Esercizio per l'esame di Intelligenza Artificiale del 25/01/21

Studente: Marco Maimonte

1 Descrizione del metodo utilizzato

L'algoritmo oggetto di questo esercizio è chiamato *simulated annealing*, in riferimento al processo di ricottura dei materiali al fine di studiarne alcune proprietà (si veda [1], [3]). È un algoritmo di ricerca locale che cerca di porre rimedio al problema dei minimi locali e degli altipiani incontrato da *hill climbing* accettando non solo azioni favorevoli, ma anche azioni che fanno aumentare il punteggio dello stato corrente, che in analogia al processo fisico di ricottura viene indicata come **energia**. Ciò avviene sulla base del valore di un parametro di controllo T che, sempre per analogia, è chiamato **temperatura**, che varia con il tempo secondo una funzione dello stesso chiamata *schedule*. La probabilità con cui una transizione viene accettata anche in caso di un aumento dell'energia di $\Delta E \in \mathbb{R}_0^+$ è data dalla distribuzione di Boltzmann i.e.:

$$P(T) = \frac{1}{z} e^{\frac{-\Delta E}{k_B T}},$$

dove $\frac{1}{z} \in \mathbb{R}$ è un normalizzatore e k_B è una costante legata al tipo di materiale.

L'annealing fisico termina quando il materiale ha raggiunto un punto di equilibrio a una data temperatura. Per l'annealing simulato il punto di equilibrio lo si ha quando $P(T) \to 0$, ossia per $T \to 0$. Si dichiara quindi terminata la procedura una volta che la temperatura raggiungerà lo 0. Visto inoltre che l'annealing è simulato, la scelta di k_B è del tutto arbitraria, per cui si pone $k_B := 1$.

Dato che la distribuzione viene valutata dal metodo solo quando $\Delta E \geq 0$, è chiaro che questa restituisca un valore compreso nell'intervallo $[0; 1]$, per cui in questo caso si avrà la seguente distribuzione di probabilità:

$$P(T) = e^{\frac{-\Delta E}{T}} \quad \Delta E \geq 0$$

Da questa legge si deduce che per valori alti di T relativamente a δE si accetteranno anche transizioni sfavorevoli con probabilità prossime ad 1; via via che la *schedule* abbassa la temperatura $P(T)$ decrescerà esponenzialmente, causando sempre meno aumenti di energia, fino ad arrivare ad un punto in cui sarà pressoché impossibile accettare ulteriori aumenti. Da qui in poi la soluzione si stabilizzerà in un minimo locale che, sperabilmente, sarà anche il globale o comunque prossimo ad esserlo.

Detto questo, appare quindi evidente che la scelta della funzione *schedule* e della temperatura iniziale t_0 siano fattori fondamentali per individuare una soluzione ottima. In questo esercizio sono stati implementate 3 diverse *schedule*:

- logaritmica:

  $$\text{schedule}(t) = \frac{t_0}{\log(t + \sigma)}, \quad \sigma \in \mathbb{R}^+$$

  Come dimostrato in [7] che per $t \to \infty$ converge alla soluzione ottima, sebbene molto lentamente;

- lineare:

  $$\text{schedule}(t) = t_0 - ct$$

  dove $c \in \mathbb{R}^+$. Più rapida della *schedule* logaritmica, ma non ci sono garanzie di convergenza. È l'unica *schedule* che può portare a 0 la temperatura;

- geometrica:

  $$\text{schedule}(t) = \alpha^t t_0$$

  dove $\alpha \in (0, 1)$. La schedule più veloce, ma è anche quella che potrebbe produrre soluzioni più lontane dall'ottimo.

Generalmente t_0 deve essere scelto in maniera tale che nella fase iniziale del metodo possano essere accettate anche mosse che peggiorano "abbastanza" l'energia dello stato corrente, in modo da non iniziare subito a discendere un minimo qualunque. Purtroppo cosa significhi "abbastanza" a priori è molto complicato, per cui l'opzione migliore è determinarlo empiricamente.

Ad ogni iterazione, il metodo seleziona casualmente una mossa tra quelle disponibili ed effettua la transizione secondo le modalità sopradescritte. Poiché deve tenere conto dello stato corrente e di quello appena generato,

l'algoritmo ha un'occupazione di memoria costante. Tale occupazione potrebbe essere talvolta dimezzata se fosse possibile "invertire" l'azione eseguita all'ultima transizione, ossia esistesse un'azione che permetta di tornare allo stato corrente da quello nuovo. In tal caso si potrebbe sovrascrivere allo stato corrente il nuovo stato (tenendo ovviamente conto dell'informazione sulla variazione dell'energia). Sebbene questo comporti un piccolo beneficio di memoria e uno enorme per il tempo di esecuzione (l'operazione di copia per istanze di problemi molto grandi può fare da collo di bottiglia), questo approccio non è generalizzabile, in quanto non sempre è possibile ottenere l'azione inversa o farlo in tempi computazionalmente accettabili.

