**Università degli Studi del Molise**

****

**Facoltà di scienze MM. FF. NN.**

**Corso di Laurea in Informatica**

**Progetto di**

**Intelligenza Artificiale:**

**“Risoluzione di problemi NP-Completi**

**relativi alla teoria dei grafi**

**mediante algoritmi genetici ”**

|  |  |
| --- | --- |
| **Relatore:** | **Studenti:** |
| **Prof. Remo Pareschi** | **Gianandrea Giovanni**  **Lepore Antonio**  **Valente Lorenzo** |

**ANNO ACCADEMICO 2011/2012**

Sommario

**Algoritmo Genetico** **4**

Principi di funzionamento2

Dettaglio del funzionamento2

Crossover3

Single point crossover4

Mutazione4

**GeneticGraph** **5**

**Problema del commesso viaggiatore** **6**

Definizione6

Modello Matematico7

Complessità Computazionale7

NP-Completezza8

Struttura dell’Algoritmo Genetico8

**Massimo Matching9**

Definizione9

Struttura dell’Algoritmo Genetico10

**Massimo Set Indipendente** **11**

Definizione11

Struttura dell’Algoritmo Genetico11

**Minimo Sottoinsieme dei Vertici Dominanti13**

Definizione13

Struttura dell’Algoritmo Genetico13

**Minimo Set degli Archi Dominanti15**

Definizione15

Struttura dell’Algoritmo Genetico15

Algoritmo Genetico

L'**algoritmo genetico** è un [algoritmo](http://it.wikipedia.org/wiki/Algoritmo) di ottimizzazione e appartiene ad una particolare classe di algoritmi utilizzati in diversi campi, tra cui l'[intelligenza artificiale](http://it.wikipedia.org/wiki/Intelligenza_artificiale). È un metodo [euristico](http://it.wikipedia.org/wiki/Euristico) di ricerca ed ottimizzazione, ispirato al principio della [selezione naturale](http://it.wikipedia.org/wiki/Selezione_naturale) di [Charles Darwin](http://it.wikipedia.org/wiki/Charles_Darwin) che regola l'[evoluzione](http://it.wikipedia.org/wiki/Evoluzione) biologica.

Il nome deriva dal fatto che i suoi primi pionieri si ispirarono alla [natura](http://it.wikipedia.org/wiki/Natura) e alla [genetica](http://it.wikipedia.org/wiki/Genetica), branca della [biologia](http://it.wikipedia.org/wiki/Biologia).

Gli algoritmi genetici sono applicabili alla risoluzione di un'ampia varietà di problemi d'ottimizzazione non indicati per gli [algoritmi classici](http://it.wikipedia.org/w/index.php?title=Algoritmo_classico&action=edit&redlink=1), compresi quelli in cui la [funzione](http://it.wikipedia.org/wiki/Funzione_%28matematica%29) obiettivo è [discontinua](http://it.wikipedia.org/wiki/Funzione_continua), non [derivabile](http://it.wikipedia.org/wiki/Derivata), [stocastica](http://it.wikipedia.org/wiki/Processo_stocastico), o fortemente non lineare.

*Cosa hanno a che fare con l'intelligenza artificiale?*

**La teoria dell'evoluzione della specie può essere utilizzata per simulare al computer dei processi di affinamento progressivo delle potenziali soluzioni alle problematiche connesse ai sistemi complessi.**

Nel campo dell'intelligenza artificiale gli AG sono spesso usati per istruire le macchine in determinati problemi. Il metodo usato è quello dei sistemi classificatori: la macchina, il cui compito è quello di gestire in maniera ottimale un determinato sistema, riceve una serie di istruzioni del tipo if…then, li testa sul sistema in esame e a seconda delle performance misurate (che rappresentano pertanto la fitness dell'insieme di regole adottato), decide se tale sistema è appropriato o meno.

## Principi di funzionamento

Un tipico [algoritmo](http://it.wikipedia.org/wiki/Algoritmo) genetico parte da un certo numero di possibili soluzioni (individui) chiamate popolazione e provvede a farle evolvere nel corso dell'esecuzione: a ciascuna iterazione, esso opera una selezione di individui della popolazione corrente, impiegandoli per generare nuovi elementi della popolazione stessa, che andranno a sostituire un pari numero d'individui già presenti, e a costituire in tal modo una nuova popolazione per l'iterazione (o generazione) seguente. Tale successione di generazioni evolve verso una soluzione ottimale (locale o globale) del problema assegnato.

La loro evoluzione viene ottenuta attraverso una parziale ricombinazione delle soluzioni, ogni individuo trasmette parte del suo patrimonio genetico ai propri discendenti, e l'introduzione di [mutazioni](http://it.wikipedia.org/wiki/Mutazione_genetica) casuali nella popolazione di partenza, sporadicamente quindi nascono individui con caratteristiche non comprese tra quelle presenti nel corredo genetico della specie originaria. La mutazione serve, in genere, per inserire un po' di caos tra le soluzioni ed evitare che si cada in un minimo locale.

