Ricerca locale in Python
Progetto per il corso di Intelligenza Artificiale

GIACOMO MANZOLI
Università degli Studi di Padova
30 gennaio 2016

Sommario

Implementazione di alcuni algoritmi che effettuano una ricerca locale in Python. Gli algoritmi implementati vengono poi confrontati tra loro nella risoluzione del problema delle N-Regine.

Indice

1 Introduzione

Affrontare un problema di ricerca nello spazio degli stati con una ricerca basata su un albero può richiedere troppa memoria e rendere impossibile risolvere il problema.

Ci sono però delle tipologie di problemi che richiedono la ricerca di una configurazione ottima oppure di una configurazione che rispetti determinati vincoli. Per questo tipo di problemi non interessa il cammino che porta alla soluzione, ma interessa solamente la descrizione dello stato goal.

In questo caso conviene utilizzare un algoritmo di ricerca locale, ovvero un algoritmo che mantiene in memoria un unico stato e che tenta di migliorarlo applicando l'azione migliore tra le possibili, renendo così costante il consumo di memoria.

Dal momento che un algoritmo di ricerca locale non tiene traccia del percorso che ha fatto, non è possibile effettuare il back-tracking delle azioni effettuate, pertanto possono verificarsi delle situazioni in cui l'algoritmo si blocca in uno stato dal quale non è possibile raggiungere stati migliori, ma che non rappresenta una soluzione del problema.

L'implementazione più semplice della ricerca locale è data dall'algoritmo HIll Climbing, il quale cerca di passare dallo stato corrente allo stato migliore tra tutti quelli possibili e, quando non sono presenti stati migliori, termina la ricerca producendo una soluzione sub-ottima.

La scelta dell'azione migliore viene fatta utilizzando una funzione euristica che valuta la "distanza" che c'è tra lo stato e uno stato goal. La scelta di questa funzione risulta particolarmente importante, sia perché influenza il numero di mosse necessarie per trovare una soluzione, sia perché la presentaza di massimi o minimi locali può portare l'algoritmo di ricerca locale a fermarsi senza trovare una soluzione.

Per diminuire la probabilità che l'algoritmo di ricerca termi trovando una soluzione sub-ottima, sono state proposte delle varianti:

- **Hill Climbing Stocastico**: effettua una ricerca Hill Climbing, scegliendo un'azione tra tutte quelle che migliorano lo stato corrente. La probabilità che un'azione venga scelta è influenza dalla bontà dell'azione.
- **HIll Climbing con mosse laterali**: effeutta una ricerca Hill Climbing con la differenza che, quando non esistono azioni migliori, vengono prese in considerazione anche le azioni che non peggiorano lo stato attuale. La possibilità di poter eseguire anche queste mosse introduce la possibilità di ciclare infinitamente su un insieme di stati, pertanto conviene limitare il numero di mosse possibili.
- **Hill Climbing con riavvio casuale**: effettua una ricerca HIll Climbing e se non viene trovata una soluzione la ricerca viene riavviata a partire da uno stato casuale. I riavvi terminano quando viene trovata una soluzione ottima.
- **Simulated Annealing**: effettua la ricerca scegliendo casualmente lo stato successore. Vengono presi in considerazione anche stati peggiori dello stato attuale, tuttavia la probabilità di spostarsi su uno stato peggiore diminuisce all'avanzare della ricerca.

1.1 N-Regine

È un problema che consiste nel disporre N regine su una scacchiera di dimensione $N * N$ in modo che non si minaccino tra loro.

Questo problema risulta adatto per testare gli algoritmi di ricerca locale, in quanto interessa solamente la disposizione finale della scacchiera e non le mosse necessarie a raggiungere tale configurazione.

Inoltre, trattandosi di un problema combinatori, il numero di possibile di stati risulta essere $\frac{(N*N)!}{N!(N*N-N)!}$ quindi applicare un algortimo di ricerca ad albero potrebbe richiedere troppa memoria. Ad esempio, con $N = 8$ si hanno 4.426.165.368 possibili stati.

Tuttavia, il numero di stati può essere ridotto aggiungendo il vincolo che ci sia una regina per colonna, così facendo i possibili stati diventano N^N e ogni stato ha $N * (N - 1)$ successori. Questo nuovo vincolo non influsce sul numero di soluzioni dal momento che uno stato con due regine sulla stessa colonna non sarà mai una soluzione del problema.

Come euristica per valutare la bontà di uno stato viene utilizzato il numero di coppi di regine che si minacciano tra loro, così facendo l'obiettivo dell'algoritmo di ricerca diventa quello di trovare un stato il cui numero di minacce risulta essere 0.

1.2 Attività svolte

Il progetto comprende le seguenti attività:

- Analisi del codice già disponibile nel repository del libro "Intelligenza Artificiale: Un approccio moderno"[1] relativo alla ricerca Hill Climbing;
- Implementazione degli algoritmi precedentemente menzionati:
  - Hill Climbing classico;
  - Hill Climbing stocastico;
  - Hill Climbing con mosse laterali;
  - Hill Climbing a riavvio casuale;
  - Simulated Annealing.
- Modellazione e risoluzione del problema delle N-Regine utilizzando i vari algoritmi sviluppati;
- Analisi dei risultati delle prove effettuate.

[1] https://code.google.com/p/aima-python/

2 Architettura della soluzione

L'architettura della soluzione è stata progettata a partire dal codice presente nel repository del ilbro "Intelligenza Artificiale: Un approccio moderno", il quale contiene l'implementazione Python dell'algortimo Hill Climbing standard e di alcune classi quali: `Problem` che definisce l'interfaccia di un problema di ricerca e `Node` che rappresenta il nodo di un albero di ricerca.

Dal momento che il codice di queste classi si trovava all'interno di un unico modulo Python, è stato necessario apportare opportune modifiche in modo da ottenere un codice più strutturato:

- Le funzioni di supporto, come la scelta casuale di un massimo, sono state racchiuse nel modulo `util`:
- È stato creato un package `problem` che contiene la classe astratta `Problem` e la sua derivata `NQueens`;
- Le classi relative alla ricerca, ovvero `Node`, `HillClimbingSearch` e le altre versioni implementate, sono state racchiuse nel package `search`.

2.1 problem

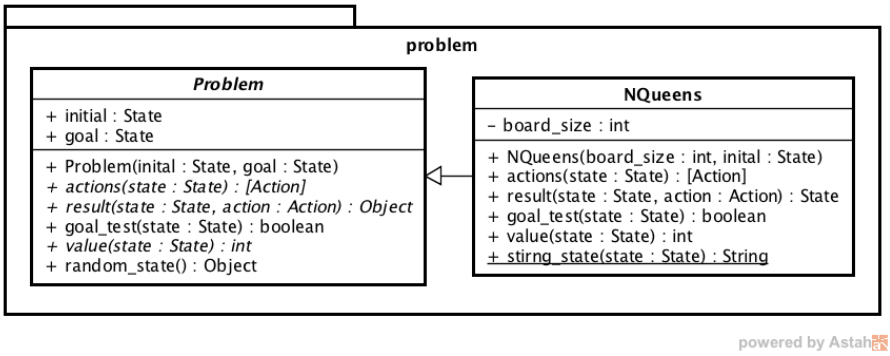

Figura 1: Diagramma UML del modulo `problem`.

