Università degli Studi di Salerno

Dipartimento di Informatica

Corso di Laurea Triennale
Fondamenti di Intelligenza Artificiale

Elaborato Finale

STUDY HALL

Professore:
Fabio Palomba

Partecipanti:
Maria Immacolata Colella
[0512110431]
Michele Mattiello
[0512110185]
Carlo Sorrentino
[0512110884]

https://github.com/IS-studyHall/FiaModule.git

Anno Accademico 2022–2023

Indice

1 Introduzione **3**
1.1 Obiettivi 3

2 Analisi dell'Ambiente **4**
2.1 Specifica P.E.A.S. 4
2.2 Caratteristiche dell'ambiente 4

3 Algoritmi genetici **5**
3.1 Inizializzazione 6
3.2 Selezione 7
3.3 Algoritmo di Crossover 7
3.4 Algoritmo di Mutazione 8
3.5 Stopping Condition 8
3.6 Funzione di fitness 9
3.7 Improvement 9
3.7.1 Fronte di Pareto 9
3.7.2 Preference Sorting 9

4 Algoritmi **10**
4.1 Prima Funzione di Fitness 10
4.2 Seconda Funzione di Fitness 11
4.3 Inizializzazione 12
4.4 Roulette Wheel Selection 13
4.5 Single Point Crossover 13
4.6 Two Point Crossover 14
4.7 Inversion Mutation 14

5 Conclusioni **15**

1 Introduzione

Il Progetto nasce dall'idea di voler ottimizzare l'applicativo proposto al corso di Ingegneria del Software, il cui obiettivo è quello di mettere a disposizione degli studenti una piattaforma che permetta la prenotazione di un posto, presso le aule studio, da qualunque luogo che abbia accesso a Internet; questo perché tutti necessitano di un ambiente di studio sereno ed accessibile ma non sempre si è in grado di evitare lo spreco di tempo dovuto al ricercare posti liberi in giro per l'intera l'Università.

1.1 Obiettivi

Il progetto nasce col fine di aiutare gli studenti nella risoluzione di questa problematica, cercando un modo per disporli nelle aule fornite dall'Ateneo, in maniera efficace, tenendo conto di due considerazioni: le loro esigenze ed il numero di ore che intendono riservare allo studio.

Lo scopo del progetto, quindi, è proprio quello di dare la possibilità agli studenti di indicare sia in quali giorni e in quali fasce orarie si offre disponibilità a studiare che quante ore settimanali si intendono dedicare ad esso, cosi da poter definire un'ottima assegnazione dei posti nelle aule studio.

L'ottimalità che si intende raggiungere è quella di lasciare quanti più posti liberi nel minor numero possibile di aule studio, in modo tale da fornire agli studenti uno spazio che garantisca un posto ma che allo stesso non risulti troppo caotico, in previsione di ingenti prenotazioni.

2 Analisi dell'Ambiente

2.1 Specifica P.E.A.S.

Di seguito viene riportata la specifica relativa alla situazione corrente:

- **Performance:** le misure di prestazione sono adottate per valutare l'operato di un agente; viene valutato se esso riesce ad allocare le prenotazioni in modo tale che rispettino le richieste dello studente.
- **Environment:** la descrizione degli elementi che formano l'ambiente; i dataset preesistenti, i dati in input forniti dagli utenti, tutte le combinazioni di prenotazioni.
- **Actuators:** gli attuatori disponibili all'agente per intraprendere le azioni; l'agente agisce assegnando posti nelle aule studio in base agli slot (fasce orarie) di prenotazione tra cui scegliere nell'arco della settimana.
- **Sensors:** i sensori attraverso i quali l'agente riceve gli input percettivi costituiti dall'insieme dei dati forniti in input finalizzati al raggiungimento dell'obiettivo; una semplice lettura da file (datatset).

