



UNIVERSITÀ DEGLI STUDI DI CAGLIARI

FACOLTÀ DI INGEGNERIA

Corso di laurea specialistica in ingegneria elettronica

**Metodi innovativi per l'analisi e la valutazione di
funzioni euristiche per algoritmi di ricerca globale**

Relatore:

Prof. Giuliano Armano

Correlatore:

Prof. Carlos Linares López

Tesi di Laurea di:

Filippo Giuseppe Ledda

Anno accademico 2005-2006

Questa tesi¹ è stata realizzata prevalentemente nell'ambito del progetto erasmus con la collaborazione dell'università Carlos III di Madrid e del gruppo PLG (planning and learning group), guidato dal prof. Daniel Borrajo.

Un sentito ringraziamento a tutto il gruppo, per me un esempio di serietà e cortesia. Un ringraziamento particolare al prof. Carlos Linares López, che ha avuto la pazienza di seguirmi e consigliarmi in questi sei stupendi mesi a Madrid.

Voglio inoltre ringraziare tutte le persone speciali che ho avuto la fortuna di conoscere e che hanno allietato questi mesi in Spagna, tra i tanti con affetto particolare per Andrea, Carlo, Enza, Federica, Paolo, Sabrina.

Grazie infine alla mia famiglia per aver permesso di completare i miei studi con serenità.

¹ Documento redatto con Openoffice [OPENOFFICE_SITE].

INDICE

Introduzione.....	3
1 Nozioni preliminari sulla ricerca e le funzioni euristiche.....	6
1.1 Problem solving.....	7
1.1.1 Definizione di un problema di ricerca.....	7
1.1.2 Problemi classici risolvibili con algoritmi di ricerca.....	8
1.1.2.1 N-Puzzle.....	8
1.1.2.2 Il cubo di Rubik.....	10
1.1.2.3 Il labirinto.....	10
1.1.2.4 Travelling Salesman Problem (TSP).....	11
1.2 Algoritmi di ricerca.....	12
1.2.1 Difficoltà di un problema di ricerca.....	12
1.2.1.1 Branching Factor.....	13
1.2.1.2 Distanza dalla soluzione ottima.....	13
1.2.2 Ricerca non informata.....	13
1.2.2.1 Ricerca in ampiezza (breadth-first).....	14
1.2.2.2 Ricerca in profondità (depth-first).....	14
1.2.2.3 Ricerca a profondità limitata.....	14
1.2.2.4 Ricerca ad approfondimento iterativo (iterative-deepening, ID).....	15
1.2.2.5 Ricerca bidirezionale.....	15
1.2.2.6 Comparativa tecniche di ricerca non informata.....	15
1.2.3 Ricerca informata.....	16
1.2.3.1 Algoritmi Best-First.....	16
1.2.3.1.1 Greedy search.....	16
1.2.3.1.2 Ricerca a costo uniforme.....	16
1.2.3.1.3 A*.....	17
1.2.3.1.4 Weighted-A*.....	17
1.2.3.2 Algoritmi a memoria lineare.....	17
1.2.3.2.1 Iterative-Deepening-A* (IDA*).....	18

1.2.3.2.2 Recursive Best-First Search (RBFS).....	18
1.2.3.3 Algoritmi Bidirezionali.....	19
1.2.3.4 Algoritmi di perimetro.....	19
1.2.3.4.1 BIDA*.....	20
1.2.3.5 Algoritmi real-time.....	20
1.2.3.5.1 Real-time A* (RTA*).....	21
1.2.3.5.2 Learning real-time A* (LRTA*).....	21
1.2.3.5.3 Oltre LRTA*.....	22
1.2.3.6 Ricerca locale euristica.....	22
1.2.3.6.1 Hill-Climbing.....	22
1.2.3.7 Complessità degli algoritmi di ricerca informata.....	22
1.2.4 Ricerca locale stocastica (stochastic local search, SLS).....	23
1.3 Le funzioni euristiche.....	24
1.3.1 Proprietà generali delle funzioni euristiche.....	25
1.3.1.1 Ammissibilità.....	25
1.3.1.2 Monotonicità.....	25
1.3.1.3 Consistenza.....	26
1.3.1.4 ϵ -ammissibilità, e-ammissibilità.....	26
1.3.1.5 P-ammissibilità e “sovrastime ammissibili”.....	27
1.4 Definizione di funzioni euristiche.....	27
1.4.1 L'euristica come soluzione del “problema rilassato”.....	28
1.4.2 Generazione automatica di funzioni euristiche.....	29
1.4.3 Euristiche da pattern database.....	29
1.4.4 Tecniche subsimboliche.....	30
1.4.5 Miglioramento di euristiche da perimetro.....	31
1.5 Esempi di implementazione e euristiche di alcuni problemi classici.....	31
1.5.1 N-Puzzle.....	31
1.5.1.1 Misplaced-tiles (tiles-out-of-place).....	32
1.5.1.2 Distanza di manhattan.....	33
1.5.1.3 Linear-conflict.....	33
1.5.1.4 Corner-tiles.....	36
1.5.1.5 Corner-deduction.....	37
1.5.1.6 Last-moves.....	38
1.5.1.7 Non-linear-conflict.....	39
1.5.1.8 Combinazione delle tecniche di manhattan-correction.....	40
1.5.2 Labirinto.....	41
1.5.2.1 Distanza di manhattan.....	41
2 Valutazione di euristiche: definizione del problema e stato dell'arte.....	42
2.1 La valutazione delle funzioni euristiche: considerazioni preliminari.....	44
2.1.1 Framework sperimentale.....	48
2.2 Dominanza informativa di una funzione euristica.....	50
2.3 Metodi basati sulla funzione di distribuzione di probabilità.....	51
2.3.1 La funzione di distribuzione di probabilità.....	51
2.3.2 Dominanza stocastica di una funzione euristica.....	51
2.3.2.1 Risultati sperimentali.....	52
2.3.3 Confronto di funzioni euristiche non ammissibili.....	60
2.3.3.1 Analisi stocastica delle euristiche non ammissibili.....	61
2.3.3.1.1 Risultati Sperimentali.....	61
2.3.4 Metodo di Korf-Reid per il computo dei nodi espansi dall'algoritmo IDA*.....	62
2.3.4.1 La equilibrium distribution.....	64
2.3.4.2 Calcolo del branching factor per l'N-Puzzle.....	65
2.3.4.3 Osservazioni.....	66

2.3.4.4	Analisi sperimentali.....	66
2.4	Metodi matematici per il calcolo dei nodi generati (cenni).....	72
2.5	Calcolo di tempo – memoria.....	72
2.5.1	Framework per l'analisi di algoritmi e funzioni euristiche.....	72
2.5.2	Metodo per il calcolo del tempo per l'algoritmo BIDA*.....	73
2.6	Metodi di valutazione allo stato dell'arte per gli algoritmi di ricerca locale stocastica.....	74
2.6.1	Search Landscape analysis.....	75
2.6.1.1	Tipi di posizioni e loro distribuzione.....	75
2.6.1.2	Numero, densità e distribuzione dei minimi locali e plateau.....	75
2.6.2	Fitness-distance analysis.....	75
2.6.3	Landscape Ruggedness.....	76
2.7	Studi sulle euristiche nel planning.....	76
3	Metodi innovativi per la valutazione di funzioni euristiche.....	77
3.1	Estensione dei metodi usati per euristiche SLS alla ricerca euristica.....	78
3.1.1	Search Landscape analysis.....	78
3.1.1.1	Numero, densità e distribuzione dei minimi locali e plateau.....	81
3.1.1.2	Risultati sperimentali.....	82
3.1.2	Stima della profondità della soluzione basata sulla distribuzione di minimi locali e plateau.....	89
3.1.2.1	Risultati sperimentali.....	94
3.1.3	Fitness-distance analysis.....	98
3.1.3.1	Fitness-distance correlation.....	98
3.1.3.2	Risultati sperimentali.....	99
3.1.4	Landscape ruggedness.....	100
3.1.4.1	Risultati sperimentali.....	101
3.2	La matrice di transizione.....	102
3.2.1	Calcolo di errore basato su catene di Markov per l'algoritmo Hill-climbing.....	102
3.2.1.1	Concetti base sulle catene di Markov.....	103
3.2.1.2	Definizione degli stati Markoviani – clusterizzazione dello spazio.....	103
3.2.1.3	Descrizione dell'algoritmo Hill-Climbing con le catene di Markov.....	103
3.2.1.4	Risultati sperimentali.....	104
3.2.2	Metodo per la stima dei nodi generati da IDA* e profondità della soluzione.....	106
3.2.2.1	Stima della profondità media della soluzione.....	110
3.2.2.2	Applicazione stocastica del metodo.....	110
3.2.2.3	Risultati sperimentali.....	111
4	Conclusioni e sviluppi futuri.....	113
	Appendice A: Campionamento dello spazio degli stati.....	115
	Appendice B: Clusterizzazione dello spazio degli stati.....	117
	Appendice C: Risultati sperimentali: tempo e nodi espansi.....	119
	Bibliografia.....	123

PREMESSA

La ricerca è una delle tecniche storiche di risoluzione dei problemi di quella branca dell'informatica denominata (forse con un po' di presunzione!) intelligenza artificiale. La funzione euristica è ciò che porta un processo di ricerca per il problem solving a essere per così dire "intelligente", intendendo per intelligenza la capacità di risolvere dei problemi come facciamo noi esseri umani, ossia avvalendosi di conoscenza ricavabile dalla formulazione stessa del problema e da altre fonti assimilate con l'esperienza.

In questa tesi si analizza principalmente come sia possibile valutare l'efficacia delle funzioni euristiche. Si tratta di un compito non affatto semplice e riguardo al quale poco lavoro è stato svolto fino ad ora. Si presenteranno in questo documento le possibilità offerte dallo stato attuale della scienza, a cui si aggiungono una serie di metodi innovativi ideati in questi mesi dedicati allo studio delle funzioni euristiche. Il tutto è doverosamente accompagnato da una ricca presentazione di esperimenti che hanno lo scopo di verificare l'efficacia e le possibilità di quanto presentato teoricamente, e insieme stimolare il lettore in vista di possibili applicazioni future delle tecniche descritte.

Valutare l'efficacia di una funzione euristica è sicuramente auspicabile prima di intraprendere ricerche in spazi di dimensioni elevate: un'euristica efficace sarà in grado di potare sapientemente i rami sterili dell'albero di ricerca, lasciando poco spazio a errori e indecisioni, che si traducono in una inefficiente ricerca in zone dello spazio lontane dalla via per la soluzione. Un'applicazione decisamente interessante offerta dalla valutazione delle euristiche è la possibilità di scegliere l'alternativa migliore da un set di euristiche dalla complessità ed efficacia crescente; ad esempio attraverso il moderno concetto di algoritmi di perimetro è possibile avere a disposizione una collezione di questo tipo per pressoché qualunque problema di ricerca invertibile, e uno dei problemi più interessanti è proprio

individuare quale sia il miglior compromesso tra efficienza (intesa come tempo necessario a calcolare atomicamente l'euristica) ed efficacia, per poter effettuare la scelta più conveniente.

Come lascia intendere il titolo, ci si riferirà salvo diversa indicazione ad euristiche nell'ambito degli algoritmi di ricerca globale. Diverse sono le tecniche algoritmiche per risolvere un problema attraverso la ricerca, ognuna delle quali ha le sue caratteristiche peculiari e applicazioni. Gli algoritmi di ricerca globale possono essere considerati la famiglia di più classici metodi di ricerca, così detti perché mirano a eseguire una ricerca il più possibile esaustiva esaminando globalmente le possibili alternative.

Il documento si articola in tre sezioni fondamentali: la prima è un'esaustiva e sintetica introduzione preliminare alla ricerca e le euristiche presenta tutto quello che è auspicabile sapere prima di intraprendere la lettura delle sezioni dedicate al tema centrale di questo studio, la valutazione delle funzioni euristiche. Le sezioni dedicate specificatamente a questo tema sono invece due, una dedicata ai metodi già noti, l'altra riguarda tutte le idee che sono state sviluppate in questi mesi dedicati allo studio delle funzioni euristiche. Questi due capitoli si dividono in sottosezioni separate, ognuna dedicata alla descrizione di uno specifico metodo, seguita quando ritenuto opportuno da una sottosezione dedicata ai risultati sperimentali ottenuti. Anche nella parte dedicata allo stato dell'arte non manca il lavoro sperimentale, dedicato soprattutto a definire l'applicazione pratica dei suddetti metodi.

Infine il doveroso capitolo conclusivo e alcune utili appendici, che riguardano dei temi che riguardano parallelamente diverse sezioni del testo.

Nell'introduzione iniziale, e, più in dettaglio, all'inizio di ogni capitolo, è riassunto il contenuto del documento sezione per sezione, al fine di facilitare una lettura che potrebbe altrimenti risultare un po' dispersiva per la quantità di argomenti trattati, spesso non collegati fra loro da un unico filo conduttore.

Lo scopo finale di questa tesi non sarà dunque di mettere la parola fine a un argomento ancora così poco esplorato, quanto di presentare il più possibile obiettivamente un'ampia tavolozza di strumenti utilizzabili al fine di valutare l'efficacia di una funzione euristica e/o di studiarne il comportamento. Per questo motivo non si è evitato di fornire dettagli implementativi spesso trascurati, ma necessari in un qualsiasi contesto pratico; la lettura non dovrebbe in ogni caso risultarne appesantita essendo questi dettagli descritti nelle sezioni dedicate riguardanti i risultati sperimentali.

INTRODUZIONE

La ricerca euristica è un metodo generale ed efficace di soluzione di problemi nell'intelligenza artificiale. Essa si basa sulla combinazione di due componenti fondamentali: una strategia di ricerca e una funzione euristica. In questo lavoro si studia come valutare l'efficacia delle funzioni euristiche per gli algoritmi di ricerca globale.

Ma cosa sono in realtà le euristiche? Prendiamo ad esempio la definizione data da Pearl: le euristiche sono metodi, criteri, o principi per decidere quale fra diverse possibilità di azione sia più promettente per raggiungere un qualche obiettivo. Rappresentano un compromesso tra due fondamentali requisiti: devono essere semplici (e dunque efficienti) ma allo stesso tempo devono essere in grado di discriminare correttamente tra scelte buone e meno buone. Per questo e per altri motivi che si vedranno la valutazione delle euristiche è un problema di notevole complessità pratica e concettuale: è possibile costruire euristiche notevolmente precise ma il cui utilizzo non porta alcun vantaggio alla ricerca per via della loro inefficienza. Un semplice esempio spesso citato per la sua paradossalità è il seguente: un'euristica senza errori può essere banalmente ottenuta eseguendo una ricerca “di nascosto”; ovviamente questo non gioverebbe affatto per il tempo necessario a effettuare questa sotto-ricerca: a cosa può servire infatti un'euristica che non migliori l'efficienza della ricerca?

In questa tesi è stato fatto un lavoro di ricerca teorica e sperimentale su diverse tecniche, alcune già incontrate in letteratura e diverse altre innovative. Infatti non si incontra molto materiale già pubblicato riguardo alla valutazione delle euristiche: il punto di riferimento per quanto riguarda l'argomento è certamente il libro *Heuristics* di Judea Pearl [Pearl84], nel quale sono presenti ben tre capitoli riguardo alla valutazione delle euristiche. Si incontrano poi una manciata di lavori che sono stati pensati principalmente per valutare gli algoritmi, ma che possono essere senza un eccessivo sforzo concettuale adattati alle euristiche. Molto più

lavoro invece è stato fatto di recente riguardo agli algoritmi di ricerca locale stocastica; si vedrà come alcune delle tecniche esistenti riguardo alle funzioni di valutazione possano essere applicati alle euristiche per algoritmi di ricerca globale.

Molta enfasi è stata posta sulla parte sperimentale: alla descrizione di ogni metodo è associata una sottosezione in cui sono riportati i più rilevanti risultati sperimentali. Perché se è difficile valutare le euristiche ancora più difficile è valutare i metodi di valutazione delle euristiche. Queste infatti hanno un comportamento difficilmente prevedibile; non è affatto semplice prevedere a priori gli errori di una funzione euristica e gli effetti che questi avranno sulle prestazioni di un algoritmo. D'altra parte un'analisi a posteriori ha un'utilità limitata dal fatto che il risultato ottenuto per un'istanza di un problema è per esperienza poco generalizzabile. Molta attenzione deve essere fatta in qualsiasi generalizzazione, ogni problema ha le sue caratteristiche particolari che lo rendono imprevedibilmente diverso da tutti gli altri, anche nel caso di diverse formulazioni dello stesso problema. Di certo se i problemi non fossero così difficili e imprevedibilmente vari non sarebbe neanche tanto interessante e utile trovare sempre nuovi modi per risolverli!

Ora si vede un po' più in dettaglio il contenuto delle varie sezioni.

Il primo capitolo, *Nozioni preliminari sulla ricerca e le funzioni euristiche*, è pensato per aiutare il lettore a inquadrare il contesto della ricerca euristica allo stato dell'arte e a definire tutti i concetti necessari a comprendere appieno il contenuto delle analisi successive. Sono definiti i concetti fondamentali del problem solving attraverso la ricerca e le proprietà basilari sulle funzioni euristiche. Sono inoltre presentate le più importanti strategie di ricerca e alcuni algoritmi che implementano queste idee, dando particolare enfasi a quelli maggiormente citati nei capitoli successivi e/o che presentino caratteristiche interessanti per il modo che hanno di utilizzare le informazioni delle f.e.¹. In secondo luogo sono descritti i principali metodi che possono essere usati per calcolare le euristiche. Infine sono presentati alcuni problemi classici risolvibili con la ricerca euristica, e in particolare, per il puzzle di Sam Loyd (che si indicherà nella sua definizione più generale come N-Puzzle) è presente la descrizione di alcune euristiche più o meno note, poi citate intensivamente nelle sperimentazioni. Questo problema, nelle sue diverse dimensioni, è quello che è stato principalmente utilizzato nelle sperimentazioni dei metodi di valutazione delle euristiche; infatti è possibile definire una discretamente varia classe di f.e. da confrontare, e le ridotte dimensioni delle più semplici formulazioni del problema permettono di eseguire delle ampie sperimentazioni statistiche che sono di grande utilità per comprendere aspetti importanti anche in istanze più complesse. Talvolta è stato utilizzato anche il comune labirinto, che seppure non disponga di una altrettanto interessante varietà di euristiche ha delle proprietà interessanti che lo accomunano o che lo distinguono rispetto al puzzle di Sam Loyd.

Il secondo capitolo è dedicato allo stato dell'arte riguardo alla valutazione delle funzioni euristiche. Dopo alcune definizioni sull'informatività da tempo note è presente la descrizione di un metodo a priori per il calcolo dei nodi espansi dall'algoritmo IDA*, l'algoritmo di ricerca allo stato dell'arte sicuramente più utile per le sue performance e semplicità implementativa. Sono descritti poi tutti i riferimenti forse secondari ma comunque di interesse, e infine i metodi allo stato dell'arte nel mondo della ricerca locale stocastica (*stochastic local search*, SLS), ripresi e riadattati nel successivo capitolo alla ricerca euristica globale "classica". In questo capitolo sono presentate alcune sperimentazioni inedite il cui

1 Funzioni euristiche. Spesso si userà l'abbreviazione per evitare una ripetizione veramente insistente in questo documento.

scopo primario è inquadrare il campo di applicazione delle metodologie più interessanti. La forma di presentazione preferita è quella grafica, ma non mancano tabelle e numeri che potrebbero ritornare utili a chi volesse cimentarsi nell'utilizzo dei metodi proposti. Potrebbe destare qualche perplessità il fatto che le sperimentazioni siano fatte soltanto per problemi relativamente semplici. La perplessità sarebbe giustificata dal fatto che per problemi di questo genere non è necessaria alcuna analisi delle euristiche visto che già le più semplici tra quelle già presenti permettono di risolverne tutte le istanze in tempi relativamente ridotti. Bisogna però tenere conto del fatto che lo scopo non è nel nostro caso tanto applicare i diversi metodi a un caso reale quanto invece inquadrare l'applicazione dei metodi proposti e offrirne una possibile comparazione.

Nel terzo capitolo finalmente si presentano i veri e propri studi inediti realizzati per questa tesi. Potrebbe sorprendere la quantità di diverse metodologie proposte, ma si è voluto cercare di proporre il maggior numero possibile idee in modo da fornire una discreta varietà di strumenti che possono aggiungersi e completare quelli già esistenti. Ogni sottosezione è dedicata a un metodo diverso e si conclude con i risultati più interessanti delle numerose sperimentazioni eseguite in questi mesi.

Infine tre appendici di fondamentale importanza, riguardanti campionamento e clusterizzazione dello spazio degli stati, e ai risultati sperimentali.

Nell'Appendice A, dedicata al campionamento dello spazio degli stati, sono descritte nel dettaglio le diverse strategie di sampling menzionate nelle varie sezioni dedicate alle sperimentazioni; se infatti il campionamento non è un concetto necessario alla comprensione teorica dei metodi è invece fondamentale all'atto dell'applicazione pratica, ed è spesso una fase essenziale da cui dipende il buon esito o meno di un'analisi. Sono presentate alcune tecniche non incontrate in letteratura, che talvolta possono dimostrarsi più efficaci del campionamento casuale, sicuramente il più ovvio dal punto di vista statistico.

L'appendice B è invece dedicata alle tecniche di clusterizzazione dello spazio degli stati utilizzate nelle sperimentazioni. Si tratta solo di alcune semplici tecniche, e in realtà molto lavoro deve essere svolto ancora per capire quali siano le strategie migliori per raggruppare i nodi con caratteristiche di comportamento simili dal punto di vista della funzione euristica.

Ultima della serie l'appendice C, in cui si presentano risultati sperimentali sulla ricerca risolvendo alcune piccole istanze di problemi con diverse euristiche, gli stessi analizzati con i diversi metodi proposti. Sono tabulati i reali risultati sui nodi espansi e sul tempo di esecuzione, in modo da fornire un'utile comparazione con le analisi effettuate per mezzo delle tecniche di valutazione descritte.

1 NOZIONI PRELIMINARI SULLA RICERCA E LE FUNZIONI EURISTICHE

Questo secondo capitolo è dedicato alla presentazione dei concetti di base della ricerca euristica, i suoi scopi e le proprietà fondamentali. Innanzitutto sono presentati brevemente i motivi del *Problem Solving*, con una breve panoramica di problemi comunemente risolti tramite la ricerca. Segue la formulazione basilare di un problema nella ricerca globale. Dunque si vedono per la prima volta due concetti di fondamentale importanza nella valutazione degli algoritmi di ricerca ad albero: il fattore di ramificazione (*branching factor*) e la distanza della soluzione ottima.

Si continua e degli algoritmi di ricerca non informata più noti, e finalmente si arriva alla definizione delle funzioni euristiche e delle loro proprietà fondamentali, con alcuni tra i più interessanti e/o usati algoritmi di ricerca informata. È inoltre presentato brevemente il concetto di ricerca locale stocastica, di cui non si tratterà direttamente ma i cui studi sono stati un utile spunto per alcune metodologie di valutazione delle euristiche. Molto del materiale in questa sezione si può trovare in un qualunque testo di intelligenza artificiale, ad esempio [RussellNorvig1998].

Infine una descrizione delle diverse metodologie concrete di calcolo delle euristiche, cui segue una presentazione delle euristiche implementate negli esperimenti di questa tesi.

1.1 Problem solving

Uno dei principali obiettivi dell'intelligenza artificiale è stato fin dall'inizio quello di risolvere problemi. Una delle strategie di maggior successo nell'affrontare i problemi è la ricerca. Esistono tre categorie fondamentali di problemi risolvibili con la ricerca:

- **Problemi di ricerca di percorso a singolo agente**

Si tratta di trovare un percorso che permetta di arrivare da uno stato A a uno stato B, senza che intervenga alcuna modificazione esterna ad alterare lo stato. Problemi di questo tipo, oltre al classico labirinto, sono tra gli altri cubo di Rubik, N-Puzzle e travelling-salesman-problem.

- **Giochi a due contendenti**

Problemi a due contendenti sono ad esempio i giochi degli scacchi e othello. Attualmente i migliori giocatori di scacchi e othello sono programmi per computer.

- **Problemi di soddisfacimento di vincoli**

La soluzione di un problema di questo tipo è il soddisfacimento di una serie di vincoli. Un esempio classico è il problema delle otto regine, e una nutrita serie di problemi reali (ad esempio i problemi di *planning*).

Benché non tutti i concetti che si vedranno sulla valutazione delle euristiche siano esclusivamente riferibili a questi, ci si occuperà esclusivamente della prima categoria di problemi.

1.1.1 Definizione di un problema di ricerca

Un problema di ricerca ben formulato può essere definito attraverso una tupla costituita da:

- Uno spazio degli stati S (rappresentato da un *grafo*)
- Uno stato iniziale S_0 (rappresentabile da un *nodo* N_0^1)
- Un *goal-test*. Restituisce valore booleano *Vero* quando applicato a uno stato obiettivo.
- Un set di operatori
- Funzione di path-cost², $g(n)$

Gli operatori, applicati a un nodo, permettono di generare altri nodi, detti *figli*, a partire da quello di partenza (detto a sua volta *padre* o *genitore*). Il processo di applicare un insieme di operatori a un nodo è chiamato *espansione* dello stesso.

Un processo di ricerca consiste nell'applicare operatori partendo dallo stato iniziale, fino a ottenere un nodo per cui abbia successo il goal-test. I diversi algoritmi di ricerca si contraddistinguono per la strategia con la quale viene scelto l'ordine di espansione dei nodi.

1 Lo stesso stato potrebbe essere rappresentato da nodi distinti nel processo di ricerca.

2 Alcuni autori non riportano questa funzione tra le caratteristiche essenziali perché costo degli archi potrebbe essere a rigore sottinteso nel concetto generale di grafo; in questo caso si vuole porre l'accento su una caratteristica importante della ricerca.

Dal punto di vista operativo è importante considerare il concetto di codice da associare a uno stato sul calcolatore: questo codice deve essere in grado di rappresentare univocamente qualsiasi stato del problema in un nodo del grafo.

1.1.2 Problemi classici risolvibili con algoritmi di ricerca

Segue una breve descrizione di alcuni problemi “giocattolo” che spesso sono utilizzati per applicazioni e test di tecniche di problem-solving. L'utilizzo di giochi al posto di problemi cosiddetti “reali” è una prassi ormai consolidata che trova le sue ragioni nella maggiore semplicità di implementazione a fronte di una complessità facilmente variabile e controllabile che può andare dall'estremamente semplice all'impossibilmente difficile con pochissimi accorgimenti di programmazione. I giochi presentano dunque tutta la serie di caratteristiche che possono apparire nei problemi reali ma con una maggiore generalità e controllabilità.

1.1.2.1 N-Puzzle

L'N-Puzzle, comunemente conosciuto in italiano nella sua formulazione più classica col nome di *gioco del quindici*, è un rompicapo inventato dal famoso creatore di enigmi matematici Sam Loyd ormai nel lontano 1870. Come tutti i giochi di successo, si basa delle regole immediatamente comprensibili da chiunque, ma che permettono di avere un'elevata complessità e numerosissime situazioni diverse di gioco. Si tratta di risistemare in una griglia di dimensione $n \times m$ dei tasselli numerati in maniera crescente fino a $N = n \times m - 1$, avendo a disposizione una casella vuota nella quale far scivolare le tessere adiacenti.

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Figura 1.1.: Il gioco del quindici o 15-puzzle – possibile configurazione obiettivo

Il gioco ebbe inizialmente un enorme successo grazie a una originale iniziativa di Sam Loyd: egli offrì infatti 1000 dollari a chi per primo sarebbe riuscito a risolvere il gioco partendo da una configurazione identica all'originale, ma con due tessere invertite. Tantissima gente si convinse di essere riuscita a risolvere il rompicapo, contribuendo ad alimentare la curiosità intorno al curioso concorso, ma nessuno riuscì a ripetere l'operazione allo scopo di richiedere l'ambito premio. Solo molto più tardi si dimostrò quello che Sam Loyd già sapeva e che fu dimostrato matematicamente solo molti anni più tardi: la configurazione proposta non poteva essere risolta perché appartenente a una sezione di grafo completamente disgiunta rispetto a quella cui appartiene l'obiettivo [Johnson1879], [Wilson1974].

Ci si limita a citare la condizione necessaria e sufficiente affinché una configurazione sia

risolvibile: il numero di inversioni di tessere rispetto alla soluzione deve avere parità identica al numero totale di movimenti che le tessere dovrebbero compiere per arrivare alla soluzione potendosi muovere liberamente (questa definizione è quella che si vedrà più avanti indicata come distanza di manhattan).

$$\text{solvable} \Leftrightarrow \text{numberOfInversions mod } 2 = \text{manhattan mod } 2 \quad (1.1)$$

Qualsiasi puzzle ha uno spazio di possibili configurazioni divisibili in due grafi disgiunti secondo la condizione (1.1). L'operatore *mod* è il resto di una divisione intera.

Le proprietà che invece rendono il problema interessante dal punto di vista del problem-solving e dell'IA sono la crescita esponenziale della dimensione dello spazio degli stati e della difficoltà del problema¹ con la dimensione della griglia. Se *n* è il numero di caselle della griglia il numero totale di permutazione è infatti pari a $n!$; il numero totale di configurazioni risolvibili sarà pari alla metà in base alla (1.1).

Osservando la tabella 1.1 è possibile rendersi conto di come rapidamente cresca il numero di stati:

<i>Puzzle</i>	<i>N</i>	<i>Stati</i>	<i>Ordine</i>
2x2	3	12	10^1
3x2	5	360	10^2
3x3	8	181440	10^5
4x3	11	239500800	10^8
4x4	15	10461394944000	10^{13}
5x4	19	$1.216451 \cdot 10^{18}$	10^{18}
5x5	24	$7.7556050 \cdot 10^{24}$	10^{24}
6x6	35	$1.8599666 \cdot 10^{41}$	10^{41}

Tabella 1.1. Dimensione dello spazio degli stati per diverse dimensioni dell'N-puzzle

Per farsi un'idea della portata di queste dimensioni proviamo a pensare che un ipotetico agente “sfortunato” che dovesse visitare tutti i nodi una volta prima di trovare la soluzione, nel momento in cui impiegasse 1 nanosecondo a risolvere un'istanza del 8-Puzzle², impiegherebbe andando alla stessa velocità 0,36 microsecondi per risolvere il 15-puzzle, 135 anni per il 24-Puzzle e ben 3 miliardi di miliardi di anni per il 35-puzzle!

Allo stato attuale con le tecniche più evolute si riesce a trovare la soluzione ottima senza nessun problema qualsiasi istanza del 15-puzzle, mentre i tempi sono ancora elevati con istanze mediamente difficili del 24-puzzle. Quest'ultimo può essere affrontato utilizzando complesse euristiche basate su pattern database.

Il fatto che rende incredibilmente interessante questo problema allo scopo delle analisi di questa tesi è, oltre ai motivi già elencati, la presenza di una classe relativamente ampia di funzioni euristiche da analizzare.

1 Si vedrà più avanti una definizione operativa di difficoltà di un problema

2 Notare che la velocità ipotizzata è irrealistica per quanto questa è piccola su qualsiasi macchina attuale

1.1.2.2 Il cubo di Rubik

Il Cubo di Rubik è un celebre rompicapo inventato dal professore di architettura e scultore ungherese Ernő Rubik nel 1974.

La versione standard è costituita da un cubo $3 \times 3 \times 3$, con ogni sottocubo, o cubetto, contraddistinto da un diverso colore. Lo scopo del gioco è di risalire alla posizione originale dei cubetti portando il cubo ad avere per ogni faccia un colore uguale. Il cubo può assumere ben 43.252.003.274.489.856.000 combinazioni possibili (circa $4 \cdot 10^{20}$), solo una delle quali è quella corretta.

Per arrivare alla soluzione è possibile ruotare ogni faccia $3 \times 3 \times 1$ di 90, 180 o 270 gradi relativamente al resto del cubo.

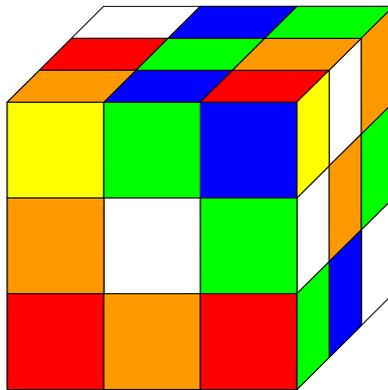


Figura 1.2.: Il cubo di Rubik

Inizialmente progettato da Rubik a scopi didattici, all'inizio si diffuse solo tra i matematici ungheresi, interessati ai problemi statistici e teorici che il cubo poneva. Qualche anno più tardi un matematico inglese scrisse su quest'oggetto un articolo che portò la sua fama fuori dai confini dell'Ungheria. Nel giro di pochi anni, il cubo di Rubik invase i negozi europei ed americani, diventando il rompicapo più venduto della storia.

Tuttora è oggetto di competizioni a livello mondiale, nelle quali dei veri esperti si confrontano nella sfida di chi riesce a risistemare in minor tempo le diverse combinazioni. Tipicamente i migliori contendenti di questo tipo di gare riescono a risistemare qualsiasi permutazione del cubo in meno di quindici secondi, tempi tuttora impensabili per un'applicazione di intelligenza artificiale.

Non si realizzeranno esperimenti con questo problema, per due motivi principali: innanzitutto non è presente una serie di euristiche da confrontare; inoltre perché non è possibile avere, come per l'*N*-Puzzle, versioni dello stesso abbastanza semplici da poter eseguire le analisi statistiche fondamentali per comprendere gli aspetti dell'analisi delle funzioni euristiche.

1.1.2.3 Il labirinto

Il labirinto (in inglese *maze*) è certamente il più classico problema di ricerca di percorso. Si tratta di trovare un percorso che permetta a un agente di arrivare da un punto all'altro di una

griglia. Il problema è complicato dal fatto che non sempre è possibile raggiungere un punto da quello immediatamente adiacente.

Sono state ideate versioni di labirinto più complicate, nelle quali ad esempio è necessario raccogliere dei “tesori” prima di arrivare alla soluzione finale, o con punti di non ritorno. Si possono dunque riprodurre tutte le situazioni reali di un problema di ritrovamento di percorso di un agente mobile.

Benché sia stato uno dei primi problemi affrontati per mezzo della ricerca, non è tuttora utilizzato intensivamente dai ricercatori che si occupano di algoritmi di ricerca euristica. Il principale problema di un labirinto è la sua rappresentazione, che non potrà mai essere compatta come quella di un N-Puzzle o del cubo di Rubik. Il ricercatore A che volesse ripetere gli esperimenti del ricercatore B sarebbe costretto a farsi consegnare dal primo una matrice $n \times m$ con un flag per ogni transizione tra nodi adiacenti che indichi se percorribile o meno. Un'altra possibile rappresentazione è quella della lista di transizioni di ogni nodo. Non è comunque completamente messo in un angolo l'uso dei labirinti nei test degli algoritmi di ricerca: ad esempio sono usati tra gli altri in [KaindlKainz1997].

Un aspetto interessante dal punto di vista dello studio delle euristiche è che la stessa euristica può comportarsi in maniera estremamente diversa per distinte istanze di labirinto.

Una importante definizione è quella di *perfezione* di un labirinto: un labirinto si dice *perfetto* quando è presente sempre uno e un solo percorso che permetta di arrivare da un punto all'altro. Equivalentemente, un labirinto è perfetto quando non ha zone inaccessibili, percorsi circolari o aree aperte.

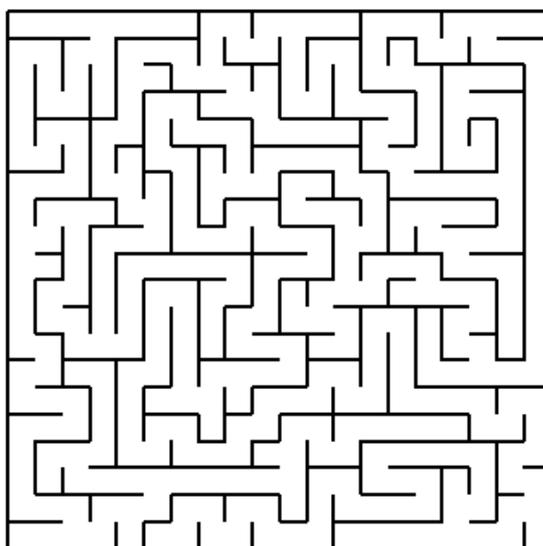


Figura 1.3.: Un labirinto perfetto

Esistono metodi per generare automaticamente dei labirinti perfetti della dimensione desiderata. Si realizzeranno alcuni esperimenti utilizzando piccoli labirinti quando sia necessario porre in risalto alcune caratteristiche che non sarebbero evidenziate usando l'N-Puzzle.

1.1.2.4 Travelling Salesman Problem (TSP)

Il venditore ambulante che debba visitare una serie di città sarà interessato a ritrovare il

percorso più breve che consenta di visitarle tutte, senza passare inutilmente più volte nella stessa città. La soluzione del TSP consiste più astrattamente nel trovare il percorso migliore che consente di visitare una e una sola volta una serie N di nodi in un grafo completamente connesso non orientato, in cui a ogni transizione è associato un costo determinato. È noto per essere un problema NP-completo, in cui ogni algoritmo conosciuto impiega un tempo esponenziale rispetto al numero di nodi per trovare la soluzione nel caso peggiore.

Sicuramente si tratta di un problema almeno altrettanto interessante che l' N -Puzzle dal punto di vista della varietà di euristiche implementabili, ma dalla più difficile e meno compatta implementazione. In più rispetto a questo ha la caratteristica certamente interessante del costo variabile delle transizioni, caratteristica che d'altra parte è stata un po' messa da parte nei lavori di Korf, che preferisce considerare problemi con costo del passo costante e unitario.

Naturalmente ciò non toglie nulla alla portata del problema, e, nonostante non siano stati eseguiti esperimenti con esso, potrebbe essere suggerito come futura applicazione di alcuni dei metodi proposti in questo lavoro.

Un'altra caratteristica che contraddistingue questo problema da tutti gli altri visti è che richiede nella sua stessa formulazione l'ottimizzazione della soluzione: una soluzione è accettabile solo quando essa sia la migliore possibile. La distanza della soluzione è inoltre nota a priori, in quanto pari al numero di città da visitare.

1.2 Algoritmi di ricerca

Un algoritmo di ricerca definisce la strategia di controllo dell'applicazione degli operatori. La diversa strategia determina alcune proprietà di un algoritmo di ricerca: un algoritmo è detto *completo* quando è in grado di visitare qualunque stato dello spazio di ricerca. Un algoritmo completo e che non visiti mai per due volte lo stesso stato è detto *sistematico*. Un algoritmo di ricerca sistematico garantisce il ritrovamento di una soluzione, quando questa esista. Si è utilizzata inoltre e ancora si utilizzerà la definizione di algoritmo di ricerca *globale*: la globalità o località della strategia si riferisce all'ambito di conoscenza dello spazio degli stati: più tecnicamente un algoritmo si può definire globale quando può avere memoria infinita dei passi precedentemente fatti. Un algoritmo che miri ad essere sistematico non può prescindere da un approccio globale alla soluzione. Un'altra proprietà importante è quella dell'*ammissibilità* (o *ottimalità*). È detto ammissibile o ottimale un algoritmo che tra tutte le possibili soluzioni individui sempre quella ottima, ovverosia quella con costo totale del percorso (*path-cost*) minimo.

1.2.1 Difficoltà di un problema di ricerca

Il concetto di difficoltà di un problema è un punto di cui solitamente si ha una sensazione ma che non è semplice da definire univocamente. Non è certamente possibile dare una definizione generale di difficoltà di un problema di ricerca: si potrebbe pensare che una ricerca sia difficile quando debba essere eseguita in uno spazio molto grande; un indicatore possibile può essere la dimensione dello spazio degli stati. Questo singolo indice in realtà non è neanche lontanamente lo specchio della difficoltà effettiva che si incontra nel ritrovare una soluzione: c'è tutta una serie di altri fattori che determina la facilità o meno di soluzione automatica di un problema.

Si opererà per una definizione operativa, relativamente al modo in cui gli algoritmi di

ricerca risolvono i problemi. Da questo punto di vista può essere ricavata una definizione a posteriori, in termini di risorse richieste dall'algoritmo per risolvere il problema. La quantità di tempo richiesta è detta *complessità temporale*, la quantità di memoria è invece denominata come *complessità spaziale*. Bisogna essere scrupolosi nel ricordarsi che queste ultime sono proprietà dell'algoritmo di ricerca, e non del problema in quanto tale.

Una caratteristica che accomuna molti algoritmi di ricerca è il fatto che cercano la soluzione all'interno di un albero, detto albero di ricerca. La complessità spaziale e temporale può essere espressa per mezzo della dimensione di quest'albero (più è grande l'albero, maggiore sarà il tempo che si impiega a percorrerlo, e la memoria necessaria a memorizzarlo).

La dimensione di un albero dipende da due importanti proprietà del problema: il fattore di ramificazione (*branching factor*) e la distanza (numero di passi) della soluzione ottima.

1.2.1.1 Branching Factor

Il fattore di ramificazione di un problema di ricerca è definito come il numero di figli generati in media dall'espansione di un nodo.

È spesso indicato semplicemente con b nelle formule.

Non tutti i problemi hanno un branching factor costante con la profondità, e il suo valore può essere spesso determinato analiticamente. Determina la larghezza dell'albero di ricerca, e nel computo del numero totale dei nodi espansi sta generalmente alla base di un esponenziale; per questo motivo la stima del suo valore deve essere estremamente accurato.

1.2.1.2 Distanza dalla soluzione ottima

La distanza della soluzione (ottima) è il numero di operatori che devono essere applicati a partire dal nodo iniziale nel migliore dei casi perché sia generato un nodo soluzione. Determina la profondità dell'albero di ricerca. La distanza è posta generalmente all'esponente in una relazione riguardante il numero di nodi espansi o generati da un algoritmo, e per un gran numero di problemi non è nota prima di eseguire la ricerca.

Spesso è indicata con d nelle formule.

Nel capitolo 3 sono presentati due diversi metodi inediti per stimare la profondità media della soluzione.

