Laurea di Ingegneria Informatica Online
Esame di Intelligenza Artificiale
Seconda prova 2021-2022

GOBBI FRATTINI MATTEO
886787

Tema scelto: **ALGORITMI GENETICI**

1. DESCRIZIONE DELLA TEMATICA

Gli algoritmi genetici sono il tipo più diffuso di algoritmi evolutivi, ovvero algoritmi stocastici di ottimizzazione basati sui principi della selezione naturale e sui principi di genetica delle popolazioni, e vengono usati comunemente per problemi di ottimizzazione e di ricerca.
In un algoritmo genetico delle soluzioni di partenza evolvono verso una soluzione migliore attraverso operazioni ispirate alla biologia, come la mutazione e la ricombinazione. Ogni soluzione candidata è detta *individuo*, e il loro insieme è detto *popolazione*.

Per poter rendere applicabile lo stesso procedimento a un ampio spettro di problemi è necessario codificare i vari individui, qualsiasi cosa essi rappresentino nella realtà, allo stesso modo. Questa **rappresentazione**, negli algoritmi genetici, corrisponde a una stringa binaria, quindi ogni individuo è codificato come una sequenza di un numero fissato di bit che deve descrivere le sue caratteristiche. Ogni individuo viene chiamato anche *cromosoma*, e ogni bit che lo compone viene chiamato *gene*.

Un altro elemento fondamentale per far procedere l'evoluzione nella direzione giusta è la **funzione di fitness**, che valuta la qualità dell'individuo (la sua *fitness*), cioè quanto tale individuo si avvicina alla soluzione bersaglio.

Ora resta da definire ciò che fa avvenire l'evoluzione vera e propria della popolazione, ovvero gli **operatori**. Essi sono la selezione, la ricombinazione, la mutazione e la sostituzione:

La selezione è l'operatore che implementa il meccanismo di selezione naturale, e lo fa generando una nuova popolazione scegliendo gli individui secondo criteri basati sulla fitness. Questi criteri saranno analizzati nella sezione "metodologie".
Lo scopo della selezione è generare una nuova popolazione a cui sottoporre gli operatori di ricombinazione e mutazione, per ottenere così una nuova popolazione che verrà passata interamente o in parte alla generazione successiva secondo criteri di sostituzione.

La ricombinazione (o crossover) è l'operatore che implementa la riproduzione, ovvero fa sì che due individui "genitori" combinino il loro patrimonio genetico per generare un individuo "figlio". Possiamo dire che è la controparte della riproduzione sessuale biologica. Anche in questo caso ci sono diversi modi di applicare la ricombinazione, che saranno analizzati nella sezione "metodologie", ma in generale l'individuo figlio sarà formato da alcuni bit (geni) presi da un genitore e da alcuni presi dall'altro.

La mutazione è l'operatore che modifica i tratti del singolo individuo pseudocasualmente, analogamente a come avvengono mutazioni casuali in biologia. Essa comporta che ogni gene di un

individuo abbia una probabilità definita di mutare, ovvero il bit di cambiare valore da 0 a 1 o viceversa. Tale probabilità è solitamente molto piccola (un valore tipico è 0,01).

La sostituzione è l'operatore che determina quali saranno gli individui che costituiranno la prossima generazione. Esso sceglie questi individui tra quelli della popolazione originale della generazione precedente e quelli "nuovi" nati dai processi di selezione, ricombinazione e mutazione.
Ci sono diversi tipi di sostituzione, che verranno analizzati nella sezione "metodologie".

L'algoritmo genetico lavora secondo i seguenti passi:

1) Generazione casuale di una popolazione di cromosomi
2) Valutazione della fitness di questa popolazione
3) Selezione degli individui di questa popolazione per generarne una nuova
4) Applicazione dell'operatore di crossover sulla nuova popolazione
5) Applicazione dell'operatore di mutazione sulla nuova popolazione
6) Valutazione della fitness della nuova popolazione
7) Applicazione dell'operatore di sostituzione per creare la popolazione della generazione successiva
8) Ripartire dal passo 2

Questo processo si ripete finché non viene soddisfatta una condizione di **terminazione** (ad esempio il raggiungimento di una soluzione soddisfacente). Esistono diverse tipologie di condizioni di terminazione, che verranno analizzate nella sezione "metodologie".

Un particolare impiego degli algoritmi genetici è la loro applicazione all'evoluzione di programmi, chiamata **programmazione genetica** e introdotta da Koza nel 1992.
Essa ha la stessa struttura di esecuzione appena vista per gli algoritmi genetici, ma applicata a una rappresentazione in cui gli individui sono programmi e la funzione di fitness viene valutata eseguendo il programma e valutandone il tempo computazionale e/o la bontà dell'esecuzione rispetto al comportamento voluto.
Lo scopo è trovare un programma che realizza una certa funzionalità e viene visto come un modo per programmare i computer in modo automatico.

2. METODOLOGIE

Come già anticipato, ciascuno degli operatori descritti nella sezione precedente ha diversi modi in cui può essere applicato. Verranno analizzati uno ad uno di seguito.