2.2 Caratteristiche dell'ambiente

L'ambiente operativo è:

- **Completamente Osservabile:** i sensori di un ambiente danno accesso allo stato completo dell'ambiente in ogni momento; conoscono quante prenotazioni sono state effettuate e quanti posti sono disoponibili/occupati.
- **Deterministico:** lo stato successivo dell'ambiente dipende dallo stato corrente e dall'azione eseguita dall'agente; di fatto l'ambiente a seconda di dove l'agente assegna il posto allo studente, varia e quindi esso dipende esclusivamente dall'operato dell'agente.
- **Statico:** l'ambiente è invariato mentre l'agente sta deliberando ed una volta raccolti tutti gli input percettivi, effettuerà delle azioni che successivamente andranno a modificare lo stato corrente dell'ambiente; gli studenti potranno fornire all'agente i dati di input solo nei primi 5 giorni settimanali, poi in uno dei restanti due l'agente si ritirerà per deliberare ed è in questi giorni che l'ambiente non potrà subire modifiche restando invariato.
- **Discreto:** l'ambiente fornisce un numero limitato di percezioni e azioni distinte, chiaramente definite, in quanto vi è una soglia massima di ore di studio settimanali possibili (40) e un numero di posti limitato a seconda dell'aula studio.
- **Singolo:** l'ambiente consente la presenza di un unico agente.

3 Algoritmi genetici

Una prima idea per l'implementazione del progetto ricade sugli algoritmi di ricerca locale, in quanto essi permettono di esplorare lo spazio degli stati (cioè l'insieme delle configurazioni che portano ad uno stato obiettivo) per trovare una soluzione senza tener conto del percorso utilizzato.

A differenza degli algoritmi tradizionali che tengono conto di interi cammini per trovare lo stato obiettivo, gli algoritmi di ricerca locale considerano solo lo stato corrente con lo scopo di migliorarlo. Inoltre, proprio per questo motivo, gli algoritmi di ricerca tradizionali hanno una complessità sia spaziale che temporale esponenziale, mentre scegliendo un algoritmo di ricerca locale vi è la possibiltà di ridurla, nonostante l'elevato numero di prenotazioni che possono effettuare tutti gli studenti di Unisa. Infatti l'obiettivo, in queste circostanze, è quello di trovare un'allocazione quanto più vicina alla soluzione migliore e non quello di ricordare il modo in cui l'agente l'ha ottenuta.

Tra le varie opzioni messe a disposizione da questo tipo di algoritmi, la scelta più adatta, e soluzione al problema, è stata quella degli algoritmi genetici (GA) i quali costituiscono una procedura ad alto livello ispirata alla genetica. Un GA evolve una popolazione di individui (soluzioni candidate) producendo di volta in volta soluzioni sempre migliori rispetto ad una funzione obiettivo, chiamata funzione di fitness, fino a raggiungere l'ottimo o un'altra condizione di terminazione.

La creazione di nuove generazioni di individui avviene applicando degli operatori genetici, precisamente:

- **Selezione:** una copia di alcuni individui nella successiva generazione; si cercano soluzioni sempre migliori, per cui la probabilità di sopravvivenza di un individuo dipende dal valore della funzione di fitness;
- **Crossover:** l'accoppiamento di individui (parents) per crearne di nuovi (offsprings), da aggiungere alla nuova generazione; si selezionano uno o più punti (in base all'algoritmo) nella codifica degli individui e si incrociano gli individui.
- **Mutazione:** una modifica casuale di alcune parti del corredo genetico. Selezione e crossover tendono a favorire delle caratteristiche non sempre ottimali; la mutazione serve a "reintrodurre" ciò che è stato perduto ed esplorare meglio lo spazio di ricerca;

Menzionata precedentemente, la funzione di fitness è una funzione in grado di associare un valore ad ogni soluzione ed è fondamentale per la definizione di un GA. Misura il livello di adeguatezza degli individui rispetto al problema considerato e guida il processo di selezione.

