N-QUEENS PROBLEM

Alessio Marco Rinaldo

November 2, 2024

Contents

1 Constraint propagation & backtracking

1.1 Introduzione

L'algoritmo di constraint propagation consente di ridurre la dimensione del dominio di possibili valori che ad ogni iterazione le variabili possono assumere; mentre il backtracking serve per correggere l'operazione compiuta in caso ci si renda conto che non porta ad una soluzione del problema.
Applicato al problema delle N-regine ciò significa che, nel piazzamento delle future regine, scarto a priori tutte le celle della scacchiera che sono già tenute sotto scacco dalle regine precedentemente immesse (*arc consistency*). Ciò riduce ad ogni piazzamento il numero di possibili celle utilizzabili, rendendo più veloce la soluzione. Nel caso durante il processo ci si accorga che tutte le celle sono già tenute sotto scacco ma mancano ancora regine da piazzare, occorrerà rivedere le precedenti mosse.

1.2 Descrizione algoritmo

La configurazione iniziale del problema presenta una scacchiera senza alcuna regina piazzata di dimensioni $N \times N$. Il valore di N viene chiesto in input all'utente ad ogni esecuzione.
Il piazzamento delle regine avviene sempre dalla prima cella in alto a sinistra della scacchiera. Una volta piazzata la prima regina vengono da subito scartate come posizioni possibili le celle facenti parte della prima riga, della prima colonna e della diagonale principale della scacchiera. L'iterazione avviene sulle colonne: si passa alla seconda colonna e si posiziona la seconda regina nella prima cella libera da scacchi, procedendo analogamente alla rimozione delle relative celle tenute sotto scacco da quest'ultima. L'iterazione successiva avviene nella terza colonna,cercando sempre la prima cella disponibile per piazzare la terza regina. Dopo k iterazioni, qual ora si arrivi ad una colonna della scacchiera che ha tutte le sue celle sotto scacco dalle precedenti regine, interviene l'algoritmo di backtracking: viene ricontrattato il piazzamento della *k-1-esima* regina (la quale si ricorda, era stata posizionata nella *k-1-esima* colonna della scacchiera nella PRIMA posizione libera da scacchi disponibile). Quest'ultima viene spostata dalla posizione in cui si trova per venire piazzata alla SECONDA cella libera da scacchi secondo una ricerca del tipo depth-first (sempre sulla stessa colonna, così da mantenere sempre una sola regina per ogni colonna della scacchiera). Qual ora anche la *k-1-esima* regina non abbia possibilità di spostamento si interverrà sulla *k-2-esima* e così via. L'algoritmo si ferma quando si riesce a raggiungere l'ultima colonna della scacchiera e piazzare l'ultima regina.

1.3 Analisi dati

N	Simulazione 1 [s]	Simulazione 2 [s]	Simulazione 3 [s]	Media [s]
4	0.000117	0.000074	0.000072	0.000088
5	0.000055	0.000035	0.000035	0.000042
6	0.000467	0.000446	0.000445	0.000453
7	0.000121	0.000104	0.000102	0.000109
8	0.002397	0.002826	0.002779	0.002667
9	0.001021	0.000931	0.000993	0.000982
10	0.002863	0.002702	0.002697	0.002754
11	0.001651	0.001845	0.001842	0.001779
12	0.009157	0.010648	0.010091	0.009965
13	0.004779	0.004177	0.004189	0.004382
14	0.092108	0.086156	0.087472	0.088579
15	0.071612	0.067621	0.067155	0.068796
16	0.543221	0.572111	0.554050	0.556461
17	0.355138	0.318431	0.336537	0.336035
18	2.698301	2.672116	2.643544	2.671320
19	0.183068	0.177101	0.185382	0.181850
20	15.886586	16.444361	15.258886	15.863944
21	0.914740	0.753283	0.720940	0.796988
22	164.601116	163.634063	169.951315	166.062831
23	2.900028	2.718106	2.703176	2.773770
24	48.661843	49.054037	45.487955	47.734612

