altra parte, un utilizzo eccessivo della mutazione, rende instabile l'algoritmo e non assicura più la sua convergenza, per questo viene applicato con una probabilità molto bassa che oscilla nell'intervallo [0.001,0.01].

Nel caso di cromosomi rappresentati con una codifica reale la mutazione può essere ottenuta seguendo dei principi che favoriscono l'*exploration* dello spazio di ricerca. Descrivendo il *BLX-α crossover* nel paragrafo precedente si è evidenziato come valori alti del parametro α determinano questo comportamento. Per questo viene introdotta una variante dell'operatore di crossover che, con la stessa probabilità della mutazione, aumenta il parametro α.

In altre parole α assume un valore costante dato dalle impostazioni del crossover che viene aumentato con una probabilità [0.001,0.01], per favorire l'*exploration* dello spazio di ricerca (figura 2.7b).

2.3 L'evoluzione

La parte dell'algoritmo ispirata al principio della selezione naturale si occupa di classificare gli individui e selezionare le soluzioni migliori all'interno della popolazione. Con queste operazioni viene sollecitata la riproduzione tra gli individui migliori e viene garantita la convergenza dell'algoritmo verso una soluzione ottima che può essere locale o globale. Per ottenere una convergenza globale è importante che la popolazione sia composta da individui con un patrimonio genetico vario. Questo si traduce in un insieme eterogeneo di soluzioni con una visione globale dello spazio di ricerca, capace di individuare la completa topologia del problema.

2.3.1 Fitness

La *fitness* di un individuo è un valore che si calcola con una funzione e indica quanto il cromosoma e la sua codifica sia una buona soluzione del problema. La funzione di *fitness*, chiamata anche funzione obiettivo, permette il confronto tra gli individui e quindi l'ordinamento della popolazione.

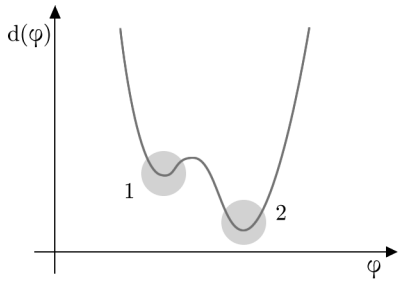


Figura 2.8 – Esempio della funzione di fitness.

Nel grafico (figura 2.8) viene mostrato l'andamento della funzione di *fitness* legata al problema del percorso minimo della figura 2.2. Lo scopo dell'algoritmo genetico è trovare il valore o i valori dell'angolo φ per cui la funzione $d(\varphi)$ assuma valore minimo perché rappresenta la distanza tra i punti A e B. In questo caso $d(\varphi)$ è la funzione di *fitness*.

La soluzione ottima del problema si trova nella area del grafico contrassegnata dal numero 2 (figura 2.8). Si nota però che esiste un punto di minimo locale (figura 2.8, punto 1). Se la popolazione è composta da individui che si trovano solo in quella zona l'algoritmo convergerà verso questa soluzione non ottima.

In conclusione è imporante che la popolazione sia composta da individui con caratteristiche eterogenee, almeno nelle prime generazioni, e che gli operatori di ricombinazione (mutazione e crossover) permettano l'*exploration* dello spazio delle soluzioni.

2.3.2 Selezione

Una volta calcolata la *fitness* di ogni individuo viene scelta una sottopopolazione per generare nuove soluzioni. Esistono varie tecniche per selezionare l'insieme delle soluzioni per la riproduzione chiamato *mating pool*. La più immediata e semplice consiste nel ordinare gli n individui della popolazione in base al valore della funzione di fitness e scegliere le prime k soluzioni (con $k < n$).

Consideriamo $P(t)$ la popolazione alla generazione t-esima formata da n individui ordinati secondo il valore dello loro *fitness* in modo crescente.

$$P(t) = \{x_1, x_2, ..., x_n\} \tag{2.3}$$

Se l'algoritmo genetico deve risolvere un problema di minimo è sufficiente selezionare i primi k elementi (figura 2.9a), altrimenti gli ultimi k elementi se è un problema di massimo (figura 2.9a).

Un'altra tecnica molto usata prende il nome di *proportional selection* che prevede di selezionare gli individui in base ad una probabilità proporzionale alla *fitness*. Come primo passo viene calcolata la *fitness totale* F della popolazione $P(t)$.

$$F_{\max} = \sum_{i=1}^{n} f(x_i) \qquad F_{\min} = \sum_{i=1}^{n} \frac{1}{f(x_i)} \tag{2.4}$$

In questa espressione $f(x_i)$ rappresenta il valore della *fitness* dell'individuo x_i. Inoltre il calcolo della *fitness totale* dipende se stiamo considerando problemi di minimo o di massimo.

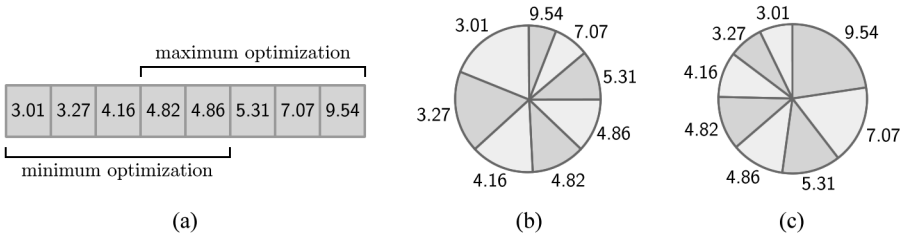


Figura 2.9 – Selezione lineare e proportional selection.

Grazie a questo valore è possibile ricavare la probabilità con cui le soluzioni saranno selezionate. Per ottenere queste probabilità si ricorre alle seguenti formule, utili nel caso di problemi di minimo e di massimo.

$$p_i = \frac{1}{f(x_i)} \cdot \frac{1}{F_{\min}} \qquad p_i = \frac{f(x_i)}{F_{\max}} \tag{2.5}$$

Il risultato di queste due espressioni possono essere rappresentate con una ruota suddivisa in spicchi, la cui grandezza è proporzionale alla probabilità associata alle soluzioni (figura 2.9b e c). La ruota viene fatta girare per k volte e vengono estratte le soluzioni corrispondenti che andranno a fa parte del *mating pool*.

2.3.3 Rimpiazzamento

L'operazione di rimpiazzamento è una strategia utilizzata per comporre la popolazione della generazione successiva $p(t+1)$. In questa fase vengono scelti quali e quanti individui devono far parte della prossima popolazione, stabilendo il numero di cromosomi figli da generare e scegliendo anche i cromosomi padre da rimpiazzare.

Una delle tecniche più utilizzate è creare r individui con la ricombinazione (crossover e mutazione), dove r è un numero minore di n, la dimensione della popolazione. Le rimanenti n-r soluzioni vengono scelte nella popolazione $p(t)$ con una delle possibili tecniche di selezione.

Il parametro r influenza la velocità della convergenza dell'algoritmo, infatti con valori piccoli si ha un basso ricambio generazionale che rallenta la ricerca della soluzione ottima. D'altra parte, non è possibile aumentare troppo il valore di r perché mantenere delle soluzioni della popolazione precedente produce dei vantaggi, soprattutto in termini di stabilità. Proprio questa caratteristica verrà ampiamente utilizzata nei capitoli successivi e negli algoritmi genetici multiobiettivo con il meccanismo degli archivi.

Capitolo 3
Algoritmi genetici multiobiettivo

L'ottimizzazione multiobiettivo è stata ed è tuttora al centro di molti studi. Nel corso degl'anni sono stati sviluppati numerosi algoritmi e diverse applicazioni anche se la maggior parte di essi hanno un approccio poco multiobiettivo. Infatti, molti metodi utilizzano una trasformazione per convertire il problema in una forma equivalente a singolo obiettivo, con il vantaggio di poter utilizzare i numerosi strumenti già presenti per questo tipo di ottimizzazione.

In questo contesto l'ottimizzazione multiobiettivo diventa una semplice estensione di quella a singolo obiettivo, anche se intuitivamente, quest'ultima sembrerebbe essere una caso particolare della prima. Infatti, oltre a questa famiglia di metodi di risoluzione, esistono delle tecniche che trattano l'ottimizzazione multiobiettivo senza le funzioni di trasformazione e senza ricondursi a problemi a singolo obiettivo.

Per analizzare questi metodi faremo riferimento ad un problema a due obiettivi ripreso dall'esempio del percorso minimo nel capitolo precedente.

3.1 Problemi multiobiettivo

I problemi di ottimizzazione multiobiettivo (*MOOP*) sono caratterizzati da più funzioni che devono essere ottimizzate simultaneamente. L'esempio riportato nel capitolo precedente prevedeva una sola funzione obiettivo che indicava la distanza tra due punti, infatti si trattava di un problema di percorso minimo a singolo obiettivo (paragrafo 2.2).

Per considerare un problema multiobiettivo introduciamo un seconda funzione che valuta il costo dello spostamento dal punto A al punto B. Il valore della funzione dipende da φ perché è legato al percorso ma non alla distanza, infatti, il tragitto più breve non è il più facile da percorrere e quello meno costoso non è il più corto. In questo caso i due obiettivi si dicono in *conflitto* perché non è possibile trovare una soluzione ottima che vada bene per entrambi.

Consideriamo *d(φ)* funzione obiettivo della distanza e *c(φ)* funzione obiettivo del costo definite dalle seguenti formule:

$$\begin{aligned} d(\varphi) &= \varphi^2 + 1 \\ c(\varphi) &= (\varphi - 2)^2 + 1 \end{aligned} \tag{3.1}$$

Il fine del problema di ottimizzazione è minimizzare entrambe le espressioni che hanno un punto di minimo nei punti 0 e 2, rispettivamente per la funzione di distanza e per la funzione di costo. A causa di questo non esiste alcun valore di φ per il quale entrambe le espressioni assumano valore minimo e si vede facilmente che le due funzioni sono in contrasto perché migliorando il valore di una si peggiora il risultato dell'altra.

Quest'esempio mostra chiaramente il concetto di *conflitto* tra due funzioni e introduce un'importante caratteristica dei problemi multiobiettivo. Analizzando il sistema non siamo in grado di minimizzare entrambe le espressioni (3.1) perché i loro valori sono in contrasto. Per cui non è possibile individuare un'unica soluzione ottima visto che abbiamo due criteri di scelta tra loro contrastanti e quindi non è possibile capire se una soluzione e meglio dell'altra.

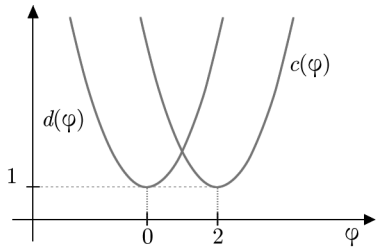


Figura 3.1 – Esempio di funzioni obiettivo legate al costo e alla distanza.

Per risolvere questo problema bisogna considerare valido un intero insieme di soluzioni per il quale non siamo in grado di definire una preferenza. Questo insieme sarà formato da soluzioni ritenute tutte valide come soluzioni del sistema.

La cardinalità delle soluzioni è la principale differenza tra i problemi multiobiettivo e i problemi a singolo obiettivo dove la soluzione, se esiste, è unica. Nell'esempio del tragitto le soluzioni ottime appartengono all'intervallo [*0,2*] perché sono un buon compromesso tra la distanza e il costo.

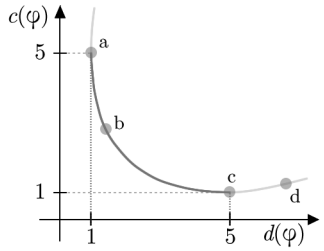


Figura 3.2 – Soluzioni ottime dell'esempio del tragitto.

Il precedente diagramma mostra la composizione dell'insieme delle soluzioni dal punto di vista delle funzioni obiettivo (figura 3.2, linea scura). Il punto *a* di coordinate (*1,5*) è la soluzione ottima per la distanza, mentre il punto c è la soluzione ottima del costo. Anche la soluzione intermedia *b* (*1.49*, *2.69*) è ancora valida rispetto alle altre due. Infatti se consideriamo il tragitto associato al punto *a* si nota che ha un costo maggiore rispetto a *b*, ma allo stesso tempo ha una distanza minore. Non potendo scegliere tra il costo e la distanza queste due soluzioni sono entrambe valide.

Queste considerazioni sono le stesse che permettano di scegliere i punti corretti e ricavare l'insieme delle soluzioni. Per determinare questo esistono delle tecniche ben precise che verranno illustrare nei paragrafi successivi, comunque possiamo fare una prima analisi delle soluzioni mediante la figura 3.2. Pertanto consideriamo il punto *d* di coordinate (*6.76,1.16*) e notiamo che entrambi gli obiettivi sono maggiori del punto *c*. In questo caso si può dire che il tragitto identificato dal punto *d* è peggiore del tragitto identificato da *c* e per questo motivo non viene inserito nell'insieme delle soluzioni.

In generale questo insieme viene formato da i soli punti per i quali non è possibile trovare soluzioni che siano migliori. In altre parole non esistono soluzioni che siano migliori delle soluzioni contenute nell'insieme ottimo.

3.1.1 Due approcci alla risoluzione

Alla fine della ricerca quello che si desidera è un'*unica* soluzione, mentre abbiamo visto che l'ottimizzazione multiobiettivo produce un'insieme di soluzioni. Dato che gli elementi all'interno dell'insieme sono tutti validi, si richiede all'utente di fare una scelta, che può esser fatta in base a parametri che sono estranei alle funzioni obiettivo. In genere questi parametri sono più soggettivi e possono essere legati a particolari campi di utilizzo.

Esistono per cui due tipi di informazioni che permettano la risoluzione del problema. Le prime sono generali e si possono quantificare in una funzione obiettivo, mentre le seconde chiamate informazioni ad *alto livello,* sono soggettive e specifiche di un determinato settore di utilizzo.

Queste due differenti tipologie di informazioni si utilizzano in fasi diverse durante la risoluzione del problema. A seconda dell'ordine con cui si decide di adoperare i dati si

possono avere due modi per trovare le soluzioni, che portano comunque allo stesso risultato.

La prima tecnica di ricerca si basa sull'analisi diretta delle funzioni obiettivo con metodi di ottimizzazione multiobiettivo. Grazie a questi viene trovato l'insieme delle soluzioni ottime e viene estratta la soluzione desiderata con le informazioni ad alto livello che possiede l'utente (figura 3.3). La informazioni generali e meno soggettive vengono utilizzate nella prima fase per definire le funzioni obiettivo.

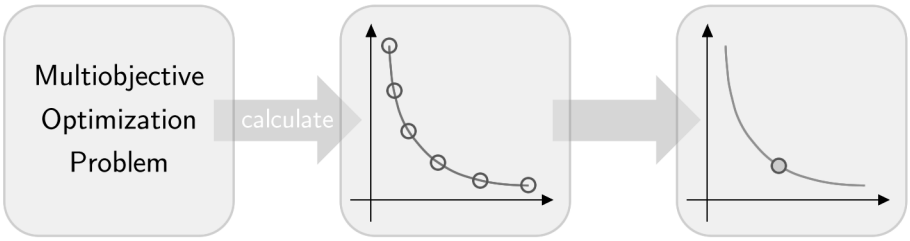


Figura 3.3 – Schema dell'approccio multiobiettivo.

La seconda tecnica utilizza le informazioni ad alto livello per assegnare delle preferenze degli obiettivi all'inizio del procedimento. Queste preferenze vengono quantificate in pesi in modo che si possa trasformare il problema in uno equivalente a singolo obiettivo. Questo è possibile considerando un'unica funzione obiettivo data dalla somma pesata delle funzioni obiettivo originali.

Figura 3.4 – Schema della strategia basata sulle preferenze.

Questo tipo di approccio alla risoluzione fa parte delle strategie basate sulle preferenze (*preference-based strategy*) [4] perché utilizza le informazioni ad alto livello per privilegiare alcuni obiettivi rispetto ad altri. Riportato il problema nella forma a singolo obiettivo si possono adoperare le tecniche classiche di ottimizzazione e ricavare l'unica soluzione (figura 3.4).

Il primo problema che si incontra utilizzando le strategie basate sulle preferenze è creare pesi. Infatti i dati ad alto livello in possesso dall'utente devono essere interpretati perché sono informazioni qualitative, alle quali è difficile associare dei valori.

L'interpretazione delle preferenze dell'utente introduce anche un altro problema legato al riutilizzo delle informazioni. Consideriamo il caso in cui non si è riusciti ad esprimere correttamente ciò che l'utente desiderava o, più semplicemente, è cambiato il contesto di utilizzo e le preferenze sono differenti. In questi casi la seconda strategia (figura 3.4) deve essere rieseguita completamente per ottenere la nuova soluzione, con evidenti costi in termini di tempo e calcolo.

Nell'approccio ottenuto per mezzo dall'ottimizzazione multiobiettivo diretta (figura 3.3) si crea l'insieme di tutte le soluzioni ottime e poi, in base alle informazioni dell'utente, viene scelta la soluzione corretta. Per cui se le preferenze sugli obiettivi dovessero cambiare non occorre ripetere di nuovo i calcoli, ma è sufficiente scegliere una nuova soluzione.

3.1.2 Algoritmi evoluzionistici

La maggior parte degli strumenti classici per risolvere i problemi di ottimizzazione a singolo obiettivo sono metodi iterativi ad un punto. Queste tecniche calcolano ad ogni iterazione un'approssimazione della soluzione in base alla precedente, spostandosi di punto

in punto nello spazio di ricerca (*point-by-point approach*). In questo modo si viene a creare una successione di soluzioni o punti che convergono verso la soluzione ottima.

Negl'ultimi anni sono state introdotte delle tecniche meno convenzionali per risolvere problemi di ottimizzazione che prendono il nome di algoritmi genetici o evoluzionistici (*evolutionary algorithm*, *EA*). Questi strumenti descritti nel capitolo precedente seguono una strategia di ricerca basata su un insieme di soluzioni (popolazioni di individui) che si evolvono verso la soluzione ottima seguendo un unico obiettivo.

La natura di questi strumenti li rende particolarmente efficaci nel campo dell'ottimizzazione multiobiettivo dove il numero di soluzioni da trovare è maggiore di uno. Infatti il risultato di un algoritmo evoluzionistico è sempre un insieme di soluzioni. Nel caso dell'ottimizzazione a singolo obiettivo tale insieme è formato da elementi tutti simili e vicini alla soluzione ottima, mentre nel caso dell'ottimizzazione multiobiettivo è composto da quell'insieme di soluzioni che costituiscono il *trade-off* tra gli obiettivi del problema.

3.2 Soluzioni ottime

Nel paragrafo precedente è stata illustrata in maniera non rigorosa la composizione dell'insieme delle soluzioni di un problema di ottimizzazione multiobiettivo. In questo paragrafo verranno illustrati i criteri per individuare le soluzioni ottime e le caratteristiche che accompagnano tale insieme.

Prima di tutto definiamo lo spazio degli obiettivi con i valori delle funzioni obiettivo: per ogni punto dello spazio di ricerca si calcolano i valori delle funzioni e si assumono come le coordinate del nuovo spazio. In questo modo esiste una corrispondenza tra i punti dello spazio di ricerca e i punti dello spazio degli obiettivi (figura 3.5).

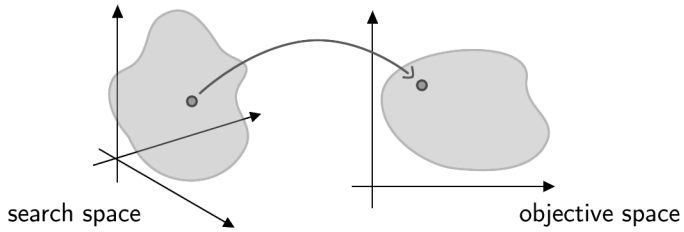


Figura 3.5 – Corrispondenza tra lo spazio di ricerca e lo spazio degli obiettivi.

Per meglio capire il concetto riutilizziamo l'esempio del tragitto con le due funzioni obiettivo legate al costo e alla distanza. In questo caso lo spazio di ricerca ha una sola dimensione e corrisponde alla variabile φ. Ogni valore assunto dalla variabile corrisponde ad un particolare tragitto (figura 2.3), al quale viene associata una distanza e un costo con le funzioni obiettivo.

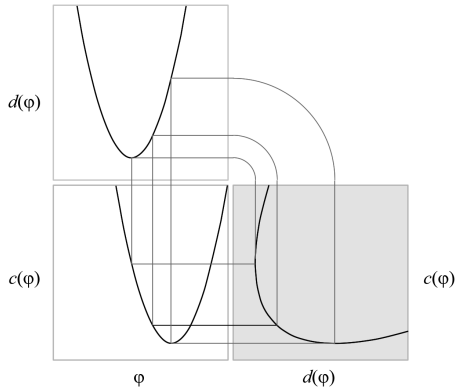


Figura 3.6 – Esempio di costruzione dello spazio degli obiettivi.

Questi due valori formano le coordinate dei punti all'interno dello spazio degli obiettivi come si vede dal grafico dalla figura 3.6. A questo punto siamo in grado di identificare l'intero spazio delle soluzioni, ma non siamo ancora capaci di definire quali soluzioni sono

ottime e quali no. Per fare questo bisogna illustrare alcuni concetti che caratterizzano le soluzioni ottime e che permettano di distinguerle dalle altre.

3.2.1 Il concetto di dominanza

All'inizio capitolo sono stati presentati i problemi multiobiettivo ed è stata data una prima e poco formale definizione dell'insieme delle soluzioni ottime (paragrafo 3.1). In questo paragrafo e nel successivo illustreremo le tecniche e le teorie che permettono di definire in modo rigoroso questo insieme.

Il primo aspetto che analizziamo è il *concetto di dominanza* tra le soluzioni. Per questo consideriamo per cui un problema di ottimizzazione con m obiettivi e un insieme di soluzioni x_i dal quale vogliamo selezionare le componenti migliori.

Il concetto di dominanza si applica quando si confrontano due soluzioni e i corrispettivi valori obiettivo associati. Ogni singolo valore è direttamente confrontabile per cui, se considero le soluzioni x_1 e x_2, siamo in grado di determinare quale è la migliore per un determinato obiettivo j.

$$f_j(x_1) < f_j(x_2) \qquad j \in \{1, 2, \ldots, m\} \tag{3.2}$$

Nell'espressione 3.2 il valore di $f_j(x_1)$ è minore del valore di $f_j(x_2)$, dove $f_j(x)$ rappresenta la j-esima funzione obiettivo. Nel caso dei problemi di minimo la soluzione x_1 si considera migliore di x_2 secondo l'obiettivo j. Quest'informazione non dà nessuna indicazione generale sul rapporto tra le soluzioni perché non può essere estesa agl'altri obiettivo. Infatti sebbene sia possibile calcolare per ogni obiettivo il valore assunto dalla soluzione, non è possibile riassumere queste informazioni in un'unica valutazione. In altre parole, se la soluzione x_1 è migliore di x_2 per k obiettivi, ma peggiore nei rimanenti m-k, non è possibile determinare quale sia la migliore in generale. Questo problema si presenta quando non esiste alcuna preferenza tra gli obiettivi e non è possibile attribuire più importanza ad uno rispetto all'altro.

In questo contesto viene definito il *concetto di dominanza* per confrontare le soluzioni quando mancano le preferenze tra gli obiettivi:

> *Si dice che una soluzione x_1 domina un'altra soluzione x_2, se entrambe le condizioni 1 e 2 sono vere:*
> 1. *La soluzione x_1 non è peggiore di x_2 in tutti gli obiettivi, o $f_j(x_1) \leq f_j(x_2)$ per tutti i valori $j = 1, 2, \ldots, m$ (problema di minimo).*
> 2. *La soluzione x_1 è strettamente migliore di x_2 per almeno un obiettivo, o $f_j(x_1) < f_j(x_2)$ per almeno un valore $j = 1, 2, \ldots, m$ (problema di minimo).*

Seguendo questa definizione, la dominanza è una condizione che una soluzione può assumere rispetto ad un'altra quando tutti gli obiettivi sono concordi e non si presenta il problema di dover esprime una preferenza.

Consideriamo adesso un insieme di sette soluzioni in un problema di minimo con due obiettivi (figura 3.7a). Se esaminiamo le soluzioni c e b, e i valori obiettivo corrispondenti, si nota che la prima soluzione domina la seconda. Questo perché entrambi gli obiettivi di c sono minori (o migliori) di b (figura 3.7b).

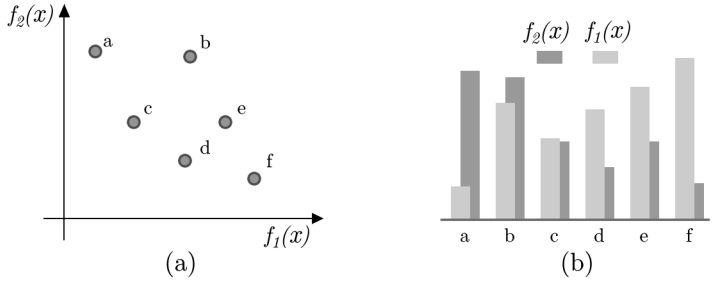


Figura 3.7 – Dominanza in un problema a due obiettivi.

Possiamo osservare che la soluzione c ha lo stesso valore obiettivo $f_2(x)$ di e, mentre l'altro obietto è minore: anche in questo caso la soluzione c domina l'altra soluzione.

3.2.2 L'insieme ottimo Pareto

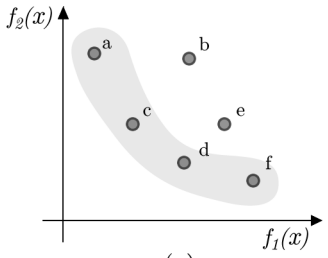
Il concetto di dominanza è alla base di molti algoritmi per risolvere i problemi multiobiettivo, infatti è fondamentale per riconoscere l'insieme delle soluzioni ottime. Dato un insieme P di n soluzioni è possibile considerare una soluzione x_i e confrontarla con le altre rimanenti x_j, con $j = 1, 2, ...,n$ e $j \neq i$. Dal confronto si hanno $n-1$ risultati che possono valere *dominato* o *non-dominato*. A questo punto la soluzione x_i diventa *non-dominata* se tutti gli $n-1$ confronti hanno dato *non-dominato*, viceversa, è *dominata* se almeno un confronto è stato *dominato*.

Procedendo con il confronto di tutte le soluzioni x_i (con $j = 1, 2, ...,n$) si può dividere P in due sottoinsiemi chiamati *insieme delle soluzioni non-dominate* e *insieme delle soluzioni dominate* seguendo questa definizione:

> *Dato un insieme di soluzioni P, l'insieme delle soluzioni non-dominate P' è formato da tutte quelle soluzioni che non sono dominate da nessun altro membro dell'insieme P.*

In riferimento all'esempio di figura 3.8 si osserva che le soluzioni c e d dominano entrambe le soluzioni b ed e (figura 3.8b), mentre le altre non sono dominate da nessuno: seguendo la definizione l'insieme P' è formato dalle soluzioni $\{a, c, d, f\}$.

L'insieme appena costruito è formato dalle soluzioni che sono le migliori all'interno di P, mentre lo scopo degli algoritmi che risolvono problemi multiobiettivo è trovare le soluzioni che sono migliori in tutto lo spazio di ricerca S.



(a)

	a	b	c	d	e	f
f	N	N	N	N	N	
e	N	N	N	N		N
d	N	D	N		D	N
c	N	D		N	D	N
b	N		N	N	N	N
a		N	N	N	N	N

(b)

Figura 3.8 – Fronte di non-dominanza in un problema a due obiettivi.

Per questo motivo viene definito un altro insieme, *l'insieme delle soluzioni ottime di Pareto*, che contiene le soluzioni ottime del problema multiobiettivo:

> *L'insieme non-dominato su tutto l'intero spazio di ricerca ammissibile S viene chiamato l'insieme ottimo di Pareto.*

In altre parole l'insieme ottimo di Pareto è un particolare insieme di non-dominanza ottenuto considerando lo spazio di ricerca S al posto di P, oppure con $P = S$.

Grazie a queste due definizioni un algoritmo genetico è in grado di calcolare le soluzioni di un problema. Queste soluzioni sono contenute nell'insieme ottimo di Pareto che in genere ha dimensione infinita, come per esempio il problema di figura 3.2, dove le soluzioni erano comprese tra 1 e 5 in entrambi gli obiettivi (linea scura). In questi casi un algoritmo genetico non è in grado di rappresentare tutte le soluzioni ottime di Pareto perché ha a disposizione solo un numero finito di individui.

Da questa considerazione si capisce che il compito di un algoritmo genetico non è solamente cercare alcune soluzioni ottime, ma fare in modo che queste rappresentino al meglio l'insieme ottimo di Pareto.

3.3 Algoritmi

Gli algoritmi genetici multiobiettivo (*MOGA*) sono gli strumenti per calcolare quel gruppo di soluzioni che sono il risultato di un problema di ottimizzazione multiobiettivo. In questo paragrafo verranno mostrate alcune procedure per ricavare l'insieme non-dominato. Queste tecniche sono utilizzate da gli algoritmi genetici per selezionare le soluzioni migliori che in seguito verranno utilizzate per creare la popolazione della generazione successiva.