3.1 Inizializzazione

L'inizializzazione indica la codifica iniziale degli individui e con quale criterio viene creata la prima generazione. I dati che vengono passati in input all'agente sono i giorni e le fasce orarie in cui lo studente è disponibile per studiare e il numero di ore che vuole dedicare allo studio. Il tutto verrà formattato utilizzando una tabella dove le colonne indicano i giorni della settimana, le righe indicano lo studente, e ogni cella contiene 4 bit che rappresentano la disponibilità di ogni studente per ognuna delle quattro fasce orarie.

Studente	Luendì	Martedì	Mercoledì	Giovedì	Venerdì	Ore Totali
A	1110	0001	1001	0000	1100	10
B	0101	0000	1111	0011	1100	14
C	1111	1111	1111	1111	0000	22

- **Rappresentazione degli individui:** L'output di ogni aula studio, che indica l'assegnazione degli studenti all'interno di una determinata aula studio", sarà una stringa in cui le virgole "," saranno il carattere separatore delle fasce orarie, mentre ogni carattere all'interno delle virgole rappresenta la codifica uno studente ovvero l'occupazione di una fascia oraria in un determinato giorno da parte di esso. Per rappresentare ogni studente verrà utilizzato un carattere di un alfabeto composto da lettere maiuscole, minuscole e dalle dieci cifre in modo tale da avere 62 caratteri a disposizione per identificare gli studenti.

  Il singolo gene sarà costituito dalla concatenazione delle stringhe descritte in precedenza. Per semplicità, verranno considerate le aule studio appartenenti ad un singolo edificio che contiene tre aule studio.

  AC,ABC,AC,BC,C,C,C,AC,ABC,BC,BC,ABC,C,C,BC,BC,AB,AB,,,

  La generazione iniziale verrà generata randomicamente.

- **Size Popolazione:** il numero di individui di ciascuna generazione; in questo caso la size è fissa e data la complessità dell'algoritmo e la diversità della popolazione, la size sarà 4. Questa scelta è dovuta anche al fatto che durante il crossover, se la size fosse dispari, un individuo della generazione rimarrebbe invariato.

- **Size Mating Pool:** il numero di individui selezionati e che partecipano alla riproduzione; sarà uguale alla size della popolazione.

3.2 Selezione

L'algoritmo di selezione indica come vengono scelti gli individui. Tra i vari algoritmi conosciuti ne sono stati analizzati alcuni rispetto al problema in questione:

- *Casuale:* dato che aumenta le chance di non convergenza ed è poco efficiente, non è stata presa in considerazione.
- *Rank:* si compie un ordinamento totale degli individui rispetto alla fitness e si assegna a ciascun individuo il rango in base alla posizione. Essendo anche questo poco efficiente, non è stato considerato.
- *Roulette Wheel:* gli individui ricevono una probabilità di selezione pari al valore della loro fitness relativa all'intera popolazione. Dato che fornisce il giusto compromesso tra efficienza e convergenza, è stato scelto questo algoritmo.

3.3 Algoritmo di Crossover

L'algoritmo di crossover indica come avviene il crossover tra individui.

- *Single point:* si seleziona un punto del patrimonio genetico dei genitori e si procede alla generazione di due figli tramite scambio di cromosomi.
- *Two point:* simile al single point, ma che va a considerare due punti di incrocio in modo da aumentare la diveristà dei geni degli individui generati.
- *K-point:* generalizzazione dei precedenti, è poco utilizzato poichè richiede la definizione di una strategia di combinazione dei k cromosomi dei genitori.

È stato deciso di effettuare una sperimentazione empirica per la scelta dell'algoritmo di crossover tra Single point e Two point, selezionando quello che fornisce maggior probabilità di restituire una soluzione quanto più vicina all'ottimo globale.

3.4 Algoritmo di Mutazione

L'algoritmo di mutazione indica come avviene la mutazione degli individui.