Per rappresentare un problema di ricerca è stata utilizzata la stessa rappresentazione proposta dal codice del repository e per modellare il problema delle N-Regine è stata creata una nuova classe che deriva da `Problem`, la quale concretizza i vari metodi astratti.

Gli stati del problema delle N-Regine vengono rappresentati con una lista di interi di lunghezza *N* i cui elementi vanno da *0* a *N-1*. Con questa rappresentazione l'elemento *i-esimo* della lista indica la riga in cui si trova la regina della colonna *i*.

Così facendo non è necessario definire un tipo ad-hoc per la rappresentazione dello stato, è sufficente la lista di interi[2].

Le azioni del problema vegnono invece rappresentate con una tupla di due interi `(c,r)` la quale rapppresenta l'azione che sposta la regina della colonna *c* nella riga *r*[3].

[2]Nei diagrammi UML, quando un metodo riceve come parametro uno stato viene utilizzato il tipo `State` al posto di `[int]` per specifcare che si tratta di uno stato, anche se l'implementazione effettiva è una lista di interi

[3]Anche in questo caso, nei diagrammi UML viene utilizzato il tipo `Action` per indicare la tupla `(c,r)`

2.2 search

Tutte le classi che rappresentano un algoritmo di ricerca hanno la stessa classe base `BaseSearch`, la quale definisce due metodi astratti `search(problem)` e `search_from_state(problem, state)`, che permettono rispettivamente di ricerca la soluzione di un problema a partire dallo stato iniziale o da uno stato specifico.

Questi due metodi ritornano una tupla `(state, node, cnt)` contenente il risultato della ricerca:

- `state`: rappresenta lo stato del problema sul quale è terminata la ricerca e non è detto che sia uno stato goal;
- `node`: rappresenta il nodo sul quale è terminata la ricerca;
- `cnt`: rappresenta il numero di passi che sono stati eseguiti durante la ricerca. Nel caso di una ricerca a riavvio casuale rappresenta il numero di riavvi.

Per rappresentare i nodi che vengono visitati dagli algoritmi di ricerca viene utilizzata la classe `Node`, la quale può essere utilizzata anche per le ricerche ad albero dal momento che contiene informazioni relative al cammino che c'è dal nodo iniziale al nodo corrente e anche il costo di tale cammino. Anche se queste informazioni non sono necessarie per una ricerca locale sono state mantenute dal momento che rappresentano lo storico dei passaggi fatti dall'algoritmo di ricerca e possono tornare utili per il debug o per visualizzare l'andamento della ricerca.

Le classi che rappresentano gli algoritmi di ricerca sono:

- `HillClimbingSearch`: implementa la ricerca Hill Climbing classica. Il costruttore ha come parametro opzionale `minimize`, un booleano che specifica se la ricerca deve minimizzare o massimizzare il valore della funzione di valutazione.
- `HillClimbingStocasticSearch`: implementa la ricerca Hill Climbing stocastica. Anche in questo caso il costruttore ha come parametro `minimize`.
- `HillClimbingLateralSearch`: implementa la ricerca Hill Climbing con mosse laterali. Il costruttore ha due parametri opzionali: `minimize` e `lateral_move`.
- `HillClimbingRandomRestartSearch`: implementa la ricerca a riavvio casuale. La ricerca da utilizzare può essere specificata con il parametro `search` del costruttore. Se questo parametro non viene specificato, viene utilizzata la ricerca `HillClimbingSearch`.
- `SimulatedAnnealingSearch`: implementa la ricerca secondo simulated annealing. Il costruttore della classe accetta due parametri, `minimize` che specifica se il problema di risolvere è di minimizzazione o meno, e `temperature_fn`, la funzione di raffreddamento. Se non viene specificata una funzione, viene utilizzata la funzione

$$\begin{aligned} temperature\_fn(t) &= 20e^{-0,05t} \text{ se } t < 1000 \\ &= 0 \text{ se } t \geq 1000 \end{aligned}$$