Selezione

Il tipo di selezione più utilizzato è la selezione a roulette, nella quale ogni individuo ha probabilità di essere scelto direttamente proporzionale alla propria fitness. Il funzionamento è il seguente:

- Si calcola la somma delle fitness degli individui della popolazione
- Ogni individuo ha probabilità di essere estratto pari al rapporto tra la propria fitness è la somma delle fitness.
- Si posizionano queste probabilità su una roulette e si procede all'estrazione della popolazione successiva

È possibile che lo stesso individuo venga estratto più volte, avendo così più cloni nella nuova popolazione.

Un altro tipo di selezione è la selezione a torneo, in cui gli individui vengono raggruppati in "tornei", e viene estratto un individuo per ogni torneo in questo modo:

- Si estraggono casualmente degli individui dalla popolazione per essere messi nel torneo
- All'individuo con migliore fitness viene data probabilità p
- Al secondo migliore viene data probabilità p*(1-p)
- Al terzo migliore viene data probabilità $p*((1-p)^2)$, e così via
- Si estrae il vincitore del torneo
- Si ripete questo processo finché non ho estratto il numero di individui necessario

Un altro tipo di selezione è la selezione di Boltzmann, in cui la probabilità delle scelte caria a seconda della generazione, favorendo l'esplorazione all'inizio dell'algoritmo e gli individui migliori man mano che si va avanti con le generazioni.

Crossover

Il crossover avviene con una probabilità p_c il cui valore tipico è 0,6. Questa probabilità può essere implementata in due modi:

- Dalla popolazione di n individui vengono selezionate $(n/2)*p_c$ coppie di individui a cui applicare il crossover
- Ogni individuo che voglio generare ha probabilità p_c di essere "figlio" di crossover tra una coppia di individui e probabilità $1-p_c$ di essere "figlio" di un individuo unico, sostanzialmente clonando tale individuo

Uno dei tipi di crossover più diffuso è il crossover a un punto, che consiste nello scegliere casualmente un punto del cromosoma (ad esempio tra il terzo e il quarto bit), "tagliare" in due i cromosomi in quel punto e invertire la prima metà di uno con la prima metà dell'altro. Così facendo genereremo due figli, uno con la prima parte del genitore a e la seconda parte del genitore b, e l'altro viceversa.

Un altro tipo di crossover è il crossover a due punti, che, similmente al crossover a un punto, consiste nello scegliere casualmente due punti del cromosoma e scambiare tra i due cromosomi i geni (bit) compresi tra i due punti. Così facendo si genereranno due figli, uno con la parte centrale del genitore a e testa e coda del genitore b, e l'altro viceversa.

Il discorso del crossover a uno e due punti può essere esteso k volte per avere il crossover a k punti.

Un altro tipo di crossover molto importante è il crossover uniforme, che genera un singolo figlio decidendo, un gene per volta, se tale bit dovrà essere ereditato da un genitore o dall'altro, tramite un lancio di moneta. Tale moneta potrà anche avere un bias per un genitore o per l'altro, per far si che il figlio erediti più materiale genetico dal genitore con fitness più alta (crossover uniforme biased).

Un ulteriore tipo di crossover è il crossover aritmetico, che consiste nell'utilizzare un'operazione aritmetica per creare il nuovo individuo (come lo AND e OR tra i genitori).

Mutazione

La mutazione non ha grandi particolarità, l'unica cosa che si può variare è la probabilità di mutazione, che se troppo alta può portare a troppe mutazioni nei cromosomi a ogni generazione, che possono rendere vane le operazioni di crossover e selezione, mentre se troppo bassa può portare a una stagnazione delle soluzioni.

Sostituzione

Un criterio di sostituzione molto semplice e intuitivo è la sostituzione generazionale, in cui tutti i discendenti (la popolazione generata dalle operazioni di selezione, ricombinazione e mutazione) sostituiscono i parenti indipendentemente dai valori di fitness (tutti i discendenti e nessun genitore passano alla prossima generazione)

Un altro criterio è la sostituzione steady-state, in cui alcuni dei migliori discendenti sostituiscono i parenti peggiori, indipendente dai valori di fitness (può verificarsi che il migliore discendente abbia fitness peggiore dell'ultimo dei genitori, ma lo sostituirà comunque)

Il criterio più complesso è la sostituzione elitaria, che può essere di due tipi:
- $(\mu+\lambda)$ in cui dai $\mu+\lambda$ individui ottenuti combinando genitori e discendenti vengono scelti i migliori μ individui
- (μ, λ) in cui μ tra i migliori λ discendenti sostituiscono μ genitori

Praticamente così facendo mi ritrovo con una combinazione degli individui migliori (alcuni o tutti) tra i genitori e i discendenti.

