UNIVERSITÀ DEGLI STUDI DI PARMA
FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

Corso di Intelligenza Artificiale

Anno Accademico: 2011-2012

RELAZIONE SUI PROGETTI 1, 2, 4

Studenti:
ALESSANDRO COSTALUNGA
DAVIDE VALERIANI

Esercitazione 1

Scopo di questa esercitazione è comprendere a fondo i metodi di apprendimento automatico utilizzando gli *alberi di decisione*, modelli predittivi in cui ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e una foglia il valore predetto per la variabile obiettivo a partire da i valori delle altre proprietà, che nell'albero è rappresentato del cammino (*path*) dal nodo radice (*root*) al nodo foglia.

Normalmente un albero di decisione viene costruito utilizzando tecniche di apprendimento a partire dall'insieme dei dati iniziali (*data set*), il quale può essere diviso in due sottoinsiemi: il *training set* sulla base del quale si crea la struttura dell'albero e il *test set* che viene utilizzato per testare l'accuratezza del modello predittivo così creato.

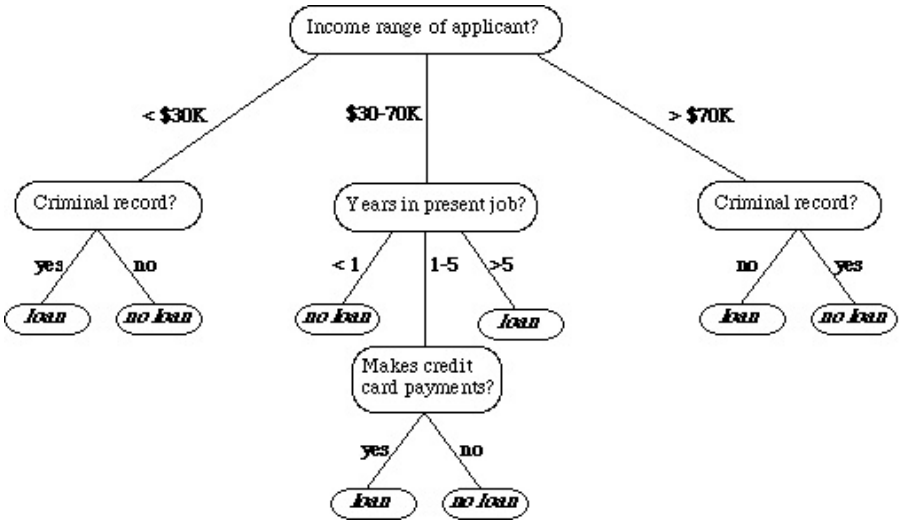

Figura 1: Esempio di albero di decisione sulla possibilità di concedere o meno un prestito.

In questa esercitazione è stato utilizzato Weka 3.6.7 come programma per la realizzazione di alberi decisionali, software opensource interamente scritto in Java.

Esercizio 1

Il primo esercizio ha lo scopo di analizzare il data set *circletrain.arff* composto dalle coordinate (x,y) dei punti di un quadrato e di un cerchio e, successivamente, classificare questi punti come apppartenenti a un quadrato o a un cerchio.

Il data set considerato è composto da 100 istanze, ovvero 50 punti appartenenti a un cerchio e 50 appartenenti a un quadrato. Di seguito è riportato un frammento del file *circle_train.arff*.

```
@relation circle
@attribute x real
@attribute y real
@attribute class {q,c}
```

```
@data
 0.8703995,0.4926300,q
-1.0335922,-1.0629654,q
-1.2409592,-1.1921310,q
 0.5479050,0.3514099,c
...
```

Per la costruzione dell'albero di decisione, si è utilizzato il classificatore J48 di Weka, con numero minimo di istanze per foglia pari a 2 e selezionando l'opzione *random split* al 66% che consente di utilizzare 66 istanze come training set e le rimanenti 34 come test set.

Di seguito è mostrato l'albero generato a partire dalle istanze di training set, sia in versione testuale, fornita da Weka, che grafica (fig. 2). Come si può notare, l'albero ha in totale 9 nodi, di cui 5 foglie (in grassetto), e utilizza entrambe le coordinate dei punti per la classificazione.

```
J48 pruned tree
------------------

y <= 0.843849
|   y <= -0.980568: q (15.0)
|   y > -0.980568
|   |   x <= 0.918846
|   |   |   x <= -0.937157: q (4.0)
|   |   |   x > -0.937157: c (55.0/3.0)
|   |   x > 0.918846: q (9.0/1.0)
y > 0.843849: q (17.0)

Number of Leaves  :  5

Size of the tree :  9
```

La classificazione delle istanze appartenenti al test set generato prendendo il 34% delle istanze del data set a disposizione, ha portato a classificare correttamente 30 istanze su 34, pari a circa l'88% dei dati del test set.

```
=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances          30       (88.2353 %)
Incorrectly Classified Instances         4       (11.7647 %)
Kappa statistic                          0.7655
Mean absolute error                      0.1544
Root mean squared error                  0.3561
Relative absolute error                 30.6701 %
Root relative squared error             70.2629 %
Total Number of Instances               34
```

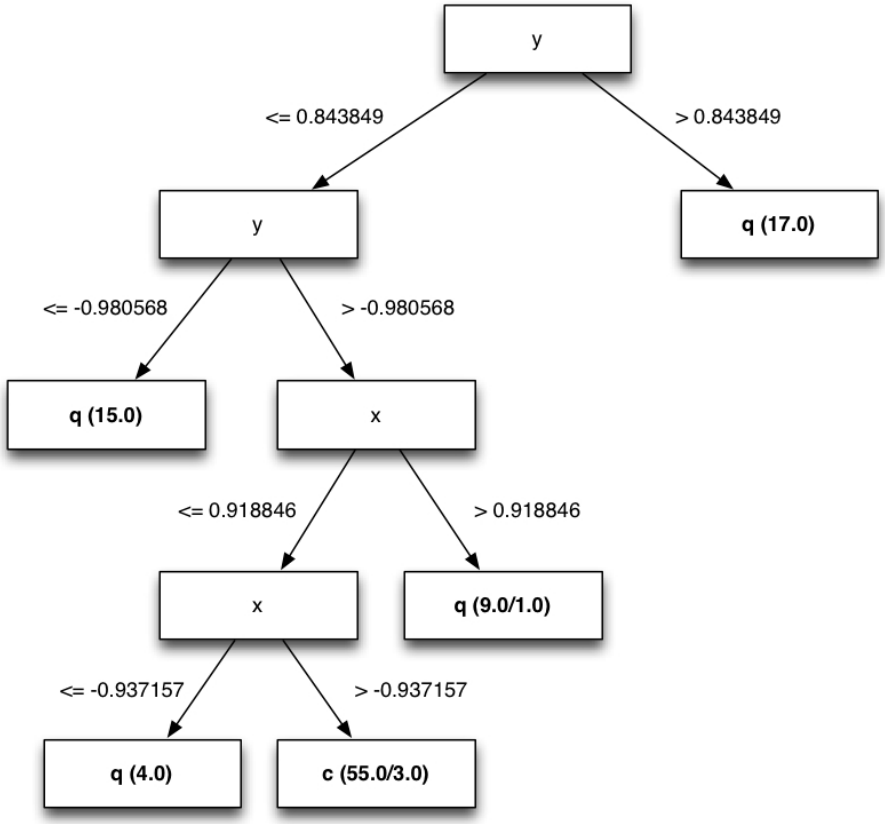

Figura 2: Albero di decisione generato

Tramite la matrice di confusione fornita come output dal classificatore, è possibile osservare che il 75% delle classificazioni errate sono dovute alla classificazione di un cerchio come quadrato.

```
  a  b   <-- classified as
 15  3 |  a = q
  1 15 |  b = c
```

Si è proceduto quindi alla modifica per 5 volte del seme per la selezione random degli esempi, inizialmente posto uguale a 1, e si è ripetuta la classificazione del test set, pari al 34% del data set. I risultati sono stati i seguenti:

```
*** SEED = 2 ***
Correctly Classified Instances          28       (82.3529 %)
Incorrectly Classified Instances         6       (17.6471 %)
Kappa statistic                          0.6421
Mean absolute error                      0.1828
Root mean squared error                  0.416
Relative absolute error                 36.6815 %
Root relative squared error             83.4442 %
Total Number of Instances               34
```

```
=== Confusion Matrix ===
  a  b   <-- classified as
 12  3 |  a = q
  3 16 |  b = c

*** SEED = 3 ***
Correctly Classified Instances          31       (91.1765 %)
Incorrectly Classified Instances         3       (8.8235 %)
Kappa statistic                          0.8247
Mean absolute error                      0.1255
Root mean squared error                  0.2763
Relative absolute error                 24.935  %
Root relative squared error             54.5092 %
Total Number of Instances               34
=== Confusion Matrix ===
  a  b   <-- classified as
 15  3 |  a = q
  0 16 |  b = c

*** SEED = 4 ***
Correctly Classified Instances          30       (88.2353 %)
Incorrectly Classified Instances         4       (11.7647 %)
Kappa statistic                          0.7647
Mean absolute error                      0.1176
Root mean squared error                  0.343
Relative absolute error                 23.5294 %
Root relative squared error             68.3339 %
Total Number of Instances               34
=== Confusion Matrix ===
  a  b   <-- classified as
 15  2 |  a = q
  2 15 |  b = c

*** SEED = 5 ***
Correctly Classified Instances          31       (91.1765 %)
Incorrectly Classified Instances         3       (8.8235 %)
Kappa statistic                          0.8235
Mean absolute error                      0.1005
Root mean squared error                  0.2949
Relative absolute error                 20.1678 %
Root relative squared error             59.0895 %
Total Number of Instances               34
=== Confusion Matrix ===
  a  b   <-- classified as
 15  1 |  a = q
  2 16 |  b = c
```

```
*** SEED = 6 ***
Correctly Classified Instances          29       (85.2941 %)
Incorrectly Classified Instances         5       (14.7059 %)
Kappa statistic                          0.7059
Mean absolute error                      0.1667
Root mean squared error                  0.371
Relative absolute error                 33.3333 %
Root relative squared error             73.9067 %
Total Number of Instances               34
=== Confusion Matrix ===
  a  b   <-- classified as
 15  2 |  a = q
  3 14 |  b = c
```

Per ciascuna classificazione è stata calcolata la sensibilità S pari a:

$$S = \frac{veri\_positivi}{veri\_positivi + falsi\_negativi} \tag{1}$$

e la specificità S_p, pari a:

$$S_p = \frac{veri\_negativi}{veri\_negativi + falsi\_positivi} \tag{2}$$

I risultati sono riassunti nella tabella 1.

Seme	**Sensibilità**	**Specificità**
2	0.8	0.8421
3	1.0	0.8421
4	0.8824	0.8824
5	0.8824	0.9412
6	0.8333	0.875
Media	0.8796	0.8766

Tabella 1: Valori di sensibilità e specificità nei casi presi in esame

Si è quindi utilizzato come test set il file *circletest.arff*, contenente 100 istanze, di cui 40 quadrati e 60 cerchi. I risultati dell'operazione di classificazione sono mostrati di seguito.

```
Correctly Classified Instances          88       (88      %)
Incorrectly Classified Instances        12       (12      %)
Kappa statistic                          0.7521
Mean absolute error                      0.1528
Root mean squared error                  0.3362
Relative absolute error                 30.9336 %
Root relative squared error             67.9343 %
```

```
Total Number of Instances               100
=== Confusion Matrix ===

  a  b   <-- classified as
 35  5 |  a = q
  7 53 |  b = c
```

Come è possibile vedere dai risultati, in questo caso il valore di sensibilità $S = 0.8333$ risulta essere leggermente inferiore alla media calcolata in precedenza, mentre il valore di specificità $S_p = 0.9138$ risulta superiore alla media. Queste lievi differenze fanno comunque dire che il classificatore ha buone performance.

Esercizio 1b

Questo esercizio consiste nel creare un data set contenente la descrizione di un gruppo di persone caratterizzate dai seguenti attributi:

- *Altezza* (valore numerico)
- *Peso* (valore numerico)
- *Colore dei capelli* (nero, castano, biondo, rosso, bianco, calvo)
- *Lunghezza dei capelli* (assenti, corti, normali, lunghi, molto lunghi)
- *Baffi o barba* (si, no)
- *Colore degli occhi* (nero, castano, verde, azzurro)
- *Sesso* (M, F)

Tra questi attributi, occorre utilizzare *Sesso* quale classe.

Il training set creato consiste di 30 istanze, 15 maschi e 15 femmine, ed è riportato di seguito:

```
@relation persone
@attribute altezza real
@attribute peso real
@attribute colore_capelli {nero,castano,biondo,rosso,bianco,
calvo}
@attribute lunghezza_capelli {assenti,corti,normali,lunghi,
molto_lunghi}
@attribute baffi {si,no}
@attribute barba {si,no}
@attribute colore_occhi {nero,castano,verde,azzurri}
@attribute class {M,F}
```

```
@data

170,68,nero,corti,no,si,nero,M
165,87,calvo,assenti,no,no,verde,M
180,78,biondo,normali,no,no,azzurri,M
167,56,castano,lunghi,no,no,azzurri,F
130,45,biondo,molto_lunghi,no,no,verde,F
190,100,castano,corti,si,si,nero,M
173,82,bianco,normali,no,no,verde,M
165,54,bianco,lunghi,no,no,azzurri,F
205,105,nero,corti,no,si,nero,M
176,67,rosso,lunghi,no,no,castano,F
169,69,nero,corti,no,no,verde,F
172,90,biondo,corti,no,no,nero,F
167,50,nero,lunghi,no,no,azzurri,F
178,72,nero,normali,no,si,castano,M
182,81,castano,molto_lunghi,si,no,castano,M
191,78,castano,lunghi,no,no,nero,F
192,80,calvo,assenti,si,si,azzurri,M
177,77,bianco,normali,no,no,castano,F
172,88,bianco,corti,si,no,nero,M
170,62,rosso,corti,no,no,verde,F
177,78,castano,corti,no,no,verde,F
189,89,nero,lunghi,no,no,nero,M
171,80,nero,corti,no,no,nero,M
172,76,castano,lunghi,no,no,castano,F
173,81,biondo,normali,no,si,verde,M
174,75,rosso,normali,no,no,azzurri,F
175,82,bianco,molto_lunghi,no,no,nero,M
176,74,rosso,molto_lunghi,no,no,castano,F
177,83,calvo,assenti,no,no,verde,F
178,73,nero,normali,no,no,azzurri,M
```

Ad occhio, è possibile osservare che gli attributi maggiormente significativi per la classificazione sono *barba* (assente in tutte le femmine), *baffi* (assenti in tutte le femmine) e il *peso* (più basso, di norma, nelle femmine).

Utilizzando il classificatore J48 del software Weka è stato creato l'albero di decisione, mostrato di seguito in versione testuale e in fig. 3 in versione grafica, il cui obiettivo è stabilire se un individuo sia maschio o femmina sulla base degli altri attributi indicati.

```
J48 pruned tree
------------------
barba = si: M (6.0)
barba = no
|   peso <= 78: F (15.0/2.0)
|   peso > 78: M (9.0/2.0)
```

```
Number of Leaves  :  3

Size of the tree :  5
```

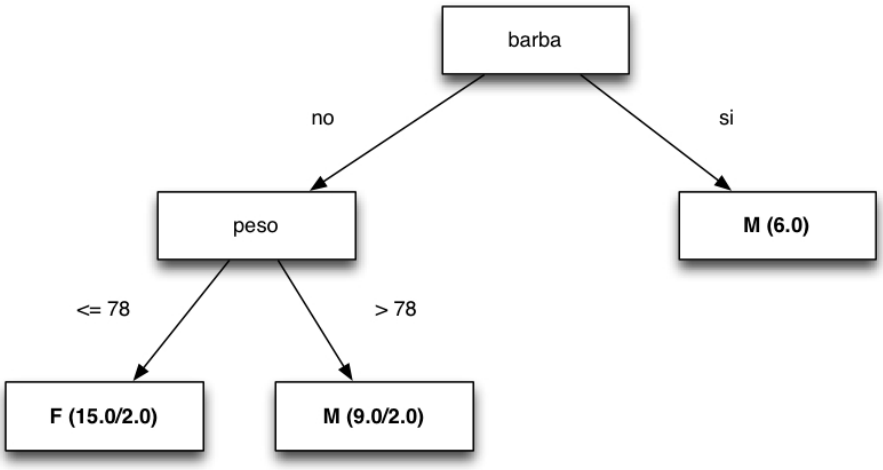

Figura 3: Albero di decisione generato

Come è possibile notare, l'albero utilizza due dei tre attributi individuati ad occhio per la classificazione (*barba* e *peso*). Il numero di nodi dell'albero ammonta a 5, di cui 3 foglie.