Poiché tutte le azioni dei problemi considerati in questo esercizio sono invertibili, oltre alla versione generale di *simulated annealing* è stata realizzata una versione che sfrutta la possibilità sopracitata. Essa è contenuta nel branch `copyless` della *repository* Github.

1.1 Note sui risultati sperimentali

Data l'impossibilità di stimare a priori i parametri dell'algoritmo che ne ottimizzano il funzionamento, si è optato di riportare i risultati conseguiti su instanze di problemi via via più grandi con tre configurazioni distinte:

- *schedule* logaritmica `Log`, con $t_0 = 1$, $\sigma = 1$ e numero massimo di iterazioni 10^6;
- *schedule* geometrica `Geo`, con $t_0 = 10$, $\alpha = 0.99$ e numero massimo di iterazioni 10^6;
- *schedule* lineare `Lin`, con $t_0 = 5$, $c = 10^5$; questa configurazione permette di raggiungere la temperatura di equilibrio (0) in $5 \cdot 10^5$ iterazioni;

2 Il problema delle n-regine

Il problema delle n-regine può essere così formulato:

Dato un numero naturale n, posizionare n regine su una scacchiera $n \times n$ in modo tale che nessuna si trovi sotto attacco.

Ogni regina è in grado di attaccare tutti i pezzi non ostacolati da altre pedine che si trovano sulla sua stessa riga e colonna o nelle stesse diagonali. Di conseguenza è chiaro che una soluzione al problema delle n-regine non potrà avere due regine sulla stessa colonna. Appare quindi naturale memorizzare la configurazione della scacchiera come la lista degli indici di riga delle regine. Tale approccio è estremamente vantaggioso rispetto a quello ingenuo che prevede la memorizzazione del contenuto di ogni cella della scacchiera, in quanto:

- l'occupazione di memoria è lineare in n invece che quadratica;
- lo spazio degli stati si riduce enormemente, passando da $\frac{(n^2)!}{(n^2-n)!}$ possibili configurazioni a n^n, con ovvie ripercussioni sull'efficacia della ricerca;

Bisogna dunque definire le azioni eseguibili sulla scacchiera. Una possibilità sarebbe quella di selezionare a caso una delle colonne e spostare casualmente la corrispondente regina. Nonostante la semplicità, questo approccio è tuttavia lontano dall'essere ottimale, visto che non tiene conto in alcun modo delle celle sotto attacco dalle altre regine. Si potrebbe quindi decidere di escludere tutte le celle che sono sotto attacco, ma questa operazione avrebbe delle ripercussioni sull'efficienza del metodo, visto il costo almeno lineare.

L'approccio migliore prevede di disporre le regine fin da subito su delle righe differenti, e di permettere come mossa lo scambio di riga tra due regine scelte casualmente. In questo modo lo spazio degli stati possibili si riduce, passando da n^n configurazioni a $n!$, e la scelta della mossa da intraprendere avviene in tempo costante.

Resta infine da stabilire come assegnare l'energia a ciascuno stato. Viene immediato pensare al numero di regine attualmente sotto attacco, visto che è la quantità che il problema intende minimizzare, oltre ad avere un valore minimo globale noto a priori, 0. Tale scelta non è però molto efficiente, poiché richiederebbe un riconteggio di tutte le regine sotto attacco ad ogni transizione, operazione che richiede tempo quadratico in n. Inoltre, facendo questa scelta, i valori di energia possibili per uno stato sono molto limitati (da 0 a n), per cui anche la distinzione tra stati "migliori" e stati "peggiori" si appiattisce.

Per ovviare ai problemi sopracitati, si è deciso di prendere come energia il numero di pezzi attaccati da ciascuna regina. Questo approccio ha due notevoli vantaggi: si ha una maggiore distinzione tra la qualità degli stati (l'algoritmo tende a sfavorire configurazioni in cui una regina attacca molte regine), e l'implementazione è molto più computazionalmente conveniente grazie alla tecnica delle diagonali descritta in [8], che consiste nel tener conto

del numero di regine all'interno di ciascuna delle $2n-1$ diagonali sinistre e $2n-1$ diagonali destre (il costo in termini memoria rimane quindi sempre lineare in n). Ciò permette di calcolare l'energia della nuova configurazione in tempo costante (ignorando il costo dovuto alla copia dello stato, che rimane invece lineare in n), migliorando sensibilmente le prestazioni dell'algoritmo di ricerca.