1.2.2 Ricerca non informata

Le strategie di ricerca non informata eseguono la scelta sul successivo nodo da espandere senza avere nessuna informazione esterna che guidi questa scelta. Queste tecniche vengono anche dette di forza bruta (*brute-force*), perché analizzano tutte i possibili percorsi fino a trovarne uno che arriva alla soluzione, senza avere nessuna strategia "evoluita".

La famiglia più comune di algoritmi brute-force è quella degli algoritmi ad albero; l'implementazione di qualsiasi algoritmo ad albero si basa su una lista di nodi da espandere (frontiera). La particolare strategia definisce in che ordine debbano essere inseriti i nodi nella lista.

1.2.2.1 Ricerca in ampiezza (breadth-first)

La lista è utilizzata come una coda FIFO: è sempre espanso il primo nodo inserito nella lista e figli generati sono inseriti in coda secondo un ordine casuale.

Nel corso della ricerca si espande un albero in ampiezza, perché tutti i nodi alla medesima profondità sono espansi prima di espandere un nodo di profondità successivo.

La ricerca in ampiezza è completa, e ammissibile quando il costo delle transizioni sia costante. È sistematica quando venga implementato un sistema per non reinserire lo stesso nodo nella coda più di una volta. Per evitare i duplicati si implementano due liste: oltre alla lista dei nodi della frontiera (*open-list*) si mantiene una lista (o meglio, un set, ossia una collezione non necessariamente ordinata senza duplicati) dei nodi già espansi (*closed-list*). Prima di inserire un nodo nella *open-list* si controlla che questo non sia presente nella *closed-list*.

Questa tecnica, benché ottimizzabile per mezzo di tabelle hash, introduce un consistente overhead all'algoritmo, e la sua applicazione potrebbe degradare le prestazioni nel caso in cui non siano generati un gran numero di nodi duplicati, oltre a richiedere un maggior quantitativo di memoria (si veda a proposito [Korf2005]). Una tecnica alternativa spesso utilizzata è non generare il genitore di un nodo in presenza di operatori reversibili; questa tecnica, sebbene non elimini completamente i duplicati, è implementabile molto più efficientemente e consente di ridurre il branching factor di un'unità.

La complessità spaziale e temporale vanno entrambe come $O(b^d)$; il limite di questo tipo di ricerca è costituito dalla richiesta di memoria, che eccede i limiti di una qualunque macchina anche per problemi risolvibili teoricamente in tempi più che ragionevoli.

1.2.2.2 Ricerca in profondità (depth-first)

La lista è utilizzata come uno stack: l'ultimo nodo inserito sarà il primo ad essere espanso. L'albero di ricerca è dunque esplorato in profondità fino a quando non venga trovata una soluzione.

È un algoritmo certamente non ammissibile (se la soluzione si trovasse in un ramo a profondità l che non è stato scelto per primo l'algoritmo troverebbe la soluzione su un altro ramo più in profondità). Non è neanche completo quando il problema permetta di espandere nodi fino a profondità infinita (l'algoritmo potrebbe continuare all'infinito cercando una soluzione, sebbene essa esista).

Il grande vantaggio di quest'algoritmo rispetto alla ricerca in ampiezza è che la quantità di memoria richiesta è lineare con la profondità massima dell'albero m ($O(bm)$). La complessità spaziale è invece sempre esponenziale ($O(b^m)$).

1.2.2.3 Ricerca a profondità limitata

È eseguita una ricerca in profondità fino a un limite prefissato l . Questo permette di rendere l'algoritmo completo, purché valga la condizione $d(N_0) \leq l$. In caso contrario la ricerca fallirà, non restituendo alcuna soluzione.

1.2.2.4 Ricerca ad approfondimento iterativo (iterative-deepening, ID)

Quest'algoritmo permette di mantenere il vantaggio della complessità spaziale della ricerca in profondità, mantenendosi allo stesso tempo ammissibile e completo.

Vengono eseguite iterativamente delle ricerche a profondità limitata, con limite crescente partendo da zero. Potrebbe sembrare a prima vista una tecnica che spreca una gran quantità di risorse inutilmente perché vengono eseguite d ricerche invece di una sola; in realtà l'ultima ricerca incide in complessità quanto la somma di tutte le precedenti. La complessità spaziale è dunque $O(bd)$, mentre la complessità temporale è pari a $O(2 \cdot b^d) = O(b^d)$.

1.2.2.5 Ricerca bidirezionale

Alla base della ricerca bidirezionale c'è il principio di eseguire due ricerche (ad esempio in ampiezza), partendo in una dal nodo iniziale e in una dal nodo obiettivo. La ricerca termina quando viene espanso lo stesso nodo nei due alberi di ricerca.

Questo stratagemma permette in teoria di avere due alberi di dimensione notevolmente inferiore rispetto a quello di partenza (complessità spaziale e temporale $O(b^{d/2})$ usando la strategia in ampiezza), ma risente tuttora di diverse difficoltà implementative che non permettono di adottare questa strategia in una gran parte di problemi di ricerca.

Innanzitutto non sempre essa è implementabile: è infatti necessario che tutti gli operatori siano reversibili. Un altro problema è eseguire efficientemente il controllo che venga espanso lo stesso nodo.

1.2.2.6 Comparativa tecniche di ricerca non informata

La tabella 1.2 mostra una breve sintesi sulla complessità e proprietà fondamentali delle principali tecniche di ricerca non informata.

	<i>Breadth-First</i>	<i>Depth-First</i>	<i>Limited-DF</i>	<i>ID</i>	<i>Bidirectional</i>
time	$O(b^d)$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
space	$O(b^d)$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
complete	yes	no	If $l \geq d$	yes	yes
optimal	yes	no	no	yes	yes
systematic	yes	no	no	no	yes

Tabella 1.2. Comparazione tecniche di ricerca non informata

d : profondità della soluzione ottima

b : branching factor

m : profondità massima dell'albero di ricerca

l : limite della ricerca limitata

Per quanto riguarda l'ottimalità è stato considerato il caso di edge-cost costante. In caso contrario tutti gli algoritmi sono da considerarsi in generale non ottimali, perché un percorso a profondità più elevata potrebbe fornire una soluzione a costo inferiore.

1.2.3 Ricerca informata

Un algoritmo di ricerca si dice informato quando fa uso di una funzione di valutazione informata per determinare l'ordinamento dei nodi da espandere. La funzione di valutazione $f(n)$ restituisce un numero per convenzione positivo che risulta più piccolo quanto più il nodo n è considerato promettente ai fini della ricerca.

La funzione di valutazione dipende a sua volta da due funzioni fondamentali:

- funzione euristica $h(n)$: è una valutazione esterna all'algoritmo della vicinanza del nodo n all'obiettivo.
- path-cost $g(n)$: la somma totale degli edge-cost dal nodo di partenza fino al nodo n

Dunque in generale

$$f(n) = f(g(n), h(n)) \quad (1.2)$$

In questa definizione per la prima volta si vede un utilizzo pratico di una funzione euristica. Non è necessario conoscere i dettagli dell'implementazione delle funzioni euristiche per definire teoricamente un qualsiasi algoritmo di ricerca informata.

1.2.3.1 Algoritmi Best-First

Un algoritmo best-first è un algoritmo di ricerca ad albero che espande per primo il nodo della frontiera più promettente secondo la funzione di valutazione. A livello implementativo è molto simile a un algoritmo di ricerca ad albero non informato come la ricerca in ampiezza. Si utilizza sempre una lista per rappresentare la frontiera (o due liste per evitare duplicati¹), e i nodi sono inseriti passando per una funzione di inserimento che a sua volta va a richiamare la funzione di valutazione scelta. Gli algoritmi best-first più comuni sono greedy-search², ricerca a costo uniforme, A* e weighted A*.

1.2.3.1.1 Greedy search

La ricerca greedy (golosa, o avida) sceglie il successivo nodo da espandere in base al solo valore euristico:

$$f_{greedy}(n) = h(n) \quad (1.3)$$

La ricerca ha questa denominazione per il fatto che cerca di minimizzare il valore euristico senza tenere in conto il costo del percorso: si preferisce dunque un nodo con valore euristico migliore anche quando si trovi a profondità elevate. Questo fatto non permette all'algoritmo di essere ottimale né tantomeno completo: potrebbe incastrarsi in loop infiniti nel caso in cui lo spazio degli stati sia rappresentato da un grafo fortemente connesso.

La non ammissibilità della ricerca greedy potrebbe non essere sempre uno svantaggio: il fatto di non trovare la soluzione ottima potrebbe spesso tradursi in un incremento prestazionale rispetto a strategie più "sagge".

1.2.3.1.2 Ricerca a costo uniforme

La ricerca a costo uniforme è in realtà una degenerazione della ricerca best-first nella

1 L'implementazione è identica a quella descritta per la ricerca in ampiezza

2 Spesso la ricerca greedy è indicata col termine best-first, che qui si usa in termini più generali.

ricerca non informata. La funzione di valutazione è pari al path-cost:

$$f_{ucs}(n) = g(n) \quad (1.4)$$

Nel caso di costo costante degenera in una ricerca in ampiezza. È ammissibile e completa ma, non usando funzione euristica, non ha nessun altro dei vantaggi della ricerca informata.

1.2.3.1.3 A*

L'algoritmo A* può dirsi l'algoritmo best-first per eccellenza: sfrutta la conoscenza euristica al massimo livello mantenendosi completo, e ammissibile quando lo sia anche l'euristica¹.

La funzione di valutazione è somma di euristica e path-cost:

$$f_{A^*}(n) = g(n) + h(n) \quad (1.5)$$

In prima analisi A* sembrerebbe essere la soluzione ottimale per qualsiasi tipo di ricerca; usando una funzione euristica che stimi perfettamente il reale path-cost di ogni nodo visita esclusivamente i nodi nel percorso della soluzione. In realtà con qualsiasi funzione euristica reale la complessità temporale e spaziale si mantiene esponenziale, e i limiti di memoria spingono spesso a preferire soluzioni a memoria lineare come IDA*, o a compromessi ottimalità-prestazioni (come weighted-A*).

1.2.3.1.4 Weighted-A*

L'algoritmo weighted-A* [Pohl1970] mira a ottenere alcuni dei vantaggi della non ammissibilità mantenendo comunque sotto controllo l'ottimalità della soluzione. L'idea di base è che l'algoritmo A* perde una grande quantità di risorse per discriminare tra alternative che tutto sommato non variano consistentemente l'una dall'altra. In questo tipo di situazioni può essere più conveniente effettuare una scelta che avvantaggi una strada promettente piuttosto che un'altra.

La funzione di valutazione è una somma pesata della funzione euristica e del path-cost:

$$f_{wA^*}(n) = wg(n) + (1-w)h(n) \quad (1.6)$$

con w che può variare in maniera continua tra 0 e 1. Ovviamente per $w=1$ si ottiene la ricerca a costo uniforme, per $w=0$ la ricerca greedy, per $w=1/2$ A*. Valori intermedi rappresentano un ibrido tra questi algoritmi.

Incrementi prestazionali rispetto ad A* si potrebbero ottenersi per $1/2 \leq w \leq 1$. Solitamente le migliori prestazioni si ottengono comunque per w prossimo a $1/2$ (per maggiori informazioni vedi [Pohl1970], [Gaschnig1979]).

1.2.3.2 Algoritmi a memoria lineare

Come visto, negli algoritmi best-first la quantità di memoria richiesta è spesso il vero limite alla ricerca. Si vedono in questa sezione due diverse strategie che permettono di mantenere l'ottimalità di A* mantenendo un'occupazione di memoria che varia linearmente con la profondità della soluzione.

¹ La proprietà di ammissibilità di un'euristica sarà discussa in dettaglio nel paragrafo 1.3.1.1.

1.2.3.2.1 Iterative-Deepening-A* (IDA*)

Introdotta da Richard Korf nel 1985 [Korf1985], è stato il primo algoritmo in grado di risolvere tutte le istanze del 15-puzzle. Il funzionamento è molto semplice ma incredibilmente funzionale: viene eseguita una ricerca ad approfondimento iterativo dove il raggiungimento del limite è riferito alla funzione di valutazione del nodo, che come per A* è pari a $g(n)+h(n)$. È implementato come un normale algoritmo depth-limited ad approfondimento iterativo, nel nodo viene dunque inserito nello stack dei nodi da espandere dalla funzione di inserimento solamente quando il valore della funzione di valutazione sia inferiore o uguale al limite corrente. Se la ricerca termina senza trovare la soluzione si ricomincia la ricerca dal nodo di partenza con un limite aggiornato. È ragionevole porre come inizializzazione limite il maggiore tra i costi dei nodi generati ma non espansi¹.

L'algoritmo IDA* è ottimale e completo come A*, al prezzo di un incremento del numero di nodi espansi dovuto alla ricerca iterativa. Grazie alla sua semplice implementazione è possibile ottenere comunque prestazioni velocistiche superiori ad A*: infatti l'inserimento di nodi in una lista ordinata più snella è un'operazione più efficiente (inserire un nodo in una lista ordinata di N elementi richiede $\log(N)$ confronti). Korf riporta un ordine di velocità di espansione di nodi di tre volte superiore di IDA* rispetto ad A* per gli N-puzzle.

IDA* ha comunque dei limiti teorici, che potrebbero diventare significativi per alcuni particolari problemi. Innanzitutto il numero di nodi espansi dall'algoritmo IDA* sarebbe molto superiore ad A* nel caso in cui il costo dei nodi vari notevolmente; nel caso peggiore (irrealistico) in cui ogni nodo abbia un valore euristico diverso infatti la complessità sarebbe $O(N^2)$ per uno stesso problema in cui A* espanderebbe $O(N)$ nodi. Inoltre si ha un decremento di prestazioni in spazi di ricerca nel quale è presente un gran numero di duplicati: il check dei duplicati non può infatti essere eseguito nel caso di IDA* dal momento che i nodi già espansi non sono mantenuti in memoria.

È possibile generalizzare l'idea della ricerca in profondità per costruire altri algoritmi simili a IDA*, più adatti a risolvere diversi tipi di problemi. Ad esempio può essere cambiata la funzione di valutazione. In problemi in cui la profondità della soluzione è nota (come il TSP) è più conveniente usare una tecnica simile denominata *depth-first branch-and-bound*: è una ricerca a profondità limitata nel quale il limite è costituito dal costo della migliore soluzione trovata. Possono essere inoltre tagliati i rami in cui il costo di $g(n)+h(n)$ ecceda il limite (tecnica di *branch-and-bound*).

L'algoritmo IDA* è quello utilizzato per eseguire tutti i processi di ricerca di questo lavoro.

1.2.3.2.2 Recursive Best-First Search (RBFS)

È una tecnica a memoria lineare introdotta anch'essa da Richard Korf nel 1993 [Korf1993]. Espande nodi seguendo un ordine best-first (fatto che non può accadere per una tecnica ID), mantenendo un'occupazione di memoria lineare con la profondità della soluzione.

È un algoritmo che potrebbe risultare inizialmente meno immediato alla comprensione rispetto ad altri, ma dai principi comunque semplici; come per IDA* si basa su un concetto di soglia, in questo caso locale invece che globale, e su un aggiornamento dinamico della

¹ Questo può essere ragionevolmente noto a priori conoscendo la variabilità dell'euristica: nel N-Puzzle e con euristiche che mantengono la parità come manhattan in ogni iterazione il limite può essere incrementato di due unità.

funzione di valutazione dei nodi. La ricorsione è alla base di questo algoritmo: in ogni singolo passo ricorsivo sono espansi seguendo una strategia best-first tutti i nodi che siano inferiori alla soglia corrente. La soglia di un nodo è rappresentata dal più piccolo valore di funzione di valutazione dei nodi nella lista dei nodi da espandere. Nel momento in cui un nodo espanso genera dei figli con funzione di valutazione superiore alla soglia questi vengono scartati e il padre è aggiunto alla lista con un valore di f aggiornato col più piccolo tra i valori di f dei figli. Un accorgimento usato per evitare inefficienze è aggiornare il valore di f dei figli a quello del padre quando esso ne sia inferiore (questo permettere di rendere la f strettamente crescente anche quando non lo sia di partenza).

L'ordine di espansione dei nodi è di tipo best-first anche con una funzione di valutazione non monotona.

1.2.3.3 Algoritmi Bidirezionali

La ricerca euristica bidirezionale è stata oggetto sin dall'inizio di grandi aspettative, ma di altrettanto grandi fallimenti all'atto pratico. Da quando fu introdotto da Pohl ormai nel 1971 l'algoritmo *BHPA* [Pohl1971] (che, semplificando, può essere visto come una versione bidirezionale di A^*), il problema principale è stato il fatto che nulla può garantire che i due fronti di ricerca guidati da una funzione euristica si incontrino nel percorso più breve. Altri tradizionali algoritmi degni di citazione sono *BS** [Kwa1989], una versione migliorata di *BHPA*, e *BHFFA2* [DeChampeaux1983]. Quest'ultimo algoritmo mira a evitare il problema di *BHPA* cercando di forzare i due alberi a incontrarsi. Nessuno di questi approcci si è rivelato in grado di competere con A^* come prestazioni. La nuova frontiera della ricerca bidirezionale è costituita dagli algoritmi di perimetro. Un'esaustiva trattazione dell'argomento allo stato dell'arte si può trovare in [KaindlKainz1997].

1.2.3.4 Algoritmi di perimetro

Gli algoritmi di perimetro sono una forma di ricerca bidirezionale ibrida: invece di espandere due fronti contemporaneamente viene inizialmente espanso un albero a partire dal nodo obiettivo fino a profondità data; i nodi alla frontiera di questa espansione costituiscono il *perimetro*. È eseguita dunque una ricerca unidirezionale che ha come obiettivo i nodi del perimetro; la strategia usata per questa ricerca determina i diversi algoritmi. All'atto pratico è implementata una normale ricerca unidirezionale usa un'euristica che dipende dal path-cost dei nodi del perimetro: indicando con $H(n, m)$ il valore stimato (euristicamente) del path-cost fra due nodi n e m , e con $H^*(n, m)$ il valore di path-cost esatto, la ricerca unidirezionale usa l'euristica

$$h_{P_d}(n) = \min_{m \in P_d} [H(n, m) + H^*(m, t)] \quad (1.7)$$

Questo tipo di tecnica può portare a vantaggi consistenti non solo in termini di nodi espansi ma anche in termini di tempo. Il vantaggio potenziale è dato esclusivamente dall'uso di un'euristica più informata, ed è tanto maggiore quanto più l'euristica di partenza è efficiente da calcolare.

Le prestazioni variano considerevolmente a seconda della profondità del perimetro; determinare la profondità ideale del perimetro è un problema ancora aperto, e che potrebbe avvantaggiarsi degli studi sulla valutazione delle euristiche.

1.2.3.4.1 BIDA*

L'algoritmo BIDA*, proposto da Manzini nel 1995 [Manzini1995], è un algoritmo che sfrutta la tecnica di ricerca a perimetro combinandola con l'algoritmo IDA*, introducendo una inedita metodologia *lazy* per diminuire il numero di valutazioni euristiche.

Infatti la presenza della soglia in IDA* permette di adottare in ogni passo un sottoinsieme del perimetro:

$$P'_d(n, T) = \{m \in P_d : g(n) + H(n, m) + H^*(m, t) \leq T\} \quad (1.8)$$

Durante la ricerca mentre si va in profondità l'insieme P' si riduce fino a svuotarsi: il controllo della (1.8) fallirà dunque una sola volta durante la ricerca in profondità per ogni nodo del perimetro.

Un'altra tecnica che può permettere di ridurre il numero di valutazioni euristiche è considerare l'edge-cost, quando questo sia noto; per un nodo n' successore di n sarà $H(n', m) \geq H(n, m) - c(n, n')$. Se dunque il valore minimo trovato di $f(n)$ è già superiore a $g(n) + H(n, m) - c(n, n')$ si può evitare di calcolare $H(n', m)$ per il nodo di perimetro m . Quest'ultima tecnica non è usata nell'implementazione per il 15-puzzle in [Manzini1995], essendo in questo problema più conveniente eseguire incrementalmente il calcolo dell'euristica. Nei test sul 15-puzzle Manzini riporta un incremento di prestazioni che arriva a 83 volte in meno come numero di nodi generati e 8 volte di tempo in meno per un perimetro pari a 14. Il risultato in termini di tempo è stato in parte ridimensionato in [KaindlKainz1997], nel quale è riportato un tempo del 27.8% di tempo di BIDA*₁₄ rispetto a IDA* per il 15-puzzle (3,6 volte in meno invece di 8).

1.2.3.5 Algoritmi real-time

Gli algoritmi visti fino ad ora non permettono di conoscere a priori il tempo di esecuzione, e il percorso della soluzione è noto soltanto al termine della ricerca. In numerose applicazioni reali il tempo a disposizione per eseguire un passo è dato e costante.

La condizione delle mosse da realizzare in tempo costante è tipica dei problemi a due giocatori; l'idea è applicare gli stratagemmi usati nei problemi con più giocatori ai problemi di ricerca a singolo agente. È un'idea del solito Korf, che introduce ufficialmente l'argomento in [Korf1990], nel quale vengono proposti anche due inediti algoritmi, RTA* e LRTA*.

Entrambi gli algoritmi si basano sul principio di *minimin lookahead search (MLS)*, adattamento per i problemi unidirezionali della strategia dei problemi a due giocatori chiamata *minimax lookahead* [Shannon1950], a cui si affianca all'atto pratico un'altra tecnica denominata *alpha-beta pruning* [HartEdwards1963].

Nel processo di MLS è eseguita una ricerca a profondità data (dipendente dal tempo a disposizione), e al termine viene eseguita una mossa in direzione del figlio più promettente, ossia quello con valore di f minimo. Se f è non decrescente può essere applicata la tecnica di *branch-and-bound*: se un nodo ha un costo superiore a quella minore trovata il ramo può essere tagliato perché sicuramente i figli avranno una funzione di valutazione superiore o uguale. La tecnica branch-and-bound si dimostra più efficace in problemi con fattore di ramificazione più elevato, permettendo di cercare più in profondità.

La ricerca minimin lookahead può essere usata per ottenere una funzione euristica più informata (il valore minimo più il path cost fino al nodo di minimo può essere usato come

funzione euristica), con informazione e complessità sempre maggiore aumentando la profondità. Attraverso la MLS è dunque possibile avere una famiglia di euristiche di complessità e informatività crescente; al contrario di quello che accade per le ricerche di perimetro, non è possibile allo stato attuale sfruttare quest'euristica per incrementare le prestazioni.

1.2.3.5.1 Real-time A* (RTA*)

La reiterazione della ricerca minimin lookahead non può essere applicata direttamente nella soluzione di qualsiasi problema: essa infatti non è completa perché non può evitare di ritornare nei nodi già visitati senza finire in loop infiniti. Non è sufficiente tenere memoria dei nodi già visitati perché potrebbero essere visitati comunque i successori degli stessi.

È necessaria, in un algoritmo real-time, oltre a una strategia di pianificazione (che potrebbe essere la MLS ma non necessariamente), una strategia di esecuzione. La strategia di esecuzione non è altro che un processo algoritmico di livello superiore: la strategia di pianificazione è incapsulata completamente o quasi nella funzione di valutazione superiore.

In RTA* la strategia è la stessa di A*: viene espanso per primo il nodo con $f(n) = g(n) + h(n)$ inferiore, con la importante differenza che stavolta $g(n)$ non è il costo dal nodo iniziale ma dal nodo corrente, e che il valore di $h(n)$ dei nodi già visitati è aggiornato dinamicamente.

Questo meccanismo è implementabile efficientemente per mezzo di una tabella hash dei nodi visitati, a ognuno dei quali è associato il proprio valore di h . Nel momento in cui viene espanso un nodo già visitato durante la fase di pianificazione, è usato direttamente il valore euristico tabulato. Nel momento in cui un nodo viene espanso la mossa successiva sarà nella direzione del figlio con f inferiore; il nodo di partenza sarà memorizzato nella tabella nel caso in cui non sia già presente, e gli sarà associato un valore euristico pari al valore di f del secondo tra i figli più promettente. Il processo si reitera fino a trovare una soluzione.

Questa tecnica permette all'algoritmo di ritornare sui propri passi quando questo sia opportuno, ma il loop infinito è evitato dal fatto che durante ogni visita dello stesso nodo l'algoritmo apprende per esso un valore euristico sempre più preciso. Ogni mossa realizzata dall'algoritmo è nella direzione di minimo valore di funzione di valutazione tra i nodi della frontiera cumulativa nel dato momento (l'algoritmo prende decisioni localmente ottime).

Si dimostra che l'algoritmo RTA* è completo quando lo spazio di ricerca sia limitato.

La qualità della soluzione trovata in real time dipende dalla qualità della funzione euristica usata; supponendo di usare una strategia lookahead la qualità sarà migliore per profondità maggiori di lookahead, avvicinandosi asintoticamente alla soluzione ottima.

1.2.3.5.2 Learning real-time A* (LRTA*)

L'algoritmo LRTA* cerca di sfruttare il fatto che, per ricerche multiple nello stesso spazio degli stati, può essere di grande utilità usare il valore euristico memorizzato da precedenti ricerche. Per sfruttare questa possibilità è necessario modificare leggermente l'algoritmo RTA* affinché memorizzi il valore del migliore dei figli per ogni nodo invece che il secondo migliore. Si dimostra che il valore memorizzato converge al valore esatto dopo la risoluzione di un gran numero di problemi.

1.2.3.5.3 Oltre LRTA*

LRTA* ha stimolato un fertile bacino di ricerca riguardo agli algoritmi di ricerca in tempo reale con apprendimento. La direzione è quella di migliorare l'efficienza e la convergenza di quest'algoritmo. Tra le proposte più interessanti si può citare FALCONS (FAst Learning and CONverging Search) [Ishida2003], che velocizza il tempo di convergenza usando dei sistemi per scegliere il miglior nodo a parità di euristica (*tie-breaking rules*, fondamentalmente a parità di f si opta per il percorso più breve) e HLRTA* (Hibrid Real Time A*), [Thorpe1994], che cerca di ottenere i vantaggi dell'apprendimento di LRTA* combinandola con l'efficienza del semplice RTA*.

Interessanti sviluppi recenti sono la combinazione di FALCONS e HLRTA* [Furcy2000], e l'utilizzo di tecniche di propagazione dei valori euristici appresi prendendo come base uno degli algoritmi menzionati (tecnica *bounded propagation*). Chi sia interessato ad approfondire la questione, veda ad esempio l'algoritmo HLRTA*(k) [HernandezMeseguer2005].

1.2.3.6 Ricerca locale euristica

Per ricerca locale si intende una strategia a memoria limitata, che prende le proprie decisioni solamente in base alle caratteristiche locali dello spazio di ricerca. È presentato in questa sezione l'algoritmo hill-climbing, ma altre tecniche più sofisticate possono essere adottate, come la *simulated-annealing*, che si incrociano in parte o del tutto col mondo degli algoritmi di stochastic local search (trattati nella sezione successiva). È stata usata una sezione differente per rimarcare una differente definizione del problema, in questo caso basato su operatori mentre per la ricerca locale stocastica si preferisce usare una definizione diversa del problema.

1.2.3.6.1 Hill-Climbing

L'algoritmo hill-climbing è la più semplice forma di strategia euristica; viene memorizzato un solo nodo per volta e il successivo nodo a essere espanso è un figlio del nodo attuale scelto casualmente tra quelli con valore euristico inferiore.

La strategia hill-climbing non garantisce di trovare la soluzione: nel momento in cui si imbatte in un minimo locale della funzione euristica non riesce a uscirne. La soluzione trovata è in generale non ottima.

1.2.3.7 Complessità degli algoritmi di ricerca informata

Determinare la complessità di un algoritmo di ricerca informata è un'operazione affatto semplice, perché fortemente dipendente dall'euristica utilizzata: la funzione euristica riduce il numero di nodi espansi durante la ricerca, aumentando d'altra parte il tempo di espansione di un nodo. L'effetto dell'euristica dipende dalla strategia usata e dalle caratteristiche della stessa: ad esempio A* può avere complessità che varia da lineare a esponenziale a seconda dell'euristica; nei problemi con euristiche reali la complessità si mantiene esponenziale.

Resta da determinare l'effetto della funzione euristica su questa relazione; è storicamente accettato che la funzione euristica contribuisca a ridurre il branching factor effettivo di un problema. Korf mette ora in dubbio questa relazione in base anche alle osservazioni sui risultati di [KorfReidEdelkamp2001], nel quale è proposto un metodo per il calcolo a priori dei nodi espansi (descritto nel dettaglio nel paragrafo 2.3.4). L'euristica non contribuirebbe ad

abbassare il branching factor, abbasserebbe invece l'effettiva profondità della soluzione.

Pearl ha proposto una classificazione grafica degli algoritmi di ricerca, in base alla capacità di selezione delle alternative più promettenti e al grado di recupero da possibili scelte sbagliate (figura 1.4).

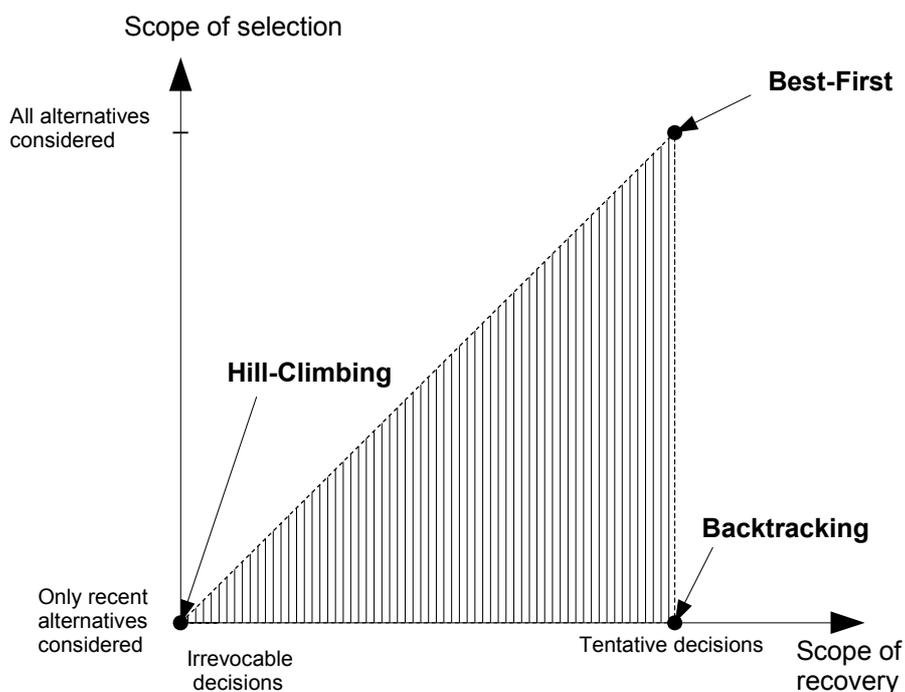


Figura 1.4.: Classificazione di Hill-Climbing, Backtracking e best-first in uno spazio bidimensionale

L'algoritmo IDA*, non ancora sviluppato quando è stata pensata questa classificazione, si porrebbe immediatamente al di sopra degli algoritmi best-first in senso verticale, nel senso che tutte le alternative sono considerate, spesso più di una volta. La funzione euristica avrebbe l'effetto di ridurre l'area del triangolo, poiché permette una migliore selezione delle alternative da considerare (dunque in verticale); risulta oltretutto inferiore il numero di tentativi da considerare (dimensione orizzontale). Una buona euristica ha dunque in questo senso l'effetto di avvicinare le prestazioni delle diverse strategie algoritmiche.

1.2.4 Ricerca locale stocastica (stochastic local search, SLS)

Con il termine di SLS si indica una famiglia di algoritmi che, oltre ad avere le caratteristiche indicate nel nome stesso (*locale* perché solo un numero finito di stati è memorizzato, *stocastica* perché il successivo stato è determinato secondo regole non deterministiche), si distingue dalla classica ricerca, trattata fino ad ora, per una diversa formulazione di base del problema.

Un problema SLS ben formulato è costituito da:

- Spazio di ricerca $S(\pi)$, costituito da *candidate soluzioni*

- Una collezione di soluzioni $S'(\pi) \subseteq S(\pi)$
- Una collezione finita di stati memorizzati $M(\pi)$
- Una relazione di vicinanza su $S(\pi)$, $N(\pi) \subseteq S(\pi) \times S(\pi)$
- Una funzione di inizializzazione $init(\pi): \emptyset \rightarrow D(S(\pi) \times M(\pi))$
- Una funzione di passo $step(\pi): (S(\pi) \times M(\pi)) \rightarrow D(N(\pi))$
- Un predicato di terminazione $terminate(\pi): S(\pi) \times M(\pi) \rightarrow D(T, F)$

Il termine $D(X)$ indica una distribuzione di probabilità nello spazio X .

Un algoritmo di ricerca locale stocastica sceglie in ogni passo un nodo corrente tra i nodi vicini (in N), scegliendo un nodo tra i vicini secondo la probabilità determinata dalla funzione di passo. L'algoritmo termina quando il test sul predicato di terminazione restituisce valore booleano vero, e il nodo corrente viene restituito.

Esempi non informati di SLS sono la *uninformed random-pick search*, nella quale il nodo successivo viene scelto completamente a caso in tutto lo spazio degli stati ($N(\pi) = S(\pi) \times S(\pi)$), oppure la *uninformed random-walk search*, in cui il nodo successivo è scelto a caso tra i vicini.

Gli algoritmi di SLS possono essere resi informati introducendo una funzione di valutazione $E(\pi)$; la funzione di passo sarà nel caso $step(\pi): (S(\pi) \times M(\pi) \times E(\pi)) \rightarrow D(N(\pi))$.

La più semplice implementazione di SLS informata è la tecnica di *iterative-improvement* (equivalente in sostanza alla già vista *hill-climbing*), nella quale il nodo successivo è scelto con pari probabilità tra i nodi vicini con funzione di valutazione inferiore.

Algoritmi più sofisticati si basano principalmente sulla randomizzazione e aggiunta di memoria al processo di *iterative-improvement* (ad es. *simulated-annealing*, *taboo-search*). La randomizzazione permette a un algoritmo di scappare dagli eventuali minimi della funzione di valutazione. Sono anche usate tecniche basate sulla popolazione simili agli algoritmi evolutivi. La trattazione specifica di questi algoritmi esula dagli scopi di questo documento. Come esaustivo riferimento sull'argomento si consiglia [HoosStutzle2005].

La ricerca locale stocastica rinuncia completamente alle pretese di ottimalità e completezza per cercare di ottenere soluzioni in maniera efficiente anche per problemi di grande difficoltà. Una buona metrica di valutazione per un algoritmo SLS è il fatto di trovare soluzioni di qualità accettabile in una buona percentuale di problemi.

1.3 Le funzioni euristiche

Fino ad ora si è parlato delle funzioni euristiche come un qualcosa di astratto, senza entrare nel merito di come effettivamente possano essere calcolate o di quali siano le proprietà che le contraddistinguono.

Come già si è detto le euristiche sono metodi, criteri, o principi per decidere quale fra diverse possibilità di azione sia più promettente per raggiungere un qualche obiettivo. Operativamente, un'euristica restituisce un valore numerico per ogni nodo che indica quanto questo è ritenuto promettente per arrivare alla soluzione. Un'euristica può essere una cosiddetta “regola del pollice”, una valutazione sommaria basata sull'esperienza che si ha del

problema, o può basarsi su complessi algoritmi o qualsiasi metodo la fantasia suggerisca.

La bontà della funzione euristica incide in misura determinante sulle prestazioni degli algoritmi di ricerca informata, e determinare un'euristica che consenta di risolvere soddisfacentemente un problema è una questione di notevole interesse, e molto meno semplice di quanto potrebbe apparire. Infatti una buona euristica deve, oltre che discriminare il più possibile correttamente tra alternative più o meno buone restituendo un valore numerico, essere efficiente da calcolare perché se diminuisce il numero di nodi espansi d'altra parte aumenta il tempo di espansione dei nodi per effetto dell'euristica.

In questa sezione sono descritte le proprietà fondamentali delle funzioni euristiche e le principali tecniche di progettazione pratica delle stesse. Infine si ha la presentazione dettagliata delle funzioni euristiche utilizzate negli esperimenti dei capitoli successivi.

1.3.1 Proprietà generali delle funzioni euristiche

In una tipica applicazione pratica un'euristica è una procedura che restituisce una stima del costo della soluzione; perché possa essere usata efficacemente nella risoluzione di un problema deve soddisfare determinate proprietà. Le proprietà si riferiscono al modo in cui questo costo è approssimato, e possono influire sulle prestazioni degli algoritmi e sulla qualità della soluzione trovata. Si dà solitamente per scontato che il valore di euristica sia zero per qualunque nodo soluzione¹.

Si vedranno ora alcune proprietà per così dire “classiche”, come ammissibilità e consistenza, insieme alla ϵ -ammissibilità e ϵ -ammissibilità trattate tra gli altri in [Pearl84], a cui si aggiunge l'interessante proprietà di p -ammissibilità, [Ernandes2003].

1.3.1.1 Ammissibilità

La definizione della proprietà di ammissibilità delle funzioni euristiche è strettamente legata all'ammissibilità dell'algoritmo A^* ; una funzione euristica è ammissibile quando usata in combinazione con A^* garantisce di trovare la soluzione ottima del problema².

Una funzione euristica ammissibile $h(n)$ è una stima ottimistica della soluzione ottima $h^*(n)$.

Definizione: Una funzione euristica si dice **ammissibile** se:

$$h(n) \leq h^*(n) \forall n \tag{1.9}$$

1.3.1.2 Monotonicità

Per una funzione euristica monotona il valore in +qualsiasi nodo è inferiore alla somma del valore euristico di qualsiasi figlio più il l'edge-cost dal nodo stesso al figlio.

Definizione: una funzione euristica si dice **monotona** quando

$$h(n) \leq c(n, n') + h(n') \forall n, n' | n' \in SCS(n) \tag{1.10}$$

¹ Una qualsiasi euristica può essere facilmente adattata perché soddisfi questa proprietà.

² L'algoritmo A^* troverà la soluzione ottima di un problema se il valore euristico di tutti i nodi espansi durante la ricerca soddisfano la 1.9. La funzione euristica è ammissibile se tutti i nodi dello spazio di ricerca soddisfano la 1.9.

1.3.1.3 Consistenza

Una funzione euristica è consistente quando considerata una qualsiasi coppia di nodi la differenza dei loro valori euristici è inferiore al path-cost tra gli stessi

Definizione: una funzione euristica è **consistente** se

$$h(n) - h(n') \leq k(n, n') \forall n, n' \quad (1.11)$$

Sembrirebbe una proprietà più restrittiva rispetto alla monotonicità. In realtà le proprietà di consistenza e monotonicità sono completamente equivalenti. La dimostrazione è molto semplice: è possibile ricavare per induzione la (1.11) a partire dalla (1.10) considerando i figli a distanza qualunque.

Teorema 1.1: consistenza e monotonicità sono proprietà equivalenti.

Esiste una ancora più importante relazione tra consistenza e ammissibilità:

Teorema 1.2: ogni euristica consistente è anche ammissibile

Dimostrazione: sostituendo a n' nella (1.11) una qualsiasi nodo soluzione e considerando che il valore euristico per ogni nodo soluzione è pari a zero si ottiene la (1.9).

La proprietà di consistenza si traduce in un vantaggio per la ricerca: infatti una funzione euristica non monotona, anche se ammissibile, porta un algoritmo come A* a visitare prima dei percorsi meno promettenti: infatti potrebbe capitare di seguire dei percorsi con dei valori euristici bassi che in realtà corrispondono a percorsi che si concludono soltanto in nodi con valori più elevati. Algoritmi con aggiornamento dinamico come RBFS e RTA* possono permettere di “accorgersi” dei casi di non monotonicità e di correggerli.

Le proprietà di consistenza e ammissibilità hanno un importante rovescio della medaglia: spesso volte risulta difficile ottenere delle euristiche consistenti e ammissibili, soprattutto nei problemi reali, e anche quando si trovino forzare il loro uso potrebbe essere controproducente. Infatti raramente interessa veramente la soluzione ottima, ma va ugualmente bene una soluzione sub-ottima con uno scarto controllabile. Sotto quest'ottica sono interessanti le proprietà di ϵ -ammissibilità e p-ammissibilità.

1.3.1.4 ϵ -ammissibilità, e-ammissibilità

Pearl propone una proprietà chiamata dell' ϵ -ammissibilità in un contesto di algoritmi non ammissibili, ma con la possibilità di tenere sotto controllo la sovrastima della soluzione. In particolare un algoritmo è detto ϵ -ammissibile quando valga

$$C(n) \leq C^*(n)(1 + \epsilon) \quad (1.12)$$

La ϵ -ammissibilità è una proprietà che, benché possa essere collegata alle euristiche¹, non le è riferita direttamente ad esse. Più immediata è invece la proprietà di e-ammissibilità².

Si parla di e-ammissibilità quando una funzione euristica può occasionalmente

1 Una funzione di valutazione dinamica come quella proposta da Pohl nel 1973 garantisce la ad esempio la ϵ -ammissibilità
 2 Spesso si indica con ϵ -ammissibilità quella che qui è indicata con e-ammissibilità. Si è usata qua la nomenclatura di [Pearl84].

sovrastimare il valore di path-cost, ma la sovrastima non può mai eccedere un parametro e . Ossia

$$h(n) \leq h^*(n) + e \quad \forall n \quad (1.13)$$

Un algoritmo A^* guidato da un'euristica e -ammissibile è anch'esso e -ammissibile, ovvero non sovrastimerà mai la soluzione più di e [Harris1974].

1.3.1.5 P-ammissibilità e “sovrastime ammissibili”

In [Ernandes2003] si presenta uno studio sull'effetto della sovrastima dell'euristica sull'ottimalità della soluzione. In particolare si dimostra il fatto che anche con un'euristica non ammissibile secondo i classici canoni si può talvolta essere certi che un algoritmo come A^* mantenga la sua ammissibilità. In particolare un'euristica h per un problema il cui l'edge-cost sia costante e pari a c , che stimi perfettamente il valore dei nodi successori delle soluzioni, porterà l'algoritmo A^* a trovare la soluzione ottima quando valga la condizione

$$h(n) \leq h^*(n) + 2c \quad \forall n \quad (1.14)$$

Questa condizione è denominata di *sovrastima ammissibile forte*. La sovrastima ammissibile ha effetti deleteri sulle prestazioni degli algoritmi. Ossia sono sì espansi tutti i nodi nella direzione della soluzione, ma in aggiunta molti altri nodi sono espansi¹.