3.3.1 Metodo base

Come primo metodo analizziamo il più semplice e immediato algoritmo che individua l'insieme delle soluzioni non dominate. Per cui, dato un'insieme P formato da n soluzioni, si considera la soluzione i-esima e si confronta con tutte le altre (figura 3.9a, punto 1). Se risulta dominata da almeno una sola soluzione si passa alla soluzione $i+1$, altrimenti si aggiunge all'*insieme non-dominato* (figura 3.9a, punto 2). Questa procedura si ripete per tutte le soluzioni x_i all'interno di P (figura 3.9a, punto 3).

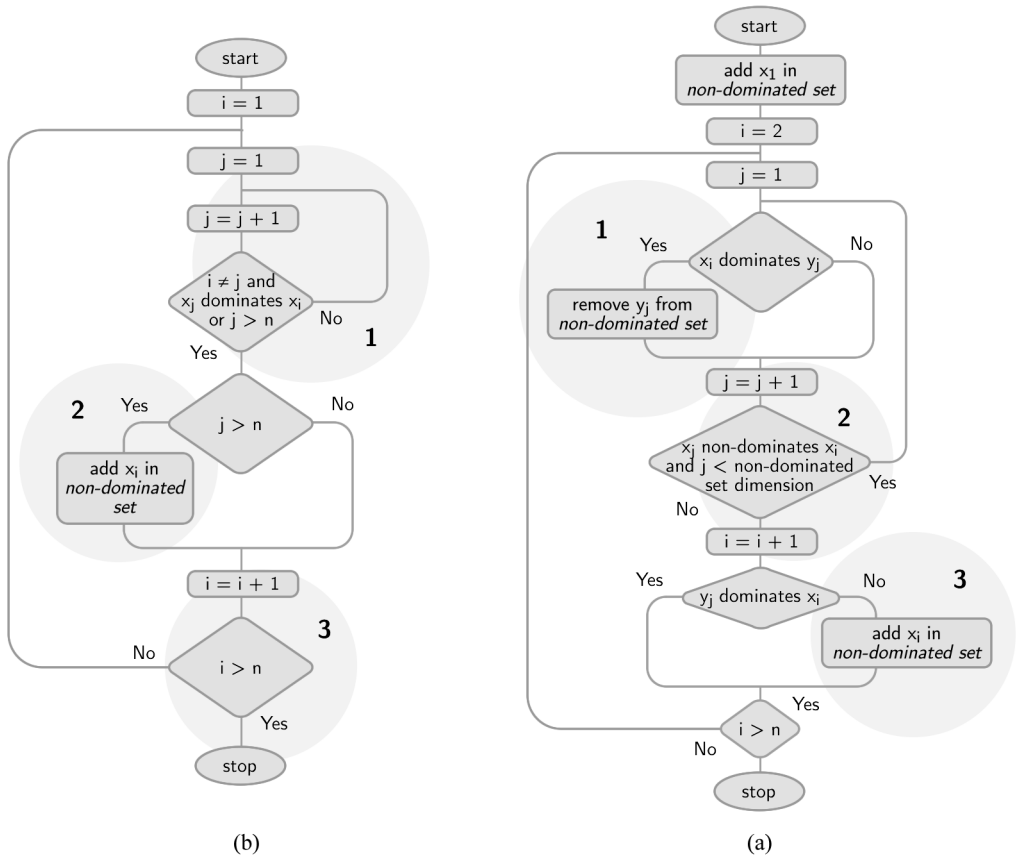


Figura 3.9 – Metodo base e variante per ricavare l'insieme di non-dominanza.

La semplicità di questo metodo si paga in termini di prestazioni, infatti il confronto viene ripetuto n volte per ognuna delle n soluzioni: questo si nota anche dal diagramma perché ci sono due cicli annidati (figura 3.9a, punto 1 e 3). La complessità dell'algoritmo è quindi $O(mn^2)$ dove n è il numero di soluzioni e m è il numero di obiettivi.

3.3.2 Metodo degli aggiornamenti continui

Il metodo degli aggiornamenti continui è una variante del metodo precedente che perfeziona la procedura del confronto tra le soluzioni migliorando le prestazioni dell'algoritmo. Per descrivere il funzionamento di questo metodo facciamo riferimento al diagramma di figura 3.9b.

Per prima cosa la soluzione x_1 di P viene messa nell'insieme non-dominato. Successivamente si passa a confrontare ogni soluzione x_i di P con le soluzioni y_j dell'insieme non-dominato. Questo confronto può dare tre esiti:

a) ogni soluzione y_j dominata da x_i viene esclusa dall'insieme di non-dominanza (figura 3.9b, punto 1).
b) se la soluzione x_i non è dominata da nessun'altra entra nell'insieme di non-dominanza e si passa alla successiva x_{i+1} (figura 3.9b, punto 3).
c) se la soluzione x_i è dominata da una soluzione y_j si passa alla successiva x_{i+1}.

Questa tecnica diminuisce il numero di confronti perché li limita solo all'insieme non-dominato. Infatti l'i-esima soluzione viene confrontata al massimo con $i-1$ soluzioni

dell'insieme, per cui in totale abbiamo *n*(n-1)/2* confronti come caso peggiore, rispetto ai n^2 del metodo precedente. Si nota però che in entrambi i metodi la complessità è la stessa.

3.3.3 Metodo efficiente di Kung

Il metodo di Kung [11] rappresenta lo strumento più efficiente per ricavare l'insieme di non-dominanza, infatti la complessità è data da $O(nlog(n)^{m-2})$ con $m > 2$ e da $O(nlogn)$ con $m = 2$. Questa tecnica si basa su una funzione ricorsiva *front(P)* e sulla strategia *divide et impera*.

Prima di utilizzare la funzione l'insieme P delle soluzioni viene ordinato secondo il primo obiettivo in modo crescente (problemi di minimo). All'interno della funzione l'insieme delle soluzioni P viene diviso in due parti, parte alta e parte bassa, e per ognuna viene rilanciata la funzione *front()* (figura 3.10, punto 1). Il risultato delle funzioni viene assegnato alle variabili T (*top*, parte alta) e B (*bottom*, parte bassa).

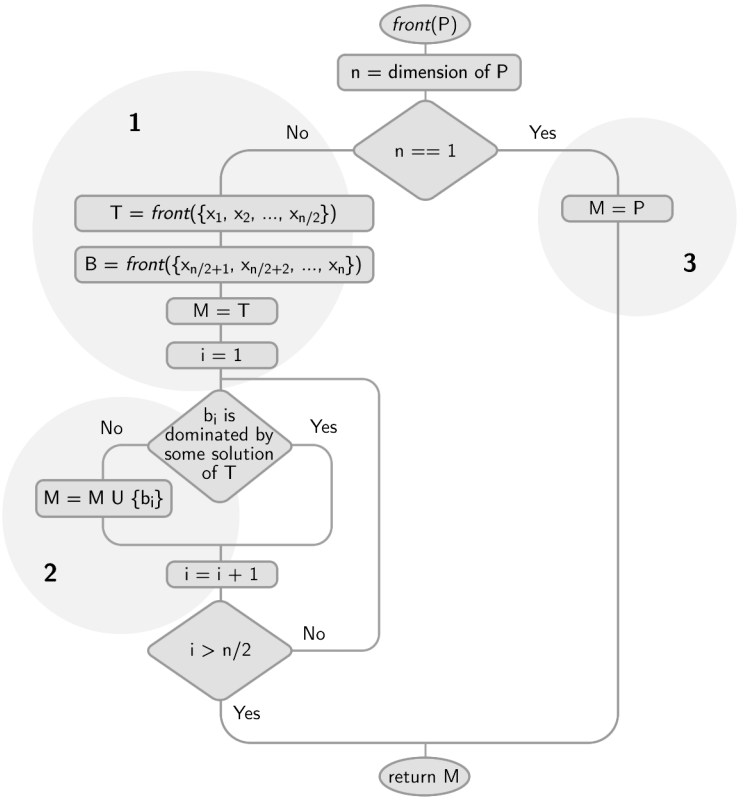


Figura 3.10 – Metodo di Kung per ricavare l'insieme di non-dominanza.

Nessuna soluzione in T può essere dominata dalle soluzioni in B perché nel primo insieme ci sono soluzioni che hanno almeno un valore obiettivo migliore per via dell'ordinamento. Tuttavia per capire se le soluzioni in B sono dominate da T si deve fare un confronto tra gli insiemi e tutte le soluzioni non dominate vengono inserite nell'insieme M di cui fa parte anche T (figura 3.10, punto 2).

La condizione di terminazione della funzione ricorsiva è data dalla dimensione dell'insieme che deve analizzare. Infatti se l'insieme è formato da una sola soluzione la funzione termina (figura 3.10, punto 3).

Il meccanismo ricorsivo di questa funzione si basa sul dividere in due parti le soluzioni. Grazie all'ordinamento di un obiettivo la parte alta non è dominata dalla parte bassa, per cui bisogna solo verificare quali soluzioni sono dominate nella parte bassa. Prima di fare questo confronto (figura 3.10, punto 2) ci si assicura che i due insiemi siano composti da soluzioni che non si dominano tra di loro: viene rilanciata la funzione su entrambi gli insiemi (figura 3.10, punto 1). Così facendo il confronto che ha sempre complessità $O(mn^2)$ non avviene su tutto P, ma solo su sottoinsiemi non dominati.

3.4 Non-dominated sorting genetic algorithm II

Risolvere un problema multiobiettivo vuol dire trovare quel gruppo di soluzioni che meglio descrivono l'insieme ottimo di Pareto. Questo insieme è formato da un numero infinito di elementi, mentre il numero di soluzioni a disposizioni è limitato dalla capacità di calcolo e di memoria dell'elaboratore. Per cui l'algoritmo di risoluzione deve trovare delle soluzioni che, oltre ad essere contenute nell'insieme ottimo di Pareto, siano delle ottime rappresentazione di tale insieme. Questa ultimo aspetto si ottiene imponendo una distribuzione uniforme delle soluzioni nello spazio degli obiettivi.

Nel capitolo precedente abbiamo visto che il calcolo della fitness è necessario per ordinare le soluzioni, applicare la selezione e scegliere gli individui migliori. Nei problemi multiobiettivo il calcolo della fitness (valore della funzione obiettivo) non è più sufficiente per la selezionare gli individui migliori perché ci sono più obiettivi e un solo criterio di scelta. Per risolvere questo problema viene introdotto il concetto di dominanza e creato l'insieme ottimo di Pareto in modo da poter classificare ugualmente la popolazione. Tuttavia questo ordinamento divide la popolazione in soluzioni dominate e non, dove le prime sono peggiori delle seconde, e non indica nessun altro criterio di scelta all'interno di questi due insiemi.

In questo paragrafo e nei successivi verrà presentato un algoritmo genetico chiamato Non-dominated sorting genetic algorithm II che utilizza un particolare ordinamento a fronti e un sistema di distribuzione delle soluzioni [3].

3.4.1 Ordinamento a fronti non-dominati

Le soluzioni all'interno dell'insieme non-dominato sono migliori delle soluzioni all'interno di quello dominato. Se analizziamo quest'ultimo insieme possiamo cercare di suddividere ulteriormente le soluzioni non-dominate da quelle dominate, così da creare nuovi insiemi chiamati fronti. Questa operazione ha senso perché si distinguono le soluzione che hanno un livello di dominanza differente e si ottiene una classifica in base a questi fronti. Per cui il primo fronte conterrà le soluzioni non-dominate, il secondo le soluzioni dominate dal primo, il terzo quelle dominate dal secondo e l'ultimo le soluzioni dominate da tutti i fronti precedenti (figura 3.11).

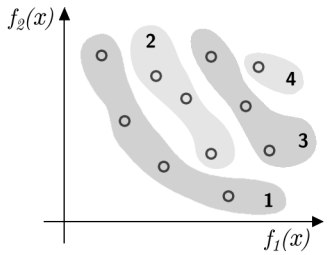


Figura 3.11 – Soluzioni divise in quattro fronti.

Per ottenere questo ordinamento è sufficiente ripetere il calcolo dell'insieme non-dominato in ogni insieme dominato. L'inconveniente di questo metodo è l'elevata complessità pari a $O(mn^3)$, per questo sono state sviluppate delle tecniche che velocizzano l'ordinamento a fronti e riducono il numero di operazioni che devono essere eseguite.

Una tecnica molto semplice seguita da NSGA II è legata al metodo base che ha complessità $O(mn^3)$. Per ogni soluzione si contano il numero di volte c_i che viene dominata e si memorizzano le soluzioni che dominano nel vettore s_i. In questo modo si possono inserire le soluzioni con c_i nullo (le soluzioni non-dominate) nel primo fronte. Per ogni soluzione inserita si considerano le soluzioni in s_i e per ognuna di esse si diminuisce il valore c_i di uno. In questo modo è possibile costruire il secondo fronte considerando le soluzioni che dopo le sottrazioni hanno c_i nullo. Questa procedura viene ripetuta fino a quando non è più possibile costruire fronti perché non ci sono più soluzioni.

3.4.2 Preservare la diversità

L'algoritmo genetico multiobiettivo deve garantire una buona distribuzione delle soluzioni per rappresentare al meglio l'insieme ottimo di Pareto. Questa caratteristica si

ottiene preservando la *diversità delle soluzioni*, ovvero cercando di avere un insieme di soluzioni che sia il più eterogeneo possibile. Per fare questo, l'algoritmo genetico classifica gli individui della popolazione in base ad un criterio prossimo alla densità, dove le soluzioni che si trovano in zone affollate ricevono una minore priorità di selezione. In questo modo si evita la formazione di gruppi di individui vicini tra loro e si favorisce la distribuzione uniforme delle soluzioni.

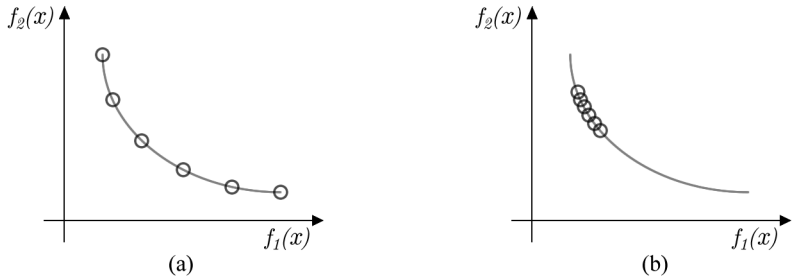


Figura 3.12 – Distribuzione delle soluzioni lungo il fronte di Pareto.

Nell'esempio in figura 3.12 sono riportati due casi di distribuzione. In entrambi i grafi le soluzioni sono ottime perché si trovano sul fronte ottimo di Pareto (linea scura). Tuttavia è evidente che solo il primo (figura 3.12a) rappresenta meglio il fronte perché ha una buona distribuzione, mentre il secondo (figura 3.12b) ha tutte le soluzioni raggruppate in una piccola parte del fronte.

L'algoritmo NSGA II utilizza il metodo della *crowding distance assignment* per classificare le soluzioni in base alla densità. Questo strumento non calcola la densità effettiva delle soluzioni, ma si basa su una particolare distanza dai vicini.

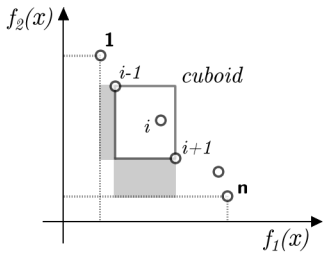


Figura 3.13 – Distribuzione delle soluzioni lungo il fronte di Pareto.

Ad ogni soluzione viene assegnata la crowding distance che corrisponde alla somma dei lati del *cuboide* (figura 3.13). Ogni lato è legato ad un obiettivo e per ricavare la lunghezza si considera la differenza tra i valori obiettivo della soluzione precedente e della successiva. In altre parole, se stiamo calcolando il lato j-esimo, ordiniamo la popolazione in base al j-esimo obiettivo e per ogni soluzione i si calcola la differenza $f_j(x_{i-1})$ - $f_j(x_{i+1})$. Questa differenza viene normalizzata dal fattore $f_j(x_1)$ - $f_j(x_n)$ che corrispondere alla distanza tra le soluzioni estreme x_1 e x_n, dove n è la dimensione della popolazione. Questo perché, prima di sommare i lati, è necessario normalizzare per non far valere la scala degl'obiettivi.

Come si vede dal grafico (figura 3.13) la crowding distance della soluzione x_i si calcola con la somma normalizzata dei due lati del cuboide, la cui dimensione è data dalle soluzioni x_{i-1} e x_{i+1}. Per le soluzioni estreme x_1 e x_n non è possibile costruire il cuboide, per cui la crowding distance assume valore infinito (∞).

Ordinando in modo crescente i valori calcolati si possono determinare le soluzioni che si trovano nelle zone più dense. Infatti questi individui occupano le prime posizioni dell'ordinamento, mentre nelle ultime posizioni ci sono le soluzioni agli estremi del fronte.

3.4.3 Ranking per la selezione

L'algoritmo NSGA II riesce ad individuare le soluzioni dell'insieme ottimo di Pareto attraverso l'ordinamento in fronti non dominati e garantisce la diversità tra gli individui con la crowding distance. Queste due tecniche vengono utilizzate insieme per ordinare la

popolazione e ottenere una classifica chiamata *ranking* che indica quali sono le soluzioni migliori.

Negli algoritmi genetici a singolo obiettivo il ranking è legato direttamente alla funzione di fitness e si ottiene attraverso il semplice ordinamento, mentre nei multiobiettivo si utilizzano tecniche più complesse perché ci sono più funzioni di fitness e altri criteri da considerare.

Nel caso del NSGA II il *ranking* si ottiene ordinamento la popolazione in fronti non-dominati e considerando la distribuzione delle soluzioni.

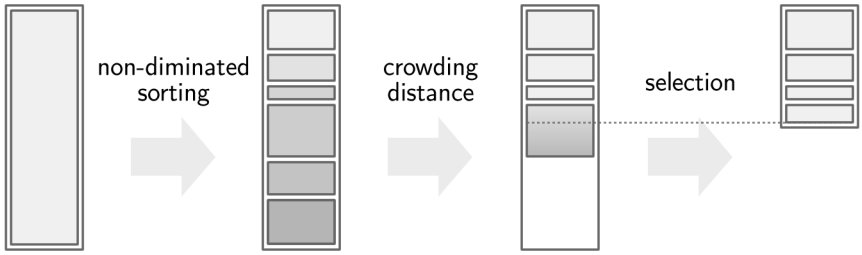


Figura 3.14 – Ranking e selezione della popolazione.

Per prima cosa le soluzioni sono divise in fronti seguendo il criterio di dominanza (figura 3.14). A questo punto è difficile che il numero di soluzioni selezionate sia esattamente contenuto in un insieme intero di fronti, infatti, capita spesso che bisogna scegliere le soluzioni all'interno di un fronte. Per fare questo è possibile utilizzare un criterio che ordina le soluzioni in un fronte in base alla distribuzione degli individui, come per esempio la crowding distance (figura 3.14).

$$x_i \prec x_j \Leftrightarrow \left(rank_i < rank_j\right) \vee \left(\left(rank_i = rank_j\right) \vee \left(distance_i > distance_j\right)\right) \tag{3.3}$$

In maniera più rigorosa viene definito un nuovo operatore chiamato *crowded comparison* [3] che definisce l'ordinamento tra gli individui, indicando quando una soluzione è migliore dell'altra. Questo operatore reputa x_i migliore di x_j se appartiene ad un fronte minore (*rank*) oppure, nel caso che il fronte sia lo stesso, se ha una crowding distance maggiore (3.3).

Capitolo 4
MOGA paralleli

In questo capitolo si studiano alcune tra le possibili implementazioni parallele di in algoritmo genetico multiobiettivo. Lo scopo di questo studio è ottenere versioni distribuite dell'algoritmo per aumentare la potenza di calcolo utilizzando più processori. Analizzando la natura dei problemi multiobiettivo si nota che il calcolo della fitness richiede un tempo di esecuzione maggiore rispetto ad altri operatori (crossover, mutazione, ranking, ecc.). Si cercano per cui tecniche che possano distribuire questo calcolo su più processori, considerando tutti i possibili aspetti e modi per scomporre un algoritmo genetico. Tutti queste considerazioni vengono affrontate tenendo ben presente il costo della comunicazione, che ha un peso paragonabile al tempo di esecuzione dell'algoritmo stesso.

4.1 Architetture di calcolo

Prima di studiare gli algoritmi genetici multiobiettivo paralleli analizzano le possibili architetture che possono ospitare questi strumenti. La tassonomia di Flynn [7] classifica i sistemi di calcolo in base al flusso di dati e al flusso di istruzioni e li raggruppa in quattro categorie: SISD, SIMD, MISD, MIMD (figura 4.1).

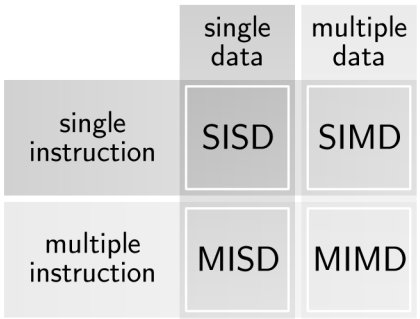


Figura 4.1 – Tassonomia di Flynn basata sui dati e sulle istruzioni.

Le architetture parallele appartengono alla categoria *multiple instruction multiple data* perché sono caratterizzate da sistemi che possono eseguire contemporaneamente più istruzioni con dati diversi. A sua volta questa categoria si divide in due sottocategorie basate rispettivamente su architetture a memoria condivisa e memoria distribuita. Alla prima appartengono le macchine formate da una o più unità di calcolo che accedono alla stessa memoria pur eseguendo programmi differenti. La seconda sottocategoria comprende tutte le macchine che sono composte da più nuclei di calcolo (nodi) con una propria memoria riservata. L'accesso di un nodo alla memoria riservata di un altro avviene solo attraverso uno scambio di messaggi o tecniche analoghe.

Gli elaboratori più veloci del pianeta si basano sui sistemi a memoria distribuita e in particolare le macchine MMP (*massively parallel processing*), composte da centinaia di processori all'interno dello stesso elaboratore, sono alla base di molti *supercomputer*. Altri esempi di sistemi a memoria distribuita sono le architetture chiamate COW (*cluster of workstations*), basate su più elaboratori connessi da una rete di comunicazione. I claster di computer rientrano in questa categoria.

4.1.1 Cluster di computer

Un cluster di computer rientra nella categoria MIMD anche se si compone di più elaboratori SISD connessi tra loro da una rete di comunicazione. I cluster a loro volta si dividono in tre categorie: *high-availability* (HA), *load-balancing* e *high-performance computing* (HPC).

Gli HA cluster sono conosciuti anche come *failover clusters* perché sono implementati per garantire il funzionamento continuativo di un servizio in presenza si guasti: questi sistemi si basano sul principio della ridondanza. I *load-balancing* cluster sono pensati per distribuire il calcolo su tutte le macchine connesse, spostando le richieste di lavoro su quelle con meno carico.

Gli HPC cluster sono i computer implementati per avere elevate prestazioni. Questi sistemi dividono il processo principale in più parti in modo che ogni nodo le possa eseguire in parallelo. Questa tipologia di cluster viene spesso adoperata per simulazioni scientifiche e utilizza una tipologia di programmi scritti appositamente per sfruttare il parallelismo di queste architetture. Questi programmi sono caratterizzati da una comunicazione attiva durante l'esecuzione e da risultati intermedi che influenzano le future elaborazioni.

4.2 La struttura dell'algoritmo

L'algoritmo genetico multiobiettivo è composto da quattro fasi distinte che si ripetono ad ogni generazione (figura 4.2): per ogni individuo viene calcolato il valore della funzione obiettivo (fitness) che permette di ordinare la popolazione seguendo il criterio di dominanza del fronte di Pareto. Grazie all'ordinamento ottenuto si possono selezionare gli individui migliori e generare una nuova popolazione per mezzo della ricombinazione.

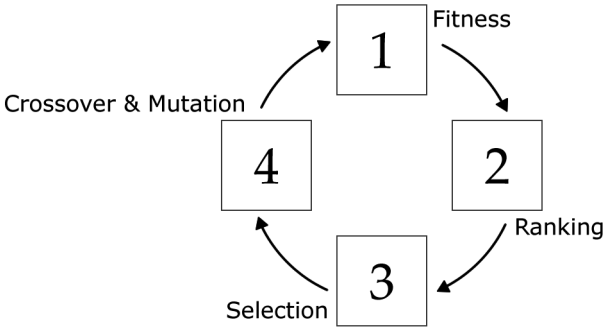


Figura 4.2 – Fasi principali dell'algoritmo genetico multiobiettivo.

Ognuna di queste fasi modifica la popolazione e può essere considerata come un processo indipendente. Da questa osservazione è possibile pensare ad un modello a pipeline dell'algoritmo che permetta di mappare le quattro fasi su altrettanti processori. Bisogna considerare però, che la forte dipendenza dei quattro processi e i vincoli che li legano, rendono questo modello del tutto inutile perché non produce alcun vantaggio significativo. Infatti, ognuna delle quattro fasi deve aspettare i dati della precedente per poter svolgere i propri calcoli. Per esempio per ottenere il ranking della popolazione bisogna conoscere il valore obiettivo di ogni soluzione, oppure per selezionare gli individui migliori, la popolazione deve essere ordinata. In questo modo ogni processore deve aspettare la fase precedente e non si ha alcun vantaggio nella parallelizzazione.

Un strategia migliore del modello a pipeline si chiama *single program multiple data* (SPMD) che prevede l'esecuzione dello stesso algoritmo su più processori con dati differenti. Questa procedura deve essere seguita da una fase dove i processori si scambiano i risultati ottenuti per verificare che non ci siano conflitti o problemi tra le soluzioni. Se quest'ultimo aspetto risulta efficiente, ovvero, se si ha una comunicazione veloce tra i processori, il metodo presenta evidenti vantaggi in termini di prestazioni perché l'esecuzione dei processi avviene in parallelo.

4.2.1 Calcolo della fitness

Il calcolo della fitness ha un elevato costo computazione perché gli algoritmi genetici multiobiettivo risolvono problemi che hanno uno spazio di ricerca esteso e multi-dimensionale. Ogni obiettivo ha la propria funzione che ricava il valore della fitness delle soluzioni. Il numero di variabili n in ingresso alla funzione è dato dalla dimensione dello spazio di ricerca, mentre il numero di funzioni obiettivo m è dato dalla dimensione dello spazio delle soluzioni. Pertanto, il calcolo della fitness di un soluzione si ottiene eseguendo m funzioni con n variabili in ingesso. Se consideriamo che gli algoritmi genetici, e in particolar modo quelli multiobiettivo utilizzano un numero elevato di individui, ci rendiamo conto che il calcolo della fitness influenza pesantemente le prestazioni.

Il calcolo della fitness su più obiettivi si presta facilmente ad un approccio parallelo. Si può pensare infatti a due metodi: il primo associa ad ogni processore il calcolo di una singola fitness (figura 4.3a), il secondo assegna una parte della popolazione ad un processore e calcola, per ogni individuo, tutte le funzioni fitness (figura 4.3b). Il vantaggio di questi due criteri è immediato, infatti, risulta evidente il miglioramento dei tempi di esecuzione che si ottiene con il calcolo in parallelo.

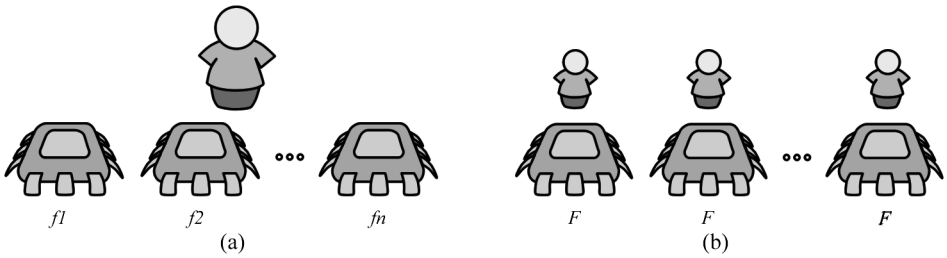


Figura 4.3 – Possibili approcci paralleli del calcolo della fitness.

La tecnica che calcola una sola funzione di fitness per processore può presentare degli inconvenienti se le funzioni obiettivo hanno una diversa complessità e se il tempo di calcolo è differente. Per cui, se non si ha un carico bilanciato, può accadere che un processore si trovi nella condizione di dover aspettare un altro che non ha ancora finito il calcolo della fitness. Per risolvere questo problema si può pensare ad una soluzione dove ogni processore calcola ancora una sola funzione di fitness alla volta, con la particolarità che questa funzione cambia da un individuo all'altro. In questo modo i processori utilizzano tutte le funzioni obiettivo (una sola per soluzione) e il carico è bilanciato. Nella tecnica a divisione della popolazione questo problema non sussiste.

Entrambi i metodi presentano un problema legato al costo della comunicazione perché i valori obiettivo che ogni processore calcola devono essere trasmesse a tutti gli altri. Quindi, sia nel caso della divisione della popolazione, sia nella divisione delle funzioni obiettivo, ogni processore deve comunicare i proprio risultati in previsione della fase successiva: il ranking.