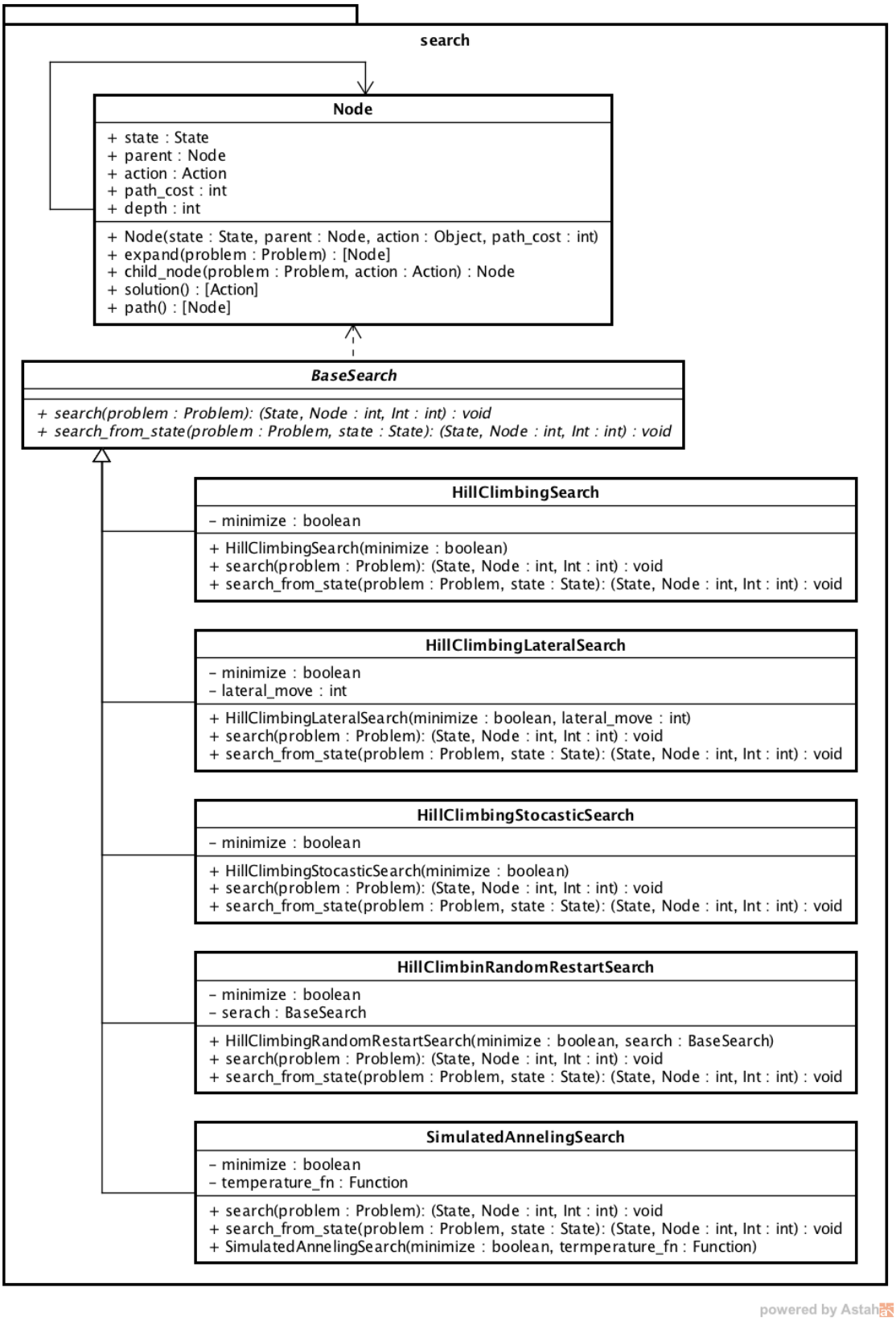

Figura 2: Diagramma uml del modulo `search`.

3 Prove effettuate

Una volta implementato il codice dei vari algoritmi e dopo aver verificato l'assenza di errori, sono state effettuate varie prove, i cui risultati vengono riportati nelle successive sotto sezioni.

C'è da osservare che nella maggior parte delle prove, gli algoritmi presi in cosiderazione sono stati eseguiti 1000 volte, pertanto il rapporto $\frac{\text{\# sol. ottime}}{\text{\# esecuzioni}}$ rappresenta una stima della probabilità di trovare la soluzione ottima per un determinato problema e questa stima risulta essere sempre meno accurata al crescere della dimensione del problema.

Di conseguenza la stima della probabilità di trovare una soluzione ottima per 4-Regine risulta molto più accurata della stima per 10-Regine, questo a casua della crescita esponziale del numero degli stati del problema.

3.1 Confronto tra i vari algoritmi

La prima prova effettuata è stata l'esecuzione di Hill Climbing, Hill Climbing con mosse laterali e Hill Climbing stocastico per risolvere N-Regine a partire da 4-Regine fino a 25-regine.

Ognuno dei problemi è stato risolto sia a partire da uno stato iniziale fisso, ovvero con tutte le regine nella prima riga, sia da uno stato generato casualmente. Inoltre, per ognuno di questi casi sono state effettuate 1000 esecuzioni di ogni algoritmo.

Dai risultati ottenuti, riportati nelle tabelle 1 e 2, si può osservare che:

- Hill Climbing laterale riesce quasi sempre a trovare una soluzione ottima, mentre le Hill Climbing normale e stocastico fanno sempre più fatica a raggiungere l'ottimo.
- Hill Climbing stocastico non migliora di molto l'Hill Climbing normale, solamente in alcuni casi trova più soluzioni ottime richiedendo più mosse. Tuttavia, uno dei vantaggi della versione stocastica di Hill Climbing è quello che dovrebbe trovare soluzioni migliori e la prova effettuata non tiene conto della qualità delle soluzioni sub-ottime. Pertanto è stata effettuata un'altra prova tenendo conto anche di questo. Tale prova è descritta nella sezione §3.4.
- Hill Climbing riesce sempre a trovare a trovare una soluzione a 4-regine partendo dallo stato inizile fisso in 3 mosse. Questo perché, le azioni possibili per la prima mossa sono `(1,3)` oppure `(2,3)` e una caratteristica delle soluzioni ottime di 4-Regine è di avere la forma `<X,0,3,X>` o `<X,3,0,X>`. Partendo da uno stato casuale invece non sempre viene scelta una di queste mosse, arrivando così a delle soluzioni sub-ottime.
- Il problema 6-Regine risulta essere particolarmente difficile da risolvere, sia a partire da uno stato fisso che da uno stato casuale. È stato quindi affrontato più nel dettaglio applicando solamente Hill Climbing laterale dal momento che è l'algoritmo che sta dando i risultati migliori. La descrizione di questa prova è nella sezione §prove:6regine.

3.2 Riavvi casuali

In questa prova sono state eseguite le precedenti versioni di Hill Climbing utilizzando però la versione a riavvio casuale.

	Hill Climbing			Hill Climbing laterale			Hill Climbing stocastico		
N	Tempo (s)	Mosse	Sol. Ottime	Tempo (ms)	Mosse	Sol. Ottime	Tempo (s)	Mosse	Sol. Ottime
4	0,00049	3	1000	0,00049	3	1000	0,00058	3,29	526
5	0,00120	3,95	793	0,00130	4,57	1000	0,00149	4,88	715
6	0,00259	4,52	213	0,02017	41,81	888	0,00335	6	116
7	0,00464	5,54	415	0,01280	16,44	946	0,00645	7,51	264
8	0,00829	6,44	184	0,02982	25,50	930	0,01200	9,10	158
9	0,01367	7,48	223	0,04485	25,34	943	0,02043	10,64	106
10	0,02197	8,47	119	0,09734	39,73	892	0,03408	12,26	69
11	0,03326	9,41	78	0,13012	38,87	916	0,05385	13,82	42
12	0,05009	10,41	74	0,17047	37,35	952	0,08068	15,31	34
13	0,07175	11,39	48	0,22080	36,75	962	0,11831	17,04	38
14	0,10134	12,46	53	0,29341	37,64	975	0,16871	18,68	43
15	0,13929	13,42	45	0,37876	37,70	971	0,23528	20,38	27
16	0,19675	14,15	40	0,49206	38,20	978	0,32068	21,96	33
17	0,26655	15,50	38	0,62832	39,02	988	0,43232	23,66	26
18	0,32632	16,49	25	0,79175	39,93	984	0,57571	25,54	24
19	0,42260	17,50	27	0,99589	41,24	983	0,74080	26,97	20
20	0,53441	18,51	25	1,21197	41,56	995	0,95826	28,85	22
21	0,67142	19,49	13	1,47746	42,32	993	1,21045	30,40	17
22	0,84227	20,60	21	1,76070	42,49	998	1,51940	32,06	13
23	1,03802	21,57	23	2,08641	43,27	996	1,89762	33,92	13
24	1,27302	22,56	20	2,55409	43,27	997	2,32966	33,50	13
25	1,56128	23,60	24	2,96620	45,52	998	2,83871	37,12	10

Tabella 1: Risultati medi di 1000 esecuzioni dei vari algoritmi a partire da uno stato iniziale fisso.