L'altro concetto introdotto è quello della *p-ammissibilità*. Un algoritmo p -ammissibile trova soluzioni ottime con una probabilità p . Data una funzione euristica che è una sottostima del costo della soluzione con probabilità p_h^* allora la probabilità di trovare soluzioni ottime sarà:

$$p(h, d) = p_h^{*d} \quad (1.15)$$

Questa formulazione può anche essere vista nell'ottica di sovrastime ammissibili; in quel caso p_h^* sarebbe la probabilità di soddisfare la condizione (1.14).

La definizione di p -ammissibilità è necessaria nello studio di euristiche subsimboliche: infatti non si può garantire che uno stimatore come una rete neurale fornisca al 100% una sottostima della distanza dalla soluzione.

1.4 Definizione di funzioni euristiche

Si vedranno ora le diverse tecniche esistenti per la progettazione e calcolo di funzioni euristiche. Si è detto che l'euristica deve essere una stima (per difetto se si desidera un'euristica ammissibile) del costo della soluzione ottima, ma non si è detto come sia possibile ricavare questa stima. La più comune forma di determinare il valore euristico è quello di usare una procedura algoritmica ad-hoc, che avrà come base informativa la codifica usata per rappresentare lo stato del problema. L'euristica è, in altre parole una funzione dello stato codificato, e niente altro: $h: S_c \rightarrow \mathbb{R}$.

I processi cognitivi che portano alla definizione di un'euristica di questo tipo non sono universalmente definibili, e strettamente legati al particolare problema; lo scopo è dotare

¹ Questo è un risultato sperimentale presentato nell'articolo ma sul quale si potrebbe evidenziare un difetto di impostazione: un'aggiunta di errori casuali, ammissibili o meno, causa sempre un aumento di nodi espansi.

l'algoritmo di informazioni utili sul problema da risolvere, e gli strumenti per realizzare ciò possono derivare sia dall'esperienza comune che da tecniche in qualche modo metodiche.

Per ricavare delle euristiche algoritmiche ammissibili ci si può basare sulla strategia del *problema rilassato*: data una rappresentazione del problema attraverso regole e vincoli, la soluzione del problema ricavato eliminando qualcuno di questi vincoli è un'euristica certamente ammissibile per il problema di partenza. Si vedrà che è possibile anche, attraverso il concetto del rilassamento, automatizzare la fase di progettazione di un'euristica.

Un'altra possibilità è usare dei valori precalcolati totalmente o parzialmente per ogni nodo. Questa tecnica è detta dei *pattern database*; esistono diverse strategie per calcolare e/o combinare i valori da inserire del database.

Le tecniche indicate come subsimboliche invece si basano sull'addestramento di stimatori come le reti neurali.

Si vuole considerare inoltre una categoria aggiuntiva di euristiche, o più esattamente una tecnica che permette in teoria di aumentare a piacimento l'informatività di qualsiasi euristica: le euristiche ottenibili costruendo un perimetro intorno al nodo obiettivo.

1.4.1 L'euristica come soluzione del “problema rilassato”

Un metodo sicuro per scovare euristiche ammissibili e consistenti è quella di usare la soluzione di un'istanza di un problema ottenibile da quello di partenza, ma con uno o più vincoli rilassati. La tecnica è concettualmente semplice, e le euristiche ottenute oltre che essere ammissibili e consistenti, sono per definizione più facili da calcolare rispetto al problema iniziale.

Il modo migliore per comprendere che cosa significhi rilassamento di un problema è quello di presentare alcuni comuni esempi: si consideri ad esempio il problema del ritrovamento di percorso in una mappa stradale: si tratta di trovare un percorso tra due città *passando* per le strade indicate nella mappa. Togliendo il vincolo di passare per le strade si ottiene l'euristica distanza euclidea, ossia la distanza in linea d'aria tra una città e l'altra. È facile convincersi che la distanza in linea d'aria sia una stima ottimistica della distanza del percorso, essendo certamente la linea retta la via più breve per congiungere due punti.

Un altro esempio spesso usato è quello del travelling salesman problem. Una soluzione corretta del TSP è un grafo con tre proprietà:

1. Passa per ogni nodo
2. Ogni nodo ha grado due, ossia ha un arco entrante e un arco uscente
3. Il grafo è connesso

Una proprietà aggiuntiva è che il percorso debba essere il più breve.

Eliminando uno dei tre vincoli si ottiene un problema rilassato, che in due casi rappresenta una buona euristica per la soluzione del TSP.

Eliminando il vincolo che ogni nodo sia di grado due il problema diventa quello di ottenere il un grafo connesso che passi per tutti i nodi. Questo è chiamato spanning graph. Lo spanning graph a costo minimo è il cosiddetto minimum spanning tree (MST), che può essere calcolato in $O(n^2)$, con n il numero di nodi. Questo tempo è certamente adatto per un'euristica di un problema dalla complessità notoriamente esponenziale come il TSP.

In alternativa, eliminando il vincolo che il grafo sia connesso si ottiene il problema rilassato di un grafo che passa per ogni nodo di grado due. Ci possono essere dunque diversi cicli nel grafo, ma non necessariamente connessi l'uno con l'altro. Si può vedere ad esempio un grafo in cui ogni città ha un'altra città assegnata ad essa. Trovare il set di assegnazioni a costo minimo è chiamato problema dell'assegnazione, e può essere risolto in un tempo $O(n^3)$, [KorfBook].

Se si considera invece l'*N*-puzzle, l'euristica di manhattan può essere vista come il problema cui si tolga il vincolo che una tessera si possa muovere soltanto nella posizione vuota. Nell'apposita sezione si vedranno i principi alla base di una serie di euristiche per risolvere questo problema.

1.4.2 Generazione automatica di funzioni euristiche

Il processo di generazione di euristiche per mezzo del rilassamento del problema può essere automatizzato. Quello di cui si ha bisogno è un linguaggio che permetta di rappresentare formalmente un problema. Un linguaggio è detto rappresentazione *STRIPS* [FikesNilsson1971]. In questa rappresentazione lo stato corrente del problema e gli operatori ammessi sono descritti da una serie di formule atomiche nella logica dei predicati. In più, una *add-list* e una *delete-list* definiscono i predicati che devono essere tolti e aggiunti a seguito dell'applicazione di un operatore; una *precondition-list* determina le precondizioni per l'applicazione di un operatore.

Data questa rappresentazione, può essere ottenuto un problema rilassato semplicemente eliminando elementi dalla *precondition-list*. Il set di predicati può essere aumentato in modo da consentire raffinamenti più vicini al problema iniziale.

D'altra parte la rappresentazione del problema in *STRIPS* necessita un consistente intervento umano; un limite teorico è che risulta difficile per un automa riconoscere quali delle rappresentazioni siano più o meno adatte a risolvere un problema, e valutare se queste rappresentazioni siano efficienti da calcolare.

L'implementazione probabilmente più famosa ed efficace di generatore automatico di euristiche è *ABSOLVER* [Prieditis1993]: questo programma è stato tra l'altro in grado di scoprire un'euristica per l'*N*-puzzle dominante rispetto a manhattan (euristica *X-Y*), nonché euristiche di sicuro interesse per il cubo di Rubik.

1.4.3 Euristiche da pattern database

I pattern database sono la base per un insieme di tecniche che permettono di ottenere euristiche notevolmente informate. Inizialmente introdotte da Schaeffer [CulbersonSchaeffer1998] per risolvere il 15-puzzle, si basano sul principio di calcolare euristiche computazionalmente complesse e memorizzare il risultato in un database, in modo da poter ripartire il costo per cluster relativamente ampi di nodi. La maniera comune per combinare euristiche ben informate con la richiesta di poter riutilizzare lo stesso risultato per insiemi consistenti di stati è quella di definire cluster di nodi con parti di codifica uguali, e risolvere il problema disinteressandosi del valore del resto (soluzione del pattern).

La categoria più semplice è quella dei pattern database non additivi; ogni nodo può appartenere a uno o più pattern, e il valore più alto tra i diversi pattern è usato come euristica, per mantenere l'ammissibilità. Un semplice esempio è la tecnica usata da Schaeffer per

risolvere il 15-puzzle: i pattern sono costituiti dalla soluzione del problema in cui si richiama di risistemare soltanto una porzione delle tessere. In particolare sono state prese delle configurazioni con uguali tessere “di frontiera” e posizione vuota (figura 1.5). L'euristica è la soluzione del puzzle in figura, con le altre tessere presenti ma equivalenti.

			3
			7
			11
12	13	14	15

Figura 1.5.: pattern con tessere di frontiera per il 15-puzzle

Il numero di diverse combinazioni da memorizzare è dato dall'insieme di combinazioni di 8 tessere (7 più la casella vuota) in 16 caselle, ossia $16!/(16-8)! = 518.918.400$; per memorizzare il numero di mosse necessarie a risolvere le tessere di frontiera (meno di un byte ognuna) sono sufficienti meno di 500 Mbyte e tutte le posizioni possono essere memorizzate eseguendo un'unica ricerca a ritroso a partire dalla soluzione. Utilizzando solo questo pattern database Schaeffer e Culberson sono riusciti a ridurre il numero di nodi generati di 346 volte rispetto alla distanza di manhattan; la riduzione del tempo è stata invece di sei volte. Combinando questo con un altro database (prendendo il migliore dei due caso per caso) la riduzione di nodi è stata di un migliaio e il tempo di 12 volte. Questa tecnica è stata usata con successo anche nella soluzione del cubo di Rubik.

Il principale difetto dei pattern database non additivi è che non possono essere scalati per problemi più grandi; usare un singolo pattern database per il 24-puzzle contenendo la memoria a livelli accettabili richiede di usare pattern con un numero di tessere molto piccolo. Nel caso del 24-puzzle, per memorizzare un pattern formato da sei tessere sono necessari più di 2 gigabyte di memoria, per ottenere un risultato inferiore alla distanza di manhattan.

Lo stato dell'arte dal punto di vista delle prestazioni al momento è costituito dai disjoint pattern databases [KorfFelner2002]; si sommano i valori ottenuti da database cosiddetti “disgiunti”: nel caso del N-puzzle due database sono disgiunti se non contengono nessuna tessera uguale. Nella soluzione di un pattern le altre tessere sono dunque eliminate dalla configurazione, perché non possa esserci nessun conflitto con tessere non presenti nel database. Ad esempio la distanza di manhattan per il 15-puzzle può essere vista come la combinazione di 15 database disgiunti, ognuno contenente una sola tessera. Si possono pensare dunque diversi modi di raggruppare le tessere, ottenendo l'equivalente di diversi database non additivi. Un metodo spesso usato è usare due raggruppamenti spazialmente ortogonali (riflessi).

La tecnica dei database disgiunti è stata applicata con successo alla soluzione del 24-puzzle.

1.4.4 Tecniche subsimboliche

Si accenna ora a una diversa possibilità di progettazione delle euristiche: usare degli stimatori che riescano ad apprendere da degli esempi e generalizzare il risultato come le reti

neurale. Usare una rete neurale per generare una funzione euristica è un'idea sicuramente semplice, che va in ogni caso incontro ad alcuni problemi teorici e pratici; innanzitutto non si può garantire che la stima di una rete neurale fornisca una sottostima del path-cost e dunque euristiche ammissibili. Inoltre è necessario costruire un dataset di addestramento utilizzando altre euristiche.

In [Ernandes2003] è descritta un'applicazione delle euristiche neurali applicata al N-puzzle, in un contesto di ricerca p-ammissibile. Le prestazioni in termini di nodi espansi avvicinano quelle dei database disgiunti, e la qualità della soluzione è spesso ottima, o molto vicina a quella ottima.

1.4.5 Miglioramento di euristiche da perimetro

Si è visto come gli algoritmi di perimetro possano avvantaggiarsi della maggiore informatività di un'euristica ottenuta sommando il valore noto di path-cost di un nodo di un perimetro costruito a profondità d a partire dal nodo obiettivo. Di fatto le ipotetiche euristiche ottenibili per crescenti profondità di perimetro costituiscono una famiglia di euristiche di informatività (e complessità) crescente. Non è noto a quale profondità si trovi il miglior compromesso tra complessità e informatività; l'aumento di complessità è dovuto al fatto che aumentando la profondità del perimetro aumenta contemporaneamente il numero di confronti da effettuare per calcolare l'euristica (si ricorda che bisogna prendere il valore minimo di distanza dai nodi del perimetro; delle tecniche lazy permettono di ridurre notevolmente il numero di confronti, che cresce dunque meno rispetto all'aumento esponenziale del numero di nodi del perimetro).

Le metodologie di valutazione delle euristiche potrebbero essere utilizzate per scegliere la distanza più conveniente.

1.5 Esempi di implementazione e euristiche di alcuni problemi classici

In questa sezione sono descritte le idee alla base delle funzioni euristiche usate negli esperimenti per i lavori presentati in questo documento, e alcune possibilità implementative. Si tratterà prevalentemente il problema N-puzzle, soprattutto per il gran numero di euristiche implementabili già note. Altro caso trattato è il labirinto e, benché si considererà una sola euristica, bisogna notare che la sua qualità notevolmente variabile nelle diverse istanze del problema garantisce comunque una grande varietà di casi.

È presentata con dovizia di dettagli la tecnica di implementazione delle suddette tecniche; spesso questo è un fatto trascurato in letteratura, ma merita di approfondimento perché affatto banale: come si vedrà sono veramente tanti i punti da definire, e conoscerne i dettagli è essenziale per comprendere in che modo e perché un'euristica possa essere più o meno efficiente di un'altra. Poter conoscere l'implementazione agevola inoltre la ripetibilità degli esperimenti.

1.5.1 N-Puzzle

Uno stato del N-Puzzle può essere rappresentato in diversi modi, ognuno più o meno adatto al calcolo delle euristiche: il modo più banale e comunemente usato è definire lo stato in un vettore unidimensionale contenente i numeri corrispondenti alla tessera nella posizione

data. Si consideri ad esempio la configurazione di 8-puzzle in figura 1.6;

	5	6
8	4	7
3	2	1

Figura 1.6.: una configurazione per l'8-puzzle

Questa configurazione, a cui spesso ci si riferirà con la seguente dicitura equivalente:

0 5 6
8 4 7
3 2 1

può rappresentarsi in un calcolatore con il vettore riga [0, 5, 6, 8, 4, 7, 3, 2, 1].

Si adotterà come configurazione obiettivo quella solitamente adottata da Korf, ossia il vettore, [0, 1, 2, 3, 4, 5, 6, 7, 8].

È possibile pensare ad altre innumerevoli rappresentazioni di uno stato del N-Puzzle. Un'altra possibilità sempre univoca ma un po' meno compatta è rappresentare la distanza di ogni tessera dalla posizione obiettivo in orizzontale e verticale. In questo caso la configurazione in figura sarebbe rappresentata dal vettore [(0, 0), (-1, -1), (2, -2), (-2, -1), (0, 0), (1, -1), (0, 1), (-1, 2), (1, 2)]. Questa seconda rappresentazione, che chiameremo anche *vettore delle incidenze*, ha una maggiore ridondanza ma può rivelarsi più efficiente nel calcolo online delle euristiche. Questa rappresentazione ha un ulteriore vantaggio pratico: permette una implementazione delle euristiche indipendente dall'obiettivo scelto.

Per quanto riguarda gli operatori, essi sono quattro (spostamento della casella vuota nella direzioni su, giù, sinistra, destra), ma per il fatto che non è possibile uscire dalla griglia sono ammessi tutti soltanto nelle caselle centrali; tre operatori sono ammessi nelle caselle laterali e due soli nelle caselle all'angolo. L'applicazione di un operatore risulta nell'effettuare uno scambio (swap) tra le casella vuota (*pivot*) e una adiacente e restituire la struttura risultante. La maniera più efficiente per implementare gli operatori è usare una tabella con, per ogni posizione della casella vuota, gli spostamenti ammessi e le posizioni da scambiare. Durante la ricerca, essendo gli operatori reversibili è possibile evitare di generare il padre di un nodo nella ricerca, riducendo così il branching factor effettivo di 1.

Un'altra possibilità per implementare un puzzle $n \times m$ è utilizzare n vettori di m elementi. Questa implementazione semplifica la scrittura e la leggibilità degli accessi, ma è da considerarsi sostanzialmente equivalente alla precedente, e la sua scelta è correlata all'efficienza del linguaggio scelto nel gestire gli array multipli.

Nel nostro caso è stata scelta la prima opzione avendo usato prevalentemente un linguaggio (python) che non ha un efficiente array statico implementato come può essere invece il caso del C o di Java.

1.5.1.1 Misplaced-tiles (tiles-out-of-place)

L'euristica misplaced-tiles è data semplicemente dal conteggio delle tessere fuori posto.

Operativamente si tratta solo di eseguire $n \cdot m$ controlli tra tessere del nodo attuale e il goal. La configurazione considerata come obiettivo rende il calcolo notevolmente efficiente, in quanto è sufficiente aggiungere una unità al computo totale quando l'indice i del vettore è differente dall'elemento in posizione i (si esclude la casella vuota dal computo).

È ammissibile e consistente, ma si tratta in ogni caso di un'euristica relativamente scadente: il valore massimo ottenibile è infatti pari a $n \cdot m - 1$.

1.5.1.2 Distanza di manhattan

È sicuramente la più nota euristica per l' N -puzzle, è possibile calcolarla in maniera efficiente e risulta notevolmente più informata dell'euristica misplaced-tiles. È ammissibile e consistente, e può essere utilizzata per risolvere qualsiasi configurazione del 15-puzzle in tempi ragionevoli sulle macchine attuali.

È costituita dalla somma delle distanze di manhattan (city block) di tutte le tessere dalla posizione goal. Ossia, la differenza tra la posizione in x (e y) della stessa tessera nel nodo corrente e nel goal (o, equivalentemente, la somma dei valori assoluti del vettore delle incidenze). Ad esempio la configurazione in figura 1.6 ha un valore di distanza di manhattan di 18.

Una caratteristica interessante della distanza di manhattan è che mantiene la parità della soluzione ottima (se la soluzione è a distanza pari il valore di manhattan sarà anch'esso pari).

L'implementazione più efficiente della distanza di manhattan durante la ricerca è quella incrementale: si sfrutta il fatto che il figlio di un nodo potrà avere un valore di manhattan inferiore o maggiore di un'unità rispetto al padre. Questo è dovuto al fatto che solo una tessera può essere spostata dal singolo operatore. È sufficiente dunque controllare se la tessera spostata si avvicina o meno all'obiettivo e sommare il valore a quello del padre.

Il valore medio riscontrato di distanza di manhattan è pari a 14,00 per l'8-puzzle e di 37 per il 15-puzzle.

1.5.1.3 Linear-conflict

È storicamente una delle prime tecniche denominate *manhattan correction*, e probabilmente la più efficace. Fu presentata nel 1992 insieme alle tecniche corner-tiles e last-moves in [HanssonMayerYung1992]. Corregge in parte una delle principali pecche di manhattan: essa infatti non tiene in nessun conto i conflitti da tessere; può essere infatti vista come il costo della soluzione del problema semplificato in cui non si tiene conto del fatto che una tessera non possa muoversi nella posizione occupata da un'altra.

L'euristica linear-conflict aggiunge due unità alla distanza di manhattan quando si individua un conflitto lineare. Si ha un conflitto lineare quando due tessere appartenenti alla stessa riga (o colonna) si ritrovano nella riga (colonna) giusta, ma invertite. È facile convincersi con un semplice esempio che questo caso rappresenta una sottostima certa di due unità alla distanza di manhattan;

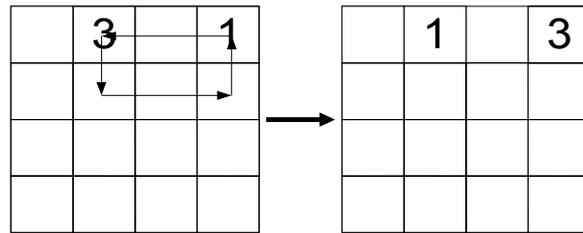


Figura 1.7.: Conflitto lineare tra le tessere 1 e 3

per risolvere la posizione di conflitto lineare delle tessere 1 e 3 in figura 1.7 (distanza parziale di manhattan quattro) sono necessarie almeno sei mosse, perché una delle tessere possa scavalcare l'altra e passare dall'altra parte. Operativamente il riconoscimento di un singolo conflitto lineare è fattibile verificando che due tessere sistemate nella giusta riga/colonna siano in posizione invertita l'una rispetto all'altra.

L'implementazione di linear conflict non è semplice e univoca come potrebbe sembrare, in quanto è necessario distinguere i casi in cui una tessera coinvolta in un doppio conflitto corrisponda a un effettivo aumento o meno del numero di mosse necessarie a risolvere il conflitto.

È possibile avere più di un conflitto lineare nella stessa riga-colonna, ma semplice il controllo di cui sopra non è più sufficiente: può infatti capitare che la soluzione di un conflitto permetta la sistemazione di tessere precedentemente invertite. In figura 1.8 si può vedere che, benché la tessera 3 sia invertita sia rispetto alla tessera 1 sia alla tessera 2, la sovrastima reale è di soltanto due mosse.

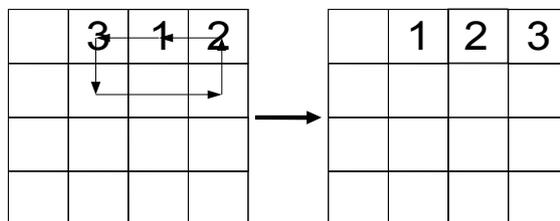
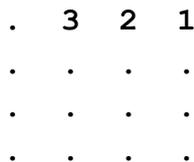


Figura 1.8.: Le due inversioni presenti nella prima riga non significano in questo caso la presenza di due conflitti.

Il conflitto sarebbe invece da rilevare per la configurazione (il punto indica una tessera qualunque, o la casella vuota):



in questo caso i conflitti rilevabili sono due, senza nessuna possibilità di sovrastima.

Un altro caso che potrebbe causare errori è quello in cui una tessera sia coinvolta in un conflitto sia in orizzontale che in verticale:

.	9	.	.
6	5	.	.
.	.	.	.
.	.	.	.

la tessera **5** è coinvolta in un conflitto lineare con la tessera **6** in orizzontale e con la tessera **9** in verticale. La distanza di manhattan è pari a quattro ma per risolvere questa configurazione sono necessarie almeno otto mosse: si è in presenza in questo caso di due conflitti lineari. Due conflitti possono essere certamente conteggiati in tutti i casi in cui una delle tre tessere coinvolte sia ben posizionata; perché le altre possano tornare a posto saranno infatti costrette a scavalcare la tessera ben posizionata, senza muovere quest'ultima; per entrambe ci sarà dunque un incremento di due mosse rispetto alla distanza di manhattan.

Le alternative implementative per l'euristica linear-conflict che permettono di garantire l'ammissibilità non individuando conflitti multipli non effettivi sono diverse. Una possibile scelta è non contare in ogni caso due conflitti per la stessa tessera; questa scelta risulta certamente in un'euristica meno accurata, e risulterà in un overhead non costante rispetto al semplice computo delle inversioni: bisogna mantenere un vettore¹ che memorizzi quando una tessera sia già stata coinvolta o meno in un conflitto; la variabilità dell'overhead è data dal fatto che questo può portare a una condizione di break anticipata eliminando di fatto alcune verifiche di inversione. Dagli esperimenti, rispetto all'implementazione che d'ora in poi si considererà come riferimento e descritta in seguito, questa versione semplificata risulta in un overhead medio inferiore del 10% circa per l'8-puzzle, e una diminuzione del valore medio di linear-conflict da 15,12 a 15,10 (0,02 la distanza di manhattan è mediamente 14,00). Bisogna comunque considerare che aumentando la dimensione del puzzle aumenta la probabilità di conflitti multipli. In un campione casuale di 200000 nodi del 15-puzzle si è trovata una differenza di 0,05 tra le due versioni (38,91 contro 38,86), invece per lo stesso campionamento con il 24-puzzle 0.08 (78.67 contro 78.59).

L'implementazione usata negli esperimenti è la seguente: viene eseguito un ciclo su tutto il vettore; se la tessera (non vuota) è posizionata sulla riga giusta, un ciclo in avanti cerca sulla stessa riga una tessera la cui posizione obiettivo sia sulla stessa riga. Se la tessera nella stessa riga ha valore inferiore (non zero) a quella di partenza, allora vengono sommate due unità al contatore di conflitti (inizializzato a zero). Il ciclo interno viene interrotto comunque anche se l'ultimo controllo non ha successo; questo garantisce di evitare di contare due volte un conflitto nella situazione in figura 1.8. Lo stesso ciclo è eseguito per colonne.

È possibile rendere più efficiente l'implementazione di linear-conflict memorizzando tutte le possibili coppie conflittuali in un piccolo database. Durante la ricerca può essere adottata, in linear-conflict come in altre manhattan corrections, una tecnica di calcolo lazy, che verifichi la condizione di conflitto solo per le tessere coinvolte nell'ultima mossa; questo migliorerebbe molto probabilmente l'efficienza del calcolo in fase di ricerca, in particolare per puzzle più grandi. In questo lavoro non si è sfruttata nessuna delle suddette tecniche incrementali per nessuna manhattan correction.

¹ In realtà sarebbe sufficiente un intero (32 bit basterebbero per un puzzle 6x5) avendo a disposizione operazioni sui bit, ma non tutti i linguaggi di programmazione mettono a disposizione questo strumento, e comunque si avrebbe un certo overhead per la gestione di maschere bit a bit.

1.5.1.4 Corner-tiles

La tecnica corner-tiles tiene conto del fatto che quando il pivot passa nella tessera all'angolo si ha a disposizione solo un movimento. Se dunque la tessera all'angolo è in posizione non corretta, ma lo è una delle tessere al suo fianco, la distanza di manhattan è in sottostima di almeno due mosse. Ad esempio nella configurazione

7	1	.
.	.	.
.	.	.

per poter far sì che la tessera **7** si sposti dall'angolo è necessario muovere necessariamente la tessera **1**. Un'altra mossa addizionale sarà necessaria per portare la **1** nella posizione originale. Possono dunque essere aggiunte senza dubbio due mosse alla parziale distanza di manhattan di tre. Il controllo di base consisterà dunque nel verificare che a fianco di una tessera all'angolo mal posizionata si trovi un'altra tessera ben posizionata. Diverse sono gli aspetti di cui tenere in conto

Bisogna fare particolare attenzione per l'angolo del pivot: se all'angolo ci fosse un **3** e non un **7** non si avrebbe una sovrastima certa perché l'ultima mossa potrebbe sistemare le due tessere in un passo.

3	1	2
0	4	5
6	7	8

Si deve notare che questo è vero non soltanto per l'ultima mossa, ma per tutte le combinazioni che possono arrivare alla configurazione di sopra senza muovere le tessere coinvolte nel possibile conflitto.

Le possibili principali scelte implementative sono due, in ordine di precisione decrescente: eseguire il controllo che la tessera a fianco al pivot non sia a distanza 1 dall'obiettivo, o non cercare direttamente i corner-tiles nell'angolo del pivot.

Anche nel caso di corner-tiles è necessario operare una scelta quando si trovino conflitti multipli: ad esempio la configurazione

7	1	5
.	.	.
.	.	.

porterebbe, applicando ingenuamente la regola descritta, a contare due condizioni di corner-tile. In realtà la tessera la sovrastima è in questo caso solo di due, perché è sufficiente spostare solo una volta la tessera **1** per muovere le tessere **1** e **5**. Questa condizione non può verificarsi in ogni caso per un puzzle più grande di 3×3 perché ovviamente la stessa tessera non potrà essere in quel caso adiacente a due angoli.

L'ultima condizione particolare da analizzare è quella in cui siano presenti non una ma due tessere ben posizionate accanto alla tessera all'angolo, ad esempio come nella configurazione:

```
. 1 8
. . 5
. . .
```

a quanto ammonta la sovrastima in questo caso? Sicuramente sia la tessera **1** sia la tessera **5** dovranno essere mosse per smuovere la **8** dall'angolo. La distanza di manhattan è due ma saranno necessarie almeno altre quattro mosse aggiuntive per riportare le tessere **1** e **5** nella posizione obiettivo. Questo caso dunque corrisponde effettivamente a una doppia sottostima da parte della distanza di manhattan.

L'implementazione scelta è la seguente: si esegue il controllo di base per ogni angolo; nel caso di puzzle inferiori o uguali a 3×3 è restituito direttamente il valore di correzione 2 senza cercare ulteriori eventuali conflitti. Per l'angolo del pivot è eseguito il controllo aggiuntivo che la tessera all'angolo non possa essere riposizionata con una mossa (dunque nel caso del 8-puzzle non si rileverà un conflitto nel caso in cui la tessera in posizione obiettivo sia **1** e la tessera all'angolo sia **3**; lo stesso vale invertendo le condizioni delle tessere **3** e **1**).

La tecnica corner-tiles ha un'efficacia inferiore a linear-conflict per l'8-puzzle (distanza media di 14,65), ma è notevolmente efficiente, il che rende il suo utilizzo decisamente appetibile; sfortunatamente, la sua efficacia tende a diminuire all'aumentare delle dimensioni del puzzle (il numero totale di corner tiles rimane pressoché invariato per i puzzle dalle 15 caselle in poi, mentre lo spazio degli stati cresce esponenzialmente).

1.5.1.5 Corner-deduction

Corner deduction, [Ernandes2003], si può considerare come una tecnica complementare a corner-tiles; non è molto potente ma il suo computo è molto efficiente. Permette di aggiungere due mosse alla distanza di manhattan (o a corner-tiles) quando si verificano le seguenti due condizioni: la casella all'angolo e la casella posta sulla diagonale sono ben collocate, mentre non è ben collocata almeno una fra le due tessere al lato di quella all'angolo.

Corner-deduction trova la sua giustificazione nel fatto che la tessera al lato di quella all'angolo può essere bloccata nel suo movimento naturale dalle altre due tessere ben posizionate. Ad esempio nella configurazione

```
. 7 2
. 4 .
. . .
```

nel momento in cui si toglie la tessera 7 dalla posizione attuale la mossa successiva consisterà necessariamente nello spostamento di una tra le tessere **2** e **4**, se si considera l'inutilità del tornare nello stesso stato precedente. Può esserci più di un caso di corner-deduction a patto che la tessera in diagonale non sia condivisa da corner-deduction per diversi angoli (dunque fino a quattro casi per ogni configurazione dal 15-puzzle in su).

La probabilità di avere una condizione di corner-deduction non è molto alta (circa l'11% delle configurazioni), ma l'overhead è abbastanza contenuto: si implementa semplicemente con una serie di controlli per ogni angolo, escluso l'angolo di chiusura. La si userà sempre in combinazione con corner-tiles, dunque quando ci si riferirà all'euristica corner-deduction ci si riferirà sempre alla combinazione delle due. Questa combinazione fornisce un valore medio di 14,71 per l'8-puzzle.

1.5.1.6 Last-moves

L'euristica last-moves è un'altra tecnica di manhattan correction, dalla definizione forse inizialmente poco intuitiva ma molto semplice ed efficiente da implementare. Si applica alle istanze in cui il pivot si trova all'angolo nella configurazione obiettivo, e si basa sul fatto che l'ultima mossa debba avvenire in quell'angolo.

La distanza di manhattan deve essere aumentata di due quando la tessera che nella configurazione obiettivo sta al lato orizzontale dell'angolo di chiusura non si trovi nella prima colonna, e la tessera che nella configurazione obiettivo sta al lato verticale dell'angolo di chiusura non si trovi nella prima riga. Benché appaia controintuitivo la riga/colonna deve essere opposta a quella di appartenenza perché non si abbia una condizione di last-moves. Si tratterà semplicemente di stabilire la posizione di due tessere e verificare se la riga/colonna è quella indicata. Unico caso particolare è la configurazione obiettivo; in quel caso non c'è ovviamente sottostima anche per tessere nella riga (colonna) giusta.

Cerchiamo ora di comprendere meglio i motivi di last-moves; l'ultima mossa sarà, nel caso dell'8-puzzle, a partire da una delle configurazioni seguenti:

1	0	2	3	1	2
3	4	5	0	4	5
6	7	8	6	7	8

Si può vedere da un semplice esempio che quando entrambe le tessere **1** e **3** non siano nella riga/colonna indicata siano necessarie due mosse aggiuntive:

.	.	.
.	1	3
.	.	.

Se si considera come goal non la configurazione obiettivo, ma una delle due configurazioni di sopra, la distanza di manhattan non è **3** ma **4**; un'altra unità dovrà essere aggiunta per ultima mossa.

L'euristica last-moves può essere dunque vista come una distanza di manhattan che considera come obiettivo questi due stati, a cui poi viene aggiunta un'unità per conteggiare l'ultima mossa. È equivalente a usare l'euristica di manhattan in una ricerca di perimetro con profondità di perimetro pari a 1, in maniera però più efficiente.

L'implementazione di last-moves è notevolmente semplice: è sufficiente controllare che la

tessera **1** sia nella prima colonna, e che la tessera **3** sia nella prima riga. Inoltre deve essere verificato che non ci si trovi nella configurazione obiettivo. Nello specifico si è deciso di controllare che la distanza di manhattan (che deve essere in ogni caso calcolata) non sia zero prima di calcolare last-moves.

Una caratteristica interessante di last-moves è che si presenta con frequenza più elevata nelle versioni più grandi del puzzle: la probabilità che due tessere non si trovino in una determinata riga colonna infatti aumenta col numero di caselle. In media per l'8-puzzle il valore di last-moves è pari a 14,89 (+0,89 rispetto a manhattan), mentre su un campione di 1 milione di nodi per il 15-puzzle si è rilevato un valore medio di 38,13 (+1,13).

1.5.1.7 Non-linear-conflict

Non-linear-conflict è una tecnica di manhattan correction incontrata sempre in [Ernandes2003], pensata per essere di complemento a linear-conflict e essere inserita insieme ad altre tecniche correttive in un'euristica di più ampio spettro denominata *conflict-deduction*. Come lascia intendere il nome mira a rilevare i conflitti non lineari, ossia di tessere appartenenti a una diversa riga o colonna; si incontra un conflitto non lineare quando una tessera che per raggiungere la sua posizione naturale deve muoversi sia in orizzontale sia in verticale, incontra una tessera che ne ostacoli il cammino, perché ben collocata o perché si deve muovere in direzione diametralmente opposta. Si consideri la configurazione

```

. . . .
. . 10 .
. 9 5 .
. . . .

```

In quest'esempio la tessera **5** si incontra in posizione di conflitto non lineare con le tessere **9** e **10**. La distanza di manhattan è tre, ma per portare tutte le tessere nella giusta posizione sono necessarie almeno cinque mosse. È utile usare il vettore delle incidenze per individuare i conflitti non lineari. Nell'esempio proposto si avrebbe il vettore

```

. . . .
. . (0,1) .
. (0,0) (-1,-1) .
. . . .

```

Se iv è il vettore delle incidenze la condizione di conflitto non lineare è, per una tessera in posizione i :

```

iv[i][0] != 0 and
iv[i][1] != 0 and

```

```
(iv[i + iv[i][0]]==(0, 0) or iv[i + iv[i][0]]==(-iv[i][0], 0)) and  
(iv[i + iv[i][1]]==(0, 0) or iv[i + iv[i][0]]==(0, -iv[i][1]))
```

Gli *and* implementati con lazy evaluation (o, equivalentemente, degli *if* annidati) garantiranno che gli indici non escano fuori dall'array. Bisogna inoltre fare in modo che una stessa tessera non sia coinvolta in più di un conflitto.

La capacità di correzione di non-linear-conflict è inferiore a quella di linear-conflict (media 14,40 per l'8-puzzle), ed è anche maggiore l'onere computazionale rispetto ad essa. Ha una caratteristica comunque molto interessante: essa si integra molto bene con linear-conflict. È in altre parole probabile che venga trovato un non-linear-conflict per delle tessere per le quali non è stato individuato un linear-conflict e viceversa (si complementano a vicenda); inoltre l'onere computazionale di calcolare le due euristiche in cascata è inferiore a quello necessario a calcolarle separatamente, perché parte del calcolo eseguito per una può essere riciclato per l'altra.

1.5.1.8 Combinazione delle tecniche di manhattan-correction

Le numerose metodologie di correzione presentate possono essere naturalmente combinate insieme per ottenere euristiche ancora più informate. La somma brutta di tutte le euristiche risulta in un'euristica non ammissibile che sarà indicata col nome di *all-corrections*. Non è infatti banale combinare tutta l'informazione fornita da tutte le diverse tecniche mantenendo l'ammissibilità. Un metodo semplificato è impedire che possa essere contato più di un conflitto per le medesime tecniche (se una tessera è coinvolta in un conflitto la si marca e per quella tessera non potrà essere coinvolta in altri conflitti). Chiameremo l'euristica così ottenuta *conflict-deduction*, come in [Ernandes2003], benché non sia esattamente identica a quella usata negli esperimenti a cui si ispira. Si indicherà con *conflicts* la combinazione ammissibile di linear-conflict e non-linear-conflict, e *last-moves-linear-conflict* la combinazione non ammissibile di last-moves e linear-conflict.

La presenza di euristiche non ammissibili è importante nel momento in cui si cercherà di capire l'effetto delle non ammissibilità sulla capacità delle euristiche di informare la ricerca. In particolare in confronto di all-corrections e conflict-deduction potrebbe dare una risposta a una non semplice domanda: è più conveniente cercare di inserire il maggior numero possibile di informazioni magari correndo il rischio di rinunciare all'ammissibilità o è meglio ricercare l'ammissibilità a tutti i costi rinunciando a qualche possibile fonte di informazione aggiuntiva?

- *all-corrections*: semplicemente la somma di tutte le correzioni a manhattan. Non tenendo in conto nessuna interferenza l'euristica risultante è ovviamente non ammissibile.
- *conflict-deduction*: per combinare tutte le diverse correzioni senza avere sovrastima si impedisce che la stessa tessera sia coinvolta in più di un conflitto. La sequenza di applicazione delle euristiche determina l'efficienza della conflict deduction: è conveniente disporre le euristiche più efficienti e/o più efficaci, al principio della procedura, in modo che se già è stato individuato un conflitto si possa evitare il calcolo di euristiche successive meno efficienti. La sequenza adottata è corner-deduction, linear-conflict, last-moves, corner-tiles, non-linear-conflict.
- *conflicts*: si sfrutta in questo caso la discreta complementarità tra linear e non-linear

conflict. Non-linear conflict è calcolata soltanto se non è stato trovato nessun conflitto lineare. L'euristica denominata *raw-conflicts* è invece la somma non ammissibile delle due.

- *last-moves-linear-conflict*: la somma delle due euristiche probabilmente più efficaci: last-moves e linear-conflict. Si è usata un'implementazione non ammissibile.

1.5.2 Labirinto

Un labirinto è essenzialmente una griglia in cui a ogni transizione è associato un marcatore che indica se il tratto è percorribile o meno. Può essere rappresentato in tutti i modi con cui può essere rappresentato un grafo non orientato, o come una sorta di bitmap in bianco e nero.

Il modo più compatto di rappresentare un labirinto è una lista di adiacenze. Questa è la rappresentazione usata negli esperimenti; ha come svantaggio il fatto che non permette di implementare semplicemente euristiche elaborate. Per codificare un nodo è sufficiente indicare le coordinate x e y .

1.5.2.1 Distanza di manhattan

La distanza di manhattan tra la casella iniziale e il goal è un'euristica certamente ammissibile per un labirinto. Benché la fantasia possa suggerire altre euristiche per risolvere un labirinto è l'unica che si utilizzerà negli esperimenti. La capacità informativa di manhattan in un labirinto è notevolmente variabile a seconda della particolare istanza; per i labirinti perfetti è di molto inferiore alla euristica omonima per l'*N*-puzzle. D'altra parte il tempo di calcolo è irrisorio (la somma delle coordinate x e y).

2 VALUTAZIONE DI EURISTICHE: DEFINIZIONE DEL PROBLEMA E STATO DELL'ARTE

Questo capitolo è interamente dedicato a presentare i diversi metodi già sviluppati a disposizione dell'analisi e valutazione di funzioni euristiche; benché sia un argomento di rilevante interesse pratico per chi deve implementare algoritmi di ricerca e funzioni euristiche, non molto lavoro è stato svolto in merito, e si concentra in particolare sulla categoria delle euristiche ammissibili.

Il testo di riferimento del settore è senza ombra di dubbio *Heuristics* di Judea Pearl [Pearl84], forse l'unico libro completamente dedicato allo studio delle funzioni euristiche. In esso è riassunta e ampliata la conoscenza del tempo sulle funzioni euristiche, seguendo un approccio prevalentemente di tipo teorico-matematico; tre interi capitoli si occupano della valutazione dell'informatività delle stesse. Si tratta di un testo decisamente ostico per la sua rigosità, nel quale sono presenti una quantità incredibile di definizioni e di studi, che spesso si tralasciano per il fatto che, per studiare un problema esclusivamente in senso logico-matematico, spesso forse si impongono delle precondizioni che rendono impossibile

l'applicazione a qualsiasi campo pratico. Le definizioni iniziali di questo capitolo, come la dominanza e la dominanza stocastica, sono principalmente tratte da questo testo. Dopo aver scritto *Heuristics*, Pearl ha cambiato indirizzo di ricerca, e non è stato pubblicato niente di particolarmente rilevante sull'argomento per almeno un decennio. Il ricercatore che probabilmente più di tutti ha contribuito all'avanzamento delle teorie relative alle euristiche e la ricerca negli ultimi anni è Richard Korf: oltre ad aver proposto una serie di algoritmi tuttora allo stato dell'arte alcuni suoi lavori riguardo alle prestazioni degli algoritmi possono essere ben adattati allo studio delle funzioni euristiche; in particolare ha introdotto nel 2001 quello che è forse il primo metodo a priori di valutazione delle prestazioni di un algoritmo di ricerca [KorfReidEdelkamp2001]. Benché non si perda di vista il rigore matematico il campionamento dello spazio degli stati diventa la colonna portante dello studio delle euristiche, e questo lavoro ha influenzato l'approccio di un gran numero di nuovi metodi proposti a seguire.

