Università degli Studi di
Napoli Federico II

Dipartimento di Ingegneria Elettrica e delle
Tecnologie dell'Informazione

Corso di Intelligenza Artificiale

Elaborato Esercitativo

Work Projects

Milo Saverio
Pommella Michele
Previtera Gabriele

Prof. Neri Filippo

Anno 2016-2017

Work Project 2

2.1 Esercizio 1: Evolutionary Optimization

2.1.1 Esperimento 1: Ottimizzazione

Attraverso gli algoritmi forniti siamo stati in grado di sperimentare le differenti tecniche di *ricerca locale* su diversi problemi. Esse si basano sul principio dei miglioramenti successivi: si cerca in primo luogo una soluzione ed in seguito ci si concentra sull'ottimizzazione. Va considerato che spesso si riesce a determinare solo un massimo locale, ottenendo risultati più o meno soddisfacenti a seconda dell'applicazione.

Random Searching

Questo algoritmo prevede semplicemente la determinazione casuale di un insieme di soluzioni e la comparazione dei relativi costi per l'identificazione di quella a costo minimo.
Abbiamo effettuato differenti esperimenti variando il numero di soluzioni estratte casualmente: ad un suo eccessivo aumento non si ottengono necessariamente risultati nettamente migliori da giustificare l'evidente incremento della complessità computazionale in termini temporali; d'altro canto, un numero esiguo di soluzioni produce risultati dalla qualità incostante.

Hill Climbing

L'`Hill Climbing` inizialmente determina una soluzione ed i suoi "vicini": se uno di essi presenta un costo minore della soluzione corrente, diventerà la nuova soluzione alla prossima iterazione, attuando, in tal modo, il processo dei miglioramenti successivi; se la soluzione corrente ha costo minore dei suoi vicini, termina la serie di miglioramenti iterativi poiché si è in presenza di un massimo locale (minimo locale rispetto al costo).
Vari esperimenti ci hanno condotto a risultati decisamente migliori del `Random Searching`, che sottolineano la maggior efficienza dell'`Hill`

`Climbing`. La variabilità nella qualità della soluzione, al seguito di svariate esecuzioni, è dipendente dai massimi locali del problema specifico.

Simulation annealing

Versione rivisitata dell' L'`Hill Climbing`, questo metodo di ottimizzazione è ispirato dal riscaldamento termico di una lega. Viene comparata la soluzione corrente ad una determinata casualmente (nel nostro caso tramite un cambiamento di quella corrente): l'algoritmo tende a spostarsi sempre verso una soluzione successiva migliore, solo con una certa probabilità verso quelle peggiori. Questa probabilità dipende dalla qualità della soluzione successiva: mosse pessime avranno minore probabilità. Viene, inoltre, introdotto un concetto di "temperatura": essa determina la probabilità di spostarsi verso soluzioni non ottime. Inizialmente questa probabilità sarà alta (temperatura elevata), in seguito essa tenderà a calare (raffreddamento), consentendo, in ultima battuta, la scelta della soluzione ottimizzata. Tale probabilità è data da:

$$e^{\frac{-(costosuccessivo+costoattuale)}{temperatura}}$$

Osserviamo che con l'abbassamento della temperatura, ad ogni iterazione dell' algoritmo, tale probabilità decresce. Le ragioni alla base di questa logica risiedono nella maggiore possibilità di trovare un massimo globale, evitando che ci si possa assestare su un punto di massimo locale come una *spalla* (shoulder).
Dalle prove sperimentali abbiamo constatato risultati migliori o uguali all'`Hill Climbing`, appurando la maggiore capacità di ottenere soluzioni con costo inferiore.

Genetic Algorithms

Ulteriore algoritmo che tenta di superare i limiti dell'`Hill Climbing` è quello genetico. Esso è ispirato dalla natura, proponendo un'evoluzione delle soluzioni tipicamente darwiniana. L'algoritmo prende le mosse selezionando un insieme casuale di soluzioni, detto *popolazione*. Gli esemplari della popolazione vengono poi valutati ed ordinati attraverso una *funzione di fitness* (rappresentata nel nostro caso dal costo minimo).
Nell'implementazione utilizzata, si procede alla creazione della successiva *generazione* a partire da un prestabilito numero di migliori soluzioni. Attraverso il processo di *elitarismo* le migliori soluzioni sopravviveranno nella nuova generazione; la restante popolazione sarà costituita attraverso la *mutazione* (cambiamento casuale ad una soluzione) ed il *crossover* (combinazione di due soluzioni) dell'elite.