	Hill Climbing			Hill Climbing laterale			Hill Climbing stocastico		
N	Tempo (s)	Mosse	Sol. Ottime	Tempo (ms)	Mosse	Sol. Ottime	Tempo (s)	Mosse	Sol. Ottime
4	0,00034	1,56	428	0,00041	2,86	1000	0,00035	1,82	341
5	0,00074	2,14	690	0,00084	3,03	1000	0,00081	2,72	693
6	0,00158	2,30	121	0,01276	45,93	814	0,00178	3,01	87
7	0,00273	2,82	276	0,00864	15,65	912	0,00323	3,81	250
8	0,00476	3,20	123	0,01836	22,71	930	0,00571	4,45	112
9	0,00791	3,65	154	0,02660	22,66	924	0,00978	5,19	168
10	0,01218	4,96	54	0,06255	39,36	860	0,01561	5,73	52
11	0,01849	4,43	41	0,09029	38,86	897	0,02436	6,41	47
12	0,02783	4,82	44	0,11035	34,62	930	0,03590	7,09	54
13	0,039987	5,29	25	0,15324	32,77	968	0,05288	7,85	36
14	0,05579	5,71	24	0,19152	32,68	964	0,07559	8,60	36
15	0,07528	6,20	43	0,26458	31,88	969	0,10317	9,22	29
16	0,10097	6,71	24	0,31388	30,47	987	0,13739	10,09	24
17	0,13565	7	30	0,42151	33,20	980	0,18781	10,69	24
18	0,17606	7,56	24	0,50562	31,41	984	0,24386	11,39	15
19	0,22452	7,86	18	0,64227	31,63	993	0,31848	12,19	16
20	0,28445	8,31	26	0,76014	31,75	989	0,40667	12,91	20
21	0,35974	8,81	13	0,93101	33,32	992	0,51567	13,55	17
22	0,44227	9,25	15	1,11264	33,58	994	0,64465	14,29	17
23	0,52948	9,71	11	1,31210	32,51	991	0,80020	14,98	11
24	0,66810	10,31	9	1,58736	32,82	996	0,98633	15,80	12
25	0,81686	10,53	9	1,85988	33,77	999	1,20711	16,68	16

Tabella 2: Risultati medi di 1000 esecuzioni dei vari algoritmi a partire da uno stato iniziale generato casualmente.

	Hill Climbing			Hill Climbing laterale (100 mosse)			Hill Climbing stocastico		
N	Media Riavvii	Riavvii attesi	Differenza (%)	Media Riavvii	Riavvii attesi	Differenza (%)	Media Riavvii	Riavvii attesi	Differenza(%)
4	1	2,34	57,20	1	1	0	2,14	2,93	27,03
5	1,22	1,45	15,82	1	1	0	1,44	1,44	0,21
6	8,22	8,26	0,54	1,15	1,23	6,39	8,20	11,49	28,66
7	3,15	3,62	13,06	1,07	1,19	2,42	3,76	4	6
8	7,15	8,13	12,06	1,07	1,08	0,49	7,68	8,93	13,98
9	5,90	6,49	9,14	1,02	1,08	5,75	7,65	5,95	28,52
10	16,88	18,52	8,85	1,14	1,16	1,96	16,31	19,23	15,19
11	18,87	24,39	22,63	1,11	1,11	0,43	20,64	21,28	2,99
12	18,60	22,73	18,16	1,06	1,08	1,42	24,77	18,52	33,76
13	24,10	28.57	15,65	1,07	1,03	3,58	25,15	27,78	9,46
14	32,57	42,67	21,83	1,06	1,04	2,18	28,89	27,78	4
15	28,84	23,26	24,01	1,02	1,03	1,16	34,55	34,48	0,20
16	45,11	41,67	8,26	1,03	1,01	1,66	38,55	41,67	7,48
17	43,61	33,33	30,83	1,02	1,02	0,04	48,29	41,67	7,48
18	53,33	41,67	22,99	1,02	1,02	0,37	51,20	66,67	23,20
19	46,83	55,56	15,71	1,01	1,01	0,29	53,07	62,50	15,09
20	51,86	38,46	34,84	1	1,01	1,10	61,56	50	23,12
21	61,48	76,92	20,08	1,02	1,01	1,18	53,35	58,82	9,30
22	67,62	66,67	1,43	1	1,01	0,60	60,67	58,82	3,14
23	66,27	90,91	27,10	1	1,01	0,90	69,21	90,91	23,87
24	86,88	111,11	21,81	1	1,01	0,40	70,11	83,33	15,87
25	90,62	111,11	18,44	1,03	1	2,90	78,64	62,50	25,82

Tabella 3: Risultati medi di 100 esecuzioni dell'Hill Climbing con riavvii casuali, utilizzando le varie versioni di Hill Climbing. Alcune differenze espresse in % sono diverse da quelle attese, questo perché la tabella riporta i risultati approssimati e le differenze sono state calcolate utilizzando i dati non approssimati.

Anche in questo caso i problemi risolti vanno da 4-Regine a 25-Regine, ognuno risolto 100 volte, sia a partire dallo stato iniziale con tutte le regine su una stessa riga, che a partire da uno stato generato casualmente.

A differenza della prova precedente, sono stati presi in considerazioni i riavvii necessari per trovare una soluzione ottima piuttosto del numero delle mosse per ottenere una soluzione ottima.

Così facendo è stato possibile verificare che il numero medio di riavvii coincida con quanto atteso, ovvero $\frac{1}{p}$, dove p è la probabilità di trovare una soluzione ottima.

I risultati ottenuti sono riportati nella tabella 3 e la differenza dei valori ottenuti da quelli attesi viene evidenziata nella nel grafico della figura 3.

Dal grafico si può notare che la maggior parte dei risultati non si discosta di molto, ci sono solamente alcuni casi in cui la differenza è notevole e questo può essere causato dal numero ridotto di prove effettuate, che per motivi di tempo sono state limitate a 100. In ogni caso, la differenza media risulta essere inferiore al 20% per Hill Climbing e poco più del 15% per Hill Climbing stocastico.