Al fine di valutare le prestazioni dell'albero di decisione così creato, si è utilizzato come test set il 34% delle istanze del data set, per un totale di 10 istanze.

I risultati sono mostrati di seguito:

```
=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances           6       (60 %)
Incorrectly Classified Instances         4       (40 %)
Kappa statistic                          0.2857
Mean absolute error                      0.4
Root mean squared error                  0.4776
Relative absolute error                 78.5714 %
Root relative squared error             93.4621 %
Total Number of Instances               10

=== Confusion Matrix ===

 a b   <-- classified as
 2 4 | a = M
 0 4 | b = F
```

Come è possibile notare, l'albero di decisione riconosce correttamente il sesso della persona nel 60% dei casi, una percentuale relativamente bassa a causa della scarsa cardinalità del training set, composto da soli 20 istanze. Tuttavia, come si può notare dalla matrice di confusione, tutti gli errori commessi dal classificatore riguardano maschi classificati come femmine.

Al fine di visualizzare l'influenza sui risultati del parametro *m* del classificatore J48, ovvero il numero minimo di istanze per foglia, si è proceduto a modificare tale valore dal valore 2 usato in precedenza ai valori 5, 10 e 15. I risultati sono mostrati di seguito.

Nel caso di $m = 5$ la struttura dell'albero non cambia (fig. 4), mentre peggiorano i risultati, che si attestano al 40% di classificazioni corrette.

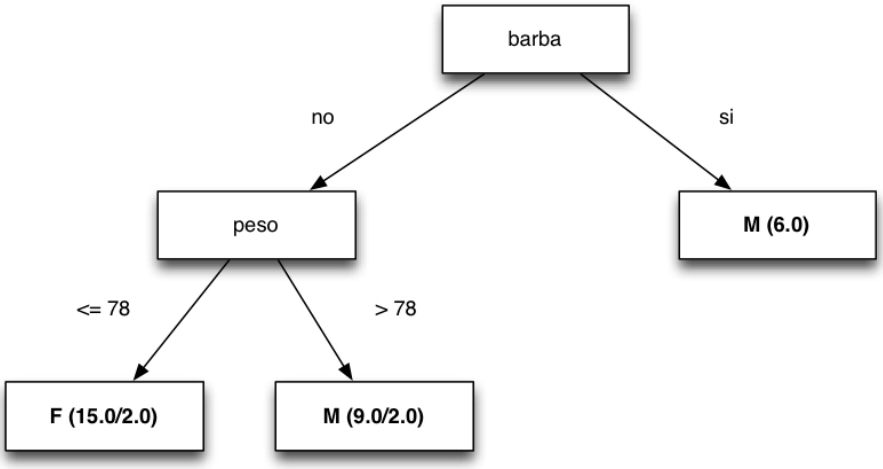

Figura 4: Albero di decisione generato con m=5

```
=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances           4               (40 %)
Incorrectly Classified Instances         6               (60 %)
Kappa statistic                         -0.1538
Mean absolute error                      0.5417
Root mean squared error                  0.5933
Relative absolute error                106.3988 %
Root relative squared error            116.1014 %
Total Number of Instances               10

=== Confusion Matrix ===

 a b   <-- classified as
 2 4 | a = M
 2 2 | b = F
```

Risultati sensibilimente differenti si ottengono nel caso di *m = 10*. In questo caso, infatti, la struttura dell'albero viene modificata (fig. 5): il numero di nodi si riduce a 3 e la classificazione viene effettuata valutando solamente l'attributo del peso, abbandonando quindi barba e baffi identificati nel riconoscimento ad occhio come attributi specifici dei maschi.

Tuttavia, quasi a sorpresa, i risultati sul test set risultano superiori, arrivando ad identificare correttamente l'80% delle istanze. Questo risulta dovuto ad un test set limitato e non a una correttezza generale del classificatore.

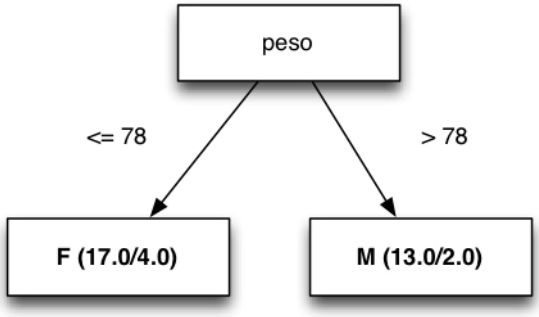

Figura 5: Albero di decisione generato con m=10

```
=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances           8       (80 %)
Incorrectly Classified Instances         2       (20 %)
Kappa statistic                          0.5455
Mean absolute error                      0.42
Root mean squared error                  0.4313
Relative absolute error                 82.5    %
Root relative squared error             84.3928 %
Total Number of Instances               10

=== Confusion Matrix ===

 a b   <-- classified as
 6 0 | a = M
 2 2 | b = F
```

Infine, nel caso di *m = 15*, l'albero viene nuovamente modificando (fig. 6), utilizzando come unico attributo di classificazione l'altezza e abbandonando, quindi, tutti gli attributi identificati ad occhio.

Come atteso, i risultati sul test set degradano a un 40% di classificazioni corrette come nel caso di *m = 5*, sebbene la struttura dell'albero sia diversa. Si nota, infatti, come tutte le istanze del test set siano identificate come femmine, mentre solamente il 40% lo sono veramente.

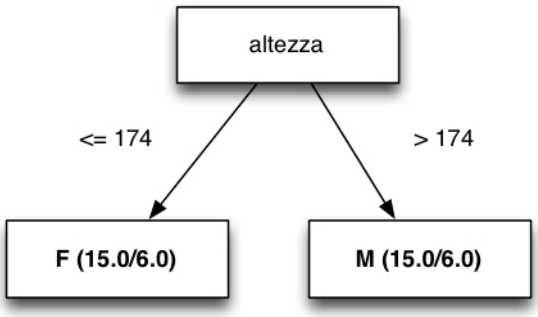

Figura 6: Albero di decisione generato con m=15

```
=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances           4        (40 %)
Incorrectly Classified Instances         6        (60 %)
Kappa statistic                          0
Mean absolute error                      0.51
Root mean squared error                  0.5123
Relative absolute error                100.1786 %
Root relative squared error            100.2568 %
Total Number of Instances               10

=== Confusion Matrix ===

 a b   <-- classified as
 0 6 | a = M
 0 4 | b = F
```

Concludendo, i risultati migliori sono stati ottenuti con *m = 10*, arrivando a classificare correttamente l'80% delle istanze del test set. Tuttavia, si nota come l'ampliamento del data set sia condizione necessaria per un miglioramento delle prestazioni del classificatore, che per *m = 5* e per *m = 15* risulta effettivamente piuttosto scarso.

Esercizio 2

Questo esercizio consiste nel realizzare il *gioco della formica*, composto da una griglia di NxN celle. In fase di inizializzazione, N celle vengono riempite con *cibo*, assegnandogli valore +1, mentre tutte le altre celle sono vuote con valore 0. Inoltre, la formica viene posizionata su una qualunque cella vuota.

La formica può muoversi in quattro direzioni: Nord (N), Sud (S), Ovest (O) e Est (E). Ogni volta che la formica si muove, essa acquista il punteggio della cella in cui si è spostata e decrementa di 1 il valore della cella stessa, lasciando così traccia del proprio passaggio. Se la formica esce dalla griglia *muore*, ovvero viene penalizzata di 3N punti,

e la partita termina.
Il campo visivo della formica un quadrato di lato 2m+1 centrato nella cella in cui si trova. Gli attributi saranno pertanto le celle adiacenti e i valori di tali attributi sarà il contenuto delle celle corrispondenti.

Nel caso in cui l'intorno della formica sia costituito da valori tutti uguali, la formica si muove casualmente, evitando di tornare sui propri passi.

Scopo del gioco è massimizzare il punteggio accumulato dalla formica in 2N mosse a partire da una posizione prefissata.

Il gioco è stato realizzato in linguaggio Java utilizzando l'IDE Eclipse Indigo.

In questo esercizio sono state giocate alcune partite in modalità manuale, memorizzando i dati relativi alle posizioni viste dalla formica ed alla direzione presa, e si sono costruiti, mediante il classificatore J48 del software Weka (usando i parametri di default), gli alberi di decisione relativi. Infine, si sono confrontati i risultati di cinque partite giocate in modalità automatica basate su sei alberi decisionali creati giocando rispettivamente 1, 5 o 10 partite e utilizzando due possibili valori di m (1 o 2). La dimensione N della griglia è stata presa pari a 10.

Di seguito sono riportati gli alberi decisionali generati.

1 partita - m=1

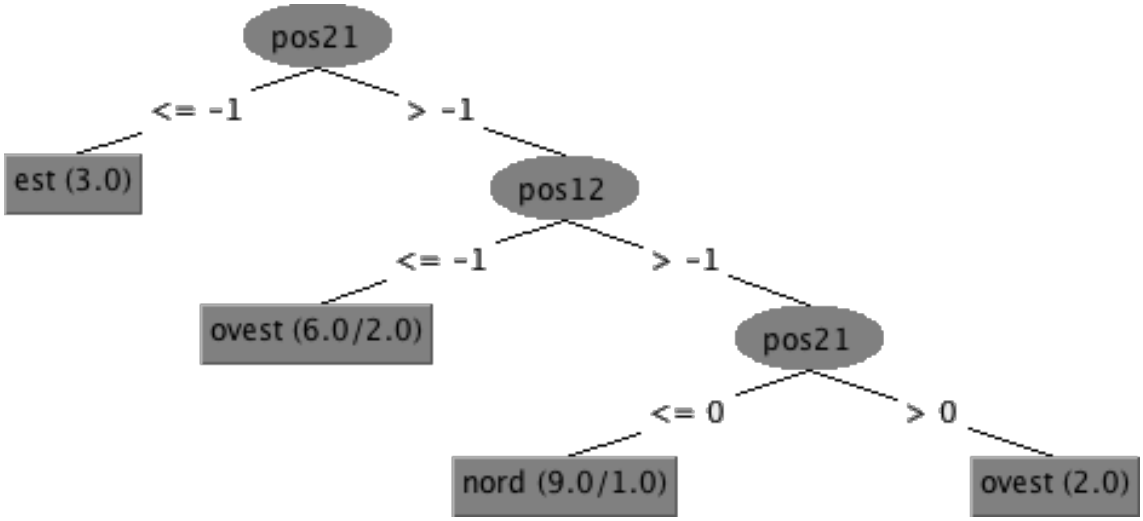

Figura 7: Albero di decisione generato con m=1 giocando 1 partita

In questo caso, l'albero generato (fig. 7) ha una dimensione pari a 7 e un numero di foglie (decisioni) pari a 4. Utilizzando il training set come test set, l'albero classifica correttamente 17 casi su 20, pari all'85%. La matrice di confusione è mostrata di seguito.

```
a b c d   <-- classified as
8 0 0 0 | a = nord
0 0 0 2 | b = sud
1 0 3 0 | c = est
0 0 0 6 | d = ovest
```

5 partite - m=1

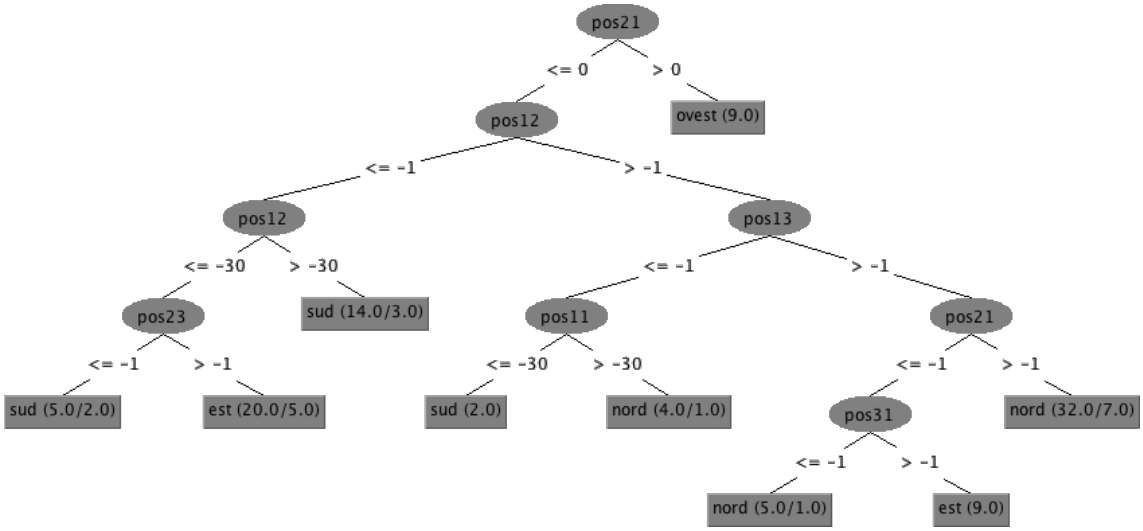

Figura 8: Albero di decisione generato con m=1 giocando 5 partite

In questo secondo caso, l'albero generato (fig. 8) ha una dimensione pari a 17 e un numero di foglie pari a 9, mentre le classificazioni corrette sono l'81%. La matrice di confusione è mostrata di seguito.

```
 a  b  c  d   <-- classified as
32  0  0  0 |  a = nord
 3 16  2  0 |  b = sud
 5  2 24  0 |  c = est
 1  3  3  9 |  d = ovest
```

10 partite - m=1

L'albero generato (fig. 9) ha una dimensione pari a 37 e un numero di foglie pari a 19, mentre le classificazioni corrette sono l'81.407%. La matrice di confusione è mostrata di seguito.

```
 a  b  c  d   <-- classified as
50  0  2  2 |  a = nord
 5 35  3  3 |  b = sud
 8  2 44  4 |  c = est
 5  3  0 33 |  d = ovest
```

Da queste prime prove, è possibile notare come l'aumentare del numero di partite, sebbene comporti un aumento della dimensione dell'albero, non significhi necessariamente un aumento delle prestazioni del classificatore. Infatti, sebbene i test siano stati effettuati utilizzando lo stesso training set, l'aumento del numero di nodi dell'albero comporta un leggero degrado dei risultati del classificatore.