La ricerca sulle prestazioni degli algoritmi si è dimostrata probabilmente più viva nel campo degli algoritmi di ricerca locale stocastica: esistono alcuni metodi in letteratura che permettono di studiare la difficoltà di un problema e le funzioni di valutazione. L'approccio seguito da questi metodi è fortemente basato sul campionamento dello spazio degli stati, e l'approccio è prevalentemente di tipo sperimentale. In questo capitolo si vedrà una breve introduzione di questi metodi; per chi fosse interessato ad approfondire l'argomento si consiglia il completo e chiaro [HoosStutzle2005]. Al riadattamento delle suddette tecniche alla ricerca globale è dedicata una intera sezione del successivo capitolo.

Lo studio dello spazio degli stati dal punto di vista delle euristiche è sicuramente all'avanguardia anche nel campo del planning; Hoffmann in [Hoffmann2001] e [Hoffmann2002] presenta un'attenta analisi nel campo dei problemi di pianificazione, solitamente dei problemi giocattolo ma che rispecchiano delle situazioni decisamente realistiche e dalla grande varietà. Queste analisi hanno parecchi punti in comune con gli studi di landscape degli algoritmi di ricerca locale stocastica.

Tutti i metodi a cui si è accennato si dedicano allo studio della capacità della funzione euristica, ma non tengono invece direttamente conto di un aspetto essenziale, ossia di correlare l'informatività di una f.e. con le prestazioni degli algoritmi. Si vedranno due studi indipendenti ([AyyoubMasoud2000] e [LinaresJunghanns2002]) che si interessano della stima di prestazioni, e che utilizzano l'approccio simile di suddividere la procedura algoritmica in sub-task; i primi seguono un approccio prevalentemente intuitivo-teorico e di validazione empirica, i secondi invece propongono un metodo matematico che, oltre a stimare i nodi espansi dall'algoritmo di perimetro BIDA* a priori, da una vera e propria stima del tempo di esecuzione.

Nella presentazione dei metodi più approfonditi si è data grande importanza alla parte sperimentale: dopo la descrizione formale di un metodo si cercherà sempre di evidenziare l'aspetto pratico-implementativo, e si vedranno i risultati di inediti esperimenti che hanno il duplice scopo di analizzare una collezione di euristiche diverse per uno stesso problema, e di dare un'indicazione sul modo in cui i metodi precedentemente descritti debbano essere applicati per avere dei risultati statisticamente attendibili, ad esempio quando si sia costretti a ricorrere a un campionamento parziale o a semplificazioni teoriche.

Il capitolo inizia con un'introduzione sulle principali questioni e problemi dello studio delle funzioni euristiche. Seguono le definizioni di dominanza informativa e stocastica, e la presentazione del metodo di Korf-Reid. Dunque si analizzano i metodi di stima delle

prestazioni, per presentare infine lo stato dell'arte sulla ricerca locale stocastica e nel planning.

2.1 La valutazione delle funzioni euristiche: considerazioni preliminari

Finalmente ci si occupa dell'argomento principale e più interessante di questo lavoro: la valutazione delle euristiche. La valutazione delle euristiche è un compito affatto semplice ma indubbiamente affascinante: l'euristica è conoscenza, una forma di conoscenza al servizio del problem-solving. Una buona euristica deve essere in grado di catturare l'esperienza acquisita su di un determinato problema, e di tradurla in una forma comprensibile da un calcolatore. Il compito di chi si cimenta a sviluppare una funzione euristica può essere visto (forse un po' romanticamente!) come una forma di avvicinamento della macchina all'intelligenza umana.

Il processo di soluzione dei problemi umano è prevalentemente euristico-mnemonico; volendo fare un paragone con la ricerca, noi utilizziamo una strategia di ricerca a memoria limitata, avvicinabile a un algoritmo simil-hill-climbing (dunque molto rudimentale). D'altra parte siamo in grado di possedere, nel momento in cui apprendiamo a risolvere un problema, una conoscenza euristica decisamente efficace. Volendo forzare un esempio, chiunque per trovare una strada casa all'accostarsi di un incrocio sceglie un'alternativa senza pensare che nel caso non abbia preso la strada migliore dovrà tornare indietro (backtracking), né tantomeno cerca di valutare tutte le vicine alternative possibili; si va avanti, e se si conosce la zona si sarà ben sicuri di arrivare a destinazione.

Il ragionamento umano dà vita a delle euristiche che rendono i nostri processi cognitivi in molti casi inavvicinabili da un sistema automatico, anche se non siamo in grado come un automa di tenere memoria di un gran numero di possibili alternative che permettano di correggere gli errori di valutazione. In alcuni ambiti applicativi la forza bruta messa a disposizione da un calcolatore permette comunque di prendere il sopravvento sulle capacità umane anche a livello di eccellenza, e questo succede quando una buona strategia è accompagnata dal frutto della conoscenza pregressa, sotto forma euristica o di memoria di soluzioni. L'esempio più famoso è probabilmente il caso del computer deep-blue, [DEEPBLUE_SITE] che qualche anno fa è riuscito ad avere la meglio contro colui che può essere annoverato come uno dei migliori se non il migliore maestro di scacchi vivente, Garry Kasparov. Deep-blue fu dotato di memoria su un numero spropositato di casi di gioco; l'avversario umano sapeva bene che l'unica possibilità di vincere contro quello che era a tutti gli effetti un gigantesco pattern-database perfettamente informato, era quella di indirizzare la partita su binari inesplorati: dal nostro punto di vista si può parlare proprio di indirizzare la ricerca in zone dello spazio degli stati in cui l'euristica sarebbe risultata necessariamente meno efficace, zone in cui la battaglia diventa tra la forza bruta della macchina e le superiori euristiche del maestro scacchista. Forza bruta che non deve essere comunque sottovalutata, perché attraverso la tecnica del look-ahead sappiamo bene che possono essere ottenute delle euristiche comunque molto informate anche in zone all'apparenza "buie", avendo a disposizione una buona potenza di calcolo. La battaglia fu discretamente alla pari, ma si risolse in quella che per certi versi può essere interpretata come una simbolica vittoria dall'intelligenza artificiale contro l'intelligenza umana. In realtà la vittoria fu più merito della grande capacità di calcolo e di memoria che frutto di una strategia "intelligente". Si può dire che il computer abbia vinto nel momento in cui ha avuto una memoria abbastanza grande e dei processori abbastanza potenti da compensare alla sua carenza di capacità di ragionamento euristico. D'altra parte si sta comparando una macchina con delle prestazioni umane a livello

di eccellenza: chiunque abbia provato a cimentarsi in una sfida con un qualunque programma di scacchi, già su macchine ormai obsolete avrà potuto apprezzare un livello di sfida molto al di sopra del giocatore medio; per chi fosse interessato ad approfondire la questione riguardo ai programmi di scacchi, [Schaeffer1997].

Tornando sui nostri passi, valutare una funzione euristica è innanzitutto determinare il grado in cui la conoscenza di un problema è stata sintetizzata in dei numeri, ma non solo. Come già si è accennato le euristiche sono il frutto di un compromesso tra capacità di informare e prestazioni. Ossia un'euristica più informata sarebbe sempre preferibile, ma come ben sappiamo per calcolare un'euristica bisogna ricorrere a una procedura, che potrebbe richiedere delle risorse che ne rendano il suo utilizzo poco conveniente. Per fare un esempio il path-cost della soluzione trovato per mezzo di una ricerca in ampiezza sarebbe un'euristica perfettamente informata, ma utilizzare questa euristica sarebbe a dir poco disastroso per la ricerca!

Dal punto di vista “filosofico”, se fosse possibile trovare sempre una procedura che traduca senza overhead la conoscenza teorica, nel senso di avere un metodo di calcolo che sia capace di tradurre ogni sforzo computazionale in un corrispettivo aumento di informatività, sarebbe sempre preferibile sfruttare la migliore euristica calcolabile. Per questo motivo la maggior parte dei metodi di valutazione delle euristiche si concentrano esclusivamente nello studio dell'informatività, intesa come accuratezza distribuita della stima del path-cost. Questo è un problema di per sé non semplice, a parte i casi più ovvi di netta superiorità il compito di determinare quale sia più informata tra delle euristiche difficilmente si risolve in una sentenza definitiva: tutte le euristiche compiono un numero di errori consistente, ma l'effetto di questi è dipendente essenzialmente dalla loro distribuzione. La distribuzione degli errori commessi da un'euristica è d'altra parte imprevedibile e il campionamento è l'unico strumento a disposizione per ottenere qualche informazione. Le regole per collegare la distribuzione degli errori alla bontà dell'euristica sono ancora da definire, e sono uno degli oggetti di studio del terzo capitolo.

Un approccio diverso e più direttamente pratico per valutare le euristiche è quello di stimare le prestazioni degli algoritmi che ne fanno uso. Si potrebbe dire che un'euristica è migliore di un'altra quando con essa un algoritmo raggiunge le prestazioni migliori. Potrebbe sembrare un'ovvietà che un approccio di questo tipo, essendo in grado di applicarne il principio, sarebbe in grado di mettere la parola fine alla questione sul fatto se un'euristica h_1 sia preferibile a un'altra h_2 per risolvere un determinato problema. In realtà alcune questioni devono essere risolte: innanzitutto bisogna essere in grado di definire attraverso delle metriche le prestazioni di un algoritmo. Ad esempio si può proporre un modello come in figura 2.1:

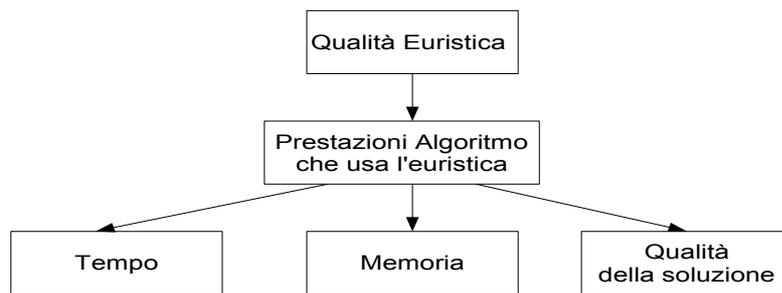


Figura 2.1: Valutazione euristica - schema generale

le metriche di base definibili sono dunque tre: tutte sono dipendenti dallo specifico algoritmo, e se tempo e memoria vanno in genere di pari passo, la qualità della soluzione, da tenere in conto nel caso odi euristica e/o algoritmo non ammissibili, è generalmente in contrasto con essi. D'altra parte la qualità della soluzione, intesa come grado di ottimalità, è spesso irrilevante, sia perché se l'euristica è abbastanza efficace la differenza media è minima (frazioni rispetto al reale path-cost), sia per il fatto che spesso interessa risolvere il problema senza interessarsi troppo se il costo non è il più piccolo possibile. La memoria è il parametro più limitante per un algoritmo come A*, mentre diventa irrilevante nel caso si consideri un algoritmo a memoria lineare come IDA*; per questo motivo si preferirà trascurare questo parametro supponendo di usare un algoritmo di questo tipo.

Una misura primitiva del tempo è moltiplicare il numero di nodi espansi (o generati), per il tempo di espansione (generazione) atomico. Si vedrà come questa relazione non sia proprio esatta, ma permette finalmente di separare l'aspetto implementativo dalle caratteristiche informative vere e proprie dell'euristica.

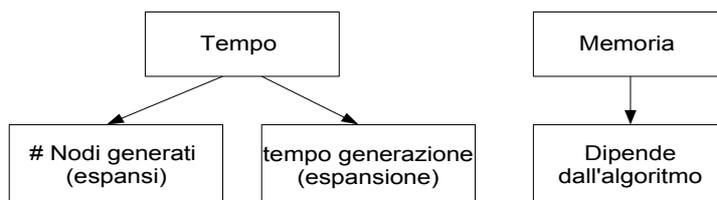


Figura 2.2: Elementi di dipendenza del tempo e della memoria

Il numero di nodi espansi dipende dalle caratteristiche informative dell'euristica (oltre che ovviamente dal particolare problema), e può essere ricavato in teoria per mezzo di ragionamenti logico-matematici, senza dover considerare alcuna variabile relativa all'ambiente esterno.

Il tempo di espansione di un nodo dipende invece da scelte implementative del problema (tra cui si include l'ambiente software), dell'algoritmo, e dall'euristica. Il suo valore assoluto dipende inoltre dalla particolare macchina usata¹. Le caratteristiche implementative sono a loro volta fortemente legate all'ambiente software utilizzato, ossia gli strumenti messi a disposizione dal linguaggio di programmazione. Quest'aspetto non deve essere assolutamente

¹ Intesa qua come combinazione di hardware e sistema operativo.

trascurato perché la differenza di prestazioni può essere di ordini di grandezza semplicemente cambiando strumento implementativo.

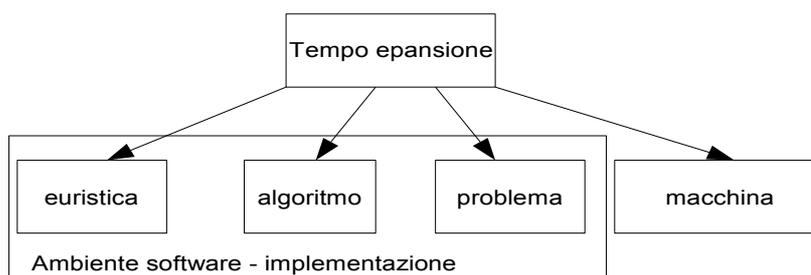


Figura 2.3.: Elementi di dipendenza del tempo di espansione di un nodo

In questa sede si vuole esaminare l'influenza delle scelte implementative e dell'ambiente software sulle prestazioni in termini di tempo. Sono state testate 6 diverse implementazioni di IDA* per la risoluzione del 15-puzzle utilizzando l'euristica di manhattan. Tre di queste sono la realizzazione in tre linguaggi diversi (C, Java 5.0 [JAVA_SITE], Python 2.4 [PYTHON_SITE])¹ del codice utilizzato da Korf² nei suoi esperimenti. Questa implementazione sfrutta tutte le possibilità di ottimizzazione realizzabili, come tabelle per gli operatori e un calcolo incrementale basato sempre su tabella di manhattan. Nessuna o quasi allocazione di memoria runtime è usata. Le restanti tre implementazioni si basano invece sulla filosofia a oggetti e dunque sulla flessibilità e generalità: realizzare uno stato per mezzo di un oggetto nodo permette infatti di usare le stesse procedure-funzioni-metodi per risolvere qualsiasi problema. Un oggetto nodo dovrà essenzialmente memorizzare lo stato, fornire il servizio di espansione, ed eventualmente calcolare il proprio valore euristico, e un nodo con la stessa interfaccia può essere sostituito in maniera trasparente ad un altro per risolvere un qualsiasi problema. Questo approccio ha un'efficienza notevolmente inferiore al precedente; il fatto di mantenere il framework generale costringe a rinunciare a tutta una serie di ottimizzazioni che dipendono dal singolo problema, e la scelta di rappresentare lo stato con un oggetto costringe ad allocare dinamicamente ogni singolo nodo, con conseguente overhead. I linguaggi utilizzati nei tre esempi sono C++, Python, Smalltalk VisualWorks® 7.3 [VISUALWORKS_SITE], la struttura a oggetti è molto simile, e per tutti è stato implementato il calcolo incrementale della distanza di manhattan.

In tabella 2.1 si può vedere il confronto del tempo di soluzione di un'istanza medio-semplice di 15-puzzle. La macchina è un portatile con processore Pentium M a 1.5 Ghz, e sistema operativo Windows XP SP2 HE. Il valore riportato è una media di cinque tentativi.

¹ Come riferimenti per la programmazione in Python, [ArmanoPython], [DowneyElkner2002].

² Il codice originale è scritto in C, gli altri sono stati realizzati con una traduzione diretta utilizzando le strutture più performanti messe a disposizione.

	<i>Time (sec)</i>	<i>Exp. nodes¹</i>	<i>Nodes/sec</i>
Static/optimized - C	0,02302	262.910	11.418.875
Static/optimized - java 5.0	0,03640	262.910	7.222.802
Static/optimized - python 2.4	1,285	262.910	204.599
Dynamic - C++	1,722	448.672	260.582
Dynamic - smalltalk Visualworks 7.3	3,012	408.952	135.769
Dynamic - python 2.4 ²	69,44	400.817	5.772

Tabella 2.1. Confronto prestazioni per diverse implementazioni dell'algoritmo IDA* per il 15-puzzle.

Come si può vedere la differenza di prestazioni è notevole, oltre quello che un iniziale buon senso potrebbe immaginare. Il rapporto tra la peggiore e la migliore implementazione è di quasi 2000 volte (200000%)! Questo fatto è un'ulteriore conferma di quanto sia difficile valutare gli algoritmi di ricerca: se si hanno questi scarti sulla stessa macchina, stessa istanza dello stesso problema, usando la stessa euristica e lo stesso algoritmo di ricerca, ci si può immaginare quale può essere la variabilità cambiando uno solo di questi parametri. Questi dati suggeriscono inoltre che non esiste nessun metodo possibile per stimare il tempo di espansione di un nodo se non quello empirico.

L'ultima considerazione è relativa al confronto delle prestazioni delle euristiche: ovviamente questo confronto sarebbe auspicabile che sia realizzato a parità di implementazione, ma non sempre questo è possibile al 100%. Innanzitutto l'implementazione di ogni euristica sarà necessariamente diversa; in questo caso la parità di condizioni potrà riferirsi al fatto che l'implementazione sia per entrambe la più ottimizzata possibile. In realtà anche questa definizione non è completamente esente da possibili disparità legate a fattori implementativi, in quanto per ottimizzare il calcolo di un'euristica potrebbe essere necessario usare una diversa codifica dello stato. Questo e altri effetti secondari non saranno in ogni caso tenuti in conto in questa sede, ma devono invitare a considerare con cautela i dati riguardo al tempo di esecuzione.

2.1.1 Framework sperimentale

Prima di iniziare la descrizione dei metodi si vuole accennare brevemente alle caratteristiche del framework usato per le sperimentazioni, che si potranno apprezzare come vero e proprio asse portante di questo lavoro. Quando ci si imbatte in un lavoro di ricerca sperimentale di questo tipo, in cui si è voluto offrire una base di sperimentazioni relative a un'ampia collezione di metodi esistenti e inediti, il parametro più importante da tenere in considerazione è sicuramente avere un framework sperimentale di base trasparente al particolare metodo usato. Si è visto inoltre il gran numero di variabili implementative da tenere in considerazione, e che potrebbero inficiare il corretto confronto delle euristiche e dei metodi.

1 Non deve sorprendere se il numero di nodi espansi possa essere diverso per diverse implementazioni dello stesso algoritmo: la ricerca depth-first potrebbe risultare in alcuni casi più "fortunata" visitando i nodi in profondità prima nella direzione della soluzione.

2 Si può notare un divario notevole di prestazioni rispetto ai risultati riportati in Appendice C. Esso è dovuto al fatto che in questo caso non si è fatto uso del modulo esterno di ottimizzazione di codice Psycho [PSYCO_SITE].

Per questi motivi si è scelto di usare un linguaggio ad oggetti potente e flessibile come python come base per l'applicazione. Le sue caratteristiche di scripting permettono di diversificare rapidamente gli esperimenti, e possiede tutti gli elementi di programmazione di alto livello necessari a una strutturazione modulare e generale del codice. Inoltre sono stati utilizzati comodi strumenti open source per il calcolo matriciale (Numeric), per la visualizzazione e salvataggio di grafici [pylab]. Si è usato l'ambiente di sviluppo Eclipse 3.2 [ECLIPSE_SITE], attraverso l'estensione per Python Pydev [PYDEV_SITE]. La comodità e la flessibilità si pagano d'altra parte con un'efficienza molto al di sotto di altre possibili alternative, che limita di fatto la dimensione dei problemi che possono essere studiati in tempi ragionevoli. In realtà questo è un handicap molto meno limitante di quanto si potrebbe pensare, al livello di un lavoro che non è di applicazione concreta dei metodi ma di testing e di confronto degli stessi: studiare quale euristica sia migliore per risolvere l'8-puzzle non è di certo il punto di arrivo di un selettore di euristiche, che auspicabilmente dovrà occuparsi di problemi di dimensione molto superiore, ma testare il selettore sul 8-puzzle permette di confrontare le predizioni dello stesso con dei completi dataset di dati reali. Per contenere il degrado di prestazioni si è inoltre utilizzato il modulo di ottimizzazione di codice Python Psyco [PSYCO_SITE], e alcune parti particolarmente onerose sono state reimplementate in dll scritte in C/C++ (ad esempio la ricerca con manhattan) e integrate con l'applicazione modulo ctypes [CTYPES_SITE].

Un processo di analisi segue i passi fondamentali:

1. definizione opzioni - problema
2. campionamento
3. esecuzione analisi
4. visualizzazione - salvataggio

I componenti principali dell'applicazione, che sfrutta appieno caratteristiche avanzate di programmazione a oggetti per rendere i vari task indipendenti l'uno dall'altro sono:

- *Nodo*: incapsula uno stato del problema. I servizi principali sono espansione, calcolo dell'euristica (che può essere settata), generazione di un nodo casuale dello stesso tipo e altre funzioni dedicate alla visualizzazione e al campionamento.
- *Analizzatore*: una classe con interfaccia standard che si occupa di eseguire l'analisi per cui è progettato, e di renderne fruibili i risultati. Riceve in ingresso le opzioni di analisi e un'istanza del nodo del problema che si deve studiare, e permette di visualizzare, salvare e graficare il risultato dell'analisi.
- *Campionatore*: esegue il campionamento dello spazio degli stati richiesto (vedere appendice per i dettagli sui diversi metodi). Fornisce l'accesso ai campioni attraverso uno stream iterabile. È importante avere a disposizione la possibilità di caricare i campioni runtime, perché un numero consistente di campioni potrebbe saturare rapidamente la memoria del calcolatore.
- *Problema*: offre il servizio di ricerca della soluzione, con la strategia definita. È stata implementata una versione di solutore con strategia IDA* che sfrutta una dll scritta in C (collegata attraverso il modulo Python ctypes) per offrire prestazioni più elevate.

Ognuno di questi componenti è una classe interfaccia che ha diverse specializzazioni che ne definiscono il comportamento particolare. È possibile così sostituire uno dei componenti

senza dover modificare il comportamento degli altri, come ad esempio eseguire analisi diverse con lo stesso campionamento. Un'utilità che è stata implementata è la possibilità di eseguire la stessa analisi in cascata cambiando solamente l'euristica, visualizzando poi il tutto nello stesso grafico.

Come già accennato sono stati implementati i problemi N-puzzle di qualsiasi dimensione e del labirinto, e tutte le euristiche descritte nel precedente capitolo. Inoltre è stata implementata la tecnica di miglioramento da perimetro su alcune euristiche per entrambi i problemi.

2.2 Dominanza informativa di una funzione euristica

Il concetto di dominanza informativa è la definizione basilare di informatività di una funzione euristica ammissibile. La funzione euristica perfettamente informata assegnerà ad ogni nodo il valore reale del costo della soluzione ottima, per cui

$$h^*(n) = C^*(n) \forall n, \quad (2.1)$$

dove $C^*(n)$ è il costo della soluzione ottima a partire dal nodo n .

Una funzione euristica sarà localmente tanto più informata quanto più il suo valore calcolato su un particolare stato sia più vicino a quello ottimo, ossia quando il suo valore sia più accurato nella stima del path-cost. Per un'euristica ammissibile, essendo sicuramente inferiore all'euristica perfettamente informata, la maggiore o minore vicinanza alla soluzione ottima equivale a un valore più alto o più basso. Se si rileva una maggiore informazione in ogni stato di un'euristica rispetto a un'altra allora si parla di dominanza informativa della prima sulla seconda.

Definizione prima di dominanza informativa: Essendo per un'euristica ammissibile $h(n) \leq h^*(n) \forall n$, date due funzioni h_1 e h_2 la prima dominerà sulla seconda (è più informata) quando valga

$$h_1(n) > h_2(n) \forall n. \quad (2.2)$$

Questo fatto è abbastanza intuitivo; è più informata la funzione euristica il cui valore si avvicina di più e per ogni nodo alla funzione euristica perfettamente informata. Questa formulazione è quella fornita da Pearl in [Pearl84].

Successivamente questa formulazione, decisamente restrittiva per il fatto di non permettere a neanche un valore delle funzioni euristiche sotto confronto di essere il medesimo (addirittura non permette di asserire che manhattan per l'N-puzzle sia più informata di tiles-out-of-place!), è stata rivisitata in seguito anche dallo stesso Pearl come

Definizione seconda di dominanza informativa: L'euristica h_1 sarà dominante su h_2 se:

$$h_1(n) \geq h_2(n) \forall n, \exists n' : h_1(n') > h_2(n'). \quad (2.3)$$

Secondo questa definizione la funzione euristica dominante può anche essere avere in qualche nodo lo stesso valore di quella dominata, quando esista almeno un nodo in cui il valore sia superiore.

Questa formulazione è decisamente meno restrittiva: è infatti molto comune che, soprattutto nei nodi più vicini alla soluzione, il valore di funzioni euristiche anche

considerevolmente differenti dal punto di vista dell'informatività sia lo stesso.

Restano comunque i principali limiti di questo metodo di confronto: spesso non si avrà una relazione di questo tipo (per qualsiasi nodo), ovverosia potranno esserci dei nodi in cui il valore di una delle due sia maggiore, altri in cui lo sia l'altra. Il confronto esaustivo di tutti i valori euristici per la totalità dello spazio degli stati potrebbe non essere un'operazione comodamente realizzabile quando questo sia notevolmente esteso.

Non fornisce inoltre di per sé un'indicazione quantitativa della maggiore informatività di un'euristica rispetto all'altra.

2.3 Metodi basati sulla funzione di distribuzione di probabilità

2.3.1 La funzione di distribuzione di probabilità

La funzione euristica può essere vista come una sorgente rumorosa di informazione del costo della soluzione ottima. La stima del valore ottimo data dall'euristica sarà affetta da un errore, necessariamente per difetto nel caso di euristiche ammissibili, che si riscontra sui vari nodi in maniera localmente imprevedibile, ma seguendo invece alcune interessanti regolarità dal punto di vista statistico.

Una importante conseguenza è che la distribuzione probabilistica dei valori euristici è strettamente correlata con la sua informatività. Intuitivamente, se è vero che una funzione euristica è più informata quando il suo valore sia più vicino alla funzione euristica perfettamente informata questo sarà vero anche dal punto di vista probabilistico.

Definizione di funzione di probabilità: la funzione di probabilità di un'euristica h , $p_h(x)$, è data da:

$$p_h(x) = p(h=x) \quad (2.4)$$

La formulazione usualmente utilizzata è la probabilità cumulativa (indicata solitamente semplicemente come *probability distribution* o *overall distribution*):

Definizione di distribuzione di probabilità (probabilità cumulativa):

$$P_h(x) = p(h \leq x) \quad (2.5)$$

In [Pearl84] è utilizzato intensivamente il concetto di probability distribution; entra nella definizione di dominanza stocastica delle funzioni euristiche, e per una serie di analisi della complessità dell'algoritmo A^* per funzioni euristiche ammissibili, quando l'errore sia dipendente dalla distanza. Estende inoltre la teoria anche al confronto di euristiche non ammissibili.

Si vedrà in seguito come la distribuzione di probabilità può essere utilizzata per il confronto diretto (principalmente grafico) delle funzioni euristiche, e per calcolare il numero medio di nodi espansi dall'algoritmo IDA*.

2.3.2 Dominanza stocastica di una funzione euristica

La dominanza stocastica è un'applicazione del concetto di dominanza, che permette una più semplice applicazione pratica facendo uso della distribuzione di probabilità.

Definizione di dominanza stocastica: date 2 funzioni euristiche ammissibili h_1 e h_2 la prima è stocasticamente più informata della seconda quando valga:

$$P_{h_1}(x) \leq P_{h_2}(x) \quad (2.6)$$

Dimostrazione:

Si consideri il valore più piccolo possibile per l'euristica h_2 , h_{2min} ; può essere definito un insieme $S_{min} \subset S$ per cui

$$\forall n \in S_{min}, n' \notin S_{min} \Rightarrow h_2(n) < h_2(n') \quad (2.7)$$

Essendo h_1 più informata di h_2 sarà anche

$$h_1(n) \geq h_2(n) \quad \forall n \in S. \quad (2.8)$$

Se per assurdo fosse $P_{h_1}(h_{2min}) > P_{h_2}(h_{2min})$ dovrebbe esistere almeno un nodo $n' \notin S_{min}$ per cui valga la relazione $h_1(n') < h_2(n')$, fatto non possibile per l'ipotesi (2.7). Per i valori superiori a h_{2min} la dimostrazione è analoga: può essere fatta ricorsivamente partendo dal valore successivo h_{2succ} ; l'insieme di nodi che soddisfano la condizione $h_2 \leq h_{2succ}$ comprenderanno l'insieme dei valori precedenti S_{prec} e l'insieme dei nodi con valori successivi di euristica h_{2succ} , S_{succ} . Se $P_{h_1}(h_{2prec}) > P_{h_2}(h_{2prec})$ dovrà essere anche $P_{h_1}(h_{2succ}) > P_{h_2}(h_{2succ})$ perché valga ancora la (2.7).

Il confronto può essere fatto con un semplice controllo visivo dei grafici delle funzioni di probabilità: la funzione che risulta "più a destra", o equivalentemente "al di sotto" è quella che corrisponde all'euristica più informata.

La relazione di dominanza stocastica (2.6) è nella sostanza equivalente alla relazione (2.3), ma questo tipo di formulazione ha una rilevanza pratica molto importante: la distribuzione di probabilità può essere facilmente approssimata attraverso un campionamento di anche una porzione relativamente ristretta dello spazio degli stati, e permette, tramite l'ispezione visiva del grafico, di avere un'idea anche quantitativa della differenza di informatività tra le euristiche sotto confronto.

Quando la relazione non sia valida in termini assoluti (ossia non sempre una delle funzioni ha valore superiore all'altra) non si ottiene alcuna indicazione direttamente utile dall'ispezione delle funzioni di distribuzione: si può ragionevolmente asserire che in questo caso potrebbe dare un discreto vantaggio utilizzare come euristica una funzione che sia la maggiore delle due punto per punto, ma un'analisi più approfondita è sicuramente indicata. Una possibilità potrebbe essere fare il confronto calcolando il numero medio di nodi espansi col metodo di Korf-Reid.

2.3.2.1 Risultati sperimentali

Si presenteranno ora una serie di grafici della distribuzione di probabilità per l'N-Puzzle con diverse euristiche ammissibili. Si vedrà il confronto per puzzle di diverse dimensioni, con lo scopo di analizzare come la crescita del dominio del problema possa influenzare la relazione tra le informatività di diverse euristiche.

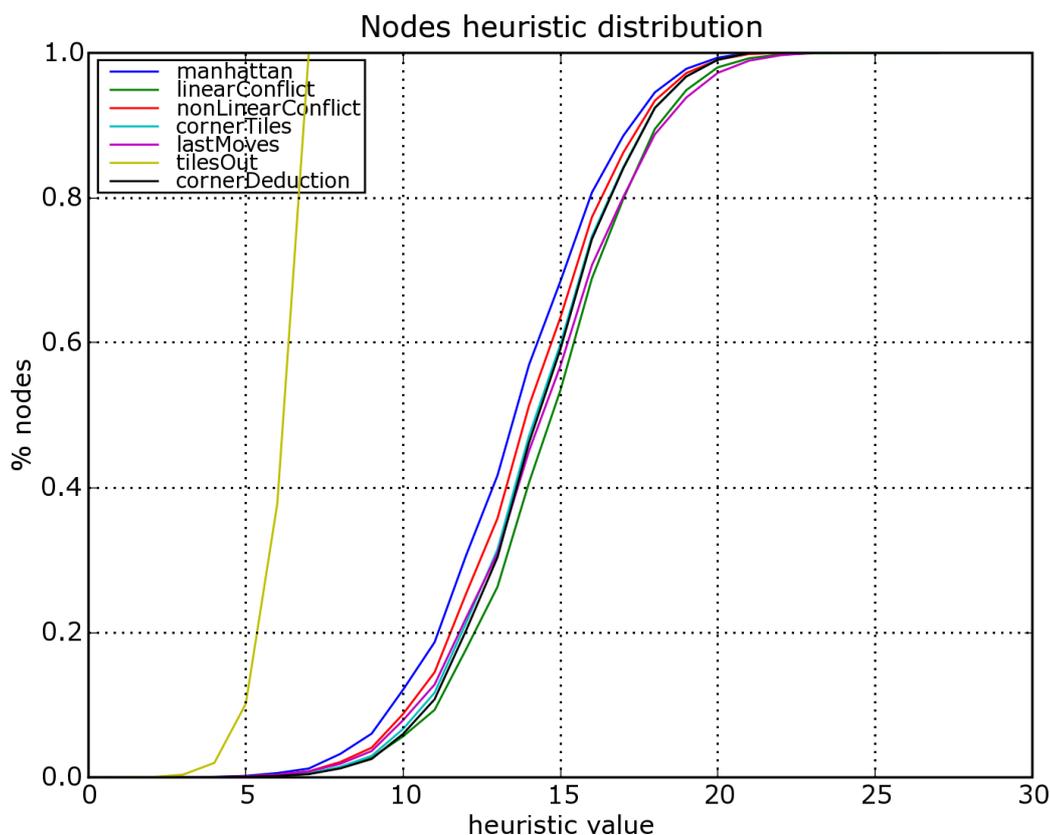


Figura 2.4.: distribuzione per alcune euristiche ammissibili; 8-puzzle

La figura 2.4 mostra la distribuzione calcolata enumerando tutto lo spazio degli stati per l'8-puzzle per tutte le euristiche ammissibili implementate, tra quelle ricavate da una singola correzione. Può essere subito notata la notevole differenza informativa tra l'euristica tiles-out-of-Junghannsplace e l'euristica di manhattan e le diverse correzioni: si trova inoltre la conferma che tutti i metodi di correzione alla distanza di manhattan forniscono delle euristiche stocasticamente più informate di quest'ultima. D'altra parte nessuna correzione da sola si dimostra stocasticamente più informata delle altre: la linear-conflict, che si mantiene al di sotto di tutte per un gran numero di valori, è superata a partire dal valore 18 dall'euristica last-moves, a sua volta al di sotto anche di corner tiles per valori più piccoli. In linea di massima non si può dire niente su quale euristica sia più informata tra last-moves e linear-conflict, anche se il buon senso suggerirebbe di preferire l'euristica linear-conflict perché si mantiene al di sotto per gran parte del grafico, e con uno scarto relativamente grande rispetto a quello rilevabile per i valori in cui si mantiene sopra last-moves. Questo risultato potrebbe suggerire una discreta complementarità di linear conflict e last-moves. Per quanto riguarda le altre correzioni si riscontra un'ambiguità ancora più marcata, che non permette di asserire nessuna sentenza riguardo alla maggiore o minore informatività dell'una rispetto all'altra. In figura 2.5 è invece rappresentata la distribuzione per le euristiche ottenute dalla combinazione di correzioni a manhattan. Si ricorda che soltanto le prime tre sono ammissibili; le euristiche non ammissibili hanno una distribuzione più allineata verso destra, ma questo fatto non deve

portare a concludere alcun assunto riguardo alla maggiore informatività di queste; le non ammissibilità infatti aumentano in genere il valore medio di distribuzione, ma questo aumento non è necessariamente correlato con una maggiore precisione.

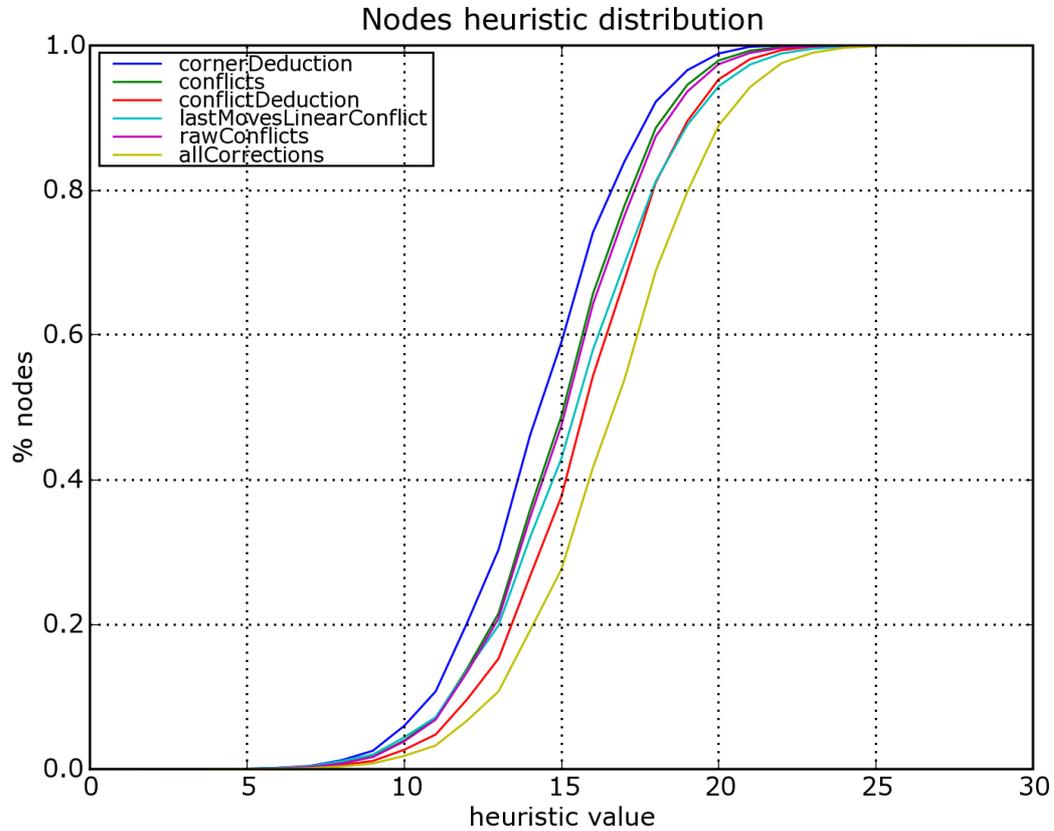


Figura 2.5.: distribuzione per combinazioni di manhattan corrections ammissibili e non; 8-puzzle

In seguito il risultato per le istanze più piccole di N-Puzzle, con cinque e sette tessere (figure 2.6 e 2.7).

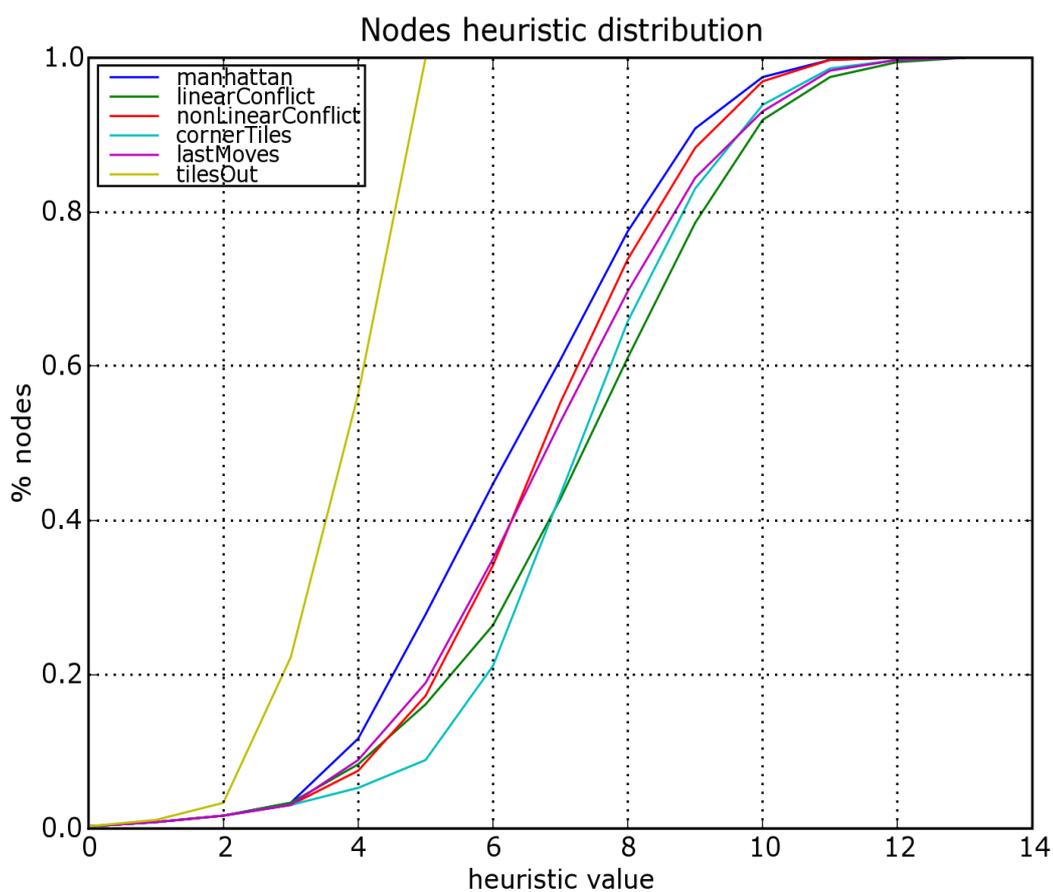


Figura 2.6.: Funzione di distribuzione per diverse euristiche – 5-puzzle

I rapporti non si mantengono uguali al caso precedente: come ci si poteva aspettare l'euristica corner-tiles si dimostra più efficace per queste piccole istanze di puzzle, così come last-moves sembra perdere parte dell'efficacia rispetto a linear-conflict.