4.2.2 Ranking

Il valore della fitness viene utilizzato per ordinare gli individui ed individuare le soluzioni migliori basandosi sul concetto di dominanza per ottenere il fronte ottimo di Pareto. Nel caso particolare dell'algoritmo NSGA II di Deb [3] la popolazione viene ordinata in fronti non dominati e all'interno di ogni fronte disposta seguendo la crowding distance. Il ranking basato sul concetto di dominanza richiede un confronto diretto di tutti gli individui su ogni valore obiettivo, con una complessità dell' ordine di $O(mn^2)$. Questa complessità può essere abbassata utilizzando versioni perfezionate dell'algoritmo, come quella proposta da Jensen [10] che arriva a $O(nlog^{m-1}n)$.

Comunque, alla base di tutti gli algoritmi di ordinamento c'è un confronto diretto tra tutti i cromosomi e i loro valori fitness, per cui, per poter ottenere il ranking si deve avere la possibilità di accedere all'intera popolazione. Questo aspetto contrasta con ogni principio di parallelizzazione e rende molto complessa la distribuzione dell'algoritmo su processori diversi.

Infine si osserva che il tempo di esecuzione del ranking, per quanto elevato, risulta sempre trascurabile se paragonato al calcolo della fitness. Infatti assumiamo che il tempo per ordinare la popolazione sia minore dell'esecuzione delle funzioni. Comunque vedremo successivamente che distribuendo il calcolo della fitness siamo costretti a distribuire anche il ranking.

4.2.3 Selezione

La selezione si basa sulla fase precedente: il ranking. Il suo compito è scegliere gli individui migliori per la ricombinazione e per la generazione successiva, per cui il costo di questo operatore è molto basso e non deve esser necessariamente eseguito in parallelo. Comunque è possibile pensare ad una soluzione distribuita, dove ogni processore possa selezionare in maniera autonoma gli individui migliori dalla propria popolazione.

4.2.4 Ricombinazione

La ricombinazione genera nuove soluzione, grazie al meccanismo combinato del crossover e della mutazione, partendo dagli individui selezionati nella fase precedente. Il crossover e la mutazione hanno alla base dei calcoli molto semplici, composti da poche operazioni con un costo contenuto. Tuttavia non si ha alcun problema se ogni processore si occupa della ricombinazione di una sola parte della popolazione perché questa avviene in genere tra coppie di individui e non occorre conoscere tutti le soluzioni.

4.3 La parallelizzazione

Abbiamo visto che esistono diversi metodi per ottenere implementazioni parallele. Queste dipendono dalle scelte del programmatore e dall'ambiente di esecuzione dell'algoritmo. Nel nostro caso consideriamo una rete di calcolatori, dove ogni elaboratore ha il proprio processore e la propria memoria (paragrafo 4.1.1). In questo modello la comunicazione tra i processori avviene mediante il meccanismo a scambio di messaggi, per cui il tempo di trasmissione rappresenta un aspetto che può influenzare le prestazioni.

Per ridurre questo tempo è possibile pensare ad un modello a memoria condivisa oppure ad architetture MMP (paragrafo 4.1), anche se queste soluzioni hanno un elevato costo in termini economici. Per questo si preferisce utilizzare un cluster di computer dove la potenza di calcolo si ottiene aumentando solamente il numero di elaboratori, piuttosto che adoprare soluzioni costose come un singolo calcolatore con prestazioni elevate.

Tuttavia la necessità di avere una maggiore potenza di calcolo non deve contrastare con la ricerca delle soluzioni perché rimane il vero obiettivo dell'algoritmo genetico. Infatti è bene tener sempre presente che lo scopo finale è ottenere una popolazione che descriva al meglio il fronte ottimo di Pareto (corretto e ben distribuito). Questo obiettivo si raggiunge grazie all'azione del ranking che privilegia le soluzioni che sono all'interno del fronte, per cui questo operatore ricopre un ruolo determinante ed è necessario tenero in forte considerazione quando si propongono implementazioni parallele.

4.3.1 Dividere la popolazione

Abbiamo visto che la fitness può essere ottenuta associando ai singoli processori il calcolo dei singoli obiettivi o il calcolo di alcune parti della popolazione. Il modello a rete di calcolatori è più indicato per il secondo schema perché ogni processore salva in memoria la sua parte di popolazione, mentre il primo schema presuppone che ogni processore conosca tutti gli individui.

Il primo problema che bisogna affrontare dividendo la popolazione è il calcolo del ranking, che come abbiamo visto rappresenta un elemento fondamentale per ottenere il fronte di Pareto. La soluzione più semplice, ma più costosa in termini di comunicazione, è individuare un processore che riceva tutti i valori obiettivo e calcoli il ranking di tutta la popolazione. Questo meccanismo, che rientra nella categoria *Master-Slave* descritta da Deb [2], introduce un elevato overhead nella comunicazione, infatti costringe i processori *Slave* ad aspettare il calcolo del ranking e quindi a rimanere inutilizzati.

Il modello a isole rappresenta una valida alternativa alla precedente soluzione (paragrafo 8.1). Questo modello divide la popolazione complessiva in sottopopolazioni

(isole) è le associa ai processori. Ognuno di essi esegue un algoritmo in maniera del tutto indipendente e isolata, fatta eccezione per la migrazione che mette in comunicazione le isole. I vantaggi di questo modello sono molti e in particolare non richiede un eccessiva comunicazione tra i processori e quindi è adatto ai cluster di computer o alle reti di computer.

Un problema di questa soluzione si ha quando bisogna scegliere come dividere la popolazione e con quale criterio assegnare le varie parti ai processori. Possibili approcci prevedono di dividere lo spazio di ricerca in modo da assegnare individui che hanno caratteristiche simili. Questo metodo presenta dei problemi, in quanto non garantisce una copertura uniforme del fronte di Pareto, per questo altri studi, come quello proposto da Hiroyasu [13] e da Deb [2], concentrano la loro attenzione sullo spazio delle soluzioni e non sullo spazio di ricerca. Questa tecnica distribuisce in maniera corretta l'insieme non-dominato tra i processori, ma ha lo svantaggio di non conoscere a priori il fronte, per cui la divisione e l'assegnazione della popolazione si deve adattare durante l'esecuzione dell'algoritmo.

A questo punto abbiamo capito che il modello a isole rappresenta la miglior soluzione se la topologia utilizzata è un cluster di computer. Inoltre questo modello deve essere affiancato a considerazioni sullo spazio di ricerca per dividere e distribuire la popolazione in base al fronte di Pareto.

Quest'ultimo aspetto rende molto complessa la scelta delle sottopopolazioni perché è ostacolata dal ranking. Infatti la ricerca del fronte di non-dominanza si ottiene avendo a disposizione l'intera popolazione per poter confrontare tutti gli individui. Per cui secondo questo criterio non è possibile distribuire il calcolo del ranking e quindi utilizzare il modello a isole. Comunque nel prossimo capitolo verrà presentata una tecnica che riesce a suddividere la popolazione in base allo spazio degli obiettivi e nel rispetto del calcolo della non-dominanza.

Capitolo 5
Cono di separazione

Nel 2004 Deb ha proposto una tecnica per parallelizzare un algoritmo genetico chiamata *Cono di Separazione*. Questo tecnica divide lo spazio degli obiettivi in zone a forma di coni o spicchi e le mette in relazione ai processori all'interno del cluster di computer. In questo modo gli individui della popolazione sono associati ai processori in base alla zona che occupano. Questo metodo viene applicato principalmente a problemi con due obiettivi e viene illustrata una possibile estensione a tre obiettivi.

5.1 Il funzionamento

L'algoritmo del cono di separazione individua ad ogni generazione un quadrato unitario che contiene il fronte ottimo di Pareto. Il quadrato viene diviso in spicchi che sono associati ai processori: gli individui contenuti nello stesso cono sono all'interno dello stesso elaboratore. Per ricavare il quadrato si cercano all'interno della popolazione gli individui che hanno la fitness migliore. Questa corrisponde al valore obiettivo minore se stiamo risolvendo problemi di minimo, o viceversa il maggiore, nei problemi di massimo.

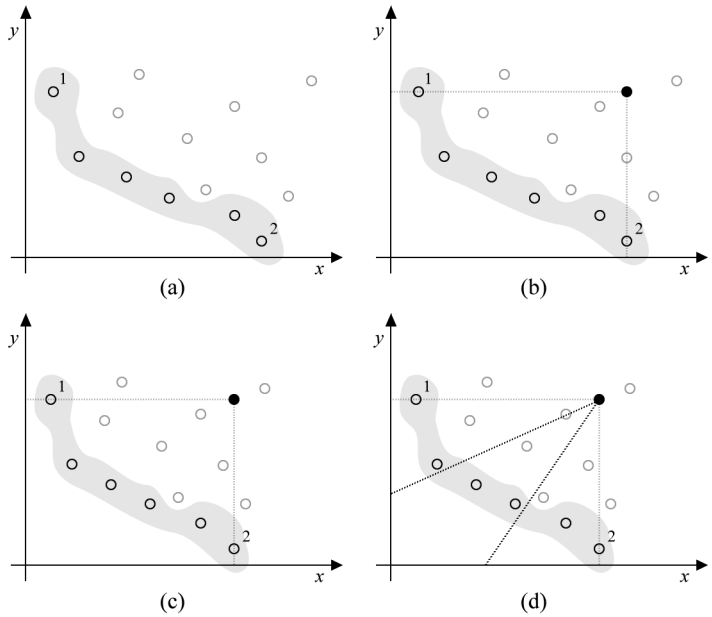


Figura 5.1 – Divisione dello spazio degli obiettivi con il cono di separazione.

Consideriamo l'esempio con le sole due dimensioni x e y in un problema di minimo (figura 5.1). I valori migliori sono i punti all'interno del fronte (zona scura) contrassegnati dai numeri *1* e *2*: il primo punto è il valore minimo rispetto alla variabile x, mentre il secondo e il minimo rispetto alla variabile y (figura 5.1a). Da questi valori è possibile individuare un ulteriore punto, chiamato Nadir [4], che è il vertice in alto a destra del quadrato (figura 5.1b). Per mezzo di questo punto viene normalizzato lo spazio degli obiettivi (figura 5.1c).

Dopo la normalizzazione l'intero fronte di Pareto si trova all'interno di un quadrato unitario. L'area del quadrato viene scomposta in coni dividendo l'angolo di 90° in parti uguali pari al numero di computer nel cluster (figura 5.1d). Per esempio se i processori sono 3, il fronte di Pareto viene diviso con angoli di 30°, per cui il primo processore si occuperà della popolazione compresa tra 0 e 30 gradi, il secondo di quella compresa tra 30 e 60 gradi, mentre il terzo tra 60 e 90 gradi.

Si nota dalla figura che alcuni individui rimango all'esterno del quadrato unitario. Questi punti vengono associati ai processori agli estremi del quadrato che sono il primo o l'ultimo.

5.1.1 La struttura principale

In questo paragrafo viene descritta la struttura dell'algoritmo genetico parallelo, mostrando quali operazioni sono eseguite dai singoli processori e quali informazioni si trasmettono.

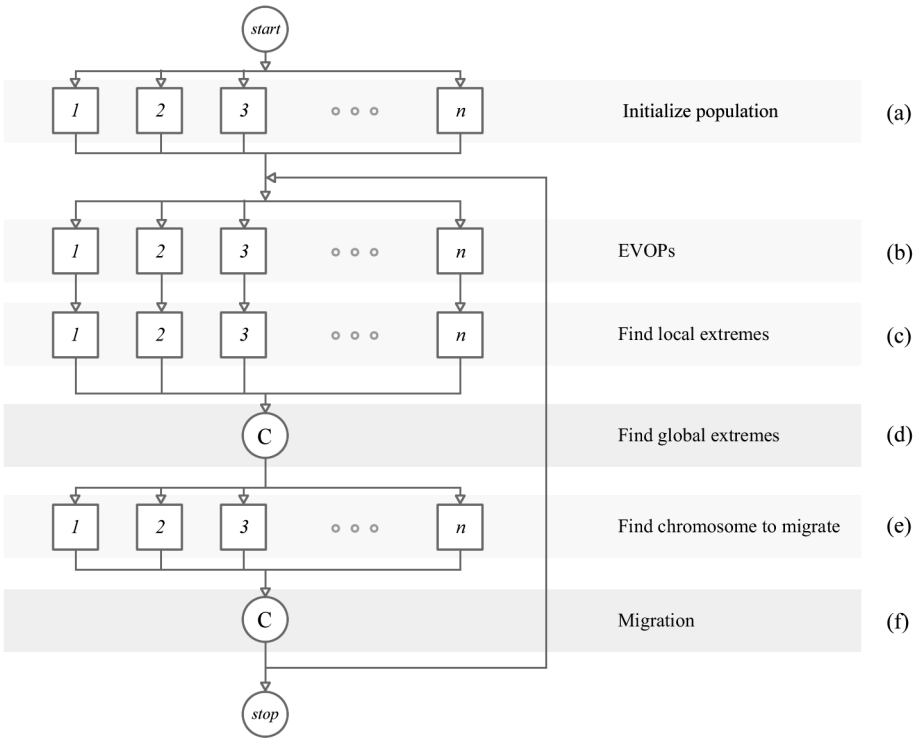


Figura 5.2 – Struttura principale dell'algoritmo genetico multiobiettivo parallelo.

Seguendo il diagramma i processori creano la propria popolazione in maniera casuale all'inizio dell'algoritmo (figura 5.2a). Si passa poi al ciclo principale dove vengono eseguite tutte le operazioni tipiche di un MOGA chiamate EVOPs (Evolutionary Operations) (figura 5.2b), che sono le quattro fasi descritte nel capitolo precedente (figura 4.2): calcolo della fitness, ranking, selezione, mutazione e crossover.

A queste operazioni si aggiunge il calcolo degli estremi sulla popolazione locale (figura 5.2c), che, unito alla comunicazione tra processori (figura 5.2d), permette di individuare gli estremi globali: da questi punti si ottengono le dimensioni e la posizione del quadrato unitario e quindi la grandezza dei coni di separazione.

Dopo il calcolo precedente ogni processore verifica se gli individui della sua popolazione sono all'interno del cono associato e in base a questo decide di migrare le soluzioni che sono all'esterno verso altri processori (figura 5.2e). L'effettiva migrazione avviene nella fase successiva dove i processori si trasmettono gli individui (figura 5.2f).

Infine, l'algoritmo termina quando viene raggiunta una condizione particolare che può essere il numero massimo di generazioni, un livello di performance desiderato oppure un limite temporale.

5.1.2 EVOPs

Nel diagramma precedente si vede che le operazioni evoluzionistiche (EVOPs) sono eseguite completamente in parallelo, seguendo il modello SPMD descritto nel capitolo precedente. In questo modo il calcolo della fitness, il ranking, la selezione e la ricombinazione vengono suddivise tra i processori.

Questo rappresenta un enorme vantaggio in termini di prestazioni perché il calcolo viene distribuito tra gli elaboratori. Inoltre questa tecnica assicura un perfetto bilanciamento dei calcoli della fitness perché rientra nel modello a divisione di popolazione descritto nel precedente capitolo (figura 4.3).

L'algoritmo così presentato sembra essere la miglior soluzione per distribuire il calcolo tra processori, infatti il peso di tutte le operazioni viene diviso e associato in modo equo al cluster di computer. Questa considerazione risulta vera se non si considera il calcolo del ranking, il cui ruolo ricordiamo, è determinante per un risultato finale corretto. Infatti, l'algoritmo presentato esegue il ranking solo su una parte della popolazione per cui questo calcolo è valido solo localmente. Dunque è possibile che una soluzione che appartiene al fronte ottimo di Pareto possa essere dominata da un'altra presente in un altro processore.

Comunque si dimostra che questo problema viene risolto attraverso i coni di separazione, che come vedremo, garantiscono che il ranking locale sia valido anche globalmente.

5.1.3 Gli estremi

Gli estremi della popolazione sono gli individui con miglior fitness. Per trovare questi punti si eseguono due passi: prima si cercano gli estremi locali nelle popolazioni dei singoli processori, poi si confrontano gli estremi trovati per ricavare quelli globali.

Alla fine il risultato si ottiene attraverso la comunicazione e la sincronizzazione dei processori. Infatti ogni computer della rete deve trasmettere gli estremi locali prima di calcolare gli estremi globali. Questo vincolo costringe i processori che sono riusciti ad eseguire i calcoli più velocemente ad aspettare quelli che sono stati più lenti, creando un punto di join tra i rami paralleli dell'algoritmo.

Questa fase può essere fatta scegliendo un computer del cluster che si occupa di ricevere i dati, calcolare gli estremi e poi ricomunicarli a tutti gli altri processori.

5.1.4 La migrazione

La migrazione è lo strumento utilizzato dal cono di separazione per garantire che ogni processore lavori sullo proprio spazio degli obiettivi. Questa operazione avviene in due parti: nella prima si selezionano gli individui che non si trovano all'interno del cono e si indicano il processori di destinazione, nella seconda si mettono in comunicazione i computer per spostare gli individui.

Attraverso gli estremi i processori ricavano il quadrato unitario e il punto R (punto di Nadir). Questo punto è utilizzato per calcolare la posizione degli individui all'interno dello spazio degli obiettivi, in relazione ai coni di separazione trovati (figura 5.3a).

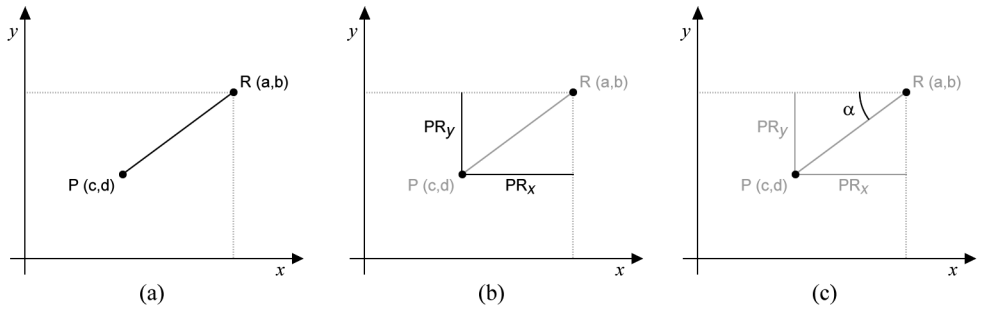


Figura 5.3 – Calcolo dell'angolo di un individuo rispetto al cono di separazione.

Per conoscere in quale processore si trova una soluzione bisogna determinare l'angolo α. Questo angolo si ricava calcolando la distanza x e y tra la soluzione P e il punto R di Nadir (figura 5.3b).

$$\mathrm{PR}_x = |a-c|,\ \mathrm{PR}_y = |b-d| \tag{5.1}$$

PR_x e PR_y rappresentano rispettivamente le componenti x e y del segmento che unisce P ad R. Per calcolare l'angolo α è sufficiente determinare la pendenza della retta che passa per i due punti:

$$\alpha = \arctan\left(\frac{\mathrm{PR}_y}{\mathrm{PR}_x}\right) \tag{5.2}$$

In base ad α e al numero totale di processori si può calcolare in quale cono di separazione o processore si trova la soluzione:

$$processor = \left\lceil \frac{\alpha}{\rho} \right\rceil,\ \rho = \frac{\pi}{2} \cdot \frac{1}{N_{Processor}} \tag{5.3}$$

Per individuare il processore l'angolo α viene diviso per ρ che rappresenta la grandezza di un singolo cono (5.3). Per esempio se il cluster è formato da tre computer si avrà un angolo ρ di $\pi_{/6}$ (30°) corrispondente all'angolo $\pi_{/2}$ diviso 3 (il numero di processori).

Gli individui che si trovano fuori dal quadrato unitario non possono essere associati ad alcun processore utilizzando la precedente formula (5.3), per cui si sceglie di collegarli ai processori più vicini che sono il primo e l'ultimo cono.

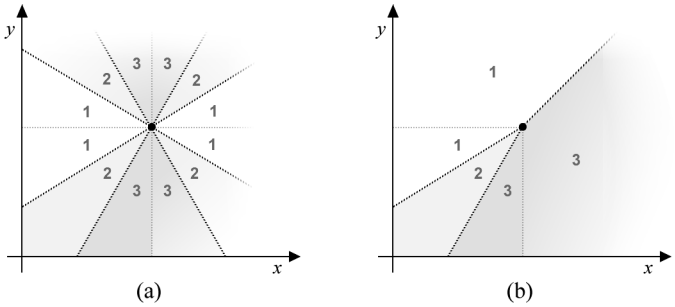


Figura 5.4 – Suddivisione dello spazio degli obiettivi con tre processori.

Nel grafico precedente si mostrano due possibili divisioni dello spazio degli obiettivi: la prima si riconduce alla formula (5.3), mentre la seconda si basa su un criterio di vicinanza delle soluzioni. Si nota che nel primo caso (figura 5.4a) si ha un'errata suddivisione delle soluzioni non supportata da alcun criterio, dove i processori si occupano di aree non adiacenti. Nell'altro caso (figura 5.4a) si presuppone che gli individui fuori dal quadrato unitario contengano informazioni utili anche se sono soluzioni dominate. Di conseguenza questi individui non vengono scartati perché è importante che si possano ricombinare con soluzioni simili, che si presuppone siano quelle più vicine nello spazio degli obiettivi. Seguendo questo criterio il primo e l'ultimo processore si ripartiscono equamente l'area fuori dal quadrato unitario (figura 5.4a).

5.2 Ranking

La garanzia del corretto funzionamento del ranking è data dal metodo con cui il cono di separazione divide lo spazio degli obiettivi. Pertanto è importante soffermarsi su questo aspetto per capire il funzionamento, i vantaggi e gli svantaggi.

5.2.1 La dominanza

La divisione dello spazio degli obiettivi in spicchi non dà alcuna certezza o indicazione sulla dominanza tra individui che appartengono a processori diversi. Dall'esempio in figura è chiaro che il punto B si trova all'interno dell'area di dominanza di A (Figura 5.5 zona scura), per cui entrambi i valori obiettivo della soluzione B sono peggiori della soluzione A (considerando un problema di minimo).

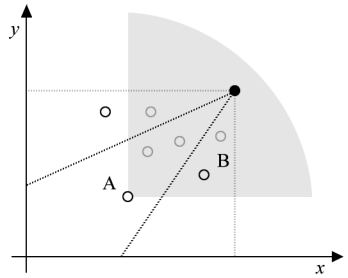


Figura 5.5 – Rapporto di dominanza tra due soluzioni di processori diversi.

Si nota che all'interno di ogni spicchio le due soluzioni A e B sono di rango uno, mentre, considerando l'intero spazio degli obiettivi, la soluzione B è dominata dalla soluzione A e non appartiene al fronte ottimo di Pareto.

Questo difetto può essere trascurato se si presenta nelle prime fasi dell'esecuzione dell'algoritmo genetico; infatti è permesso conservare attraverso le generazioni le soluzioni che non appartengono al primo rango, poiché questo concetto è alla base dell'ordinamento a livelli di non dominanza dell'algoritmo NSGA II [3]. Differente è l'effetto di questo comportamento se si manifesta nelle fasi finali, quando ogni processore ritiene di aver trovato il fronte ottimo e tutte le soluzioni di rango uno.

5.2.2 Il fronte ottimo

Ogni processore cerca nella zona che gli compete il fronte ottimo di Pareto. La tecnica di ricerca che viene utilizzata è simile a problemi con vincoli, dove i vincoli sono i limiti imposti dal cono di separazione. All'interno di ogni spicchio il processore è libero di eseguire l'algoritmo genetico e di trovare le soluzioni ottime del problema.

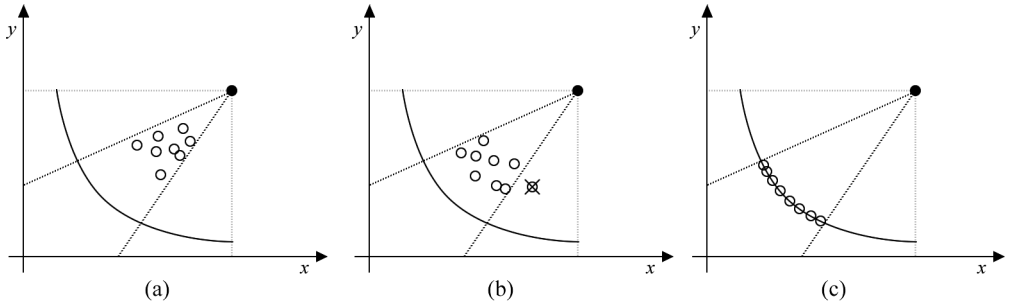


Figura 5.6 – Evoluzione dell'algoritmo all'interno del cono.

Nella sequenza (figura 5.6) viene mostrata l'evoluzione della popolazione di un singolo processore. Le nuove soluzioni generate dal crossover o dalla mutazione che cadono fuori dall'area migrano in un altro cono, per cui il processore elimina dalla propria popolazione questi individui (figura 5.6b). Infine, con il passare delle generazioni, le soluzioni raggiungono il fronte ottimo locale, dove, come fronte ottimo locale si considera l'insieme delle soluzioni all'interno dell'area delimitata dal cono di separazione che si trovano sul fonte ottimo di Pareto.

Il fronte globale si ottiene unendo i fronti locali di ogni cono e rappresenta la soluzione generale del problema. Alle volte questa soluzione non è corretta perché le discontinuità del fronte possono causare problemi alla tecnica dei coni di separazione. In questi casi non viene più generato un fronte ottimo, ma un fronte che contiene delle soluzioni dominate.

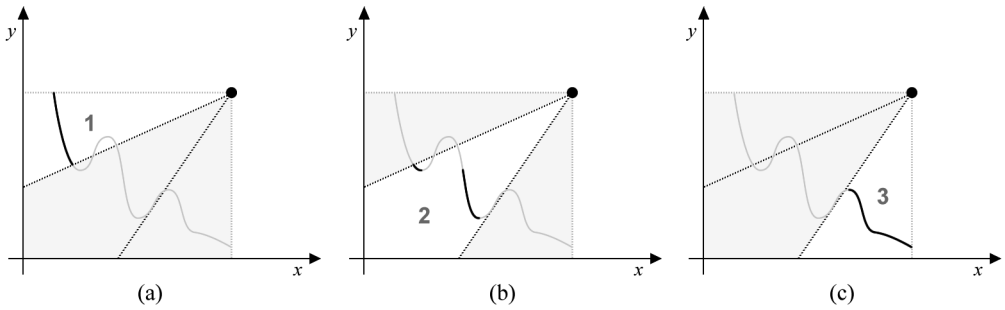


Figura 5.7 – Fronti ottimi di Pareto locali calcolati da tre processori.

Ogni grafico (figura 5.7) rappresenta il calcolo locale del fronte ottimo di Pareto all'interno di un cono. Si nota che il processore numero 3 non essendo a conoscenza del fronte del processore 2 ottiene un fronte errato con individui che nella soluzione finale sono di rango maggiore di uno (figura 5.7c).

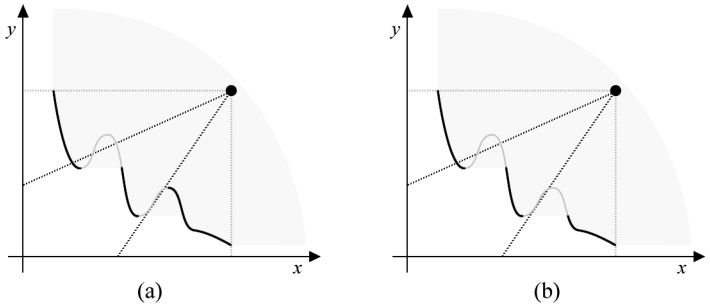


Figura 5.8 – Fronte ottimo di Pareto errato (a) e corretto (b).

Questa condizione è ben riassunta nel precedente grafico dove sono raffigurati il fronte corretto (figura 5.8b) e il fronte errato (figura 5.8a), ottenuto come l'unione delle soluzioni dei tre coni precedenti (figura 5.7). Esaminando l'area di dominanza del fronte corretto (zona scura) si osserva che parte delle soluzioni del terzo processore sono dominate dagli individui del secondo processore, per cui l'algoritmo del cono di separazione ottiene un risultato non ottimo.

Questo inconveniente si presenta in situazioni particolari dove il fronte non è continuo e i processori non conoscono esattamente l'area di dominanza del fronte globale (figura 5.8 zona scura). Analizzando meglio la natura di questi fronti ci accorgiamo che il problema in questione si manifesta quando il confine tra i coni interseca la discontinuità del fronte. Questa può essere rappresentata con segmenti che congiungono le soluzioni ottime che non sono adiacenti (figura 5.9).

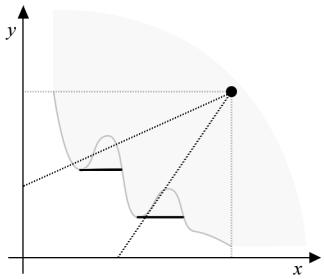


Figura 5.9 – Il confine tra il secondo e il terzo cono interseca un segmento di discontinuità.

Il difetto sulla dominanza è un errore perché alla fine dell'esecuzione dell'algoritmo ci sono soluzioni non ottime. Questo errore può essere trasformato in un risultato corretto ripetendo il ranking su tutta la popolazione così da eliminare gli individui che non sono di rango uno. Tuttavia questa tecnica deteriora le prestazioni dell'algoritmo, infatti, il fronte finale è formato da un numero di soluzioni minore e quindi i processori danno un contributo più basso alla risoluzione del problema. Per esempio se al termine

dell'algoritmo un processore ha il 40% di soluzioni dominate vuol dire che la sua resa è stata del 60% rispetto alla risorse utilizzate.