Finita la fase di evoluzione la popolazione delle soluzioni viene analizzata e vengono tenute solo le soluzioni che meglio risolvono il problema: gli individui con le qualità più adatte all'ambiente in cui si trovano hanno quindi maggiori possibilità di sopravvivere e riprodursi. Queste soluzioni subiranno una nuova fase di evoluzione e così via.

Alla fine ci si aspetta di trovare una popolazione di soluzioni che riescano a risolvere adeguatamente il problema posto. Non vi è modo di decidere a priori se l'algoritmo sarà effettivamente in grado di trovare una soluzione accettabile. Di norma gli algoritmi genetici vengono utilizzati per problemi di ottimizzazione per i quali non si conoscono algoritmi di [complessità](http://it.wikipedia.org/wiki/Teoria_della_complessit%C3%A0_computazionale) lineare o polinomiale.

Un caso particolare di applicazione di algoritmi genetici è [Acovea](http://it.wikipedia.org/w/index.php?title=Acovea&action=edit&redlink=1), un [software](http://it.wikipedia.org/wiki/Software) studiato per trovare il profilo migliore delle opzioni di ottimizzazione del [compilatore](http://it.wikipedia.org/wiki/Compilatore) [gcc](http://it.wikipedia.org/wiki/Gcc): un problema di elevata complessità.

## Dettaglio del funzionamento

La soluzione del problema viene codificata in una [struttura](http://it.wikipedia.org/wiki/Struttura_dati), di solito una [stringa](http://it.wikipedia.org/wiki/Stringa_%28formale%29), detta "[gene](http://it.wikipedia.org/wiki/Gene)".

Inizialmente viene creato un certo numero di geni in maniera [casuale](http://it.wikipedia.org/wiki/Probabilit%C3%A0) e si definisce una [funzione](http://it.wikipedia.org/wiki/Funzione_%28matematica%29) che restituisca la "bontà" di un gene come soluzione del problema, detta funzione di *fitness*.

L'algoritmo consiste nell'applicazione di operazioni, che tendono a modificare la [popolazione](http://it.wikipedia.org/wiki/Popolazione) dei geni, nel tentativo di migliorarli in modo da ottenere una soluzione sempre migliore.

L'evoluzione procede quindi in passi, per ognuno di questi viene per prima cosa eseguito un [ordinamento](http://it.wikipedia.org/wiki/Algoritmo_di_ordinamento) sui cromosomi sulla base del risultato della funzione di *fitness*. Vengono poi eseguite le operazioni su un numero di cromosomi stabilito dai parametri dell'algoritmo, che in generale determinano quanti di questi cromosomi devono subire *crossover* e [mutazioni](http://it.wikipedia.org/wiki/Mutazione), e in quale misura.

L'algoritmo evolve quindi attraverso i seguenti punti:

* generazione, in maniera casuale, di una popolazione iniziale;
* creazione di una sequenza di nuove popolazioni, o generazioni. In ciascuna iterazione, gli individui della popolazione corrente sono usati per creare la generazione successiva, e a questo scopo si compiono degli ulteriori passi:
  + ciascun membro della popolazione corrente è valutato calcolandone il rispettivo valore di [fitness](http://it.wikipedia.org/wiki/Fitness) (idoneità);
  + si determina un opportuno [ordinamento](http://it.wikipedia.org/wiki/Algoritmo_di_ordinamento) di tali individui sulla base dei valori di fitness;
  + gli individui più promettenti sono selezionati come genitori;
  + a partire da tali individui si genera un pari numero di individui della generazione successiva, e ciò può avvenire secondo due modalità distinte, vale a dire effettuando cambiamenti casuali su un singolo genitore (mutazione) oppure combinando opportunamente le caratteristiche di una coppia di genitori (incrocio);
  + gli individui così generati insieme ai genitori (la progenie) vengono selezionati,in base al loro valore nella funzione di fitness, per formare la generazione successiva;
* infine, l'algoritmo s'interrompe quando uno dei criteri d'arresto è soddisfatto.

## [Crossover](http://it.wikipedia.org/wiki/Crossing-over)

In base a un [coefficiente](http://it.wikipedia.org/wiki/Coefficiente) stabilito inizialmente, alcune parti dei geni risultati migliori vengono scambiate, nell'ipotesi che questo possa migliorare il risultato della funzione di fitness nel successivo "passo evolutivo".

## Single point crossover

Ci sono varie tecniche di crossing over. Una delle più semplice è la "single point crossing over" che consiste nel prendere due individui e tagliare le loro stringhe di codifica in un punto a caso. Si creano così due teste e due code. A questo punto si scambiano le teste e le code, ottenendo due nuovi geni. Il crossing over non è applicato sempre, ma con una probabilità p_c. Nel caso in cui non viene applicato i figli sono semplicemente le copie dei genitori.

Sperimentalmente si può vedere che il miglioramento diventa apprezzabile solo dopo un certo numero di passi. Questo a meno di casi fortunati, ovviamente.