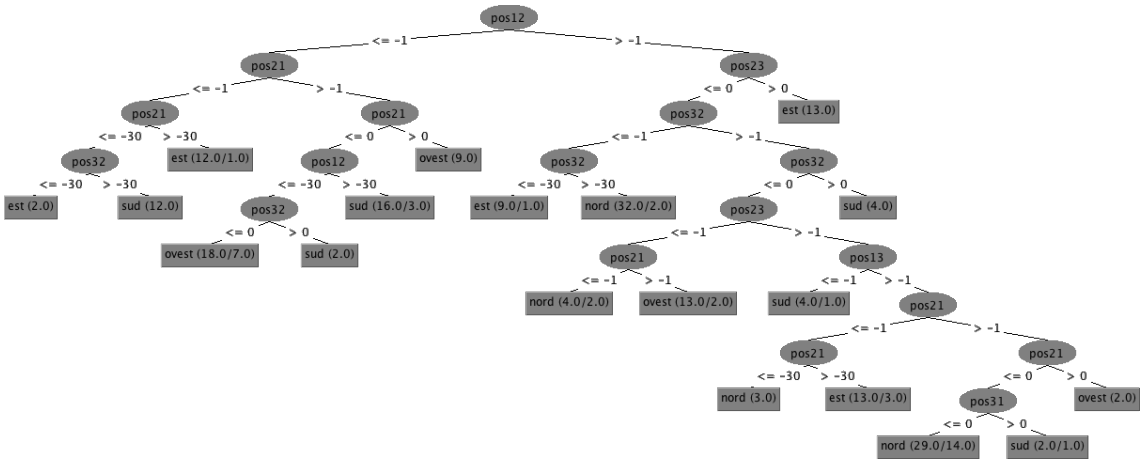

Figura 9: Albero di decisione generato con m=1 giocando 10 partite

Nonostante questo, i risultati di classificazione si mantengono al di sopra dell'80%.

Di seguito sono riportati gli alberi costruiti considerando $m = 2$. La formica può ora vedere fino a due caselle in avanti e, pertanto, tarare le proprie scelte con maggior criterio.

1 partita - m=2

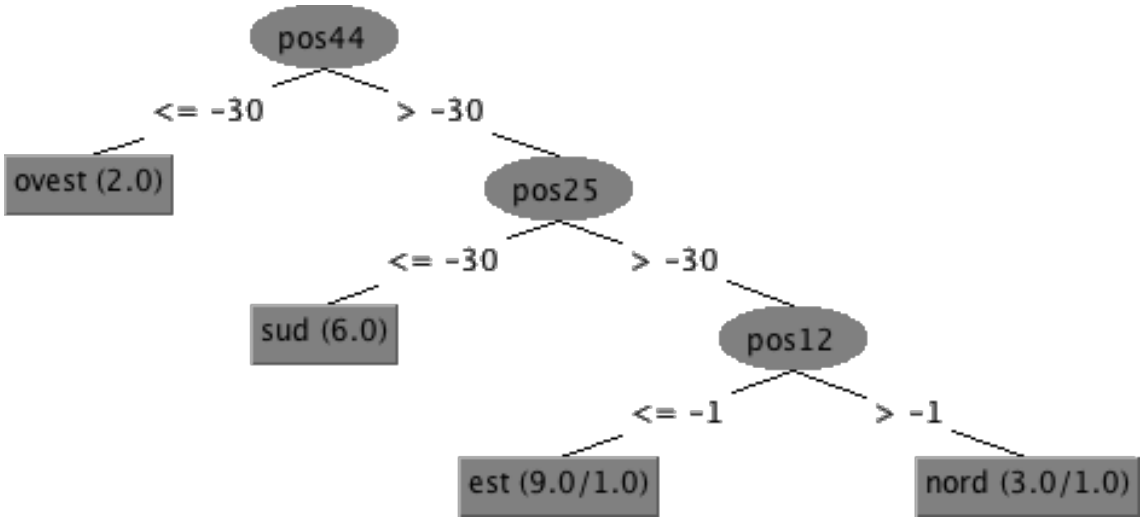

Figura 10: Albero di decisione generato con m=2 giocando 1 partita

In questo caso, l'albero generato (fig. 10) ha una dimensione pari a 7 e un numero di foglie pari a 4. Utilizzando il training set come test set, l'albero classifica correttamente 18 casi su 20, pari al 90%. La matrice di confusione è mostrata di seguito.

```
a b c d   <-- classified as
2 0 0 0 | a = nord
1 6 1 0 | b = sud
```

```
 0  0  8  0 |  c = est
 0  0  0  2 |  d = ovest
```

5 partite - m=2

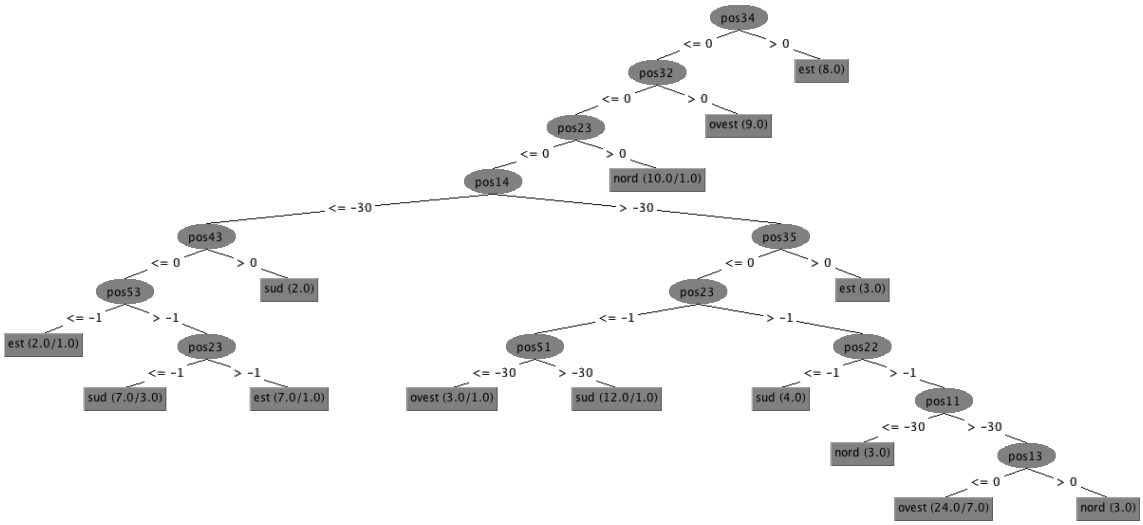

Figura 11: Albero di decisione generato con m=2 giocando 5 partite

In questo secondo caso, l'albero generato (fig. 11) ha una dimensione pari a 27 e un numero di foglie pari a 14, mentre le classificazioni corrette sono il 84.5361%. La matrice di confusione è mostrata di seguito.

```
 a  b  c  d   <-- classified as
15  0  0  3 |  a = nord
 1 21  1  4 |  b = sud
 0  2 18  1 |  c = est
 0  2  1 28 |  d = ovest
```

10 partite - m=2

L'albero generato (fig. 12) ha una dimensione pari a 55 e un numero di foglie (decisioni) pari a 28, mentre le classificazioni corrette sono l'89.1753%. La matrice di confusione è mostrata di seguito.

```
 a  b  c  d   <-- classified as
46  0  3  2 |  a = nord
 2 36  2  2 |  b = sud
 2  0 45  2 |  c = est
 3  2  1 46 |  d = ovest
```

Di seguito vengono valutati i risultati dell'applicazione di questi alberi di decisione al gioco della formica, cioè facendo muovere la formica in modo automatico.

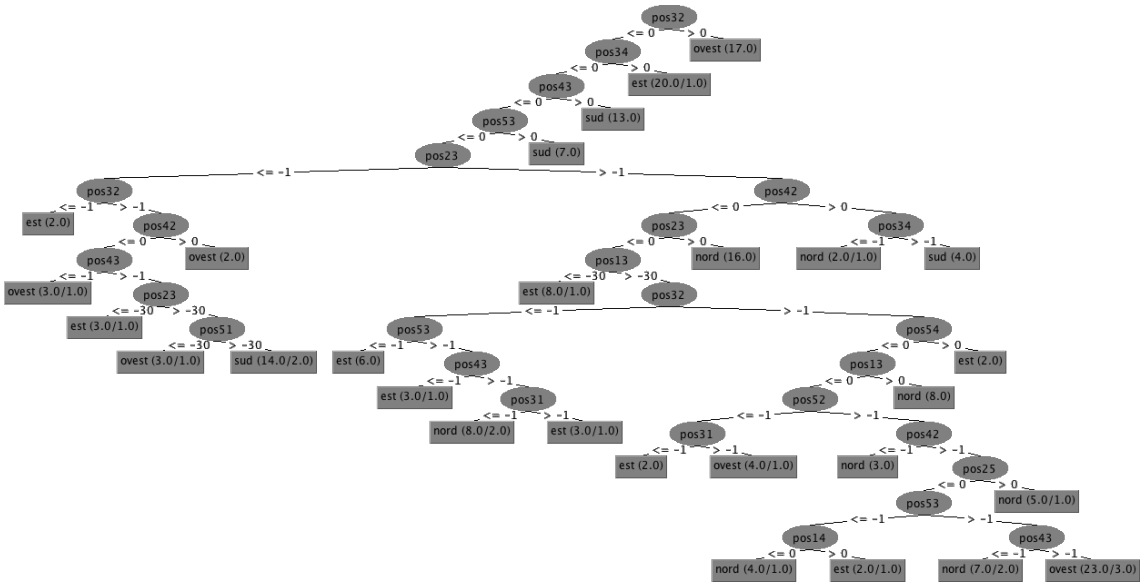

Figura 12: Albero di decisione generato con m=2 giocando 10 partite

Risultati - 1 partita

Utilizzando i dati di un'unica partita per il training, ovvero 20 sole istanze, si nota come i risultati siano piuttosto scarsi (tab. 2).

Nel caso di m=1, la formica muore nel caso arrivi al bordi superiore della griglia, in quanto l'albero le indica di andare sempre a *est* e, quindi, appena arrivata al bordo destro della griglia, muore.

Nel caso di m=2, invece, arrivata all'angolo inferiore sinistro della griglia, la formica sceglie di andare a ovest e, pertanto, muore. A parte questo caso limite, negli altri casi la formica rimane in vita, anche se non raggiunge neanche la metà delle caselle contenenti cibo.

Partita	***m=1***		***m=2***	
	Stato formica	**Punteggio**	**Stato formica**	**Punteggio**
1	Morta	-30	Viva	3
2	Viva	-5	Morta	-30
3	Morta	-33	Viva	2
4	Viva	0	Viva	0
5	Morta	-31	Viva	1

Tabella 2: Risultati delle partite giocate in automatico con addestramento di 1 partita

Risultati - 5 partite

Aumentando le partite giocate, si può notare come le prestazioni del classificatore aumentino nel caso di m=1 e degradino nel caso di m=2 (tab. 3). Questo è dovuto al fatto che

quando la formica si trova lungo il lato sinistro o nell'angolo in alto a destra, tende ad andare ulteriormente verso il muro e, quindi, a morire.

Partita	***m=1***		***m=2***	
	Stato formica	**Punteggio**	**Stato formica**	**Punteggio**
1	Viva	3	Morta	-27
2	Morta	-29	Morta	-26
3	Viva	4	Viva	5
4	Viva	3	Morta	-30
5	Viva	5	Morta	-27

Tabella 3: Risultati delle partite giocate in automatico con addestramento di 5 partite

Risultati - 10 partite

Incrementando a 10 il numero di partite giocate nella fase di training, si nota (tab. 4) come i risultati siano in linea con quelli ottenuti al passo precedente, sebbene nel caso di $m = 2$ siano leggermente migliorati, sia dal punto di vista del numero ridotto di morti della formica, sia dal punto di vista del punteggio raggiunto.

Partita	***m=1***		***m=2***	
	Stato formica	**Punteggio**	**Stato formica**	**Punteggio**
1	Viva	2	Viva	4
2	Viva	0	Viva	7
3	Morta	-30	Morta	-29
4	Viva	0	Morta	-29
5	Viva	4	Viva	5

Tabella 4: Risultati delle partite giocate in automatico con addestramento di 10 partite

Dai risultati è possibile notare come non è scontato che i risultati migliorino con l'aggiunta di dati di training, ovvero giocando più partite in modalità manuale. La performance migliore si è avuta addestrando l'albero con 5 partite e con $m = 1$.

Esercitazione 2

Lo scopo di questa esercitazione è lo studio delle *reti neurali*, modelli matematici che rappresentano l'interconnessione tra elementi definiti *neuroni artificiali*, ossia costrutti matematici che in qualche misura imitano le proprietà dei neuroni viventi e che, in queste reti, rappresentano i nodi di elaborazione. I collegamenti tra neuroni vengono detti *sinapsi*.

Un neurone artificiale (fig. 13) è costituito da due stadi in cascata:

- *sommatore lineare* che somma tutti i collegamenti in ingresso al neurone, dato da $net = \sum_j w_j i_j$, dove w_j è il peso associato all'input j;
- *funzione di attivazione* non lineare a soglia $o = f(net)$ che attiva l'uscita del neurone solo se viene superata la soglia da parte di tutti gli ingressi.

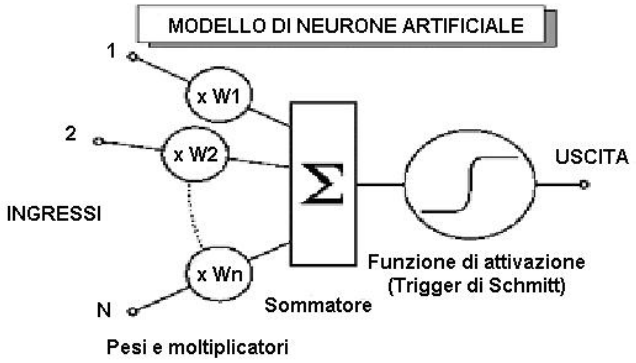

Figura 13: Modello di neurone artificiale

Una rete neurale (fig. 14) è un'architettura a più strati:

- *strato di ingresso*, riceve i segnali provenienti dall'ambiente esterno;
- uno o più *strati nascosti*, non accessibili dall'esterno, consentono la comunicazione all'interno della rete;
- *strato di uscita*, fornisce i risultati dell'elaborazione da parte dei neuroni.

Ad ogni connessione è associato un peso, utilizzato nel sommatore che costituisce il primo stadio del neurone che riceve dati attraverso la connessione.

Il processo iterativo di apprendimento di una rete neurale è il seguente:

- i pesi della rete vengono modificati sulla base delle *prestazioni* della rete su un insieme di esempi;
- si minimizza una *funzione obiettivo* che rappresenta di quanto il comportamento della rete si discosta da quello desiderato.

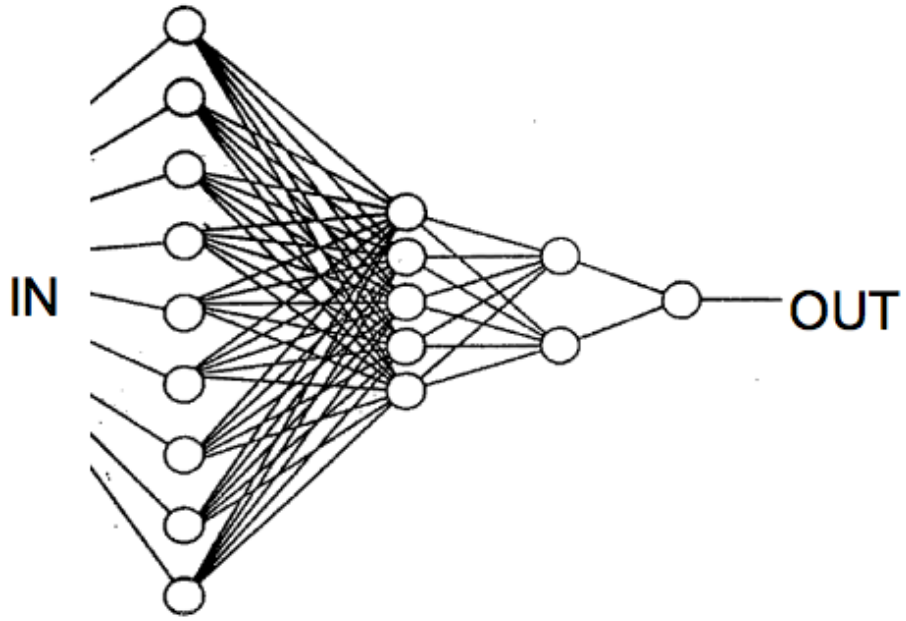

Figura 14: Modello di rete neurale

Esercizio 1

Questo esercizio consiste nell'utilizzare il file *xor.arff* fornito da Weka per addestrare una rete neurale in grado di risolvere l'operazione XOR (or esclusivo). Il file, riportato sotto, contiene la tabella di verità dell'operazione XOR.

```
@relation xor
@attribute x REAL
@attribute y REAL
@attribute out REAL
@data
0 0 0
0 1 1
1 0 1
1 1 0
```

Come classificatore è stato usato il *MultilayerPerceptron*, utilizzando il training set anche come test set e attivando l'opzione *Output Predictions*. Sono invece stati disattivate tutte le opzioni di preprocessing dei dati.