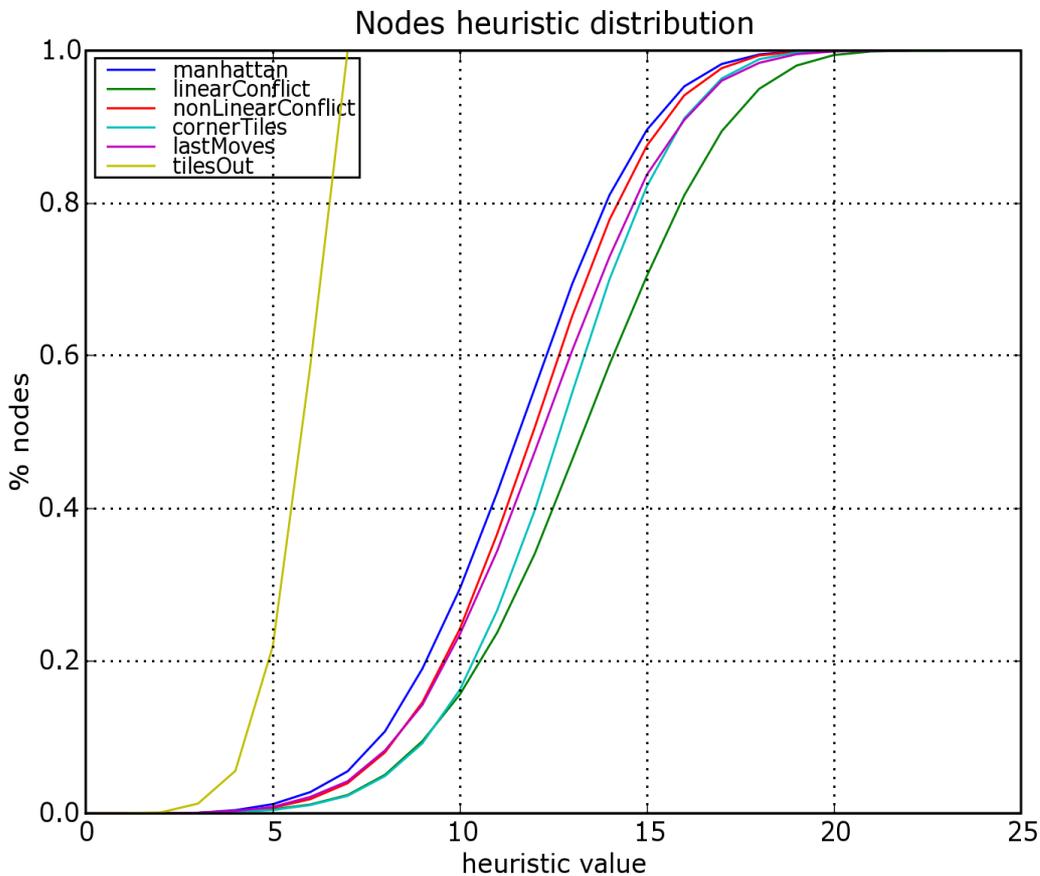


Figura 2.7.: Funzione di distribuzione per diverse euristiche – 7-puzzle

Prima di mostrare il risultato per il 15-puzzle è doveroso analizzare gli effetti del campionamento: il 15-puzzle ha infatti uno spazio degli stati di un'ampiezza tale da costringere a campionare una porzione dello stesso. Si vuole perciò capire innanzitutto come il numero di nodi campionati influenza la precisione nella stima della distribuzione di probabilità.

Per valutare l'attendibilità dei risultati campionati si è usata la seguente tecnica: dato un campionamento si N nodi si è calcolata la curva dell'errore medio, e la deviazione standard di un numero M di stime; questo secondo parametro deve essere tenuto bene in conto: se l'errore medio delle misure ottenute con un campionamento è basso ma la deviazione standard (σ) dei risultati dello stesso è elevata probabilmente è necessario considerare un numero maggiore di campioni. Una valutazione ragionevole è considerare sufficientemente stabile un campionamento che abbia una deviazione standard relativamente molto più piccola del valore attuale, ad esempio un ordine di grandezza. Sotto l'ipotesi di distribuzione gaussiana si ricorda che il 68% delle misure avranno uno scarto inferiore a σ , mentre al 99,7% l'errore si manterrà al di sotto di 3σ .

Nella figura 2.8 è mostrato l'andamento della deviazione standard per 100 campionamenti di N campioni della distribuzione di probabilità per l'euristica di manhattan per l'8-puzzle. Per brevità non è mostrato il risultato della media, che comunque non presenta offset significativo

per nessuno dei campionamenti effettuati.

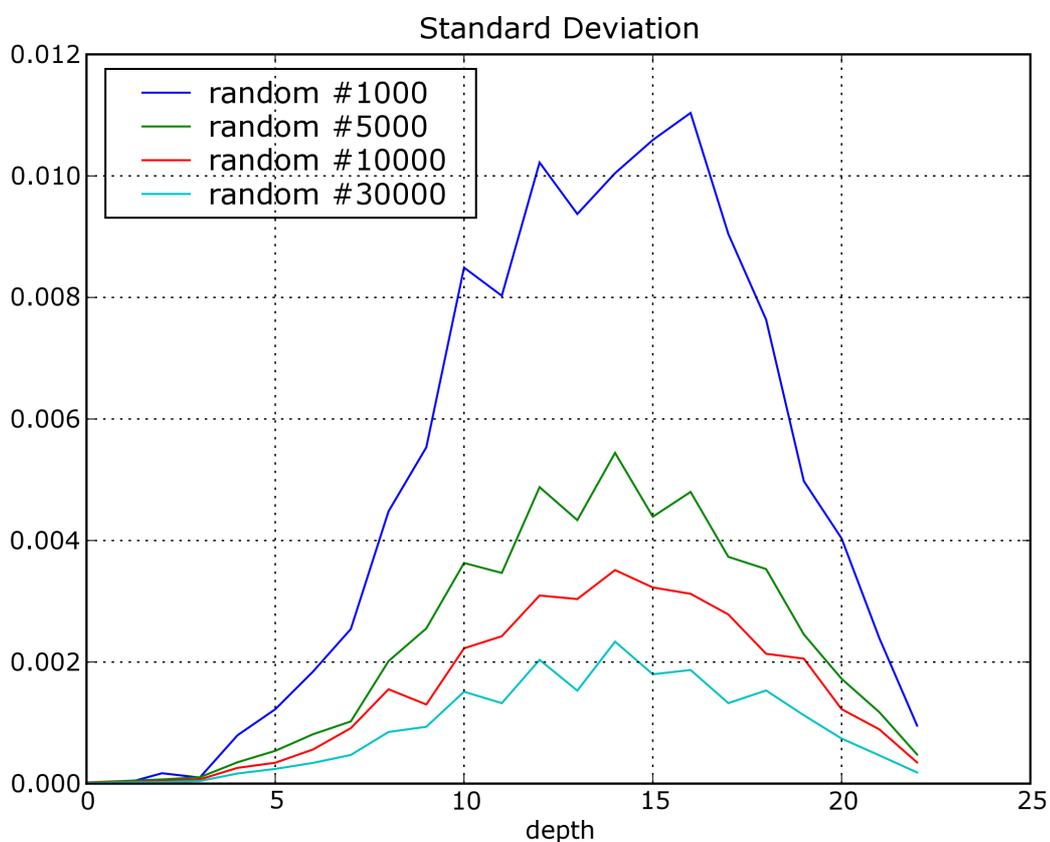


Figura 2.8.: deviazione standard del risultato di 100 campionamenti per diverse numerosità – 8-puzzle

La stabilità del risultato sembra crescere in maniera pressoché lineare col numero di campioni. Come si può vedere dalla figura 2.9 un numero di campioni di 2000 (1%) già permette di ottenere un campionamento attendibile secondo il criterio di cui sopra, per i valori più rappresentati di euristica.

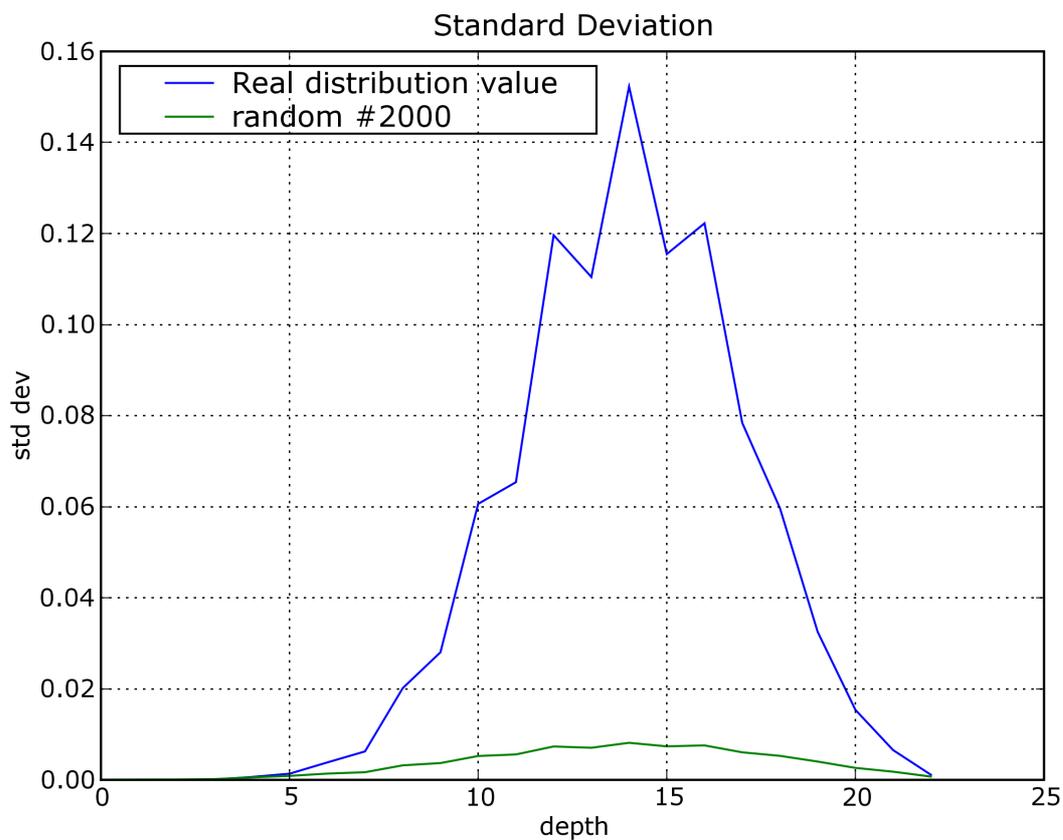


Figura 2.9.: confronto deviazione standard per 1000 istanze di 2000 campioni– valore reale

L'attendibilità del campionamento è sufficiente nelle zone in cui si ha una maggiore quantità di nodi, ossia valori euristici intorno a 14. Qualora sia necessaria una precisione relativamente alta anche per le probabilità più basse sarà necessario aumentare il numero di campioni. In figura 2.10 è presentato il risultato di un campionamento di 300000 nodi per il 15-puzzle.

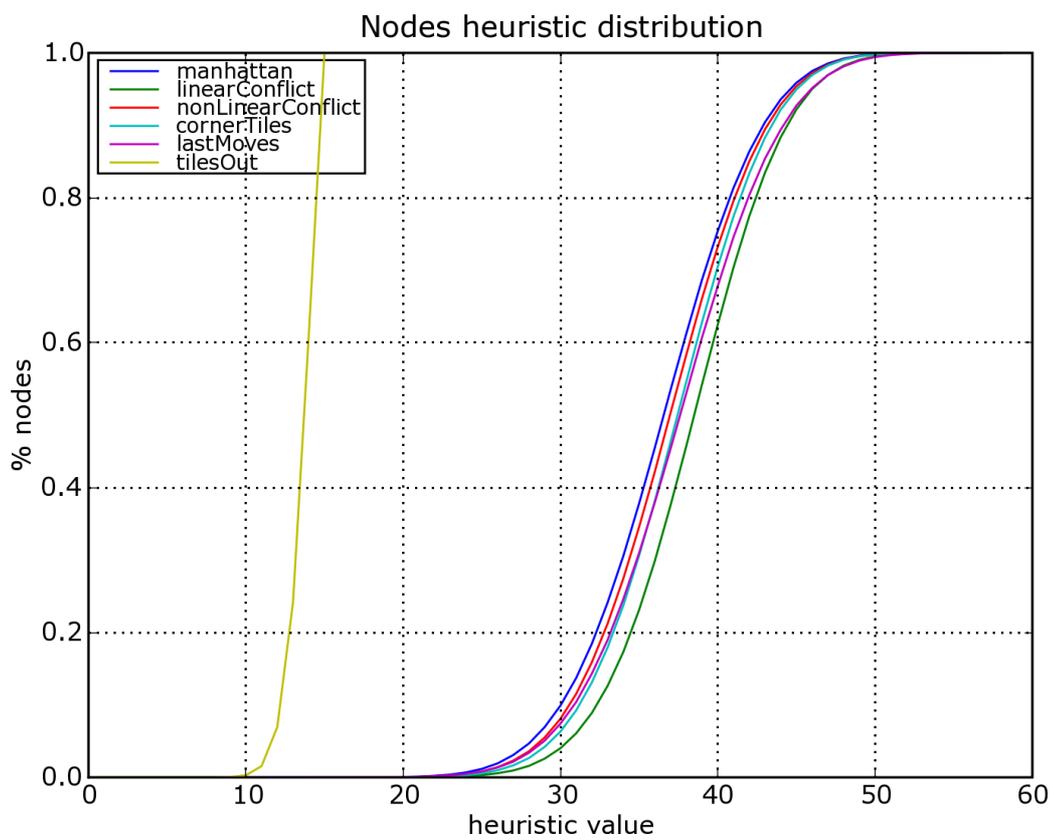


Figura 2.10.: distribuzione per diverse euristiche 15-puzzle; campionamento di 300000 nodi casuali

Ultimo risultato, la distribuzione di probabilità per le euristiche ottenute da strategia di perimetro per il 15-puzzle, usando l euristica manhattan a diverse profondità.

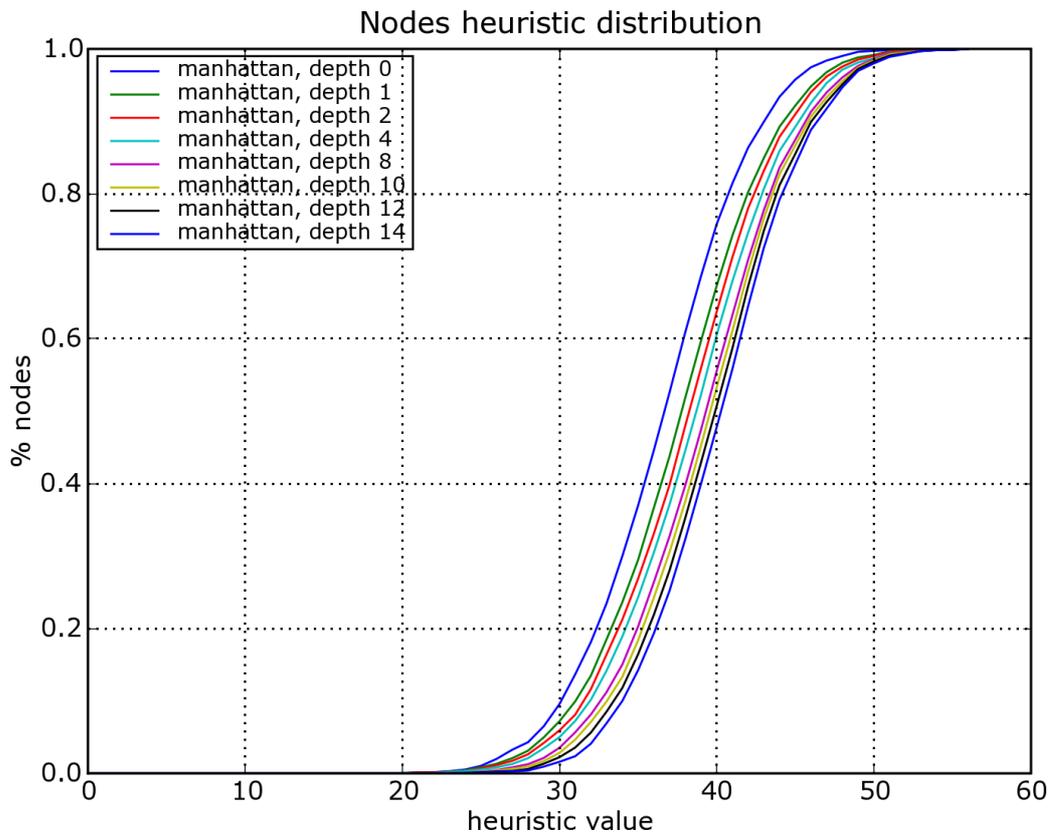


Figura 2.11.: Distribuzione nel 15-puzzle per diverse profondità di perimetro; campionamento di 300000 nodi casuali

La distanza 0 equivale alla normale manhattan e la distanza 1 alla correzione last-moves. Particolare interessante da notare è il fatto che le curve sembrano spostarsi in maniera lineare con la profondità del perimetro.

2.3.3 Confronto di funzioni euristiche non ammissibili

La valutazione di un'euristica non ammissibile è un problema decisamente più complesso rispetto alla controparte ammissibile. Questo è dovuto a due principali motivi: innanzitutto non è possibile determinare un valore limite superiore su cui si basano tutte le dimostrazioni di questo paragrafo; in secondo luogo una funzione non ammissibile porta a trovare soluzioni non ottime anche utilizzando un algoritmo come A*, di fatto estendendo il teorico confronto anche alla qualità della soluzione.

Solitamente si preferisce ignorare la qualità della soluzione, dando priorità all'euristica che consente le migliori prestazioni. Tipicamente le euristiche non ammissibili sono usate proprio con lo scopo di migliorare le prestazioni in problemi che altrimenti non giungerebbero al termine nei tempi prestabiliti per un gran numero di istanze (ad esempio in tantissimi problemi di planning).

2.3.3.1 Analisi stocastica delle euristiche non ammissibili

All'analisi delle prestazioni delle euristiche non ammissibili è interamente dedicato il settimo capitolo di [Pearl84]. Si cercherà in questa sede di riassumere le principali considerazioni e risultati. L'euristica si ricorda è definita non ammissibile quando presenta delle sovrastime rispetto al valore di path cost ottimo della soluzione. Le sovrastime hanno un effetto duplice sulla complessità degli algoritmi: esse infatti riducono la complessità quando si trovano in percorsi fuori da quello della soluzione, la aumentano invece nel caso contrario. Per questo motivo si deve considerare la complessità medio-asintotica per un gran numero di istanze: la stessa sovrastima può essere un vantaggio nella soluzione di alcune istanze, uno svantaggio in altre. In generale, una sovrastima indiscriminata si traduce in un degrado di prestazioni. Se si considerano le distribuzioni di due euristiche non ammissibili il fatto che una sia interamente alla destra di un'altra non implica affatto una maggiore informatività: lo spostamento potrebbe essere infatti causato principalmente da sovrastime, che non garantiscono affatto maggiore informatività quando sono più elevate.

Per evitare in qualche modo questa limitazione Pearl introduce il concetto di supporto superiore (higher support) r , ossia il valore più elevato riscontrato nei valori euristici. Due euristiche non ammissibili con lo stesso supporto superiore possono essere cioè confrontate graficamente per mezzo della distribuzione di probabilità. Ancora di più, se il supporto è diverso, le distribuzioni possono essere confrontate successivamente a una traslazione rigida che le porti ad avere lo stesso supporto superiore.

***Teorema:** date due euristiche in generale non ammissibili h_1 e h_2 la prima è più efficiente della seconda se la distribuzione della prima, traslata in modo da far coincidere il supporto superiore delle due, soddisfa rispetto alla seconda la condizione di dominanza stocastica (2.6).*

Dunque se dopo la traslazione la distribuzione di h_1 si mantiene punto per punto al di sotto rispetto ad h_2 questa è da considerarsi la più efficiente. Niente può essere affermato in caso contrario. Questo teorema è più che altro da considerarsi una supposizione, che si basa sul fatto che una maggiore concentrazione dei valori euristici vicino al massimo sia indice di una maggiore informatività.

Pearl propone inoltre una tecnica di miglioramento delle f.e. non ammissibili che si basa sullo stesso principio di introduzione di pesi dell'algoritmo w-A*. Utilizzando pesi inferiori a $\frac{1}{2}$ per l'euristica si può in teoria ottenere un'euristica ammissibile partendo da una non ammissibile. Il peso ottimo si può far dipendere dal supporto superiore:

$$w_0 = \frac{1}{2+r}.$$

2.3.3.1.1 Risultati Sperimentali

Di seguito, nella figura 1.12 il confronto di alcune euristiche ammissibili e non, applicando la traslazione del supporto superiore. Si ricorda che conflict-deduction e all corrections contengono tutte le correzioni a manhattan, la prima mantenendo l'ammissibilità, la seconda sommando indiscriminatamente tutte le correzioni.

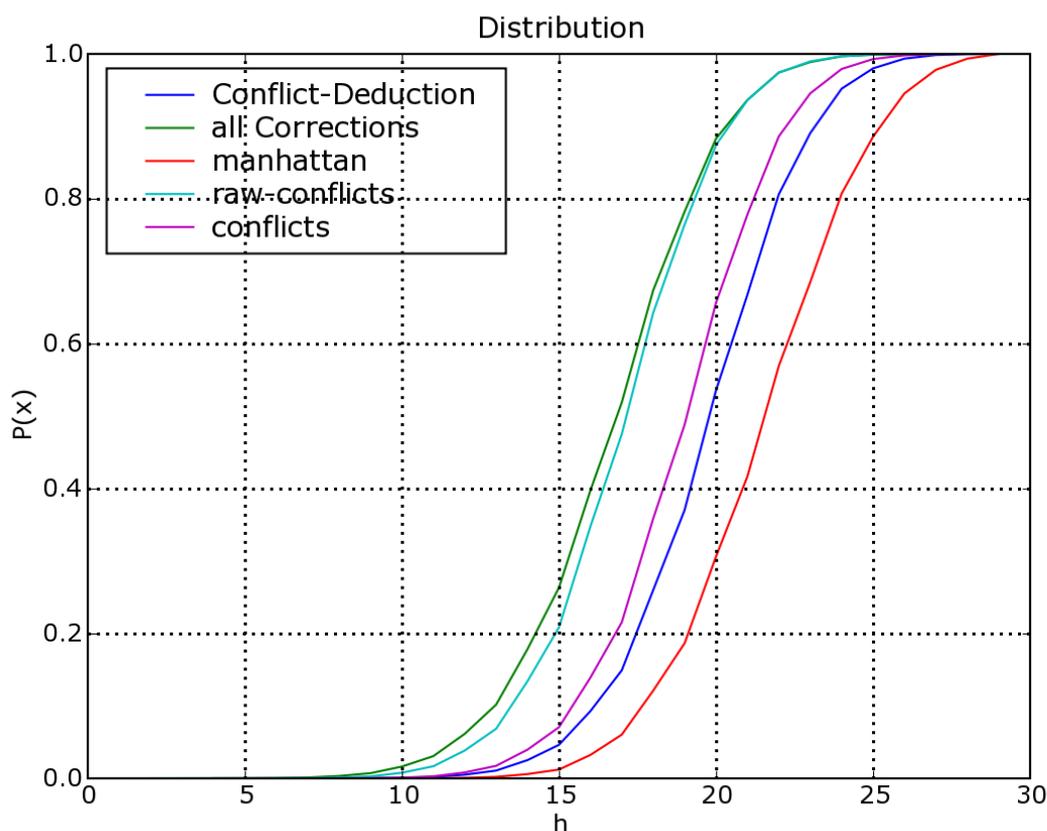


Figura 2.12.: confronto euristiche ammissibili e non

Certamente il metodo proposto non è in grado di cogliere i rapporti di informatività fra le euristiche sotto esame, mostrando invece una spiccata predilezione per le euristiche meno informate. Questo risultato può interpretarsi non come un'infondatezza della teoria, piuttosto con il fatto che il concetto di supporto superiore si rivela in questo caso non adatto a cogliere la rilevanza dello scostamento. Ossia, lo scostamento del valore più elevato non si ripercuote nel N-puzzle in pari misura su tutto lo spettro di valori euristici.

2.3.4 Metodo di Korf-Reid per il computo dei nodi espansi dall'algoritmo IDA*

In [KorfReidEdelkamp2001] viene proposto un metodo che permette di calcolare il numero di nodi espansi (e, con una piccola modifica, i nodi generati), dall'algoritmo IDA*, proposto dallo stesso Korf alcuni anni prima. Per la prima volta si ha a che fare con un modello che permette a priori, ovvero senza risolvere alcuna istanza del problema, di stimare il numero di nodi espansi da un algoritmo di ricerca, con una precisione notevole, addirittura del 100% quando tutti i parametri utili siano esatti.

Il metodo, concettualmente molto semplice, si basa sull'idea che il numero di nodi espansi da un algoritmo come IDA* o A* si possa calcolare conoscendo quelli espansi da un algoritmo di forza bruta, ed escludendo dal computo i nodi che non rispettino la condizione di soglia. La distribuzione di probabilità dei valori euristici è lo strumento che si utilizza per

sapere che percentuale di nodi deve essere esclusa dal computo.

Conoscendo il branching factor di un problema è possibile calcolare il numero di nodi espansi da un algoritmo brute-force con la formula

$$N(b, d) = \sum_{i=1}^{d+1} b^i. \quad (2.9)$$

In figura 2.13 si può mostra come sarebbero espansi i nodi in un ipotetico albero in cui ogni nodo espanda tre figli, uno a valore euristico inferiore, uno uguale, uno superiore di un'unità.

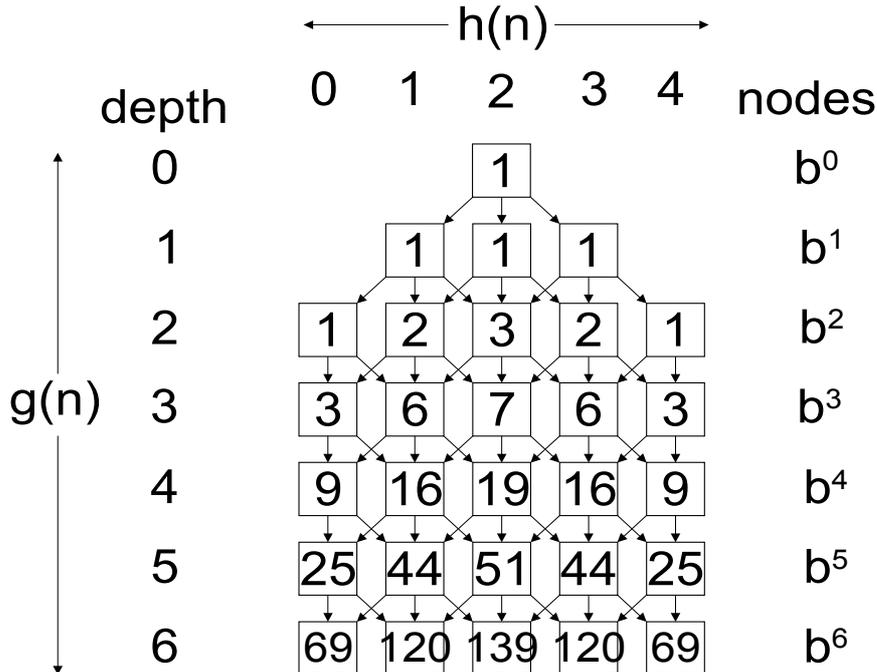


Figura 2.13.: distribuzione nodi espansi da un algoritmo brute-force

Un nodo sarà espanso dall'algoritmo IDA* solo a condizione che $g(i) + h(n) < d$, dove d è il livello di soglia dell'iterazione attuale.

Se si considera un problema in cui il costo del cammino è pari al numero di passi ($g(i) = i$) allora saranno espansi solo i nodi per cui valga $h(n) < d - i$. La probabilità che un nodo a profondità i sia espanso è dunque $P(d - i)$, dove P è la funzione di distribuzione di probabilità cumulativa, descritta nella sezione 2.3.1.

Il numero di nodi espansi dall'ultima iterazione dell'algoritmo IDA* sarà dato, mediamente nel caso peggiore¹ da:

$$N(b, d, P) = \sum_{i=1}^d N_i \cdot P(d - i + 1) \quad (2.10)$$

Questa formula permette di calcolare con grande precisione il numero di nodi espansi da

¹ Il caso peggiore è che il nodo soluzione sia trovato per ultimo tra i nodi allo stesso livello di soglia.

un'iterazione dell'algorithmo alla data profondità d . In figura 2.14 è possibile vedere più chiaramente il significato diretto della formula.

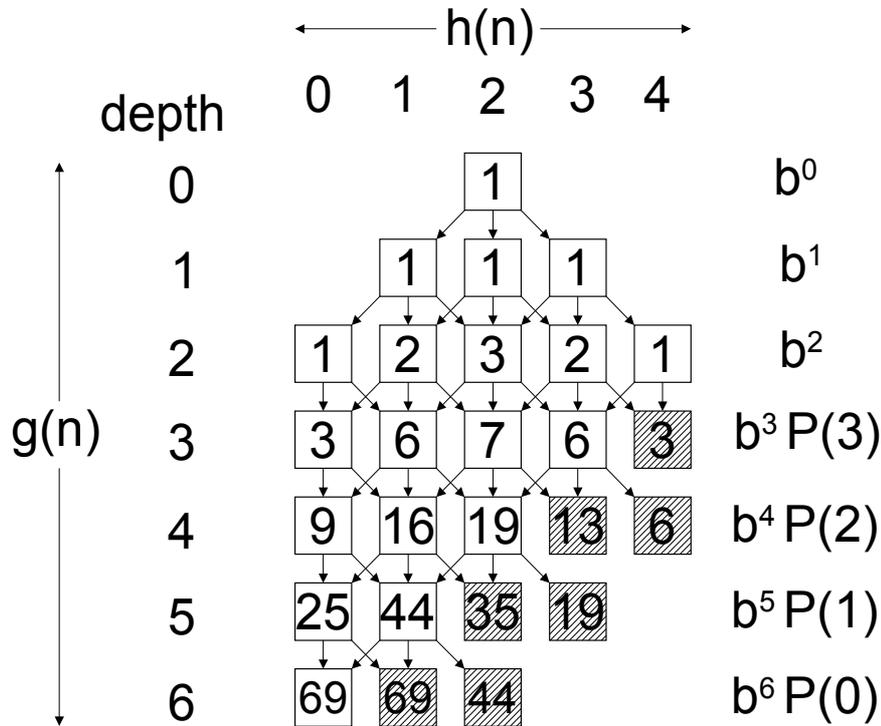


Figura 2.14.: distribuzione nodi espansi da un algoritmo IDA*

2.3.4.1 La equilibrium distribution

In realtà la probabilità che deve essere utilizzata non è quella di distribuzione nello spazio degli stati semplice, ma la distribuzione nello spazio di ricerca in condizione di equilibrio (*equilibrium distribution*); infatti i nodi non è detto che abbiano la stessa probabilità di comparire nell'albero di ricerca, e questo fatto deve essere tenuto in conto nel momento in cui si desideri un risultato altamente accurato.

Il problema principale è che il calcolo della equilibrium distribution non è altrettanto semplice che quello della probability (*overall*) distribution: in [KorfReidEdelkamp2001] è indicato brevemente come possa essere calcolata esattamente la equilibrium distribution: il fatto che la rende diversa dalla overall distribution è che la probabilità di incontrare un nodo all'interno dell'albero di ricerca non è uniforme, ma dipende dal numero di transizioni che possono portare a quel nodo. In particolare per l'N-Puzzle la proporzione tra i nodi all'angolo, al centro e al lato all'interno dell'albero di ricerca è diversa dalla loro proporzione nella totalità dello spazio.

Per ricavare la distribuzione all'equilibrio per l'N-puzzle è necessario dunque calcolare la frequenza delle diverse tipologie di nodi individuati. Nel caso del 5-puzzle si possono individuare 5 categorie: corner-corner (f_{cc}), corner-side (f_{cs}), side-corner (f_{sc}), side-side (f_{ss}). Il primo termine a pedice indica la posizione corrente della posizione vuota (blank), il secondo termine la posizione precedente¹. Le transizioni possono essere rappresentate con una

1 Non potendo un nodo generare come figlio il proprio genitore le transizioni dipendono anche dalla posizione

macchina a stati (Figura 2.15):

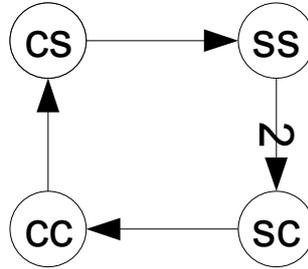


Figura 2.15.: Transizioni 5-puzzle

Il valore di equilibrio può essere ricavato eseguendo transizioni fino a una convergenza asintotica e contando le relative frequenze. La equilibrium distribution dipende dalle frequenze di side e corner:

$$f_s = f_{sc} + f_{ss}, f_c = f_{cc} + f_{cs}, \quad \text{da cui la equilibrium distribution } D(x) = f_s \cdot P(x|side) + f_c \cdot P(x|corner).$$

Come si può vedere il processo di calcolo della equilibrium distribution, oltre ad essere piuttosto macchinoso è fortemente dipendente dallo specifico problema.

Fortunatamente equilibrium distribution spesso è identica alla overall distribution, o è molto ben approssimata da questa¹; usare la distribuzione totale non è dunque causa di errori relativamente importanti nel calcolo dei nodi generati.

2.3.4.2 Calcolo del branching factor per l'N-Puzzle

Deve essere invece molto accurato il valore di branching factor: infatti un errore dello stesso si ripercuote sul risultato in maniera esponenziale. Anche in questo caso si tratta di conoscerne il valore in condizioni di equilibrio e la metodologia di calcolo si basa sugli stessi ragionamenti visti per la equilibrium distribution. Una volta calcolate le frequenze delle diverse tipologie di nodi si media il branching factor degli stessi con la frequenza; per il 5-puzzle ad esempio si ha:

$$b(d) = b_{cc} f_{cc}(d) + b_{sc} f_{sc}(d) + b_{cs} f_{cs}(d) + b_{ss} f_{ss}(d) = f_{cc}(d) + f_{sc}(d) + 2 f_{cs}(d) + 2 f_{ss}(d).$$

La dipendenza dalla profondità è dovuta al fatto che in generale il branching factor medio potrebbe non essere costante con la profondità della ricerca: ad esempio nel N-Puzzle assume un valore diverso a profondità pari e dispari dell'albero.

Il metodo proposto permette di stimare b con una precisione di diverse cifre decimali, ma richiede una consistente base teorica, e non è facilmente generalizzabile a diversi tipi di problemi: ad esempio in un labirinto è impossibile individuare dei cluster con branching factor costante come per l'N-puzzle. In questi casi è proponibile stimare il branching factor

precedente.

¹ Vedere sperimentazioni a fine paragrafo riguardo all'utilizzo della probability distribution invece della equilibrium distribution.

eseguendo un campionamento (ad esempio tree-sample¹) e contando il numero medio di nodi espansi. Nelle sperimentazioni si vedrà anche come l'uso di un fattore di ramificazione non esatto influenzi il risultato finale.

2.3.4.3 Osservazioni

Infine bisogna osservare la principale lacuna di questo metodo, che a prima vista sembrerebbe fornire la possibilità di calcolare a priori il numero di nodi espansi (e con una piccola modifica sull'indice, anche generati) dall'algoritmo IDA*: in realtà non è noto quale sia il valore di d (la profondità della soluzione) che serve per fare il calcolo, prima di far girare l'algoritmo.

Il metodo di Korf-Reid può essere dunque un ottimo strumento per confrontare le euristiche, ma non fornisce, senza conoscere la profondità media della soluzione, il vero valore del numero medio di nodi espansi da una reale ricerca. Nel capitolo successivo si vedranno due possibili metodi per stimare la reale distanza dalla soluzione.

2.3.4.4 Analisi sperimentali

Sono state eseguite diverse sperimentazioni utilizzando il metodo di Korf-Reid; l'obiettivo è duplice: innanzitutto si vuole vedere come sia possibile confrontare le funzioni euristiche utilizzando il suddetto metodo. Inoltre si vedranno delle analisi di precisione, il cui scopo è, più che verificare l'esattezza del metodo (già dimostrata essere del 100% quando tutti i parametri siano calcolati esattamente), studiare come l'errore sui parametri possa influenzare il risultato. Innanzitutto si vuole capire quale sia il modo migliore di campionare lo spazio degli stati; d'altra parte si vedrà come poter stimare il branching factor attraverso il campionamento, e l'errore che scaturisce da questa operazione.

In figura 2.16 si vede il risultato dell'applicazione del metodo di Korf-Reid per alcune euristiche per l'8-puzzle. Il calcolo è stato eseguito usando la overall distribution.

¹ Vedi Appendice A.

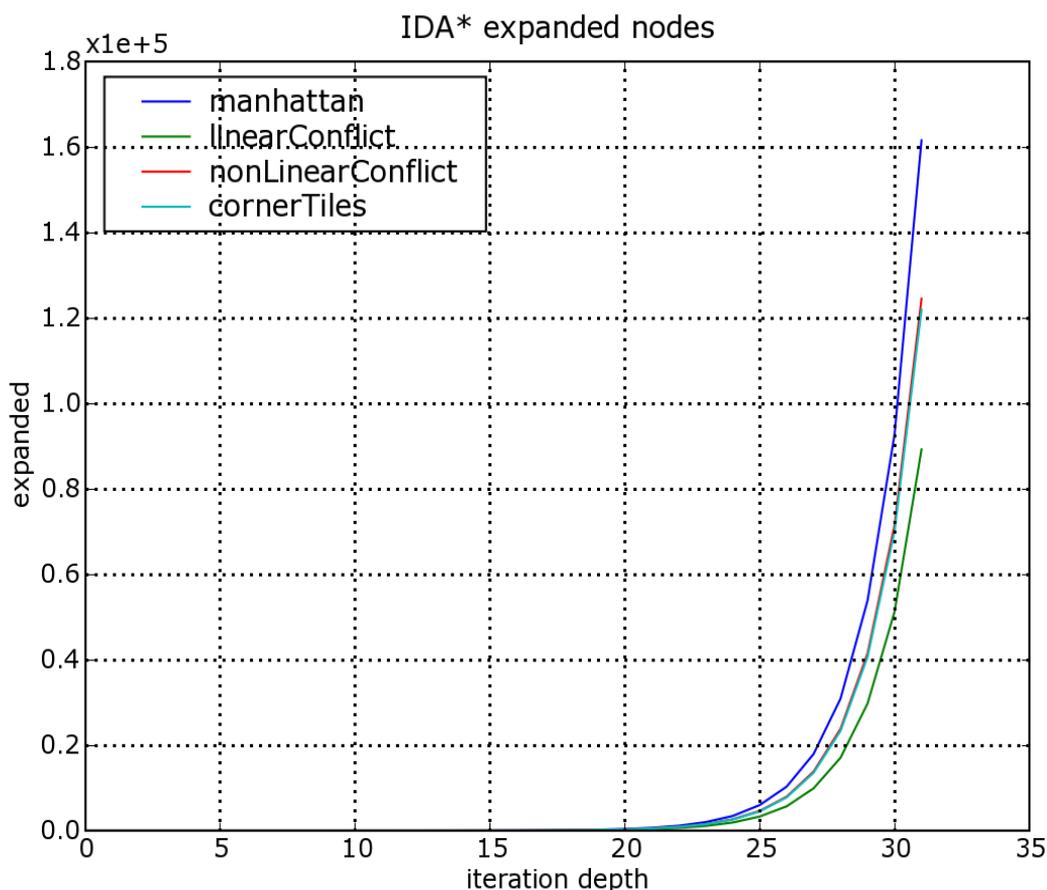


Figura 2.16.: nodi espansi 8-puzzle - confronto euristiche

I risultati si allineano a quelli visti per l'analisi di informatività: linear-conflict è l'euristica tra le manhattan correction che riesce a fornire le migliori prestazioni. La crescita del numero di nodi espansi è esponenziale con la profondità; le curve appaiono uguali ma traslate l'una rispetto all'altra; l'euristica dunque si inserisce come un termine sottrattivo alla reale distanza dalla soluzione.

Il branching factor appare immutato con la presenza dell'euristica. La curva interpolante può essere rappresentata come l'equivalente brute-force:

$$N_{\text{exp}}(d) = b^{d-d_h} \quad (2.11)$$

La costante d_h è un termine sottrattivo che dipende dall'euristica; questo risulterà più elevato quando l'euristica sia più informata. Ad esempio si ha $d_h=9.18$ per l'euristica manhattan per l'8-puzzle, $d_h=10.2$ per linear-conflict e $d_h=4,4$ per tiles-out-of-place in base all'analisi eseguita e riportata parzialmente in figura 2.16.

Nel caso in cui il branching factor non sia costante con la profondità vale la formula

$$N_{\text{exp}}(d+1) = N_{\text{exp}}(d) \cdot b(d) \quad (2.12)$$

Si può evincere che una volta nota d_k (o equivalentemente, un valore puntuale a determinata profondità) può essere indotto per via analitica dalla (2.11) o dalla (2.12).

In tabella 2.2 si ha una ulteriore conferma sperimentale di quanto appena affermato; si può vedere inoltre il confronto preciso tra i valori ottenuti con la overall distribution e i risultati invece esatti ottenibili usando la equilibrium distribution.

Depth	Experimental	Enum-overall	Error%
20	393	379	-3,56
21	657	662	0,76
22	1185	1142	-3,63
23	1977	1992	0,76
24	3561	3432	-3,62
25	5936	5980	0,74
26	10686	10301	-3,6
27	17815	17947	0,74
28	32072	30910	-3,62
29	53450	53846	0,74
30	96207	92735	-3,61
31	160167	161545	0,86

Tabella 2.2.: errore dovuto all'uso della equilibrium distribution

L'errore è relativamente contenuto, e non mostra avere una deriva all'aumentare della distanza. Si può notare che come molte altre misure per l'8-puzzle si hanno valori notevolmente diversi a profondità pari e dispari.

Ora si vuole invece analizzare il problema del campionamento. Si vuole capire quale porzione dello spazio degli stati deve essere campionata per avere dei risultati statisticamente attendibili. Non bisogna farsi ingannare in questo tipo di analisi dal fatto che spesso si possa ottenere un buon risultato campionando pochi nodi, bisogna piuttosto valutare la stabilità del risultato: un campionamento è da considerarsi buono quando, oltre ad approssimare meglio una curva reale, fornisce risultati stabili e dunque ripetibili. Bisogna anche ricordarsi che la grandezza campionata è ancora la probability distribution, e dunque una buona stima della stessa potrebbe essere fonte di maggiore errore sul risultato; si vedrà poi anche se sia possibile, utilizzando un campionamento che abbiamo denominato random-walk-pick¹, stimare direttamente la equilibrium distribution.