## Mutazione

La mutazione consiste nella modifica casuale di alcune parti dei geni con valore di *fitness* più basso, in base a coefficienti definiti inizialmente. Queste modifiche puntano a migliorare il valore della funzione per il gene in questione.

In realtà non è corretto pensare di mutare solo i cromosomi con fitness più bassa; al fine di garantire una maggiore capacità esplorativa dell'algoritmo (e non finire in "buche" di ottimo locale) sono ritenute utili anche le mutazioni di cromosomi con valore di fitness alto. In definitiva le mutazioni servono soprattutto a esplorare lo spazio di ricerca, e non hanno quindi scopo migliorativo, anche perché vengono effettuate in modo probabilistico e non deterministico e quindi ogni cromosoma potrebbe essere mutato in modo diverso.

GeneticGraph

GeneticGraph è un software concepito per la risoluzione di cinque problemi NP-completi sulla teoria dei grafi mediante algoritmi genetici.

Il programma è stato sviluppato con il linguaggio di programmazione Java utilizzando la libreria JGAP. JGAP offre i meccanismi di base per la strutturazione di algoritmi genetici, consentendo la modellazione di geni, funzioni di fitness, procedure di mutazione e di cross-over.

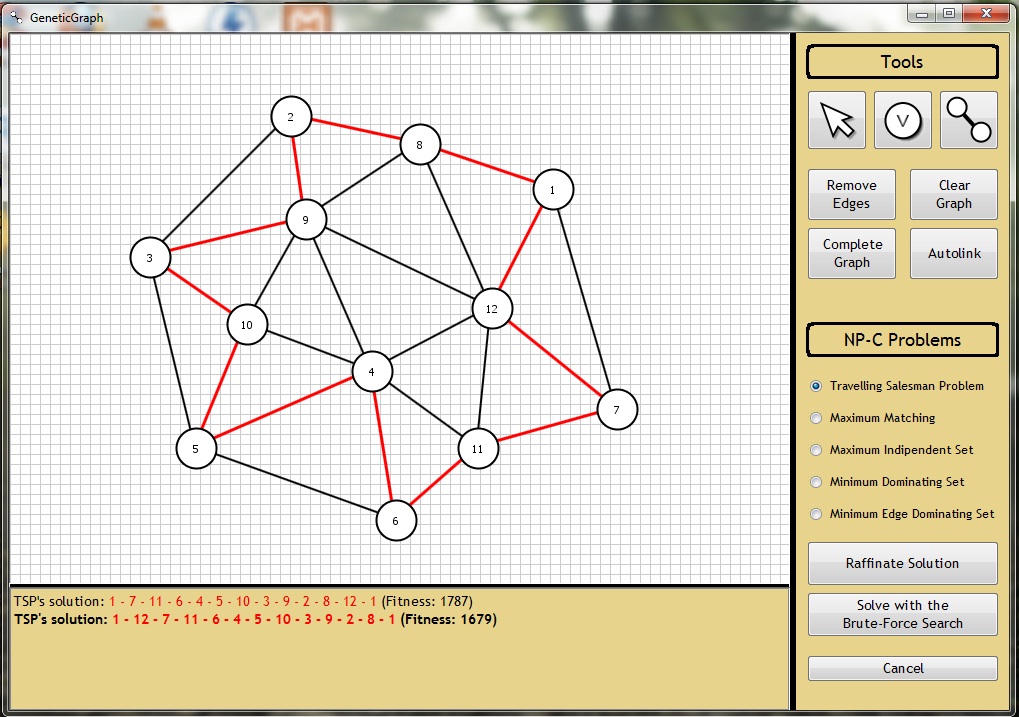
Il programma è dotato di un’interfaccia grafica semplice e intuitiva e di strumenti che permettono di realizzare semplici grafi con pochi click. Inoltre sono presenti dei comodi pulsanti per l’autocompletamento o l’eliminazione di nodi e archi. I pesi degli archi corrispondono alla distanza geometrica tra i nodi collegati.

Una volta costruito il grafo, è possibile trovare la soluzione a cinque diversi problemi NP-Completi:

1. Problema del Commesso Viaggiatore;
2. Problema del Massimo Set Indipendente di Archi (Massimo Matching);
3. Problema del Massimo Set Indipendente di Nodi
4. Problema del Minimo Set Dominante di Nodi
5. Problema del Minimo Set Dominante di Archi

Le soluzioni a tali problemi possono essere trovate tramite un algoritmo genetico costruito ad-hoc oppure tramite ricerca a forza bruta. Nel primo caso, si otterrà una soluzione non ottimale ma in tempi rapidi mentre nel secondo caso si otterrà la soluzione ottimale ma in tempi variabili esponenzialmente. Le soluzioni ottenute tramite algoritmi genetici possono essere raffinate successivamente tramite riesecuzione dell’algoritmo stesso.