In caso di rete monostrato (fig. 15), ovvero senza strati nascosti ($H = 0$), è possibile notare come la rete non riesca a classificare correttamente l'operazione XOR, commettendo un errore relativo pari al 100%.

Di seguito sono mostrati i risultati di predizione ottenuti, con il valore attuale, quello predetto e l'errore commesso, e i risultati del test di valutazione sul training set, in cui è possibile notare l'errore del 100%.

```
=== Predictions on training set ===
inst#,     actual, predicted, error
     1        0          0.6        0.6
     2        1          0.709     -0.291
     3        1          0.836     -0.164
     4        0          0.945      0.945

=== Evaluation on training set ===
Correlation coefficient                  0
Mean absolute error                      0.5
Root mean squared error                  0.5842
Relative absolute error                100      %
Root relative squared error            116.8456 %
Total Number of Instances                4
```

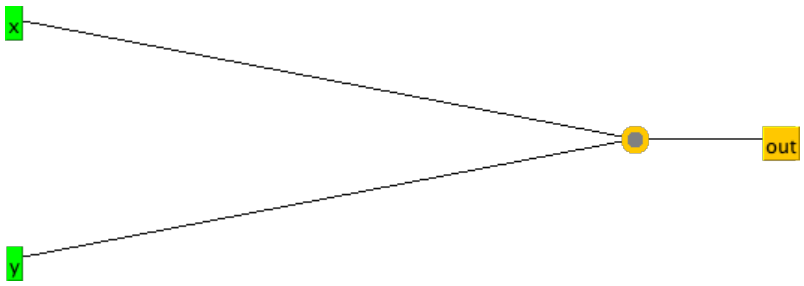

Figura 15: Rete neurale generata con N=500, H=0, L=0.3

Visti gli scarsi risultati della rete, sono stati aggiunti due strati nascosti, modificando il parametro H del classificatore a 2 e rieseguendo l'addestramento della rete. Il risultato è la rete di fig. 16, che mostra in rosso gli strati nascosti aggiunti.

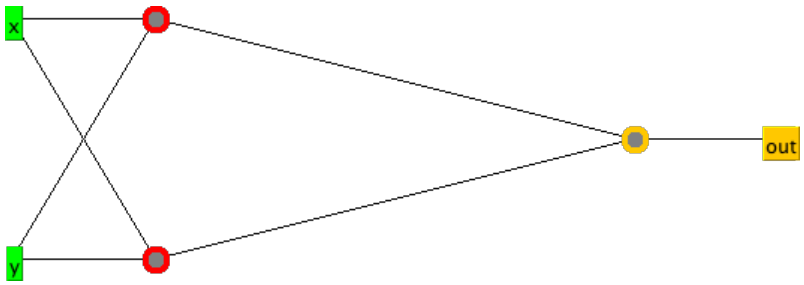

Figura 16: Rete neurale generata con N=500, H=2, L=0.3

In questo caso i risultati, mostrati di seguito, sono nettamente migliori: l'errore viene ridotto allo 0.0167% e tutti i dati del training set vengono classificati correttamente, a fronte però di una rete più complessa.

Inoltre, aumentando di poco il numero delle epoche *N*, ovvero il tempo di addestramento (da 500 a 600), l'errore relativo converge a 0.

```
=== Predictions on training set ===
inst#,     actual, predicted, error
     1        0          0          0
     2        1          1          0
     3        1          1          0
     4        0          0          0

=== Evaluation on training set ===
Correlation coefficient                  1
Mean absolute error                      0.0001
Root mean squared error                  0.0001
Relative absolute error                  0.0167 %
Root relative squared error              0.0198 %
Total Number of Instances                4
```

Per osservare il comportamento della rete neurale in corrispondenza dello spazio $[0,1]^2$, è stato creato il file *xortest.arff* contenente 40401 istanze che rappresentano le coordinate dei punti dello spazio (x,y) con un passo di 0.005 in entrambe le direzioni. Tali punti vengono classificati come 0 se la differenza tra le loro coordinate è inferiore a 0.5, altrimenti vengono classificati come 1.

Di seguito sono mostrati i risultati della classificazione, in cui sono state usate le istanze di *xor.arff* come training set e quelle del file *xortest.arff* come test set.

Nel primo caso, è stato impostato il classificatore con training time N=500, numero di strati nascosti H=0 e learning rate L=0.3. Come è possibile vedere dai risultati seguenti, il classificatore non ha buoni risultati: l'errore relativo è del 127.1786 %.

```
=== Evaluation on test set ===

Correlation coefficient                 -0.001
Mean absolute error                      0.6359
Root mean squared error                  0.6826
Relative absolute error                127.1786 %
Root relative squared error            136.5251 %
Total Number of Instances            40401
```

Introducendo due strati nascosti (H=2), i risultati sono sensibilmente migliori: l'errore relativo si riduce al 54.0935 %.

```
=== Evaluation on test set ===

Correlation coefficient                  0.6104
Mean absolute error                      0.2705
Root mean squared error                  0.3595
Relative absolute error                 54.0935 %
Root relative squared error             71.8916 %
Total Number of Instances            40401
```

Aumentando il learning rate L=0.4, l'errore relativo scende ulteriormente, attestandosi al 49.0293 %.

```
=== Evaluation on test set ===

Correlation coefficient                  0.6628
Mean absolute error                      0.2451
Root mean squared error                  0.3301
Relative absolute error                 49.0293 %
Root relative squared error             66.0223 %
Total Number of Instances            40401
```

Infine, aumentando nuovamente il learning rate L=0.5 e il momento M=0.5, l'errore relativo scende al 35.3468%.

```
=== Evaluation on test set ===

Correlation coefficient                  0.8143
Mean absolute error                      0.1817
Root mean squared error                  0.2623
Relative absolute error                 36.3468 %
Root relative squared error             52.4541 %
Total Number of Instances            40401
```

Esercizio 1b

Scopo di questo esercizio è classificare i punti all'interno di uno spazio, distinguendoli tra punti appartenenti a:

- Cerchio di raggio 1 e centro 0,0
- Quadrato di lato 2.5 e centro 0,0 (con esclusione della regione del cerchio)

Per la classificazione, sono stati utilizzati i file *circletrain.arff* (per il training) e *circletest.arff* (per il test), costituiti ciascuno da cento campioni estratti casualmente all'interno di un quadrato di lato 2.5, al cui interno è posto un cerchio di raggio 1. Ogni istanza del data set è descritta da tre attributi: coordinate (x,y) del punto all'interno dello spazio considerato e un valore numerico che indica se il punto considerato cade all'interno o all'esterno del cerchio.

Le simulazioni effettuate sono state fatte modificando il numero di strati nascosti (H), il numero di neuroni per strato e il learning rate (L).

Di seguito sono riportate le simulazioni effettuate per un numero di epoche pari a 500 e 2000.

I risultati più significativi (caso migliore e caso peggiore, evidenziati in ognitabella) sono stati utilizzati per classificare un insieme denso di punti campionati all'interno del quadrato 2.5, costituito da 63001 punti ottenuti spazzolando lo spazio 2D $[-1.25, 1.25]$ con un incremento di 0.01 in ogni direzione.

Strati nascosti	Neuroni per strato	Learning rate	Classificazioni corrette (%)
0	0	0.1	63
0	0	0.3	61
0	0	0.5	62
0	0	0.7	61
1	1	0.1	65
1	1	0.3	67
1	1	0.5	68
1	1	0.7	68
1	2	0.1	74
1	2	0.3	81
1	2	0.5	82
1	2	0.7	81
1	3	0.1	93
1	3	0.3	92
1	3	0.5	88
1	3	0.7	93
1	4	0.1	97
1	4	0.3	98
1	4	0.5	94
1	4	0.7	94
1	5	0.1	99
1	5	0.3	96
1	5	0.5	95
1	5	0.7	95
1	6	0.1	97
1	6	0.3	97
1	6	0.5	95
1	6	0.7	95
2	1	0.1	53
2	1	0.3	68
2	1	0.5	68
2	1	0.7	68
2	2	0.1	67
2	2	0.3	75
2	2	0.5	73
2	2	0.7	72
2	3	0.1	75
2	3	0.3	90
2	3	0.5	80
2	3	0.7	78
2	4	0.1	71
2	4	0.3	99
2	4	0.5	95
2	4	0.7	96

Strati nascosti	Neuroni per strato	Learning rate	Classificazioni corrette (%)
2	5	0.1	74
2	5	0.3	99
2	5	0.5	96
2	5	0.7	97
2	6	0.1	76
2	6	0.3	94
2	6	0.5	99
2	6	0.7	98

Tabella 5: Risultati ottenuti con numero epoche N=500

Strati nascosti	Neuroni per strato	Learning rate	Classificazioni corrette (%)
0	0	0.1	63
0	0	0.3	61
0	0	0.5	62
0	0	0.7	61
1	1	0.1	67
1	1	0.3	68
1	1	0.5	67
1	1	0.7	66
1	2	0.1	75
1	2	0.3	81
1	2	0.5	79
1	2	0.7	79
1	3	0.1	92
1	3	0.3	91
1	3	0.5	91
1	3	0.7	93
1	4	0.1	98
1	4	0.3	99
1	4	0.5	92
1	4	0.7	96
1	5	0.1	99
1	5	0.3	97
1	5	0.5	96
1	5	0.7	97
1	6	0.1	95
1	6	0.3	96
1	6	0.5	96
1	6	0.7	96

Strati nascosti	**Neuroni per strato**	**Learning rate**	**Classificazioni corrette (%)**
2	1	0.1	68
2	1	0.3	64
2	1	0.5	64
2	1	0.7	64
2	2	0.1	76
2	2	0.3	75
2	2	0.5	72
2	2	0.7	73
2	3	0.1	93
2	3	0.3	89
2	3	0.5	85
2	3	0.7	78
2	4	0.1	97
2	4	0.3	96
2	4	0.5	96
2	4	0.7	95
2	5	0.1	94
2	5	0.3	97
2	5	0.5	86
2	5	0.7	93
2	6	0.1	96
2	6	0.3	94
2	6	0.5	98
2	6	0.7	96

Tabella 6: Risultati ottenuti con numero epoche N=2000

Il primo caso, composto da due strati nascosti con un neurone ciascuno, un learning rate pari a 0.1 e 500 epoche, è caratterizzato da risultati scarsi: 10 % di istanze correttamente classificate e un errore relativo pari al 100.1554 %. Nonostante ciò, visto l'alto numero di campioni, il cerchio è facilmente identificabile.

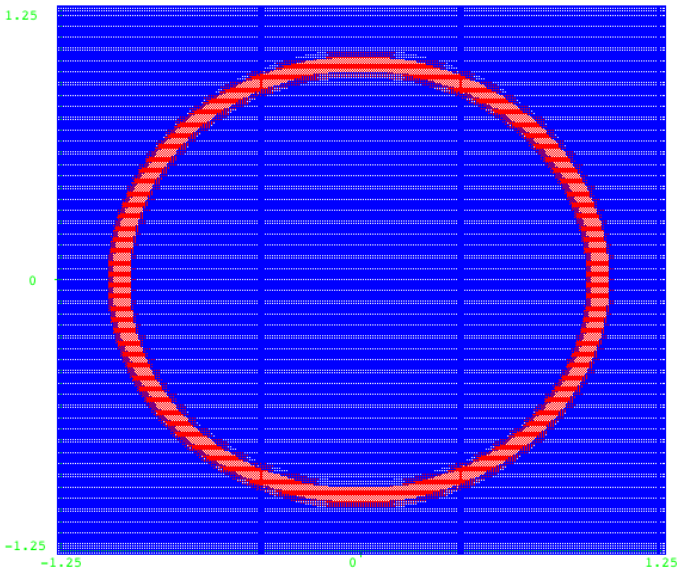

Figura 17: Visualizzazione grafica con N=500, H=1,1 e L=0.1

Il secondo caso, composto da due strati nascosti con quattro neuroni ciascuno, un learning rate pari a 0.3 e 500 epoche, è caratterizzato da risultati migliori: 50.2897 % di istanze correttamente classificate e un errore relativo pari al 89.3726 %.

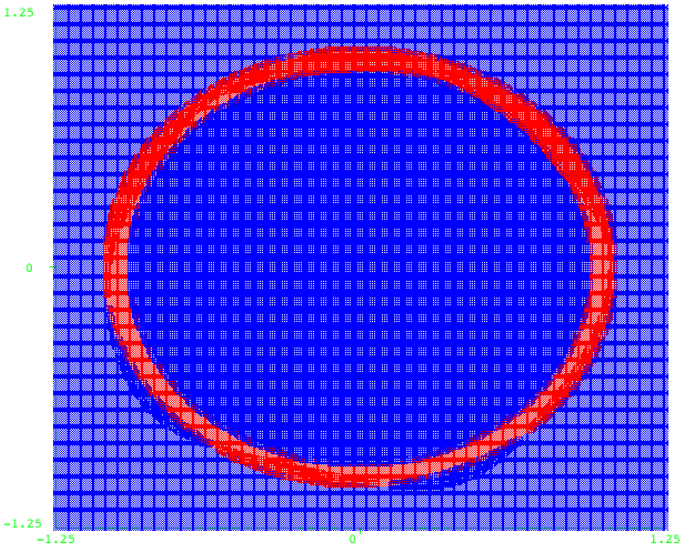

Figura 18: Visualizzazione grafica con N=500, H=4,4 e L=0.3

Il terzo caso, composto da nessuno strato nascosto, un learning rate pari a 0.3 e 5000 epoche, è caratterizzato da risultati scarsi: 25.944 % di istanze correttamente classificate e un errore relativo pari al 105.9118 %.

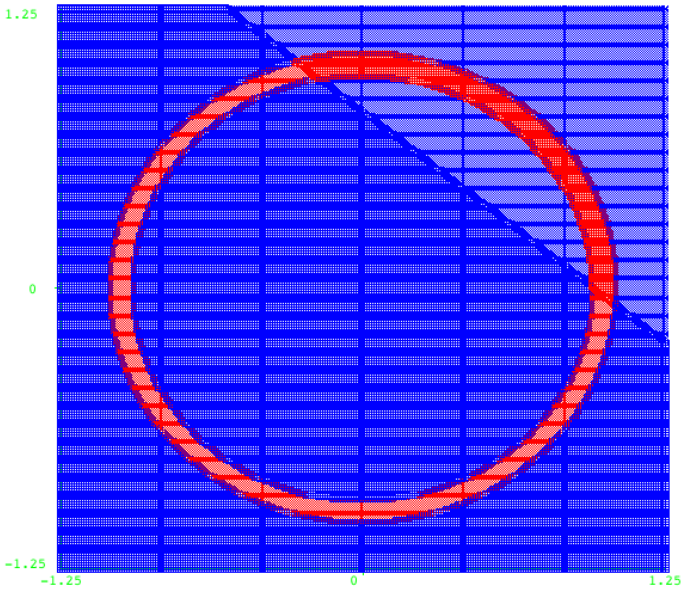

Figura 19: Visualizzazione grafica con N=5000, H=0 e L=0.3

Il quarto caso, composto da uno strato nascosti con quattro neuroni, un learning rate pari a 0.3 e 5000 epoche, è caratterizzato da risultati migliori: 50.796 % di istanze correttamente classificate e un errore relativo pari al 88.3235 %.