Terminazione

Ci sono diversi criteri che possono essere soddisfatti per portare alla terminazione dell'algoritmo genetico:
- Il raggiungimento di una soluzione soddisfacente, o comunque che soddisfi dei criteri minimi
- Il raggiungimento di un numero massimo di generazioni
- Il raggiungimento del limite massimo di tempo computazionale
- Il mancato miglioramento di fitness nelle ultime generazioni
- Ispezione manuale

Oppure una combinazione di un insieme dei criteri elencati.

Non è possibile valutare l'efficacia dei vari tipi delle varie operazioni in generale, perché ognuno è più o meno adatto al singolo problema che si vuole risolvere.

3. ESEMPIO APPLICATIVO

Un esempio applicativo di un algoritmo genetico è la ricerca della soluzione del gioco "Mastermind". In questo gioco un giocatore è il "codificatore" e ha diversi colori per formare un codice di quattro pallini colorati. Dopo che il codificatore ha finito, il suo avversario fa il suo primo tentativo di indovinare tale codice, al termine del quale il codificatore dovrà comunicare:
- Il numero di pallini del colore giusto nella posizione giusta
- Il numero di pallini del colore giusto, ma nella posizione sbagliata

Preso atto di queste informazioni, il decodificatore procederà con il suo secondo tentativo, e così via.

Dato che non è possibile sapere a priori quale combinazione dei vari tipi di crossover, selezione, mutazione e sostituzione sia la più efficiente per questo problema specifico, ho pensato di programmare questo algoritmo genetico in C++, inserendo diversi tipi di operatori e lasciandomi la possibilità di modificare facilmente tra un' esecuzione e l'altra le varie probabilità, i pesi per il calcolo della fitness, il tipo di crossover e il tipo di sostituzione.

Così facendo ho creato, in un certo senso, un problema molto semplificato di programmazione genetica, in cui userò un algoritmo genetico per trovare la combinazione di operatori e probabilità più efficiente per il problema del Mastermind valutandone la fitness simulandone l'esecuzione.

Vediamo prima come ho codificato l'algoritmo per la ricerca della soluzione del Mastermind, poi applichiamo l'algoritmo genetico.

Mastermind

Rappresentazione

Ogni individuo (o cromosoma) è una stringa binaria di 8 bit, che rappresentano i 4 pallini con due bit a testa. È quindi possibile giocare con quattro colori. I primi due bit codificheranno il colore del primo pallino, il terzo e il quarto del secondo pallino, e così via.
In particolare nel mio codice ho definito ogni cromosoma come una struttura composta da un intero che rappresenta la fitness e dalla combinazione di pallini, codificata come un vettore di quattro stringhe, ognuna delle quali contiene i due bit di un pallino:

```
struct Chromosome{
    string guess[4];
    int fitness;
};
```

La popolazione, che sarà di N individui (altro valore che può variare tra un'esecuzione e l'altra), sarà rappresentata come un vettore di cromosomi.

Funzione di fitness

La funzione di fitness conta quanti pallini sono del colore giusto al posto giusto, quanti del colore giusto al posto sbagliato e moltiplica questi due valori per dei coefficienti. Questi coefficienti sono una delle variabili modificabili tra un'esecuzione e l'altra.
Nel codice questa funzione riceve come parametro la stringa "guess", la confronta con la stringa globale "code", conta i pallini e ritorna il valore di fitness:

```
    int value=totalRight*G+rightColor*S;
    return value;
```

Selezione

La selezione è la più diffusa, ovvero la selezione a roulette così come già descritta nella sezione "metodologie". Nel mio codice è implementata attraverso due funzioni, la funzione *roulette* e la funzione *selection*. Questo perché la funzione roulette viene riutilizzata anche in fasi successive dell'algoritmo.

In particolare la funzione *roulette* riceve come parametri una popolazione e un numero *num*, esegue *num* estrazione a roulette sulla popolazione, e ritorna un array di interi che rappresentano i coefficienti nel vettore popolazione dei cromosomi estratti:

```
vector<int> roulette(vector<Chromosome> population, int num){

    vector<int> prob, c;
    int totalFitness=0, totalProb=0;

    for(int i=0;i<population.size();i++)
        totalFitness+=population[i].fitness;
    for(int i=0;i<population.size();i++){
        int tempProb=(population[i].fitness*100)/totalFitness;
        prob.push_back(tempProb);
        totalProb+=prob[i];
    }

    for(int i=0;i<num;i++){

        int r=rand() % totalProb;
        int temp=0;

        for(int j=0;j<prob.size();j++){
            temp+=prob[j];
            if(r<temp){
                c.push_back(j);
                break;
            }
        }
    }
    return c;
}
```

La funzione *selection* genera un vettore di cromosomi *newPopulation* chiamando la funzione *roulette* con gli opportuni parametri:

```
vector<Chromosome> selection(vector<Chromosome> population){

    vector<Chromosome> newPopulation;
    vector<int> c=roulette(population, N);
    for(int i=0;i<N;i++){
        newPopulation.push_back(population[c[i]]);
    }
    return newPopulation;
}
```

Crossover

Per quanto riguarda il crossover bisogna analizzare due funzionalità:

- Il modo in cui vengono scelti i genitori
- Il tipo di crossover utilizzato

Ho implementato quattro modi in cui possono venir scelti i genitori, ovvero possono essere scelti i due migliori individui della popolazione, i due peggiori, due casuali oppure due estratti tramite roulette. Tutte queste funzioni, come la già vista funzione *roulette*, ritornano un vettore con i coefficienti degli individui scelti nel vettore popolazione:

```
vector<int> findBest(vector<Chromosome> pop){
    int b1=0, b2=1;
    if(pop[b1].fitness<pop[b2].fitness)
        swap(b1, b2);
    for(int i=2;i<pop.size();i++){
        if(pop[i].fitness>pop[b1].fitness){
            swap(b1, b2);
            b1=i;
        }
        else if(pop[i].fitness>pop[b2].fitness)
            b2=i;
    }
    vector<int> foundBest;
    foundBest.push_back(b1); foundBest.push_back(b2);
    return foundBest;
}

vector<int> findWorst(vector<Chromosome> pop){
    int w1=0, w2=1;
    if(pop[w1].fitness>pop[w2].fitness)
        swap(w1, w2);
    for(int i=2;i<pop.size();i++){
        if(pop[i].fitness<pop[w1].fitness){
            swap(w1, w2);
            w1=i;
        }
        else if(pop[i].fitness<pop[w2].fitness)
            w2=i;
    }
    vector<int> foundWorst;
    foundWorst.push_back(w1); foundWorst.push_back(w2);
    return foundWorst;
}

vector<int> findRandom(vector<Chromosome> pop){
    int c1=0, c2=0;
    while(c1==c2){
        c1=rand()%N;
        c2=rand()%N;
    }
    vector<int> foundRandoms;
    foundRandoms.push_back(c1); foundRandoms.push_back(c2);
    return foundRandoms;
}
```

La scelta di quale di queste funzioni utilizzare avviene tramite uno switch nel *main*:

```
vector<int> parents;
switch(PARENTSTYPE){
    case 0:
        parents=findRandom(newPopulation);
        break;
    case 1:
        parents=findBest(newPopulation);
        break;
    case 2:
        parents=findWorst(newPopulation);
        break;
    case 3:
        parents=roulette(newPopulation, 2);
        break;
}
```

Per quanto riguarda il tipo di crossover, invece, ho implementato quattro funzioni:

- Crossover a un punto
- Crossover a due punti
- Crossover uniforme
- Crossover biased

Tutte queste funzioni ricevono come parametri la popolazione (per riferimento) e i due coefficienti dei due genitori. In più la funzione del crossover biased riceve anche il tasso di bias. Anche in questo caso la scelta di quale di queste funzioni utilizzare avviene tramite uno switch nel *main*:

```
switch(CROSSTYPE){
    case 0:
        crossoverU(newPopulation, parents[0], parents[1]);
        break;
    case 1:
        crossover1(newPopulation, parents[0], parents[1]);
        break;
    case 2:
        crossover2(newPopulation, parents[0], parents[1]);
        break;
    case 3:
        biasedCrossoverU(newPopulation, CROSSBIAS, parents[0], parents[1]);
        break;
}
```

Il codice delle varie funzioni è il seguente:

<u>Crossover a un punto</u>

```
void crossover1(vector<Chromosome> &population, int c1, int c2){

    int point=rand()%3+1;

    for(int i=0;i<point;i++)
        swap(population[c1].guess[i], population[c2].guess[i]);

}
```

Il punto dove effettuare il crossover viene selezionato casualmente, ma la scelta e limitata ai punti tra un pallino e l'altro.

Crossover a due punti

```
void crossover2(vector<Chromosome> &population, int c1, int c2){

    int point1=0, point2=0;

    while(point1==point2){
        point1=rand()%5;
        point2=rand()%5;
    }
    if(point2<point1)
        swap(point1, point2);

    for(int i=point1;i<point2;i++)
        swap(population[c1].guess[i], population[c2].guess[i]);

}
```

Anche in questo caso i punti vengono effettuati casualmente ma viene limitata la scelta in modo da impedire che vengano separati due bit che indicano lo stesso pallino.

Crossover uniforme

In questo crossover uniforme c'è una probabilità di 0,5 che l'ennesimo gene di un genitore venga scambiato con l'ennesimo gene dell'altro genitore (anche in questo con geni si intende la codifica di un pallino e non il singolo bit).

```
void crossoverU(vector<Chromosome> &population, int c1, int c2){

    int r;
    for(int i=0;i<4; i++){
        r=rand()%100;
        if(r<50)
            swap(population[c1].guess[i], population[c2].guess[i]);
    }
}
```