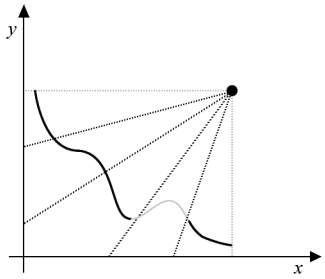


Figura 5.10 – Cono di separazione dentro la discontinuità del fronte ottimo di Pareto.

Il caso limite si ha quando il cono di separazione è contenuto interamente in un segmento di discontinuità. In questa circostanza tutte le soluzioni del cono sono dominate al punto che rende superfluo l'utilizzo di quel processore (figura 5.10).

5.2.3 La distribuzione delle soluzioni

Il ranking utilizzato dall'algoritmo NSGA II [3] è composto da due fasi distinte: l'ordinamento sui fronti e la crowding distance. Il calcolo della crowding distance garantisce che le soluzioni siano disposte in maniera uniforme sul fronte. Questo operazione viene eseguita da ogni processore sulla propria popolazione, per cui non c'è alcuna garanzia della corretta distribuzione delle soluzioni al livello globale, ma solo localmente all'interno di un cono. Può accadere infatti che alcune zone del fronte siano descritte da un numero elevato di individui, mentre altre da un numero basso.

Questo difetto si manifesta principalmente quando i fronti sono discontinui, causando una maggiore densità di copertura in alcune aree. Anche in questo caso bisogna fare delle distinzioni sulla forma dei fronti e dei coni perché esistono delle situazioni per cui questo effetto è più o meno evidente.

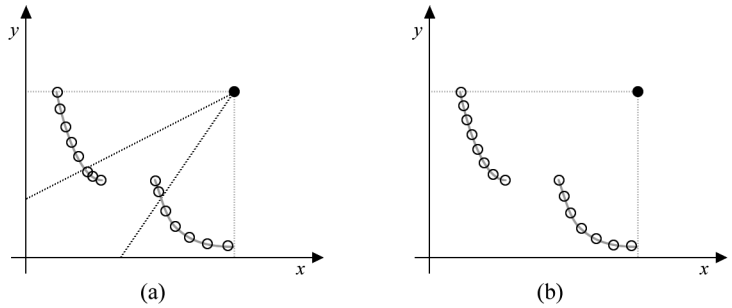


Figura 5.11 – Distribuzione non uniforme degli individui sul fronte.

Il grafico precedente (figura 5.11a) presenta un esempio di distribuzione non uniforme degli individui. L'algoritmo impone che ogni processore abbia lo stesso numero di individui per garantire il bilanciamento del carico di lavoro. Per cui, se all'interno di un cono il fronte ottimo è meno esteso perché in quella area è presente una discontinuità, si avrà una maggiore densità di soluzioni.

Nell'esempio ogni processore ha 5 individui e il fronte è discontinuo nel secondo cono (figura 5.11a). Le soluzioni in quest'area sono più ravvicinate perché il fronte è più piccolo, infatti si nota che gli individui si posizionano su una superficie minore, mentre nel resto del quadrato unitario le soluzioni sono più distanti. Chiaramente questa disposizione degli individui non è la migliore che si può ottenere perché non descrive al meglio il fronte ottimo di Pareto. La sistemazione ottima delle soluzioni si ha calcolando la crowding distance su tutta la popolazione, come si vede nell'altro grafico, dove le soluzioni sono ben distribuite (figura 5.11b).

Una cattiva distribuzione non si ha solamente quando il fronte non è continuo, delle volte dipende dal problema stesso, dalla forma del fronte e dai coni di separazione.

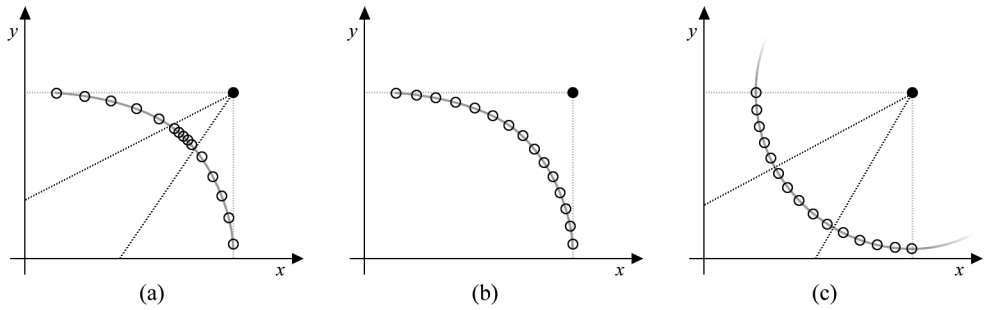


Figura 5.12 – Fronte ottimo continuo con soluzioni distribuite uniformemente (b,c) e non (a).

Infatti i coni di separazione separano l'area del quadrato unitario in base alla suddivisione dell'angolo. Questa tecnica funziona correttamente se la forma del fronte si avvicina ad un quarto di circonferenza centrata nel punto di Nadir (figura 5.12c). L'effetto opposto si ha se la forma è molto diversa come nel primo grafico (figura 5.12a), dove il cono centrale ha soluzioni molto concentrate su una piccola parte del fronte. In questo caso la soluzione ottima è data dal grafico accanto (figura 5.12b) che si ricava senza parallelizare il calcolo della crowding distance.

5.3 L'evoluzione dell'algoritmo

Il quadrato unitario che racchiude i coni di separazione viene calcolato con gli estremi contenuti nel fronte di rango uno. La dimensione e la posizione del quadrato cambia con il passare delle generazioni e di conseguenza i coni contengono aree differenti dello spazio degli obiettivi. Nelle fasi iniziali dell'algoritmo il quadrato è soggetto a grandi cambiamenti perché la popolazione non conosce ancora la forma definiva del fronte ottimo di Pareto. Questi cambiamenti si riflettono sulla migrazione degli individui, infatti, se cambia la forma del quadrato cambiano anche le posizioni dei coni e le aree che delimitano, per cui risulta molto probabile che alcune soluzioni si spostino da un processore all'altro.

5.3.1 La popolazione dei processori

Il processo di migrazione associa gli individui al processore corretto spostando le soluzioni che si trovano nei coni errati. Gli individui di un processore che sono in aree che appartengono ad altri processori vengono spostati verso questi ultimi. Alla base di questo meccanismo ci sono due aspetti che provocano la migrazione: la ricombinazione e lo spostamento del fronte.

Nel primo caso, le operazioni di crossover e mutazione generano delle soluzioni che sconfinano nel cono di un altro processore. In genere il crossover crea degli individui simili ai genitori, per cui per invadere l'area di un altro processore i genitori si devono trovar nelle vicinanze del confine del cono. Diverso è il comportamento della mutazione, che per sua natura può generare soluzioni molto differenti e con buona probabilità fuori dal cono.

In conclusione, la migrazione generata dagli operatori di ricombinazione interessa una minima parte della popolazione del processore, perché la probabilità di mutazione è bassa (1 %) e il crossover che ci interessa avviene solo tra gli individui di confine.

Nel secondo caso, il fronte di ottimo di Pareto cambia durante le generazioni e con lui cambia anche la posizione del quadrato unitario e dei coni di separazione. La migrazione delle soluzioni avviene perché si trovano in coni sbagliati dopo che l'aree associate ai processore sono cambiate.

Questo fenomeno dipende fortemente dall'evoluzione nelle generazioni dell'algoritmo e dalla natura del problema da risolvere. Spesso il cambiamento della forma del fronte modifica così radicalmente la disposizione delle soluzioni che si possono verificare dei fenomeni non attesi.

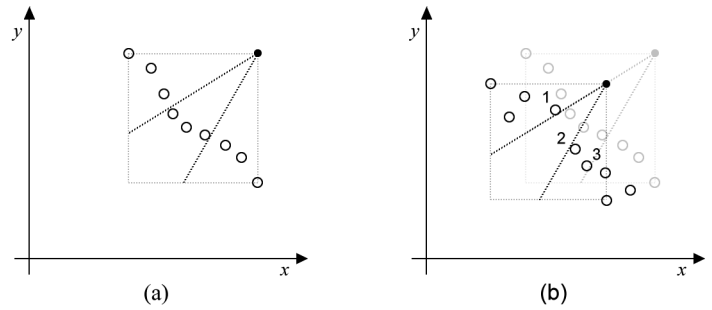


Figura 5.13 – Cambiamento del fronte e spostamento del quadrato unitario.

Nei grafici precedenti viene mostrata l'evoluzione delle soluzioni e lo spostamento del quadrato unitario (figura 5.13). Questo cambiamento provoca la migrazione di tutte le soluzioni dal cono centrale (punti numerati di figura 5.13b), rendendo inutile l'utilizzo del secondo processore che rimarrà senza popolazione fino a quando un individuo migrerà al suo interno.

L'esempio appena descritto è un caso estremo di migrazione totale. Nella maggior parte dei casi questo fenomeno interessa solo una parte della popolazione ed importante considerare gli effetti che introduce nell'algoritmo.

Ogni volta che la popolazione diminuisce, il crossover si occupa di ristabilire il numero di individui iniziale. Per fare questo ogni coppia di genitori genera un numero di figli tale da raggiungere la quota corretta. Per esempio, se ci sono 5 coppie di genitori (10 individui) e la popolazione deve essere composta da 20 individui, ogni coppia genererà 4 figli (4*5 = 20). Questo meccanismo di ripopolazione presenta degli svantaggi quando il numero di genitori è molto basso e quindi poche coppie devono generare molti figli, che alla fine saranno molto simili tra di loro. In questo modo viene meno un concetto base degli algoritmi genetici che prevede una popolazione molto varia con cui esplorare al meglio lo spazio di ricerca.

Si nota come, l'effetto causato dalla migrazione, sia completamente assente negli algoritmi classici non distribuiti, dove il numero di individui rimane costante nelle generazioni. Inoltre, se paragoniamo la migrazione per ricombinazione con quella causata dal cambiamento del fronte, si evidenzia in questa ultima un maggiore spostamento degli individui. Questo fenomeno deteriora le prestazioni visto l'alto costo della comunicazione tra i processori.

Capitolo 6
Microcono di separazione

L'algoritmo del cono di separazione ha evidenti vantaggi in termini di prestazioni e di semplicità. È strutturato in modo che tutte le operazioni evoluzionistiche avvengano in parallelo, per cui aumentando il numero di processori le prestazioni crescono in modo considerevole. Inoltre, la suddivisione dello spazio degli obiettivi avviene con un criterio molto semplice, di facile realizzazione e con un piccolissimo impatto in termini di prestazioni. Infatti, per individuare i coni di separazione è sufficiente che i processori si trasmettano solamente i punti estremi del fronte ottimo di Pareto.

Nel capitolo precedente è stato illustrato come i coni di separazione rendono globale il ranking locale dei processori e sono stati analizzati anche i casi di errore. In questo capitolo viene ripreso lo stesso concetto che sta alla base dei coni di separazione per essere utilizzato all'interno di un nuovo metodo di divisione dello spazio degli obiettivi.

In pratica si creano un numero elevato di coni, chiamati microconi, che suddividono lo spazio in base all'angolo all'interno quadrato unitario. Questi microconi sono associati in maniera uniforme ai processori.

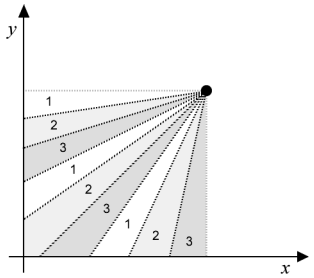


Figura 6.1 – Divisione del quadrato unitario in microconi con tre processori.

Nell'esempio di figura 6.1 abbiamo tre processori con tre microconi ciascuno. Lo spazio degli obiettivi viene diviso per cui in nove zone che assegnate in maniera omogenea ai corrispettivi proprietari.

L'intento di questa tecnica è quello di risolvere i due problemi che si possono presentare durante la fase di ranking dell'algoritmo *cone separation*. Il primo è legato alla presenza di fronti discontinui durante il calcolo della dominanza, mentre il secondo riguarda la distribuzione delle soluzioni sul fronte.

6.1 Il funzionamento

La struttura dell'algoritmo é identica alla tecnica del cono di separazione, per cui è caratterizzata da una totale parallelizzazione degli operatori evoluzionistici, dal calcolo degli estremi per individuare il quadrato unitario e i coni di separazione, dalla scelta degli individui da spostare tra i processori e infine dalla migrazione. L'unica differenza tra i due algoritmi è la scelta dei coni di separazione, della loro forma e dei processori da associare. Per cui alla fine questo algoritmo diventa una variane del precedente mirata a migliorarne alcuni aspetti.

6.1.1 La migrazione

La migrazione è direttamente collegata al ranking. La tecnica dei coni di separazione decide quando una soluzione deve essere spostata da un processore all'altro, ma in realtà è uno strumento efficace per calcolare il ranking localmente e fare in modo che si valido globalmente.

Nel capitolo precedente si è visto un esempio di fronte discontinuo che mette in crisi questa tecnica (figura 5.7 e figura 5.8). Questo errore si manifesta perché i processori hanno una conoscenza parziale del fronte, limitata al loro cono di separazione.

L'algoritmo dei microconi di separazione cerca di dare una visione globale del fronte, ma frammentata in diverse zone che possono contenere una o più soluzioni. Se i microconi contengono l'intera popolazione di un processore ritroviamo la tecnica dei coni di separazione, per cui questa variante basata sui microconi è un estensione dell'algoritmo di Deb.

Il primo passo della migrazione è capire se una soluzione di un processore si trova nel microcono corretto. In base al risultato di questo calcolo si decide se migrare o meno l'individuo verso un altro processore.

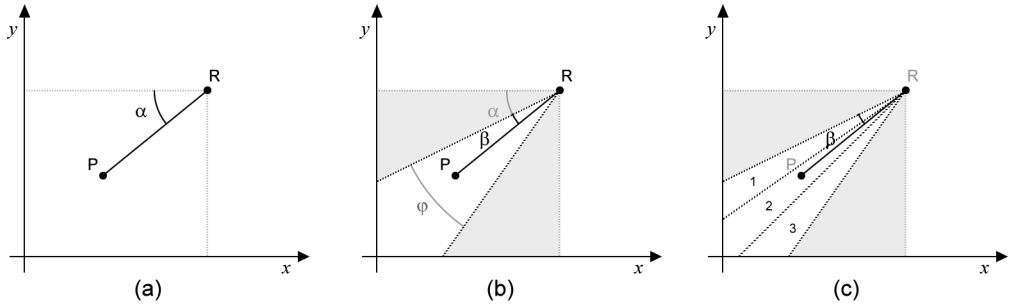


Figura 6.2 – Calcolo dell'angolo di un individuo per determinare il cono corretto.

Prima di tutto si determina l'anglo α della soluzione P rispetto al punto di Nadir R all'interno del quadrato unitario (figura 6.2a):

$$\alpha = \arctan\left(\frac{\mathrm{PR}_y}{\mathrm{PR}_x}\right) \tag{6.1}$$

Questo angolo viene calcolato alla stessa maniera dell'algoritmo dei coni di separazione, è rappresenta la pendenza della retta passante per i punti P e R.

Il quadrato unitario viene suddiviso in gruppi, dove ognuno ha tanti microconi quanti sono i processori. Per determinare la posizione di una soluzione bisogna conoscere l'ampiezza φ e N_{group}, il numero totale di gruppi, in modo da calcare β che è l'angolo all'interno di uno spicchio. Nell'esempio precedente la soluzione P è contenuta nello spicchio che corrisponde al secondo gruppo di microconi (figura 6.2b).

$$\beta = \frac{\pi}{2} - \left\lfloor \frac{\alpha}{\varphi} \right\rfloor, \quad \varphi = \frac{\pi}{2} \cdot \frac{1}{N_{group}} \tag{6.2}$$

In pratica per calcolare β si determina il resto della divisione tra l'angolo α e φ, dove φ si ottiene dividendo l'angolo $\pi_{/2}$ per il numero totale di gruppi.

Tramite l'angolo appena calcolato si ricava il processore che è associato alla soluzione P. Per fare questo si divide l'angolo β per la grandezza di un singolo microcono, corrispondente all'ampiezza del gruppo φ diviso per il numero di processori $N_{processor}$.

$$processor = \left\lceil \frac{\beta}{\rho} \right\rceil, \quad \rho = \frac{\varphi}{N_{processor}} \tag{6.3}$$

Se consideriamo il caso in cui N_{group} sia uguale ad uno, ci si accorge che i microconi diventano i coni di separazione di Deb perché contengono tutti gli individui di un processore. Questo si vede anche dalla formula (6.3) che diventa uguale alla formula (5.3) del capitolo precedente, dove l'angolo β è uguale ad α e φ è uguale a $\pi_{/2}$. Considerando invece il caso opposto, quello con N_{group} uguale $N_{population}$ (la popolazione di un processore), si ha che ogni microcono viene associato ad un solo individuo della popolazione.

6.2 Ranking

I microconi hanno un ruolo importante nel ranking della popolazione perché distribuiscono gli individui sull'intero fronte di Pareto. Abbiamo visto nel capitolo precedente come la parziale conoscenza del fronte provoca in alcuni casi un errato riconoscimento delle soluzioni non dominate. Questo perché la ricerca del fronte ottimo è divisa tra più processori, dove ognuno calcola localmente la sua parte e la soluzione è l'unione di tutti i risultati. Se la divisione dello spazio degli obiettivi nasconde ai processori la forma del fronte ottimo, questi possono generare delle soluzioni che per loro sono corrette, ma in realtà sono dominate.

Sotto questo aspetto i microconi rappresentano un vantaggio, perché permettono ai singoli processori di avere una visione globale del fronte, infatti, gli individui sono distribuiti in maniera uniforme all'interno del quadrato unitario. In questo modo ogni individuo contribuisce a delineare l'area di dominanza che è utile per ottenere il fronte ottimo di Pareto.

6.2.1 La distribuzione

Il punto fondamentale della tecnica di separazione dei microconi è la distribuzione delle soluzioni dei singoli processori. Questo aspetto dipende dal parametro N_{group} che indica quanti microconi possiede un processore e quanto è suddiviso il fronte.

Per capire meglio l'effetto delle soluzioni distribuite, si riprende lo stesso esempio usato per illustrare i limiti del cono di separazione nel capitolo precedente (figura 6.3).

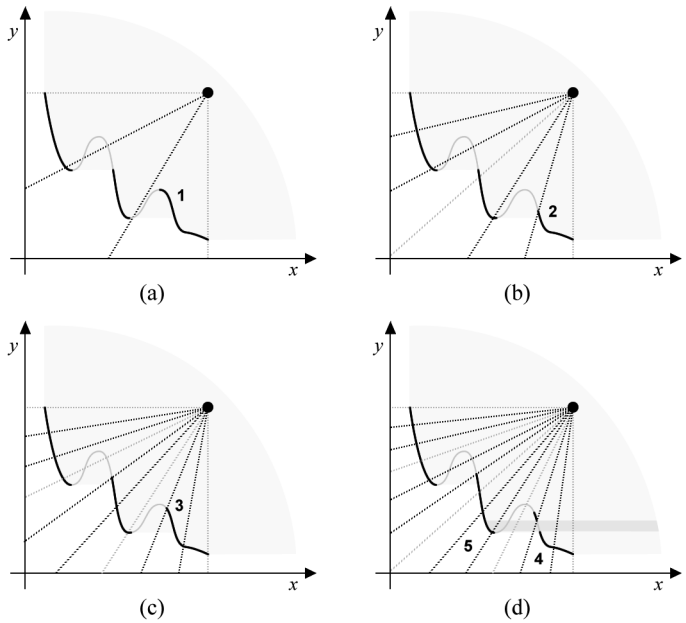


Figura 6.3 – Fronte ottimo di Pareto calcolato al variare del numero di microconi.

Nel primo grafico si nota che il terzo cono calcola un fronte che è in parte dominato dalle soluzioni del secondo processore (figura 6.3a, punto 1). Questo risultato si ottiene utilizzando l'algoritmo dei microconi di separazione con il parametro N_{group} impostato ad uno. Nei due grafici successivi questo parametro viene aumentato a due e a tre, tuttavia l'errore si manifesta ugualmente (figura 6.3b, punto 2 e figura 6.3c, punto 3).

Per osservare l'effetto dei microconi bisogna considerare il quarto grafico con N_{group} impostato a quattro (figura 6.3d). Si nota che il microcono contrassegnato dal numero 4 e dal numero 5 appartengono allo stesso processore. In questo modo gli individui nel microcono 5 individuano un area di dominanza (zona scura) che influenza il calcolo degli individui del punto 4 e permette al processore di generare soluzioni che sono localmente e globalmente non dominate.

Con l'aumentare del numero di microconi diminuisce l'errore del ranking perché ogni singolo processore è in grado di individuare l'intera area di dominanza e di conseguenza il fronte ottimo di Pareto.

6.2.2 La frammentazione

Come abbiamo visto, un numero elevato di microconi assicura una distribuzione degli individui su un'area più vasta del fronte. Lo svantaggio che si ha adottando questa tecnica è legato alla frammentazione delle soluzioni. Infatti, le soluzioni sono sparse su tutto il fronte, ma allo stesso tempo sono raggruppate o frammentate in microconi separati tra di loro. Per cui la popolazione di un processore non è distribuita in maniera omogenea ma è concentrata in particolari aree delimitate dai microconi (figura 6.4a).

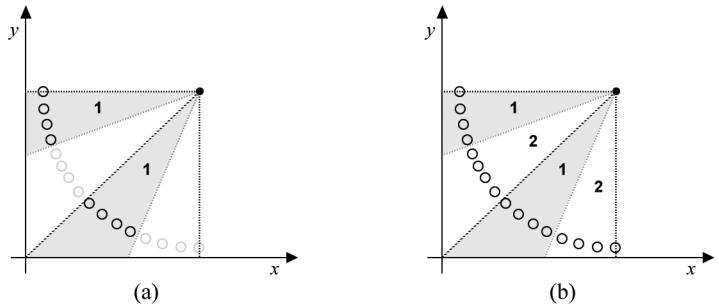


Figura 6.4 – Distribuzione delle soluzioni locali (a) e globali (b).

Questo difetto è solo apparente perché si manifesta quando si considera la popolazione di un singolo processore. In realtà, lo scopo di questo algoritmo, è avere una distribuzione uniforme degli individui considerando la popolazione totale. In questo caso si osserva che gli individui sono disposti in maniera omogenea perché appartengono all'unione di tutti i microconi e quindi a tutta l'area che contiene il fronte di Pareto (figura 6.4b).

Tuttavia se si osserva il comportamento della crowding distance di un singolo processore si accorge che siamo in presenza di un conflitto. Infatti, da una parte abbiamo uno strumento che promuove soluzioni disposte uniformemente e dall'altra un meccanismo che migra gli individui che non si trovano all'interno del microcono.

Questi due operatori entrano in conflitto quando il crossover e la mutazione generano delle soluzioni fuori dai microconi (figura 6.5a). Infatti la crowding distance considera queste soluzioni migliori perché si trovano in zone con pochi individui, mentre l'algoritmo provvede prontamente a migrarle verso un altro processore (figura 6.5c). Questa eventualità non è molto rara perché il crossover può avvenire tra due individui di microconi differenti e non esiste alcuna garanzia che la soluzione generata sia nell'area del processore.

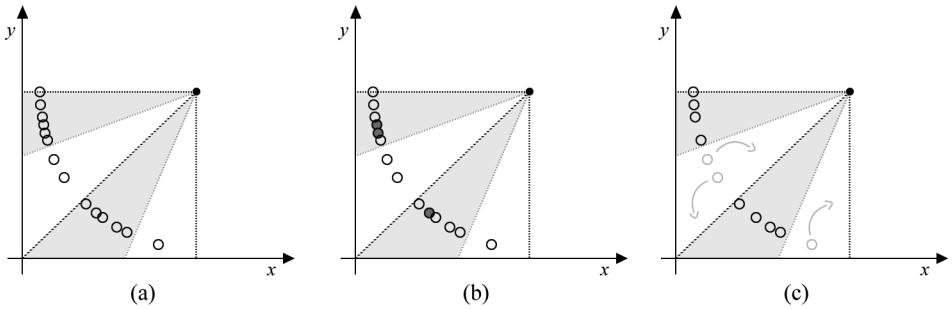


Figura 6.5 – Conflitto tra la crowding distance e la migrazione.

La crowding distance non è a conoscenza dei microconi e valuta le soluzioni all'interno di uno spazio appartenente interamente al processore: l'ordinamento degli individui avviene in base alla distribuzione delle soluzioni considerando tutto lo spazio degli obiettivi. Per questo motivo le soluzioni dentro i microconi hanno maggiore probabilità di essere scartati perché si trovano in aree più dense di individui (figura 6.5b, soluzioni scure). Per evitare il conflitto con la migrazione, la crowding distance dovrebbe occuparsi

della distribuzione delle soluzioni solo all'interno dei microconi e non considerare gli individui all'esterno perché sono trasmessi ad un altro processore durante la successiva migrazione.

In conclusione questo difetto aumenta il fenomeno della migrazione e influenza in maniera negativa il costo della comunicazione che si riflette su un deterioramento delle prestazioni dell'algoritmo.

Capitolo 7
Ranking separation

Nel capitolo precedente è stata introdotta la tecnica dei microconi per suddividere lo spazio degli obiettivi tra i processori e permettere ad ogni processore di avere una visione generale del fronte, con lo scopo di generare soluzioni locali non dominate (paragrafo 5.2.2). Il limite di questa tecnica è legata ai microconi stessi, in quanto frammentano lo spazio degli obiettivi. Infatti, ogni processore ha una visione complessiva del fronte, ma allo stesso tempo non continua. Questa particolare distribuzione degli individui aumenta il numero di migrazioni e annulla l'effetto dell'operatore di *crowding distance* (paragrafo 6.2.2).

In questo capitolo viene presentata una nuova tecnica chiamata *ranking separation* che, a partire dai microconi, cerca di risolvere i difetti che amplificano il fenomeno della migrazione. Comunque questa tecnica non è un'estensione dell'algoritmo *cone separation*, perché si basa su metodi del tutto differenti ricavati da considerazioni sull'operatore di ranking. L'unico elemento in comune è la divisione in coni/microconi che costituisce il principio per separare la popolazione e distribuire il calcolo della dominanza.

7.1 La migrazione

Gli algoritmi genetici multiobiettivo con vincoli utilizzano l'operatore di ranking e selezione per verificare l'ammissibilità delle soluzioni. L'operatore di ranking attribuisce una priorità minore agli individui che non rispettano i vincoli e la selezione sceglie le soluzioni che faranno parte della nuova popolazione in base all'ordinamento del ranking: con questi due strumenti le soluzioni non ammissibili hanno maggiore probabilità di essere scartare rispetto a quelle ammissibili. Questo modo di trattare i vincoli rientra nelle tecniche *soft constrained*, dove è tollerata la presenza di soluzioni non ammissibili all'interno della popolazione. Invece, nelle tecniche *hard constrained* le soluzioni non plausibili vengono scartate subito per non essere propagate attraverso le generazioni.

La migrazione di un singolo processore può essere considerata come un operatore che controlla l'ammissibilità delle soluzioni. Infatti verifica se i vincoli sono rispettati, ovvero se le soluzioni si trovano all'interno dei microconi corretti, in caso contrario le migra verso altri processori.

La migrazione rientra nella categoria delle tecniche *hard constrained* perché le soluzioni inammissibili (fuori dai microconi) sono eliminate dalla popolazione e trasmesse ad un altro processore. L'unica differenza tra la mutazione e un algoritmo con vincoli *hard* è la trasmissione delle soluzioni cancellate.

A questo punto possiamo considerare la migrazione senza lo spostamento degli individui e quindi al pari di una tecnica basata sui vincoli *hard*. In realtà questa assunzione non è del tutto corretta perché lo scambio di informazioni tra i processori ha un ruolo importante nella ricerca del fronte ottimo. Comunque è possibile pensare alla trasmissione come un fenomeno staccato dalla migrazione e considerarlo a parte con un altro operatore (vedi capitolo successivo).

In conclusione, la parte della migrazione che si occupa di verificare l'ammissibilità delle soluzioni può essere implementata con le stesse tecniche di un algoritmo genetico multiobiettivo con vincoli.

7.2 I vincoli

Per verificare che le soluzioni siano all'interno del microcono si è scelto di utilizzare una tecnica del tipo *soft constrained*, invece di una tecnica *hard constrained*. Infatti, la ricerca delle zone ammissibili del fronte può essere molto complessa e un modo per facilitarla è considerare anche le soluzioni non plausibili.

In genere queste tecniche vengono implementate modificando l'operatore di ranking, in modo da dare minore priorità alle soluzioni che si trovano all'esterno dei microconi. Con questo meccanismo si ha la certezza che gli individui convergono verso le aree dello spazio dedicate ai processori. Comunque bisogna avere la certezza che modificando il ranking della popolazione non cambi il funzionamento originale dell'operatore.