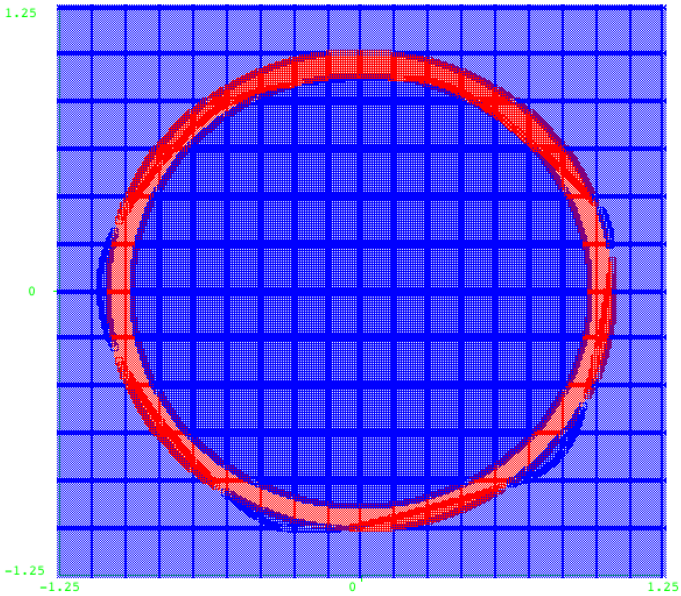

Figura 20: Visualizzazione grafica con N=5000, H=4 e L=0.3

Concludendo, è possibile notare come il risultato migliore si abbia nel quarto caso, con numero di epoche pari a 5000, un solo strato nascosto con quattro neuroni e un learning rate pari a 0.3.

Esercizio 2

Scopo di questo esercizio è l'applicazione del metodo di addestramento *backpropagation* delle reti neurali al gioco della formica introdotto nel corso dell'esercitazione 1. Il classificatore utilizzato è quindi il Multilayer Perceptron impostato con tempo di simulazione pari a 10000 epoche, learning rate pari a 0.3 e utilizzando l'intero training set come test set.

Ciascuna rete neurale analizzata prevede $2n+1$ neuroni, con n pari al numero di vicini (caselle viste dalla formica): nei casi in cui m=1, le caselle viste saranno 8 e, pertanto, si avranno reti neurali formate da 17 neuroni (fig. 21); quando m=2, le caselle viste saranno 24 e i neuroni saranno 49 (fig. 22).

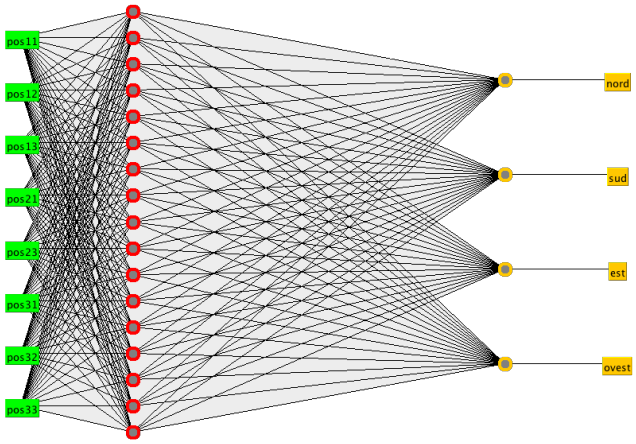

Figura 21: Rete neurale del caso m=1

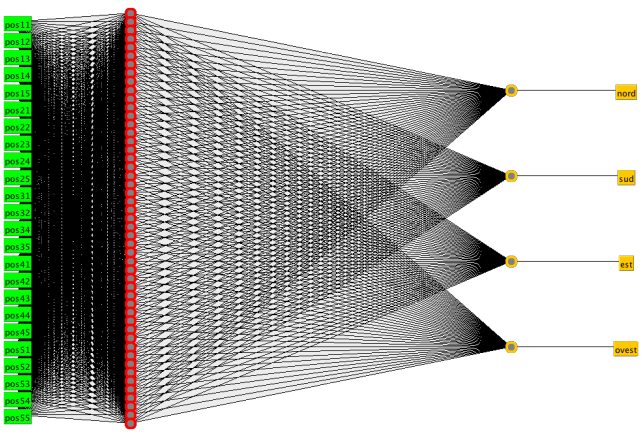

Figura 22: Rete neurale del caso m=2

Di seguito vengono mostrati i risultati ottenuti utilizzando le reti neurali e, successivamente, vengono confrontati con i risultati ottenuti utilizzando gli alberi di decisione, anche applicando i classificatori al gioco della formica, facendo muovere l'insetto virtuale in modo automatico.

1 partita - m=1

```
=== Evaluation on training set ===

Correctly Classified Instances          19        95 %
Incorrectly Classified Instances         1         5 %
Kappa statistic                          0.9275
Mean absolute error                      0.0432
Root mean squared error                  0.1375
Relative absolute error                 12.186  %
Root relative squared error             32.8463 %
Total Number of Instances               20

=== Confusion Matrix ===

 a b c d   <-- classified as
 8 0 0 0 | a = nord
 0 2 0 0 | b = sud
 1 0 3 0 | c = est
 0 0 0 6 | d = ovest
```

Applicando il classificatore al training set, in questo caso si ha un incremento delle prestazioni rispetto all'albero di decisione, che riusciva a classificare correttamente solo l'85% delle istanze. La classificazione della decisione di andare verso *sud*, infatti, questa volta risulta sempre corretta, mentre con gli alberi di decisione veniva sbagliata.

Partita	***Albero di decisione***		***Rete neurale***	
	Stato formica	**Punteggio**	**Stato formica**	**Punteggio**
1	Morta	-30	Morta	-27
2	Viva	-5	Morta	-29
3	Morta	-33	Morta	-30
4	Viva	0	Morta	-25
5	Morta	-31	Morta	-27

Tabella 7: Risultati delle partite giocate in automatico con 1 partita e m=1

Utilizzando il classificatore per far muovere automaticamente la formica, tutavia, la rete neurale ha risultati peggiori rispetto all'albero di decisione, facendo morire sempre la formica.

1 partita - m=2

```
=== Evaluation on training set ===

Correctly Classified Instances          20        100 %
Incorrectly Classified Instances         0          0 %
Kappa statistic                          1
Mean absolute error                      0.003
Root mean squared error                  0.0061
Relative absolute error                  0.884  %
Root relative squared error              1.4904 %
Total Number of Instances               20

=== Confusion Matrix ===

 a b c d   <-- classified as
 2 0 0 0 | a = nord
 0 8 0 0 | b = sud
 0 0 8 0 | c = est
 0 0 0 2 | d = ovest
```

Anche in questo caso, i risultati di valutazione sul training set risultano migliori rispetto alla versione con albero di decisione, in cui la classificazione corretta si limitava al 90% delle istanze.

Partita	***Albero di decisione***		***Rete neurale***	
	Stato formica	**Punteggio**	**Stato formica**	**Punteggio**
1	Viva	3	Morta	-30
2	Morta	-30	Morta	-29
3	Viva	2	Morta	-29
4	Viva	0	Morta	-29
5	Viva	1	Morta	-30

Tabella 8: Risultati delle partite giocate in automatico con 1 partita e m=2

Tuttavia, anche nel caso di $m = 2$, i risultati dell'applicazione al gioco della formica sono scadenti, causando sempre la morte della formica.

5 partite - m=1

```
=== Evaluation on training set ===

Correctly Classified Instances          91        91 %
Incorrectly Classified Instances         9         9 %
Kappa statistic                          0.8774
Mean absolute error                      0.0736
```

```
Root mean squared error                  0.1775
Relative absolute error                 20.0918 %
Root relative squared error             41.5093 %
Total Number of Instances              100

=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 32  0  0  0 |  a = nord
  1 18  0  2 |  b = sud
  3  0 26  2 |  c = est
  0  1  0 15 |  d = ovest
```

Aumentando il numero di partite giocate, i risultati di valutazione sul training set risultano sempre migliori rispetto alla versione con albero di decisione, in cui la classificazione corretta si limitava al 81% delle istanze.

Partita	***Albero di decisione***		***Rete neurale***	
	Stato formica	**Punteggio**	**Stato formica**	**Punteggio**
1	Viva	3	Morta	-28
2	Morta	-29	Morta	-26
3	Viva	4	Viva	8
4	Viva	3	Morta	-26
5	Viva	5	Morta	-29

Tabella 9: Risultati delle partite giocate in automatico con 5 partite e m=1

In questo caso, si riesce a ottenere una partita in cui la formica rimane viva, con un punteggio superiore rispetto alla versione con alberi di decisione. Tuttavia, nella maggioranza dei casi la formica muore, pertanto le prestazioni non risultano migliori della versione con albero di decisione.

5 partite - m=2

```
=== Evaluation on training set ===

Correctly Classified Instances          91        93.8144 %
Incorrectly Classified Instances         6         6.1856 %
Kappa statistic                          0.9161
Mean absolute error                      0.0363
Root mean squared error                  0.1502
Relative absolute error                  9.8215 %
Root relative squared error             34.9441 %
Total Number of Instances               97

=== Confusion Matrix ===
```

```
  a  b  c  d   <-- classified as
 17  0  0  1 |  a = nord
  1 25  1  0 |  b = sud
  0  3 18  0 |  c = est
  0  0  0 31 |  d = ovest
```

I risultati risultano sempre migliori rispetto all'albero di decisione, anche se leggermente meno. Infatti, mentre nei casi precedenti la differenza era di 10 punti percentuale, nel caso di albero di decisione le classificazioni corrette erano 84.5361% mentre con le reti neurali sono 93.8144%.

Partita	***Albero di decisione***		***Rete neurale***	
	Stato formica	**Punteggio**	**Stato formica**	**Punteggio**
1	Morta	-27	Morta	-26
2	Morta	-26	Morta	-26
3	Viva	5	Morta	-28
4	Morta	-30	Morta	-22
5	Morta	-27	Morta	-24

Tabella 10: Risultati delle partite giocate in automatico con 5 partite e m=2

Come si vede da questi risultati, l'aumento di campo visivo della formica non porta beneficio alla versione con rete neurale, così come non lo portava nella versione con albero di decisione.

10 partite - m=1

```
=== Evaluation on training set ===

Correctly Classified Instances         177        88.9447 %
Incorrectly Classified Instances        22        11.0553 %
Kappa statistic                          0.8516
Mean absolute error                      0.088
Root mean squared error                  0.2071
Relative absolute error                 23.6072 %
Root relative squared error             47.9691 %
Total Number of Instances              199

=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 54  0  0  0 |  a = nord
  1 41  2  2 |  b = sud
  8  1 47  2 |  c = est
  4  1  1 35 |  d = ovest
```

In questo caso, la percentuali di classificazioni corrette scende sotto al 90%, ma risulta sempre più alta della versione con alberi di decisione, in cui si attestava al 81.406%.

Partita	***Albero di decisione***		***Rete neurale***	
	Stato formica	**Punteggio**	**Stato formica**	**Punteggio**
1	Viva	2	Morta	-27
2	Viva	0	Morta	-28
3	Morta	-30	Morta	-28
4	Viva	0	Morta	-26
5	Viva	4	Morta	-29

Tabella 11: Risultati delle partite giocate in automatico con 10 partite e m=1

Applicando il classificatore al gioco della formica si può notare che, aumentando ulteriormente il numero di partite giocate, mentre nel caso con albero di decisione si notava un lieve miglioramento delle prestazioni, la rete neurale porta a risultati simili ai precedenti.

10 partite - m=2

```
=== Evaluation on training set ===

Correctly Classified Instances         189        97.4227 %
Incorrectly Classified Instances         5         2.5773 %
Kappa statistic                          0.9655
Mean absolute error                      0.0226
Root mean squared error                  0.0968
Relative absolute error                  6.0421 %
Root relative squared error             22.3692 %
Total Number of Instances              194

=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 49  0  0  2 |  a = nord
  0 40  1  1 |  b = sud
  1  0 48  0 |  c = est
  0  0  0 52 |  d = ovest
```

In questo ultimo caso i risultati si mantengono migliori rispetto al caso con alberi di decisione, con il 97.4227% di classificazioni corrette contro l'89.1753% degli alberi di decisione. La matrice di confusione, nonostante l'alto numero di istanze, risulta molto meno sparsa rispetto a quella ottenuta utilizzando gli alberi di decisione.

Anche nel caso di $m = 2$, come si vede nella tabella 12, i risultati della mossa automatica della formica sono peggiori, anche se solo leggermente, rispetto alla versione con alberi di decisione.

Partita	*Albero di decisione*		*Rete neurale*	
	Stato formica	**Punteggio**	**Stato formica**	**Punteggio**
1	Viva	4	Viva	8
2	Viva	7	Morta	-29
3	Morta	-29	Morta	-27
4	Morta	-29	Morta	-29
5	Viva	5	Morta	-26

Tabella 12: Risultati delle partite giocate in automatico con 10 partite e m=2

Concludendo, da questi confronti si è visto come i risultati di classificazione della rete neurale sul training set siano nettamente superiori rispetto a quelli dell'albero di decisione. Tuttavia, facendo giocare alla formica alcune partite in automatico utilizzando le reti neurali, i risultati sono generalmente peggiori rispetto alla versione con albero di decisione. Inoltre, anche aumentando il numero di partite giocate o il campo visivo della formica, i risultati rimangono costanti, segno che queste modifiche non influiscono sull'addestramento della rete neurale.

Esercitazione 4

Lo scopo di questa esercitazione è la risoluzione di determinati problemi mediante l'utilizzo del calcolo evoluzionistico ed, in particolare, degli algoritmi genetici.

Gli algoritmi genetici partono da popolazioni di individui, in cui ciascun individuo rappresenta una soluzione al problema, e fanno evolvere queste popolazioni per creare nuove generazioni di individui che rappresentano una soluzione migliore al problema. Un individuo viene rappresentato attraverso il cromosoma (o genotipo), che rappresenta il suo codice genetico, e il fenotipo, ossia la manifestazione dei suoi caratteri codificati. Negli algoritmi genetici è quindi necessaria la codifica e decodifica tra genotipo e fenotipo.

Il primo step di un algoritmo genetico è la creazione di una popolazione casuale e, quindi, di un insieme di cromosomi. Successivamente, avviene la decodifica di ogni cromosoma al fine di ottenere il fenotipo dell'individuo. Per ogni individuo viene quindi valutata la sua *fitness*, chiamata anche *funzione obiettivo*, che fornisce informazioni sulla bontà della soluzione.

Un gruppo di soluzioni, detto *mating pool*, viene selezionato per l'accoppiamento. Vi sono varie tecniche di selezione che, in generale, seguono il seguente principio: più una soluzione ha una fitness elevata e maggiore sarà la probabilità che venga selezionata per la riproduzione. Questa operazione crea una nuova generazione di soluzioni (i figli), ottenuti grazie all'operazione di *clonazione* e *crossover* (mescolamento) dei geni delle coppie dei genitori. Vi è anche la possibilità che alcuni geni vengano mutati casualmente durante il processo di creazione di figli.

L'ultimo step consiste nella selezione dei sopravvissuti, che può essere:

- basata sull'*età*, tenendo in considerazione tutti i figli ed eventualmente qualche genitore (se il numero dei figli generati è inferiore a quello dei genitori);
- basata sul valore della *fitness*, con metodologie simili a quelle viste per la selezione;
- basata su *entrambe le caratteristiche*.