- *Bit Flip:* consiste nella modifica di un singolo gene binario; non si adatta al problema a causa della codifica dei geni.
- *Random Resetting:* cambio casuale di un gene ad un altro valore ammissibile; stessa situazione precedente.
- *Inversion:* si sceglie un subset di geni in modo casuale e lo si ribalta; È stata scelta la Inversion in quanto, in questo caso, permette di modificare notevolmente il gene, permettendo la formazione di geni diversi da quelli iniziali.

3.5 Stopping Condition

La stopping condition indica con quale criterio decidiamo di terminare l'evoluzione oppure proseguire.

- *Tempo di esecuzione:* l'evoluzione termina se si è superato un tempo di esecuzione T, allo stop si decide di restituire l'ultima generazione o la migliore ottenuta.
- *Costo:* se è presente una funzione di costo, al raggiungimento o al superamento di quel valore l'evoluzione termina. Dato che non è presente tale funzione, questa condition viene scartata a priori.
- *Numero di iterazioni:* si itera per un massimo di X generazioni. Questa stopping condition risulta poco affidabile poiché potrebbe esplorare uno spazio di ricerca insufficientemente grande, non considerando altre soluzioni le quali potrebbero essere migliori di quelle proposte; ciò appunto limita la ricerca in quanto potrebbe fermarsi precocemente non raggiungendo una soluzione ottima.
- *Assenza di miglioramenti:* se per Y generazioni consecutive non ci sono miglioramenti significativi, allora l'evoluzione termina poiché non avrebbe senso continuare rischio cadere nella stagnazione.
- *Ibrida:* basata su una combinazione delle precedenti.

  La decisione ricade su una stopping condition basata sul Tempo di esecuzione la quale tende ad essere sufficientemente buona.

3.6 Funzione di fitness

Per quanto emerso finora, sono state considerate più funzioni di fitness:

- *f(a)= $x_a - y_a$ con $f(a) \leq y_a$:* L'obiettivo di questa funzione di fitness è quello di assegnare ogni studente a un numero di fasce orarie tali che il quantitativo di ore di esse tenda al "numero di ore" di disponibilità dello studente. Definito a come un determinato studente, x_a indica il numero di fasce orarie assegnate ad uno studente, y_a indica il numero di fasce orarie, in cui lo studente offre disponibilità, ricevute in input. Lo scopo della funzione è quello di preferire il più grande valore $x_a \leq y_a$.
- $\sum_{i=1}^{N} g(a_i)$*:* l'obiettivo di questa funzione di fitness è quello di far coincidere quante più fasce orarie assegnate dall'agente con numero di quelle segnalate dallo studente. La seguente funzione esplica ciò

  $$g(a_i) = \begin{cases} 1 \text{ se fasciaOrariaAssegnata==fasciaOrariaStudente} \\ 0 \text{ altrimenti} \end{cases} .$$

  Lo scopo è massimizzare la funzione di fitness appena descritta.

3.7 Improvement

3.7.1 Fronte di Pareto

Quando si parla di problemi multi-obiettivo, il problema sorge in fase di valutazione delle funzioni di fitness in quanto non è possibile fornire un ordinamento totale degli individui. Entra in gioco il Fronte di Pareto, ovvero l'insieme degli individui di una popolazione non migliori tra loro ma migliori di tutti gli individui fuori dal fronte.

3.7.2 Preference Sorting

Una volta stabilito il fronte di Pareto, per decidere quale sia l'individuo migliore si utilizza il Preference Sorting, ovvero una tecnica che permette di fornire un ordinamento totale tra gli individui nel fronte di Pareto attraverso l'uso di una funzione di preferenza. Questa è diversa dalla funzione di fitness poichè il criterio che governa questa funzione deriva necessariamente dalla conscenza del problema.
Dato che, in questo caso, vi sono due funzioni di fitness, andiamo a definire quale delle due riteniamo più importante ed è stata scelta la seconda funzione di fitness. Questo perchè, se avessimo scelto la prima, significherebbe che l'algoritmo avrebbe dovuto massimizzare il numero di ore in cui lo studente studia senza però accertarsi della disponibilità di quest'ultimo. Ma questo non sarebbe corretto perchè lo studente non potrebbe usufruire della propria prenotazione.