Vari esperimenti di hanno portato a riscontrare risultati simili al `Simulated Annealing`, se non migliori in taluni casi, che presentano, però, una minore variabilità con misure ripetute. Riusciamo, in definitiva, ad ottenere una buona ottimizzazione, di qualità superiore all'`Hill Climbing` ed al `Random Searching`.

2.1.2 Esperimento 2: Caso d'uso

L' ottimizzazione sulla quale ci vogliamo concentrare è quella di una funzione $F(x)$, così definita (se $x < 5.2, F(x) = 10$; se $5.2 <= x <= 20, F(x) = x^2$; se $x > 20, F(x) = x - 1$) dove la x può assumere valori tra $[-100, 100]$, la ricerca di tale valore verrà effettuato grazie ad un algoritmo genetico.Prima di tutto però dobbiamo definire la funzione di costo,così da discriminare quando la nostra soluzione tende a migliorare oppure peggiora:

```
1 def costmax(sol):
2         if sol[0]>100 or sol[0]<-100:
3                 cost=0
4         else:
5                 cost=F(sol)
6 return cost
```

molto semplicemente se il valore di cui si calcola il costo non è accettabile, perché il passaggio tra una generazione ad un' altra si è avuto un gene non voluto,cioè non accettabile per la soluzione (valore al di fuori dell' intervallo $[-100, 100]$) gli si da un costo 0 così per la creazione della prossima generazione non verrà preso in considerazione.
L' algoritmo prima di tutto genera una popolazione iniziale scegliendo a caso valori presi nello spazio delle soluzioni, ne calcola i vari score e sceglie una elite di geni(quelli che in questo caso massimizzano lo score), dopodiché per un numero di generazioni volute possiamo far mutare un elemento o fare il crossover di 2 in base ad una probabilità di mutazione da noi fornita.Il problema è che la rappresentazione numerica in float non permette di definire un' operazione di mutazione o crossover semplice, per tale ragione i numeri vengono convertiti nella loro forma binaria(in questo caso a 32 bit), definendo quindi la mutazione il cambiamento di un bit da 1 a 0 o viceversa, invece il crossover è eseguito scegliendo un numero di bit casuale di un gene che devono essere scambiati con un numero restanti di bit di un altro, affinché venga raggiunto un numero di 32 bit.Il crossover è una un' operazione di ricombinazione genetica in cui parte del materiale genetico di un gene si ricombina con un altro, così facendo ad ogni cambio generazionale si introducono geni combinazione dei cromosomi di geni padri migliori nella

generazione precedente, procedendo così ad una operazione di selezione naturale. La mutazione invece è un fattore casuale che permette di introdurre nuovi tratti nei geni, così che le nuove generazioni introducano caratteristiche distintive non presente nei genitori, permettendo alla specie di trovare diverse strade per sopravvivere. Come si può vedere, il valore che ottimizza la F(x) è dato da x=20, il risultato si riesce a trovare anche partendo da una popolazione iniziale ristretta, solo 50 elementi, e per un numero di generazioni discreto 100, la soluzione non solo viene trovata, ma viene confermata più volte consta del fatto che anche se si provano nuove approcci al massimo tramite la mutazione e la ricombinazione, il gene che massimizza la nostra fitness è stato trovato.

2.2 Esercizio 2: MDS: Visually Exploring US Senator Similarity

2.2.1 Clustering

Il Clustering o analisi di raggruppamento è una tecnica di intelligenza artificiale volta alla selezione e raggruppamento di elementi omogenei in un insieme di dati. Tutte le tecniche di clustering si basano sul concetto di distanza tra due elementi,che ci permette di definire il concetto di somiglianza tra gli elementi, infatti l'appartenenza o meno ad un insieme dipende da quanto l'elemento preso in esame è distante dall'insieme. La bontà delle analisi ottenute dagli algoritmi di clustering dipende dalla metrica, metriche diverse porteranno quasi certamente a cluster differenti.