I sopravvissuti sono una nuova popolazione e su di essa vengono riproposte le stesse operazioni precedentemente elencate, in modo iterativo.

Si può, quindi, intuire che un algoritmo genetico non ha mai fine, e necessita della definizione di un *criterio di arresto*; generalmente vengono presi in considerazione il massimo numero di iterazioni effettuate o una soglia sul valore della fitness.

Gli esercizi sono stati svolti utilizzando il linguaggio di programmazione C++ e la libreria dedicata agli algoritmi genetici *GAlib*. Questa libreria fornisce numerosi strumenti tra cui due classi base fondamentali: *GAGenome*, dove ogni sua istanza rappresenta una soluzione del problema, e *GAGenericAlgorithm*, che rappresenta l'evoluzione

effettiva, che fa evolvere la popolazione e tiene traccia delle statistiche necessarie. Per implementare in modo corretto un algoritmo bisogna quindi:

- definire la rappresentazione delle soluzioni, il loro genotipo e il loro fenotipo, comprendendo il testo del problema;
- definire quali operazioni genetiche sono necessarie;
- definire la funzione oggetto per il calcolo della fitness.

Esercizio 1

Lo scopo di questo esercizio è minimizzare la seguente funzione:

$$f(x,y,z) = [1.5 + cos(z^2)] * [\sqrt{(5-x)^2 + (10-y)^2} + 1] \tag{3}$$

utilizzando come rappresentazione dei valori x, y, z sia stringhe di bit che numeri floating point.

La funzione si può suddividere in due sottofunzioni:

- la prima, $[1.5 + cos(z^2)]$, ha un minimo quando $cos(z^2) = -1$, ovvero quando $z^2 = \pi + 2k\pi$, cioè $z = \pm\sqrt{\pi + 2k\pi}$, mentre ha un massimo quando $z = \pm\sqrt{0 + 2k\pi}$;
- la seconda $\sqrt{(5-x)^2 + (10-y)^2} + 1$ ha un minimo quando il radicando è 0, e questo accade nel punto $(x,y) = (5,10)$, mentre il suo massimo è ∞ quando uno o entrambi i parametri in ingresso tendono ad ∞.

Dato che entrambe le sottofunzioni sono sempre positive, anche la loro moltiplicazione lo sarà, e quindi il valore minimo sarà dato dalla moltiplicazione dei valori minimi delle due funzioni. In questo caso il minimo è dato da $f(x_{min}, y_{min}, z_{min}) = f(5, 10, 1.77) = 0.5$. Dato che il coseno ha una periodicità di $2k\pi$, z avrà limite $[0, \sqrt{2\pi}] = [0, 2.5]$, in modo da poter confrontare le soluzioni con un solo valore e non con le sue periodicità.

La prima fase dell'implementazione dell'algoritmo genetico è la rappresentazione delle soluzioni. Nel caso in cui si utilizzi una sequenza di bit, è stata utilizzata come classe di genotipo *GABin2DecGenome*, mentre per la rappresentazione in floating point si è usata *GA1DArrayGenome*, dove i geni sono i tre elementi x, y e z.

Le operazioni genetiche utilizzate, oltre l'inizializzazione, sono il crossover e la mutazione, con parametri che varieranno in base alle prove effettuate.

La funzione di fitness è molto semplice da stabilire dato che è l'inversa della funzione da minimizzare; infatti, gli algoritmi genetici ottimizzano una soluzione massimizzando la funzione di fitness.

Le prove effettuate mostrano le differenze di risultati nell'utilizzo di stringhe di bit e di floating point, in cui sono stati variati di volta in volta i seguenti parametri:

- dimensione della popolazione;
- criterio di arresto, ovvero il massimo numero iterazioni;

- distribuzione degli operatori per la nuova generazione, ovvero la probabilità di crossover e la probabilità di mutazione.

La prima sessione di prove tiene in considerazione come parametro variabile la dimensione della popolazione, mentre i parametri costanti sono:

- numero di iterazioni = 100;
- probabilità di crossover = 0.3;
- probabilità di mutazione = 0.1.

	Best Effort				***Average***			
Popolazione	**Score**	**x**	**y**	**z**	**Score**	**x**	**y**	**z**
10	2.37	3.00	7.00	1.72	156.64	154.05	160.16	1.69
50	2.40	2.00	12.00	1.70	38.49	47.34	43.39	1.67
100	0.83	5.00	10.00	1.51	14.88	19.63	20.14	1.72
500	0.52	5.00	10.00	1.83	3.22	5.62	10.97	1.72
1000	0.50	5.00	10.00	1.77	2.35	5.59	9.90	1.72

Tabella 13: Risultati della simulazione utilizzando la rappresentazione come stringhe di bit e come parametro variabile il numero della popolazione

Osservando la tabella 13 si può notare come i risultati inizino ad essere accettabili con una popolazione di dimensione maggiore a 500. In ogni caso, si nota che, anche se la popolazione è ridotta, vi sono alcune prove che convergono in modo notevole.

	Best Effort				***Average***			
Popolazione	**Score**	**x**	**y**	**z**	**Score**	**x**	**y**	**z**
10	2.34	1.92	11.89	11.89	9.85	4.46	5.85	158.35
50	1.08	6.07	9.79	163.26	4.42	2.45	4.95	107.38
100	1.17	3.66	9.86	9.86	3.57	2.85	7.83	108.69
500	0.64	4.93	9.75	1.73	1.85	4.50	9.37	165.17
1000	0.64	5.11	10.25	1.78	1.89	3.76	10.07	175.47

Tabella 14: Risultati della simulazione utilizzando la rappresentazione floating point e come parametro variabile il numero della popolazione

Nella tabella 14 sono indicati i risultati riguardanti l'utilizzo dei floating point. In questo caso, il miglioramento avviene quando la popolazione conta 50 individui; aumentando la popolazione i miglioramenti restano limitati.

Mediamente i risultati utilizzando i numeri floating point sono leggermente superiori rispetto a quelli ottenuti dalla rappresentazione a stringhe di bit, soprattutto in popolazioni di piccole dimensioni. Di contro, la soluzione migliore nella rappresentazione a stringhe di bit trova precisamente l'ottimo, anche con popolazioni medie, cosa che non accade con la rappresentazione in floating point.

Le prove effettuate con popolazione di 1000 individui, nel caso di rappresentazione con stringhe di bit, hanno un tempo di esecuzione di un ordine di grandezza superiore alla rappresentazione floating point (715ms contro 110ms).

La seconda sessione di prove tiene in considerazione come parametro variabile il numero di generazioni, ovvero quante iterazioni compie l'algoritmo genetico prima di arrestarsi. I parametri fissi sono:

- dimensione popolazione = 100;
- probabilità di crossover = 0.3;
- probabilità di mutazione = 0.1.

	Best Effort				***Average***			
Generazioni	**Score**	**x**	**y**	**z**	**Score**	**x**	**y**	**z**
10	18.08	35.00	8.00	1.65	497.94	515.35	522.86	1.73
50	1.00	5.00	11.00	1.76	15.80	20.04	21.54	1.74
100	0.51	5.00	10.00	1.72	3.72	6.79	11.38	1.74
500	0.50	5.00	10.00	1.77	0.88	5.02	9.79	1.78
1000	0.50	5.00	10.00	1.77	0.61	5.06	9.95	1.78

Tabella 15: Risultati della simulazione utilizzando la rappresentazione stringhe di bit e come parametro variabile il numero di generazioni

	Best Effort				***Average***			
Generazioni	**Score**	**x**	**y**	**z**	**Score**	**x**	**y**	**z**
10	1.43	4.83	8.35	449.98	3.85	2.29	6.49	237.98
50	1.40	4.69	11.77	4.69	3.81	2.71	5.94	306.82
100	1.13	5.23	8.90	1.69	3.69	2.88	6.61	121.49
500	0.66	5.28	10.15	1.77	3.71	2.92	6.69	187.19
1000	0.90	5.75	9.84	151.94	3.49	2.93	7.52	139.93

Tabella 16: Risultati esercizio uno utilizzando la rappresentazione floating point e come parametro variabile il numero di generazioni

All'aumentare delle iterazioni, la rappresentazione a stringhe di bit risulta migliore, sia rispetto all'altra sia rispetto ai dati precedenti. Da notare che con alto numero di generazioni la media delle soluzioni è molto vicina alla soluzione ottima del problema. Nel caso di rappresentazione floating point, i risultati sono più scadenti rispetto alle prove precedenti; si può dedurre che la rappresentazione floating point dà risultati migliori per popolazioni ampie rispetto ad un alto numero di generazioni.

La terza sessione di prove tiene in considerazione come parametro variabile la probabilità di mutazione, mentre i parametri fissi sono:

- dimensione popolazione = 100;

- generazioni = 100;
- probabilità di crossover = 0.3.

Nella tabella 17 sono mostrati i risultati ottenuti utilizzando la rappresentazione stringa di bit, mentre nella tabella 18 sono mostrati quelli ottenuti con la rappresentazione floating point.

	Best Effort				***Average***			
Mutazione	**Score**	**x**	**y**	**z**	**Score**	**x**	**y**	**z**
0.005	0.50	5.00	10.00	1.77	0.91	5.26	9.73	1.78
0.05	0.50	5.00	10.00	1.77	1.34	5.19	9.74	1.77
0.5	2.66	8.00	7.00	1.73	421.18	426.96	386.29	1.67

Tabella 17: Risultati della simulazione utilizzando la rappresentazione stringhe di bit e come parametro variabile la probabilità di mutazione

	Best Effort				***Average***			
Mutazione	**Score**	**x**	**y**	**z**	**Score**	**x**	**y**	**z**
0.005	1.87	4.45	12.64	1.73	7.95	6.75	13.73	40.25
0.05	0.96	4.10	10.25	113.67	3.98	2.33	6.90	153.08
0.5	0.93	5.19	9.17	244.87	3.32	3.15	7.18	136.82

Tabella 18: Risultati della simulazione utilizzando la rappresentazione floating point e come parametro variabile la probabilità di mutazione

In queste prove si può notare che la rappresentazione a stringhe di bit peggiora con l'aumento della probabilità di mutazione, mentre la rappresentazione floating point migliora, anche se leggermente, al suo variare.

La quarta sessione di prove tiene in considerazione come parametro variabile la probabilità di crossover, mentre i parametri fissi sono:

- dimensione popolazione = 100;
- generazioni = 100;
- probabilità di mutazione = 0.1.

Nella tabella 19 sono mostrati i risultati ottenuti utilizzando la rappresentazione come stringa di bit, mentre nella tabella 20 sono mostrati quelli ottenuti con la rappresentazione in floating point.

Queste ultime prove fanno notare come un aumento del crossover non influenzi eccessivamente la ricerca della soluzione della rappresentazione a stringhe di bit (peggiora lievemente), mentre la rappresentazione floating point migliora con il suo aumentare.

	Best Effort				*Average*			
Crossover	**Score**	**x**	**y**	**z**	**Score**	**x**	**y**	**z**
0.05	0.50	5.00	10.00	1.78	3.34	6.35	10.88	1.76
0.05	0.56	5.00	10.00	1.76	3.56	6.04	11.33	1.76
0.5	0.52	5.00	10.00	1.70	3.94	6.65	11.18	1.75

Tabella 19: Risultati della simulazione utilizzando la rappresentazione stringhe di bit e come parametro variabile la probabilità di crossover

	Best Effort				*Average*			
Crossover	**Score**	**x**	**y**	**z**	**Score**	**x**	**y**	**z**
0.05	1.86	2.38	9.89	548.19	9.64	1.98	9.08	358.78
0.05	2.14	2.04	9.62	19	5.23	2.39	6.27	239.59
0.5	0.89	4.72	10.67	1.71	3.52	2.89	7.49	112.53

Tabella 20: Risultati della simulazione utilizzando la rappresentazione floating point e come parametro variabile la probabilità di crossover

Esercizio 1b

Lo scopo di questo esercizio è di generare una matrice formata da 1 e 0 alternati attraverso un algoritmo genetico. Per rappresentare il cromosoma viene utilizzato *GA2DBinaryStringGenome*, ovvero una matrice di bit.

La prima sessione di prove consiste nel valutare le soluzioni fornite dall'algoritmo genetico, variando la grandezza della matrice di bit ($N = w * h$), mentre tutti gli altri parametri rimangono fissati:

- popolazione = 500 individui;
- numero di generazioni = 500;
- probabilità di crossover = 0.5;
- probabilità di mutazione = 0.2.

I risultati sono illustrati nella tabella 21, dove l'errore corrisponde ad un bit di valore errato.

N	10	20	30	40	50	60	70	80	90	100
Errori	0	1	3	5	10	11	16	18	21	22
Errore %	0	5	10	12.5	20	18.3	22.8	22.5	23.3	22

Tabella 21: Risultati variando la dimensione della griglia

La seconda sessione di prove consiste nel variare il numero di individui proporzionalmente al numero di elementi nella tabella. I parametri sono:

- popolazione = $N * 50$;

- numero di generazioni = 500;
- probabilità di crossover = 0.5;
- probabilità di mutazione = 0.2.

N	10	20	30	40	50	60	70	80	90	100
Popolazione	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
Errori	0	1	3	5	7	11	12	15	19	23
Errore %	0	5	10	12.5	14	18.3	17.1	18.7	21	23

Tabella 22: Risultati variando la dimensione della griglia e della popolazione

I risultati illustrati nella tabella 22 mostrano che un aumento proporzionale della popolazione non aumenta in modo consistente la probabilità di sequenze corrette.

Nella terza sessione di prove, i cui risultati sono illustrati in tabella 23, viene fissata come probabilità di mutazione 0.2, mentre la probabilità di crossover è stata posta a 0. Gli altri parametri sono:

- popolazione = 500;
- numero di generazioni = 500.

N	10	20	30	40	50	60	70	80	90	100
Errori	0	1	2	4	7	11	13	16	17	23
Errore %	0	5	10	12.5	14	18.3	17.1	18.7	21	23

Tabella 23: Risultati variando la dimensione della griglia con crossover nullo

Nella quarta sessione di prove, i cui risultati sono illustrati in tabella 24, viene fissata la probabilità di mutazione a 0, mentre la probabilità di crossover è stata posta a 0.5. Gli altri parametri sono:

- popolazione = 500;
- numero di generazioni = 500;

N	10	20	30	40	50	60	70	80	90	200
Errori	0	0	0	0	0	0	0	0	0	2
Errore %	0	0	0	0	0	0	0	0	0	1

Tabella 24: Risultati variando la dimensione della griglia con mutazione nulla

Nell'ultimo caso si nota come i risultati siano nettamente superiori agli altri casi precedenti che, invece, hanno risultati confrontabili.

Esercizio 2

Lo scopo di questo esercizio è la risoluzione del problema delle N regine tramite gli algoritmi genetici. Questo problema consiste nel posizionare N regine su un scacchiera NxN, senza che nessuna di esse si trovi sotto scacco da parte di un'altra.

La rappresentazione delle soluzioni è stata effettuata utilizzando una rappresentazione implicita, in particolar modo utilizzando stringhe di bit. Il genotipo è una sequenza di NxM bit, dove $M = \log_2 N$. Per ottenere il fenotipo, insiemi di M geni vengono decodificati in un vettore di N interi, tramite l'algoritmo consigliato nella consegna dell'esercizio. In questo modo, il fenotipo contiene un vettore di indici per rappresentare le posizioni delle regine; il numero dell'elemento rappresenta la riga mentre il valore dell'elemento rappresenta la colonna. Con questo tipo di decodifica è possibile semplificare il problema in quanto le regine non possono avere coordinate uguali e, quindi, nell'implementazione della funzione di fitness verranno valutati solo i casi in cui le regine sono sotto scacco diagonalmente.