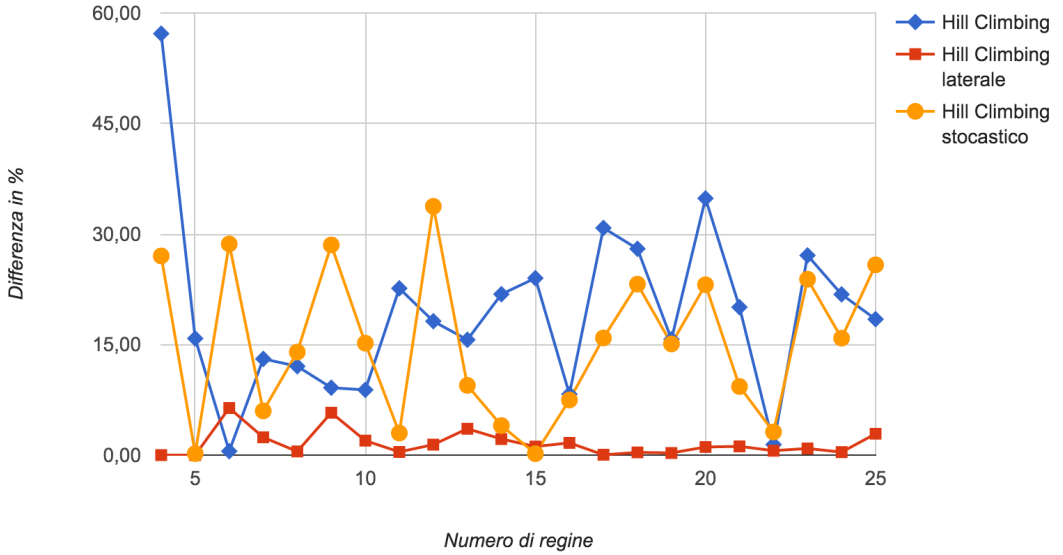

Figura 3: Valore assoluto della differenza tra il numero di riavvi attesi e quello medio, espresso in percentuale.

3.3 8-Regine

La prima volta che sono state eseguite le varie versioni di Hill Climbing per risolvere il problema delle 8-Regine, è risultata una probabilità di trovare una soluzione ottima molto più elevata di quella riportata sul libro del corso. In particolare, stando a quanto riportato, la probabilità di trovare una soluzione ottima con Hill Climbing a partire da uno stato casuale è del 14%, mentre alla prima esecuzione è stata stimata essere del 28% .

È stato quindi ricontrollato il codice prodotto è stato scoperto un errore nella funzione che genera gli stati del problema in modo casuale, la quale ogni tanto generava degli stati inziali con una regina posizionata fuori dalla scacchiera. Ad esempio nel caso delle 8-Regine potevano venir prodotti degli stati iniziali con una regina nella riga 8 anche se la scacchiera aveva solamente le righe da 0 a 7. Il problema che veniva risolto era quindi più facile e pertanto la probabilità di ottenere una soluzione ottima risultava più elevata.

Una volta corretto l'errore è stato rieseguito l'algoritmo per risolvere 100.000 volte 8-Regine in modo da poter controllare che fosse effettivamente quello il problema e, come si può notare dalla tabella 4, i risultati ottenuti si avvicinano molto ai risultati attesi.

L'errore nella generazione degli stati casuali ha influenzato anche i risultati di altre prove, le quali sono state ri-eseguite in modo da ottenere dei valori correttti. I risultati presenti nelle varie tabelle di questa relazione sono quelli ottenuti con la versione corretta del codice.

Hill Climbing			
	Probabilità sol. ottima	Mosse caso ottimo	Mosse caso sub-ottimo
Versione errata	28,19%	3,76	2,87
Versione corretta	13,9%	4,07	3,06
Valori attesi	14%	4	3
Hill Climbing con mosse laterali (100)			
	Probabilità sol. ottima	Mosse caso ottimo	Mosse caso sub-ottimo
Versione errata	96,92%	13,39	70,90
Versione corretta	94,3%	19,06	61,26
Valori attesi	94%	21	64

Tabella 4: 8-Regine: confronto tra i dati errati, quelli corretti e quelli attesi.

3.4 Hill Climbing Stocastico

Dal momento che nella prima prova effettuata l'algoritmo Hill Climbing Stocastico si è rilevato meno performante dell'Hill-Climbing classico, sia in termini di tempo di esecuzione, che in numero di soluzioni ottime trovate. È stata eseguita un'ulteriore prova per verificare se le soluzioni sub-ottime trovate dalla versione stocastica siano migliori delle soluzioni sub-ottime trovate dalla versione classica.

I risultati ottenuti sono riportati nella tabella 5 e vengono riassunti nelle figure 4 e 5, dalle quali si può notare che:

- a partire da uno stato casuale, la differenza del numero delle minacce è minima;
- a partire dallo stato con tutte le regine nella prima riga, le soluzioni sub-ottime ottenute con Hill Climbing sono migliori rispetto a quelle ottenute con la versione stocastica.

Pertanto si può concludere che Hill Climbing Stocastico non è ideale per risolvere il problema delle N-Regine.

	Hill Climbing			Hill Climbing stocastico		
N	Mosse medie	Sol. Ottime	Val. Sub-ottimo	Mosse medie	Sol. Ottime	Val. Sub-ottimo
4	1,48	375	1,123	1,81	363	1,187
5	2,15	709	1,553	2,68	692	1,571
6	2,35	120	1,202	3,04	100	1,229
7	2,82	241	1,476	3,81	273	1,512
8	3,18	126	1,509	4,49	136	1,508
9	3,68	129	1,606	5,12	132	1,561
10	4	55	1,685	5,76	62	1,696
11	4,47	41	1,780	6,38	46	1,789
12	4,85	48	1,876	7,05	38	1,894
13	5,39	49	1,917	7,86	43	1,963
14	5,79	40	1,990	8,45	32	2,082
15	6,08	21	2,114	9,38	33	2,132
16	6,57	25	2,189	9,94	31	2,220
17	7	31	2,189	10,63	16	2,264
18	7,51	17	2,287	11,42	20	2,303
19	7,93	19	2,435	12,22	19	2,389
20	8,33	18	2,469	12,66	13	2,508
21	8,81	15	2,457	13,48	6	2,487
22	9,29	16	2,513	14,28	12	2,605
23	9,51	16	2,590	15,08	14	2,628
24	10,17	19	2,651	15,85	17	2,711
25	10,52	14	2,736	16,56	19	2,723

Tabella 5: Confronto tra Hill Climbing e Hill Climbing stocastico. I risultati sono la media di 1000 prove, eseguite a partire da uno stato generato casualmente