Il ranking si ottiene ordinando le soluzioni secondo dei criteri ben precisi: ad ogni soluzione viene associato un parametro che indica il livello di non ammissibilità e in base a questo le soluzioni vengono classificate per sapere quali sono le migliori. Inoltre, bisogna considerare che questo problema non può essere risolto con le normali tecniche con vincoli perché abbiamo una regione di ammissibilità non convessa (Appendice B).

7.3 Ranking

L'operatore di ranking classifica la popolazione per individuare le soluzioni migliori. Questo è equivalente a cercare l'insieme degli individui più simile al fronte ottimo di Pareto. Per raggiungere questo risultato le soluzioni devono seguire due criteri: avvicinarsi il più possibile al fronte ed essere ben distribuite. Per cui, il compito di questo operatore si può riassumere in due azioni: la ricerca del fronte e la distribuzione delle soluzioni. Questi due aspetti sono realizzati all'interno del ranking per mezzo dell'ordinamento della popolazione in fronti non dominati e grazie al calcolo della *crowding distance* (figura 7.1a).

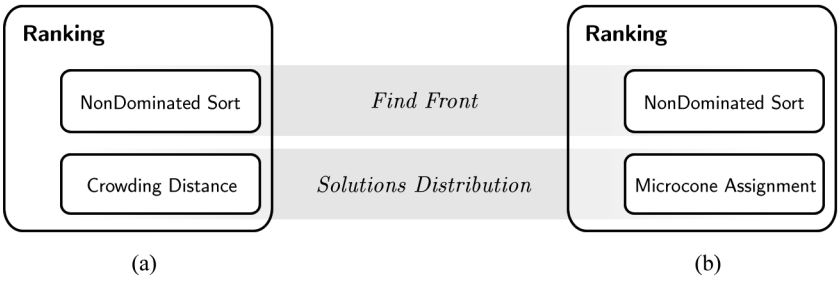


Figura 7.1 – Ricerca del fronte e distribuzione delle soluzioni con e senza microconi.

Quando consideriamo lo spazio degli obiettivi diviso in microconi ci sono alcune aree non plausibili. L'operatore di ranking deve assegnare una minor priorità alle soluzioni che si trovano in queste zone per favorire la disposizione delle soluzioni all'interno dei microconi. Quindi è possibile considerare questa nuova operazione come uno strumento per distribuire gli individui al posto della *crowding distance*.

In conclusione all'interno dell'operatore di ranking la ricerca del fronte è sempre ottenuta con l'ordinamento in fronti non dominati, mentre la distribuzione delle soluzioni viene calcolata con un nuova tecnica chiamata *microcone assignment* (figura 7.1b).

7.3.1 Microcone assignment

La *microcone assignment* influenza la disposizione delle soluzioni per favorire gli individui che sono contenuti nei microconi e la loro distribuzione. Per ottenere questo risultato ogni soluzione viene associata ad un valore che indica il livello di non ammissibilità, che in genere corrisponde alla distanza tra l'individuo e il microcono più vicino. In questo modo è possibile scegliere un insieme di soluzioni che rispetta maggiorente i vincoli perché gli individui sono all'interno dei microconi o si trovano nelle immediate vicinanze. Mentre la distribuzione all'interno dei microconi è ottenuta grazie ad una variante della *crowding distance* che viene calcolata per ogni microcono.

Se classifichiamo le soluzioni con i parametri e i criteri elencati sopra ci accorgiamo che questo ordine non è globalmente valido. In altre parole, non è corretto fare un confronto

tra gli individui dell'intera popolazione perché la distanza dai vincoli e la distribuzione sono valori relativi ai singoli microconi. Si può considerare un esempio (figura 7.2) con due microconi (zona scura) e una popolazione di quattro individui (cerchi). Nel punto (a) viene illustrata la situazione prima della selezione con le soluzioni ordinate secondo l'operatore di ranking. Se fosse utilizzato un ordinamento globale ricavato dall'operatore *microcone assignment* otterremo il risultato mostrato nel grafico centrale (figura 7.2b). Infatti sono selezionate le quattro soluzioni contenute nel secondo microcono (in basso) perché hanno una maggiore priorità rispetto a le altre che si trovano in aree non ammissibili (zone chiare).

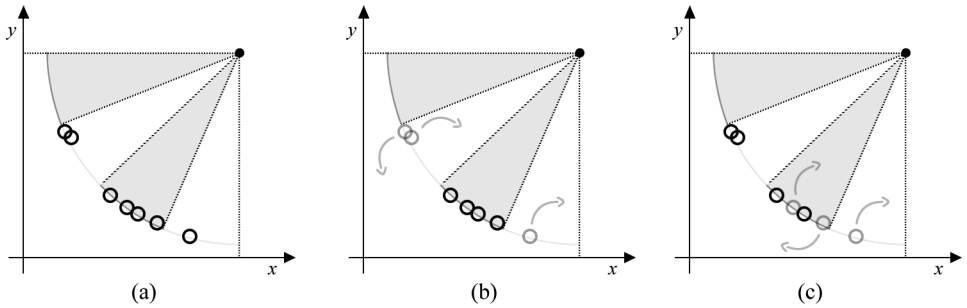


Figura 7.2 – Ordinamento delle soluzioni in base alla microcone assignment.

Anche se a prima vista questa soluzione potrebbe sembrare corretta si nota subito che il primo microcono (in alto) non ha individui, per cui non ci sono soluzioni che descrivono quella parte di fronte.

La tecnica corretta è mostrata nell'ultimo grafico (figura 7.2c). Questo risultato si ottiene confrontando le soluzioni solo all'interno della propria area, per cui non è possibile paragonare individui appartenenti a microconi diversi. Per capire meglio questo metodo facciamo riferimento al grafico sottostante (figura 7.3) dove la popolazione è ancora composta da quattro individui e lo spazio degli obiettivi è diviso tra due processori con due microconi.

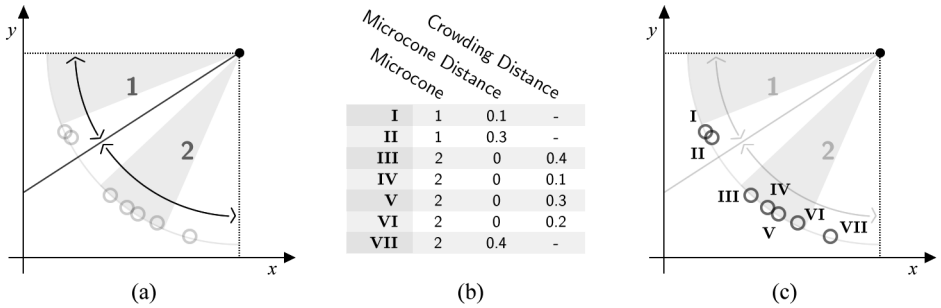


Figura 7.3 – Esempio del funzionamento della microcone assignment.

L'area ammissibile di un processore è formata dai microconi che nel grafico sono rappresentati con due spicchi scuri (figura 7.3a e figura 7.3b). Sempre nel primo grafico vengono evidenziate con le due frecce le zone di influenza dei microconi. La tabella (figura 7.3b) indica per ogni soluzione del grafico tre aspetti: il microcono di appartenenza, la distanza dall'area ammissibile e la *crowding distance*. Se consideriamo per esempio la soluzione *II* del microcono 1 si vede che non rispetta il vincolo perché è ad una distanza 0.3. Mentre la soluzione *VI* si trova dentro il microcono 2, infatti ha distanza 0 e ha un fattore di distribuzione pari a 0.2.

Per classificare le sette soluzioni si crea un ordine in base ai valori della tabella e si ottengono due classifiche, una per microcono. Il primo insieme di soluzioni è formato da [*I,II*], mentre il secondo da [*III,V,IV,VI,VII*]. In questo modo, se l'operatore di selezione deve scegliere 4 soluzioni, verranno prese le prime due da entrambi gli insiemi (figura 7.2c). Se le soluzioni fossero ordinate senza considerare i microconi otterremo un unica classifica e il grafico risultante sarebbe quello mostrato in figura 7.2b.

In conclusione l'operatore *microcone assignment* crea un numero di ordinamenti pari al numero di microconi assegnando ad ogni soluzione una posizione al loro interno.

7.4 L'implementazione

Gli strumenti analizzati nei precedenti paragrafi vengono utilizzati per implementare l'algoritmo chiamato *ranking separation*. Il principio base di questo algoritmo è separare lo spazio degli obiettivi utilizzando l'operatore di ranking al posto della migrazione. Questo è reso possibile grazie allo strumento della *microcone assignment* che favorisce la selezione delle soluzioni all'interno dei microconi.

7.4.1 La struttura principale

La struttura dell'algoritmo *ranking separation* è simile a quella degl'altri algoritmi fatta eccezione per gli operatori di migrazione e di ranking. Dal diagramma si nota che la ricerca delle soluzioni estreme viene anticipata per poter determinare i microconi che sono utilizzati dalla *microcone assignment* (figura 7.4e).

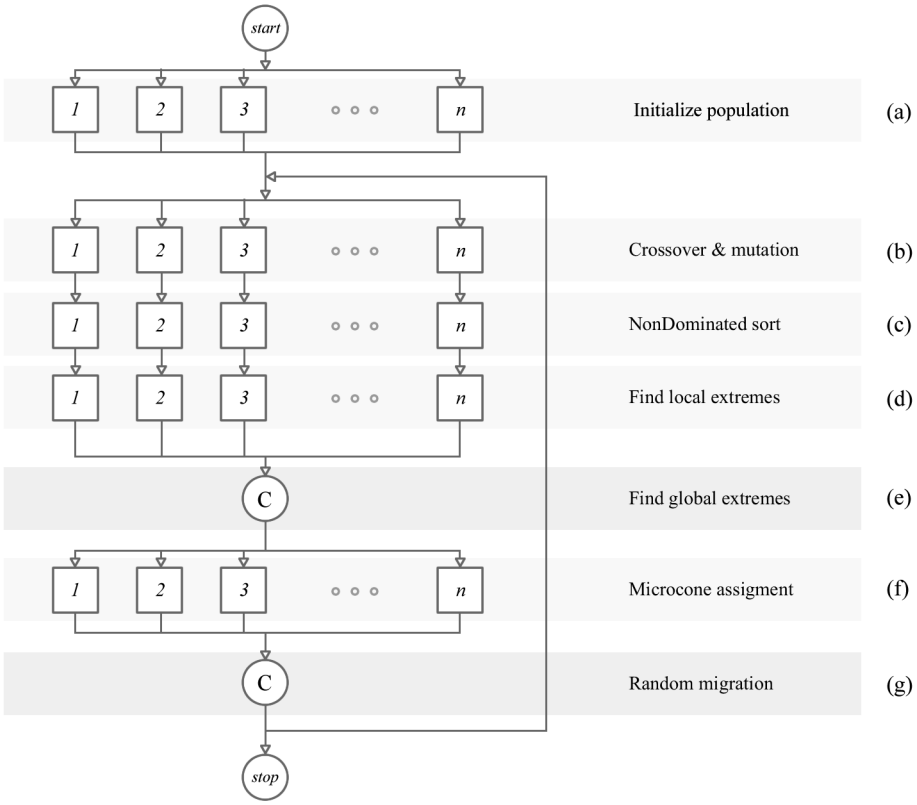


Figura 7.4 – Struttura dell'algoritmo ranking separation.

Infine la migrazione viene sostituta con una variante che sceglie un numero ben preciso di individui in maniera casuale (figura 7.4g).

7.4.2 Microcone assignment

L'operatore della *microcone assigment* viene eseguito quando l'intera popolazione da selezionare è formata da individui che appartengono al fronte di rango uno. Questo perché l'ordinamento viene applicato all'interno di un fronte e non è valido su una parte della popolazione. Questo fenomeno è simile a quello che accade per la *crowding distance* (Appendice A) anche se bisogna considerare un ulteriore aspetto. Sia la *crowding distance* assigment che la *microcone assigment* sono strumenti che distribuiscono le soluzioni lungo il fronte ottimo di Pareto. Questo tipo di operazione ha senso quando l'algoritmo genetico si trova nella fasi finali e il fronte ottimo è stato individuato. Infatti, è inutile distribuire le soluzioni su un fronte non ottimo la cui posizione e forma è del tutto diversa rispetto a

quella finale. Questa considerazione è valida soprattutto per la microcone assigment perché, come vedremo, questo operatore distribuisce le soluzioni in base ai microconi, la cui posizione dipende dal fronte stesso.

Con questo principio l'operatore della *microcone assigment* non entra in funzione nelle generazioni iniziali e nelle generazioni dove il fronte subisce grandi cambiamenti. In queste fasi le soluzioni sono selezionate solo in base ai fronti di non dominanza senza considerare la loro posizione rispetto alle regioni delimitate dai microconi. Infatti queste aree non hanno nessuna attinenza con lo spazio degli obiettivi che i processori occuperanno alla termine dell'algoritmo.

L'operatore della *microcone assigment* si compone di tre fasi: l'assegnazione dei microconi, l'ordinamento dei microconi e l'ordinamento dell'intera popolazione.

Assegnazione dei microconi

Nella prima fase ogni soluzione viene associata al microcono più vicino. Per ottenere questo si calcola la posizione dell'individuo all'interno del quadrato unitario esprimendola con l'angolo α (figura 7.5a).

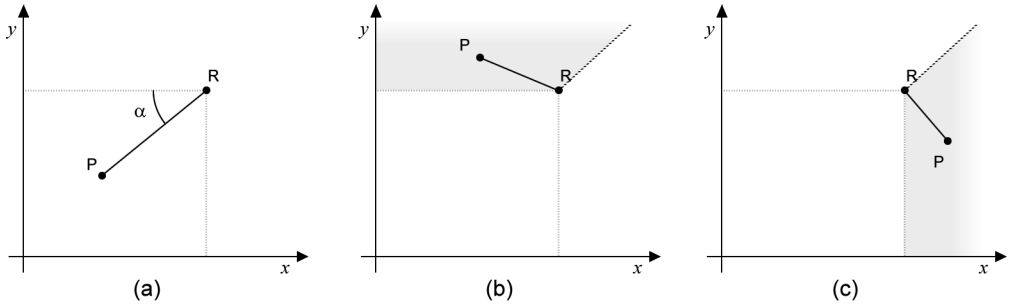


Figura 7.5 – Posizione della soluzione rispetto al quadrato unitario.

L'angolo α è relativo al quadrato unitario e si ottiene considerando il punto di Nadir R con la stessa formula utilizzata negl'algoritmi precedenti:

$$\alpha = \arctan\left(\frac{\mathrm{PR}_y}{\mathrm{PR}_x}\right) \tag{7.1}$$

Questa formula non è sempre valida, infatti, fanno eccezione le soluzioni che si trovano all'esterno del quadrato unitario. Per questi individui non occorre determinare il microcono associato perché dipende se si trovano sopra o sotto la retta passante per il punto di Nadir (figura 7.5b e c). Tale retta è la bisettrice se si considera lo spazio normalizzato rispetto al quadrato unitario.

Per calcolare la distanza tra la soluzione e i microconi bisogna esprimere in qualche modo la loro posizione all'interno del quadrato unitario. In questo caso si è scelto di considerare i punti centrali dei microconi è indicare la posizione con l'angolo relativo alla retta che passa per quei punti.

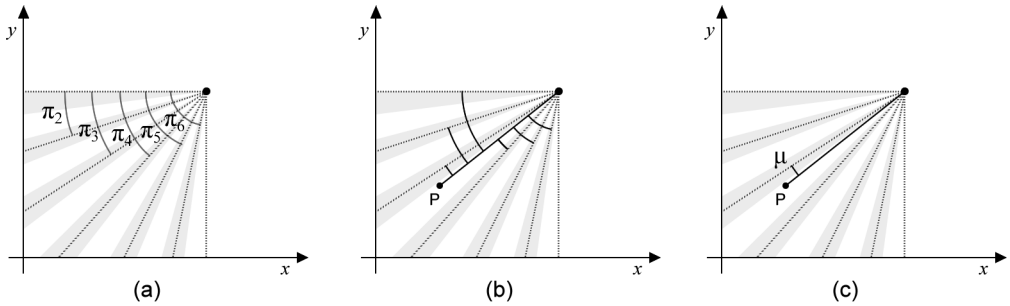


Figura 7.6 – Posizione della soluzione rispetto ai microconi.

Come si vede dal grafico (figura 7.6a) la posizione dei microconi (aree scure) è data dall'angolo compreso con il lato superiore del quadrato unitario. La formula che permette di calcolare questo angolo è la seguente:

$$\pi_i = (i-1)\cdot\varphi + \frac{\rho}{2} \qquad i = 1,2,\ldots,N_{group} \tag{7.2}$$

In questa espressione il parametro N_{group} indica il numero di microconi in un processore, mentre ρ e φ sono rispettivamente l'ampiezza di un microcono e la distanza tra microconi. Questi ultimi parametri si ricavano utilizzando $N_{processor}$ (il numero di processori) e la formula sottostante:

$$\varphi = \frac{\pi}{2}\cdot\frac{1}{N_{group}} \qquad \rho = \frac{\varphi}{N_{processor}} \tag{7.3}$$

Il vettore π viene calcolato considerando il microcono tutto concentrato in unica retta al centro dell'area (figura 7.7 linee scure). Questo è sempre valido fatta eccezione per il primo e l'ultimo microcono che appartengono rispettivamente al processore numero 1 e $N_{processor}$. In questi due casi si considerano il lati del quadrato unitario.

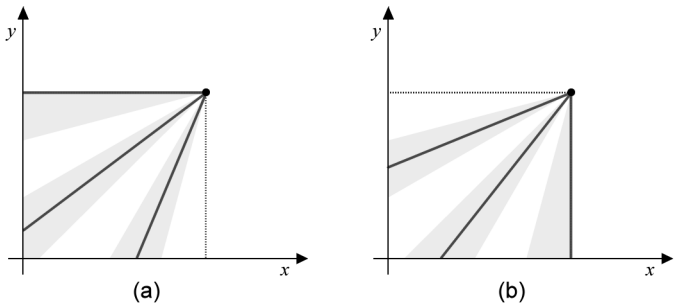


Figura 7.7 – Posizione dei microconi rappresentata da rette.

Prendiamo come esempio la suddivisione nello spazio degli obiettivi rappresentata dal grafico (Figura 7.7). Le linee scure rappresentano i punti utili per calcolare la posizione dei microconi. Per il processore numero 1 e il suo primo microcono questi punto sono il lato superiore del quadrato (Figura 7.7a), mentre per il processore numero 2 ($N_{processor}$) e il suo ultimo microcono (N_{group}) sono il lato destro (Figura 7.7b).

Il vettore delle posizioni π sottratto all'angolo α permette di ricavare la distanza tra la soluzione e i microconi chiamate *microcone distance* (figura 7.6b). Infine, scegliendo il valore μ minore (7.4) possiamo capire qual'è il microcono più vicino e quanto dista dalla soluzione (figura 7.6c).

$$\mu = \min\left(\left|\pi - \alpha\right|\right) \tag{7.4}$$

Come ultima osservazione consideriamo che le formule precedenti non valgono per le soluzioni fuori dal quadrato unitario. In questo caso si assegna una *microcone distance* pari a infinito e la soluzione si associa al primo microcono, se si trova sopra la bisettrice (figura 7.5b), o all'ultimo microcono, se si trova sotto (figura 7.5c).

Ordinamento dei microconi

Nella seconda fase la popolazione viene ordinata all'interno dei microconi. Per prima cosa le soluzioni sono classificate in base alla *microcone distance* e successivamente viene calcolata la *crowding distance* solo per gli individui che sono contenuti nel microcono: per capire quali, si valuta se la *microcone distance* è minore di metà ampiezza (ρ/2) oppure minore dell'ampiezza (ρ), considerando i microconi estremi.

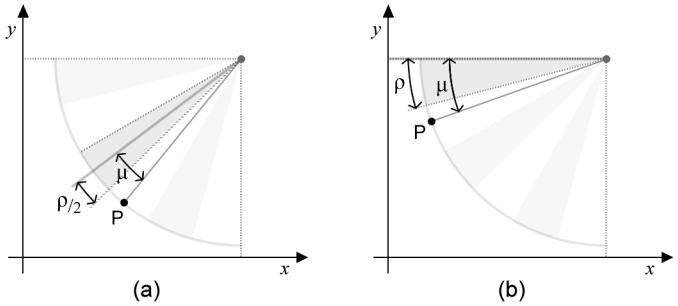


Figura 7.8 – Posizione della soluzione rispetto al microcono.

Come si vede dal grafico (figura 7.8a) le soluzioni si trovano ad una distanza μ dal centro del microcono, per cui, per trovarsi all'interno è sufficiente che tale angolo sia metà dell'ampiezza ρ. Tuttavia, se la distanza della soluzione è calcolata a partire da un'estremità del microcono (figura 7.8b) bisogna considerare l'ampiezza totale ρ e non la sua metà ρ/2.

Una volta identificate tutte le soluzioni contenute nel microcono si procede con il calcolo della *crowding distance assigment*. Il funzionamento di questo operatore è identico alla versione utilizzata nell'algoritmo NSGA II di Deb [3], fatta eccezione per l'uso di due soluzioni speciali. Infatti prima di calcolare la *crowding distance* vengono create due soluzioni che si trovano sulle estremità del microcono. In particolare sono collocate su un punto della retta di confine che è più vicino alle due soluzioni estreme.

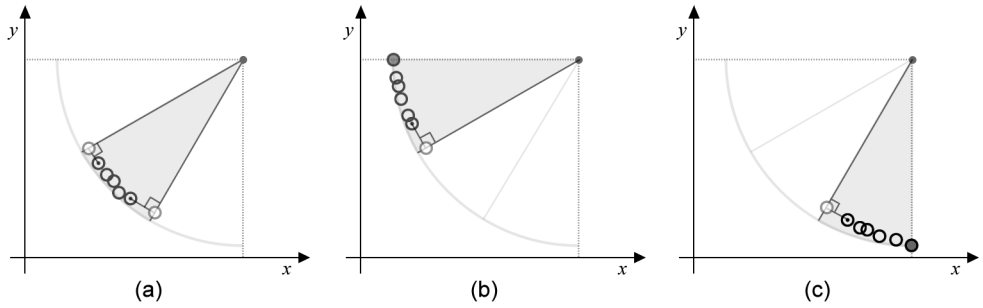


Figura 7.9 – Calcolo della crowding distance insieme alle soluzioni speciali.

Per spiegare meglio questo meccanismo consideriamo le soluzioni classificate secondo il valore di un obiettivo. In questo modo il primo e l'ultimo individuo sono gli estremi raffigurati nel grafico con un punto nel centro (figura 7.9a). A questo punto bisogna individuare l'espressione delle due rette che delimitano l'*i*-esimo microcono preso in considerazione.

Come prima cosa sappiamo che tutte le rette passano per il punto di Nadir che ha coordinate *(1,1)* nello spazio normalizzato. Inoltre conosciamo il coefficiente angolare delle rette perché si ricava dal processore e dal microcono. Prendiamo per esempio che il processore *n*-esimo stia calcolando la *crowding distance* nell'*i*-esimo microcono. Da queste informazioni si può possibile risalire ai due angoli e ai coefficienti angolari m_h e m_l con la formula sottostante:

$$\begin{aligned} m_h &= \tan\left(\left(n-1\right)\cdot\rho+\left(i-1\right)\cdot\varphi\right) \\ m_l &= \tan\left(n\cdot\rho+\left(i-1\right)\cdot\varphi\right) \end{aligned} \tag{7.5}$$

Dai coefficienti angolari e dal punto *(1,1)* si ricavano l'espressioni analitiche delle due rette che delimitano i microconi:

$$\begin{aligned} y_l &= m_l\cdot x_l+\left(1-m_l\right) \\ y_h &= m_h\cdot x_h+\left(1-m_h\right) \end{aligned} \tag{7.6}$$

Per conoscere quale punto di queste rette è più vicino alla soluzione estrema che stiamo considerando si mette a sistema questa retta con un'altra passante per la soluzione stessa e con pendenza perpendicolare al confine (-1/*m*).

$$\begin{cases} y_l = m_l\cdot x_l+\left(1-m_l\right) \\ y_l = -\dfrac{x_l}{m_l}+\left(b_l+\dfrac{a_l}{m_l}\right) \end{cases} \qquad \begin{cases} y_h = m_h\cdot x_h+\left(1-m_h\right) \\ y_h = -\dfrac{x_h}{m_h}+\left(b_h+\dfrac{a_h}{m_h}\right) \end{cases} \tag{7.7}$$

Questo sistema è stato ricavato considerando (a,b) come le coordinate di una soluzione estrema. Risolvendo si ha la posizione delle soluzioni speciali che si trovano sul confine del microcono (figura 7.9a, punti chiari):

$$\begin{cases} x_l = \dfrac{m_l^2-m_l+b_l\cdot m_l+a_l}{m_l^2+1} \\ y_l = -\dfrac{m_l^2-m_l+b_l\cdot m_l+a_l}{m_l\cdot\left(m_l^2+1\right)}+b_l+\dfrac{a_l}{m_l} \end{cases} \qquad \begin{cases} x_h = \dfrac{m_h^2-m_h+b_h\cdot m_h+a_h}{m_h^2+1} \\ y_h = -\dfrac{m_h^2-m_h+b_h\cdot m_h+a_h}{m_h\cdot\left(m_h^2+1\right)}+b_h+\dfrac{a_h}{m_h} \end{cases} \tag{7.8}$$

Le soluzioni di coordinate (x,y) ottenute dal sistema vengono utilizzate all'interno del calcolo della *crowding distance*. Il motivo di questo stratagemma è evitare che le soluzioni si dispongano sul confine, infatti, la *crowding distance assigment* tende a disporre gli individui lungo tutto lo spazio possibile, andando per cui a occupare anche le estremità dei microconi.

Il problema nasce quando si considera che i microconi adiacenti possono disporre entrambi le soluzioni sullo stesso confine. Questo difetto, oltre al caso di individui sovrapposti, rende la distribuzione totale delle soluzioni non più uniforme e genera problemi di instabilità. Infatti, se una soluzione si trova sull'estremità di un microcono, è sufficiente un piccolo cambiamento del quadrato unitario per farla uscire. Di conseguenza si modifica il calcolo della *crowding distance* e della *microcone distance*.

Per questi motivi vengono create due soluzioni sulle estremità dei microconi per evitare che i veri individui della popolazione si avvicinino al confine.

Tutto quello che è stato detto è valido anche per i due microconi situati al limite del quadrato unitario, con la differenza che per questi si deve creare una sola soluzione invece di due (figura 7.9b e c).

Come si vede questi microconi contengono delle soluzioni sul confine che sono importanti per calcolare la forma del quadrato unitario (figura 7.9b, punti scuri pieni). Per questo motivo non viene creata una sola soluzione di confine, perché la presenza di questi individui è essenziale per l'algoritmo e non deve essere ostacolata. La esistenza di queste due soluzioni giustifica anche il particolare trattamento che viene riservato a quei microconi nel calcolo della *microcone distance*.

Ottenuta la *crowding distance* le soluzioni sono classificate con questo criterio: gli individui contenuti nel microcono hanno maggiore priorità degl'altri, all'interno sono classificati con la *crowding distance* e all'esterno con la *microcone distance*.

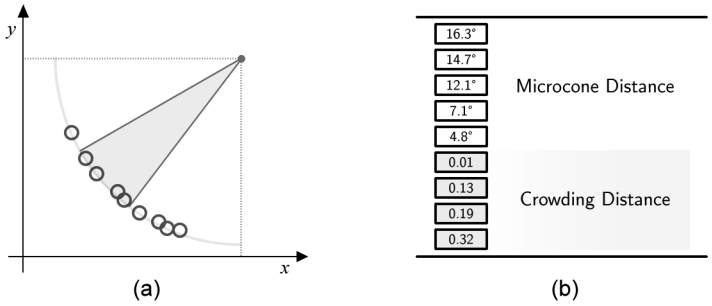


Figura 7.10 – Calcolo delle soluzioni di un microcono.

Come si vede dal grafico abbiamo 9 soluzioni, di cui 5 non ammissibili perché fuori dal microcono (figura 7.10a). L'ordinamento di queste soluzioni avviene in base alla *crowding distance* in ordine decrescente, e con la *microcone distance* in ordine crescente (figura 7.10b). Per ogni microcono del processore viene costruita questa classifica utilizzando le tecniche discusse in questo paragrafo.

Ordinamento della popolazione

L'operatore di ranking ordina le soluzioni in base a dei criteri che favoriscono le soluzioni migliori. I questo modo l'operatore di selezione può scegliere il numero di individui che andranno a formare la popolazione nella nuova generazione.

L'operatore della *microcone assigment* non crea un vero e proprio ordine tra le soluzioni, ma si limita a dividere gli individui in gruppi con diversa priorità. Questo tecnica è alla base del funzionamento di questa fase dove viene ordinata l'intera popolazione di un processore.

Nella fase precedente la popolazione è stata divisa in microconi ordinati seguendo la *crowding distance* e la *microcone distance*. Per come sono stati costruiti questi ordinamenti non è possibile confrontare due soluzioni che si trovano in diversi microconi e per questo non può esistere un ordinamento globale della popolazione.