Crossover biased

In questo crossover si individua quale dei genitori ha fitness maggiore e, dopo averlo fatto, si generano i due figli facendo sì che ogni gene abbia probabilità di bias di venir preso dal genitore migliore (anche in questo con geni si intende la codifica di un pallino e non il singolo bit).

```
void biasedCrossover(vector<Chromosome> &population, int bias, int c1, int c2){

    int r;

    if(population[c1].fitness<population[c2].fitness)
        swap(c1, c2);

    vector<Chromosome> children;
    Chromosome child;

    for(int i=0;i<2;i++){
        for(int j=0;j<4;j++){
            r=rand()%100;
            if(r<bias)
                child.guess[j]=population[c1].guess[j];
            else
                child.guess[j]=population[c2].guess[j];
        }
        children.push_back(child);
    }

    population[c1]=children[0];
    population[c2]=children[1];

}
```

Mutazione

La mutazione avviene secondo un parametro di probabilità che indica per ogni bit qual è la probabilità che esso muti. Nel codice viene implementata attraverso due funzioni:

- Una funzione *invert* che riceve come parametri la stringa della codifica del pallino (per riferimento) e un valore *k* che indica quale dei due bit della stringa deve mutare, e inverte tale bit.
- Una funzione *mutate* che riceve come parametri la popolazione (per riferimento) e la probabilità di mutazione e decide per ogni bit se effettuare la mutazione (chiamando *invert*) o meno.

```
void invert(string &s, int p){
    if(s[p]=='1')
        s.replace(p, 1, "0");
    else if(s[p]=='0')
        s.replace(p, 1, "1");
}

void mutate(int prob, vector<Chromosome> &population){

    for(int i=0; i<population.size();i++){
        for(int j=0;j<4;j++){
            for(int k=0;k<2;k++){
                int e=rand()%100;
                if(e<prob){
                    invert(population[i].guess[j], k);
                }
            }
        }
    }
}
```

Sostituzione

Ho implementato due tipi di sostituzione: la sostituzione generazionale e la sostituzione elitaria.
La sostituzione generazionale è implementata molto semplicemente rimpiazzando la popolazione genitore con la popolazione *newPopulation*, ovvero quella che è stata generata attraverso le operazioni di selezione, crossover e mutazione.
La sostituzione elitaria è invece implementata dalla funzione *elitistSub* che riceve come parametri le due popolazioni (quella genitrice e quella discendente), le unisce in un unico vettore di cromosomi, lo ordina per fitness crescente e ritorna la nuova popolazione, formata dagli ultimi N elementi dell'array ordinato (quindi quelli con maggiore fitness):

```
vector<Chromosome> elitistSub(vector<Chromosome> a, vector <Chromosome> b){
    vector<Chromosome> newPop;
    a.insert(a.end(), b.begin(), b.end());
    sort(a.begin(), a.end(), [](Chromosome x, Chromosome y){return x.fitness < y.fitness;});
    for(int i=0;i<N;i++){
        newPop.push_back(a.back());
        a.pop_back();
    }
    return newPop;
}
```

Anche in questo caso la scelta di quale tipo di sostituzione utilizzare avviene tramite uno switch nel *main*:

```
switch(SUBTYPE){
    case 0:
        population=newPopulation;
        break;
    case 1:
        population=elitistSub(population, newPopulation);
        break;
}
```

Terminazione

La terminazione avviene quando si è trovata la soluzione, ovvero quando si è generato un cromosoma che ha per tutti e quattro i pallini "colore giusto al posto giusto". Quando ciò avviene, quell'individuo avrà una fitness pari a 4*G, dove G è il coefficiente per "colore giusto al posto giusto".
Il raggiungimento di tale condizione si controlla tramite una variabile booleana globale *guessed* che viene inizializzata su falso, e che diventerà vera quando si troverà un valore di fitness pari a 4*G. Questo test viene effettuato ogni volta che viene calcolata la fitness degli individui della popolazione, cosa che avviene tramite la funzione *calculateFitness*:

```
void calculateFitness(vector<Chromosome> &population){
    for(int i=0; i<population.size();i++){
        population[i].fitness=fitness(population[i].guess);
        if(population[i].fitness==(4*G))
            guessed=true;
    }
}
```

Esecuzione

L'esecuzione comincia generando casualmente il codice da indovinare *code* e N cromosomi (la popolazione) attraverso le funzioni *generateChromosome* e *generateGene*:

```
string generateGene(){
    string S="";
    for(int i=0;i<2;i++)
        S += to_string(((int)rand()%2));
    return S;
}

void generateChromosome(string chromosome[4]){

    for(int i=0;i<4;i++)
        chromosome[i]=generateGene();
}
```

```
int main(){

    int gen=0;
    srand (time(NULL));
    generateChromosome(code);
    vector<Chromosome> population;
    for(int i=0;i<N;i++){
        Chromosome c;
        generateChromosome(c.guess);
        c.fitness=fitness(c.guess);
        if(c.fitness==(4*G))
            guessed=true;

        population.push_back(c);
    }
```

Dopo aver generato la popolazione, e aver controllato che tra di essa non ci sia già la soluzione, si entra in un ciclo *while* che ha come condizione *!guessed*.
In questo ciclo viene incrementato il valore *gen* che tiene conto del numero di generazioni, viene generata una nuova popolazione tramite la selezione, ad essa vengono applicati gli opportuni operatori di crossover e mutazione secondo i rispettivi switch, viene calcolata la sua fitness tramite la funzione *calculateFitness* e infine viene fatta l'opportuna sostituzione secondo lo switch.
L'operatore di crossover viene applicato una sola volta per generazione, secondo la formula $(n/2)*p_c$. Infatti useremo popolazioni di 2, 3, 4 e 5 individui, per tutti questi valori la formula ci dice di fare il crossover su una coppia di individui.
Se in questi passaggi è stata trovata la soluzione, la funzione *calculateFitness* porrà la variabile *guessed* a true, quindi si uscirà dal ciclo e si stamperà la generazione in cui si è trovata la soluzione.