Problema del commesso viaggiatore



## Definizione

Il **problema del commesso viaggiatore**, o **TSP** (dall'[inglese](http://it.wikipedia.org/wiki/Lingua_inglese) *Travelling Salesman Problem*) è un problema di [teoria dei grafi](http://it.wikipedia.org/wiki/Teoria_dei_grafi), uno dei casi di studio tipici dell'[informatica](http://it.wikipedia.org/wiki/Informatica) teorica e della [teoria della complessità](http://it.wikipedia.org/wiki/Teoria_della_complessit%C3%A0_algoritmica).

Il nome nasce dalla sua più tipica rappresentazione: data una rete di città, connesse tramite delle strade, trovare il percorso di minore lunghezza che un commesso viaggiatore deve seguire per visitare tutte le città una e una sola volta per poi tornare alla città di partenza.

Espresso nei termini della teoria dei grafi è così formulato: dato un [grafo completo](http://it.wikipedia.org/wiki/Grafo_completo) pesato, trovare il cammino di costo minore visitando tutti i nodi una sola volta e tornando al nodo di partenza. La rete di città può essere rappresentata come un [grafo](http://it.wikipedia.org/wiki/Grafo) in cui le città sono i nodi, le strade gli [archi](http://it.wikipedia.org/wiki/Arco_%28teoria_dei_grafi%29) e le distanze i pesi sugli archi. La sostanziale differenza rispetto al [Ciclo Hamiltoniano](http://it.wikipedia.org/wiki/Cammino_hamiltoniano) si trova nel fatto che quest'ultimo viene formulato su di un grafo privo di pesi.

Può essere dimostrato che specificare o meno il ritorno alla città di partenza non cambia la [complessità computazionale](http://it.wikipedia.org/wiki/Teoria_della_complessit%C3%A0_algoritmica) del problema.

Il problema è di considerevole importanza pratica, al di là delle ovvie applicazioni nella [logistica](http://it.wikipedia.org/wiki/Logistica) e nei [trasporti](http://it.wikipedia.org/wiki/Trasporti). Un esempio classico è la costruzione di [circuiti stampati](http://it.wikipedia.org/wiki/Circuito_stampato), nella pianificazione del percorso del trapano per creare i fori nella piastra. Nelle applicazioni di foratura o di rifinitura automatica eseguite da [robot](http://it.wikipedia.org/wiki/Robot), le "città" sono pezzi da rifinire o fori (anche di varie dimensioni) da praticare, e il "costo di viaggio" include i tempi morti (ad esempio il tempo che il robot impiega, se necessario, per cambiare la punta con cui lavora).

## Modello Matematico

Al problema del TSP è associabile un grafo G=(V,A), in cui V è l'insieme degli *n* nodi (o città) e A è l'insieme degli archi (o strade). Si indica con il costo dell'arco per andare dal nodo *i* al nodo *j*. Il TSP è simmetrico se , altrimenti si dice asimmetrico. A seconda che il problema sia simmetrico o asimmetrico, esso può essere descritto con modelli matematici differenti. Detta la generica variabile binaria tale che se l'arco appartiene al circuito e altrimenti, nel caso più generale di problema asimmetrico, una possibile formulazione matematica del problema è:

Sottoposto a

La relazione (1) è la funzione obiettivo che rappresenta la minimizzazione del costo del cammino. I vincoli (2) e (3) indicano che in ogni nodo i entra ed esce un solo arco, e sono detti vincoli di assegnazione. Poiché tali vincoli non assicurano una soluzione costituita da un unico circuito vengono inseriti anche i vincoli (4) che assicurano l'assenza di sottocircuiti. In particolare essi definiscono che, comunque si scelga un sottoinsieme di nodi Q in V, deve esistere almeno un arco che colleghi un nodo di Q con un nodo non appartenente a Q.

## Complessità computazionale

Non esistono [algoritmi](http://it.wikipedia.org/wiki/Algoritmo) efficienti per la risoluzione del TSP, l'unico metodo di risoluzione è rappresentato dall'[enumerazione](http://it.wikipedia.org/wiki/Enumerazione_%28matematica%29) totale, ovvero nell'elaborazione di tutti i possibili cammini sul grafo per la successiva scelta di quello migliore. Tuttavia, la complessità dell'operazione la rende impraticabile per grafi di dimensioni comuni nei problemi reali: in un grafo di n nodi, bisognerà calcolare, nel caso peggiore in cui ogni nodo è connesso con tutti gli altri, n! (n [fattoriale](http://it.wikipedia.org/wiki/Fattoriale)) possibili cammini, il che implica una complessità [esponenziale](http://it.wikipedia.org/wiki/Esponenziale) (in base all'[approssimazione di Stirling](http://it.wikipedia.org/wiki/Approssimazione_di_Stirling)). Il TSP rappresenta inoltre un esempio di problema di [programmazione lineare intera](http://it.wikipedia.org/wiki/Ricerca_operativa#Programmazione_lineare_intera) nel quale risulta pressoché inattuabile ottenere valutazioni approssimate tramite la tecnica del [rilassamento continuo](http://it.wikipedia.org/wiki/Ricerca_operativa#Rilassamenti_ed_euristiche) poiché il problema di PL risultante si troverebbe ad avere un numero di vincoli che cresce in modo esponenziale con i nodi, rendendo così intrattabili le matrici associate ad esso. Per scopi pratici, il metodo della ricottura simulata ([simulated annealing](http://it.wikipedia.org/wiki/Simulated_annealing" \o "Simulated annealing)) ha effettivamente risolto il TSP.  
Una rete di mille nodi, molto più comune di quanto si possa pensare, comincerebbe già a creare seri problemi computazionali.