In figura 2.17 e 2.18 è mostrato il risultato ottenuto eseguendo N diverse analisi con campionamento dello spazio degli stati casuale, con diverse numerosità dei campioni per il 5-puzzle e per l'8-puzzle.

¹ Vedi Appendice A.

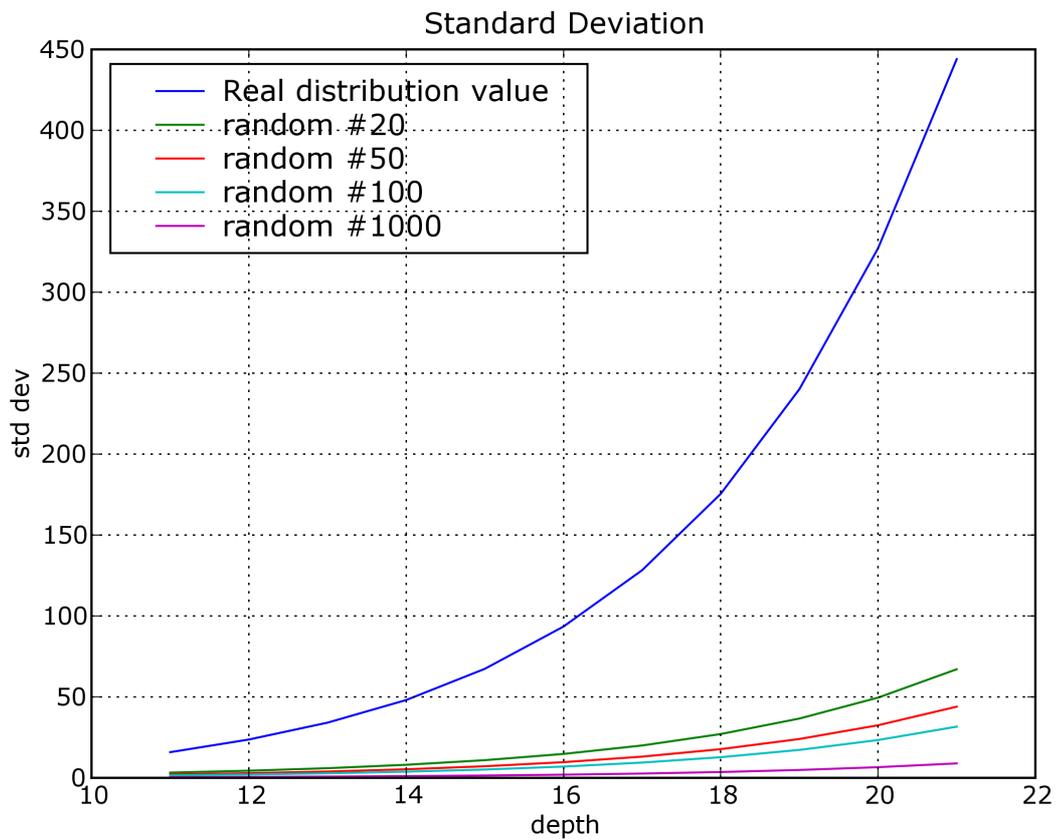


Figura 2.17.: deviazione di 1000 analisi effettuate per il 5-puzzle con diversi campionamenti

Per il 5-puzzle 50 campioni (14% dello spazio) rappresentano un sample set sufficientemente significativo in base ai parametri già descritti nella sezione 2.3.2.1.

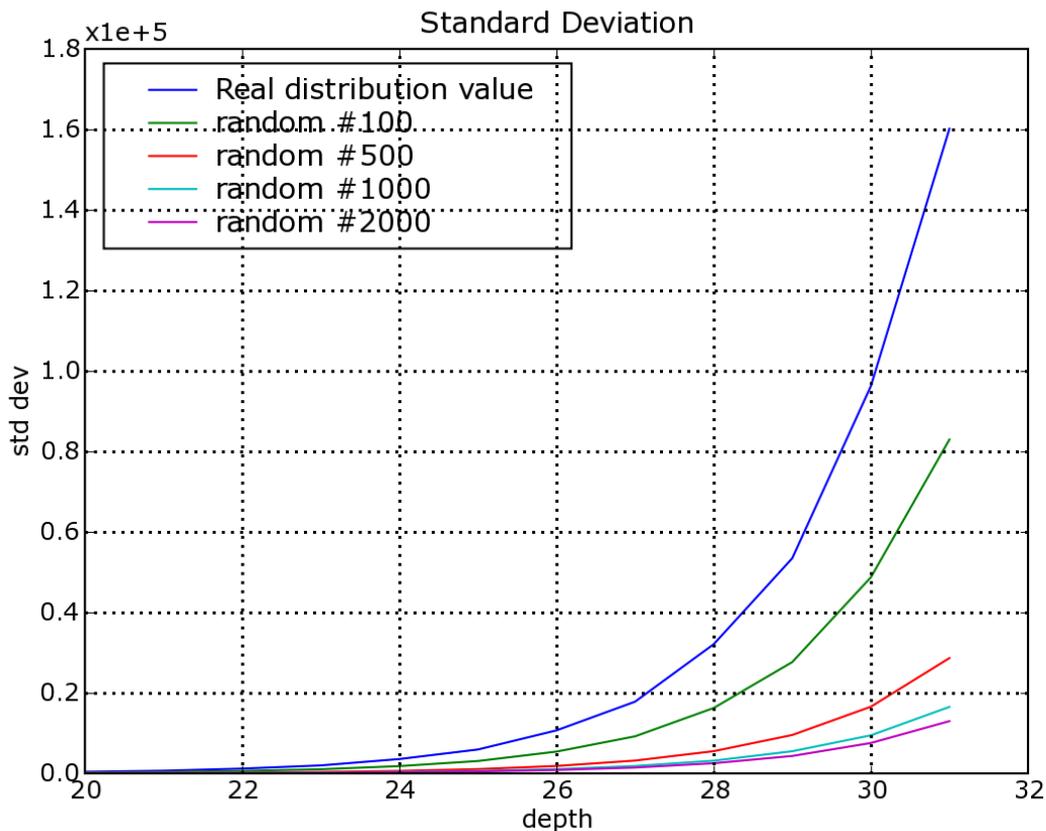


Figura 2.18.: deviazione di 100 analisi effettuate per l'8-puzzle con diversi campionamenti

Il numero di campioni è uguale a quello visto per il caso migliore riguardo all'analisi del campionamento della semplice distribuzione (2000 campioni sono sufficienti, l'1% dello spazio degli stati). La porzione di spazio da campionare si riduce notevolmente all'aumentare della dimensione del puzzle. Sarà dunque ragionevole campionare una porzione di spazio molto inferiore all'1% per l'8-puzzle. Il valore di media tende asintoticamente a zero aumentando la numerosità del sample set. Per quanto riguarda l'ultima analisi l'errore medio si mantiene al di sotto del 2,5% con 2000 campioni. Lo stesso esperimento ha stabilito che un sample set di 200000 nodi per il puzzle 4x3 (0.08% dello spazio) permette la stessa precisione. In questo caso è stato usato come paragone il valore reale, ma il valore medio ricavato dagli stessi esperimenti.

Se considera il problema 15-Puzzle, che come noto ha uno spazio degli stati di numerosità dell'ordine di 10^{13} , è improponibile eseguire un'analisi statistica come quella proposta per istanze più piccole del problema. Si può indurre, visto che la significatività del campione aumenta in proporzione con la dimensione del problema, che sarà sufficiente una porzione dello spazio degli stati molto inferiore all'1% per avere un'attendibilità e accuratezza comparabile con quella dell'esempio precedente. Korf usa un sample set di 10 miliardi di nodi (0,1% dello spazio) per eseguire gli esperimenti sul 15-puzzle, ottenendo un errore di circa l'1% rispetto al valore reale calcolato per 100.000 nodi fino a profondità 50. In base alle osservazioni sulla attendibilità statistica fatte precedentemente si potrebbe indurre che è possibile utilizzare un sample set decisamente inferiore per il 15-puzzle. Interpolando

linearmente la curva dei valori precedenti in scala logaritmica si ottiene un valore di 0.001% per il 15-puzzle (100 milioni di nodi). Quando non si sia interessati al valore esatto di nodi espansi, ma si voglia piuttosto usare questo metodo per confrontare delle funzioni euristiche un numero di campioni molto inferiore è certamente sufficiente: in questo caso infatti, nell'ipotesi che il comportamento delle funzioni euristiche sia discretamente uniforme nello spazio degli stati, usare lo stesso sample set per le diverse euristiche da confrontare può essere ugualmente attendibile nonostante un campionamento notevolmente parziale.

Si vuole in ultima istanza analizzare l'effetto di un errore del branching factor sul calcolo dei nodi espansi. In figura 2.19 si presenta il risultato ottenuto per lo stesso sample set per l'8-puzzle, usando il branching factor esatto, il branching factor medio (nessuna distinzione per profondità pari e dispari), e il branching factor calcolato come numero medio di figli dei nodi del sample set. I metodi elencati vanno dal più preciso e al più generale ma presumibilmente meno esatto.

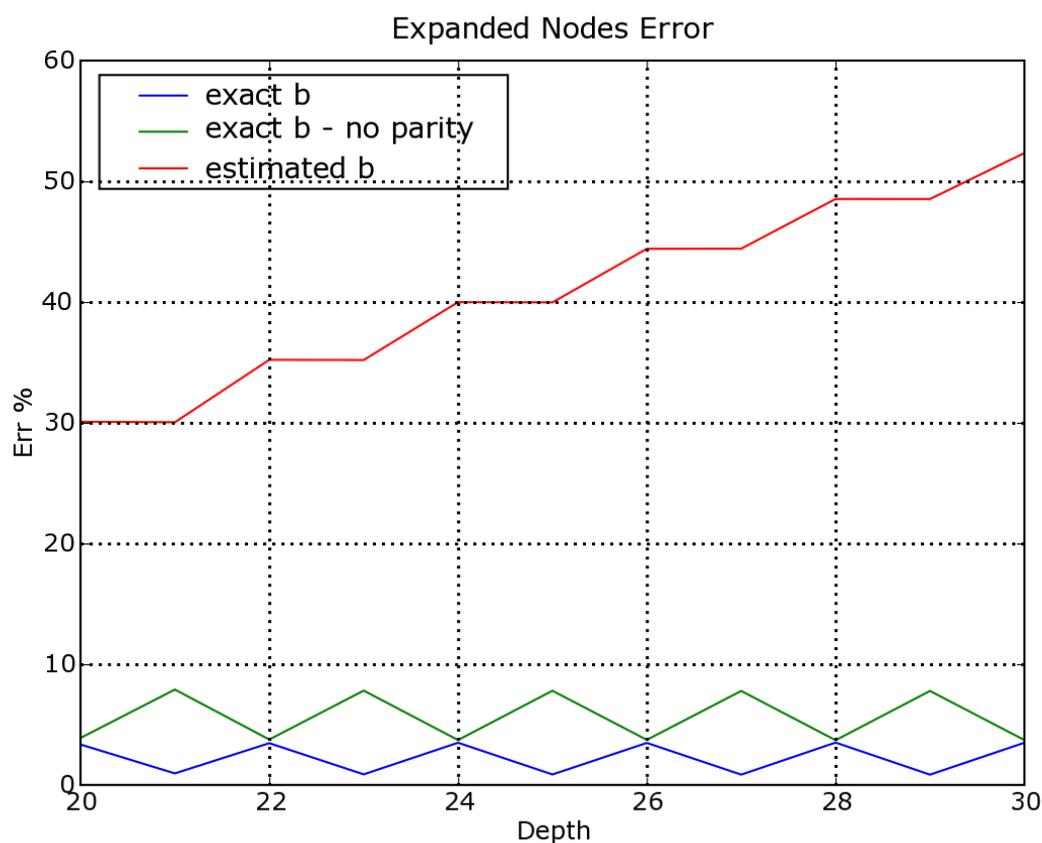


Figura 2.19.: effetto del branching factor sull'errore

Come si può vedere risulta in errori decisamente elevati usare un branching factor stimato da campionamento, nonostante si siano usati tutti i nodi dello spazio degli stati. Questa operazione fornisce in branching factor di $1.\bar{6}$, invece di 1.732. Questo valore è pari a quello che si troverebbe mediando il branching factor di ogni casella, senza tenere in conto le probabilità all'equilibrio.

2.4 Metodi matematici per il calcolo dei nodi generati (cenni)

In [Pearl84] è presentata, all'inizio del quinto capitolo, dedicato all'analisi di prestazioni, una serie di trattazioni teoriche per determinare il numero di nodi espansi generati con diverse strategie di ricerca, che riassumono alcuni articoli di quegli anni di diversi ricercatori. I problemi considerati sono il ritrovamento di un percorso in una griglia usando la distanza in linea d'aria, e il percorso in una mappa stradale con la stessa euristica. È eseguita un'esaustiva analisi parametrica dei problemi in questione, fino a individuare un'approssimazione per i nodi generati, l'euristica e la dimensionalità del problema come parametri. La dimostrazione è portata avanti rigorosamente per mezzo di assunzioni geometrico-matematiche sullo spazio bidimensionale che costituisce il problema. Non si approfondirà la questione per via del carattere decisamente poco generalizzabile delle seppur interessanti dimostrazioni.

2.5 Calcolo di tempo – memoria

Si è già accennato in introduzione al capitolo alla questione del calcolo delle prestazioni in termini di tempo e memoria. Si presentano in questa sezione due proposte incontrate in letteratura, che si basano entrambe sulla suddivisione del processo di ricerca in task atomici a tempo costante.

2.5.1 Framework per l'analisi di algoritmi e funzioni euristiche

In [AyyoubMasoud2000] è proposto un framework generale per la valutazione degli algoritmi e delle funzioni euristiche sotto diversi punti di vista, come tempo, memoria, qualità della soluzione. Il loro metodo si basa sulla misura di una serie di parametri ricavati misurando alcuni parametri da delle istanze di soluzione su un sample set di nodi dello spazio degli stati.

Viene calcolata la frequenza (ϕ) delle principali attività svolte dagli algoritmi di ricerca:

- espansione di nodi (ξ)
- calcolo di stime euristiche (η)
- check di nodi duplicati (ν)
- operazioni di test e set (ω)

Ogni attività x ha associato un tempo $\tau_M(x)$ che dipende dalla macchina e dall'implementazione (M).

Avendo a disposizione le suddette misure di frequenza è possibile stimare il tempo medio necessario a giungere a una soluzione tramite la formula:

$$T(A, M) = \tau_M(\xi)\phi_A(\xi) + \tau_M(\eta)\phi_A(\eta) + \tau_M(\nu)\phi_A(\nu) + \tau_M(\omega)\phi_A(\omega)$$

Gli autori propongono altre tre feature sintetiche calcolabili tramite il loro metodo:

- Spazio Richiesto medio:

$$S(A, M) = \gamma_A \times \rho_M,$$

dove γ_A è il numero di nodi generati dall'algoritmo A^* , e ρ_M il numero di word richieste per rappresentare un nodo del grafo.

- Qualità della soluzione:

$$Q(A) = \frac{\Pi}{\pi_A},$$

dove Π è la lunghezza del percorso della soluzione ottima e π_A è la lunghezza della soluzione ottenuta dall'algoritmo A.

- Efficacia della ricerca

$$E(A) = \beta \frac{\pi_A}{\gamma_A},$$

dove β è la ramificazione media del grafo.

L'efficacia della ricerca fornisce informazioni sulla misura in cui la ricerca è diretta nella direzione dell'obiettivo. Un valore di E prossimo all'unità è indice di una ricerca molto efficace.

Tutti i parametri incontrati dipendono da tre fattori: il problema, la macchina, e la funzione euristica usata.

In particolare si ha per tutti una dipendenza dal problema specifico, ρ_M e τ_M dipendono macchina, mentre la funzione euristica affetta il parametro $\phi_A(\xi)$, e di conseguenza $\phi_A(\eta)$, $\phi_A(\nu)$, $\phi_A(\omega)$, π_A e γ_A .

Sempre in [AyyoubMasoud2000] si incontrano una serie di sperimentazioni atte a confrontare le prestazioni di quattro diversi algoritmi: A*, IDA*, BHPA e BHFFA. Gli ultimi due algoritmi, non descritti nel dettaglio in questo documento, sono due (obsoleti) metodi di ricerca bidirezionale. Il dominio scelto è l'8-puzzle, e viene fatto un calcolo empirico dei parametri eseguendo la ricerca su un sample set del 25% dello spazio degli stati.

Questo tipo di campionamento ha permesso di fare una stima del tempo di esecuzione medio notevolmente accurata. L'algoritmo che è riuscito ad ottenere le prestazioni migliori è IDA*.

Questa accuratezza dipende da quanto a sua volta è accurata la conoscenza delle misure descritte; una prospettiva interessante sarebbe sicuramente ricavare i diversi parametri con una buona precisione senza dover eseguire un numero così elevato di istanze degli algoritmi. In questo modo il metodo potrebbe essere usato per valutare la scelta dell'algoritmo e/o dell'euristica più adatti a risolvere un determinato problema.

2.5.2 Metodo per il calcolo del tempo per l'algoritmo BIDA*

In [LinaresJunghanns2002] viene proposto un modello abbastanza simile a quello visto nel precedente paragrafo; il modello è usato dunque analizzare le prestazioni dell'algoritmo BIDA* [Manzini1995].

Si ricorda che l'incremento di prestazioni di un algoritmo di perimetro come BIDA* è dovuto all'uso di un'euristica più informata, ottenuta dalla somma del valore euristico della distanza dal nodo da valutare al nodo del perimetro e il valore vero di distanza dal nodo di perimetro al goal (che viene memorizzato dopo la creazione del perimetro). D'altra parte ad ogni passo deve essere valutata la funzione euristica dal nodo da espandere a tutti i nodi del perimetro. La profondità del perimetro è dunque un fattore fondamentale dal punto di vista

delle prestazioni: un perimetro molto profondo ridurrà maggiormente l'albero di ricerca grazie a un'euristica maggiormente accurata, a fronte d'altra parte di una crescita esponenziale del numero di nodi del perimetro, e parallelamente del numero di valutazioni da fare per ogni singolo passo.

Gli autori suddividono le attività principali degli algoritmi in tre task principali, allo scopo di rendere l'analisi il più possibile indipendente dalla macchina. Rispetto a [AyyoubMasoud2000] vengono ignorate le operazioni di test and set, incluse nell'espansione del nodo. Viene poi proposto di usare una versione estesa del metodo di Korf [KorfReidEdelkamp2001] per avere una stima dei nodi espansi e generati.

Il tempo di esecuzione medio dell'algoritmo IDA* può essere calcolato da

$$T(d) = N(b, d, P)t_h + N(b, d-1, P)t_e,$$

dove t_e è il tempo di espansione di un nodo e t_h il tempo di calcolo della funzione euristica.

Per quanto riguarda il costo computazionale degli algoritmi di perimetro si può adottare la seguente decomposizione in quattro parti indipendenti:

1. Il tempo necessario a creare e memorizzare il perimetro $T_g(B, p_d)$
2. Il tempo di espansione dei nodi t_e
3. Il tempo speso a calcolare $h_{p_d}(n)$, t_h
4. Il tempo per verificare se un nuovo nodo generato appartiene al perimetro t_c

Il tempo totale sarà la somma pesata di questi tempi:

$$T_p(d, p_d) = T_g(B, p_d) + N_e(b, d - p_d - 1, P_{p_d})t_e + N_h(b, d - p_d, P_{p_d})t_h + N_c(b, d - p_d, P_{p_d})t_c \quad (2.13)$$

Le frequenze N_x sono ispirate al numero previsto dal metodo di Korf, ma sono in generale differenti dalla (2.10), e la formulazione è differente per ogni caso x . Gli autori propongono un'analisi matematica che tiene in conto tutti i principali parametri che contribuiscono a modificare le prestazioni di un algoritmo bidirezionali, come il numero effettivo di nodi utili del perimetro, sui quali soltanto deve essere realizzato il confronto passo per passo, che decresce con la profondità. Si tralasciano i dettagli perché eccessivamente specifici e oltre gli scopi di questa trattazione. Sono in aggiunta presentate alcune sperimentazioni che confermano la bontà dell'analisi proposta, e ancora una volta come l'applicazione di un algoritmo di perimetro sia una strategia vincente nella pratica.

2.6 Metodi di valutazione allo stato dell'arte per gli algoritmi di ricerca locale stocastica

Nel campo della classe degli algoritmi denominati Stochastic Local Search (SLS), si hanno alcuni metodi di studio delle funzioni di valutazione che possono essere rivisti ed estesi per essere utilizzati nei classici algoritmi di ricerca, anche per via della somiglianza concettuale della funzione di valutazione con le funzioni euristiche. La trattazione sarà in questo paragrafo molto sintetica essendo un argomento ripreso ampiamente nel successivo capitolo riguardo agli algoritmi di ricerca globale. Un riferimento che descrive tutti questi metodi è il libro [HoosStutzle2005].

2.6.1 Search Landscape analysis

Molte delle idee sulla valutazione delle euristiche per gli algoritmi SLS si basano sul concetto di landscape. Per landscape si intende l'astrazione del valore euristico nello spazio di ricerca come se si trattasse di un'altezza in ipotetico uno spazio tridimensionale. Su questa rappresentazione si basano lo studio delle posizioni e lo studio di rugosità (ruggedness).

2.6.1.1 Tipi di posizioni e loro distribuzione

Rappresentando il landscape in tre ipotetiche dimensioni (l'altezza è data dal valore euristico) ci si rende conto come questo sia assimilabile a una superficie con tanto di salite, discese, zone piane e minimi locali. Si può classificare un punto del piano (che altri non è, nel campo SLS, che una possibile soluzione) a seconda della sua posizione nel piano rispetto ai suoi vicini. Di particolare interesse sono le posizioni di minimo locale e plateau.

2.6.1.2 Numero, densità e distribuzione dei minimi locali e plateau

Si può pensare a un algoritmo di ricerca locale come a un processo di ricerca di minimo nella superficie del landscape. Il processo di ricerca può essere avvicinato al movimento di una sfera che partendo da un punto dato o casuale e guidata dalla gravità navighi all'interno del landscape. Un processo di questo tipo in assenza di inerzia si può accostare molto da vicino con l'algoritmo hill-climbing (iterative-improvement nella nomenclatura SLS). In questo contesto il ritrovamento della soluzione coinciderebbe con il raggiungimento di un minimo globale; risulta evidente che la presenza di minimi locali non soluzione bloccherebbe l'algoritmo hill-climbing, e sarebbe comunque d'impiccio ad algoritmi più sofisticati. Analogamente una situazione di plateau (piano orizzontale del landscape) risulterebbe in un'indecisione locale, una zona in cui l'algoritmo non sia guidato nel suo percorso, diventando localmente casuale.

Gli esperimenti confermano una forte correlazione tra il numero di minimi locali e plateau e la difficoltà di risoluzione di un problema. Più esattamente ciò che interessa non è tanto il numero di minimi ma la densità di questi nello spazio di ricerca, in relazione anche al numero di soluzioni. Infatti le soluzioni reali (che possono essere in numero variabile) sono anch'esse dei minimi locali, essendo un minimo globale anche minimo locale (spesso inoltre ci si accontenta di trovare come dei minimi locali sufficientemente "buoni" senza cercare il minimo globale ad ogni costo); la presenza di un gran numero di soluzioni è sintomo non di difficoltà del problema al contrario dei minimi non soluzione, anzi semplifica il ritrovamento di una soluzione. La densità non può essere tipicamente misurata direttamente, ma può essere spesso stimato o in casi particolari determinato analiticamente.

Un'analisi ancora più dettagliata tiene invece in conto la distribuzione dei minimi locali; la densità dei minimi non è costante nelle diverse zone dello spazio. La distribuzione di questi influenza la probabilità che vengano incontrati nel processo di ricerca, e dunque la difficoltà del problema. L'analisi della distribuzione può essere realizzata tramite tecniche di analisi di cluster.

2.6.2 Fitness-distance analysis

Con fitness si indica il valore della funzione di valutazione, che si presume possa essere una stima più o meno accurata della distanza dalla soluzione. Un alto valore di correlazione

tra funzione di valutazione e distanza reale dalla soluzione più vicina è indice di bontà della funzione utilizzata.

Questo tipo di analisi richiede la conoscenza della soluzione reale ed è dunque utilizzabile solo per domini relativamente ristretti. Nonostante tutto può essere un buono strumento di analisi del problema e i risultati ottenuti per piccole istanze possono essere estesi a più grandi istanze dello stesso problema.

2.6.3 Landscape Ruggedness

Se consideriamo un semplice algoritmo di iterative improvement (hill-climbing) è intuitivo realizzare che una superficie con meno falsi avvallamenti possa guidare più facilmente l'algoritmo stesso verso la soluzione. Questo è vero in generale anche per algoritmi più complessi. Per questo può essere un buon indicatore misurare la rugosità della superficie del landscape al fine di valutare la maggiore o minore difficoltà di un problema con diverse funzioni di valutazione. Ci sono stati numerosi tentativi di formalizzare il concetto di rugosità di una superficie. Uno possibile è misurare la correlazione dei livelli posti a una determinata distanza i fra di loro in un dato landscape. Si utilizzano indici tipici dell'analisi di superfici e di segnali come il coefficiente di correlazione e lunghezza di correlazione.

2.7 Studi sulle euristiche nel planning

Hoffmann ([Hoffmann2001], [Hoffmann2002]) propone una caratterizzazione dello spazio degli stati e delle euristiche per un ambito di ricerca locale pensata per i problemi della pianificazione (planning); lo scopo del planning è trovare automaticamente una sequenza di azioni che permetta, partendo da uno stato iniziale, di arrivare a uno stato obiettivo, tipicamente utilizzando tecniche di intelligenza artificiale. Senza entrare eccessivamente nel merito, a un solutore di problemi di planning si chiede tipicamente di riuscire a trovare quante più possibili soluzioni sub-ottime a una serie di problemi reali o pseudo-reali, in un tempo limitato. Per come è definito un problema di pianificazione la ricerca euristica è sicuramente uno dei metodi naturali e più efficaci di soluzione. La ricerca nel planning deve solitamente far fronte a problemi di tipo diverso rispetto alla ricerca sistematica, per via della maggiore variabilità che si cerca di fronteggiare, e per l'obiettivo. Un discorso a parte meritano le euristiche utilizzate per questo tipo di problemi, solitamente non altrettanto efficaci rispetto a quelle viste per i problemi di ricerca sistematica, e quasi mai ammissibili.

Negli studi di Hoffmann è realizzata una esaustiva classificazione dei problemi di planning, sulla base delle caratteristiche dello spazio degli stati e delle euristiche. Per quanto riguarda le euristiche i concetti sono da vicino imparentati con quelli visti per la ricerca locale stocastica: si fa riferimento a concetti quali plateau e minimi locali, con particolare per i modi in cui sia possibile uscire da questi; in particolare la distanza di uscita da un punto di plateau o minimo risulta essere una metrica fortemente caratterizzante della difficoltà di un problema.

3 METODI INNOVATIVI PER LA VALUTAZIONE DI FUNZIONI EURISTICHE

Questo capitolo è dedicato alla presentazione dettagliata di tutti quei metodi di valutazione e analisi di funzioni euristiche non incontrati in letteratura. Si comincia con uno studio che si basa sul riadattamento di alcuni metodi già noti nel campo degli algoritmi della ricerca locale stocastica, riadattati per il mondo degli algoritmi di ricerca globale.

Il concetto basilare su cui si poggiano in gran parte questi metodi è il concetto di *landscape*: nella prima sezione è definito questo concetto, e lo si utilizza come pretesto per un'analisi locale dei nodi dello spazio di ricerca. Risultano di particolare interesse i punti di cosiddetto minimo locale e di *plateau* locale, per i quali vengono suggeriti anche degli studi sulla loro distribuzione. Gli studi di distribuzione hanno a loro volta suggerito un metodo innovativo per stimare la distanza reale della soluzione dato un valore euristico, descritto successivamente nella stessa sezione, nonostante non derivi direttamente da alcun metodo degli algoritmi SLS. Altra analisi sempre derivata dalle tecniche per la ricerca locale stocastica è l'analisi *fitness-distance*, che consiste nel confrontare le distanze reali con quelle predette euristicamente. In ultimo si presenta la tecnica di analisi di rugosità.

Un'altra famiglia di metodi innovativi che viene presentata è quella delle tecniche basate su matrice di transizione. Questi si basano su un tentativo di simulazione semplificata

dell'algoritmo, che tenga in conto le transizioni essenziali che avvengono durante le ricerca. Due sono i metodi sviluppati in questa direzione, che si distinguono, più che per il funzionamento, per lo scopo che intendono raggiungere, sebbene in entrambi i casi l'ultimo fine possa intendersi la valutazione di funzioni euristiche. Il primo mira a stimare l'errore medio causato nell'algoritmo hill-climbing dagli errori della funzione euristica, usando lo strumento delle catene di Markov per modellare il processo. Il secondo invece si propone come metodo alternativo al metodo di Korf per la stima dei nodi espansi/generati da IDA*/A*, aggiungendo anche la possibilità di suddividere il calcolo per cluster e di stimare insieme la profondità della soluzione.

3.1 Estensione dei metodi usati per euristiche SLS alla ricerca euristica

Si vedrà ora in dettaglio come possano essere estesi alcuni metodi esistenti per l'analisi del comportamento delle funzioni di valutazione degli algoritmi stochastic local search alle funzioni euristiche dei classici algoritmi di ricerca.

Il passaggio è stato fatto tenendo conto che molti dei concetti usati nella letteratura degli algoritmi SLS hanno una controparte parallela nel mondo degli algoritmi di ricerca. Questo non significa che algoritmi SLS e classici siano in realtà uguali, piuttosto abbiamo delle somiglianze in alcuni concetti che fanno ragionevolmente supporre che si possa adattare parte del lavoro fatto su uno dei due metodi per utilizzarlo proficuamente anche nell'altro.

Quello che negli algoritmi SLS è indicato come *candidate solution* ha come naturale corrispettivo il nodo degli algoritmi di ricerca. Ugualmente la relazione di vicinanza (*neighbourhood*) può essere associata all'insieme dei nodi raggiungibili in un solo passo applicando una trasformazione a un nodo. Infine, fatto più importante, il concetto funzione di valutazione può essere avvicinato al concetto di funzione euristica. Si vuole sottolineare che questo non significa che funzione di valutazione e funzione euristica siano due nomi diversi per indicare il medesimo concetto; è infatti diverso l'uso che i diversi metodi di ricerca fanno di esse. Negli algoritmi SLS la funzione di valutazione è infatti usata, oltre che per guidare l'algoritmo più rapidamente verso la soluzione, anche spesso per identificare la soluzione stessa, fatto che non accade in un algoritmo di ricerca basato su nodi, che utilizza invece un goal test indipendente in principio dall'euristica.

Si cercherà, nell'analisi che seguirà, di ricordare le differenze esistenti tra i diversi metodi di ricerca, e di evidenziare gli aspetti che possono essere utili nel nostro dominio a scapito di quelli invece meno interessanti.

3.1.1 Search Landscape analysis

Il landscape può essere definito con una tripla formata da spazio degli stati, relazione di vicinanza (determinata dagli operatori), e valore euristico. Questa definizione teorica vuole più semplicemente cercare di rappresentare in qualche modo la variazione dei valori euristici all'interno dello spazio di ricerca.

Per avere una visualizzazione concreta del concetto possiamo pensare al landscape come a una superficie discreta in tre dimensioni: le prime due dimensioni sono definite dalla vicinanza dei nodi dello spazio di ricerca (si avrà dunque in generale la rappresentazione di un grafo in due dimensioni), la terza dimensione invece è costituita dal valore euristico dei nodi.

Lo studio delle caratteristiche del landscape può così fornire informazioni utili sulle caratteristiche del problema.

Lo studio che si propone è uno studio strettamente locale, basato su una classificazione dei nodi in base alla posizione relativa rispetto ai propri vicini. Un nodo può essere classificato in base al suo valore euristico rispetto a quello dei suoi figli in una delle seguenti classi:

- SLOPE: Un nodo di passaggio; il valore euristico dei figli è in alcuni casi minore e in altri maggiore rispetto a quello del nodo slope. La presenza di uno slope può essere associata a un buon comportamento della funzione euristica in quel punto. Infatti ci si aspetta che in presenza di una sola soluzione ci sia un figlio con valore euristico inferiore al padre e a tutti gli altri, e presumibilmente il resto con un valore superiore al padre. Una funzione euristica perfettamente informata avrà solamente nodi di tipo SLOPE¹, se si esclude la soluzione e i nodi a valore euristico massimo assoluto.

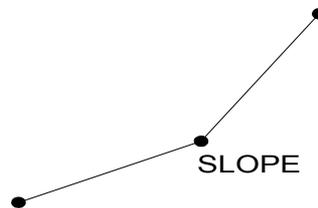


Figura 3.1: Nodo di tipo SLOPE

-
- SLMIN: Tutti i nodi figli hanno valore euristico superiore al padre. Siamo in presenza di un minimo relativo. Nel caso in cui il nodo non sia il goal abbiamo certamente un errore della funzione euristica. Un algoritmo di hill-climbing sarà attratto e non riuscirà a uscire da questo punto, mentre una ricerca di tipo best-first sarà costretto ad analizzare diversi percorsi per individuare la direzione da prendere, aumentando notevolmente il numero di nodi visitati. In ogni caso saranno necessari almeno 2 passi per ritornare a un nodo nel cammino verso la soluzione, uno per “uscire” dal minimo, uno per arrivare a un nodo con valore euristico almeno uguale a quello del nodo SLMIN. Il valore euristico del nodo SLMIN è stato sottostimato del costo dei due passi.

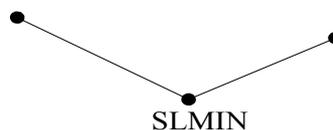


Figura 3.2: Nodo di tipo SLMIN

- LMIN: Tutti i figli hanno valore euristico maggiore o uguale al padre, almeno uno strettamente maggiore e almeno uno uguale. È certamente indice di un errore, ma meno grave rispetto al caso SLMIN. Più che di errore si può parlare di indecisione

¹ Si considera sempre il caso in cui ci sia solo una soluzione.

della funzione euristica. La sottostima della funzione euristica è di almeno il costo di un passo.

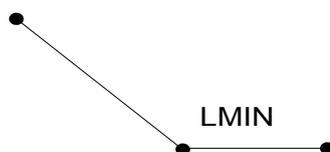


Figura 3.3: Nodo di tipo LMIN

- SLMAX: Tutti i figli hanno valore euristico inferiore al padre. Non è propriamente un errore dell'euristica, piuttosto un'indecisione. Ci si aspetta di avere un numero di massimi totale che cresce con il numero di minimi; infatti se la funzione euristica commette un errore sottostimando il valore euristico di alcuni nodi, nel cammino verso la soluzione avremo un riassetamento verso l'alto dei valori euristici (una "salita"), a cui dovrà seguire un andamento monotono decrescente se non sono presenti altri errori. In questo andamento non si può fare a meno di incontrare almeno un massimo (SLMAX o LMAX).

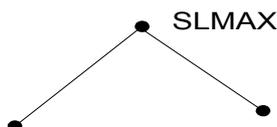


Figura 3.4: Nodo di tipo SLMAX

- LMAX: Tutti i figli hanno valore euristico minore o uguale al padre, almeno uno strettamente maggiore e almeno uno uguale.

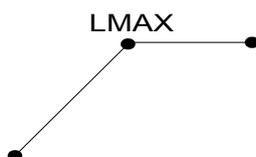


Figura 3.5: Nodo di tipo LMAX

- IPLAT: un punto di plateau; tutti i figli hanno lo stesso valore euristico del padre. È un punto di forte indecisione della funzione euristica. La sottostima del nodo IPLAT è di almeno il costo di un operatore.



Figura 3.6: Nodo di tipo IPLAT

- LEDGE: come SLOPE, ma almeno un figlio ha valore euristico uguale al padre. Sebbene non sia solitamente desiderabile che dei figli abbiano valore euristico uguale al padre un punto di tipo LEDGE non causerà direttamente overhead all'algoritmo.

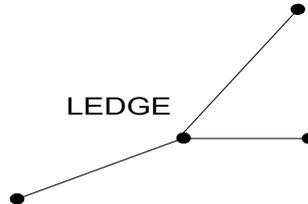


Figura 3.7: Nodo di tipo LEDGE

3.1.1.1 Numero, densità e distribuzione dei minimi locali e plateau

Il numero di minimi locali e plateau (nodi di tipo LMIN, SLMIN, IPLAT) può essere usato come misura diretta di confronto sommario fra euristiche. Più precisamente quello che realmente interessa è la densità di queste misure in rapporto al numero totale di nodi.

Come già accennato i nodi di questi tipi denotano con certezza la presenza di un errore di valutazione della funzione euristica. La quantificazione delle differenze confronto in presenza dei diversi tipi di errori non è tuttavia banale, soprattutto in presenza dei diversi tipi di errori.

Un numero uguale di punti critici non dà alcuna indicazione certa di essere di fronte a euristiche equivalenti; infatti la distribuzione degli stessi è altrettanto importante per valutare il comportamento delle euristiche. Il modo più semplice e intuitivo per studiare la distribuzione è raggruppare i nodi rispetto al loro valore euristico, e analizzare la distribuzione degli errori all'interno delle diverse classi. Ci si aspetta che una distribuzione di minimi molto concentrata nei valori bassi di euristica sia indice di un maggior impatto degli errori stessi sulle prestazioni dell'algoritmo. La spiegazione è questa: nel cammino verso la soluzione in ogni caso si sarà costretti a passare in zone a valore euristico basso, visto che qualsiasi algoritmo tende a minimizzare in qualche modo il valore euristico dei nodi scelti per l'espansione. Data questa premessa un nodo a valore euristico basso sarà con più probabilità visitato rispetto a un nodo a valore più alto, per cui un errore su di esso influenzerà un numero maggiore di istanze del problema. Dall'osservazione della distribuzione sarà dunque da preferire, a parità di numero di errori, una funzione euristica nella quale gli errori si manifestino per valori più alti.

Un altro fattore che potrebbe essere doveroso tenere in conto è quello che si indicherà come *raggio di fuga* (o *escape*) dai punti critici. Non è lo stesso dal punto di vista della ricerca trovarsi di fronte a un minimo dal quale è possibile uscire in due passi, rispetto a una diversa condizione in cui è necessario esplorare una zona più estesa prima di individuare un punto a valore euristico inferiore. Un parametro sintetico che potrebbe essere preso in considerazione è il raggio di fuga medio, ossia il numero di passi mediamente necessari a uscire da un minimo o plateau.

3.1.1.2 Risultati sperimentali

Sono infiniti gli studi possibili riguardo alla distribuzione dei tipi di nodi nello spazio di ricerca. Ci si limiterà a confrontare la densità di nodi per le varie categorie per le diverse euristiche implementate per l'N-puzzle (5, 8 e 15 puzzle), e per l euristica di manhattan per alcuni labirinti. Per quanto riguarda la distribuzione si è usata invece una clusterizzazione in base al valore euristico.