I test, per ogni combinazione di parametri, sono stati effettuati facendo la media su 100 prove, tenendo in considerazione l'errore medio, ossia quante coppie di regine sono sotto scacco, e la percentuale di soluzioni esatte.

Nel primo caso di studio verrà considerata un scacchiera classica, quindi con $N = 8$. Nella tabelle 25, 26 e 27 vengono mostrati i risultati nei casi in cui i parametri variabili sono il numero di generazioni e/o la dimensione della popolazione, mentre i parametri fissi sono:

- la probabilità di mutazione = 0.1;
- la probabilità di crossover = 0.3;

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti(%)**
10	100	0.03	97
50	100	0.00	100
100	100	0.00	100
500	100	0.00	100
1000	100	0.00	100

Tabella 25: Risultati del gioco con 8 regine, al variare della popolazione

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti(%)**
100	10	0.08	92
100	50	0.00	100
100	100	0.00	100
100	500	0.00	100
100	1000	0.00	100

Tabella 26: Risultati del gioco con 8 regine, al variare del numero di generazioni

Popolazione	Generazioni	Errore Medio	Probabilità Esatti(%)
10	10	0.77	28
50	50	0.00	100
100	100	0.00	100
500	500	0.00	100
1000	1000	0.00	100

Tabella 27: Risultati del gioco con 8 regine, al variare della popolazione e del numero di generazioni

Come si può notare, per $N = 8$ bastano popolazioni e iterazioni abbastanza ridotte per convergere alla soluzione ottimale.

La tabella 28 mostra i risultati variando le probabilità di mutazione e crossover. Come si può notare i miglioramenti dipendono soprattutto dalla probabilità di mutazione. I parametri fissi sono:

- popolazione = 100;
- generazioni = 100;

Mutazione	Crossover	Errore Medio	Probabilità Esatti (%)
0.01	0.01	0.45	55
0.01	0.1	0.41	59
0.01	0.5	0.21	79
0.1	0.01	0.00	100
0.5	0.01	0.01	99
0.1	0.1	0.00	100
0.5	0.5	0.00	100

Tabella 28: Risultati del gioco con 8 regine, al variare di probabilità di mutazione e crossover

Una possibile soluzione al problema delle otto regine è illustrata in Figura 23.

Figura 23: Possibile soluzione

Nel secondo caso di studio la scacchiera viene aumentata a $N = 16$. Nelle tabelle 29, 30 e 31 vengono mostrati i risultati variando popolazione e/o numero di generazioni, mentre i parametri fissi sono:

- probabilità mutazione = 0.1;
- probabilità crossover = 0.3;

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti (%)**
10	100	2.55	0
50	100	2.07	0
100	100	1.69	0
500	100	1.18	0
1000	100	0.9	13

Tabella 29: Risultati del gioco con 16 regine, al variare della popolazione

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti (%)**
100	10	2.64	0
100	50	1.9	0
100	100	1.74	0
100	500	1.12	9
100	1000	0.87	14

Tabella 30: Risultati del gioco con 16 regine, al variare del numero di generazioni

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti (%)**
10	10	3.88	0
50	50	2.20	1
100	100	1.74	2
500	500	0.76	24
1000	1000	0.39	61

Tabella 31: Risultati del gioco con 16 regine, al variare del numero di generazioni e della popolazione

I risultati, ovviamente, sono peggiori dei precedenti, in quanto il problema risulta più complicato. Per avere una buona probabilità di convergenza occorre aumentare di molto sia la dimensione della popolazione sia il numero di generazioni.

Di seguito, nella tabella 32, sono invece mostrati i risultati variando le probabilità di mutazione e di crossover e mantenendo fissi i parametri:

- popolazione = 100;
- numero di generazioni = 100;

Mutazione	Crossover	Errore Medio	Probabilità Esatti (%)
0.01	0.01	1	17
0.01	0.1	0.92	21
0.01	0.5	0.69	35
0.1	0.01	1.68	2
0.5	0.01	1.82	0
0.1	0.1	1.73	1
0.5	0.5	1.83	0

Tabella 32: Risultati del gioco con 16 regine, al variare di probabilità di mutazione e crossover

In questo caso la probabilità di crossover incide molto sull'individuazione del valore ottimale, mentre l'operazione mutazione peggiora notevolmente i risultati al suo aumentare.