## NP-completezza

È stato dimostrato che TSP è un problema [NP-difficile](http://it.wikipedia.org/wiki/Teoria_della_complessit%C3%A0_algoritmica) (più precisamente, è [NP-completo](http://it.wikipedia.org/wiki/Classi_di_complessit%C3%A0_P_e_NP) per la [classe di complessità](http://it.wikipedia.org/wiki/Classe_di_complessit%C3%A0) [FPNP](http://it.wikipedia.org/wiki/Glossario_delle_classi_di_complessit%C3%A0); v. [problema di funzione](http://it.wikipedia.org/w/index.php?title=Problema_di_funzione&action=edit&redlink=1)), e la versione [decisionale](http://it.wikipedia.org/w/index.php?title=Problema_decisionale&action=edit&redlink=1) del problema ("dati i pesi e un numero x, decidere se ci sia una soluzione migliore di x") è NP-completa.

Il problema rimane NP-difficile anche in molte sue versioni ridotte, come nel caso in cui le città siano in un piano con [distanze euclidee](http://it.wikipedia.org/wiki/Distanza). Inoltre, omettere la condizione di visitare una città "una e una sola volta" non rimuove la NP-difficoltà, poiché si nota facilmente che nel caso piano un cammino ottimale visiterebbe comunque le città una volta sola.

Anche il [problema del commesso viaggiatore per collo di bottiglia](http://it.wikipedia.org/w/index.php?title=Problema_del_commesso_viaggiatore_per_collo_di_bottiglia&action=edit&redlink=1) è NP-difficile.

## Struttura dell’algoritmo genetico

Per prima cosa viene creato un individuo modello formato da N geni di tipo intero dove N è il numero di vertici del grafo. Ogni gene ha come allele il numero corrispondente all’ i-esimo vertice.

Viene poi creata una popolazione di 500 individui clonati e randomizzati a partire dall’individuo modello. Il primo gene di ogni individuo rimane lo stesso (poiché indica il vertice sorgente).

Si fa evolvere la popolazione per 100 generazioni, dopodiché viene restituito l’individuo migliore (col fitness più basso).

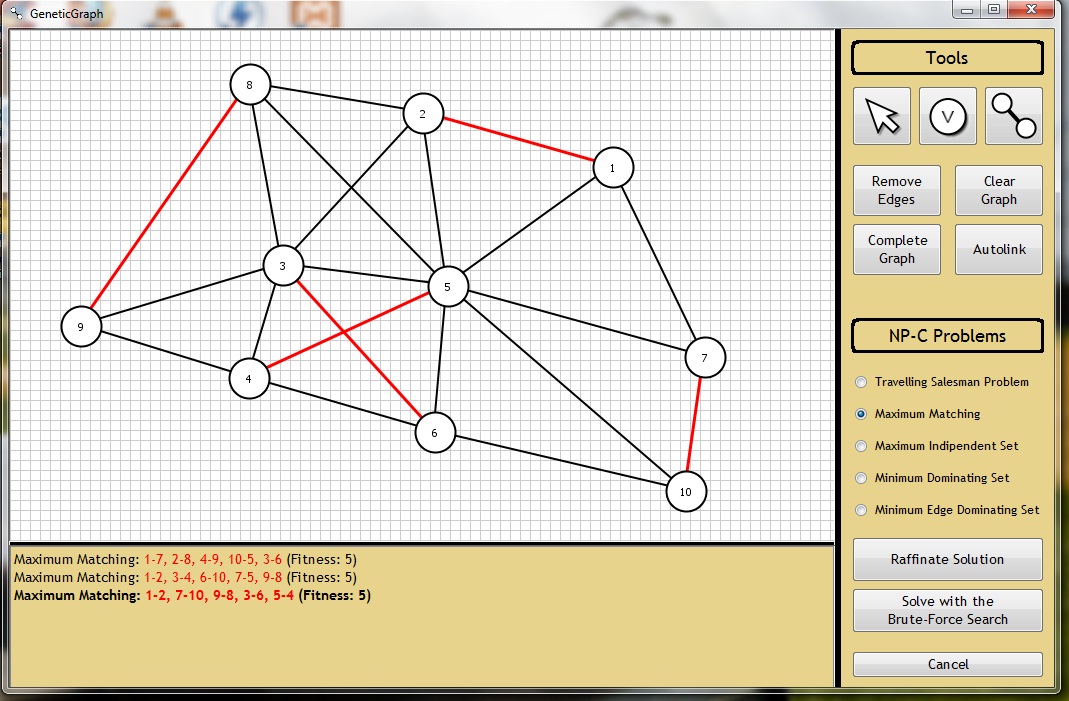
Gli operatori che agiscono durante l’evoluzione sono:

* Una mutazione di tipo **swapping** che si occupa di mescolare i geni, applicata al 50% della popolazione;
* Un cross over di **tipo greedy** che si occupa di generare nuovi individui.