Un esempio di ouput dell'esecuzione è il seguente:

```
1 generazione
2 generazione
Combinazione trovata alla 2 generazione
```

Valori modificabili

Come già detto, in questo programma ci sono diversi valori modificabili al fine di trovare la combinazione di essi più efficiente per questo tipo di problema. Essi sono stati dichiarati come costanti a inizio codice, e sono:

```
const int N=4; //numero di individui nella popolazione
const int G=5; //coefficiente fitness giusto al posto giusto
const int S=2; //coefficiente fitness giusto al posto sbagliato
const int PROBMUT=5; //probabilità di mutazione
const int CROSSBIAS=85; //bias biased crossover
const int SUBTYPE=1; //Tipo di sostituzione: 0 generazionale, 1 elitaria
const int CROSSTYPE=3; //Tipo di crossover: 0 uniforme, 1 un punto, 2 due punti, 3 biased
const int PARENTSTYPE=3; //Tipo di scelta dei genitori: 0 casuale, 1 migliori, 2 peggiori, 3 a roulette
```

Modificando opportunamente questi valori secondo un algoritmo si troverà la combinazione più efficiente per la risoluzione di questo problema.

Programmazione genetica

Ora che abbiamo pronto il codice per simulare il comportamento delle varie combinazioni di algoritmi possiamo applicare un algoritmo genetico, che sarà così strutturato:

Rappresentazione

Ogni individuo è una stringa binaria di 13 bit, che rappresentano le seguenti caratteristiche:

Bit	Caratteristica	Variabile nel Codice
2	Numero di individui nella popolazione	*N*
2	Combinazione di coefficienti per la funzione di fitness	*G* e *S*
2	Tipo di scelta dei genitori per il crossover	*PARENTSTYPE*
2	Tipo di crossover	*CROSSTYPE*
2	Percentuale di bias per biased crossover	*CROSSBIAS*
2	Percentuale di probabilità di mutazione	*PROBMUT*
1	Tipo di sostituzione	*SUBTYPE*

Più in particolare, le caratteristiche sono codificate in questo modo:
Numero di individui nella popolazione:

- 00 -> 2
- 01 -> 3
- 10 -> 4
- 11 -> 5

Combinazione di coefficienti per la funzione di fitness: (S, G)

- 00 -> 1, 2
- 01 -> 1, 3
- 10 -> 1, 5
- 11 -> 2, 5

Tipo di scelta dei genitori per il crossover:
- 00 -> Casuale
- 01 -> Migliori
- 10 -> Peggiori
- 11 -> Roulette

Tipo di crossover:
- 00 -> Uniforme
- 01 -> Un punto
- 10 -> Due punti
- 11 -> Biased

Percentuale di bias per biased crossover:
- 00 -> 60
- 01 -> 70
- 10 -> 80
- 11 -> 90

Percentuale di probabilità di mutazione:
- 00 -> 1
- 01 -> 5
- 10 -> 10
- 11 -> 15

Tipo di sostituzione:
- 0 -> Generazionale
- 2 -> Elitaria

Funzione di fitness

Per valutare la bontà di un programma usiamo il numero medio di generazioni che esso ha, quindi simuliamo cinque volte l'esecuzione, eliminiamo il valore migliore e il peggiore (per non dar peso a situazioni particolarmente fortunate e sfortunate) e facciamo la media dei restanti tre valori.
Così facendo abbiamo però una situazione in cui il valore più basso è il migliore, ma vogliamo una funzione di fitness che cresca per individui migliori. Per realizzare questo dividiamo 1 per il numero di generazioni usate, avendo così fitness function:

$$f(x)=1/gen(x)$$

dove gen(x) è il numero medio di generazioni che impiega il programma x a trovare la soluzione.

Selezione

Per questo problema usiamo selezione a roulette.

Crossover

Per questo problema usiamo crossover uniforme su due coppie di individui per ogni generazione.

Mutazione

In questo problema ogni bit può mutare con probabilità di mutazione 0,1.

Sostituzione

In questo problema usiamo sostituzione generazionale.

Terminazione

Essendo questo un esempio applicativo a scopo didattico, poniamo come condizione di terminazione il raggiungimento della sesta generazione, anche se probabilmente insufficiente a trovare una soluzione ottima.