	<i>SLMIN</i>	<i>LMIN</i>	<i>IPLAT</i>	<i>LEDGE</i>	<i>SLOPE</i>	<i>LMAX</i>	<i>SLMAX</i>
manhattan	26.1	0.0	0.0	0.0	56.1	0.0	17.8
linearConflict	20.6	0.0	0.0	0.0	64.4	0.0	15.0
nonLinearConflict	25.6	0.0	0.0	0.0	56.1	0.0	18.3
cornerTiles	23.3	0.0	0.0	0.0	57.2	0.0	19.4
lastMoves	24.4	0.0	0.0	0.0	57.8	0.0	17.8
tilesOut	2.5	22.5	38.9	5.0	5.0	0.0	26.1
cornerDeduction	22.2	0.0	0.0	0.0	59.2	0.0	18.6
conflicts	24.4	0.0	0.0	0.0	53.3	0.0	22.2
conflictDeduction	23.1	0.0	0.0	0.0	52.2	0.0	24.7
lastMovesLinearConflict	20.3	0.0	0.0	0.0	63.6	0.0	16.1
rawConflicts	25.6	0.0	0.0	0.0	51.9	0.0	22.5
allCorrections	24.2	0.0	0.0	0.0	52.2	0.0	23.6
manhattan, depth 1	24.4	0.0	0.0	0.0	57.8	0.0	17.8
manhattan, depth 2	22.5	0.0	0.0	0.0	55.6	0.0	21.9
manhattan, depth 4	27.2	0.0	0.0	0.0	45.6	0.0	27.2
manhattan, depth 6	24.4	0.0	0.0	0.0	49.4	0.0	26.1
manhattan, depth 8	19.2	0.0	0.0	0.0	51.7	0.0	29.2
manhattan, depth 10	15.3	0.0	0.0	0.0	60.8	0.0	23.9

Tabella 3.1.densità per i diversi tipi di posizioni – 5-puzzle

Metodi innovativi per la valutazione di funzioni euristiche

	SLMIN	LMIN	IPLAT	LEDGE	SLOPE	LMAX	SLMAX
manhattan	24.4	0.0	0.0	0.0	62.0	0.0	13.6
linearConflict	21.1	0.0	0.0	0.0	67.4	0.0	11.5
nonLinearConflict	20.3	0.0	0.0	0.0	66.5	0.0	13.2
cornerTiles	21.4	0.0	0.0	0.0	64.4	0.0	14.2
lastMoves	22.9	0.0	0.0	0.0	62.6	0.0	14.5
tilesOut	0.9	19.4	49.2	5.4	1.5	0.0	23.7
cornerDeduction	21.4	0.0	0.0	0.0	64.6	0.0	13.9
conflicts	20.2	0.0	0.0	0.0	65.8	0.0	14.0
conflictDeduction	21.3	0.0	0.0	0.0	63.4	0.0	15.4
lastMovesLinearConflict	20.5	0.0	0.0	0.0	67.2	0.0	12.2
rawConflicts	19.2	0.0	0.0	0.0	67.0	0.0	13.8
allCorrections	18.5	0.0	0.0	0.0	67.1	0.0	14.4
Manhattan, depth 1	22.9	0.0	0.0	0.0	62.6	0.0	14.5
Manhattan, depth 2	23.9	0.0	0.0	0.0	61.5	0.0	14.6
Manhattan, depth 4	23.2	0.0	0.0	0.0	61.3	0.0	15.5
Manhattan, depth 6	23.5	0.0	0.0	0.0	59.3	0.0	17.1
Manhattan, depth 8	23.4	0.0	0.0	0.0	57.9	0.0	18.6
Manhattan, depth 10	24.3	0.0	0.0	0.0	55.6	0.0	20.2

Tabella 3.2. Densità per i diversi tipi di posizioni – 8-puzzle

	SLMIN	LMIN	IPLAT	LEDGE	SLOPE	LMAX	SLMAX
manhattan	19.7	0.0	0.0	0.0	70.6	0.0	9.7
linearConflict	18.5	0.0	0.0	0.0	72.6	0.0	8.8
nonLinearConflict	18.2	0.0	0.0	0.0	72.4	0.0	9.4
cornerTiles	19.9	0.0	0.0	0.0	71.1	0.0	9.0
lastMoves	19.2	0.0	0.0	0.0	70.6	0.0	10.2
tilesOut	0.1	14.8	65.8	2.5	0.2	0.0	16.5
cornerDeduction	20.0	0.0	0.0	0.0	71.0	0.0	9.0
conflicts	18.2	0.0	0.0	0.0	72.3	0.0	9.6
conflictDeduction	18.4	0.0	0.0	0.0	71.5	0.0	10.1
lastMovesLinearConflict	18.1	0.0	0.0	0.0	72.9	0.0	9.1
rawConflicts	17.7	0.0	0.0	0.0	72.9	0.0	9.4
allCorrections	17.7	0.0	0.0	0.0	72.8	0.0	9.5
Manhattan, depth 1	19.2	0.0	0.0	0.0	70.6	0.0	10.2
Manhattan, depth 2	19.2	0.0	0.0	0.0	70.5	0.0	10.3
Manhattan, depth 4	19.0	0.0	0.0	0.0	70.5	0.0	10.5
Manhattan, depth 8	19.1	0.0	0.0	0.0	70.2	0.0	10.7
Manhattan, depth 10	18.9	0.0	0.0	0.0	70.0	0.0	11.1

Tabella 3.3. Densità per i diversi tipi di posizioni – alcuni labirinti

<i>h/type</i>	<i>SLMIN</i>	<i>LMIN</i>	<i>IPLAT</i>	<i>LEDGE</i>	<i>SLOPE</i>	<i>LMAX</i>	<i>SLMAX</i>
20x20 maze 1, manhattan ¹	16.3	0.0	0.0	0.0	67.8	0.0	16.0
20x20 maze 2, manhattan	18.0	0.0	0.0	0.0	65.0	0.0	17.0
50x50 maze, manhattan	17.7	0.0	0.0	0.0	65.6	0.0	16.6

Tabella 3.4. Densità per i diversi tipi di posizioni – 15-puzzle, 200000 campioni casuali

Come si può evincere dai dati tabulati, la semplice densità di minimi e plateau, sebbene discretamente correlata con l'informatività dell'euristica, non è certamente uno strumento sufficiente a dare una valutazione esaustiva della bontà di un'euristica. Il numero di minimi e/o plateau, come ci si poteva aspettare, tende a diminuire per le euristiche più informate. Il numero di slope aumenta dunque di pari passo. Quest'analisi permette di cogliere la grande differenza tra tiles-out-of-place e manhattan, a la discreta differenza tra manhattan e linear-conflict, ma resta un'indecisione di alcuni punti percentuali, per i quali il numero di minimi non è necessariamente collegato all'efficacia dell'euristica. In particolare non sempre coglie le differenze tra le diverse profondità di perimetro. Dunque, oltre al numero di minimi, deve essere tenuto in conto la frequenza con cui si incappa nei minimi stessi durante la ricerca, e il diametro delle zone di minimo. Usando la metafora del landscape come superficie ci si può rendere conto della differenza che può esserci nel saltare un minimo che interessa una vasta zona rispetto a un minimo che interessi una zona circoscritta. Nel primo caso l'errore dell'euristica costringe di fatto l'algoritmo a esplorare una zona sterile più ampia.

L'analisi di distribuzione può permettere di evidenziare questi aspetti in maniera più esaustiva. Un'analisi di distribuzione prevede la suddivisione dello spazio in cluster per evidenziare la frequenza dei tipi di posizioni all'interno delle diverse classi di nodi. Una clusterizzazione certamente efficace è la suddivisione dei nodi in base al valore euristico. Un fatto che deve essere tenuto conto infatti è che i nodi a valore euristico più basso saranno con più probabilità attraversati durante la ricerca: la ricerca è infatti attratta dai nodi a valore euristico più basso, per il fatto che tende a minimizzare il valore euristico. Si è eseguita quest'analisi sul 8-puzzle, è il fine è evidenziare gli aspetti della distribuzione che determinano le prestazioni dell'euristica al di là della semplice densità.

¹ Vedi appendice C per i dettagli sui labirinti

<i>h/type</i>	<i>SLMIN</i>	<i>LMIN</i>	<i>IPLAT</i>	<i>LEDGE</i>	<i>SLOPE</i>	<i>LMAX</i>	<i>SLMAX</i>
0	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000
2	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000
3	0.20000	0.00000	0.00000	0.00000	0.80000	0.00000	0.00000
4	0.80870	0.00000	0.00000	0.00000	0.19130	0.00000	0.00000
5	0.26016	0.00000	0.00000	0.00000	0.72358	0.00000	0.01626
6	0.43885	0.00000	0.00000	0.00000	0.50647	0.00000	0.05468
7	0.23280	0.00000	0.00000	0.00000	0.74250	0.00000	0.02469
8	0.51929	0.00000	0.00000	0.00000	0.43830	0.00000	0.04241
9	0.20063	0.00000	0.00000	0.00000	0.75020	0.00000	0.04917
10	0.42431	0.00000	0.00000	0.00000	0.49295	0.00000	0.08273
11	0.13252	0.00000	0.00000	0.00000	0.78469	0.00000	0.08279
12	0.41535	0.00000	0.00000	0.00000	0.49698	0.00000	0.08767
13	0.08653	0.00000	0.00000	0.00000	0.79770	0.00000	0.11577
14	0.37444	0.00000	0.00000	0.00000	0.50534	0.00000	0.12022
15	0.04629	0.00000	0.00000	0.00000	0.78715	0.00000	0.16656
16	0.34702	0.00000	0.00000	0.00000	0.50839	0.00000	0.14459
17	0.01785	0.00000	0.00000	0.00000	0.76677	0.00000	0.21538
18	0.33090	0.00000	0.00000	0.00000	0.48277	0.00000	0.18633
19	0.00000	0.00000	0.00000	0.00000	0.72659	0.00000	0.27341
20	0.28996	0.00000	0.00000	0.00000	0.40287	0.00000	0.30717
21	0.00000	0.00000	0.00000	0.00000	0.68803	0.00000	0.31197
22	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000

Tabella 3.5. Distribuzione dei tipi di nodi in base al valore euristico, 8-puzzle

In tabella 3.5 è indicato la densità di nodi di ogni tipo per ogni valore euristico, e ci sono diversi aspetti interessanti che devono essere notati. Innanzitutto i nodi sono esclusivamente di tipo slope fino al valore euristico 2: questo fatto è legato alla perfetta informatività di manhattan fino a quel valore euristico. Un aspetto ancora più interessante è però la percentuale di SLMIN a valore euristico 4; proprio per questo valore si ha infatti il massimo picco di aumento dell'errore di manhattan: per una distanza stimata di 4 il valore medio di distanza è infatti prossimo a 13, e da lì in poi l'errore si mantiene elevato nonostante il numero di minimi scenda; si può indurre che a parità di minimi una funzione euristica che presenti gli stessi per valori più bassi è molto probabilmente quella che corrisponderà a prestazioni inferiori. In altre parole un errore evidenziato per un valore euristico basso si propaga nei valori euristici più elevati. Questo fatto è abbastanza intuitivo se si considera una funzione euristica consistente, per la quale tutti i valori euristici inferiori a quello di partenza sono visitati almeno una volta nel percorso verso la soluzione. In figura 3.8 si può vedere l'andamento della densità di minimi per diverse funzioni euristiche¹

¹ Un leggero shift è stato applicato alle curve per agevolare la lettura del grafico; i valori delle ascisse sono da intendersi in ogni caso discreti interi.

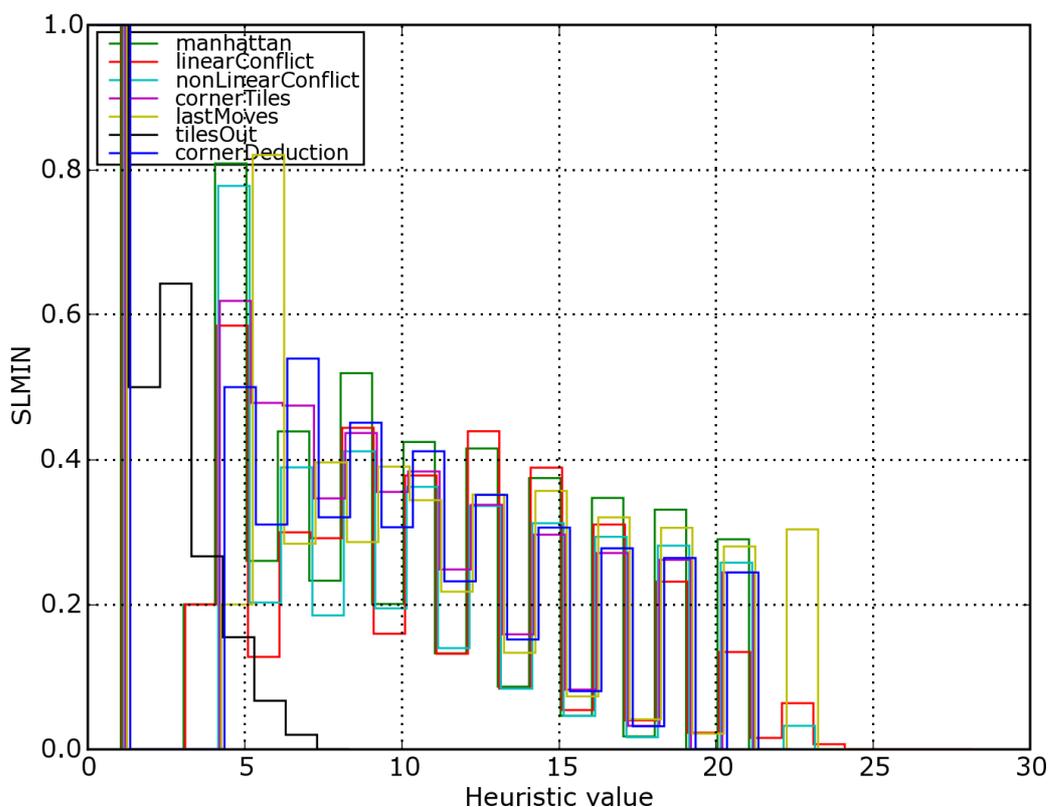


Figura 3.8.: Distribuzione dei minimi locali per diverse profondità di perimetro– 8-puzzle

Si può notare che il picco a valore 4 è inferiore per le euristiche più informate. Last-moves in particolare ha l'effetto di spostare in avanti questo picco. Questo suggerisce che sebbene abbia una densità di minimi superiore ad esempio a corner-tiles, possa essere comunque più efficace di questa. Questo effetto è dovuto alla correzione apportata alle ultime mosse (si ricorda che last-moves equivale a una manhattan con perimetro 1). L'analisi riportata in figura 3.9, oltre a confermare l'ultima affermazione fornisce un risultato notevole: l'altezza del primo picco è proporzionale alla profondità del perimetro. Questo fenomeno che può essere spiegato col fatto che in un algoritmo con profondità di perimetro d nessun errore euristico sarà fatto per i valori inferiori o uguali a d .

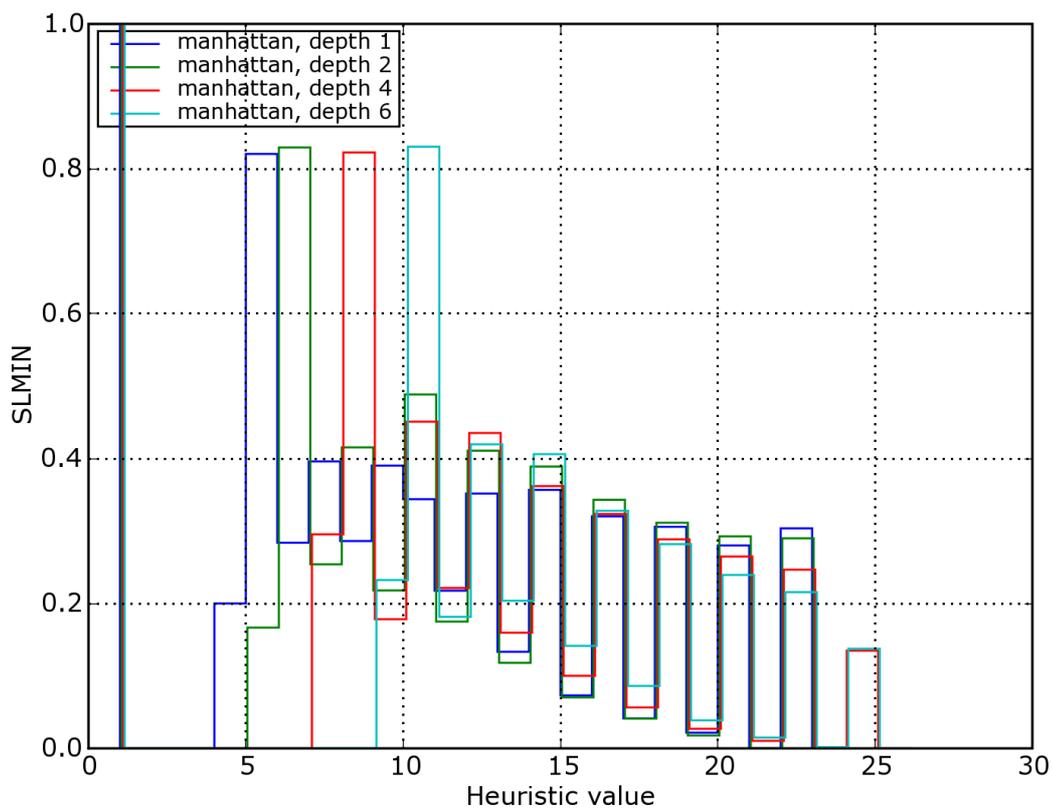


Figura 3.9.: Distribuzione dei minimi locali per diverse profondità di perimetro-8-puzzle

In figura 3.10 è visualizzata la stessa analisi condotta sull'istanza 1 di labirinto 20×20 . Anche in questo caso si presenta uno shift simile nella distribuzione degli errori, anche se c'è una certa tolleranza dovuta alle caratteristiche singolari del labirinto. Un'altra caratteristica interessante del labirinto è il fatto che più che il numero di minimi ciò che influenza maggiormente le prestazioni di manhattan è il raggio di fuga dagli stessi; se si pensa alla struttura di un labirinto perfetto se ne capisce anche il motivo: una volta presa la strada sbagliata, si può andare avanti per molti passi prima di accorgersi che in realtà quella strada non portava alla soluzione.

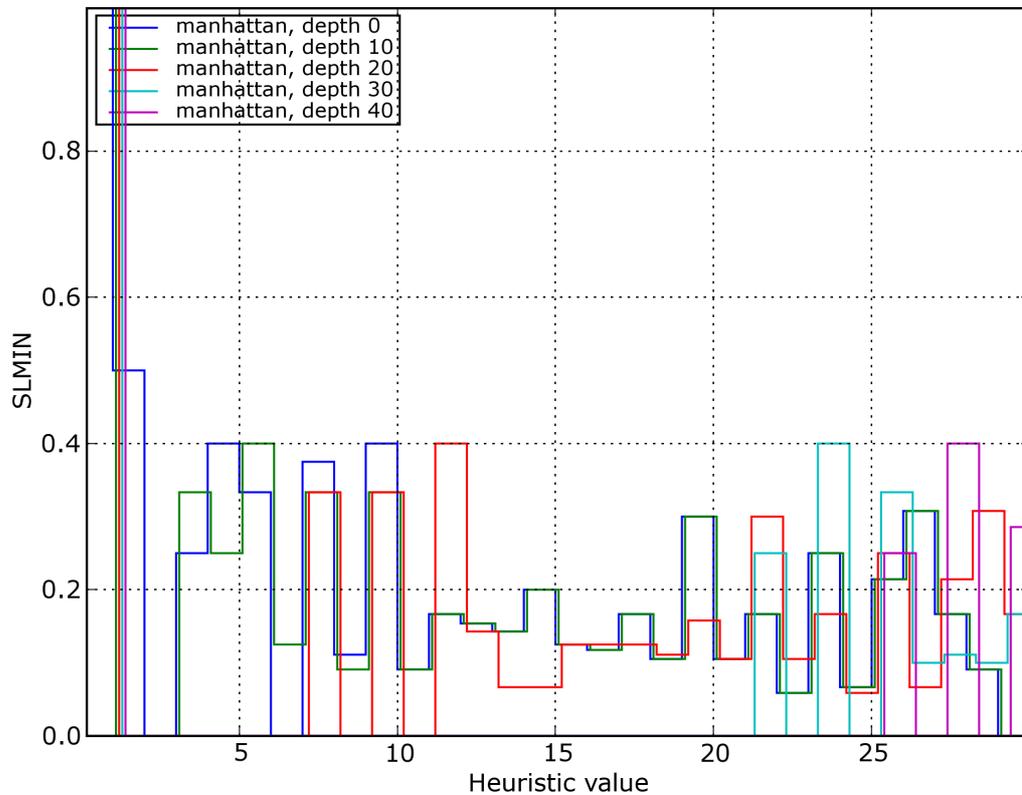


Figura 3.10.: Distribuzione dei minimi locali per diverse profondità– Labirinto 20x20, istanza 1

L'ultima analisi riguarda proprio il raggio di fuga dai minimi e plateau: è calcolato facendo una piccola ricerca locale intorno a ogni minimo o plateau fino a trovare un nodo con valore euristico inferiore. Il raggio è proprio la distanza di questo punto dal punto di partenza.

<i>Problem, heuristic</i>	<i>SLMIN</i>	<i>IPLAT</i>
8-puzzle, manhattan	2.5	0.0
8-puzzle, linearConflict	2.7	0.0
8-puzzle, nonLinearConflict	2.5	0.0
8-puzzle, cornerTiles	2.7	0.0
8-puzzle, lastMoves	2.4	0.0
8-puzzle, tilesOut	5.4	2.8
8-puzzle, cornerDeduction	2.7	0.0
8-puzzle, conflicts	2.6	0.0
8-puzzle, conflictDeduction	2.5	0.0
8-puzzle, lastMovesLinearConflict	2.6	0.0
8-puzzle, rawConflicts	2.7	0.0
8-puzzle, allCorrections	2.5	0.0
8-puzzle, manhattan, depth 0	2.5	0.0
8-puzzle, manhattan, depth 2	2.5	0.0
8-puzzle, manhattan, depth 4	2.5	0.0
8-puzzle, manhattan, depth 6	2.6	0.0
15-puzzle, manhattan	2.3	0.0
Maze 20x20 1, manhattan, depth 0	10.7	0.0
Maze 20x20 1, manhattan, depth 10	11.0	0.0
Maze 20x20 1, manhattan, depth 20	12.4	0.0
Maze 20x20 1, manhattan, depth 30	11.9	0.0
Maze 20x20 1, manhattan, depth 40	12.0	0.0
Maze 20x20 2, manhattan	9.5	0.0
Maze 50x50, manhattan	11.5	0.0

Tabella 3.6. Raggio di escape medio per diverse euristiche e problemi

Dall'analisi sul raggio di escape si può notare che l'euristica non contribuisce in maniera sensibile a modificarne il valore. Il raggio è notevolmente più elevato per i labirinti: dunque sebbene la densità di minimi sia paragonabile al N-puzzle per la distanza di manhattan, questa risulta decisamente meno efficace per un labirinto perfetto. Non si nota infine alcuna correlazione tra la profondità di perimetro e il raggio medio.

3.1.2 Stima della profondità della soluzione basata sulla distribuzione di minimi locali e plateau

Si vede in questa sezione un inedito approccio che può essere utilizzato per stimare la profondità media della soluzione ottima di un problema, basato sullo studio di landscape locale. Conoscere a priori la profondità della soluzione che ci si aspetta per i diversi cluster dello spazio degli stati può essere, oltre che di per sé un ottimo metro di confronto delle funzioni euristiche, una utile indicazione sulle caratteristiche di un problema di ricerca. La conoscenza di un valore attendibile della profondità d permetterebbe di usare il valore stesso come parametro nel metodo di Korf (nel quale il valore della profondità non è meglio specificato), che si completerebbe potendo dare una stima del numero di nodi espansi o generati in media nel processo di ricerca e dunque del tempo necessario in media a risolvere le istanze di un problema.

L'idea di base è individuare la percentuale di errori ai diversi livelli e attraverso di essa correggere verso l'alto il valore medio stimato dall'euristica.

È stato già accennato nel precedente paragrafo nella descrizione dei tipi di nodi come alcuni di questi siano indice di un errore certo della funzione euristica. È possibile quantificare questo errore:

SLMIN: maggiore o uguale al costo di 2 operatori

LMIN: maggiore o uguale al costo di 1 operatore

IPLAT: maggiore o uguale al costo di 1 operatore

Questo semplice assunto potrebbe in realtà non risultare ovvio in prima analisi, ma si basa su delle considerazioni molto semplici sul modo in cui si comporta una funzione euristica perfettamente informata. Un'euristica di questo tipo, che ha come valore per ogni nodo la distanza reale dalla soluzione¹, guiderà un algoritmo A* alla soluzioni espandendo solo i nodi che vanno dal nodo iniziale alla soluzione (vedi dimostrazioni sull'ottimalità di A*, ad esempio [Pearl84]). Diretta conseguenza di questo fatto è che qualunque minimo locale sarà necessariamente minimo globale e dunque soluzione, così come i massimi saranno soltanto i nodi a valore euristico massimo.

Dim: Si supponga per assurdo di avere un nodo N_f minimo locale con valore euristico h_f che non sia soluzione; qualunque figlio N_c di N_f avrà dunque un valore euristico $h_c > h_f$. Il costo del cammino del figlio sarà invece $g_c = g_f + c_{fc}$, dove c_{fc} è il costo del cammino da N_f a N_c . Essendo $c_{fc} > 0$ si avrebbe:

$h_c + g_c > h_f + g_f$, andando contro l'ipotesi iniziale di euristica perfettamente informata.

Una funzione euristica perfettamente informata non avrà dunque alcun minimo locale o plateau, se si escludono ovviamente i nodi obiettivo. Questa osservazione permette di indurre che la presenza di un minimo locale o di un punto di plateau sia corrispondente a un errore della funzione euristica. Nel momento in cui si individua un errore, si può applicare la dovuta correzione alla stima del nodo affetto dallo stesso.

Supponendo, ponendosi nel migliore dei casi, che il valore euristico del figlio nel percorso verso la soluzione sia esatto (ossia $h_c = h_c^*$), per mantenere costante il valore di $h + g$ si deve aggiungere a un nodo minimo locale due volte il costo del percorso dal padre al figlio².

Dim: Affinché sia $h_c + g_c = h_f^* + g_f$ dovrà essere $h_f^* = h_c + g_c - g_f$. Essendo $g_c = g_f + c_{fc}$ (il costo del figlio è uguale al costo del padre più il cammino dal figlio al padre) allora $h_f^* = h_c^* + c_{cf} = h_c + c_{cf}$. Sostituendo $h_c = h_f + c_{fc}$ si ha $h_f^* = h_f + 2c_{fc}$.

Nella figura si può vedere più chiaramente il fenomeno per un nodo di tipo SLMIN.

-
- 1 Questo equivale a dire che la funzione di valutazione $h + g$ è costante. In realtà la condizione non è necessaria, sarebbe un'euristica perfettamente informata ad esempio anche quella che si otterrebbe applicando un bias all'euristica descritta. Una traslazione di questo tipo non porterebbe in ogni caso ad alcun cambiamento nelle considerazioni successive.
 - 2 Si supponrà d'ora in poi che il costo di un operatore sia costante per semplicità; potrebbe essere fatta una trattazione simile per il caso più generale.

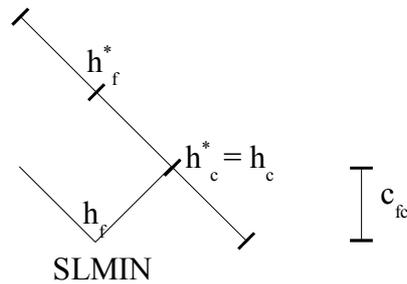


Figura 3.11: Errore minimo per un nodo di tipo SLMIN

Un'analoga dimostrazione a quella di cui sopra può essere fatta per i punti di plateau¹; in questo caso avremo un coefficiente 1 invece di 2.

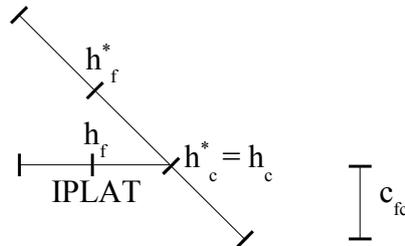


Figura 3.12: Errore minimo per un nodo di tipo IPLAT

In questo caso può essere notato un altro fatto interessante: il nodo padre di un nodo di tipo IPLAT avrà avuto necessariamente un errore di classificazione di 2 passi².

Merita ancora un'osservazione più attenta il caso di un nodo di tipo LMIN; infatti questo tipo di nodo può corrispondere a diverse situazioni teoriche, a seconda del valore del figlio che effettivamente si trova sulla strada per la soluzione.

1. Il figlio ha lo stesso valore euristico del nodo LMIN. In questo caso la sottostima nel caso migliore sarà pari a 1 passo.

¹ Si omette la dimostrazione, nella sostanza uguale alla precedente.

² Si ipotizzi che gli operatori siano reversibili: in questo caso il padre può essere sempre generato a partire da un figlio e dunque sarà compreso nella classificazione dello stesso.

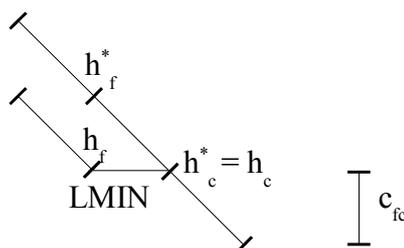


Figura 3.13: Errore minimo per un nodo di tipo LMIN: caso 1

2. Il figlio ha valore euristico superiore a LMIN. Questo caso è equivalente dal nostro punto di vista a quello di un nodo SLMIN, e dunque la sottostima sarà sempre superiore a 2 passi.

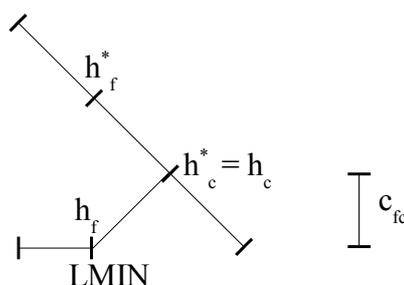


Figura 3.14: Errore minimo per un nodo di tipo LMIN: caso 2

3. Si potrebbe in realtà considerare il caso in cui sia il padre del nodo LMIN ad avere stesso valore euristico; è facilmente intuibile che la situazione è equivalente dal punto di vista del nostro calcolo. Il padre sarebbe invece sottostimato di almeno tre passi.

Risulta impossibile individuare quale sia la reale situazione cui corrisponde un nodo classificato come LMIN. Trattandosi nel nostro caso di sottostime, potrebbe essere adottato il coefficiente 1 per tutti i nodi senza fare alcuna distinzione.

Ora si vede come questi presupposti possano essere usati per stimare la reale distanza media dalla soluzione.

Si considerano in prima battuta, per semplificare la trattazione, soltanto i nodi di tipo SLMIN, e un'euristica ammissibile e consistente, in cui il costo di un operatore sia sempre pari a un'unità. Potremmo incrementare di almeno due unità il valore euristico di ogni nodo di questo tipo per migliorarne la stima.

Il punto di partenza è la conoscenza della distribuzione di questi errori per i diversi valori euristici, ovverosia la probabilità che un nodo di valore euristico h sia un nodo di tipo SLMIN. Un'altra preconditione inoltre è che la probabilità di visitare i nodi con lo stesso valore euristico sia uniforme. Quest'ultima condizione in realtà non si verifica mai, ma può essere abbastanza ragionevole per i nodi a basso valore euristico. Si vedrà nelle sperimentazioni come il fatto che questa condizione non si verifichi sia causa di una

deviazione abbastanza consistente nella stima.

Sia $p_{SLMIN}(h)$ la probabilità che un nodo con valore euristico h sia di tipo SLMIN. Si può supporre che in media la vera distanza dalla soluzione per un nodo di valore euristico h sia almeno:

$$d(h, p_{SLMIN}(h)) = h + 2 \cdot p_{SLMIN}(h) \quad (3.1)$$

Nel percorso verso la soluzione, nel momento in cui si “esce” da un SLMIN si arriva necessariamente a un nodo con valore euristico $h+1$; da esso si arriverà nuovamente a un nodo di valore euristico h , che a sua volta potrà essere un minimo con la stessa probabilità del precedente. Si arriverà poi a nodi di valore euristico $h-1$ e così via fino a 0, e ogni nodo di passaggio avrà la sua probabilità di essere un minimo. Dunque un errore su un nodo di un determinato valore euristico possa ripercuotersi sui nodi a valore più alto, che saranno maggiormente sottostimati.

Si può così generalizzare, introducendo la correzione media $k(h)$, nella (3.1):

$$d(h, p_{SLMIN}(h)) = h + 2 \cdot k(h), \quad (3.2)$$

dove

$$\begin{aligned} k(1) &= 2 \cdot \text{converge}(p_{SLMIN}(1)) \\ k(h) &= k(h-1) + 2 \cdot \text{converge}(p_{SLMIN}(h)). \end{aligned} \quad (3.3)$$

La funzione *converge* tiene conto della probabilità di incappare in altri minimi durante il percorso verso la soluzione.

Una possibile semplice formulazione di questa funzione è la seguente:

$$\text{converge}(x) = \sum_{i=0}^N x^i. \quad (3.4)$$

Questa funzione tiene conto della probabilità di tornare a un nodo dello stesso valore di quello di partenza che sia anch'esso un minimo.

Il valore di N è tale da avere una convergenza asintotica del valore della funzione, in modo che $x^N \ll x$.

Lo stesso metodo può essere usato nel caso in cui ci siano errori di tipo IPLAT, eliminando il termine moltiplicatore 2. Più complesso è il caso degli LMIN, che sono delle vie di mezzo tra minimi e plateau. Il termine moltiplicatore sarà ragionevolmente compreso tra 1 e 2, tendendo maggiormente a 1 essendo più probabile che il nodo di pari valore euristico porti alla soluzione.

Per tenere conto di tutti gli effetti congiuntamente non è sufficiente la semplice somma dei termini correttivi calcolati separatamente: nel cammino verso la soluzione si potranno incontrare un numero di nodi dei diversi tipi con una probabilità che dipenderà da tutte le probabilità per tutte le tipologie di errore.

Quello che spesso accade è che per una data coppia problema-euristica si ha la prevalenza di uno dei tre tipi di errore. Ad esempio se si considera l'N-Puzzle con l'euristica manhattan non si riscontra nessun LMIN e IPLAT, mentre nel caso dell'euristica tiles-out-of-place c'è una netta predominanza di nodi di tipo IPLAT.

Questo approccio è interessante, più che per i numeri che permette di ottenere allo stato attuale di approssimazione, il cui valore come si vedrà dagli esperimenti ha un errore ancora elevato, per comprendere più a fondo la relazione fra la distribuzione dei diversi tipi di nodi e le caratteristiche delle funzioni euristiche.

3.1.2.1 Risultati sperimentali

In questo paragrafo sono presentati i risultati ottenuti nella stima della distanza della soluzione ottima. Il problema considerato è l'8-puzzle. Si presenteranno i risultati del confronto di diverse funzioni euristiche e con diversi tipi di campionamento, valutando quale sia il tipo di campionamento più adatto allo scopo.

In figura 3.15 si può vedere il valore stimato dall'implementazione usata del metodo descritto nel paragrafo, per il problema 8-puzzle e usando l'euristica di manhattan. Per ricavare le statistiche dei tipi di nodi è stato eseguito il campionamento dell'intero spazio degli stati. Il valore è confrontato con la reale distanza media della soluzione ottima.

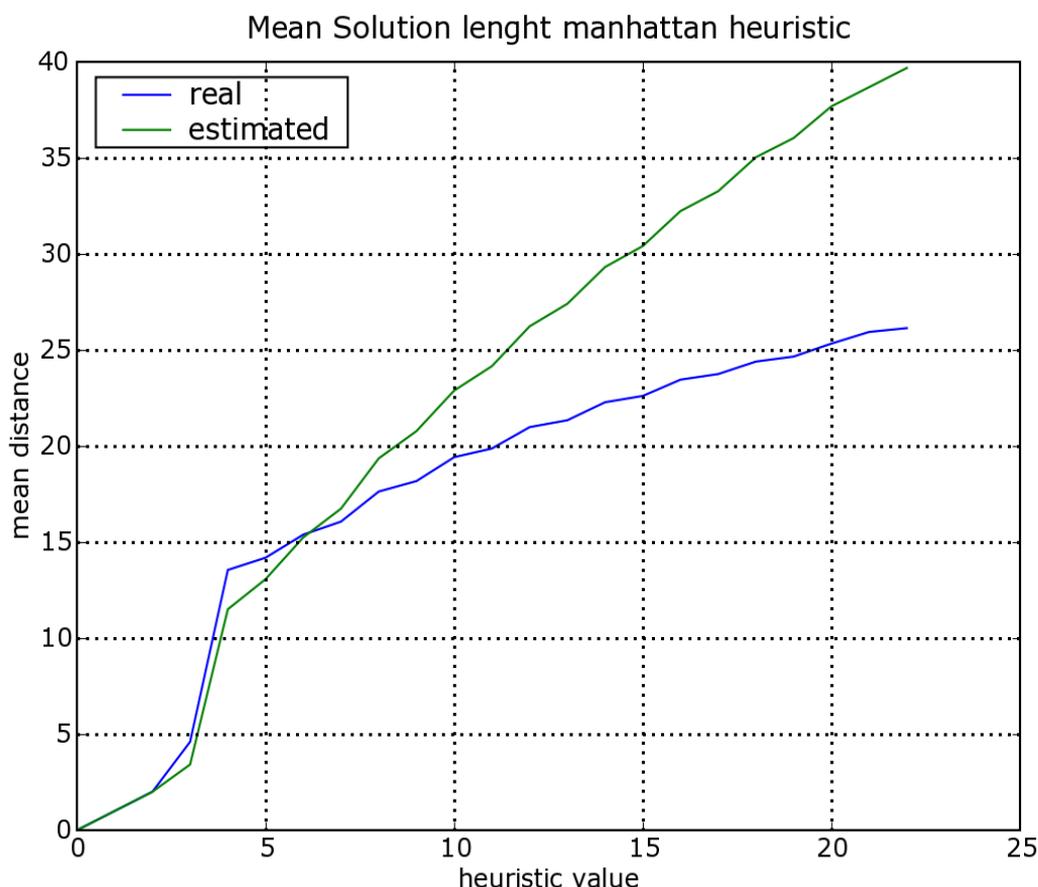


Figura 3.15: distanza media dalla soluzione stimata per l'8-puzzle; euristica di manhattan; campionamento esaustivo; confronto con la distanza media reale

Si può osservare come il modello proposto riesca a stimare con buona approssimazione la distanza per valori euristici relativamente piccoli; il fatto più interessante è che riesce a cogliere il fenomeno di salita rapida del valore di distanza rispetto al valore stimato

dall'euristica che si riscontra per i valori 3 e 4, per i quali è risaputo che l'euristica distanza di manhattan ha un brusco calo di informazione. La stima invece risulta sempre meno accurata all'aumentare del valore euristico: infatti l'errore dell'euristica di manhattan diventa relativamente più piccolo all'aumentare della distanza dalla soluzione. Il modello usato non riesce a cogliere questo fatto per via dell'ipotesi iniziale che gli errori siano distribuiti uniformemente nei diversi percorsi verso la soluzione. Quello che invece accade nell'*N*-puzzle è che partendo da un nodo lontano dalla soluzione si passerà per percorsi meno affetti da errori. Ovverosia, il nodo di euristica h che si trova su un percorso di un nodo lontano dalla soluzione avrà una probabilità di essere un punto di minimo o un plateau inferiore alla media.

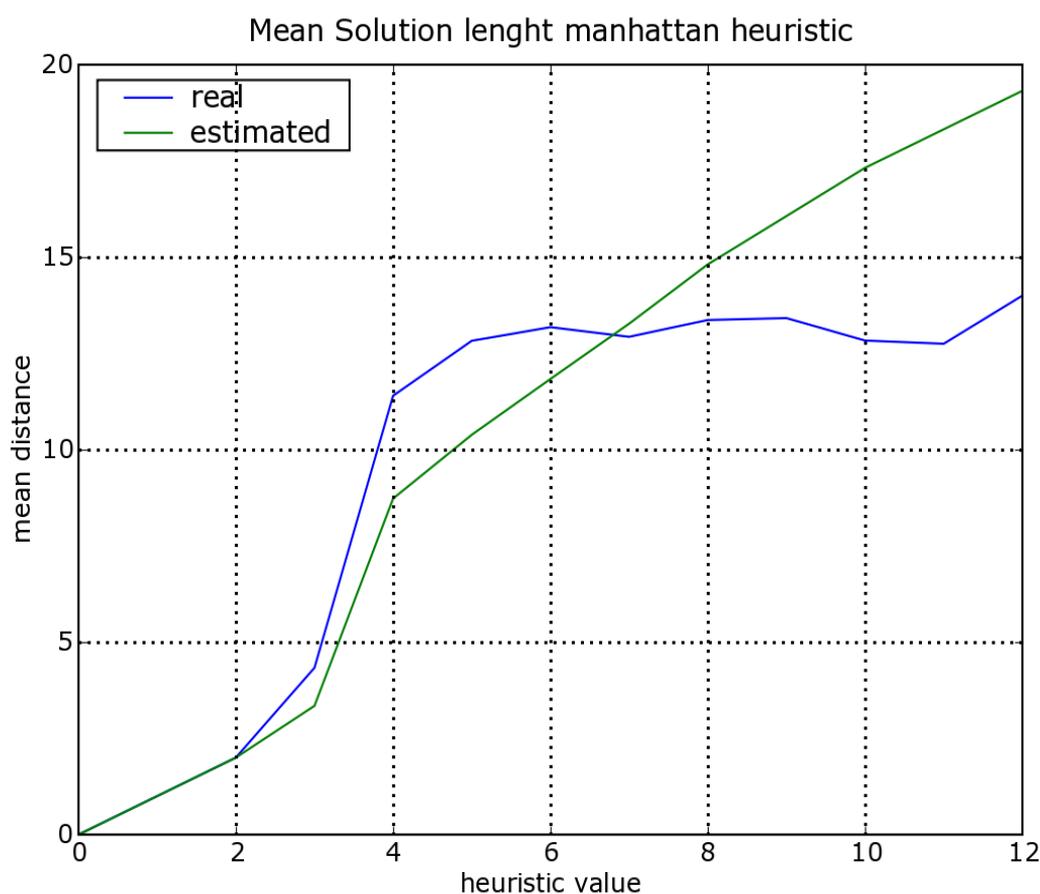


Figura 3.16: Confronto distanza predetta e reale per il 5-puzzle (3x2).

In figura 3.16 e 3.17 vediamo invece il risultato per il medesimo esperimento per il 5-puzzle e il 7-puzzle; anche in questi casi si riscontra un calo di precisione della stima all'aumentare del valore euristico.

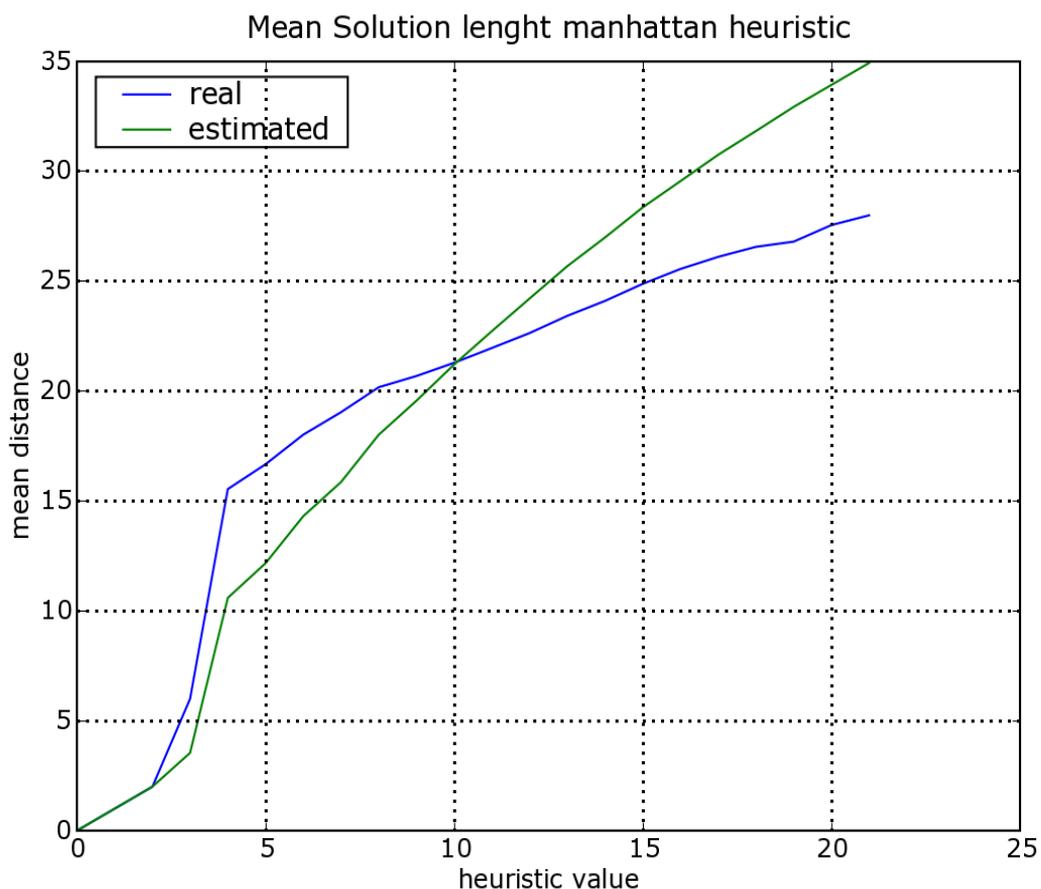


Figura 3.17.: Confronto distanza predetta e reale per il 7-puzzle (4x2).