Quello che si osserva però, è l'esistenza di un ordine che ricorda i fronti di non dominanza nella prima fase del ranking. Infatti si possono considerare le soluzioni migliori

di ogni microcono come appartenenti ad un fronte di rango uno, le seconde migliori appartenenti al rango 2, le terze al rango 3, ecc... A questo punto è ragionevole classificare i fronti in base al rango e considerare che le soluzioni con rango minore hanno più probabilità di essere selezionate. Seguendo questo criterio si crea un ordinamento parziale della popolazione perché le soluzioni all'interno dello stesso fronte hanno la stessa priorità.

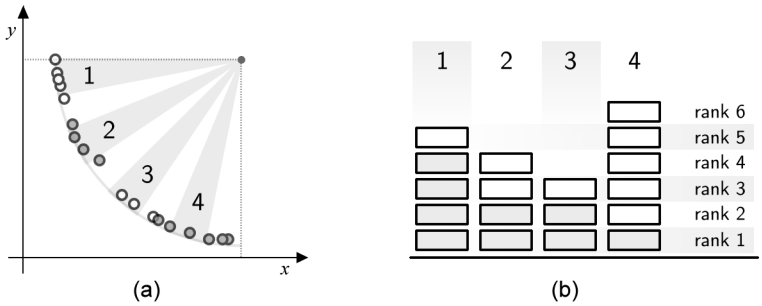


Figura 7.11 – Esempio di ordinamento della popolazione in base alla microcone assigment.

Nel grafico è illustrato un esempio di calcolo della microcone assigment su un processore con quattro microconi (figura 7.11a, zone scure). Nel diagramma accanto si mostra il risultato dell'ordinamento e della divisione in fronti della popolazione (figura 7.11b). I rettangoli scuri rappresentano le soluzioni ammissibili (contenute nei microconi).

Le soluzioni all'interno dello stesso fronte sono equivalenti, per dare un ordine a questi individui si è scelto un meccanismo che assegni una priorità in modo del tutto casuale. In questo modo nessun microcono risulta avvantaggiato.

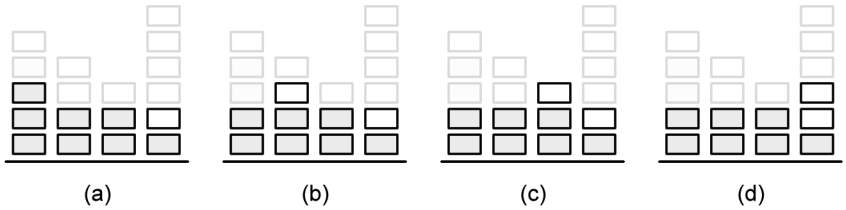


Figura 7.12 – Possibili selezioni di una soluzione in un fronte.

Riprendendo l'esempio precedente consideriamo che la selezione debba scegliere 9 soluzioni. In questo modo abbiamo la certezza che saranno presi gli individui dei primi due fronti, mentre la nona soluzione deve essere scelta a caso nel terzo fronte. In questo esempio ci sono quattro possibili scenari di selezione ognuno dei quali è ugualmente valido. Queste possibilità sono rappresentate nei grafici di figura 7.12.

7.4.3 Migrazione casuale

La migrazione dell'algoritmo *ranking separation* ha il compito di spostare le soluzioni da un processore all'altro. Il numero di individui che migrano è fissato a priori e non dipende dall'esecuzione dell'algoritmo come accade nei coni di separazione.

Il compito di questo operatore è di mettere in comunicazione i singoli processori per ottenere una cooperazione che aiuti a risolvere meglio e più velocemente il problema. Nei coni di separazione la migrazione veniva utilizzata per dividere lo spazio degli obiettivi in coni, mentre nell'algoritmo *ranking separation* questo compito è svolto dall'operatore *microcone assigment*. Quest'ultimo non utilizza la comunicazione tra processori, per cui per ottenere la cooperazione viene utilizzato un operatore di migrazione separato. Inoltre l'operatore viene implementato nel modo più semplice possibile per non alterare eccessivamente il comportamento e le prestazioni dell'algoritmo.

Seguendo questo criterio ogni processore sceglie a caso t soluzioni e le trasmette ai processori in modo *random*: il parametro t non dipende dal funzionamento dell'algoritmo, ma viene scelto dall'utente. Inoltre, la scelta degli individui non segue alcun criterio, come per esempio selezionare le soluzioni migliori, e il processore di destinazione di ogni individuo è puramente casuale. In questo modo la migrazione non facilita ne ostacola la

divisione dello spazio degli obiettivi, ma mette solamente in comune le informazione che hanno i processori.

7.5 Osservazioni

Nei capitoli successivi verranno analizzati e confrontati con dei problemi di test gli algoritmi presentati in questa tesi. Comunque è possibile fare alcune considerazioni sul *ranking separation* già in questo capitolo.

Il primo aspetto che analizziamo è l'ultimo che è stato trattato: la migrazione. I benefici di questo operatore verranno discussi nel capitolo successivo, anche se il contributo dato all'algoritmo è secondario perché rappresenta una parte completamene a sé, aggiunta solamente per recuperare la cooperazione presente nei coni di separazione.

Comunque la forza di questo operatore è data proprio dalla sua indipendenza, infatti il funzionamento non è influenzato da nessun aspetto dell'algoritmo. Questo gli permette di avere un completo controllo sul numero di individui da migrare e sulla quantità di dati da trasmettere. Questa caratteristica risulta molto importante perché rappresenta un aspetto fondamentale che influenza le prestazioni.

All'interno dell'algoritmo *ranking separation* la migrazione trasmette gli individui nel modo più semplice possibile per non alterare il comportamento degl'altri operatori. In realtà si possono pensare a tecniche più avanzate di trasmissione, anche se vanno aldilà degl'obiettivi di questa tesi. Per esempio si potrebbe pesare ad un metodo che modifica il numero di soluzioni in base a qualche criterio. Nell'implementazione presentata in questo capitolo tale numero è fissato a priori e non tiene in considerazione l'effettiva necessità di cooperazione. Può capitare infatti, che un processore abbia una popolazione migliore e convenga che condivida le sue informazioni più degli altri. Per cui dovrebbe esistere un sistema che tiene conto di questo aspetto e decida di aumentare o diminuire il numero di individui da migrare. Una semplice strategia potrebbe valutare per tutte le popolazioni la media delle soluzioni migliori di ogni obiettivo, per poi decidere quale processore deve migrare più individui.

Il secondo aspetto che consideriamo riguarda l'operatore di *microcone assignment* e più precisamente l'ultima caratteristica descritta: l'ordinamento delle soluzioni. Questa permette di realizzare un'unica classifica delle soluzioni partendo dagli ordinamenti interni dei microconi, ottenuti con la *crowding distance* e la *microcone distance*. Alla base di questa tecnica c'è una scelta ciclica delle soluzioni migliori, dove le prime ad essere selezionate sono le migliori dei ogni microcono, poi le seconde migliori, le terze, le quarte, ecc... Seguendo questo procedimento vengono scelti in maniera uniforme i migliori individui di ogni microcono.

I veri vantaggi di questa tecnica di selezione si manifestano in presenza di fronti discontinui. In questi casi i microconi che contengono le zone discontinue non hanno soluzioni e quindi gli individui possono essere ridistribuiti verso gli altri microconi.

Capitolo 8
Separazione casuale

In questo capitolo verrà illustrata una tecnica che viene spesso utilizzata per parallelizzare gli algoritmi genetici a singolo obiettivo. Questa tecnica, conosciuta con il nome di *island model*, si basa su considerazioni legate alla teoria dell'evoluzione naturale degli organismi viventi. Secondo questa teoria la popolazione complessiva è divisa in sottopopolazioni, ognuna delle quali è caratterizzata da un'evoluzione isolata e dalla possibilità di poter comunicare con le altre per mezzo della migrazione. Questo modello viene studiato nel caso dei problemi multiobiettivo e confrontato con gli algoritmi dei capitoli precedenti. In particolare si osserva il ruolo della migrazione all'interno dell'esecuzione dell'algoritmo *ranking separation*.

8.1 Modello a isole

La risoluzione di problemi ad alta complessità richiede l'uso di algoritmi genetici che adottano l'esecuzione parallela per poter aumentare la potenza di calcolo. Una famiglia di queste tecniche prende il nome di modello a isole. Nel modello a isole la popolazione complessiva è divisa in sottopopolazioni chiamate isole. Ogni isola ha un evoluzione indipendente e isolata dalle altre, ed è in grado di comunicare mediante lo scambio di individui. L'effetto che si ottiene con l'interazione delle isole è legato alla migrazione e alla capacità di cooperazione delle sottopopolazioni.

8.1.1 Isole e migrazione

Il modello a isole non è caratterizzato solamente da esecuzioni in parallelo dello stesso algoritmo, ma anche da un'interazione tra le isole stesse attraverso lo scambio di individui. Questa interazione ha un effetto molto importante sul risultato del problema e dipende principalmente da due parametri: il numero di soluzioni trasferite e la frequenza di trasmissione, chiamati anche intervallo e dimensione di migrazione. È stato dimostrato con esperimenti [17] che le prestazioni migliori si ottengono con moderati intervalli di migrazione, combinati con un basso numero di individui trasmessi. Infatti, contrariamente a quanto si possa pensare, migrazioni frequenti o di grandi dimensioni possono deteriorare i risultati e le prestazioni.

Un altro aspetto della migrazione da considerare è la politica che indica quali soluzioni devono essere migrate. Una delle più utilizzate viene chiamata *best-worst policy*, in quanto viene trasmessa la soluzione migliore di un processore per sostituire la peggiore di un altro. La tecnica utilizzata in questa tesi è chiamata *random-random policy* ed è caratterizzata da un criterio di scelta random: una soluzione a caso di un primo processore sostituisce un'altra a caso di un secondo processore. Questo approccio ha la particolarità di avere un'influenza minima sul comportamento dell'algoritmo e sulla selezione degli individui.

8.1.2 Problemi multiobiettivo

Il modello a isole può essere utilizzato per risolvere un famiglia di problemi che sono *linearmente separabili* in sottoproblemi [19]. L'esistenza di questa proprietà permette ad ogni isola di occuparsi di una parte dell'ottimizzazione, infatti, si può considerare che il problema complessivo sia la somma di problemi che si possono risolvere singolarmente.

Questi problemi si adattano perfettamente agli algoritmi genetici a singolo obiettivo paralleli. Comunque, anche in mancanza della separabilità lineare, gli algoritmi genetici ottengono ottimi risultati se il modello a isole viene affiancato dalla teoria del *punctuated equilibrium*. Questa teoria, elaborata da Eldredge e Gould [5] nel campo dell'evoluzione della specie, è caratterizzata da rapidi cambiamenti degli organismi, seguiti da periodi di stasi. Questo si ritrova nel modello a isole perché ogni sottopopolazione ha un evoluzione che è caratterizzata da una velocità variabile: quando le isole comunicano (migrazione) si possono manifestare cambiamenti repentini, mentre nei periodi di isolamento si ha un'evoluzione con una velocità tipica dell'algoritmo. Per ottenere questo effetto occorre impostare i parametri come è stato descritto nel paragrafo precedente. Per cui la dimensione della migrazione deve essere contenuta per garantire una diversità tra le popolazioni e l'intervallo di migrazione deve essere adeguato per avere un'evoluzione isolata.

All'interno degli algoritmi genetici a singolo obiettivo il criterio del *punctuated equilibrium* facilita l'exploration dello spazio di ricerca perché gli intervalli di migrazione moderati (né piccoli, né grandi) garantiscono una popolazione eterogenea.

Negli algoritmi multiobiettivo il ruolo di questo criterio si complica perché ogni individuo della popolazione è connesso con tutti gli altri e l'isolamento delle sottopopolazioni non è più applicabile. Infatti, il valore una soluzione non si ottiene direttamente dalle sue caratteristiche, ma dal rapporto che queste hanno con tutte le altre soluzioni. Per questo motivo il concetto di dominanza contrasta con la parallelizzazione e quindi con tutte le tecniche ad essa connessa.

Comunque, grazie alla divisione dei coni descritta nei capitoli precedenti, si ha uno strumento che permette di distribuire il calcolo della dominanza ed è possibile considerare ancora il *punctuated equilibrium*. Tuttavia, nel contesto multiobiettivo, l'attenzione di questo principio si sposta nello spazio degli obiettivi e si riflette sulla ricerca del fronte ottimo di Pareto.

La competenza di ogni isola è ben definita dalla divisione dei coni ed è caratterizzata da un evoluzione indipendente e isolata. Gli individui che migrano da un cono all'altro non sono in genere considerati dall'isola che li riceve perché provengono da un settore del fronte che non è di sua competenza. Tuttavia può accadere che le isole si evolvano con diverse velocità e che lo scambio di individui possa aiutare quella più lenta. Se la migrazione è molto frequente questa differenza di velocità non si manifesta perché le isole non avuto il tempo di evolversi in maniera isolata. Per questo si preferisce un intervallo di migrazione moderato rispetto ad uno più piccolo e una dimensione di migrazione non eccessiva.

8.2 Random Separation

Per studiare l'effetto della migrazione viene costruita una versione distribuita di NSGA II chiamata algoritmo a *separazione casuale*. Questa implementazione è composta da esecuzioni indipendenti di NSGA II che comunicano attraverso la migrazione. Tuttavia questa comunicazione non segue un preciso criterio, come per esempio la divisione in coni, ma piuttosto una politica di trasmissione casuale data dalla *random-random policy*. In questo modo il calcolo della dominanza e lo spazio degli obiettivi non sono divisi tra i processori e non viene data alcuna garanzia sulla capacità di trovare le soluzioni non-dominate.

Questo algoritmo non produce un fronte ottimo di Pareto e quindi non risolve il problema multiobiettivo. Le soluzioni di un processore sono isolate da tutte le alte e non è possibile determinare se dominano o sono dominate dagli individui di un'altra sottopopolazione. L'unico mezzo con cui le isole possono comunicare è la migrazione che diventa in questo algoritmo l'unico strumento con cui si può determinare il fronte ottimo di Pareto globale.

In conclusione è interessante osservare il risultato di questo strumento al variare della migrazione, esaminando la percentuale di soluzioni che appartengono all'insieme non-dominato e la velocità di ricerca del fronte ottimo di Pareto.

8.2.1 Il suo utilizzo

L'algoritmo a separazione casuale prende il nome dalla politica di scambio delle soluzioni (*random-random policy*). Abbiamo visto nel paragrafo precedente che questo algoritmo non risolve i problemi multiobiettivo, però è utile per analizzate l'effetto della migrazione sulla velocità di converga e per misurare il livello di cooperazione tra i processori. Queste considerazioni sono interessanti perché vengono riutilizzate all'interno dell'algoritmo *ranking separation*.

Nell'algoritmo *ranking separation* la divisione del calcolo della non-dominanza si ottiene senza la trasmissione degli individui, per cui l'operatore di migrazione non viene più utilizzato per questo scopo. Comunque il suo ruolo è importante per le ragioni mostrate in questo capitolo e deve essere considerato all'interno dell'algoritmo.

L'operatore di migrazione viene implementato per favorire la ricerca delle soluzioni ottime anche se presenta due elementi che possono alterare la *ranking separation*. Il primo è legato al comportamento dell'operatore, infatti, la mutazione non deve influenzare il funzionamento dell'algoritmo perché vogliamo analizzare la sua efficacia. Per questo motivo viene utilizzata la *random-random policy* in modo da non alterare il ranking e la selezione delle soluzioni. Il secondo aspetto riguarda le prestazioni perché la comunicazione tra i processori ha un grosso impatto sulla velocità di calcolo ed è bene tener presente questo elemento nella scelta della dimensione della migrazione.

La cooperazione tra i processori facilita la ricerca del fronte ottimo grazie alla condivisione dei risultati ottenuti dalle popolazioni. I coni caratterizzati da un'evoluzione più lenta possono sfruttare la conoscenza raggiunta da altri per velocizzare la loro ricerca.

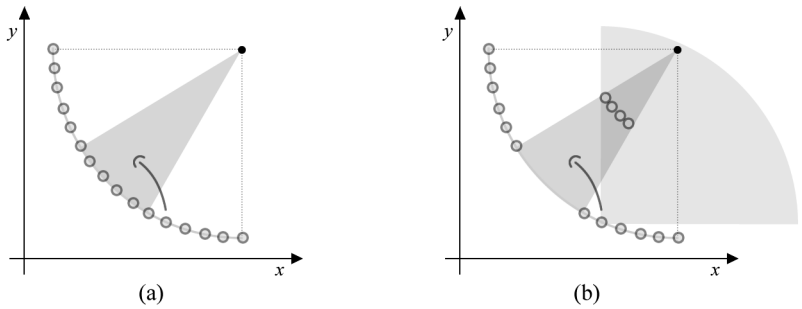


Figura 8.1 – Migrazione tra coni con una diversa velocità di evoluzione.

Questo comportamento è mostrato nel grafico di figura 8.1b dove ci sono tre processori caratterizzati da una diversa velocità di ricerca. La soluzione che migra nel cono centrale (quello più lento) domina tutte le soluzioni del processore (l'area di dominanza è la zona scura). Questa soluzione non verrà scartata per via della dominanza e contribuirà a generare delle soluzioni migliori.

Un differente comportamento si ha nell'altro grafo (figura 8.1a) dove l'evoluzione ha la stessa velocità. In questo caso si nota che la stessa soluzione non viene considerata dal cono centrale perché le altre hanno maggiore priorità. Infatti, sebbene appartengano tutte al fronte non dominato, la soluzione appena migrata si trova fuori dal cono e quindi ha una priorità minore.

In conclusione, se l'intervallo di migrazione è troppo piccolo, si verifica la seconda condizione e le soluzioni trasmesse sono inutili al cono che li riceve. Questo operazione non influenza la ricerca del fronte, ma deteriora le prestazioni dell'algoritmo genetico, per cui si cerca di intervallare la migrazione con diverse generazioni di isolamento.

Capitolo 9
Confronto tra gli algoritmi distribuiti

In questo capitolo si mostrano i risultati dei test condotti sugli algoritmi discussi nei capitoli precedenti. I test fanno riferimento a implementazioni in ambiente Matlab® di alcuni dei più famosi problemi presenti nella letteratura degli algoritmi genetici multiobiettivo. In particolare, le prove sono state condotte valutando le metriche introdotte da Deb [4], che permettono di controllare la distribuzione e la vicinanza delle soluzioni rispetto al fronte ottimo di Pareto.

9.1 Metriche

Per determinare l'efficacia degli algoritmi nella risoluzione di un problema si considerano alcune metriche che permettono di misurare il corretto funzionamento degli algoritmi. Questi parametri sono utili anche per valutare comportamenti differenti degli algoritmi e per capire il loro approccio alla risoluzione dei problemi.

Per valutare il corretto risultato di un ottimizzazione multiobiettivo ci sono due aspetti da considerare: la convergenza verso il fronte ottimo di Pareto e la capacità di mantenere una diversità tra le soluzioni. Questi due aspetti non possono essere misurati con unica metrica, per questo è necessario utilizzare due parametri distinti.

L'ottimizzazione multiobiettivo parallelizata introduce nuove caratteristiche da considerare rispetto alla versione sequenziale. Per poter analizzare questi aspetti è necessario fare riferimento a ulteriori metriche. In particolare è importante misurare la comunicazione tra i processori e la capacità di generare il fronte ottimo di Pareto a partire dall'unione dei risultati dei singoli processori.

9.1.1 Distanza dal fronte ottimo

Questa metrica Υ è stata introdotta da Deb [4] per poter misurare la distanza tra l'insieme delle soluzioni rispetto al fronte ottimo di Pareto, infatti prende il nome di *distance metric*. Questo strumento si può utilizzare solo se siamo a conoscenza dell'espressione analitica del fronte, per cui è limitato ad alcuni problemi di test.

Per mezzo dell'espressione analitica si calcola un insieme di soluzioni che appartengono al fronte ottimo. Nel caso specifico dei test condotti in questo capitolo si è scelto di utilizzare un insieme composto da 1000 soluzioni. Nel grafico sono rappresentati dai punti scuri A che si trovano sul fronte ottimo (figura 9.1, linea scura), mentre le soluzioni generate dall'algoritmo sono i punti chiari B.

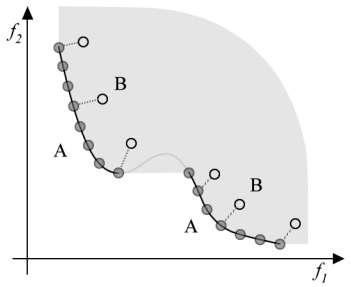


Figura 9.1 – Calcolo della distanza tra le soluzioni e il fronte ottimo.

Per ricavare Υ si calcola per ogni soluzione B la distanza euclidea rispetto ai punti A che si trovano sul fronte ottimo. Di ogni distanza si cerca il minimo rispetto ai punti B e poi viene fatta la somma. Questo calcolo viene riassunto dalla formula (9.1).

$$\Upsilon = \sum_i \min\left(\sqrt{\sum_m \left(s_i^{(m)} - o^{(m)}\right)^2}\right) \quad i = 0,1,2,...,N \quad o = (o_0, o_2, o_2, ..., o_P) \tag{9.1}$$

Per ogni soluzione s_i viene calcolata la distanza euclidea rispetto al vettore o formato dalle P soluzioni ottime, ricavate con la formula analitica del fronte. Il risultato di questa espressione produce un vettore di distanze, dal quale viene estratto solo il minimo. Questo rappresenta la distanza della soluzione i-esima dal fronte (figura 9.1, riga tratteggiata). La somma di tutte queste distanze e la metrica stessa.

Valori di Υ più bassi, indicano una miglior capacità di un algoritmo nel raggiungere il fronte ottimo di Pareto, mentre la soluzione esatta del problema si ottiene quando questa metrica si annulla (Υ=0).

9.1.2 Distribuzione delle soluzioni

La distanza dal fronte è sicuramente un aspetto da tenere in considerazione per valutare un algoritmo, ma risulta del tutto inutile se non si esamina anche la distribuzione delle soluzioni generate. Infatti, si può considerare un caso limite in cui tutte le soluzioni sono coincidenti e si trovano sul fronte. In questo esempio la metrica Υ avrebbe valore nullo, ovvero il risultato ottimale, mentre appare ovvio che un insieme di soluzioni coincidenti non costituiscono una soluzione accettabile.

Per considerare anche questo aspetto viene introdotta la metrica Δ chiamata *diversity metric* che è grado di valutare la distribuzione di un insieme di soluzioni.

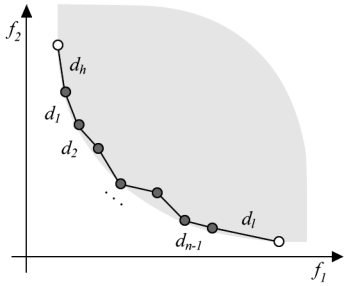


Figura 9.2 – Calcolo della distribuzione delle soluzioni rispetto al fronte ottimo.

Per calcolare Δ è necessario calcolare *N+1* distanze, dove N è il numero di soluzioni generate dall'algoritmo. Le N soluzioni vengono ordinate secondo un obiettivo, nel nostro esempio consideriamo l'obiettivo f_1. In questo modo la soluzione s_1 ha il valore obiettivo f_1 più piccolo e la soluzione s_n ha il valore più grande. Seguendo questo ordine vengono calcolate le *N-1* distanze euclidee con la formula (9.2).

$$d_i = \sqrt{\sum_i \left(s_{i+1}^{(m)} - s_i^{(m)}\right)^2} \qquad i = 1,2,...,N-1 \tag{9.2}$$

Oltre a valori d_i vengono calcolati anche d_h e d_l per considerare anche l'effettiva distribuzione rispetto a fronte ottimo complessivo. Infatti per poter calcolare queste due distanze si utilizzano o_1 e o_P, i valori estremi del fonte di Pareto ottenuti dalla formula analitica.

$$d_h = \sqrt{\sum_m \left(s_1^{(m)} - o_1^{(m)}\right)^2} \qquad d_l = \sqrt{\sum_m \left(o_P^{(m)} - s_N^{(m)}\right)^2} \tag{9.3}$$

Le distanze ottenute sono utili per poter calcolare la metrica Δ come riportato nella formula (9.4).

$$\Delta = \frac{d_h + d_l + \sum_{i=1}^{N-1} \left| d_i - \overline{d} \right|}{d_h + d_l + (N-1)\overline{d}} \qquad \overline{d} = \frac{1}{N-1} \sum_{i=1}^{N-1} d_i \tag{9.4}$$

Valutiamo la formula (9.4) considerando d_h e d_l nulli. In questo caso le soluzioni s_1 e s_N trovate dall'algoritmo coincidono con i punti estremi o_1 e o_P che delimitano il fronte ottimo. In generale quando si ha una distribuzione ottima delle soluzioni il parametro Δ

vale zero, infatti il numeratore si annulla quando tutte le distanze d_i sono uguali alla media delle distanze, ovvero quando sono tutte equidistanti.

In conclusione il valore della sommatoria al numeratore è un indice della distribuzione delle soluzioni, mentre le distanze d_h e d_l stabiliscono il livello di copertura del fronte ottimo di Pareto.

9.1.3 Fronte ottimo di Pareto

Il risultato di un MOGA parallelo è dato dall'unione dei risultati dei singoli processori calcolati in maniera sequenziale. In generale l'insieme delle soluzioni che formano il risultato devono tener conto delle metriche legate al fronte (paragrafi 9.1.1 e 9.1.2), mentre nel caso di algoritmi distribuiti bisogna anche considerare che l'unione dei fronti locali può non generare un fronte di rango uno.

Questo si verifica quando alcune soluzioni dominano altre appartenenti ad un differente processore, perché solo il calcolo del ranking ottenuto sull'intera popolazione garantisce che non ci siano soluzioni dominate. Per questo è importante misurare la percentuale di soluzioni che sono di rango uno, perché diventa uno strumento che valuta il funzionamento degli algoritmi. Infatti, questa metrica indica la capacità di generare un fronte di rango uno, ottenuto dall'unione delle soluzioni dei singoli processori.

9.1.4 Migrazione

Le precedenti metriche misuravano l'efficacia degli algoritmi, ovvero la capacità di produrre soluzioni corrette. Oltre a questi aspetti è importante considerare anche l'efficienza degli algoritmi, poiché l'utilizzo di un algoritmo parallelo è spinto dalla esigenza di migliori prestazioni. Quindi è proprio nell'interesse di questa tesi valutare metriche che tengono conto di queste caratteristiche.

Misurare il tempo di esecuzione degli algoritmi è in genere un ottimo sistema per determinare le prestazioni di un algoritmo, ma allo stesso tempo ha lo svantaggio che può essere fortemente condizionato dall'implementazione. Per valutare le prestazioni nei problemi di test si è scelto di non valutare questa metrica, ma di considerare il numero di individui che migrano tra i processori perché risulta un parametro molto significativo indipendente dall'implementazione.

La decisione di utilizzare la migrazione come parametro dell'efficienza di un algoritmo è legata all'architettura considerata. Infatti abbiamo sempre fatto riferimento ad un cluster di computer dove il costo della comunicazione ha un peso molto maggiore rispetto ai tempi di calcolo dei singoli processori.

9.2 Problemi di test

In seguito verranno illustrati quattro categorie di problemi di test utilizzati per confrontare gli algoritmi discussi nei capitoli precedenti. Ogni problema di test è utile perché mostra gli aspetti e i fenomeni emersi durante la presentazione degli algoritmi. In questo modo ogni ragionamento viene associato a dei casi concreti è dimostrato con i risultati delle simulazioni.

9.2.1 Schaffer (SCH)

Il più semplice e il più studiato problema di test è stato introdotto da Schaffer nel 1984 [15]. Si tratta di un problema a singola variabile con due obiettivi da ottimizzare che si ottiene minimizzando due parabole.

$$\text{SCH}: \begin{cases} \text{Minimize} & f_1(x) = x^2 \\ \text{Minimize} & f_2(x) = (x-2)^2 \\ & -10^5 < x < 10^5 \end{cases} \tag{9.5}$$

Il fronte ottimo di Pareto del problema SCH è un insieme convesso di soluzioni appartenenti all'intervallo $x = [0,2]$ (figura 9.3a), la cui forma è determinata dalla seguente espressione:

$$f_2^* = \left(\sqrt{f_1^*} - 2\right)^2 \qquad f_1^* \in [0,4] \tag{9.6}$$

Le soluzioni ottime appartengono all'intervallo [*0,4*] (figura 9.3b linea scura), mentre per gli altri valori obiettivo fuori dall'intervallo si hanno soluzioni dominate.

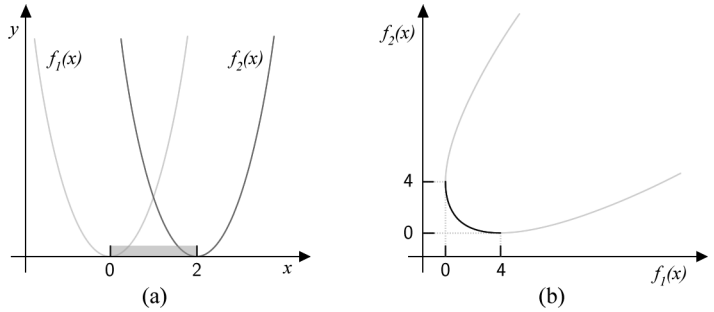


Figura 9.3 – Fronte ottimo di pareto nel problema di Schaffer.