Table 1: Tempi di esecuzione per diversi valori di N

Si nota come vi sia un incremento medio di tempo d'esecuzione all'aumentare di N.
Si notano inoltre alcuni valori di N (i pari in genere) che in media hanno tempi di calcolo maggiori (addirittura con N=22 si raggiunge un tempo di calcolo di quasi 3 minuti a singola esecuzione). Ciò è dovuto all'approccio puramente deterministico utilizzato che quindi sarà molto avvantaggiato da una determinata configurazione della scacchiera iniziale piuttosto che da un'altra, a favore però di una varianza associata alla media delle simulazioni fatte con lo stesso valore di N molto bassa (perché le mosse fatte saranno identiche). In questo caso la diversità delle condizioni iniziali dipende unicamente dal valore di N in quanto si è scelto di iniziare a piazzare la prima regina sempre in posizione [0,0].
Implementando il posizionamento randomico della prima regina e ripetendo il test probabilmente si otterrebbe una crescita più monotona dei tempi medi rispetto al parametro N ma una varianza maggiore tra le simulazioni fatte con lo stesso N in quanto sarà questa nuova condizione iniziale ad influenzare maggiormente le mosse future e non più la grandezza della scacchiera.

2 Simulated annealing

2.1 Introduzione

L'algoritmo di simulated annealing prende il nome dall'omonimo processo in metallurgia dove un materiale viene scaldato e poi raffreddato lentamente, per ridurre il più possibile i difetti e le imperfezioni. Il simulated annealing è utile qual ora il problema in questione presenti molti minimi (o massimi) locali della funzione che si sta cercando di minimizzare (o massimizzare).
Applicato al problema delle N-regine ciò significa che si cercherà sempre di minimizzare il numero totale di coppie di regine sotto scacco, ma all'inizio verrà data comunque una probabilità alta di passare ad uno stato peggiore e in caso avvenga una stagnazione in una configurazione che presenta poche coppie sotto scacco deve essere comunque data la possibilità di spostarsi temporaneamente ad una configurazione meno ottimale con una probabilità non nulla, per permettere di esplorare una parte dello spazio delle soluzioni che ancora non era stato esplorato.

2.2 Descrizione algoritmo

La configurazione iniziale del problema presenta una scacchiera di dimensioni $N \times N$ con tutte N le regine già piazzate in una posizione random della scacchiera, avendo come unico vincolo quello di non piazzare mai due regine sulla stessa colonna. Si ha una funzione da minimizzare che è quella che conta il numero di coppie di regine sotto scacco totali in una determinata configurazione.
Viene inizialmente generata una schedule di temperature tramite dei parametri prestabiliti (che stabilisce il numero di iterazioni da fare e, come spiegato di seguito, la probabilità di cambio configurazione della scacchiera). Impostare dei parametri adeguati è cruciale per l'efficacia dell'algoritmo: una temperatura iniziale o una velocità di raffreddamento troppo bassa non permetto di fare sufficienti iterazioni e diminuisce di conseguenza la probabilità di arrivare ad una soluzione del problema. Se la soluzione viene trovata prima della fine della schedule il ciclo viene interrotto e la soluzione viene restituita.
Per ogni valore della schedule viene generato un possibile stato successivo a quello attuale, scegliendolo randomicamente da una pool di neighbors generati come segue: basandosi sempre sulla configurazione attuale, per ogni neighbor scelgo una colonna random della scacchiera e modifico la posizione della relativa regina (rimanendo sempre nella stessa colonna). Sia la dimensione della pool di neighbors, sia il metodo scelto per la generazione di quest'ultimi sono fondamentali per permettere un'adeguata esplorazione dello spazio delle soluzioni. In questo caso è stato scelto un approccio conservativo per entrambi i fattori (ogni neighbor differisce dallo stato attuale per la posizione di solamente una delle N regine e la dimensione della pool cresce linearmente con l'aumentare di N quando invece le possibili configurazioni crescono come N^2), così facendo all'aumentare di N si limita sempre di più l'esplorazione delle configurazioni possibili, a favore di un tempo di calcolo inferiore.
Viene confrontata la funzione da minimizzare (si ricorda essere il numero totale di coppie di regine che si tengono sotto scacco) applicata alla configurazione attuale e al possibile stato successivo. Nel caso lo stato successivo abbia meno coppie sotto scacco dell'attuale configurazione viene compiuto il passaggio dallo stato attuale al neighbor selezionato. Nel caso lo stato attuale invece presenti meno coppie sotto scacco del suo neighbor, la transizione può comunque avvenire con una probabilità dettata dalla distribuzione di Boltzmann: $e^{-de/T}$ dove $de = couplesInCheck(next) - couplesInCheck(current)$ (sarà quindi positivo se lo stato attuale è più conveniente del successivo) e T è la temperatura alla corrente iterazione (si ricorda che schedule è una funzione monotona decrescente). Questa funzione tende a 0 per $T \to 0$ e ad 1 per $T \to \infty$: a temperature alte la probabilità di transizione in stati peggiori è alta per permettere un'esplorazione dello spazio delle soluzioni più completa; man mano che il numero di iterazioni cresce, la temperatura scende e minore sarà la probabilità di transizione, in quanto si vuole mantenere una configurazione pseudo-ottimale.