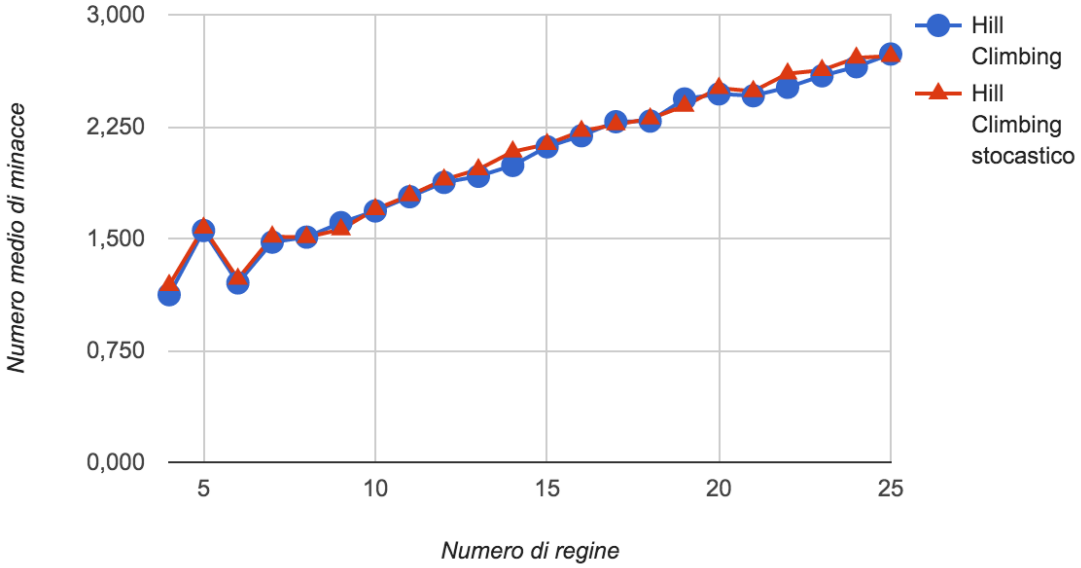

Figura 4: Differenza tra le soluzioni sub-ottime trovate con Hill Climbing e con Hill Climbing stocastico.

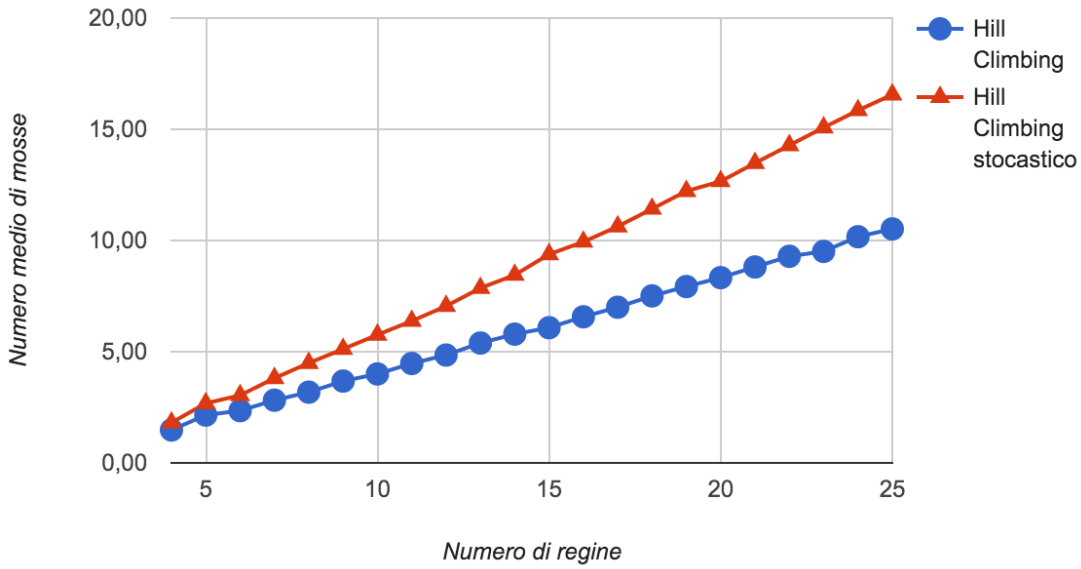

Figura 5: Numero di mosse effettaute da Hill Climbing e Hill Climbing stocatico

3.5 Hill Climbing Laterale e 6-Regine

Nella prima prova effettuata è emerso che Hill Climbing e le sue varianti hanno dei problemi a risolvere 6-Regine, sia a partire dallo stato iniziale che a partire da uno stato generato casualmente.

Si è quindi provato ad eseguire Hill Climbing con mosse laterali tenendo traccia del numero di mosse medie necessarie per ottenere una soluzione sub-ottima, provando a risolvere N-Regine con N da 6 a 9. I risultati ottenuti sono riportati nella tabella 6.

Osservando il numero di mosse medie in caso di fallimento è possibile riuscire a distinguere quando l'algoritmo termina perché finisce in un minimo locale o quando termina perché finisce in un plateaux o spalla.

Infatti, se il numero medio di mosse è vicino al numero di mosse massime l'algoritmo ha terminato l'esecuzione in quanto è entrato in un plateux dal quale non è riuscito ad uscirne, mentre se il numero di mosse è inferiore al numero massimo, l'algoritmo ha terminato l'esecuzione a causa di un minimo locale, ovvero è finito in uno stato in cui tutti i suoi successori erano peggiore dello stato corrente.

Pertanto il problema che incontra HIll Climbing con 100 mosse laterali nel risolvere 6-Regine è causato da un plateaux che fa ciclare l'algoritmo finché non termina le mosse a disposizione.

Sono state quindi aumentate le mosse laterali a disposizione dell'algoritmo, da 100 a 1000, in modo da aumentare la probabilità che l'algoritmo riesca a superare il plateaux.

Dalla tabella 7 si può notare che:

- l'algoritmo riesce sempre a trovare una soluzione ottima partendo dallo stato con tutte le regine nella prima riga, questo grazie al maggior numero di mosse laterali che gli permetto di oltrepassare dei plateaux che richiedono più di 100 mosse;
- il numero medio di mosse per trovare una soluzione ottima è aumentato, questo perché alcune delle soluzioni trovate dall'algoritmo richiedono più di 100 mosse;
- ci sono ancora dei plateaux che l'algoritmo non riesce a scavalcare e che vengono incontrati solamente partendo da uno stato casuale.

Hill Climbing Laterale (100 mosse, stato iniziale fisso)			
N	Sol. Ottime	Mosse medie caso ottimo	Mosse medie caso sub-ottimo
6	881	34,04	103,86
7	948	14,13	29,71
8	916	20,74	100,70
9	962	22,32	94,47
Hill Climbing Laterale (100 mosse, stato iniziale casuale)			
N	Sol. Ottime	Mosse medie caso ottimo	Mosse medie caso sub-ottimo
6	814	32,66	101,71
7	912	12,88	31,71
8	930	18,90	61,49
9	924	19,67	71,61

Tabella 6: Numero medio di mosse effettuate da Hill Climbing Laterale

Hill Climbing Laterale (1000 mosse, stato iniziale fisso)			
N	Sol. Ottime	Mosse medie caso ottimo	Mosse medie caso sub-ottimo
6	1000	48,52	0
Hill Climbing Laterale (1000 mosse, stato iniziale casuale)			
N	Sol. Ottime	Mosse medie caso ottimo	Mosse medie caso sub-ottimo
6	937	48,30	1001,93

Tabella 7: Hill Climbing con 1000 mosse laterali per risolvere 6-Regine

Per verificare se l'aumento delle mosse possa permettere di superare ulteriori plateux, sono stati tracciati gli stati in cui l'algoritmo si blocca perché esaurisce tutte le mosse a disposizione.