4 Algoritmi

4.1 Prima Funzione di Fitness

L'algoritmo per la funzione di fitness "f(a)= $x_a - y_a$ con $f(a) \leq y_a$" scorre ogni gene a cui esso viene applicato e verifica quante ore l'agente ha assegnato ad uno studente rispetto alle ore che quest'ultimo ha inserito in input inizialmente

```
def fitnessQuantita(gene: str, quantita: list, studenti: list): #range [0, 1]
    output = 0
    for j in range(len(studenti)):
        i = 0
        for c in gene:
            if c == studenti[j]:
                i+=1
        output += i-int(quantita[j]) if int(quantita[j]) > i else int(quantita[j]) - i
    if output < 0:
        output = 1 / (-1 * output)
    if output == 0:
        output = 1
    return output
```

4.2 Seconda Funzione di Fitness

L'algoritmo per la seconda funzione di fitness dapprima divide il gene in tre parti, ognuna per ogni aula studio. Una volta ottenute le tre stringhe (una per ogni aula studio)e controlla quante volte uno studente è stato assegnato ad un'aula studio e in quale fascia oraria.

```
def fitnessFasceOrarie(gene: str, fasceStudenti: list, studenti: list, studyroom: int):
    output = 0
    separateGene = gene.split(',')
    studyroomList = []
    n = int(len(separateGene) / studyroom)
    for i in range(studyroom):
        studyroomList.append(separateGene[i*n: (i+1)*n])
    for k in range(len(studenti)):
        for i in range(len(studyroomList[0])):
            count = 0
            for j in range(len(studyroomList)):
                if studenti[k] in studyroomList[j][i]:
                    count += 1
            if (count == 1 and fasceStudenti[k][i] == 1):
                output += 5
            elif count == 0 and fasceStudenti[k][i] == 0:
                output += 3
            elif count == 1 and fasceStudenti[k][i] == 0:
                output += 1
            elif fasceStudenti[k][i] == 1 and count == 0:
                output += 2
            elif output - count > 0:
                output -= count
    return output
```

4.3 Inizializzazione

Il primo metodo genera randomicamente le assegnazioni degli studenti per ogni fascia oraria. Il secondo metodo compone le assegnazioni per un'aula studio. Il terzo metodo compone un gene. Il quarto metodo crea la popolazione inziale.

```
import random

def generateOutputTime(studenti: list):
    time = ""
    for i in studenti:
        value = random.uniform(0, 1)
        value = i if value > 0.5 else ''
        time += value
    return time

def generateOutputStudyroom(studenti: list, timeSlot):
    output = ""
    for i in range(timeSlot):
        output+=generateOutputTime(studenti)+","
    return output

def generateGene(studenti: list, numberStudyroom, timeSlot):
    output = ""
    for i in range(numberStudyroom):
        output+=generateOutputStudyroom(studenti, timeSlot)
    return output[0:len(output)-1]

def generatePopulation(studenti: list, sizePopulation: int, timeSlot: int, numberStudyroom: int):
    output = []
    for i in range(sizePopulation):
        output.append(generateGene(studenti, numberStudyroom, timeSlot))
    return output
```