2.3 Analisi dati

Simulazioni fatte con i seguenti parametri iniziali:

- Temperatura iniziale = 100000

- Cooling rate = 0.995
- Temperatura minima = 0.01
- Numero di neighbors generati = $N * 5$

N	**Simulazione 1 [s]**	**Simulazione 2 [s]**	**Simulazione 3 [s]**	**Soluzioni trovate**
4	0.0443	0.0621	0.0001	3/3
5	0.2809	0.2624	0.0730	3/3
6	4.5173	3.2732	3.7519	2/3
7	4.6076	4.5343	4.4875	3/3
8	5.9591	7.4861	7.1035	2/3
9	9.7308	9.7652	9.7723	2/3
10	11.6263	13.5789	10.5001	2/3
11	16.3058	17.0534	15.0956	1/3
12	16.1317	21.2037	19.5604	1/3
13	22.8861	24.6752	22.0531	0/3
14	29.3264	27.7922	28.0539	0/3
15	29.5804	33.2315	32.1456	1/3
16	43.8022	35.6518	39.4039	1/3
17	45.6999	47.5532	42.2518	0/3
18	48.9767	49.1215	47.2055	0/3
19	57.9137	55.4555	57.6182	0/3
20	62.1245	65.0497	63.3373	0/3

Table 2: Risultati delle simulazioni per ogni valore di N

È evidente come con l'aumentare dei valori di N aumenti anche il tempo impiegato per generare la soluzione. Come già accennato inoltre c'è una forte dipendenza da come è stato creato lo schedule. Ad esempio è stata fatta un'ulteriore simulazione per N=13 (che in nessuna delle tre tabellate ha restituito una soluzione) aumentando il cooling rate da 0.995 a 0.998 che ha restituito una soluzione in un tempo di 83.1272 secondi. Ciò ha fatto aumentare il numero di iterazioni possibili di più del doppio rispetto al valore precedente, permettendo di esplorare meglio lo spazio delle soluzioni. Aumentando ulteriormente i parametri di temperatura o la dimensione della pool dei neighbors probabilmente si potrebbero trovare soluzioni anche per N maggiori, a discapito di un tempo di calcolo di molto maggiore.

3 Genetic algorithm

3.1 Introduzione

Il genetic algortihm è un algoritmo di ottimizzazione basato su concetti evolutivi come la selezione naturale e la mutazione genetica.
Applicato al problema delle N-regine ciò significa che partendo da una popolazione di molti individui (scacchiere con diverse configurazioni), attraverso un processo di selezione solo gli individui più promettenti (con meno coppie sotto scacco) sopravvivranno e miglioreranno a loro volta.