2.1 Risultati sperimentali

Di seguito si riportano i risultati conseguiti con le tre configurazioni di parametri:

- configurazione `Log`: il metodo restituisce pressoché sempre soluzioni corrette o al più prossime ad esserlo fino ad $n \approx 5000$. Oltre questa soglia, il metodo continua a produrre in tempi accettabili soluzioni sub-ottime di qualità fino $n \approx 10^4$.

  Per esempio, per $n = 10^5$, eseguendo tre volte il metodo si sono ottenute delle soluzioni aventi energie dell'ordine di 10^1;

- configurazione `Geo`: questa configurazione porta a risultati del tutto analoghi a quelli visti nel precedente caso;

- configurazione `Lin`: il metodo restituisce soluzioni ottime o sub-ottime fino ad $n \approx 500$. Superata questa soglia, le soluzioni trovate iniziano ad essere di qualità sempre peggiore, specialmente se confrontate con i risultati ottenuti con le altre configurazioni. Ponendo per esempio, $n = 1000$, l'algoritmo restituisce un minimo locale dell'ordine delle decine, quando le altre configurazioni trovano usualmente soluzioni ottime.

3 Il problema del quadrato magico

Il problema del quadrato magico può essere così formulato (vedi [4]):

Dato un numero naturale n, determinare un quadrato magico $n \times n$, ossia una matrice $Q = (q_{ij}) \in \mathbb{N}^{n \times n}$ che ha per elementi tutti i numeri naturali da 1 a n^2 tale che:

$$\sum_{i=1}^{n} q_{ih} = \sum_{j=1}^{n} q_{kj} = \sum_{i=1}^{n} q_{ii} = \sum_{i=1}^{n} q_{i(n-i+1)} = M_n \quad \forall h, k = 1, \ldots, n$$

Il valore M_n è detto **costante magica** [4] ed è così calcolata:

$$M_n = \frac{n(n^2+1)}{2}.$$

Uno stato del problema sarà quindi descritto da una matrice $n \times n$ (o, equivalentemente, da un vettore di lunghezza n^2) contenente i numeri naturali da 1 a n^2 in un determinato ordine. Lo stato iniziale semplicemente sarà quello descritto dalla matrice:

$$\begin{pmatrix} 1 & 2 & \ldots & n \\ n+1 & n+2 & \ldots & 2n \\ \vdots & \vdots & \ddots & \vdots \\ n^2-n+1 & n^2-n+2 & \ldots & n^2 \end{pmatrix}$$

L'energia associata allo stato sarà la differenza in valore assoluto tra le somme degli elementi sulla stessa riga/colonna/diagonale destra/diagonale sinistra e la costante magica M_n

L'azione permessa al metodo sarà lo scambio di posizione di una coppia di elementi scelti a caso. Ciò permette di generare il nuovo stato in tempo costante (ignorando il costo dovuto alla copia dello stato, che è quadratico in n), in quanto il calcolo della nuova matrice e della nuova energia può partire dagli attributi dello stato corrente.

Tutte le scelte intraprese non permettono però di ridurre il numero di stati distinti, che rimane $(n^2)!$.

3.1 Risultati sperimentali

Di seguito si riportano i risultati conseguiti con le tre configurazioni di parametri:

- configurazione `Log`: la configurazione permette di ottenere soluzioni quasi corrette (energia rimasta nell'ordine delle unità) fino a $n \approx 40$. Fino $n \approx 60$, gli stati finali hanno energie nell'ordine delle decine, mentre oltre questa soglia il metodo si allontana via via sempre di più dall'ottimo;

- configurazione `Geo`, questa configurazione permette di ottenere minimi locali di energia nell'ordine delle decine fino a $n \approx 50$, risultando quindi inferiore alla configurazione `Log`.
- configurazione `Lin`: permette di ottenere soluzioni che hanno un energia al di sotto di 10^2 fino a $n \approx 45$.

I risultati conseguiti permettono di constatare una maggiore difficoltà nel trovare soluzioni a questo problema piuttosto che al problema delle n-regine. Ciò probabilmente è da attribuirsi:

- ad uno spazio degli stati incredibilmente più vasto ($(n^2)!$ contro $n!$);
- una più ripida crescita delle possibili differenze di energia tra stati, fatto che rende molto più improbabile l'accettazione di una transizione sfavorevole, con una conseguente minore esplorazione dello spazio degli stati;

4 Il problema del commesso viaggiatore

Il problema del commesso viaggiatore può essere così formulato (vedi [2], [5]):

Dato un grafo pesato completo $\mathcal{G}$, determinare un circuito hamiltoniano di costo minimo.