Gli algoritmi sono stati testati sul problema SCH per 200 generazioni utilizzando 4 processori con ciascuno 40 individui. I risultati ottenuti evidenziano comportamenti simili in tutti e quattro gli algoritmi per quanto riguarda la capacità di identificare correttamente il fronte. Mentre se consideriamo il numero di soluzioni che migrano si hanno delle differenze (figura 9.4). Infatti, l'algoritmo dei microconi e dei coni di separazione evidenziano un numero molto alto di migrazioni, con picchi concentrati nelle prime generazioni (figura 9.4a cono di separazione, figura 9.4b microcono di separazione).

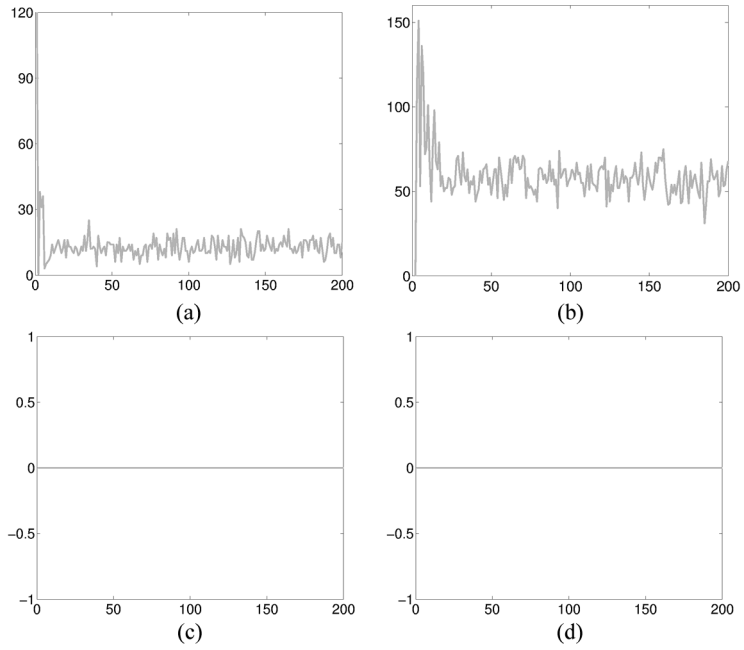


Figura 9.4 – Numero totale di migrazioni della popolazione durante le generazioni.

Bisogna sottolineare che nel caso degli algoritmi di separazione casuale e a graduatoria (figura 9.4c e figura 9.4d) è possibile regolare il livello di migrazione. In queste simulazioni si è scelto di non usare alcuna migrazione, perché entrambi gli algoritmi individuano correttamente il fronte ottimo di Pareto, nonostante che la comunicazione tra i processori sia minima o nulla.

Questi risultati possono essere forvianti perché dimostrano come sia possibile trovare il fronte ottimo senza scambiare informazioni tra i processori. Questo osservazione non è corretta perché tale fenomeno è strettamente legato alla semplicità del problema SCH e non è valido in generale. Se consideriamo i risultati da un altro punto di vista, ci accorgiamo che l'algoritmo dei coni di separazione, e in particolare quello dei microconi, hanno un alto numero di migrazioni, nonostante che gli altri algoritmi riescano a risolvere il problema senza scambiare soluzioni tra i processori.

Dai risultati emersi in questo problema si può arrivare ad un'importante considerazione. In tutti e quattro gli algoritmi la migrazione mette in comunicazione i

processori per poter condividere i risultati e aumentare la velocità di convergenza (paragrafo 8.1.2), mentre nell'algoritmo dei microconi e dei coni di separazione la migrazione ha anche un altro ruolo: quello di spostare le soluzioni che non si trovano nelle aree dedicate al processore (verifica dell'ammissibilità). Adesso, considerando che il problema SCH sia risolvibile senza la cooperazione tra i processori, risulta molto probabile che gli algoritmi dei coni e dei microconi utilizzino la migrazione principalmente per trasferire le soluzioni non ammissibili. Questo utilizzo della migrazione non giustifica però un così alto livello di comunicazione tra i processori, soprattutto perché avviene in fasi dove oramai è stata raggiunto il fronte ottimo di Pareto.

In conclusione gli algoritmi dei microconi e dei coni di separazione hanno elevato costo di comunicazione legato a migrazioni che risultano non necessarie e che possono deteriorare le prestazioni considerevolmente.

Il problema di test SCH permette di confrontare l'algoritmo dei coni di separazione di Deb con l'algoritmo dei microconi. Nel capitolo 6 si è visto come il primo sia un caso particolare del secondo, quando ogni processore ha un solo microcono. Quindi variando il numero di microconi è possibile valutare l'andamento del numero di migrazioni tra i processori.

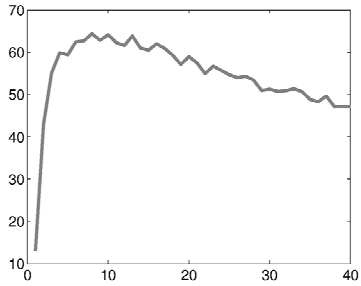


Figura 9.5 – Media delle migrazioni al variare del numero di microconi per processore.

Per ottenere il grafico (figura 9.5) sono state fatte 5 prove per 50 generazioni utilizzando 4 processori con 40 individui ciascuno. Per ogni prova si è fatta una media delle migrazioni dalla 20-esima generazione in poi per avere una distribuzione di soluzioni stabile. Infine per ricavare il grafico si è calcolata un ulteriore media dei valori trovati per ogni prova.

Si nota che il numero di migrazioni minore si ha nel caso di un solo microcono (figura 9.5), che è la situazione equivalente all'algoritmo dei coni di separazione, mentre per un numero di microconi maggiore si hanno valori più alti. Questo andamento è legato al differente comportamento della mutazione e del crossover.

Quando le soluzioni si trovano fuori dei microconi vengono migrate. Nell'algoritmo dei coni di separazione queste soluzioni sono create dalla mutazione (figura 9.6a) e dal crossover tra soluzioni che si trovavano sul confine del cono (figura 9.6b), mentre nel caso dei microconi di separazione si aggiunge un terzo caso (figura 9.6c).

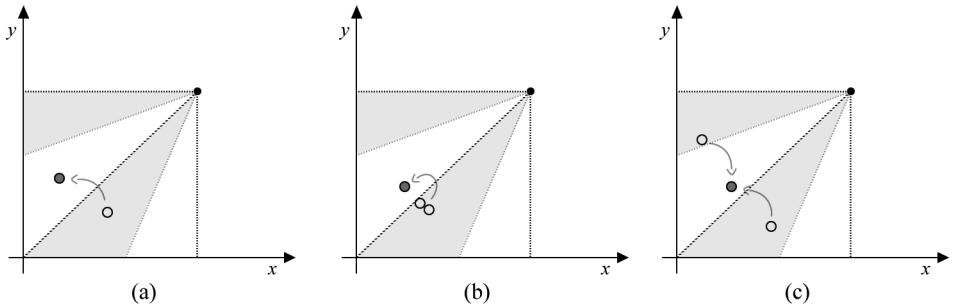


Figura 9.6 – I tre tipi di ricombianzione (mutazione, crossover e crossover tra microconi diversi).

Quando un processore possiede più di un microcono lo spazio ammissibile degli obiettivi è frammentato in più aree. Per cui può accadere che il crossover avvenga tra due soluzioni di microconi diversi e che il risultato sia una soluzione non ammissibile (figura 9.6c). Inoltre, più microconi ci sono, più è facile che il crossover del secondo tipo (figura 9.6b) generi una soluzione non ammissibile, perché il numero aree separate è aumentato e

quindi anche il numero di confini. La stessa cosa accade anche per la mutazione perché le soluzioni si trovano in microconi più piccoli ed è più facile oltrepassare i confini. Questi tre fenomeni uniti aumentano il numero di migrazioni tra i processori.

L'andamento del grafico (figura 9.5) entra in contraddizione con quanto detto perché non ha un andamento completamente crescente. Questo fenomeno si spiega semplicemente con il meccanismo della frammentazione illustrato nel capitolo dei microconi (paragrafo 6.2.2). Quando il numero di microconi è molto elevato si ha una distribuzione migliore nello spazio e il fenomeno della frammentazione diminuisce perché si crea un minor conflitto con la crowding distance. Bisogna aggiungere che, sebbene la migrazione si riduca, rimane sempre molto maggiore rispetto al cono si separazione, che è il caso con un solo microcono.

Un ultima osservazione sul problema SCH riguarda il comportamento dell'algoritmo del cono di separazione nelle prime generazioni. Proprio questo particolare esempio porta alla luce un difetto di questo algoritmo. Si osserva che nelle prime generazioni si hanno un numero di migrazioni tale che tutti i processori, tranne uno, trasmettono tutte le loro soluzioni. Infatti, si misurano 120 migrazioni che corrispondo a tre processori con 40 soluzioni ciascuno (figura 9.4a).

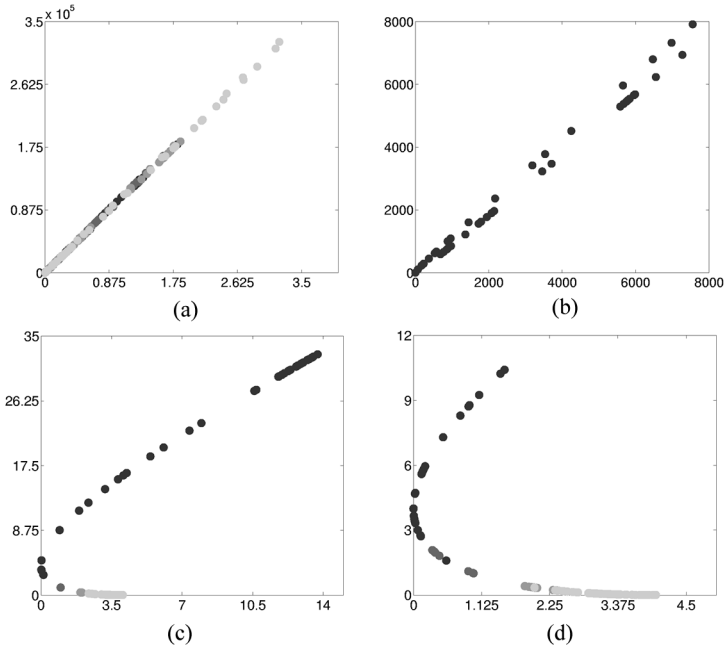


Figura 9.7 – Fronte generato dell'algoritmo del cono di separazione al variare delle generazioni.

Dal grafico si vede che alla prima generazione si hanno le soluzioni di tutti i processori (figura 9.7a), mentre nella seconda le soluzioni di uno solo (figura 9.7b). Questo comportamento ha un enorme influenza sulle prestazioni perché si hanno generazioni dove è in funzione un solo processore. Inoltre, i computer senza soluzioni hanno difficoltà a ripopolarsi perché devono aspettare che qualche soluzione migri nella loro area, con la possibilità che questo non accada.

9.2.2 Fonseca e Fleming (FON)

Nel 1995 Fonseca e Fleming [8] presentarono un problema di ottimizzazione a due obiettivi con n variabili definite nell'intervallo [*-4,4*].

$$\text{FON}: \begin{cases} \text{Minimize} \quad f_1(x) = 1 - \exp\left(-\sum_{i=1}^{n} \left(x_i - \frac{1}{\sqrt{n}} \right)^2 \right) \\ \text{Minimize} \quad f_2(x) = 1 - \exp\left(-\sum_{i=1}^{n} \left(x_i - \frac{1}{\sqrt{n}} \right)^2 \right) \\ -4 < x_i < 4 \qquad i = 1, 2, \ldots, n \end{cases} \tag{9.7}$$

L'insieme delle soluzioni del problema di minimo (9.7) si dispongono su un fronte dalla forma non convessa la cui espressione analitica è data dalla seguente formula:

$$f_2^* = 1 - \exp\left\{-\left[2 - \sqrt{-\ln\left(1 - f_1^*\right)}\right]^2\right\} \qquad f_1^* = \left[0, 1 - \exp(-4)\right] \tag{9.8}$$

È interessante notare che la forma del fonte non dipende dal numero di variabili in ingresso (figura 9.8). Nei casi e nelle prove presentate in questo testo sono state utilizzate tre variabili.

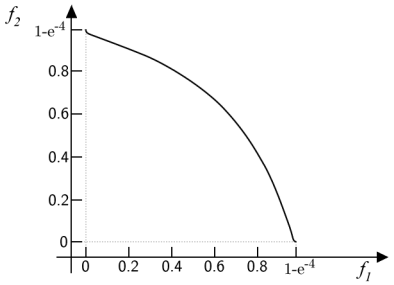


Figura 9.8 – Fronte ottimo di pareto nel problema di Fonseca e Fleming.

Il problema di test FON ha una complessità che dipende dal numero di variabili in ingresso. Aumentando questo numero gli algoritmi incontrano una maggiore difficoltà nel trovare il fronte ottimo di Pareto. In questi contesti è più facile osservare le differenze nella risoluzione del problema quando si variano i parametri degli algoritmi, mentre risulta difficile in problemi più semplici come SCH, dove si ha sempre un comportamento identico perché la convergenza non è influenzata dalla scelta dai parametri.

Nel paragrafo 5.2.3 sono stati illustrati alcuni casi in cui l'algoritmo dei coni di separazione non ha una distribuzione uniforme delle soluzioni. La natura non convessa del fronte nel problema FON ci permette di osservare uno di questi esempi.

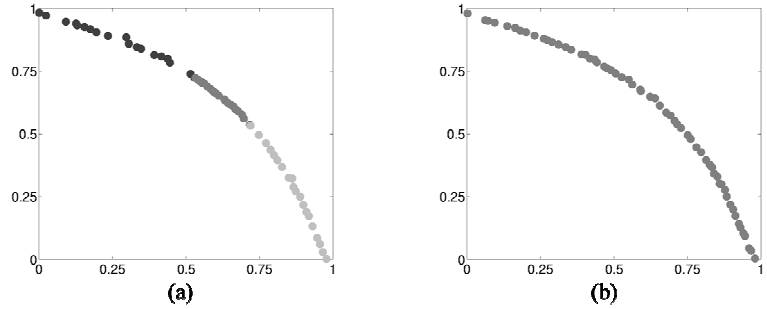


Figura 9.9 – Distribuzione delle soluzioni con tre processori e con un singolo processore.

Questo grafico è il risultato di due simulazioni della durata di 100 generazioni: la prima con 3 processori e 20 individui ciascuno (figura 9.9a), la seconda con un solo processore e 60 individui (figura 9.9b). Ripetendo entrambe le simulazioni per 15 volte è possibile mettere a confronto la media e la deviazione standard della metrica Δ (*diversity*):

	Media ± Dev. Standard
Singolo processore	0.01008 ± 0.00453
3 processori	0.01089 ± 0.00660

Tabella 9.1 – Metrica Δ (diversity).

Dal confronto diretto della metrica si nota che utilizzando tre processori si ottiene una distribuzione peggiore (tabella 9.1). Questo perché il processore centrale ha un'area del fronte più piccola (paragrafo 5.2.3), come si vede anche dal grafico delle soluzioni (figura 9.9).

In generale è interessante valutare quanto la migrazione possa modificare il risultato finale di un problema. Questa tesi non vuole analizzare tutti gli aspetti della comunicazione e la cooperazione tra i processori, ma solo osservare alcuni comportamenti

che si verificano in determinati contesti. Per cui, utilizzando il problema FON, sono stati fatte alcune simulazioni per determinare l'influenza della migrazione sulle metriche presentate all'inizio di questo capitolo. Le prove sono state condotte sull'algoritmo a separazione casuale con 80 individui per ognuno dei 4 quattro processori, per un totale di 100 generazioni.

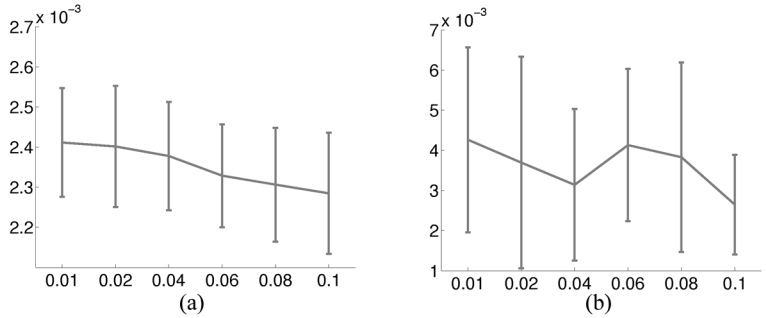


Figura 9.10 – Distanza dal fronte ottimo e diversità delle soluzioni al variare della migrazione.

Il grafico (figura 9.10) mostra l'andamento della media e della deviazione standard legato alle metriche Υ e Δ al variare della percentuale di mutazione. Si nota che le prestazioni dell'algoritmo migliorano quando si ha una comunicazione maggiore tra i processori.

		Distanza dal fronte (Υ)	Diversità delle soluzioni (Δ)
Percentuale di migrazione	0.01	0.0024116 ± 0.0001693	0.0042627 ± 0.0027561
	0.02	0.0024017 ± 0.00013055	0.0036965 ± 0.0020839
	0.04	0.0023775 ± 0.00013837	0.0031405 ± 0.0014617
	0.06	0.0023286 ± 0.00013565	0.0041308 ± 0.0023335
	0.08	0.0023063 ± 0.00014617	0.0038302 ± 0.0017674
	0.10	0.0022846 ± 0.00016074	0.0026451 ± 0.001367

Tabella 9.2 – Metrica Υ *(distance) e metrica* Δ *(diversity) al variare della migarzione.*

Nella tabella 9.2 sono riassunti i risultati delle simulazioni, riportando per entrambe le metriche la media e la deviazione standard.

Infine, il problema di Fonseca e Fleming è stato utilizzato per valutare il comportamento dell'algoritmo ranking separation quando varia il numero di microconi per processore. Questo tipo di prova è stata condotta per capire l'influenza di questo parametro sulla risoluzione dei problemi. Le simulazione hanno impiegato 4 processori con ciascuno 80 individui e un numero di microconi da uno a 40, ottenendo per cui, nel caso maggiore due individui per microcono. Le simulazioni sono state ripetute 5 volte per ogni valore assegnato ai microconi su un totale di 100 generazioni ciascuna.

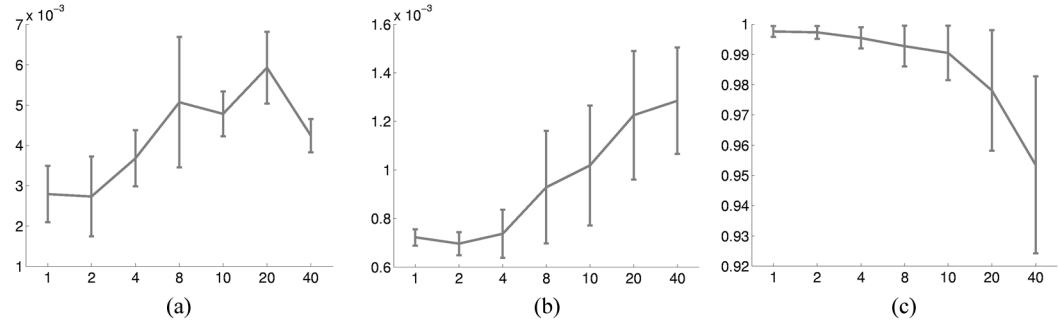


Figura 9.11 – Distanza (a), diversità (b) e % rango uno (c) con un diverso numero di microconi.

I tre grafici mostrano l'andamento della distanza dal fronte (figura 9.11a), della diversità delle soluzioni (figura 9.11b) e la percentuale di popolazione di rango uno (figura 9.11c). Si nota che tutte e tre le metriche peggiorano quando aumenta il numero di

microconi per processore. In realtà si può dimostrare con ulteriori simulazioni che un numero maggiore di microconi diminuisce la velocità di convergenza. Per cui, per avere gli stessi risultati occorrono più generazioni.

Numero di microconi per processore	Distanza dal fronte (Y)	Diversità delle soluzioni (Δ)	Popolazione rango uno (%)
1	0.00072 ± 0.00003	0.00279 ± 0.00070	0.99762 ± 0.00180
2	0.00070 ± 0.00005	0.00273 ± 0.00099	0.99736 ± 0.00208
4	0.00074 ± 0.00010	0.00368 ± 0.00069	0.99550 ± 0.00353
8	0.00093 ± 0.00023	0.00507 ± 0.00162	0.99282 ± 0.00671
10	0.00102 ± 0.00025	0.00478 ± 0.00056	0.99057 ± 0.00900
20	0.00123 ± 0.00026	0.00592 ± 0.00089	0.97816 ± 0.01990
40	0.00129 ± 0.00022	0.00424 ± 0.00042	0.95355 ± 0.02923

Tabella 9.3 – Metrica Y, *metrica* Δ *e popolazione di rango uno variando il numero microconi.*

Nella tabella sono riportati i valori medi con deviazione standard per ognuna delle tre metriche presentate. Da questi valori si nota che l'andamento di Δ (*diversity*) migliora quando si ha un numero elevato di microconi. Questa inversione di tendenza si può associare alla natura delle divisioni dei microconi, che aiutano a distribuire le soluzioni lungo il fonte.

9.2.3 Kursawe (KUR)

Il problema di usato da Frank Kursawe nel 1990 [12] è caratterizzato da un fronte ottimo con forma non convessa e discontinua (figura 9.12). È un problema di ottimizzazione a tre variabili nell' intervallo [*-5,5*] con due obiettivi da minimizzare.

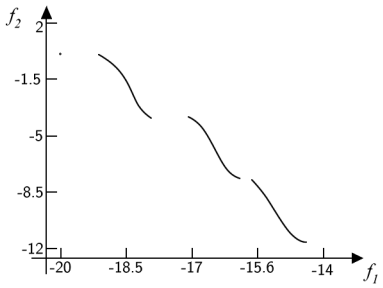


Figura 9.12 – Fronte ottimo di pareto nel problema di Kusawe.

In realtà questo non è il problema originale proposto da Kursawe, ma si tratta di una variante di Veldhuizen [18] nata per semplificare l'esempio (9.9). Infatti il numero di variabili n viene fissato a 3, tutte limitate nell'intervallo [*-5,5*].

$$\text{KUR}: \begin{cases} \text{Minimize} \quad f_1(x) = \sum_{i=1}^{n-1}\left[-10 \cdot \exp\left(-0.2 \cdot \sqrt{x_i^2 + x_{i+1}^2}\right)\right] \\ \text{Minimize} \quad f_2(x) = \sum_{i=1}^{n}\left[|x_i|^{0.8} + 5 \cdot \sin\left(x_i^3\right)\right] \\ -5 < x_i < 5 \qquad i = 1, 2, ..., n \end{cases} \tag{9.9}$$

L'espressione delle funzioni obiettivo è molto complessa anche nella versione semplificata, infatti, non è possibile risalire alla formula analitica del fronte ottimo di Pareto. Questo non ci permette di utilizzare le metriche che valutano la distanza dal fronte e la diversità delle soluzioni.

Comunque questo problema risulta particolare perché il fronte di Pareto non è continuo ed è interessante analizzare il comportamento degli algoritmi, valutando la capacità di generare un fronte globale non dominato. In particolare si analizzano i gli algoritmi *cone separation* e *ranking separation* in un problema caratterizzato da 4 processori con una popolazione di 60 individui ciascuno.

Le simulazioni sono state eseguite per 5 volte con 60 generazioni ed è stata estrapolata la media e la varianza della migrazione e della percentuale di soluzioni non-dominate del primo fonte. I risultati sono riassunti nella tabella sottostante:

	Popolazione rango uno (%)	Migrazione (%)
Cone separation	0.9666 ± 0.0118	13.746 ± 3.768
Ranking Separation	0.9870 ± 0.0023	0.000 ± 0.000

Tabella 9.4 – Paragone tra due algoritmi sulla capacità di generare fronti non-dominati.

I risultati mostrano la capacità, dell'algoritmo *ranking separation* rispetto all'algoritmo *cone separation*, di determinare un fronte non-dominato formato da più soluzioni. Questi risultati sono ancora più interessanti, se si considera che sono stati raggiunti senza l'utilizzo della migrazione. Questo vuol dire che, anche in problemi più complessi come questo, è possibile risalire al fronte ottimo di Pareto riducendo al minimo la comunicazione tra i processori.

9.2.4 Zitzler, Deb e Thiele (ZDT)

Zitzler, Deb e Thiele hanno ideato sei problemi (ZDT1-6) basati sullo stesso principio di costruzione. Questi problemi ottimizzano due obiettivi e sono caratterizzati completamente da tre funzioni $f_1(x)$, $g(x)$ e $h(x)$.

$$\text{ZDT(1-6)}: \begin{cases} \text{Minimize} & f_1(x) \\ \text{Minimize} & f_2(x) = g(x) \cdot h\big(f_1(x), g(x)\big) \end{cases} \tag{9.10}$$

I sei problemi si distinguono per le differenti formule che determinano le tre funzioni. Queste definiscono la forma del fronte ottimo di Pareto, le sue proprietà (concavo o convesso) e le sue caratteristiche (continuo e non). Le soluzioni ottime si ottengono per $g(x)$=1 eccetto che nel problema ZDT5 , che però non verrà considerato nella trattazione. Le simulazioni riguarderanno solo i primi tre problemi escludendo, quindi, anche i test ZDT4 e ZDT6.

ZDT1

Questo problema di test è caratterizzato da uno spazio di ricerca a n dimensioni (n = 30), per cui l'algoritmo generico deve ottimizzare delle funzioni a 30 variabili caratterizzate dalla seguente formula:

$$\text{ZDT1}: \begin{cases} f_1(x) = x_1 \\ g(x) = 1 + \dfrac{9}{n-1} \cdot \sum_{i=2}^{n} x_i \\ h(f_1, g) = 1 - \sqrt{f_1 / g} \end{cases} \tag{9.11}$$

Tutte le variabili sono definite nell'intervallo $[0,1]$ e l'insieme ottimo di Pareto si ha in corrispondenza di $x_1 \in [0,1]$ e $x_i = 0$, con $i = 2, 3, \ldots, n$.

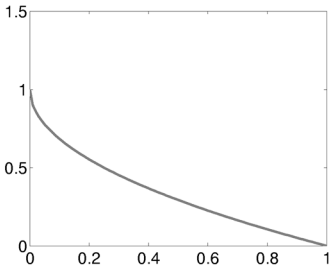


Figura 9.13 – Fronte ottimo di pareto nel problema ZDT1.

Il grafico illustra la natura convessa del fronte del problema ZDT1 che è, molto probabilmente, il più semplice dei sei perché è continuo e uniformemente distribuito.

ZDT2

Il problema ZDT2 è simile al precedente, ma è caratterizzato da un fonte non convesso. Abbiamo per cui lo stesso numero di variabili (n = 30) e lo stesso intervallo di definizione $x_i \in [0,1]$.

$$\text{ZDT2}:\begin{cases} f_1(x) = x_1 \\ g(x) = 1 + \dfrac{9}{n-1}\cdot\sum_{i=2}^{n} x_i \\ h(f_1, g) = 1 - (f_1/g)^2 \end{cases} \tag{9.12}$$

Il fronte ottimo di Pareto si ottiene con i valori delle variabili $x_1 \in [0,1]$ e $x_i = 0$, con $i = 2, 3, \ldots, n$, e assume la seguente forma.

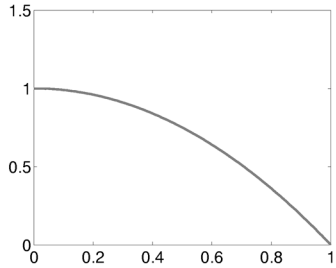


Figura 9.14 – Fronte ottimo di pareto nel problema ZDT2.

Come il precedente problema si ha un insieme di soluzioni che sono uniformemente distribuite nello spazio di ricerca e continuo nello spazio degli obiettivi.

ZDT3

Questo problema risulta molto interessante perché propone un fronte ottimo di Pareto discontinuo, infatti l'insieme delle soluzioni ottime viene diviso in cinque aree dello spazio degli obiettivi. Il problema ZDT3 è caratterizzato sempre da 30 variabili in ingresso e dalle seguenti formule:

$$\text{ZDT3}:\begin{cases} f_1(x) = x_1 \\ g(x) = 1 + \dfrac{9}{n-1}\cdot\sum_{i=2}^{n} x_i \\ h(f_1, g) = 1 - \sqrt{f_1/g}\cdot(f_1/g)\cdot\sin(10\pi f_1) \end{cases} \tag{9.13}$$

Come i precedenti problemi l'insieme delle soluzioni ottime si ricava con i seguenti valori x_i in ingresso: $x_1 \in [0,1]$ e $x_i = 0$, con $i = 2, 3, \ldots, n$.

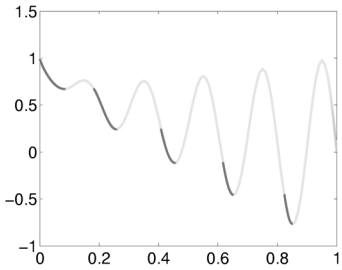


Figura 9.15 – Fronte ottimo di pareto nel problema ZDT3.

