UNIVERSITÀ DEGLI STUDI FIRENZE

7012348
Alessandro Ugolini

Relazione elaborato Intelligenza Artificiale

L'obiettivo:

Ricerca locale per problemi di sodisfacimento di vincoli

In questo esercizio si implementa una strategia di ricerca locale basata sull'euristica minconflicts descritta in in R&N 2009 6.4. L'algoritmo può essere implementato in un linguaggio di programmazione a scelta. La base del codice dovrebbe essere sufficientemente generale da permettere di risolvere un problema generico ma le descrizioni del dominio e dei vincoli possono essere direttamente codificate nel programma (a differenza di quanto accadrebbe con un risolutore general purpose che sarebbe tipicamente in grado di leggere la specifica del CSP in un linguaggio formale). Il codice sviluppato dovrà essere testato su due problemi giocattolo: n-queens e map-coloring (per quest'ultimo è facile generare problemi casuali seguendo la strategia suggerita nell'esercizio 6.10 in R&N 2009). Si studi empiricamente il tempo di esecuzione del programma realizzato in funzione del numero di variabili n del problema, facendo crescere n quanto più possibile (nei limiti ragionevoli imposti dall'hardware disponibile).

L'euristica MinConflicts

L'algoritmo implementato e studiato per la risoluzione dei problemi CSP è l'algoritmo di ricerca locale basato sull'euristica MinConflicts, il cui comportamento può essere facilmente rappresentato tramite il seguente pseudocodice.

function MIN-CONFLICTS(csp, max_steps) **returns** a solution or failure
  inputs: csp, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

  $current \leftarrow$ an initial complete assignment for csp
  for i = 1 to max_steps **do**
    if $current$ is a solution for csp **then return** $current$
    $var \leftarrow$ a randomly chosen conflicted variable from csp.VARIABLES
    $value \leftarrow$ the value v for var that minimizes CONFLICTS(var, v, $current$, csp)
    set $var = value$ in $current$
  return *failure*

Analisi dell'algoritmo a priori

Per questo suo comportamento l'algoritmo di risoluzione agisce come un algoritmo di ricerca locale, (molto simile a Hill-Climbing) e proprio per questo motivo soffre delle stesse problematiche e gode degli stessi pregi di questo tipo di algoritmi .

Pregi :

-Se l'algoritmo converge a una soluzione ottima quest'ultima viene raggiunta in un numero molto piccolo di passi.

-Numero di passi necessari a trovare una soluzione bassi e generalmente indipendenti dalla dimensione del problema

Svantaggi

-Come la maggior parte degli algoritmi di ricerca locale, rischio di bloccarsi in un plateau o in un punto di minimo locale (rischio di incompletezza).
-Se si incappa in una "configurazione plateau/minimo locale" indipendentemente dal numero di passi effettuati non si riesce a raggiungere la soluzione, nonostante si sia spesso molto vicini a essa (dagli 1 ai 10 conflitti in genere).

Date le seguenti caratteristiche e data la natura dei problemi CSP (spesso caratterizzati da non una unica soluzione al problema, ma da molteplici soluzioni "geometricamente simmetriche"), l'algoritmo si presta bene alla soluzione di questi problemi, inoltre esistono diverse tecniche che permettono di ovviare agli svantaggi della tecnica utilizzati.
Alcuni esempi di queste tecniche sono il simulated annealing e il random restart.

Sviluppo del programma

Il primo problema da risolvere è stato la costruzione di classi e funzioni abbastanza generali da poter rappresentare un qualsiasi CSP a dominio finito, a questo scopo sono state quindi designate 3 classi:
-la classe Variabile, contenente al suo interno un attributo identificatore, un attributo che rappresentasse il valore assunto dalla variabile e un attributo che rappresentasse il suo dominio;
-la classe Vincolo, contente al suo interno un set di variabili su cui il vincolo è definito e la condizione (da esprimere come funzione lambda) che permette di definire se il vincolo è soddisfatto o meno;
-infine la classe Problema costituita da un insieme di Variabili e un insieme di Vincoli su queste variabili.

(Ogni classe comprende inoltre al suo interno alcuni metodi ausiliari per la modellazione del problema, come funzionalità per l'aggiunta di variabili e vincoli al problema, per il calcolo del numero di conflitti nella configurazione attuale e per la stampa dell' attuale stato del problema.)

Una volta definite queste classi è stato poi possibile andare a implementare il risolutore del problema secondo l'euristica MinConflicts.
Da notare che l'algoritmo MinConflicts di base non assicura la risoluzione del problema, di conseguenza nelle situazioni in cui si testa il suo funzionamento questo deve essere eseguito più volte seguendo una tecnica di "random restart" che permetta così di ovviare alla naturale incompletezza dell'algoritmo risolutivo.
Definito l'algoritmo la fase successiva è stata quella di andare a definire due costruttori di problemi CSP su cui testare MinConflicts, ovvero il problema delle n-regine e il problema della colorazione di un grafo.

N QUEENS

Per il problema delle n-regine la costruzione è stata abbastanza immediata, questo tipo di problema infatti è facilmente costruibile a partire dal numero di regine su cui è definito: una variabile del problema andrà a rappresentare una generica delle n colonne della scacchiera che potrà assumere un valore compreso tra 1 e n (la casella all'interno della colonna nel

quale la regina si trova) e dovrà essere vincolata ai valori assunti da tutte le altre variabili (nessun valore uguale e nessun valore uguale sulla diagonale).