2.2.2 MDS: Multidimensional scaling

Il MultiDimensional Scaling è una tecnica di analisi statistica usata per mostrare graficamente le differenze o somiglianze tra elementi di un insieme. È una generalizzazione del concetto di ordinamento: partendo da una matrice quadrata, contenente la "somiglianza" di ogni elemento di riga con ogni elemento di colonna, l'algoritmo di scaling multidimensionale assegna a ogni elemento una posizione in uno spazio N-dimensionale, con N stabilito a priori.In pratica questa tecnica parte con un sistema con tante dimensioni quanti gli elementi del sistema, e riduce le dimensioni fino a un certo numero N. Nel fare questo quindi c'è un'inevitabile perdita di informazione.

2.2.3 Riflessioni

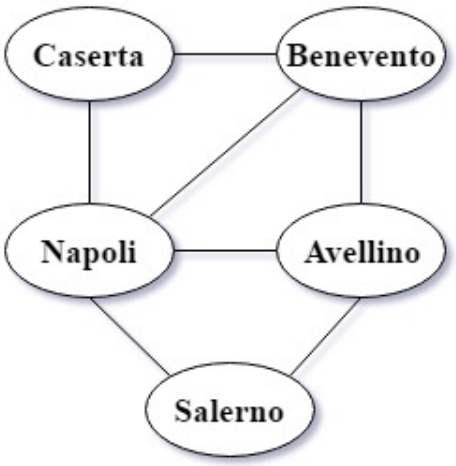
Nel codice che ci e' stato fornito un abbiamo a disposizione due esempi, il primo si basa su il giudizio da parte di quattro venditori riguardo a 6 prodotti, e il secondo esempio si basa sull'analisi dei voti di 10 congressi americani. Lo scopo è quello di raggruppare i venditori e i senatori in base alle loro preferenze. Nel codice sono forniti anche i dataset, alcune funzioni di utilità che permettono di convertire i dataset in dataset più compatti. Inoltre la metrica utilizzata per la distanza è quella euclidea.
Dal Multidimensional scaling del primo esempio giungiamo alla seguente conclusione :" i recensori A e D hanno valori di metrica abbastanza vicini tra loro mentre per i recensori B e C no. Analizzando i dati che abbiamo a disposizione non rimaniamo sorpresi di questi risultati,l'algoritmo classifica A e D come simili infatti hanno espresso quattro pareri concordanti su sei, mentre B e C hanno espresso solo due concordanti e non è un buon risultato per classificarli come simili. Ovviamente per considerazioni più approfondite abbiamo bisogno di un numero maggiore di dati."
Per il secondo esempio costituito da un dataset reale e molto ampio giungere a delle conclusioni è stato più difficile. Una prima immagine che ci fornisce il risultato dell'elaborazione consiste nello schieramento dei Repubblicani a destra e dei Democratici a sisnistra e inoltre vediamo che la maggior parte dei senatori è schierata in uno dei due partiti. Nelle immagini successive abbiamo i nomi dei senatori e che appartengono ad un determinato partito ed effettuando delle ricerche su internet vediamo che effettivamente la classificazione ottenuta è buona, per esempio Obama viene correttamente classificato come Repubblicano mentre Jeffords, il cui ha un passato da repubblicano viene riportato a destra ma ha un colore diverso in quanto ora è un indipendente. L'ultima serie di immagini che l'algoritmo ci fornisce, possiamo vedere che le scelte di voto dei partiti sono consistenti.Cioè se un partito sceglie di elargire quel voto, tutti gli iscritti al partito si attengono, in generale, a quella scelta.

2.3 Esercizio 3: Constraint Satisfaction Problems

Quello di un'agente intelligente che deve colorare le province della regione Campania evitando che le confinanti abbiano lo stesso colore è un tipico esempio di problema a soddisfacimento di vincoli. Individuiamo, quindi:

Variabili: Caserta, Benevento, Napoli, Avellino, Salerno
Vincoli: province confinanti devono avere colori diversi

Possiamo, dunque, ricavare il grafo dei vincoli, i cui nodi sono le variabili ed i cui archi connettono coppie di variabili che partecipano ad un vincolo.