Esecuzione

Cominciamo generando casualmente una popolazione di sei individui:

00	11	10	00	10	00	1
10	01	00	10	10	00	0
10	11	11	00	01	01	1
01	00	01	00	01	01	1
01	00	11	01	01	01	0
11	01	11	01	00	00	1

Valutiamo la fitness di questi individui simulando per cinque volte:

Individuo	1	2	3	4	5	Media	Fitness
0011100010001	28	15	~~125~~	~~3~~	85	43	0.0233
1001001010000	~~28~~	~~984~~	543	112	277	311	0.0032
1011110001011	30	14	25	~~63~~	~~10~~	23	0.0435
0100010001011	~~21~~	11	17	3	~~2~~	10	0.1000
0100110101010	56	54	~~58~~	~~31~~	33	48	0.0208
1101110100001	~~110~~	98	63	43	~~25~~	68	0.0147

Generazione 1

Effettuo la selezione a roulette di sei elementi. Fitness totale = 0,2055

La probabilità di ognuno è la propria fitness diviso la fitness totale. Le estrazioni avvenute sono:
5-4-2-3-4-4

Estraggo due coppie su cui effettuare il crossover: 4-1 e 2-6, ovvero il quarto estratto (3) con il primo estratto (5) e il secondo estratto (4) con il sesto estratto (4).

Dobbiamo quindi svolgere il crossover tra l'individuo 5 e l'individuo 3 della popolazione iniziale e tra l'individuo 4 e l'individuo 4 della popolazione iniziale. Il secondo crossover non ha conseguenze perché è fatto su due cloni, mentre il primo crossover genera i seguenti individui:

- 1001110001011
- 1010110001011

La nuova popolazione è quindi la seguente:

0100010001011
0100010001011
0100010001011
1001001010000
1001110001011
1010110001011

Attuo l'operatore di mutazione, i bit che mutano sono:
- Il dodicesimo del primo individuo
- Il quinto del quarto individuo

La popolazione attuale è quindi:

01	00	01	00	01	00	1
01	00	01	00	01	01	1
01	00	01	00	01	01	1
10	01	10	10	10	00	0
10	01	11	00	01	01	1
10	10	11	00	01	01	1

Valutiamo la fitness dei nuovi individui e riportiamo la fitness di quelli già osservati:

Individuo	1	2	3	4	5	Media	Fitness
0100010001001	~~2249~~	268	~~17~~	53	85	135	0.0074
0100010001011							0.1000
0100010001011							0.1000
1001101010000	~~138~~	75	47	~~35~~	125	82	0.0121
1001110001011	23	16	~~94~~	27	~~1~~	22	0.0454
1010110001011	23	~~6~~	8	23	~~23~~	18	0.0556

Considerando che stiamo usando la sostituzione generazionale, tutti i nuovi individui passano avanti.

<u>Generazione 2</u>

Effettuo la selezione a roulette di sei elementi. Fitness totale = 0,3205

La probabilità di ognuno è la propria fitness diviso la fitness totale. Le estrazioni avvenute sono:
6-6-3-2-6-2.

Estraggo due coppie su cui effettuare il crossover: 4-6 e 1-3, ovvero il quarto estratto (2) con il sesto estratto (2) e il primo estratto (6) con il terzo estratto (3).
La prima coppia, essendo crossover di cloni, non porta modifiche alla popolazione, mentre la seconda coppia genera i seguenti individui:

- 1010010001011
- 0110110001011

La nuova popolazione è quindi la seguente:

1010110001011
1010110001011
0100010001011
0100010001011
1010010001011
0110110001011

Attuo l'operatore mutazione. I bit che mutano sono:

- Il settimo bit del sesto individuo

La nuova popolazione è quindi:

10	10	11	00	01	01	1
10	10	11	00	01	01	1
01	00	01	00	01	01	1
01	00	01	00	01	01	1
10	10	01	00	01	01	1
01	10	11	10	01	01	1

Valutiamo la fitness dei nuovi individui e riportiamo la fitness di quelli già osservati:

Individuo	1	2	3	4	5	Media	Fitness
1010110001011							0.0556
1010110001011							0.0556
0100010001011							0.1000
0100010001011							0.1000
1010010001011	~~17~~	30	30	21	~~34~~	27	0.0370
0110111001011	~~287~~	~~11~~	70	81	11	54	0.0185

Considerando che stiamo usando la sostituzione generazionale, tutti i nuovi individui passano avanti.

<u>Generazione 3</u>

Effettuo la selezione a roulette di sei elementi. Fitness totale = 0,3667

La probabilità di ognuno è la propria fitness diviso la fitness totale. Le estrazioni avvenute sono:
3-4-3-4-2-3

Estraggo due coppie su cui effettuare il crossover: 2-4 e 1-6. In questo caso entrambe le operazioni di crossover non portano cambiamenti alla popolazione in quanto sono tra due individui cloni.