La funzione di fitness si occupa di valutare la bontà di una soluzione. Una generica soluzione *S* avrà come funzione di fitness:

dove calcola la distanza geometrica tra il vertice e il vertice .

Massimo Matching



## Definizione

Dato un grafo *G* = (*V*,*E*), un **matching** M in G è un sottoinsieme di archi non adiacenti; ovvero non esistono due archi appartenenti al matching che condividono lo stesso vertice.

[Maximal-matching.svg](http://en.wikipedia.org/wiki/File:Maximal-matching.svg)

Si definisce **massimo matching** M di un grafo G il matching che contiene il più grande numero possibile di archi.

In altre parole, M è un massimo matching se:

* M è un matching
* Aggiungendo in M un qualsiasi arco non appartenente ad M, M non risulta più essere un matching.

Il massimo matching non è sottoinsieme di nessun altro possibile matching di G.

[Maximum-matching-labels.svg](http://en.wikipedia.org/wiki/File:Maximum-matching-labels.svg)

In un grafo possono esserci diversi massimi matching. Si definisce **matching number** E(G) di un grafo G il numero di archi del massimo matching.

Si definisce **matching perfetto** (o **matching completo**) un matching i cui archi toccano tutti i vertici del grafo, ovvero, ogni vertice del grafo risulta essere incidente ad esattamente un arco del matching.

Si definisce **matching quasi perfetto** un matching dove un solo vertice non risulta essere incidente con nessun arco del matching. I matching quasi perfetti possono sussistere solamente in grafi con numero dispari di vertici.

## Struttura dell’algoritmo genetico

Per prima cosa viene creato un individuo modello formato da N geni di tipo booleano dove N è il numero di archi del grafo. Ogni gene viene inizialmente impostato con il valore false.

Viene poi creata una popolazione di 500 individui clonati e randomizzati a partire dall’individuo modello.

Si fa evolvere la popolazione per 100 generazioni, dopodiché viene restituito l’individuo migliore (col fitness più alto).

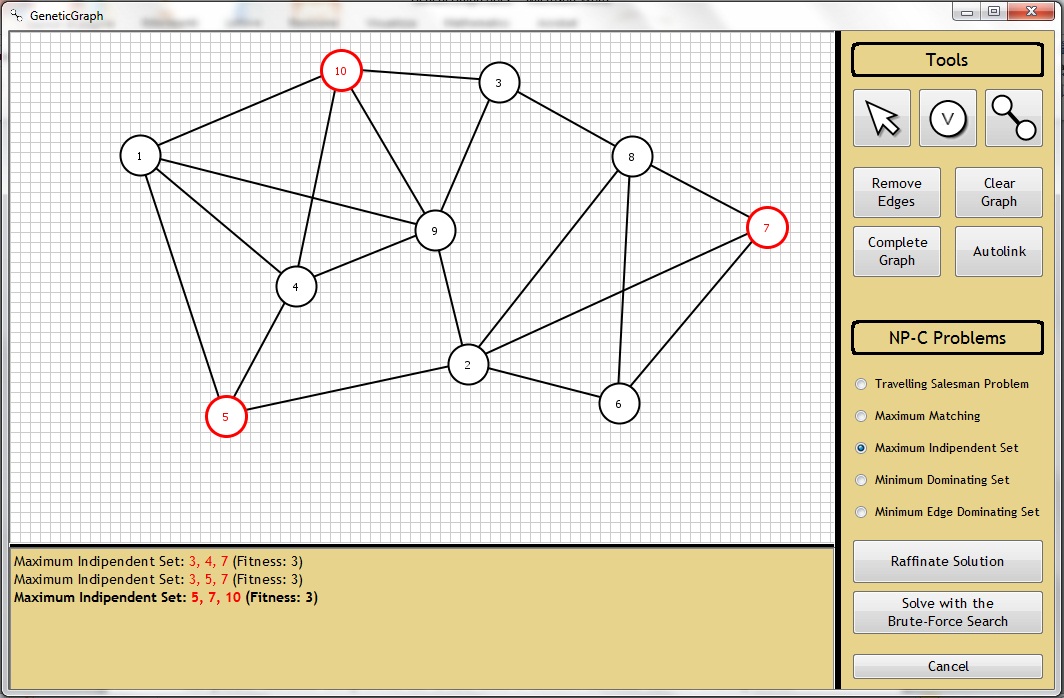
Gli operatori che agiscono durante l’evoluzione sono:

* Una mutazione di tipo **standard** che si occupa di creare nuovi geni vicini ai vecchi, applicata al 10% della popolazione;
* Un cross over di **tipo standard** che si occupa di generare nuovi individui, applicata al 35% della popolazione.