4.4 Roulette Wheel Selection

L'algoritmo Roulette Wheel Selection, per ogni funzione di fitness assegna una probabilità di selezione ad ogni gene della popolazione e, tramite la libreria NumPy, sceglie i geni migliori da portare avanti.

```
import numpy as np
def rouletteWheelSelection(resultsFitnessFasceOrarie, resultsFitnessQuantita, population):
    newPopulation = []
    totallyFasceOrarieValue = sum(resultsFitnessFasceOrarie)
    #if totallyFasceOrarieValue > 0:
    probabilitiesFasceOrarie = [value/totallyFasceOrarieValue for value in resultsFitnessFasceOrarie]
    for i in range(len(population)):
        newPopulation.append(np.random.choice(population, p=probabilitiesFasceOrarie))
    #else:
    #    totallyQuantitaValue = sum(resultsFitnessQuantita)
    #    probabilitiesQuantita = [value/totallyQuantitaValue for value in resultsFitnessQuantita]
    #    for i in range(len(population)):
    #        newPopulation.append(np.random.choice(population, p=probabilitiesQuantita))
    return newPopulation
```

4.5 Single Point Crossover

L'algoritmo Single Point Crossover splitta un gene per ogni evenienza del carattere ",", successivamente otterremo una lista, in cui ogni cella indica una fascia oraria; infine viene divisa la lista a metà ed effettuiamo il crossover col secondo gene.

```
def singlePoint(population: list):
    part = (len(population[0].split(',')) - 1)/2
    firstCut = int(part)
    newPopulation = []
    for i in range(0, len(population), 2):
        firstGene = population[i].split(',')
        secondGene = population[i + 1].split(',')
        newFirstGene = firstGene[0:firstCut] + firstGene[firstCut:]
        newSecondGene = secondGene[0: firstCut] + secondGene[firstCut:]
        newPopulation.append(','.join(newFirstGene))
        newPopulation.append(','.join(newSecondGene))
    return newPopulation
```

4.6 Two Point Crossover

L'algoritmo Two Point Crossover splitta un un gene per ogni evenienza del carattere ",", successivamente otterremo una lista, in cui ogni cella indica una fascia oraria; infine viene divisa la lista in due punti ed effettuiamo il crossover col secondo gene

```
def twoPoint(population: list):
    part = (len(population[0].split(',')) - 1)/3
    firstCut = int(part)
    secondCut = int(2 * part)
    newPopulation = []
    for i in range(0, len(population), 2):
        firstGene = population[i].split(',')
        secondGene = population[i + 1].split(',')
        newFirstGene = firstGene[0:firstCut] + secondGene[firstCut: secondCut] + firstGene[secondCut:]
        newSecondGene = secondGene[0: firstCut] + firstGene[firstCut: secondCut] + secondGene[secondCut:]
        newPopulation.append(','.join(newFirstGene))
        newPopulation.append(','.join(newSecondGene))
    return newPopulation
```

4.7 Inversion Mutation

L'algoritmo Inversion Mutation genera randomicamente due indici del gene creandone un subset, lo inverte e lo rimette nella sua posizione all'interno del gene.

```
from random import randint

def inversion(population: list):
    newPopulation = []
    for i in range(len(population)):
        newGene = population[i].split(',')
        while True:
            firstValue = randint(0, len(newGene))
            secondValue = randint(0, len(newGene))
            if firstValue < secondValue:
                break
        part = newGene[firstValue: secondValue]
        part.reverse()
        newPopulation.append(','.join(newGene[0:firstValue] + part + newGene[secondValue:]))
    return newPopulation
```

5 Conclusioni

- La sperimentazione Empirica riguardante il crossover ha dimostrato che l'algoritmo single point riesce a fornire soluzioni migliori rispetto al two point. Infatti, le funzioni di fitness delle popolazioni andavano sempre a peggiorare.
- L'algoritmo fornisce soluzioni di gran lunga migliori con l'aumento del tempo di esecuzione. Infatti, dal test risulta che l'esecuzione, passando da un tempo pari a 2 minuti ad un tempo pari a 10 minuti, fornisce risultati migliori nel secondo caso di test.
- L'algoritmo funziona meglio eliminando la prima funzione di fitness; ciò vuol dire che il preference sorting non verrà utilizzato. Inoltre, la funzione di fitness eliminata verrà usata come funzione di valutazione dell'output finale.