È stata così ottenuta la figura 6 dalla quale si può vedere come l'algoritmo è rimasto incastrato in un plateaux che corrisponde ad un minimo locale, ovvero un insieme di stati tra loro raggiungibili ai quali corrisponde un minimo locale della funzione di valutazione e che hanno come vicini altri stati dell'insieme oppure stati peggiori. In questo caso l'aumento del numero delle mosse laterali possibili non porterebbe alcuni miglioramento, in quanto l'algoritmo non ha modo di superare questa tipologia di platueax se non spostandosi su uno stato peggiore.

La maggiore difficoltà è quindi causata dalla funzione euristica utilizzata per scegliere le mosse, la quale è caratterizzata da questi minimi locali e plateux minimi che fanno terminare Hill Climbing con una soluzione sub-ottima e probabilmente l'utilizzo di un'euristica differente permette di evitare questi problemi.

In ogni caso c'è da tenere in considerazione che 6-Regine ha solamente 4 soluzioni mentre altri problemi più complessi come 7 e 8-Regine hanno rispettivamente 40 e 92 possibili soluzioni distinte[4].

[4]Questi risultati sono stati ottenuti enumerando tutte le soluzioni di N-Regine risolto come CSP. Maggiori informazioni sono riportate in appendice A

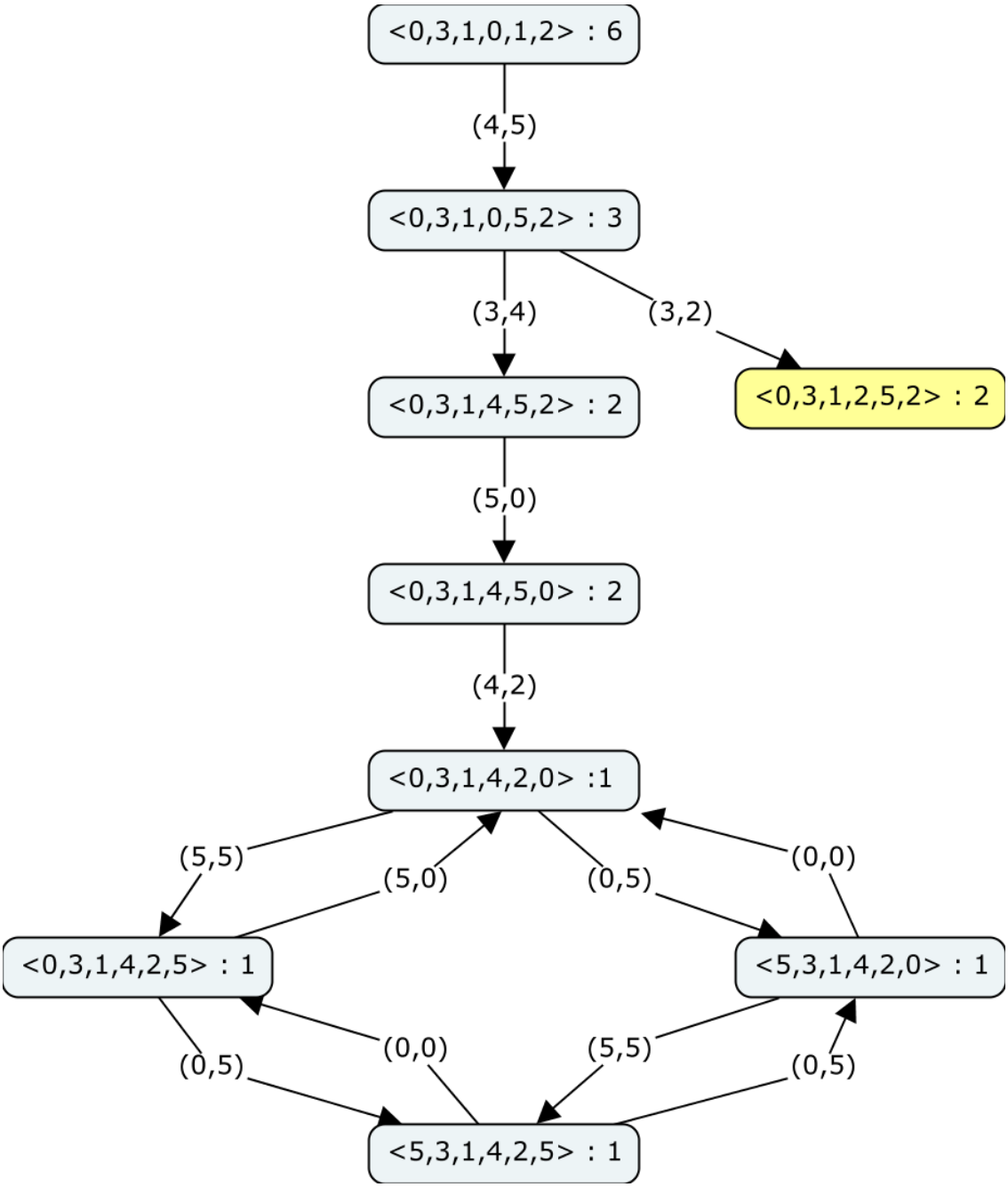

Figura 6: Grafo che rappresenta gli stati visitati da Hill Climbing con 1000 mosse laterali quando incontra un plateaux che coincide con un minimo locale. Lo stato indicato in giallo è un'alternativa che l'algoritmo **avrebbe** potuto prendere per sorpassare il primo plateaux e che forse avrebbe evitato il secondo plateux.

3.6 Simulated Annealing

Come ultima prova è stato eseguito l'algoritmo Simulated Anneling per confrontare le sue prestazioni con quelle ottenute dai vari Hill Climbing.

Sono state quindi eseguite 100 prove utilizzando come funzione di raffreddamento

$$\begin{aligned} temperature\_fn(t) &= 20e^{-0,05t} \text{ se } t < 1000 \\ &= 0 \text{ se } t \geq 1000 \end{aligned}$$

Dai risultati riportati nella tabella 8 si può osservare che l'algoritmo richiede più tempo per l'esecuzione e in alcuni casi non riesce a trovare neanche una soluzione ottima. Questo può essere causato dal ridotto numero di prove così come dalla scelta di una funzione di raffreddamento non ottimale.

Infatti, la funzione di raffreddamento utilizzata è un variante di quella proposta dal libro di riferimento con dei parametri leggermente modificati. Questo perché la funzione proposta produce dei risultati pessimi.

Ad esempio, utilizzando la funzione $termperature\_fn(t)$ per risolvere 6-Regine sono state ottenute 514 soluzioni ottime su 1000 tentativi, mentre con la funzione proposta dal libro, $temperature\_fn\_libro(t)$ sono state ottenute solamente 2 soluzioni ottime.

$$\begin{aligned} temperature\_fn\_libro(t) &= 20e^{-0,005t} \text{ se } t < 100 \\ &= 0 \text{ se } t \geq 100 \end{aligned}$$

L'algoritmo Simulated Anneling riesce quindi ad ottenere risultati migliori di Hill Climbing e HIll Climbing stocastico, richiedendo però più tempo. Hill Climbing laterale risulta comunque migliore, sia in termini di soluzioni trovate che in termini di tempo d'esecuzione, anche nel problema delle 6-Regine, dove Simulated Anneling dovrebbe essere avvantaggiato dal momento che può uscire da un plateaux minimo con una mossa peggiorativa.