La funzione di fitness si occupa di valutare la bontà di una soluzione. Una generica soluzione *S* avrà come funzione di fitness:

dove calcola il **numero di archi** della soluzione .

Massimo Set Indipendente



## Definizione

Dato un grafo G=(V,E) si definisce **set indipendente** un sottoinsieme S dei vertici V tali che non esista nessuna coppia di vertici adiacenti in S.

Si definisce **massimo set indipendente** (o **massimo set stabile**) un set indipendente che non è sottoinsieme di nessun altro set indipendente. In altre parole, S è un set indipendente se nessun arco del grafo collega due vertici appartenenti S ed ogni vertice non appartenente ad S ha almeno un vertice adiacente in S.

## Struttura dell’algoritmo genetico

Per prima cosa viene creato un individuo modello formato da N geni di tipo booleano dove N è il numero di vertici del grafo. Ogni gene viene inizialmente impostato con il valore false.

Viene poi creata una popolazione di 500 individui clonati e randomizzati a partire dall’individuo modello.

Si fa evolvere la popolazione per 100 generazioni, dopodiché viene restituito l’individuo migliore (col fitness più alto).

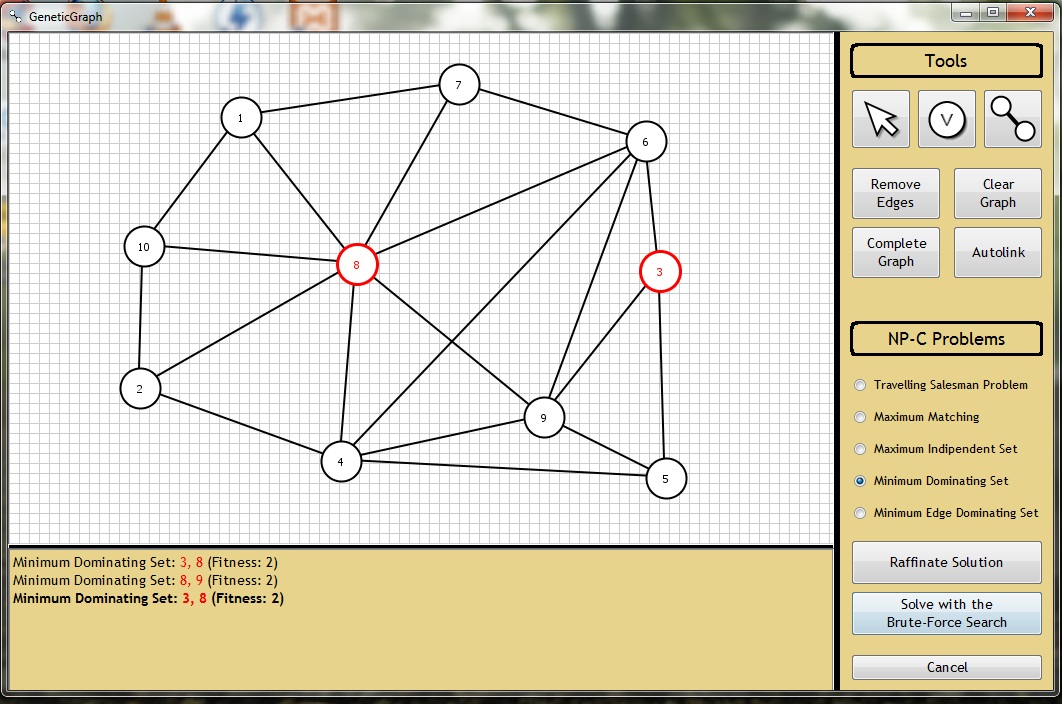
Gli operatori che agiscono durante l’evoluzione sono:

* Una mutazione di tipo **standard** che si occupa di creare nuovi geni vicini ai vecchi, applicata al 10% della popolazione;
* Un cross over di **tipo standard** che si occupa di generare nuovi individui, applicata al 35% della popolazione.

La funzione di fitness si occupa di valutare la bontà di una soluzione. Una generica soluzione *S* avrà come funzione di fitness:

dove calcola il **numero di vertici** della soluzione .

Minimo Sottoinsieme dei Vertici Dominanti



## Definizione

Una **copertura tramite vertici** di un [grafo](http://it.wikipedia.org/wiki/Grafo) non orientato G = (V, E)è un sottoinsieme Sdei suoi vertici tale che ogni arco ha almeno un'estremità in S.

Il **problema di copertura tramite vertici** è il problema di [ottimizzazione combinatoria](http://it.wikipedia.org/w/index.php?title=Ottimizzazione_combinatoria&action=edit&redlink=1) di trovare la più piccola copertura tramite vertici in un grafo.

La copertura tramite vertici è strettamente correlata al [massimo set indipendente](http://it.wikipedia.org/w/index.php?title=Massimo_insieme_indipendente&action=edit&redlink=1). Un insieme di vertici è una copertura tramite vertici se e solo se il suo complemento  \bar S = V \setminus Sè un insieme indipendente. Segue che un grafo con nvertici avente una copertura tramite vertici di cardinalità kse e solo se il grafo ha in insieme indipendente di cardinalità n-k. In questo senso, i due problemi sono duali.