Una possibile soluzione al problema delle sedici regine è illustrata in Figura 24.

```
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

Figura 24: Possibile soluzione

Esercizio 2b

In questo esercizio viene risolto il problema delle N regine utilizzando la codifica diretta del cromosoma. Il cromosoma è composto da una matrice di bit, dove la riga rappresenta una caratteristica, x o y, di una regina, mentre la riga è una rappresentazione binaria della coordinata della regina; si è utilizzata la classe *GA2DBinaryStringGenome* per rappresentare il genoma.

La funzione di fitness è un po' diversa dalla precedente dato che bisogna confrontare anche le regine su colonne o righe uguali, con la possibilità che siano anche sulla stessa casella.

Le tabelle 33, 34 e 35 mostrano come variano le soluzioni al variare di popolazione e numero di generazioni. I parametri fissi sono:

- probabilità mutazione = 0.1;
- probabilità crossover = 0.3;

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti (%)**
10	100	3.18	0
50	100	2.67	0
100	100	2.49	0
500	100	1.95	0
1000	100	1.73	1

Tabella 33: Risultati del gioco con 8 regine, al variare della popolazione

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti (%)**
100	10	3.52	0
100	50	2.77	0
100	100	2.54	0
100	500	1.65	2
100	1000	1.39	2

Tabella 34: Risultati del gioco con 8 regine, al variare del numero di generazioni

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti (%)**
10	10	4.7	0
50	50	2.98	0
100	100	2.46	1
500	500	1.44	4
1000	1000	0.96	12

Tabella 35: Risultati del gioco con 8 regine, al variare della popolazione e del numero di generazioni

In queste prove si nota un leggero miglioramento con l'aumento del numero di iterazioni, rispetto all'aumento della popolazione. Tuttavia, i risultati sono molto peggiori rispetto all'implementazione dell'esercizio precedente: per avere una buona probabilità di ottenere la soluzione corretta bisogna aumentare di molto la popolazione e il numero di generazioni.

Nella tabella 36 sono presentati i risultati variando probabilità di crossover e mutazioni. I parametri fissi sono:

- popolazione = 500;
- generazioni = 500;

Mutazione	Crossover	Errore Medio	Probabilità Esatti (%)
0.01	0.01	0.09	91
0.01	0.1	0.22	78
0.01	0.5	0.21	79
0.1	0.01	1.32	4
0.5	0.01	1.85	1
0.1	0.1	1.35	6
0.5	0.5	1.86	0

Tabella 36: Risultati del gioco con 8 regine, al variare di probabilità di mutazione e crossover

In questo caso l'aumento della probabilità di mutazione peggiora sensibilmente la soluzione. Per paragonarlo all'esercizio precedente sono stati impostati i seguenti parametri fissi (tabella 37):

- popolazione = 50;
- generazioni = 50;

Mutazione	Crossover	Errore Medio	Probabilità Esatti (%)
0.01	0.01	1.37	9
0.01	0.1	1.37	10
0.01	0.5	1.30	13
0.1	0.01	2.89	0
0.5	0.01	3.47	0
0.1	0.1	3.06	0
0.5	0.5	3.49	0

Tabella 37: Risultati del gioco con 8 regine, al variare di probabilità di mutazione e crossover

Anche in questo caso una probabilità alta di mutazione peggiora drasticamente la ricerca della soluzione.

Le tabelle 38, 39 e 40 fanno riferimento al problema con $N = 16$ e mostrano come variano le soluzioni al variare di popolazione e numero di generazioni. I parametri fissi sono:

- probabilità mutazione = 0.1;
- probabilità crossover = 0.3;

La tabella 41 mostra i risultati variando probabilità di crossover e mutazione e lasciando come parametri fissi:

- popolazione = 500;
- generazioni = 500;

Dai risultati è evidente che con la codifica diretta la ricerca dell'ottimo è estremamente più difficile.

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
10	100	11.10	0
50	100	10.27	0
100	100	9.83	0
500	100	8.98	0
1000	100	8.58	0

Tabella 38: Risultati del gioco con 16 regine, al variare della popolazione

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
100	10	11.64	0
100	50	10.25	0
100	100	10.02	0
100	500	8.59	0
100	1000	8.19	0

Tabella 39: Risultati del gioco con 16 regine, al variare del numero di generazioni

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
10	10	13.65	0
50	50	10.82	0
100	100	9.85	0
500	500	8.03	0
1000	1000	7.73	0

Tabella 40: Risultati del gioco con 16 regine, al variare della popolazione e del numero di generazioni

Mutazione	Crossover	Errore Medio	Probabilità Esatti (%)
0.01	0.01	2.56	0
0.01	0.1	2.55	0
0.01	0.5	3.09	0
0.1	0.01	7.93	0
0.5	0.01	8.44	0
0.1	0.1	7.98	0
0.5	0.5	8.6	0

Tabella 41: Risultati del gioco con 16 regine, al variare di probabilità di mutazione e crossover

Esercizio 3

Questo esercizio è incentrato sulla soluzione del problema del commesso viaggiatore (*TSP, Travelling Salesman Problem*), che consiste nel trovare il percorso ottimo che unisca *N* luoghi da visitare. Come percorso ottimo si considera quello che minimizza la distanza e che non ripassa più volte dallo stesso luogo.

I luoghi da visitare sono rappresentati dalle loro coordinate su un piano euclideo 2D

di dimensione $100x100$. Dato che il piano è euclideo, la distanza tra due punti, utilizzata per calcolare la distanza totale e la fitness, viene rappresentata dalla distanza euclidea:

$$distanza = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Come rappresentazione dei dati sono state utilizzate le permutazioni, in modo da eliminare direttamente la possibilità di passare per uno stesso punto. La funzione di fitness calcola la distanza totale per ogni individuo della popolazione utilizzando la matrice delle distanze, tentando di minimizzarla.

La disposizione dei luoghi da visitare (dati di ingresso) è fissa, in modo tale da poter valutare in modo corretto i vari casi che si incontrano modificando determinati parametri. Per facilitare la comprensione della soluzione è stata scelta una distribuzione circolare dei luoghi, come mostrato in tabella 42 e fig. 25 per il caso di $N = 8$.

Indice	0	1	2	3	4	5	6	7
x	40	60	90	95	85	30	10	15
y	80	90	85	65	40	25	10	60

Tabella 42: Coordinate dei luoghi con N=8

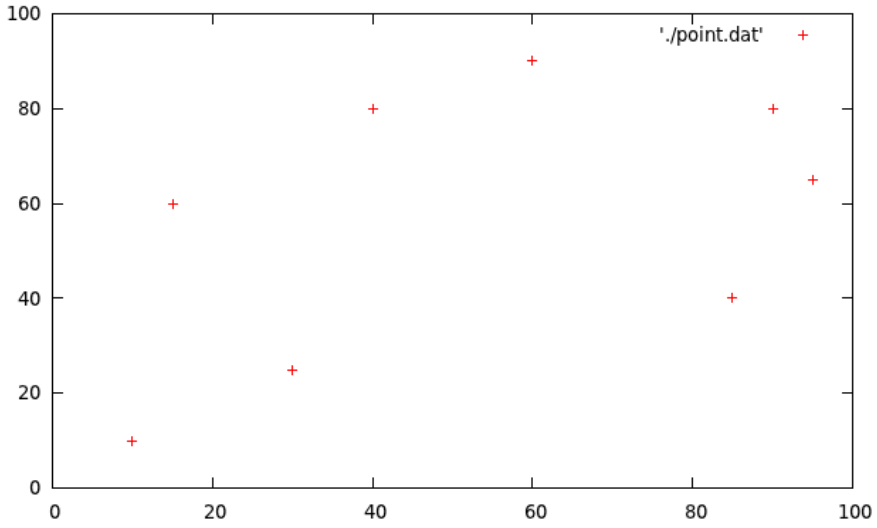

Figura 25: Luoghi con N=8

Come parametri di valutazione vengono considerati il percorso medio e la probabilità di successo, sempre considerati su un test set di dimensione 100. La probabilità di successo è confrontata con la distanza minima trovata su prove con popolazioni e iterazioni di grandi dimensioni, quindi considerata come ottimo effettivo.

I risultati dei primi test, effettuati con $N = 8$, al variare della dimensione della popolazione e del numero di generazioni sono mostrati nelle tabelle 43, 44 e 45 . I parametri fissi sono:

- probabilità di mutazione = 0.1;
- probabilità di crossover = 0.3;

Popolazione	**Generazioni**	**Distanza Media**	**Probabilità Esatti (%)**
10	100	265.30	80
50	100	258.38	99
100	100	258	100
500	100	258	100
1000	100	258	100

Tabella 43: Risultati del problema con 8 luoghi, al variare del numero di popolazione

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti (%)**
100	10	263.35	74
100	50	258	100
100	100	258	100
100	500	258	100
100	1000	258	100

Tabella 44: Risultati del problema con 8 luoghi, al variare del numero di generazioni

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti (%)**
10	10	309.09	12
50	50	258.22	98
100	100	258	100
500	500	258	100
1000	1000	258	100

Tabella 45: Risultati del problema con 8 luoghi, al variare del numero di popolazione e generazioni

Dai risultati si può intuire che per un piccolo numero di nodi la ricerca dell'ottimo è piuttosto semplice e risulta possibile anche con popolazioni o generazioni ridotte. La sequenza ottima è [0 1 2 3 4 5 6 7], illustrata in fig. 26, con distanza minima pari a 258.

La seconda serie di prove consiste nell'aggiungere quattro luoghi, ovvero impostando $N = 12$, inseriti come mostrato in tabella 50 (in grassetto i luoghi aggiunti) e fig. 27.

Indice	0	1	2	3	4	5	6	7	8	9	10	11
x	40	60	90	95	85	30	10	15	**70**	**60**	**40**	**10**
y	80	90	85	65	40	25	10	60	**10**	**40**	**20**	**40**

Tabella 46: Coordinate dei luoghi con N=12

Nelle tabelle 47, 48 e 49 sono mostrati i risultati ottenuti variando dimensione della popolazione e numero di generazioni. I parametri fissi sono:

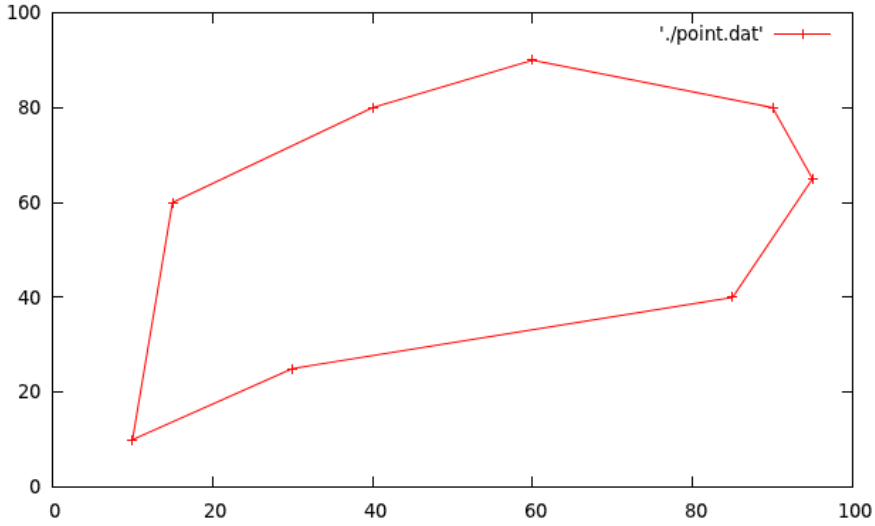

Figura 26: Soluzione Ottima con N=8

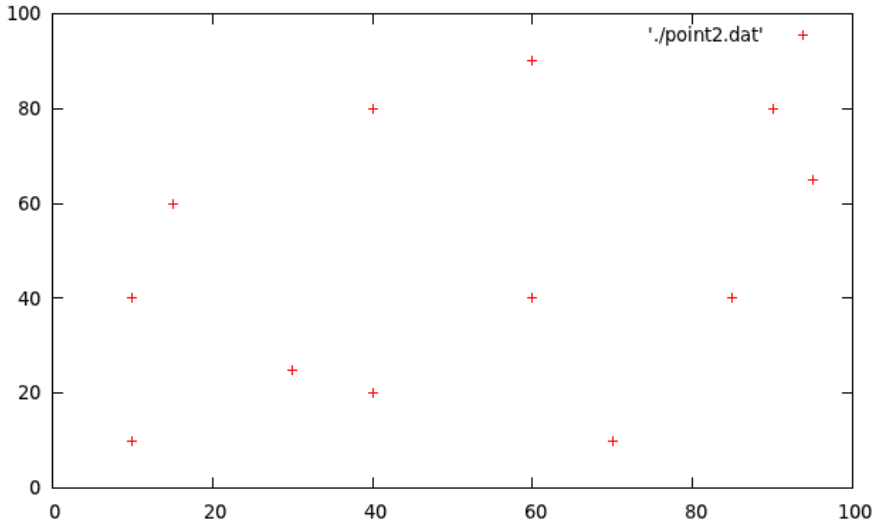

Figura 27: Luoghi con N=12

- probabilità di mutazione = 0.1;
- probabilità di crossover = 0.3;

Dai risultati ottenuti si nota una difficoltà molto maggiore nella ricerca dell'ottimo anche solo con l'aggiunta di quattro luoghi. La sequenza ottima è [1 2 3 4 9 8 10 5 6 11 7 0], illustrata in fig. 28, con distanza minima di 308.

L'ultima serie di prove consiste nel testare l'algoritmo con un numero di nodi $N = 16$, disposti come illustrato in tabella 50 e fig. 29.

Popolazione	Generazioni	Distanza Media	Probabilità Esatti (%)
10	100	370.27	4
50	100	338.38	9
100	100	324.20	22
500	100	309.96	46
1000	100	305.64	53

Tabella 47: Risultati del problema con 12 luoghi, al variare della popolazione

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
100	10	402.22	0
100	50	336.77	6
100	100	324.61	21
100	500	313.24	31
100	1000	313.58	37

Tabella 48: Risultati del problema con 12 luoghi, al variare del numero di generazioni

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
10	10	453.83	0
50	50	351.58	7
100	100	320.80	22
500	500	304.33	56
1000	1000	301.96	69

Tabella 49: Risultati del problema con 12 luoghi, al variare della popolazione e del numero di generazioni

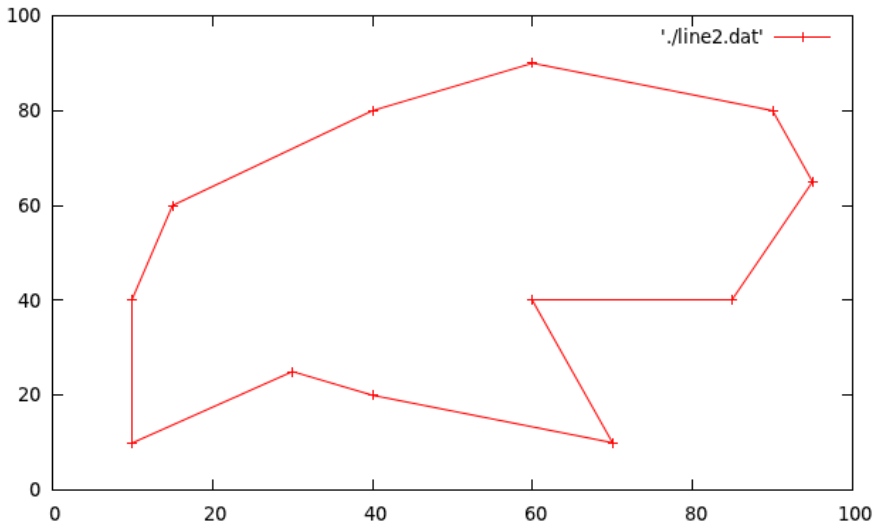

Figura 28: Soluzione ottima con N=12

Indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x	40	60	90	95	85	30	10	15	70	60	40	10	**20**	**50**	**70**	**80**
y	80	90	85	65	40	25	10	60	10	40	20	40	**80**	**50**	**85**	**95**

Tabella 50: Coordinate dei luoghi con N=16

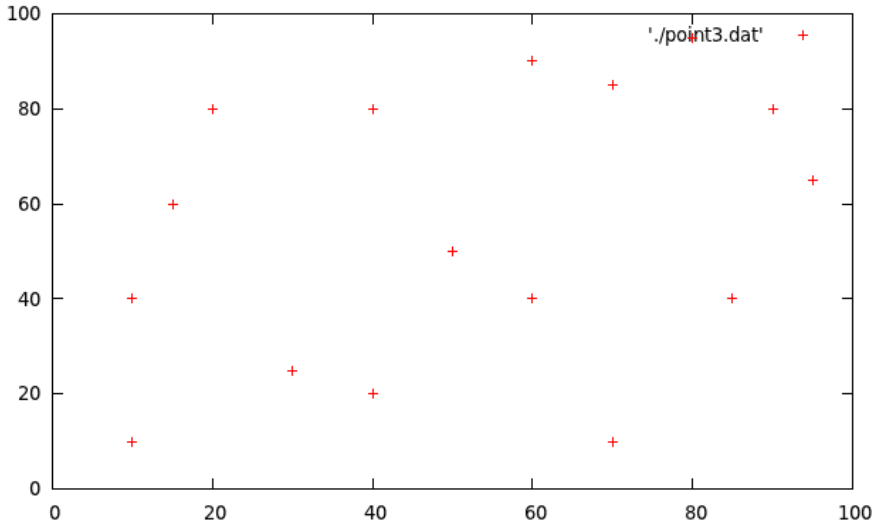

Figura 29: Luoghi con N=16

Le tabelle 51, 52 e 53 illustrano i risultati del problema con N=16, considerando come parametri fissi:

- probabilità mutazione = 0.1;
- probabilità crossover = 0.3;

Popolazione	**Generazioni**	**Distanza Media**	**Probabilità Esatti (%)**
10	100	570.38	0
50	100	530.74	0
100	100	514.55	0
500	100	478.21	1
1000	100	471.81	0

Tabella 51: Risultati del problema con 16 luoghi, al variare della popolazione

Come si può notare dai risultati, la complessità aumenta in modo esponenziale rispetto al numero di nodi presenti nel grafo. Nel caso di $N = 16$ diventa molto difficile trovare un ottimo assoluto. La soluzione migliore trovata è [0 10 1 8 2 3 9 4 15 14 13 12 6 5 7 11], illustrata in fig. 30.

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
100	10	621.34	0
100	50	542.24	0
100	100	511.47	0
100	500	475.59	1
100	1000	471.11	0

Tabella 52: Risultati del problema con 16 luoghi, al variare del numero di generazioni

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
10	10	690.58	0
50	50	568.19	0
100	100	515.34	0
500	500	469.1	0
1000	1000	445.98	0

Tabella 53: Risultati del problema con 16 luoghi, al variare della popolazione e del numero di generazioni

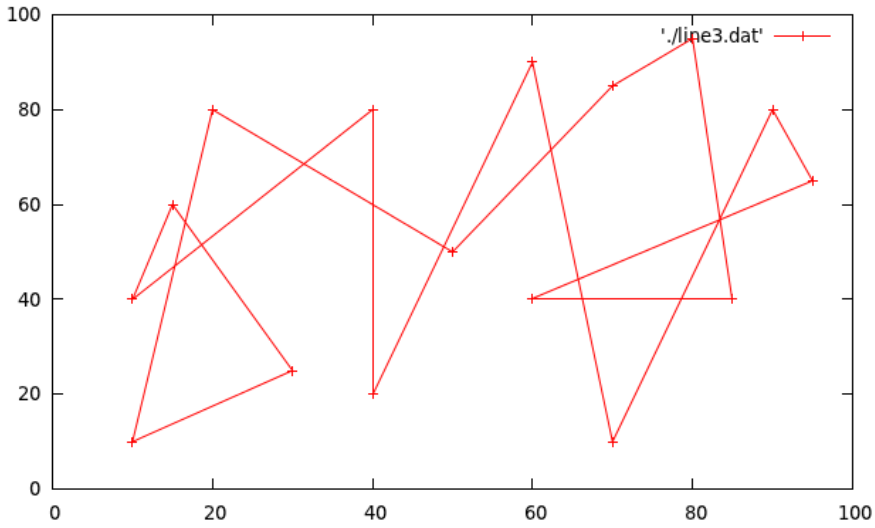

Figura 30: Migliore soluzione trovata con N=16

Esercizio 3b

Questo esercizio consiste nel risolvere il problema dell'esercizio precedente (TSP) tramite l'uso di codifica diretta. La rappresentazione è quindi diretta, cioè il cromosoma è composto da un vettore di geni, i quali assumono l'indice di un luogo. La funzione di fitness è leggermente diversa dall'implementazione precedente, dato che è possibile passare su uno stesso luogo per più di una volta; questo caso viene penalizzato aumentando di cento volte la distanza effettiva.

I primi test sono stati effettuati con 8 luoghi ($N = 8$), dislocati come in tabella 42, e

i risultati al variare della dimensione della popolazione e del numero di generazioni sono mostrati nelle tabelle 54, 55 e 56. I parametri fissi sono:

- probabilità mutazione = 0.1;
- probabilità crossover = 0.3;

Popolazione	**Generazioni**	**Distanza Media**	**Probabilità Esatti (%)**
10	100	2511.87	6
50	100	488.92	61
100	100	259.51	89
500	100	258	100
1000	100	258	100

Tabella 54: Risultati del problema con 8 luoghi, al variare della popolazione

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti (%)**
100	10	618.63	0
100	50	268.8	52
100	100	259.51	89
100	500	258	100
100	1000	258	100

Tabella 55: Risultati del problema con 8 luoghi, al variare del numero di generazioni

Popolazione	**Generazioni**	**Errore Medio**	**Probabilità Esatti (%)**
10	10	3469.75	0
50	50	406.36	36
100	100	260.25	86
500	500	258	100
1000	1000	258	100

Tabella 56: Risultati del problema con 8 luoghi, al variare della popolazione e del numero di generazioni

Anche con questa implementazione si riesce a trovare la soluzione ottima, anche se per essere paragonata alla precedente ha bisogno di un più alto numero di generazioni e/o popolazione.

Le tabelle 57, 58 e 59 riguardano il caso di $N = 12$, con variazione di popolazione e generazione. I parametri fissi considerati sono:

- probabilità mutazione = 0.1;
- probabilità crossover = 0.3;

Popolazione	Generazioni	Distanza Media	Probabilità Esatti (%)
10	100	5094.91	0
50	100	1426.70	0
100	100	691.33	1
500	100	362.83	2
1000	100	354.71	2

Tabella 57: Risultati del problema con 12 luoghi, al variare della popolazione

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
100	10	2684.75	0
100	50	851.13	0
100	100	734.64	0
100	500	720.34	22
100	1000	666.39	51

Tabella 58: Risultati del problema con 12 luoghi, al variare del numero di generazioni

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
10	10	6958.22	0
50	50	1743.43	0
100	100	706.69	0
500	500	317.86	33
1000	1000	301.96	69

Tabella 59: Risultati del problema con 12 luoghi, al variare della popolazione e del numero di generazioni

I punti sono distribuiti come indicati nelle tabella 50.

Anche in questo caso, generalmente, la ricerca della soluzione ottima è molto più complessa rispetto all'implementazione precedente.

Come ultima sessione di prove il numero di nodi è stato aumentato a sedici con punti dislocati come in tabella 29. I risultati sono mostrati nelle tabelle 60, 61 e 62, utilizzando come parametri variabili dimensione della popolazione e numero di generazioni e come parametri fissi:

- probabilità mutazione = 0.1;
- probabilità crossover = 0.3;

Dai risultati ottenuti si nota che, in condizioni di popolazioni e generazioni ridotte, i risultati sono molto peggiori rispetto all'implementazione precedente; invece, per grandi popolazioni e elevate generazioni, i risultati sono quasi equivalenti o addirittura migliori.

Esercizio 4

Lo scopo di questo esercizio è la risoluzione del gioco della formica, utilizzato nelle esercitazioni precedenti, tramite algoritmi genetici.

Popolazione	Generazioni	Distanza Media	Probabilità Esatti (%)
10	100	7782.26	0
50	100	3219.81	0
100	100	1614.73	0
500	100	607.44	0
1000	100	550.87	0

Tabella 60: Risultati del problema con 16 luoghi, al variare del numero di popolazione

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
100	10	5315.16	0
100	50	2101.14	0
100	100	1743.35	0
100	500	1253.51	0
100	1000	1043.3	1

Tabella 61: Risultati del problema con 16 luoghi, al variare del numero di generazioni

Popolazione	Generazioni	Errore Medio	Probabilità Esatti (%)
10	10	12526.2	0
50	50	3631.1	0
100	100	1575.29	0
500	500	498.78	1
1000	1000	434.67	5

Tabella 62: Risultati del problema con 16 luoghi, al variare del numero di popolazione e generazioni

In questo esercizio, il cromosoma, che rappresenta il percorso che effettua la formica, è un vettore di $2 * N$ sequenze (mosse) di 2 bit che rappresentano la direzione:

- 00 = nord
- 01 = est
- 10 = sud
- 11 = ovest

Nella sessione di test la dimensione della griglia è pari a 20, con un numero di mosse massimo pari a 40. Nella fig. 31 è illustrata la disposizione del cibo nella griglia.

Nelle tabelle 63, 64 e 65 sono indicati i risultati al variare di dimensione della popolazione e del numero di generazioni, considerando come parametri fissi:

- probabilità di mutazione = 0.1;
- probabilità di crossover = 0.3;

Nella tabella 66 viene invece analizzata la variazione delle probabilità di crossover e mutazione, considerando come parametri fissi:

Figura 31: Distribuzione del cibo sulla griglia 20x20

Popolazione	Generazioni	Percorso Migliore	Percorso Medio
10	100	5	0.72
50	100	5	1.58
100	100	6	1.81
500	100	6	2.77
1000	100	6	2.98

Tabella 63: Risultati del gioco della formica, al variare della popolazione

Popolazione	Generazioni	Percorso Migliore	Percorso Medio
100	10	3	-1.32
100	50	4	0.89
100	100	5	1.69
100	500	7	3.74
100	1000	7	4.57

Tabella 64: Risultati del gioco della formica, al variare del numero di generazioni

Popolazione	Generazioni	Percorso Migliore	Percorso Medio
10	10	1	-4.68
50	50	3	0.63
100	100	6	1.76
500	500	7	3.77
1000	1000	8	4.6

Tabella 65: Risultati del gioco della formica, al variare della popolazione e del numero di generazioni

- popolazione = 100;
- generazioni = 100;

Mutazione	Crossover	Percorso Migliore	Percorso Medio
0.01	0.01	8	5.86
0.01	0.1	8	5.51
0.01	0.5	9	5.59
0.1	0.01	5	2.27
0.5	0.01	3	0.4
0.1	0.1	5	2.02
0.5	0.5	4	0.32

Tabella 66: Risultati del gioco della formica, al variare di probabilità di mutazione e crossover

Dai risultati ottenuti si può notare che il valore del percorso aumenta in correlazione con l'aumento di iterazioni e dimensione della popolazione. I migliori risultati si ottengono impostando una probabilità di mutazione molto bassa.

Il percorso migliore trovato è illustrato nell'immagine 31.

Figura 32: Possibile percorso