MAP COLORING

Per il problema del map coloring invece la costruzione è più complessa in quanto questo problema non ha una "struttura fissa" dipendente dalla dimensione del problema, infatti la costruzione in questo caso prevede prima la generazione di un grafo randomico con n nodi. Il metodo implementato è quello descritto dalla seguente procedura:

"Generate random instances of map-coloring problems as follows: scatter n points on the unit square; select a point X at random, connect X by a straight line to the nearest point Y such that X is not already connected to Y and the line crosses no other line; repeat the previous step until no more connections are possible. The points represent regions on the map and the lines connect neighbors "

(esercizio 6.10 in R&N 2009)

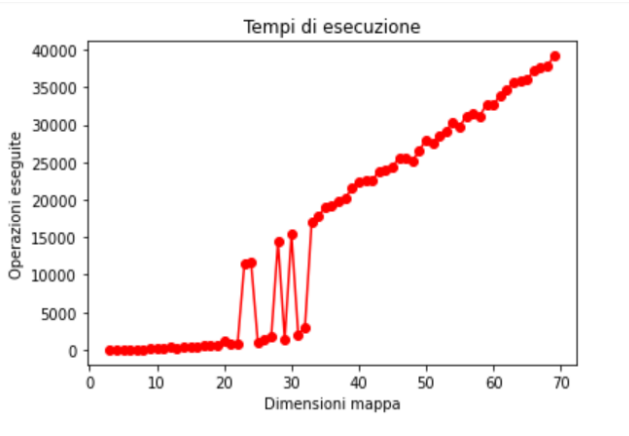
Una volta individuato il grafo la costruzione però è abbastanza immediata, a ogni nodo si associa una variabile, che potrà assumere un valore tra 1 e k (rappresentanti i k colori a disposizione) e si inserisce un vincolo per ogni arco che lega due nodi che specificherà il vincolo di assumere 2 valori (colori) diversi se i nodi sono adiacenti.

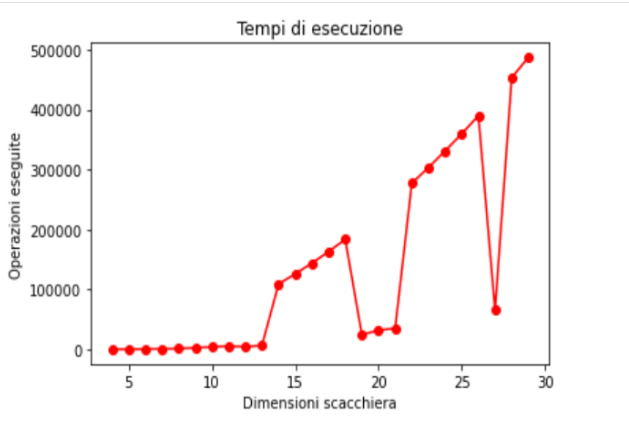
Ho testato poi i metodi di costruzione dei problemi precedentemente citati, e il metodo di risoluzione su una singola istanza dei problemi.

L'ultimo passo è stato lo studio dell'algoritmo e del numero di operazioni eseguite da quest'ultimo all'aumentare della dimensione dei problemi.

Dopo aver individuato la dimensione massima dei due problemi sostenibile dall'hardware (n=30 per le n regine e n=70 per il map coloring), ho provveduto poi a iterare la procedura di creazione di un problema di dimensione n e la sua successiva risoluzione andando a memorizzare in 2 coppie di vettori la dimensione del problema e il tempo necessario a risolverlo. Infine la relazione tra le dimensioni e il tempo di esecuzione è stata rappresentata su un grafico per poter meglio visualizzare il rapporto tra queste due grandezze, ottenendo i seguenti risultati.

Analisi dei risultati





Dai grafici rappresentati i risultati dei due problemi al crescere di n si possono intuire diverse informazioni.
Come si nota per entrambi i problemi il grafico è come suddiviso in 2 parti con andamento indipendente, la parte più bassa rappresenta le operazioni effettuate dall'algoritmo al crescere di n nel momento in cui questo trova una soluzione.
Come discusso durante la formulazione a priori delle caratteristiche del problema il numero di operazioni necessarie alla risoluzione del problema risulta essere molto basso (sotto il migliaio di operazioni) anche per problemi di dimensioni molto grandi, quando l'algoritmo riesce ad arrivare una soluzione.
Nella parte alta del grafo invece sono racchiusi i punti rappresentanti i casi in cui l'euristica minconflict non è riuscita a individuare una soluzione al problema, avvenimento che non è causato nè dalla dimensione eccessiva del problema nè dal numero di passi troppo basso, ma solo dalla natura intrinseca dell'algoritmo (che di base è incompleto).
Il numero di operazioni necessarie alla risoluzione per entrambi gli algoritmi risulta essere dipendente da 2 fattori: il numero di steps eseguiti dall'algoritmo e il numero di vincoli presenti nel problema.
Come discusso in precedenza gli steps necessari a trovare una soluzione al problema sono praticamente indipendenti dalla sua dimensione (lo si denota dall'andamento quasi costante della parte inferiore del grafico) qualora questa venga trovata.
Nella parte superiore invece le operazioni risultano essere invece dipendenti dalla dimensione del problema, questo perchè l'algoritmo andrà a effettuare un numero di operazioni che cresce come un O (max-steps*n_vincoli), essendo l'operazione di verifica di tutti i vincoli la più onerosa all'interno dell'algoritmo.
L'andamento dei due grafici permette inoltre di capire come il numero di operazioni nel caso peggiore dipenda dalla dimensione in maniera lineare per map coloring (distribuzione su una retta) e quadratica per N Queens (distribuzione su un ramo di parabola).
I dati sono quindi perfettamente coerenti con la natura dei problemi : N Queens ha sempre un numero di vincoli pari a 3*N*(N-1), mentre il map coloring di N ha in media un numero di vincoli O(c*N).