In generale, $\mathcal{G}$ può essere sia orientato che non orientato. In questo esercizio $\mathcal{G}$ è stato considerato non orientato i.e. ci si è concentrati sulla risoluzione del problema del commesso viaggiatore *simmetrico*. Tale decisione ha un'immediata ripercussione: la scelta della punto da cui parte (e conseguentemente finisce) il circuito hamiltoniano è totalmente irrilevante al fine di determinarne uno di costo minimo. Da ciò ne consegue che il primo (ultimo) vertice del circuito è costante, per cui la memorizzazione di quest'ultimo sarebbe teoricamente inutile. Nella pratica si è però deciso di memorizzare comunque questo vertice sia all'inizio del circuito che alla fine, in modo da semplificare grandemente i calcoli e rendere più leggibile l'implementazione.

Uno stato è costituito da due elementi: un vettore corrispondente al circuito hamiltoniano considerato e la lunghezza dello stesso, che viene naturalmente considerata dal metodo come la sua energia. L'azione permessa al metodo è lo scambio di due città nel circuito, operazione che permette di effettuare la transizione tra stati in tempo costante (ignorando il costo dovuto alla copia dello stato, che è lineare nel numero di punti). Sono state provate anche le mosse suggerite da [3] e [6], ma senza ottenere risultati apprezzabili (sebbene ([10]) sia riuscito a migliorare in modo sostanzioso le prestazioni attraverso un'implementazione molto più elaborata rispetto a quella vista in questo esercizio).

4.1 Risultati sperimentali

I grafi sono memorizzati in file con estensione `.graph` come una serie di coordinate reali x, y dei vertici, una per linea. Il peso di un arco coincide con la distanza euclidea dei suoi estremi, per cui non è necessario memorizzarla.

L'algoritmo è stato testato su tre grafi: uno creato artificialmente di 5 punti (`penta.graph`), mentre i due rimanenti (`djibouti.graph` (38 punti), `luxembourg.graph` (980 punti)) contengono le coordinate $2D$ delle città degli stati del Djibouti e del Lussemburgo (reperibili su [9]).

Tutte e tre le configurazioni trovano la soluzione ottima di `penta.graph`, mentre per `djibouti.graph` si sono registrate grandi variazioni nelle lunghezze del circuito trovato ($2000 - 4000$ unità superiori al minimo globale conosciuto, 6656). Per `luxembourg.graph` non si ottengono, purtroppo, buone soluzioni, specialmente per quanto riguarda la *schedule* `Lin` (le lunghezze dei circuiti sono ≈ 5 volte quelle del minimo conosciuto, 11340).

Riferimenti bibliografici

[1] Stuart J. Russel, Peter Norvig - *Artificial Intelligence: A Modern Approach* - Sez. 4.3, 2a edizione, 2003

[2] Stuart J. Russel, Peter Norvig - *Artificial Intelligence: A Modern Approach* - Sez. 3.2, 2a edizione, 2003

[3] S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi - *Optimization by Simulated Annealing* - Science Vol. 220 N. 4598, 13 Maggio 1983

[4] Wolfram MathWorld - *Magic Square* - https://mathworld.wolfram.com/MagicSquare.html

[5] Wolfram MathWorld - *Travelling Salesman Problem* - https://mathworld.wolfram.com/TravelingSalesmanProblem.html

[6] S. Lin, B. W. Kernighan - *An Effective Heuristic Algorithm for the Traveling-Salesman Problem*- Operation Research Vol. 21 I. 2, Aprile 1973

[7] V. Granville, M. Krivanek. J. P. Rasson - *Simulated annealing: a proof of convergence* - IEEE Transactions on Pattern Analysis and Machine Intelligence Vol. 16 I. 6, Giugno 1994

[8] I. Martinjak, M. Golub - *Comparison of Heuristic Algorithms for the N-Queen Problem* - TI 2007 29th Int. Conf. on Information Technology Interfaces, 25-28 Giugno 2007

[9] *TSP Data Set* - http://www.math.uwaterloo.ca/tsp/data/index.html

[10] David S. Johnson, Lyle A. McGeoch - *The Traveling Salesman Problem:A Case Study in Local Optimization* - Sez. 5, 20 Novembre 1995