Come si vede dalla figura, la particolarità e la complessità di questo problema sono date dal fronte discontinuo. Questo aspetto è interessante da esaminare perché evidenzia il limite della divisione in coni dello spazio degli obiettivi.

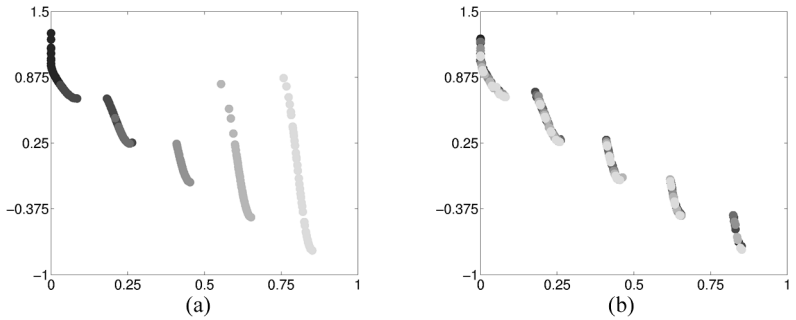


Figura 9.16 – Fronte ottimo di pareto ricavato con i coni e con i microconi.

Dal primo grafico(figura 9.16a), ottenuto con utilizzando i coni di separazione, si ritrova l'errato comportamento descritto nel paragrafo 5.2.2. Infatti, è evidente che parte degli individui di alcuni fronti locali (figura 9.16a, punti chiari) sono dominati da sottopopolazioni di altri processori. Mentre, si può osservare che con l'utilizzo dei microconi questo inconveniente non si manifesta (figura 9.16b).

Simulazioni

I problemi di test di Zitzler, Deb e Thiele hanno una complessità maggiore dei precedenti ed è interessare confrontare i risultati che si ottengono utilizzando l'algoritmo la *ranking separation* e l'algoritmo *cone separation*.

		Cone separation	Ranking separation
ZDT1	Distanza dal fronte (Y)	0.0410 ± 0.0048	0.0337 ± 0.0006
	Diversità delle soluzioni (Δ)	0.2255 ± 0.0024	0.1908 ± 0.0007
	Popolazione rango uno (%)	0.9742 ± 0.0102	0.9683 ± 0.0118
	Migrazione (%)	8.5455 ± 3.8565	2.7273 ± 1.5551
ZDT2	Distanza dal fronte (Y)	0.1273 ± 0.0059	0.1134 ± 0.0062
	Diversità delle soluzioni (Δ)	0.2616 ± 0.0071	0.2230 ± 0.0200
	Popolazione rango uno (%)	0.9727 ± 0.0186	0.9765 ± 0.0102
	Migrazione (%)	9.3636 ± 3.6903	3.1818 ± 1.0787
ZDT3	Distanza dal fronte (Y)	0.0177 ± 0.0016	0.0221 ± 0.0037
	Diversità delle soluzioni (Δ)	0.1423 ± 0.0089	0.1251 ± 0.0104
	Popolazione rango uno (%)	0.8330 ± 0.0215	0.9024 ± 0.0321
	Migrazione (%)	6.7935 ± 2.1893	3.0839 ± 1.2204

Tabella 9.5 – Metrica Y, *metrica* Δ, *popolazione di rango e migrazione nei problemi di Ziztler.*

I dati riportati in tabella paragonano il funzionamento dei due algoritmi. Da questo confronto si osserva che l'algoritmo *ranking separation* ottiene i valori migliori in quasi tutte le metriche. Inoltre è importante sottolineare che questi risultati sono stati ottenuti con una migrazione più bassa, la cui dimensione è stata scelta a priori ($\cong$3 individui per generazione) e rimane costante per tutta l'esecuzione. Questi dati sono ancora più interessanti se si considera che, nel calcolo della media, non sono state considerate le prime generazioni. Infatti, come si è visto nel problema di Shaffer, le prime fasi di esecuzione dell'algoritmo *cone separation* sono caratterizzate da un alto numero di individui trasmessi.

9.3 Sintesi dei risultati

In questo capitolo sono stati ripercorsi attraverso degli esempi concreti tutti gli aspetti emersi nella tesi. Ogni argomento affrontato o discusso durante la presentazione degli algoritmi è stato giustificato con prove, simulazioni e grafici; tutto questo, partendo dalla presentazione dei singoli problemi di test e valutando volta per volta il comportamento degli algoritmi.

In questo paragrafo si riassumono tutti gli aspetti seguendo un altro ordine (quello degli algoritmi), per avere un quadro complessivo che segua lo stesso filo logico con cui gli argomenti sono stati presentati.

9.3.1 Cono di separazione

Il funzionamento del ranking è il primo aspetto analizzato nell'algoritmo. In particolare viene osservato il comportamento dei coni di separazione e la loro capacità di dividere il fronte globale in fronti locali. Da questa emerge un primo difetto che è illustrato nel problema di test ZDT3 (figura 9.16) e KUR, con riferimento alla capacità di generare una popolazione di rango uno.

I problemi che riguardano la distribuzione delle soluzioni si ritrovano nel sistema di Fonseca (FON) all'interno della figura 9.9 e della tabella 9.1. Mentre il comportamento

della migrazione è analizzato in Kursawe (KUR), Ziztler (ZDT) e Schaffer (SCH). In particolare in quest'ultimo, si osserva la trasmissione degli individui in due fasi distinte (iniziale e a regime), mentre negl'altri due problemi si ha un confronto con l'algoritmo *ranking separation*.

9.3.2 Microcono di separazione

L'algoritmo microcone separation è un'esenzione dell'algoritmo di Deb per risolvere i problemi legati ai fronti locali. I vantaggi che si hanno utilizzando questa variante sono riassunti nella figura 9.16. Gli svantaggi di questa tecnica sono legati alla migrazione e sono mostrati all'interno del problema di Schaffer (SCH). Le simulazioni sono state condotte per esaminare tale fenomeno al variare del numero di microconi (figura 9.5). Inoltre sono illustrate le cause dell'eccessiva migrazione, paragonandola con l'algoritmo *cone separation*.

9.3.3 Ranking separation

Il funzionamento dell'algoritmo viene valutato in tutti i problemi di test e spesso è confrontato direttamente con l'algoritmo *cone separation* (KUR). Un'analisi più approfondita del suo comportamento viene fatta in Fonseca (FON), dove si esamina l'effetto della divisione del fronte in microconi (figura 9.11), e in Kursawe (KUR), dove viene valutata la percentuale di popolazione di rango uno.

9.3.4 Separazione casuale

L'algoritmo random separation viene utilizzato per analizzare l'effetto della migrazione sulla ricerca del fronte. Variando il numero di individui che possono essere trasmessi ad ogni generazione si valuta l'andamento delle soluzioni ottenute dall'algoritmo (figura 9.10).

Capitolo 10
Conclusioni

Nel passato il *parallel computing* è sempre stato considerato uno stratagemma per aumentare la potenza di calcolo, in attesa che il progredire della tecnologia producesse soluzioni sequenziali altrettanto potenti. Nell'ultimo decennio lo scenario tecnologico è molto cambiato e le vecchie considerazioni, che guidavano lo sviluppo delle architetture, sono state completamente rivalutate [1]. I tradizionali campi di ricerca che spingevano ad incrementare le prestazioni, come la frequenza di clock o il parallelismo a livello di istruzione (ILP), hanno mostrato un sostanziale rallentamento negl'ultimi anni [16]. A questi concetti se ne sono aggiunti di nuovi, legati all'efficienza energetica e al contenimento dei costi di progettazione, che hanno orientato lo sviluppo delle architetture verso soluzioni parallele.

In questo nuovo scenario il *parallel computing* non è stato più considerato come un'alternativa al calcolo sequenziale, ma ha rappresentato un fenomeno che si è affermato sempre di più in ogni settore. Il suo utilizzo, infatti, non si è limitato solo al campo dei supercomputer, ma ha riguardato anche gli elaboratori più comuni, grazie all'avvento delle tecnologie multicore.

Il parallelismo non è definito solamente da una nuova architettura di calcolatori, ma si riferisce anche a tecniche di programmazione che tengono in considerazione la possibilità di eseguire il codice in parallelo. Grazie a questi strumenti, siamo in grado di raggiungere prestazioni elevate e di favorire lo sviluppo di programmi che richiedono un grande sforzo computazionale, come per esempio gli algoritmi genetici.

Come abbiamo visto, solo gli algoritmi genetici a singolo obiettivo si adattano facilmente alle implementazioni parallele, mentre gli algoritmi multiobiettivo incontrano un grosso ostacolo nel distribuire l'elaborazione a causa del calcolo della non-dominanza (paragrafo 4.3.1). Fortunatamente nel 2004, Deb [2] ha elaborato una tecnica per parallelizare questo calcolo basata su un'opportuna divisione della popolazione. Questa tecnica, chiamata *cone separation*, nasce da un'attenta analisi dello spazio degli obiettivi ed individua, nel fronte di Pareto, alcune caratteristiche che garantiscano la dominanza anche in presenza della divisione in coni.

L'algoritmo *cone separation* possiede alcuni difetti che deteriorano le prestazioni senza influenzare, nella maggior parte dei casi, la correttezza del risultato finale. In particolare si evidenziato due problemi legati ai fronti discontinui (paragrafo 5.2.2) e all'eccessiva migrazione (paragrafo 6.2.2).

Il primo difetto viene risolto introducendo il concetto dei microconi per dividere lo spazio degli obiettivi. Questo strumento è un'estensione dei coni di Deb, che offre ad ogni processore una visione generale del fronte, risolvendo il conflitto tra fronte globale e fronti locali (paragrafo 6.2.1). Tuttavia, nell'algoritmo *microcone separation* aumenta in modo considerevole il numero di individui che migrano tra i processori e di conseguenza le prestazioni peggiorano al punto che i microconi diventano del tutto inutili. Questo fenomeno era gia presente nell'algoritmo *cone separation* e costituiva un difetto che poteva influenzare le prestazioni negativamente.

Per diminuire l'eccessiva migrazione, viene introdotto un ulteriore algoritmo chiamato *ranking separation*, che rivaluta completamente il meccanismo di divisione del fronte ottenuto con i coni. In questo algoritmo viene conservata l'idea originale di Deb della divisione che rispetta la dominanza, ma viene modificata interamente la tecnica con cui

questa viene attuata. Infatti si sostituisce il metodo basato sulla migrazione con una tecnica ripresa dai problemi con vincoli *soft constrained*. Questa tecnica assegna un priorità minore alle soluzioni che si trovano fuori dai coni, così che la selezione privilegia le soluzioni all'interno durante la scelta (paragrafo 7.3.1).

L'algoritmo *ranking separation* ha lo stesso comportamento dell'algoritmo *cone separation* e non ha la migrazione degli individui, per cui il calcolo dei singoli processori è fortemente parallelo. Comunque, per essere precisi, esistono ancora due operatori che utilizzano la comunicazione e le informazioni ad essa connesse: la *microcone assignment* e la *random migration*. Il funzionamento del primo si basa sul quadrato unitario, per cui i processori trasmetto le soluzioni migliori di ogni obiettivo per determinarne dimensione e forma (paragrafo 7.4.2). Il secondo introduce un meccanismo controllato di migrazione per ottenere un effetto cooperativo tra i processori: questa tecnica segue i principi del modello a isole (paragrafo 8.1). In entrambi i casi la comunicazione è minima: nel primo è limitata a poche soluzioni, mentre nel secondo avviene ad intervalli di generazioni.

In conclusione l'algoritmo *ranking separation* sfrutta al massimo un architettura parallela limitando al minimo la comunicazione. Inoltre non presenta alcun problema di scalabilità perché il numero di processori può essere aumentato senza alcun inconveniente e la comunicazione non risente assolutamente di questo cambiamento.

Un'ultima osservazione riguarda la strategia con cui è stato implementato l'algoritmo *ranking separation*. Sappiamo che la dominanza è l'unico strumento per capire se una soluzione è migliore di un'altra, infatti la popolazione viene divisa in fronti non-dominati, così le soluzioni sono ordinate ed è possibile selezionare le migliori. Visto che gli individui all'interno dello stesso fronte hanno uguale priorità vengono utilizzate delle tecniche per distinguerli. Queste tecniche creano una classifica delle soluzioni in base a criteri di distribuzione della popolazione, come per esempio la *crowding distance* e la *microcone distance*. È importante sottolineare che l'ordinamento all'interno dei fronti non fa parte del problema multiobiettivo (trovare il fronte ottimo di Pareto), ma è uno stratagemma imposto dai limiti dell'algoritmo. Per esempio, il fronte ottimo di Pareto è caratterizzato da un insieme infinito di soluzioni, mentre sappiamo che l'algoritmo utilizza un numero limitato di individui. Per rappresentare al meglio quest'insieme è necessario che le soluzioni siano ben distribuite grazie ad un operatore come la *crowding distance*.

Inoltre, quando si ha un algoritmo distribuito, gli individui di un processore devono essere disposti solo in alcune aree dello spazio. Per ottenere questo, viene introdotto l'operatore *microcone distance* nel ranking anche se non è collegato direttamente alla ricerca del fronte di Pareto.

In conclusione l'ordinamento per la selezione mescola due diversi aspetti: il primo legato alla ricerca delle soluzioni ottime e il secondo connesso al funzionamento dell'algoritmo. Da questo si capisce che l'operatore di ranking può avere un ulteriore ruolo che bisogna tenere fortemente in considerazione. Infatti, ogni qualvolta sia necessario obbligare le soluzioni ad un determinato comportamento, è possibile modificare l'ordinamento dell'operatore. Questo è valido per qualsiasi richiesta, anche quando vengono inseriti due criteri in contrasto, perché è sufficiente utilizzare il concetto di dominanza.

Appendice A
Crowding distance assigment

La crowding distance assigment assicura un ottima distribuzione delle soluzioni all'interno dello stesso fronte. Viene utilizzata come secondo criterio di scelta all'interno del ranking dopo la dominanza: il meccanismo della selezione sceglie tra le soluzioni a parità di rango quelle con un valore di crowding distance maggiore. Questa metodo risulta poco efficace quando viene utilizzato per classificare i fronti di rango maggiore di uno, soprattutto se questi sono formati da poche soluzioni.

Per analizzare questo difetto facciamo riferimento all'esempio raffigurato nel grafico (figura a.1). L'operatore di selezione deve scegliere 10 soluzioni su 12 totali per cui dovrà scartare 2 individui appartenenti al fronte di rango 2 (punti nella zona chiara) perché il primo fronte è formato da 7 individui (punti nella zona scura).

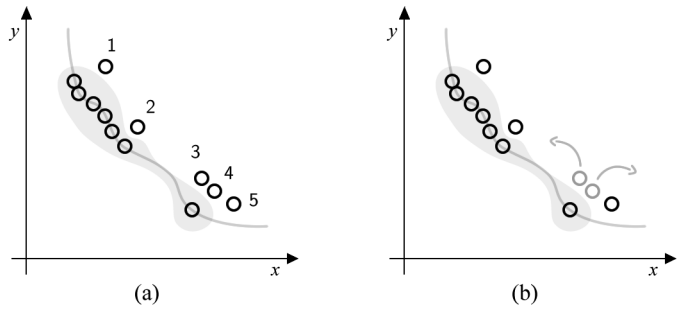


Figura A.1 – Crowding distance assigment sul fronte di rango due.

Il calcolo della crowding distance avviene all'interno del fronte di rango 2. Le soluzioni che hanno i valori maggiori sono la *1*, *2*, e *5* perché si trovano agli estremi (*1* e *5*) oppure perché sono distanti dalle altre soluzioni (*2*). Le altre soluzioni (*3* e *4*) vengono scartate anche se si trovano in un'area a bassa densità (figura a.1b). In conclusione la crowding distance assigment non ha un comportamento corretto perché assegna dei valori che non corrispondo alla vera distribuzione e non tengono conto delle soluzioni degl'altri fronti.

Quando la crowding distance è applicata su fronti con rango maggiore di uno è probabile che possa funzionare in maniera sbagliata, delineando una distribuzione valida solo all'interno del fronte stesso. In questi casi il suo utilizzo diventa inutile ed è possibile pensare di non impiegare questo operatore, lasciando al ranking solo l'ordinamento in fronti non dominati. Per costruire questa variante viene eseguito un controllo ad ogni generazione per verificare che la crowding distance venga applicata solo al fronte di rango uno. In caso contrario questo operatore assegna dei valori in maniera casuale.

Appendice B
Algoritmi genetici multiobiettivo con vincoli

In questa sezione verranno presentati cinque metodi per risolvere i problemi con vincoli mediante l'utilizzo degli algoritmi genetici multiobiettivo. Questi approcci sono stati illustrati nel libro di Deb [4] con riferimento ad un problema generico caratterizzato dalla seguente forma:

$$\begin{cases} \text{Minimize} \quad f_m(x) \\ \qquad\quad g_j(x) \le 0 \\ \qquad\quad m = 1,2,...,N \ \text{ e } \ j = 1,2,...,K \end{cases} \tag{B.1}$$

Questi sistemi sono formati da N funzioni obiettivo e K vincoli. Ogni vincolo riduce lo spazio di ricerca andando a modificare la forma del fronte ottimo di Pareto.

Nell'ultima parte dell'appendice si considera se le tecniche sono applicabili in contesti simili a quelli affrontati nell'introduzione dell'algoritmo ranking separation (Capitolo 7).

B.1 Trattare i vincoli

Deb tiene in considerazione entrambe le tecniche *soft* e *hard constrained*, ma giudica le prime molto più valide perché si adattano meglio a problemi reali, che per loro natura sono più complessi. La prima tenica segue un approccio *hard constrained*, mentre le successive sono tutte *soft constrained*.

B.1.1 Ignorare le soluzioni

L'approccio più semplice e immediato è scartare tutte le soluzioni che non rispettano i vincoli. L'inconveniente di questo metodo si manifesta in tutti quei casi dove è difficile trovare il fronte ottimo di Pareto, per cui risulta utile considerare le soluzioni non ammissibili perché facilitano la ricerca di quelle ammissibili. In altre parole, per raggiungere aree dello spazio corrette, l'algoritmo deve passare dalle zone che non rispettano i vicoli.

B.1.2 Penalty function

Questo metodo utilizza una tecnica che penalizza le funzioni obiettivo delle soluzioni x_i non ammissibili. In questo modo gli individui vengono spostati in zone dello spazio degli obiettivi che il ranking giudica peggiori. Per ottenere questo effetto viene utilizzata una funzione che modifica i valori obiettivo.

Come prima cosa, si verifica se le soluzioni x_i rispettano il j-esimo vincolo calcolando il valore assoluto di $g_j(x_i)$. Successivamente il calcolo viene ripetuto per tutti i vincoli e i valori ottenuti vengono sommati, così da determinare la totale inammissibilità di x_i. La somma coinvolge solo le soluzioni che non rispettano i vincoli (minori di zero).

$$\Omega(x_i) = \sum_{j=1}^{K} \left| \min\left(g_j(x_i), 0\right) \right| \tag{B.2}$$

Il valore $\Omega(x_i)$ viene moltiplicato per R_m e sommato alla funzione obiettivo $f_m(x_i)$ per penalizzare la soluzione x_i.

$$F_m(x_i) = f_m(x_i) + R_m \cdot \Omega(x_i) \tag{B.3}$$

In questo modo si peggiora il valore delle funzioni obiettivo (in problemi di minimo) e la soluzione che non rispetta un vincolo avrà meno possibilità di essere selezionata, visto che avrà un ranking peggiore delle altre.

Il vero difetto di questa soluzione è il parametro R_m, introdotto per adattare il valore della funzione di penalizzazione $\Omega(x_i)$ alla scala della funzione obiettivo $f_m(x_i)$. Questo parametro determina completamente l'effetto del metodo sul problema: per valori troppo piccoli non funziona, mentre per valori troppo grandi diventa una tecnica *hard constrained*.

B.1.3 Jiménez Verdegay Goméz-Skameta

Questo metodo utilizza una variante dell'operatore di selezione chiamato *binary tournament selection*. Dall'insieme delle soluzioni selezionate vengono prese due soluzioni alla volta e vengono confrontate per formare la nuova popolazione:

a) se entrambe le soluzioni sono ammissibili si confrontano con un archivio di soluzioni ammissibili scegliendo quella non-dominata, se entrambe sono non-dominate o dominate si ricorre ad un meccanismo di selezione che valuta la distribuzione delle soluzioni.
b) se una è ammissibile e l'altra no si sceglie quella ammissibile.
c) se entrambe le soluzioni non sono ammissibili si confrontano con un archivio di soluzioni non ammissibili. In particolare si confrontano con la soluzione migliore di questo archivio e il paragone è in base ad un valore che quantifica la violazione dei vincoli. Se le entrambe soluzioni sono migliori o peggiori della soluzione dell'archivio viene utilizzato un criterio che sceglie la soluzione meglio distribuita. Invece se una è migliore e l'altra no si sceglie la migliore.

B.1.4 Constrained Tournament

Questo metodo viene implementato ridefinendo il concetto di dominanza. Una soluzione x_a domina x_b se è vera una delle seguenti condizioni:

a) x_a è ammissibile e non x_b.
b) entrambe non sono ammissibili, ma x_a viola meno i vincoli.
c) entrambe le soluzioni sono ammissibili e x_a domina x_b rispetto ai valori delle funzioni obiettivo.

Con questa definizione la popolazione viene orinata in fonti non-dominati e l'operatore di selezione premia le soluzioni che appartengono ai fronti migliori di dominanza e, a parità di fonte, sceglie le soluzioni meglio distribuite (*niche count*, *head count*, *crowding distance*).

B.1.5 Ray Tai Seow

Questo metodo utilizza tre ordinamenti della popolazione basati sul concetto di non-dominanza. Il primo R_{obj} valuta le funzioni obiettivo, il secondo R_{con} utilizza il valore che indica la violazione dei vincoli e il terzo R_{com} combina entrambi gli aspetti.

Per selezionare una nuova popolazione si scelgono tutte le soluzioni ammissibili (in base a R_{con}) all'interno del rango uno di R_{com}. Se la popolazione non è piena si sceglie la miglior soluzione x_a secondo R_{obj} e si scelgono x_b e x_c attraverso R_{con}; valutando quale soluzione è la migliore con il seguente criterio:

a) se entrambe le soluzioni x_b e x_c sono ammissibili si sceglie la soluzione che ha miglior rango in R_{obj}, se entrambe hanno lo stesso rango si ricorre ad un meccanismo di selezione che valuta la distribuzione delle soluzioni.
b) se una è ammissibile e l'altra no si sceglie quella ammissibile.
c) se entrambe le soluzioni x_b e x_c non sono ammissibili si sceglie la soluzione che ha miglior rango in R_{con}, se entrambe hanno lo stesso rango si ricorre ad una tecnica che coinvolge x_a. Si contano i vincoli rispettati a comune tra x_a e le due soluzioni x_b e x_c. Se sono maggiori quelli con x_b viene scelta questa, se sono maggiori quelli con x_c viene scelta quest'altra. Invece se sono lo stesso numero la scelta è casuale.

In base a questi tre criteri si può seleziona una soluzione tra x_b e x_c. Quella migliore contribuirà insieme a x_a a formare altre due soluzioni che faranno parte della nuova popolazione.

B.2 Considerazioni

I vincoli che considera l'algoritmo *ranking separation* non si possono esprimere con funzioni in un sistema simile a (b.1), infatti, sono definiti nello spazio degli obiettivi e non nello spazio di ricerca. Quest'aspetto semplifica notevolmente la trattazione dei vincoli perché si trovano nello stesso spazio di azione del ranking. Nel paragrafo b.1.2 abbiamo visto che il metodo *penalty function* modifica la posizione delle soluzioni non ammissibili

all'interno dello spazio degli obiettivi. In questo modo gli individui si trovano in aree differenti e viene cambiato l'ordinamento della non-dominanza. Nell'algoritmo *ranking separation* i vincoli sono definiti nello spazio di ricerca, per cui il ranking è in grado di riconoscere le soluzioni non ammissibili e costruire i fronti non-dominati in base a questa informazione. Chiaramente il nuovo ordinamento deve penalizzare le soluzioni non ammissibili.

I metodi presentati all'inizio in questa appendice non si possono applicare perché sono definiti nello spazio di ricerca. Anche pensando a varianti definite nello spazio degli obiettivi, non è possibile utilizzare queste tecniche all'interno dell'algoritmo *ranking separation*. Infatti, i vincoli del sistema (b.1) delimitano un area ammissibile caratterizzata da una forma convessa, mentre i coni dell'algoritmo sono definiti con vincoli in contrasto tra di loro e formano un'area ammissibile che non è continua nello spazio.

Riferimenti bibliografici

[1] Asanovic K., et. al., *The Landscape of Parallel Computing Research: A View from Berkeley*. Electrical Engineering and Computer Sciences, University of California at Berkeley, Technical Report No. UCB/EECS-2006-183, Dec. 18, 2006.

[2] Branche J., Schmeck H. and Deb K., *Parallelizing Multi-Objective Evolutionary Algorithm: Cone Separation*, 2004.

[3] Deb K., *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II*. 2002

[4] Deb K., *Multi-Objective Optiomization using Evolutionary Algorithms*. Chichester, U.K.: Wiley, 2001.

[5] Eldredge N. and Gould. S., *Punctuated equilibria: an alternative to phyletic gradualism*. In T. Schopf, editor, Models of Paleobiology, Freeman, Cooper, San Francisco, 1972. pp. 82–115

[6] Eshelman L.J. and Schaffer J.D., *Real-coded genetic algorithms and interval-schemata*, Foundations of Genetic Algorithms, vol.2, 1993. pp. 187–202

[7] Flynn M. J., *Very high-speed computing systems,* Proceedings of the IEEE Volume 54, Issue 12, Dec. 1966. pp. 1901-1909

[8] Fonseca C.M. and Fleming P.J., *Multiobjective Genetic Algorithms Made Easy: Selection, Sharing and Mating Restriction*. Proceedings of the 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications. Settembre: IEE, 1995. pp. 45-52

[9] Herrera F., Lozano M. and Verdegay J.L., *Tackling Real Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis*, 1998.

[10] Jensen M.T., *Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms*, 2003.

[11] Kung H.T., Luccio F. and Preparata F.P., On finding the maxima of a set of vector. Journal of the Association for Computing Machinery, 22(3), 1975. pp. 469-476

[12] Kursawe F., *A Variant of Evolution Strategies for Vector Optimization. Parallel Problem Solving from Nature, 1496*. Lecture Notes in Computer Science, edited by H. P. Schwefer and R. Männer. Berlino: Springer-Verlag, October 1990. pp 193-197

[13] Miki M., Hirotasu T. and Watanabe S., *The new model of parallel genetic algorithm in multi-objective optimization problems – divided range multi-objective genetic algorithm*. 2000

[14] Radcliffe N.J., *Equivalence Class Analysis of Genetic Algorithms*, 1991.

[15] Schaffer J.D., *Multiple objective optimization with vector evaluated genetic algorithms* in "Proceedings of the First International Conference on Genetic Algorithms", 1987. pp. 93-100

[16] Shalf J., *The New Landscape of Parallel Computer Architecture*. Journal of Physics: Conference Series 78, 2007.

[17] Skolicki Z. and De Jong K., *The influence of migration sizes and intervals on island models* in "Proceedings of the Geneticand Evolutionary Computation Conference". ACM Press, 2005.

[18] Veldhuizen D. Van, *Multiobjective Evolutionary Algorithms: Classification, Analyses and New Innovations*. Air Force Institute of Technology, Dayton, OH, Technical Report N° AFIT/DS/ENG/99-01, 1999.

[19] Whitley D., Rana S. and Heckendorn R. B., *The island model genetic algorithm: On separability, population size and convergence*. Journal of Computing and Information Technology, 1999. pp. 33–47

Indice analitico

best-worst policy 55
blx-α crossover 12
carico bilanciato 28
cluster 30; 32; 33
cluster of workstations 26
cono di separazione **31**
crossover 26; 35; 38; 43
crowded comparison operator 25
crowding distance **24**; 37; 43; 45; 63
cuboide 24
dimensione di migrazione 55
distance metric 58
diversità delle soluzioni 24
dominanza 19
evolutionary algorithm 18
exploitation 12
exploration 12
fitness totale 13
flat crossover 12
frammentazione 43
fronti 23
hard constrained 45; 46; 72
high-availability 27
high-performance computing 27
informazioni alto livello 16
insieme non-dominato 20
insieme ottimo di Pareto 20
instruction level parallelism 71
intervallo di migrazione 55
island model 55
join, punto di 33
load-balancing 27
massively parallel processing 26
mating pool 13
microcone assignment 46
microcone distance 50
microcono di separazione **40**
modello a isole 55
multiobjective genetic algorithm 20
multiobjective optimization problem 15
multiple instruction multiple data 26
multiple instruction single data 26
mutazione 26; 35; 38; 43
nadir, punto di 31; 38
nsga II **23**; 28; 35; 37; 51
obiettivi in conflitto 15
pipeline 27
point-by-point approach 18
preference-based strategy 17
proportional selection 13
punctuated equilibrium 56
random-random policy 55
ranking 25
ranking separation **45**; 48; 49; 50; 51; 55
separazione casuale **56**
single instruction multiple data 26
single instruction single data 26
single program multiple data 27
soft constrained 7; 45; 46