	Simulated Annealing (stato fisso)			Simulated Annealing (stato casuale)		
N	Tempo (s)	Sol. Ottime	Val. Sub-ottimo	Tempo (s)	Sol. Ottime	Val. Sub-ottimo
4	0,08554	100	0	0,08479	100	0
5	0,12830	100	0	0,12786	100	0
6	0,18384	59	1	0,18248	58	1
7	0,24730	79	1	0,24532	76	1
8	0,32037	69	1	0,31955	69	1
9	0,40274	63	1	0,40091	62	1,03
10	0,50208	28	1,01	0,49912	48	1,02
11	0,61138	30	1,03	0,60908	32	1,01
12	0,73064	25	1,09	0,72928	26	1,04
13	0,86026	18	1,12	0,85847	30	1,14
14	1,00010	15	1,20	1,00350	26	1,12
15	1,15310	16	1,30	1,15272	14	1,22
16	1,31396	9	1,31	1,31579	12	1,24
17	1,62402	8	1,27	1,62186	8	1,30
18	1,83081	10	1,49	1,82625	5	1,42
19	2,03998	5	1,63	2,03762	8	1,65
20	2,26909	4	1,69	2,26810	3	1,76
21	2,50740	3	1,76	2,50461	4	1,83
22	2,75745	3	1,87	2,75102	5	2,13
23	3,06395	2	2,07	3,06710	2	1,91
24	3,34183	3	2,25	3,34711	0	2,21
25	3,63938	0	2,50	3,64312	0	2,26

Tabella 8: Risultati derivati dall'applicazione di Simulated Annealing. Per ogni problema sono state fatte 100 prove.

4 Conclusioni

Dalle prove effettuate, l'algoritmo di ricerca locale migliore risulta essere l'Hill Climbing con 100 mosse laterali, il quale riesce a trovare quasi sempre una soluzione ottima e in un tempo inferiore rispetto agli altri algoritmi.

Tuttavia bisogna tenere in considerazione che la prova riguarda solamente un problema e sempre con la stessa funzione euristica. Non è quindi detto che quanto ottenuto sia valido in generale.

Inoltre, l'algoritmo Simulated Annealing ha come iper-parametro la funzione di raffreddamento la quale influsisce molto sulla qualità delle soluzioni trovate e quella utilizzata nella prova non è detto che sia la migliore per affrontare il problema delle N-Regine, quindi può essere che una funzione diversa porti il Simulated Annealing ad essere migliore dell'Hill Climbing laterale.

4.1 Consuntivo

Ore	Attività
1	Analisi del codice Python esistente e definizione dell'architettura della soluzione
15	Codifica
1	Pianificazione delle prove
-	Esecuzione automatizzata delle prove
2	Analisi dei risultati ottenuti
1	Confronto con l'approccio CSP
15	Redazione della relazione

A N-Regine come CSP

Il problema delle N-Regine può essere anche formulato come problema di soddisfacimento di vincoli, andando a codificare l'assenza di minacce con dei vincoli definiti sui possibili assegnamenti delle variabili. Una possibile formulazione è:

Variabili:

$$x_i \in \{0 \dots N-1\} \qquad i = 0 \dots N-1$$

Vincoli:

$$\forall i, j \in \{0 \dots N-1\}, \quad i \neq j$$
$$x_i \neq x_j$$
$$x_i - x_j \neq j - i$$
$$x_i - x_j \neq i - j$$

Con il problema così codificato è possibile utilizzare un risolutore di CSP che cerca di produrre un assegnamento di variabili in modo che tutti i vincoli siano soddisfatti.

È stata fatta una prova utilizzando or-tools, un risolutore CSP manutenuto da Google, per verificare l'efficacia di questo approccio e per paragonarlo alla ricerca locale.

È risultato che per risolvere 26-Regine sotto forma di CPS e utilizzando il risolutore di default, sono necessari $846ms$, Andando a modificare il criterio con il quale il risolutore assegna un valore alle variabili in modo che vengano assegnate prima le variabili con il dominio più piccolo e assegnate al valore più basso possibile, si riesce ad ottenere una soluzione in $5ms$ e in circa 5 secondi si ottiene una soluzione per 300-Regine.

Questa differenza di prestazioni deriva dal fatto che il risolutore CSP riesce ad utilizzare i vincoli per effettuare una riduzione dei domini delle variabili, riducendo così lo spazio di ricerca. Inoltre è possibile ottimizzare la strategia di ricerca in modo che vengano assegnate per prime le variabili per cui si ha meno scelta, ovvero con un dominio più piccolo. Così facendo il risolutore riesce a riconoscere prima se il ramo dell'albero che sta valutando porta ad una soluzione o meno.

Con la ricerca locale, l'unica informazione che si ha a disposizione è la funzione euristica, la quale fornisce meno informazioni rispetto ai vincoli di un problema con vincoli. Quindi può succedere che la ricerca locale esplori un'insieme di stati che il risolutore CSP riesce a scartare perché rileva subito che non sono soluzioni. Inoltre, la ricerca locale è influenzata dallo stato iniziale: se vicino allo stato iniziale c'è uno stato a cui corrisponde un minimo locale della funzione euristica, l'algoritmo andrà probabilmente a bloccarsi in quello stato, mentre con un risolutore CSP non c'è questo problema perché lo stato iniziale è sempre lo stato in cui nessuna variabile è assegnata.

Infine, N-Regine rientra nella categoria dei problemi combinatori, problemi che sono facilmente risolvibili con la programmazione a vincoli.

A.1 Enumerazione delle soluzioni di N-Regine

Un'altra caratteristica dei CSP è che una volta definito il modello, questo può essere utilizzato sia per trovare una soluzione ottima che per enumerare tutte le possibili soluzioni.

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
#Sol	1	0	0	2	10	4	40	92	352	724	2680	14200	73712	365596	2279184

Tabella 9: Numero di soluzioni per *N*-Regine

Si è quindi utilizzato lo stesso modello per contare quante sono le soluzioni possibili delle istanze di *N*-Regine affrontate nel progetto. Purtroppo l'enumerazione è molto più complessa in quanto il risolutore deve attraversare tutto l'albero di ricerca e non può fermarsi alla prima soluzione trovata, pertanto l'enumerazione delle soluzioni è stata effettuata fino a 15-Regine.

Il risultato ottenuto è riportato nella tabella **??**, dalla quale è possibile notare la crescita esponenziale del numero di soluzioni e che 6-Regine ha meno soluzioni di 5-Regine.