3.2 Descrizione algoritmo

La configurazione iniziale del problema presenta una popolazione iniziale di molte possibili configurazioni randomiche della scacchiera di grandezza $N \times N$ (mantenendo il vincolo di avere una sola regina per colonna)
Nel caso all'interno della popolazione vi sia già una soluzione, quest'ultima viene restituita.
Nel caso non siano già presenti soluzioni all'interno della popolazione, viene calcolato il numero di coppie sotto scacco per ciascuna configurazione (che rimane la funzione da minimizzare). La mating pool viene generata tramite estrazioni randomiche di elementi dalla popolazione, con una probabilità PESATA dal valore che assume la funzione couplesInCheck(): meno coppie sotto scacco ha una determinata configurazione, più probabile sarà la sua estrazione. Per fare ciò definisco: $f_i = \frac{1}{couplesInCheck(i)}$ come la pseudo-probabilità della i-esima configurazione che sarà maggiore tanto più il numero di coppie sotto scacco associato sarà piccolo. La probabilità effettiva associata si ottiene normalizzando il valore ottenuto per la somma di tutti i valori possibili: $p_i = \frac{f_i}{\sum_k f_k}$. In base a queste probabilità, si genererà una mating pool in cui mediamente le configurazioni sono già buone.
Successivamente interviene l'operatore di crossover: prendendo a coppie gli elementi dalla mating pool (1° e 2°, 3° e 4°...), con una probabilità relativamente alta (70% o superiore, anche in base alla stagnazione della mating pool) scambio le prime k colonne dei due stati (k numero random compreso tra 1 e N-1), così da migliorare l'esplorazione dello spazio delle soluzioni.
Il secondo e ultimo operatore implementato è quello di mutazione, il quale itera per tutti gli elementi della mating pool (ai quali è già stato applicato l'operatore di crossover) e per ciascuno di essi, con una probabilità bassa (inferiore al 5%), modifica la posizione di una sola regina nella scacchiera (viene variato solo l'indice della riga della cella nella quale si trova la regina, mantenendo invariata la colonna).
Al termine del processo, la mating pool ottenuta (i "figli") sarà la nuova popolazione iniziale, ripetendo ricorsivamente la procedura.

3.3 Analisi dati

Simulazioni fatte con i seguenti parametri:

- Cardinalità della popolazione iniziale = $N * 5$
- Cardinalità della mating pool = int($\frac{2}{3} * \#_{popolazione}$)
- Numero massimo di cicli ricorsivi = 10000

N	Simulazione 1 [s]	Simulazione 2 [s]	Simulazione 3 [s]	Soluzioni trovate
4	0.058	0.507	0.001	3/3
5	0.004	0.274	0.001	3/3
6	7.783	0.013	0.342	3/3
7	0.038	0.273	21.327	3/3
8	0.056	31.181	30.712	2/3
9	0.102	6.954	8.616	3/3
10	3.667	1.277	61.233	2/3
11	78.446	78.726	85.751	0/3
12	122.745	126.989	122.829	0/3
13	129.911	132.365	134.482	0/3

Table 3: Tempi di simulazione per valori di N e soluzioni trovate

È evidente come si possa stabilire quasi un trade-off tra il valore di N e l'efficacia dell'algoritmo (per come è stato implementato e per le scelte dei parametri fatte).
Possibili miglioramenti:

- Aumentare la grandezza della popolazione iniziale
- Aumentare la grandezza della mating pool
- Aumentare il numero massimo di cicli ricorsivi
- Implementare una variante più efficiente dell'operatore di crossover
- Scegliere in modo più accurato le probabilità di crossover e mutazione per evitare alterazioni casuali di configurazioni già pseudo-ottimali
- Introdurre una componente di elitismo generazionale (ad esempio rimpiazzando le k peggiori configurazioni della popolazione attuale con le k migliori della popolazione precedente)

Si nota inoltre una forte variabilità del tempo di calcolo anche tra le simulazioni con lo stesso valore di N, tipica degli algoritmi che presentano un'importante componente randomica al loro interno.