## Struttura dell’algoritmo genetico

Per prima cosa viene creato un individuo modello formato da N geni di tipo booleano dove N è il numero di vertici del grafo. Ogni gene viene inizialmente impostato con il valore false.

Viene poi creata una popolazione di 500 individui clonati e randomizzati a partire dall’individuo modello.

Si fa evolvere la popolazione per 100 generazioni, dopodiché viene restituito l’individuo migliore (col fitness più basso).

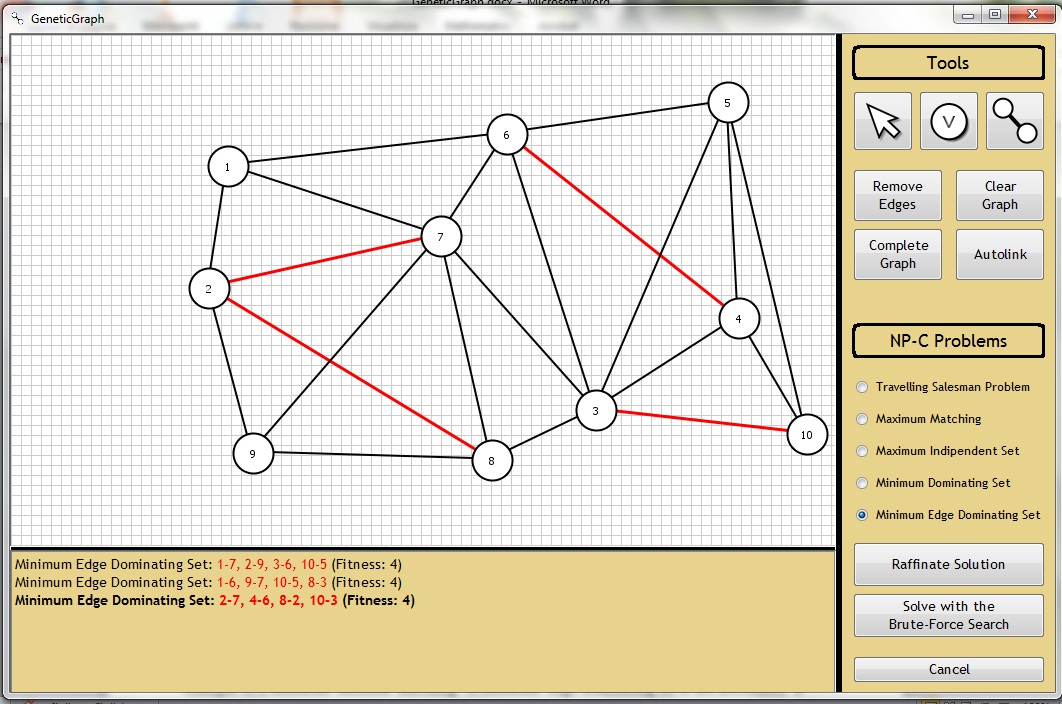
Gli operatori che agiscono durante l’evoluzione sono:

* Una mutazione di tipo **standard** che si occupa di creare nuovi geni vicini ai vecchi, applicata al 10% della popolazione;
* Un cross over di **tipo standard** che si occupa di generare nuovi individui, applicata al 35% della popolazione.

La funzione di fitness si occupa di valutare la bontà di una soluzione. Una generica soluzione *S* avrà come funzione di fitness:

dove calcola il **numero di vertici** della soluzione .

Minimo Set degli Archi Dominanti



## Definizione

Dato un grafo G=(V,E) si definisce **set degli archi dominanti** (o **set delle linee dominanti**) un sottoinsieme D degli archi E tali che ogni arco non appartenente a D sia adiacente ad almeno un arco appartenente a D

Si definisce **minimo set degli archi dominanti** (o **minimo set delle linee dominanti**) di un grafo G, il più piccolo set degli archi dominanti che è possibile trovare in G.

## Struttura dell’algoritmo genetico

Per prima cosa viene creato un individuo modello formato da N geni di tipo booleano dove N è il numero di archi del grafo. Ogni gene viene inizialmente impostato con il valore false.

Viene poi creata una popolazione di 500 individui clonati e randomizzati a partire dall’individuo modello.

Si fa evolvere la popolazione per 100 generazioni, dopodiché viene restituito l’individuo migliore (col fitness più basso).

Gli operatori che agiscono durante l’evoluzione sono:

* Una mutazione di tipo **standard** che si occupa di creare nuovi geni vicini ai vecchi, applicata al 10% della popolazione;
* Un cross over di **tipo standard** che si occupa di generare nuovi individui, applicata al 35% della popolazione.

La funzione di fitness si occupa di valutare la bontà di una soluzione. Una generica soluzione *S* avrà come funzione di fitness:

dove calcola il **numero di archi** della soluzione .