Analizziamo il caso in cui l'agente disponga di solo 2 colori.

$$\text{Dominio} = \{\text{Colore1}, \ \text{Colore2}\}$$

Applicando il `Backtracking Search` si perviene presto alla conclusione dell'impossibilità di risolvere il problema vincolato. Ad esempio, applicando le euristiche per migliorare l'efficienza della ricerca, assegniamo a Napoli (`Degree Heuristic`: variabile con più vincoli sulle rimanenti variabili) il valore Colore1; se procediamo assegnando Colore2 a Caserta (o qualsiasi altra variabile dato che ora presentano tutte un solo valore ammissibile), si ottiene un fallimento, poiché tramite il `Forward Checking` si osserva che Benevento non può assumere nessun valore ammissibile. Questa risultato è comune anche agli altri path dell'albero di ricerca e tutto ciò è intuitivamente comprensibile poiché con soli 2 colori non siamo in grado di soddisfare i loop di 3 nodi presenti nel grafo dei vincoli.
Analizziamo il caso in cui l'agente disponga di 3 colori.

$$\text{Dominio} = \{\text{Colore1}, \ \text{Colore2}, \ \text{Colore3}\}$$

Proviamo un approccio differente, iniziamo ad assegnare colori alle province e poi scegliamo quella con il numero minimo di scelte possibili `Minimum remaining values`.Partiamo da Napoli e le assegniamo il Colore1,notiamo che questa scelta ha ridotto tutte le scelte rimanenti per le altre province a due, allora coloriamo Caserta con il Colore2 ora Benevento è la nostra prossima scelte perché le può essere assegnata solo un colore invece per le restanti due province ancora due, quindi Benevento avrà Colore3, Avellino di conseguenza Colore2 e Salerno Colore 3, quindi possiamo affermare che è possibile soddisfare i vincoli.

Anche il caso con quattro colori sicuramente permetterà di soddisfare i vincoli, perché si è aumentato il numero di valori ammissibili rimando invariate le informazioni iniziali, comunque verifichiamo che sia così.

$$Dominio = \{Colore1,\ Colore2,\ Colore3,\ Colore4\}$$

Cambiamo euristica è scegliamo la `Least constraining value`, in questo caso andiamo a scegliere come variabile successiva quella che minimizza il numero di vincoli ammissibile per le altre. La prima che andiamo a colorare è la Campania assegnandole il Colore1,perché così facendo riduciamo tutti i colori possibili per le altre province, avendo la Campania il maggior numero di vincoli, Avellino sarà la nostra prossima scelta permettendo di diminuire il numero di valori per Salerno e Benevento, assegnandole il Colore2, poi possiamo a Benevento che riduce i colori di Caserta a 2, colorandola con il Colore3, dopodiché possiamo dare a Caserta il Colore2 e Salerno il Colore3. Come si può vedere l' aggiunta di un nuovo elemento nel Dominio,non alterando i vincoli, permette comunque la risoluzione del problema, evitando anche di utilizzare il nuovo valore, quindi possiamo affermare che con un determinato grafo dei vincoli, una volta trovata la soluzione con un determinato numero di valori,questa si trova anche se il Dominio diventa più grande, con la differenza di "elasticizzare" i vincoli, nel senso che se con tre elementi nel Dominio il numero di soluzioni possibili, sicuramente sarà molto più limitato rispetto a quello con quattro elementi, avendo molte più possibilità di scelta, potendo dire che le soluzioni che si trovano con il Dominio "allargato" racchiudono quelle del Dominio più compatto.

2.4 Esercizio 4

Un approccio possibile per trovare la soluzione con l' ausilio di algoritmi genetici potrebbe essere questo: Definire come funzione di costo il numero di conflitti, province confinanti con lo stesso colore, il nostro obiettivo è quindi quello di minimizzare questo valore.La popolazione iniziale sarà costituita da delle cartine colorate in modo casuale,dopodiché definiamo una probabilità di mutazione,se non siamo in quest' ultimo caso allora avverrà l' operazione di crossover che consiste nel prendere in modo casuale dall' elite due elementi e scegliere probabilisticamente, quanti cromosomi(province) di uno unire con i restanti dell' altro,altrimenti per la mutazione invece basterà scegliere un elemento dalla cartina e cambiarli colore a scelta tra gli altri tre, escludendo ovviamente quello già assegnato.