La nuova popolazione è la seguente:

1010110001011
0100010001011
0100010001011
0100010001011
0100010001011
0100010001011

Attuo l'operatore mutazione. I bit che mutano sono:
- L'undicesimo del sesto individuo

La nuova popolazione è quindi:

10	10	11	00	01	01	1
01	00	01	00	01	01	1
01	00	01	00	01	01	1
01	00	01	00	01	01	1
01	00	01	00	01	01	1
01	00	01	00	01	11	1

Valutiamo la fitness dei nuovi individui e riportiamo la fitness di quelli già osservati:

Individuo	1	2	3	4	5	Media	Fitness
1010110001011							0.0556
0100010001011							0.1000
0100010001011							0.1000
0100010001011							0.1000
0100010001011							0.1000
0100010001111	20	7	7	~~51~~	~~6~~	11	0.0909

Considerando che stiamo usando la sostituzione generazionale, tutti i nuovi individui passano avanti.

<u>Generazione 4</u>

Effettuo la selezione a roulette di sei elementi. Fitness totale = 0,5465

La probabilità di ognuno è la propria fitness diviso la fitness totale. Le estrazioni avvenute sono:
2-2-5-4-6-2

Estraggo due coppie su cui effettuare il crossover: 4-5 e 3-1, ovvero tra il primo estratto (2) e il terzo estratto (5) e tra il quarto estratto (4) e il quinto estratto (6). La coppia 2-5 non porta a cambiamenti in quanto i due genitori sono cloni, mentre la coppia 4-5 genera gli individui:

- 0100010001011
- 0100010001111

La nuova popolazione è la seguente:

0100010001011
0100010001011
0100010001011
0100010001011
0100010001011
0100010001111

Attuo l'operatore mutazione. I bit che mutano sono:
- Il primo del quarto individuo
- Il quinto del quinto individuo
- Il settimo del sesto individuo
- L'ottavo del sesto individuo

La nuova popolazione è quindi:

01	00	01	00	01	01	1
01	00	01	00	01	01	1
01	00	01	00	01	01	1
11	00	01	00	01	01	1
01	00	11	00	01	01	1
01	00	01	11	01	11	1

Valutiamo la fitness dei nuovi individui e riportiamo la fitness di quelli già osservati:

Individuo	1	2	3	4	5	Media	Fitness
0100010001011							0.1000
0100010001011							0.1000
0100010001011							0.1000
1100010001011	16	8	~~39~~	~~7~~	29	18	0.0555
0100110001011	~~59~~	38	28	~~10~~	33	33	0.0303
0100011101111	8	~~40~~	8	~~5~~	10	9	0.1111

Considerando che stiamo usando la sostituzione generazionale, tutti i nuovi individui passano avanti.

<u>Generazione 5</u>

Effettuo la selezione a roulette di sei elementi. Fitness totale = 0,4969

La probabilità di ognuno è la propria fitness diviso la fitness totale. Le estrazioni avvenute sono:
2-1-3-6-6-5

Estraggo due coppie su cui effettuare il crossover: 1-3 e 4-5. Entrambe le coppie sono di individui cloni, quindi il crossover non porta a variazioni nella popolazione.

La nuova popolazione è la seguente:

0100010001011
0100010001011
0100010001011
0100110001011
0100011101111
0100011101111

Attuo l'operatore mutazione. I bit che mutano sono:
- Il nono del terzo individuo

La nuova popolazione è quindi:

01	00	01	00	01	01	1
01	00	01	00	01	01	1
01	00	01	00	11	01	1
01	00	11	00	01	01	1
01	00	01	11	01	11	1
01	00	01	11	01	11	1

Valutiamo la fitness dei nuovi individui e riportiamo la fitness di quelli già osservati:

Individuo	1	2	3	4	5	Media	Fitness
0100010001011							0.1000
0100010001011							0.1000
0100010011011	21	18	~~1~~	36	~~259~~	25	0.0400
0100110001011							0.0555
0100011101111							0.1111
0100011101111							0.1111

Considerando che stiamo usando la sostituzione generazionale, tutti i nuovi individui passano avanti.

<u>Generazione 6</u>

Abbiamo raggiunto la condizione di terminazione (raggiungimento sesta generazione).

Possiamo dire che il programma più efficiente trovato è 0100011101111, ovvero il programma che usa
- Popolazioni da 3 individui
- Funzione di fitness con coefficiente 1 per i colori giusti al posto sbagliato e 2 per colori giusti al posto giusto
- Scelta dei genitori migliori per effettuare il crossover
- Crossover biased
- Bias del 70%
- Probabilità di mutazione del 15%

- Sostituzione elitaria

Siamo sicuri che sia questo il programma più efficiente?
No, è probabilmente non lo è. Per trovare il programma più efficiente sarebbero necessarie moltissime generazioni in più, ma essendo un esempio applicativo svolto su carta e non simulato con un calcolatore mi sono limitato a un numero di generazioni finito a scopo dimostrativo.

Un’esecuzione di algoritmo genetico fino al raggiungimento della soluzione desiderata è quella ottenuta attraverso il codice che ho usato per simulare il gioco del Mastermind.