Un'altra analisi interessante è il confronto delle funzioni euristiche attraverso questo metodo: una euristica maggiormente accurata ha infatti mediamente un valore di distanza più vicino al valore euristico. La figura 3.18 mostra graficamente l'andamento reale per diverse euristiche per l'8-puzzle. Ci si aspetterebbe che graficamente tutte le curve siano sottese alla curva dell'euristica di manhattan; questo fatto si verifica in gran parte ma non è necessariamente vero in assoluto. L'euristica non linear-conflict mostra di apportare un discreto miglioramento per il valore euristico 3, superiore a qualsiasi altro metodo di correzione, mentre d'altra parte i valori classificati con il valore 4 risultano meno accurati (questo non vuol dire assolutamente che non-linear-conflict sia meno accurata per una classe di nodi, essendo già stato appurato essere dominante rispetto a manhattan!).

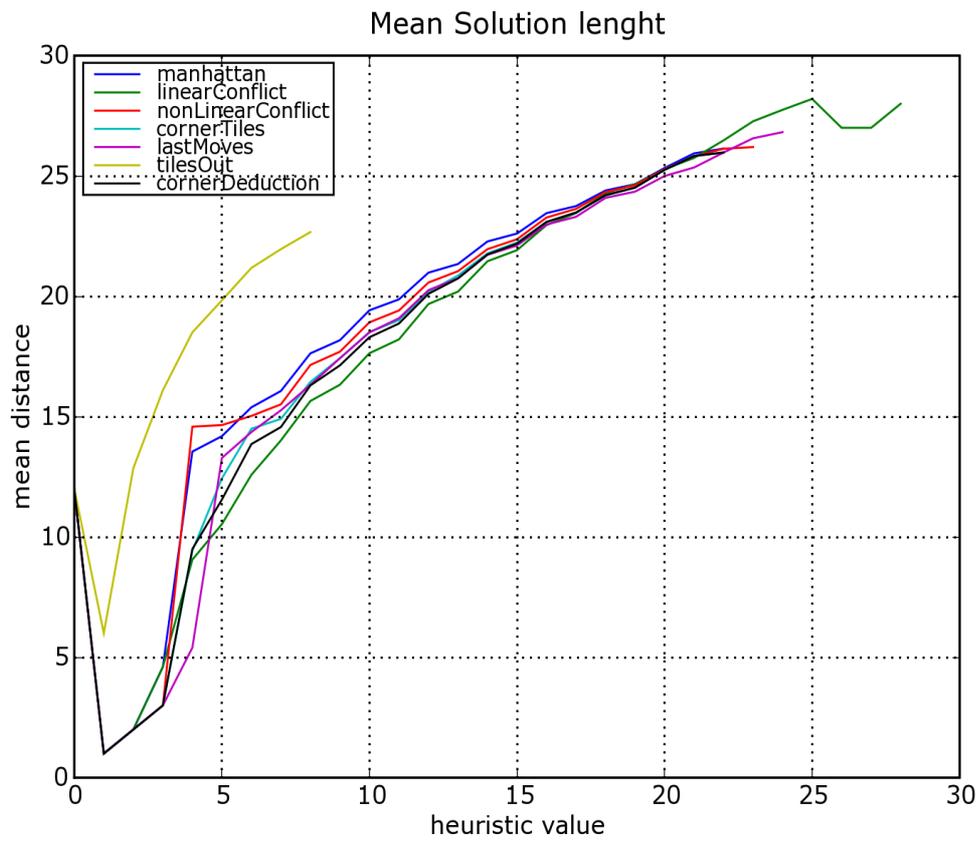


Figura 3.18: Distanza reale media della soluzione per l'8puzzle – confronto euristiche

In figura 3.19 c'è il risultato dell'analisi col metodo di stima della distanza per diverse euristiche; i rapporti sono discretamente mantenuti, anche se non riesce a cogliere lo strano innalzamento di non-linear-conflict per il valore euristico 4.

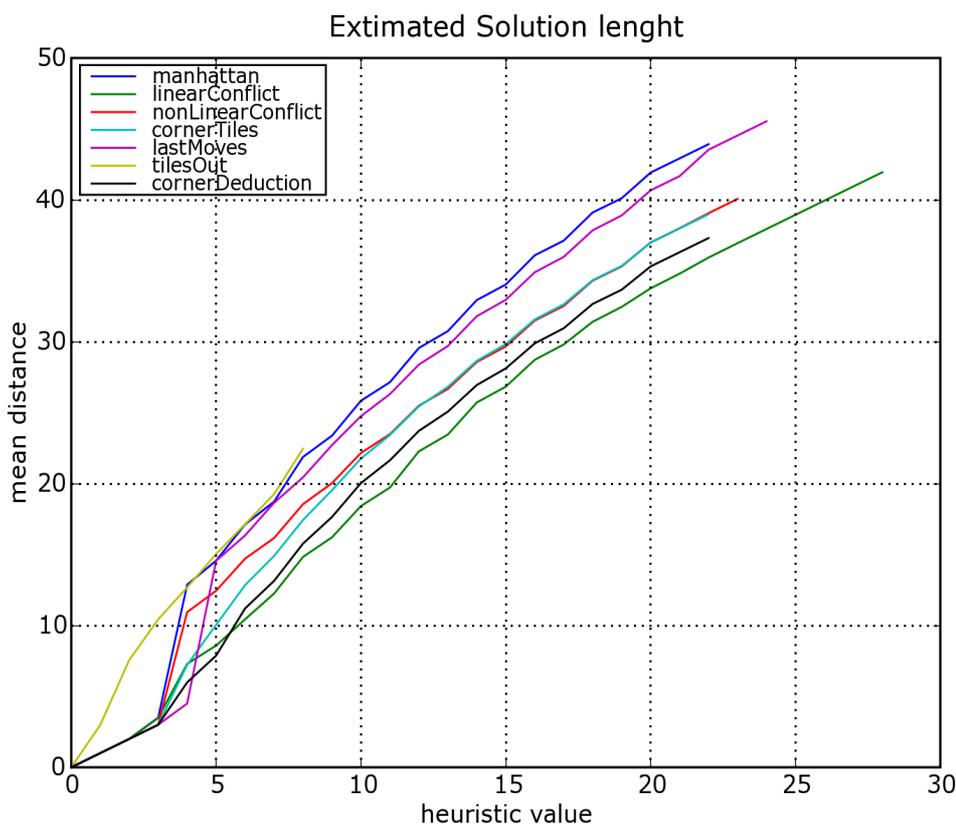


Figura 3.19.: confronto distanze stimate per diverse euristiche sull'8-puzzle

In conclusione si può affermare che il metodo segue con buona approssimazione l'andamento reale della distanza dalla soluzione; ci sono inoltre buone possibilità di miglioramento essendo stata usata una statistica notevolmente semplice. I punti in cui concentrare eventuali successivi lavori sono la funzione *converge*, e la formulazione di $k(h)$, che potrebbe tenere conto di interazioni più complesse tra gli errori ai diversi livelli e/o del campionamento del raggio di escape medio.

3.1.3 Fitness-distance analysis

L'analisi di tipo fitness-distance consiste nel confronto tra i vettori dei valori euristici e delle reali distanze dalla soluzione. È evidente che l'unico modo per conoscere le distanze reali sia quello di risolvere tutte le istanze del problema per il sample set.

Per questo motivo questa tecnica non può essere considerata come un metodo di valutazione delle euristiche a priori. È comunque interessante studiare questo tipo di relazione, essendo il risultato certamente più attendibile di altri metodi approssimati, e in particolare può essere interessante la generalizzazione dei risultati a istanze più grandi del medesimo problema.

3.1.3.1 Fitness-distance correlation

Un aspetto interessante è il valore del coefficiente di correlazione tra il vettore di fitness

(valori euristici) e distanze.

Questo tipo di studio è completamente generale e non richiede alcuna supposizione sul tipo di euristica e sul problema. Infatti il coefficiente di correlazione, più di altre misure d'errore come ad esempio l'MSE, tiene conto della reale efficacia dell'euristica nel guidare l'algoritmo nella ricerca.

Come semplice esempio, si consideri un'euristica che abbia un offset costante rispetto a un'altra; l'efficacia delle 2 euristiche è la medesima, e sarà il medesimo il coefficiente di correlazione. L'MSE sarà invece differente, più precisamente proporzionale al quadrato dell'offset.

Un valore di correlazione pari a 1 apparterrà a un'euristica perfettamente informata, mentre un'euristica assolutamente non informata (valori casuali) avrà un coefficiente di correlazione vicino a 0.

Va da sé che un'euristica con coefficiente di correlazione sia con ogni probabilità più informata.

3.1.3.2 Risultati sperimentali

In tabella 3.7 è riportato il coefficiente di correlazione ricavato per diverse analisi fitness-distance. Il coefficiente di correlazione sembra sintetizzare bene i rapporti di informatività tra le varie euristiche; per l'8-puzzle si registra il coefficiente più alto per l'euristica conflict-deduction fra le manhattan corrections, che risulta più o meno equivalente al coefficiente per un'euristica da perimetro tra 6 e 8. È invece meno efficace l'analisi per quanto riguarda il labirinto, per cui si presentano valori notevolmente variabili tra le istanze, e il coefficiente non sempre aumenta con la profondità del perimetro. La distanza di manhattan mostra infine una correlazione maggiore con le distanze reali per istanze più grandi del puzzle .

<i>Problem, heuristic</i>	<i>Fitness-Distance Correlation coefficient</i>
8-puzzle, manhattan	0.557673508268
8-puzzle, linearConflict	0.6575125679
8-puzzle, nonLinearConflict	0.564342427029
8-puzzle, cornerTiles	0.569551088612
8-puzzle, lastMoves	0.599778394014
8-puzzle, tilesOut	0.247039525699
8-puzzle, cornerDeduction	0.573093741885
8-puzzle, conflicts	0.666570049535
8-puzzle, conflictDeduction	0.68292830249
8-puzzle, lastMovesLinearConflict	0.680196031035
8-puzzle, rawConflicts	0.669947675167
8-puzzle, allCorrections	0.661584555451
8-puzzle, manhattan, depth 2	0.604714223626
8-puzzle, manhattan, depth 4	0.632103745623
8-puzzle, manhattan, depth 6	0.670303245131
8-puzzle, manhattan, depth 8	0.700851275227
Maze 20x20 1, manhattan, depth 0	0.0602103127859
Maze 20x20 1, manhattan, depth 10	0.0487504247832
Maze 20x20 1, manhattan, depth 20	0.0325572278418
Maze 20x20 1, manhattan, depth 30	0.256727731998
Maze 20x20 1, manhattan, depth 40	0.302538061583
Maze 20x20 2, manhattan	0.441451076641
5-puzzle, manhattan	0.268519415639
7-puzzle, manhattan	0.443971754762
15-puzzle, manhattan, 1000 random samples	0.762727597029
15-puzzle, linearConflict, 1000 random samples	0.80748939565
15-puzzle, nonLinearConflict, 1000 random samples	0.775553074807
15-puzzle, cornerTiles, 1000 random samples	0.77668698838
15-puzzle, lastMoves, 1000 random samples	0.772501717591
15-puzzle, tilesOut, 1000 random samples	0.256020197491

Tabella 3.7. Coefficiente di correlazione

3.1.4 Landscape ruggedness

Questa tecnica consiste nello studio dell'euristica sulla base della rugosità della superficie del landscape; l'idea di base è che un'euristica maggiormente informata mantenga dei valori abbastanza simili e con un andamento regolare fra un nodo e i suoi vicini. Si vuole in altre parole pensare al landscape come a un'immagine perturbata da rumore pseudocasuale; il rumore è dovuto proprio agli errori di valutazione della funzione euristica. La misura della rugosità della superficie del landscape può essere dunque indice dell'informatività della funzione stessa.

Per poter misurare la rugosità del landscape è necessario dare una definizione operativa dei due termini, fino ad ora introdotti soltanto come concetti puramente teorici.

Si è definito il landscape come una superficie in x dimensioni in cui ogni punto è rappresentato da un nodo col suo valore euristico; la vicinanza tra i nodi è data dalla relazione

di parentela (un nodo è vicino ad un altro se uno dei due può essere generato dall'altro).

Dal punto di vista pratico il modo più semplice di operare è costruire una superficie 1D eseguendo un campionamento random-walk a partire da uno o più nodi dati in sequenza (ad esempio casuali).

La ruggedness della superficie può invece essere definita tramite la funzione di autocorrelazione r . In particolare è interessante il valore di autocorrelazione col nodo vicino ($r(1)$). Sotto l'ipotesi di andamento esponenziale decrescente può essere definita la lunghezza di correlazione, la costante che governa l'andamento esponenziale:

$$l = \frac{1}{\ln(|r(1)|)}.$$

Ci si aspetta una lunghezza di correlazione maggiore quando l'euristica sia maggiormente informata.

Questo metodo può essere interessante perché, così come le altre analisi di landscape, può essere applicato a funzioni euristiche ammissibili e non ammissibili indistintamente. Negli esperimenti più nel dettaglio la relazione tra i parametri sintetici proposti e le performance degli algoritmi.

3.1.4.1 Risultati sperimentali

Nella tabella 3.8 è riportato il risultato di un'analisi eseguita per l'8-puzzle per 2000 random walk si profondità 100 che partono da altrettanti campioni casuali.

<i>Problem, heuristic</i>	<i>Correlation lenght</i>
8-puzzle, manhattan	14.208005513
8-puzzle, linearConflict	13.2336335004
8-puzzle, nonLinearConflict	13.4928941241
8-puzzle, cornerTiles	12.1894549434
8-puzzle, lastMoves	15.3194222036
8-puzzle, tilesOut	6.59795164045
8-puzzle, cornerDeduction	11.7892095523
8-puzzle, conflicts	12.0827269933
8-puzzle, conflictDeduction	11.9276834312
8-puzzle, lastMovesLinearConflict	13.9097842598
8-puzzle, rawConflicts	11.6338500267
8-puzzle, allCorrections	9.33199427468
8-puzzle, manhattan, depth 2	14.3813038225
8-puzzle, manhattan, depth 4	14.0658052067
8-puzzle, manhattan, depth 6	12.1574327222
8-puzzle, manhattan, depth 8	11.0073859575

Tabella 3.8. Lunghezza di correlazione per diverse euristiche per l'8-puzzle

Il risultato è abbastanza deludente: ci si aspettava una lunghezza di correlazione più elevata per euristiche più informate, aspettativa non confermata dall'analisi eseguita. Questo

risultato può essere dovuto al fatto che l'andamento delle distanze presenta già di per sé una discreta rugosità in un cammino casuale. Per migliorare l'analisi si potrebbe fare in modo di scegliere un campionamento che presenti una variabilità di base meno accentuata; un esempio sarebbe la strada verso la soluzione, che ha un andamento monotono decrescente. In questo caso si perderebbe forse il vantaggio della rapidità di questo tipo di analisi, nella sua formulazione originale completamente priori.

3.2 La matrice di transizione

In questo paragrafo sono descritti alcuni metodi di studio delle funzioni euristiche che si basano sulla cosiddetta matrice di transizione. Questi si basano essenzialmente su una visione dell'evoluzione dell'algoritmo come un processo markoviano la cui evoluzione da uno stato iniziale può essere descritta appunto da una matrice di transizione. È possibile definire un'equazione o un algoritmo che permetta, avendo a disposizione la suddetta matrice, di riprodurre fedelmente il comportamento di un algoritmo durante la ricerca, permettendo così di individuare degli aspetti interessanti della ricerca senza il bisogno di eseguire operativamente la stessa su tutte le istanze del problema.

Il compito è complicato dal fatto che una matrice di transizione completa conterrebbe una riga e una colonna per ogni nodo dello spazio; una matrice di questo tipo sarebbe intrattabile per qualsiasi problema reale, e in ogni caso una procedura basata su essa sarebbe più complessa dell'algoritmo di ricerca che si andrebbe a studiare.

In questo contesto nasce la necessità di adottare una metodologia di clustering che riesca a raggruppare i nodi che hanno un comportamento simile. Una buona clusterizzazione dello spazio sarebbe in grado di preservare l'informazione sull'evoluzione delle transizioni, mantenendo un numero limitato di cluster.

Due sono i metodi possibili per ottenere i valori della matrice di transizione: si può innanzitutto adottare un approccio teorico-matematico, basandosi sull'osservazione del problema e delle possibili transizioni, oppure c'è la possibilità di eseguire un campionamento dello spazio degli stati. Si può anche pensare a una tecnica ibrida che permetta di compensare le lacune dei due metodi. L'approccio teorico è limitato dalla possibilità di trovare facilmente delle condizioni di evoluzione dell'albero di ricerca; d'altra parte il campionamento è notevolmente carente per quanto riguarda i cluster con un numero limitato di nodi, per i quali l'unica strategia che garantisce il calcolo esatto è l'enumerazione esaustiva. È importante limitare gli oneri computazionali del calcolo la matrice di transizione eseguendo un campionamento di un numero il più possibile limitato di nodi dello spazio. La tecnica che si potrebbe adottare consisterebbe nel campionare una porzione dello spazio degli stati, e cercare di riempire il resto della matrice individuando delle regolarità in gruppi di cluster¹.

3.2.1 Calcolo di errore basato su catene di Markov per l'algoritmo Hill-climbing

Attraverso la matrice di transizione è possibile modellare l'algoritmo hill-climbing euristico come un processo stocastico, e derivare interessanti informazioni riguardo al comportamento dell'algoritmo; si vedrà come, attraverso le proprietà delle catene di Markov, si possa fare una stima della probabilità di giungere alla soluzione nel numero esatto di passi

¹ Questa tecnica potrebbe essere adottata ad esempio per l'*N*-puzzle, ma non per il labirinto.

indicati dalla funzione euristica¹.

Il processo di ricerca dell'algoritmo hill-climbing si presta naturalmente ad essere modellato per mezzo delle catene di Markov grazie alla sua caratteristica di essere senza memoria (questa proprietà è detta anche *Markoviana*): ad ogni passo ci si dimentica di tutto e si sceglie come prossimo nodo un nodo a caso tra quelli che hanno tra tutti il valore euristico inferiore tra i figli del nodo attuale (che rappresenta da solo anche lo stato corrente).

3.2.1.1 Concetti base sulle catene di Markov

Considerando i successivi stati X_i di una catena di Markov a tempo discreto possiamo definire la proprietà Markoviana (rif. ad esempio [stewart94]):

$$P(X_{n+1}=x_{n+1}/X_0=x_0, X_1=x_1, \dots, X_n=x_n) = P(X_{n+1}=x_{n+1}/X_n=x_n) \quad (3.5)$$

dove $P(X_{n+1}=x_{n+1}/X_n=x_n)$ è data dalle probabilità di transizione di un singolo step, ossia la probabilità di avere una transizione dallo stato x_n allo stato x_{n+1} . In seguito si userà la notazione $p_{ij} = P(X_{n+1}=j/X_n=i)$ per indicare la probabilità di transitare da uno stato i a uno stato j generici in un solo passo. La matrice P formata ponendo i vari p_{ij} in riga i e colonna j è la matrice di transizione; la detta matrice ha la proprietà di essere stocastica se sono soddisfatte tre proprietà:

1. $p_{ij} \geq 0 \forall i, j$
2. $\sum_i p_{ij}$
3. nessuna colonna contiene solo zeri.

3.2.1.2 Definizione degli stati Markoviani – clusterizzazione dello spazio

La maniera più banale di definire lo stato markoviano sarebbe considerare come stato il singolo nodo attraversato dall'algoritmo di ricerca. Una definizione di questo tipo, seppur corretta dal punto di vista formale, non sarebbe di alcuna utilità: in presenza di N stati si deve infatti costruire una matrice $N \times N$. Questo numero, se consideriamo lo spazio degli stati² di un problema di ricerca non banale, è estremamente elevato, per cui è necessario definire lo stato Markoviano in modo da ridurre la matrice di transizione a dimensioni maneggevoli. Si deve cercare di raggruppare i nodi che verosimilmente si comportano in maniera simile quando siano attraversati dall'algoritmo, nel caso specifico Hill-climbing euristico. Si utilizzerà per le sperimentazioni quella che abbiamo nominato clusterizzazione *tau*³: in breve un cluster *tau* è caratterizzato da una tupla formata da valore euristico, branching factor, e numero di figli con valore euristico inferiore al padre.

3.2.1.3 Descrizione dell'algoritmo Hill-Climbing con le catene di Markov

Data la matrice delle probabilità di transizione di un singolo passo le equazioni di Chapman-Kolmogorov permettono di calcolare la probabilità al passo n -esimo. La matrice $P^{(n)}$, il cui elemento $p_{ij}^{(n)} = P(X_{m+n}=j/X_m=i)$, è così definita, per una catena di Markov

1 Tutto il lavoro relativo al metodo basato sulle catene di Markov è da attribuirsi a Carlos Linares.
 2 Attenzione a non confondere lo stato di un problema di ricerca (che all'interno del grafo indichiamo anche col termine nodo) con lo stato Markoviano
 3 Vedi appendice B.

discreta (eq di Chapman-Kolmogorov):

$$P^{(n)} = PP^{(n-1)} = P^{(n-1)}P \quad (3.6)$$

Da qui si ricava un importante risultato per quanto riguarda lo studio delle funzioni euristiche: si contrassegna con 0 lo stato Markoviano corrispondente alla soluzione, allora p_{i0}^n sarà la probabilità di trovare la soluzione in n passi, partendo dallo stato i .

Se la funzione euristica è perfettamente informata ovviamente si avrà un valore pari a 1 per p_{x0}^x , dove x denota tutti gli stati Markoviani con valore euristico x . D'altra parte, più piccolo sarà questo valore, maggiore sarà la probabilità di incontrare un errore lungo il cammino. In questo caso p_{x0}^x denota la probabilità di non commettere errori lungo il cammino da un nodo con un dato valore euristico al goal. Questo valore può essere dunque considerato come una misura pessimistica dell'accuratezza della funzione euristica.

Un altro interessante risultato può essere ottenuto attraverso il cosiddetto primo passaggio medio M_{ij} (mean first passage time), ossia il numero di passi da effettuare mediamente per arrivare allo stato j partendo dallo stato i della catena di Markov. Questo può essere ottenuto da

$$M_{ij} = \sum_{n=1}^{\infty} n f_{ij}^{(n)}, \quad (3.7)$$

dove $f_{ij}^{(n)}$ è la probabilità di raggiungere lo stato j partendo dallo stato i in esattamente n passi:

$$f_{ij}^n = p_{ij}^n - \sum_{l=1}^{n-1} f_{ij}^{(l)} p_{jj}^{(n-l)}. \quad (3.8)$$

Ovviamente $f_{ij}^{(1)} = p_{ij}$.

M_{i0} sarà la lunghezza media della soluzione partendo dallo stato Markoviano i .

3.2.1.4 Risultati sperimentali

In tabella 3.9 è presentato il risultato per quanto riguarda la probabilità di raggiungere la soluzione in un numero di passi pari al valore euristico (euristica manhattan), per un campionamento totale dello spazio degli stati per il 7-puzzle e 8-puzzle, e il raffronto con dei valori osservati sperimentalmente. Si può notare una precisione assoluta decisamente notevole, in particolare per quanto riguarda i nodi più vicini alla soluzione.

h	2x4		3x3	
	Obs. \bar{p}_x	Pred. \hat{p}_x	Obs. \bar{p}_x	Pred. \hat{p}_x
0	1	1	1	1
1	1	1	1	1
2	1	1	1	1
3	0.75	0.75	0.8	0.8
4	0.142857	0.142857	0.13913	0.13913
5	0.081761	0.077268	0.073170	0.068879
6	0.044025	0.038978	0.041726	0.036011
7	0.028130	0.026218	0.030864	0.025866
8	0.016414	0.013524	0.015230	0.011262
9	0.010649	0.008110	0.010228	0.007179
10	0.007405	0.004935	0.006235	0.003555
11	0.005022	0.002939	0.004706	0.002504
12	0.003513	0.001775	0.002991	0.001266
13	0.002069	0.001016	0.002023	0.000873
14	0.001298	0.000600	0.001432	0.000470
15	0.000712	0.000325	0.001081	0.000319
16	0.000254	0.000178	0.000912	0.000178
17	9.6×10^{-5}	9.8×10^{-5}	0.000765	0.000119
18	3.1×10^{-5}	5.0×10^{-5}	0.000821	7.0×10^{-5}
19	3.2×10^{-5}	2.8×10^{-5}	0.000790	4.4×10^{-5}
20	0	1.5×10^{-5}	0.001358	2.9×10^{-5}
21	0	1.1×10^{-5}	0.001213	1.6×10^{-5}
22	-	-	0.001469	1.5×10^{-5}

Tabella 3.9. Probabilità di raggiungere la soluzione nel numero di passi predetto dall'euristica per 7 e 8-puzzle

I valori sono così ottenuti:

$$\bar{p}_x = \frac{1}{N_x} \sum_{\forall n, h(n)=x} \bar{p}(n),$$

dove N_x è il numero di nodi con $h(n)=x$, e $\bar{p}(n) = \frac{1}{c(n)} \sum_{i=1}^c (n) \bar{p}_i(n_i), n_i \in SCS(n)$ ($c(n)$ è il numero di figli con valore euristico inferiore al padre). Il valore osservato \hat{p}_x è invece ottenuto attraverso il teorema della probabilità totale sugli stati markoviani con medesimo valore euristico.

In figura 3.23 è invece visualizzato l'andamento della lunghezza media della soluzione per 7, 8, 9-puzzle. È decisamente difficile dare una controprova sperimentale attendibile di questo risultato, in quanto essendo non deterministica la ricerca hill-climbing sarebbe necessario effettuare una numerosità significativa di ricerche per ogni nodo dello spazio degli stati. D'altra parte l'andamento segue un percorso abbastanza ragionevole, mostrando un deciso calo

di prestazioni per il valore euristico 4, come ci si poteva aspettare dato il noto brusco calo di informatività di manhattan.

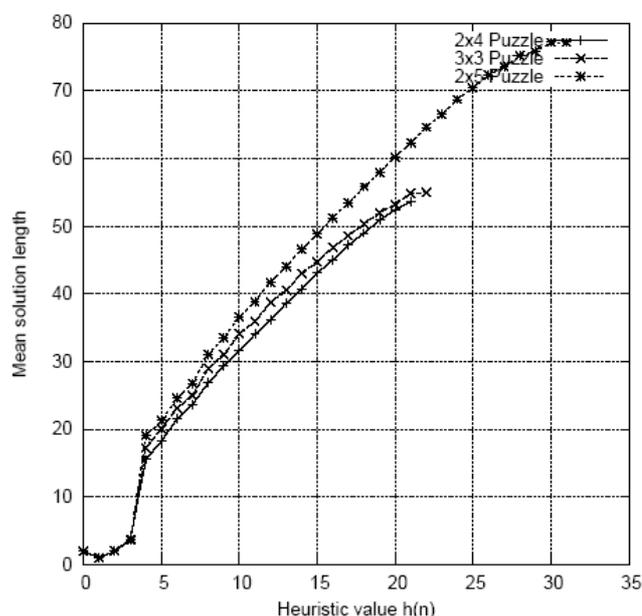


Figura 3.20.: distanza media della soluzione per l'algoritmo hill-climbing

3.2.2 Metodo per la stima dei nodi generati da IDA* e profondità della soluzione

Si presenta ora un metodo innovativo per la stima a priori dei nodi espansi o generati dall'algoritmo IDA*, basato su matrice delle transizioni. Questo metodo permette di ottenere informazioni aggiuntive rispetto al metodo di Korf-Reid, in quanto è possibile calcolare i nodi espansi partendo da un nodo di un determinato cluster, è applicabile a euristiche non consistenti e non ammissibili, e fornisce una stima della profondità della soluzione. Di contro l'applicazione è relativamente più complessa.

Il metodo si basa su una particolare matrice di transizione T , nella quale l'elemento T_{ij} è il numero di nodi espansi da un nodo di cluster j che appartengono al cluster i . Chi conosce la teoria dei grafi avrà riconosciuto la somiglianza con la comune matrice delle adiacenze: in effetti il metodo si basa su una rappresentazione sintetizzata del grafo di ricerca (sintesi ottenibile mediante i cluster), che in assonanza col regular tree di Korf potremmo chiamare *regular graph*. Per mezzo di questo grafo si può realizzare una vera e propria simulazione del processo di ricerca per ottenere le più svariate informazioni sullo stesso. Si applica a problemi in cui l'edge-cost sia costante e unitario.

Per la spiegazione del metodo ci si aiuterà con un semplice esempio, nel quale il cluster è connotato semplicemente dal valore euristico, e ogni nodo espanda un nodo di valore euristico inferiore, uno superiore e uno di medesimo valore, quando sia possibile senza uscire dai limiti. La matrice di transizione corrispondente per una piccola istanza sarebbe

1	1	0	0	0
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	0	1	1

Figura 3.21.: Semplice esempio di matrice di transizione

Dovrebbe essere già intuibile osservando la figura come il numero dei nodi espansi al primo livello dell'albero da un nodo di euristica x sia la somma dei nodi nella riga x della matrice. Per l'esempio considerato al primo livello dell'albero saranno espansi tre nodi di valore euristico 2, come nell'esempio in figura 3.22.

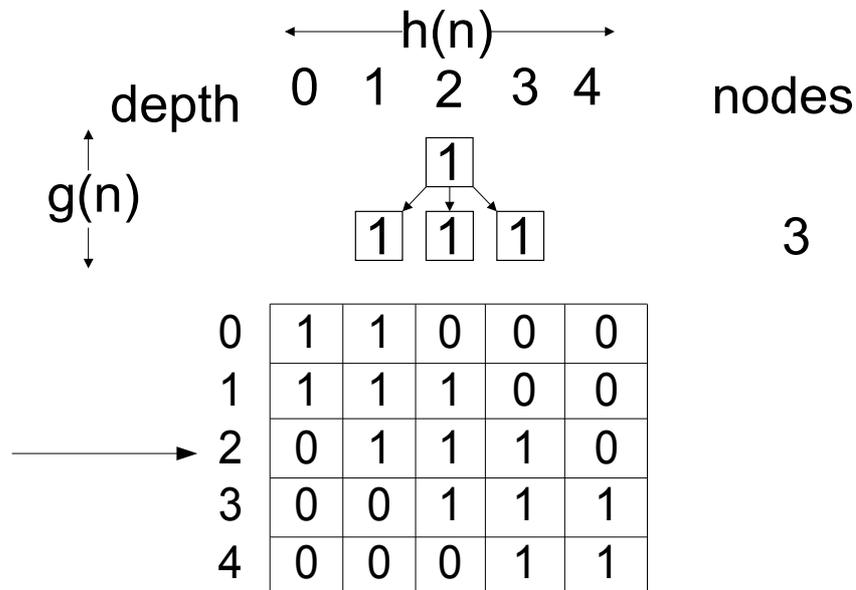


Figura 3.22.: nodi espansi al livello 1 da un nodo di valore euristico 2.

Il passo successivo è considerare il numero di nodi espansi al livello n -esimo dell'albero. La transizione al livello n -esimo è rappresentata semplicemente dalla matrice T^n : in posizione T^n_{ij} ci sarà infatti il numero di nodi appartenenti alla classe i generati in una ricerca che parta da un nodo di classe j . In figura 3.23 è presentato l'andamento dell'albero-grafo a livello 2, e come questo sia rispecchiato in T^2 .

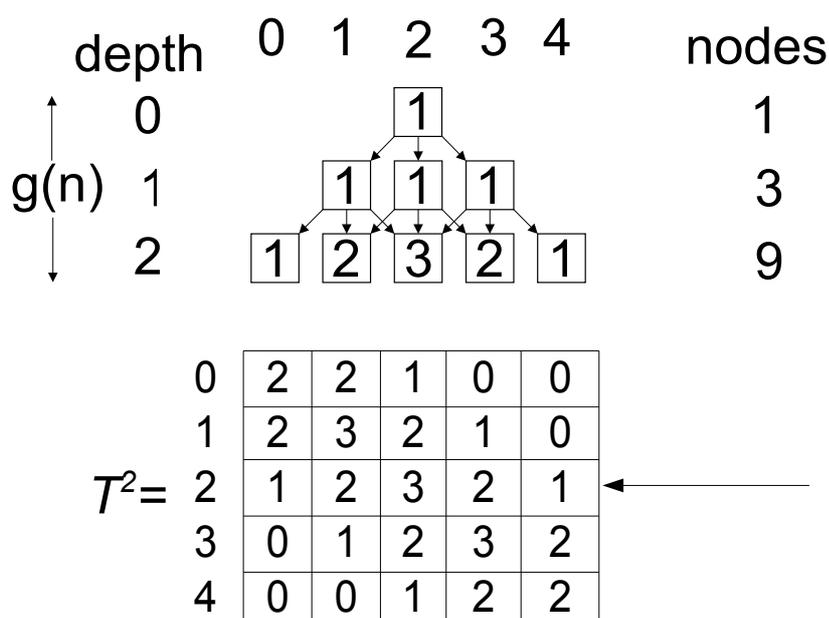


Figura 3.23.: esempio: nodi espansi al livello 2 da un nodo di valore euristico 2.

A questo punto si potrà accettare il risultato generale senza ulteriore dimostrazione:

Teorema: nodi espansi da un algoritmo brute force: data una matrice delle transizioni T in cui nell'elemento T_{ij} sia indicato il numero medio di nodi appartenenti alla classe j generati da un cluster i , nell'ipotesi di clusterizzazione perfetta¹ il numero di nodi espansi a profondità d partendo da un nodo di cluster i da un algoritmo brute force sistematico sarà nel caso peggiore esattamente pari a:

$$N(i, d) = \sum_{k=0}^d \sum_j T_{ij}^k \quad (3.9)$$

Per ottenere il numero di nodi espansi dall'algoritmo IDA* nell'ultima iterazione è sufficiente considerare nella somma soltanto le colonne che rappresentano i nodi che rispettino la condizione $h + g < l$. Questo può essere ottenuto usando nella moltiplicazione degli step intermedi la matrice T modificata eliminando le colonne che non soddisfino la suddetta condizione.

Definizione: Sia la matrice $T^{(k)}$ la matrice definita a partire dalla matrice T in cui

$$\left\{ \begin{array}{l} T_{ij}^{(k)} = T_{ij} \text{ if } h(j) + k < l \\ T_{ij}^{(k)} = 0 \text{ otherwise} \end{array} \right\} \quad (3.10)$$

Sia dunque $T(k)$ la matrice definita ricorsivamente come:

$$\left\{ \begin{array}{l} T(1) = T \\ T(k) = T(k-1)T^{(k)} \end{array} \right\} \quad (3.11)$$

¹ Vedi appendice B

Il numero di nodi espansi dall' algoritmo IDA* in una ricerca a limite l sarà, nelle stesse ipotesi della 3.9, aggiungendo che l'edge-cost sia costante e unitario:

$$N_{IDA}(i, l) = \sum_{k=1}^l \sum_j T_{ij}(k) \quad (3.12)$$

L'aggiornamento della matrice è realizzabile semplicemente nella procedura iterativa di calcolo.

In figura 3.24 sono indicati in verde i nodi espansi da una ricerca IDA* con limite 6, in marroncino i nodi generati ma non espansi. Nella transizione da profondità 2 a 3 i nodi di euristica 3 e 4 non genereranno nessun nodo di euristica 4. Questo equivale a eliminare la colonna 4 da T alla terza iterazione dell'algoritmo.

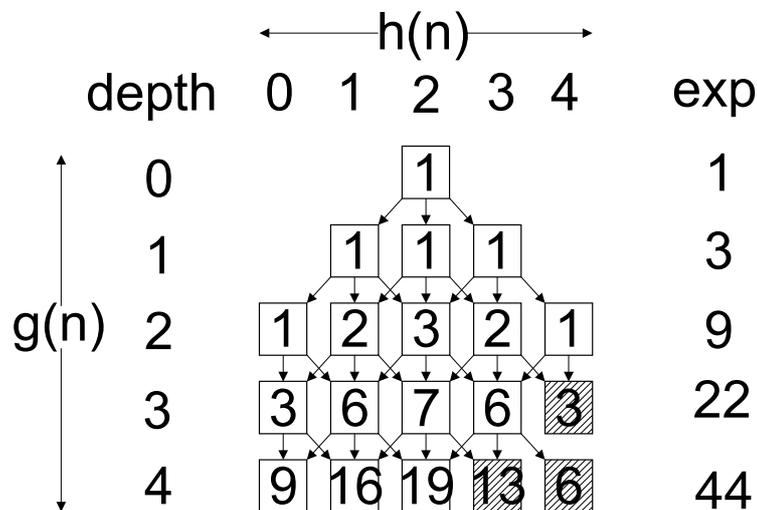


Figura 3.24.: nodi espansi da IDA* con limite

Questo tipo di aggiornamento rende possibile utilizzare il metodo proposto anche quando l'euristica sia non consistente e dunque non ammissibile. Quello che infatti limita il metodo di Korf-Reid alle sole euristiche esclusivamente consistenti è che quando la condizione di consistenza non sia verificata un nodo non espanso dalla procedura IDA* potrebbe generare nell'albero brute-force dei nodi che non violerebbero la condizione di limite. Questi nodi sarebbero contati dal metodo di Korf-Reid, ma non sono generati dall'algoritmo. In figura 3.25 è mostrato un esempio di matrice di transizione per un'euristica non consistente.

1	1	0	0	0
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	1

Figura 3.25.: Esempio di matrice di transizione per un'euristica non consistente

La proprietà di consistenza è violata nella riga 5 ($h=4$), per il fatto che un nodo a valore euristico 4 genera un nodo a valore euristico 2 in un passo (è violata la (1.11)) essendo $h(n)=4$, $h(n')=2$ e $k(n, n')=1$). I tre nodi generati ma non espansi a profondità 3 in figura 3.24 a valore euristico 4 generano in questo caso altrettanti tre nodi a profondità 4 nella ricerca brute-force, che IDA* invece non espande e dunque non devono essere contati. L'uso della matrice modificata $T^{(k)}$ previene dal contare questi nodi, rendendo dunque il metodo applicabile anche nel caso di euristiche non consistenti. Questo allarga lo spettro anche alle euristiche non ammissibili, venendo meno la preconditione del teorema 1.2.

3.2.2.1 Stima della profondità media della soluzione

Si è visto come uno dei principali limiti del metodo di Korf-Reid sia quello di fornire la stima dei nodi mediamente espansi a profondità d , senza però fornire un'indicazione sul reale valore che assumerà in media il parametro succitato.

La possibilità di stimare la profondità della soluzione per mezzo del metodo presentato si basa su una semplicissima considerazione: avendo a disposizione un cluster formato solo e soltanto da nodi soluzione (che si suppone stiano nella riga/colonna 0 della matrice di transizione) il numero di nodi soluzione mediamente espansi a profondità k in una ricerca che parta da un nodo di cluster i sarà $T_{i0}(k)$. Si può ragionevolmente supporre, per un algoritmo di ricerca sistematica, che la soluzione sia nel caso migliore trovata quando questo numero sia superiore a zero. Il valore di k per il quale si verifica questa condizione sarà una stima della profondità media della soluzione per una ricerca che parta dal cluster i .

Questa tecnica può essere usata come condizione di stop per l'algoritmo di calcolo dei nodi, fornendo realmente una stima dei nodi espansi durante la ricerca.

3.2.2.2 Applicazione stocastica del metodo

Il metodo, per come descritto fino ad ora, senza fare alcuna approssimazione, non potrebbe essere certamente di grande utilità: infatti risulta impossibile individuare tutte le diverse classi di metodi che abbiano delle transizioni completamente equivalenti; sarà necessario adottare una clusterizzazione sub-ottima, e fare delle assunzioni di tipo statistico che permettano di calcolare i nodi mediamente espansi dai nodi appartenenti a un determinato cluster, e la distanza media dalla soluzione.

La soluzione proposta è la seguente: una volta determinata la clusterizzazione si pone nella matrice il numero medio di nodi espansi nel cluster di arrivo. Il metodo è in grado in questa condizione di approssimare il numero medio di nodi espansi.

Per quanto riguarda la distanza dalla soluzione, questa si considera mediamente individuata quando il numero medio di nodi soluzione generati sia pari o superiore a 1.

3.2.2.3 Risultati sperimentali

Il metodo descritto, come qualsiasi metodo basato su matrice di transizione, ha nella fase applicativa i punti di maggiore difficoltà. Non tanto per l'algoritmo in sé, la cui implementazione è da considerarsi discretamente semplice avendo a disposizione uno strumento per maneggiare il calcolo matriciale, quanto per l'ottenimento di una buona matrice rappresentativa del problema. Per gli esperimenti ci si è limitati all'8-puzzle, e nonostante siano ancora in fase di definizione sono riusciti a fornire dei buoni risultati.

La clusterizzazione usata è stata semplicemente il valore euristico, e la matrice è stata calcolata nel seguente modo: è stato campionato lo spazio degli stati per mezzo di un campionamento tree-sample a profondità 9 e 10, e per ogni campione è stata aggiunto alla matrice nella posizione relativa il numero di figli con dato valore euristico. La matrice è stata dunque normalizzata (ogni riga somma uno) e moltiplicata per il branching factor medio.

In figura 3.26 si può vedere il risultato del numero totale medio di nodi espansi. Per avere il numero totale si è eseguita una convoluzione con la distribuzione di probabilità per ottenere la media per i diversi valori euristici.

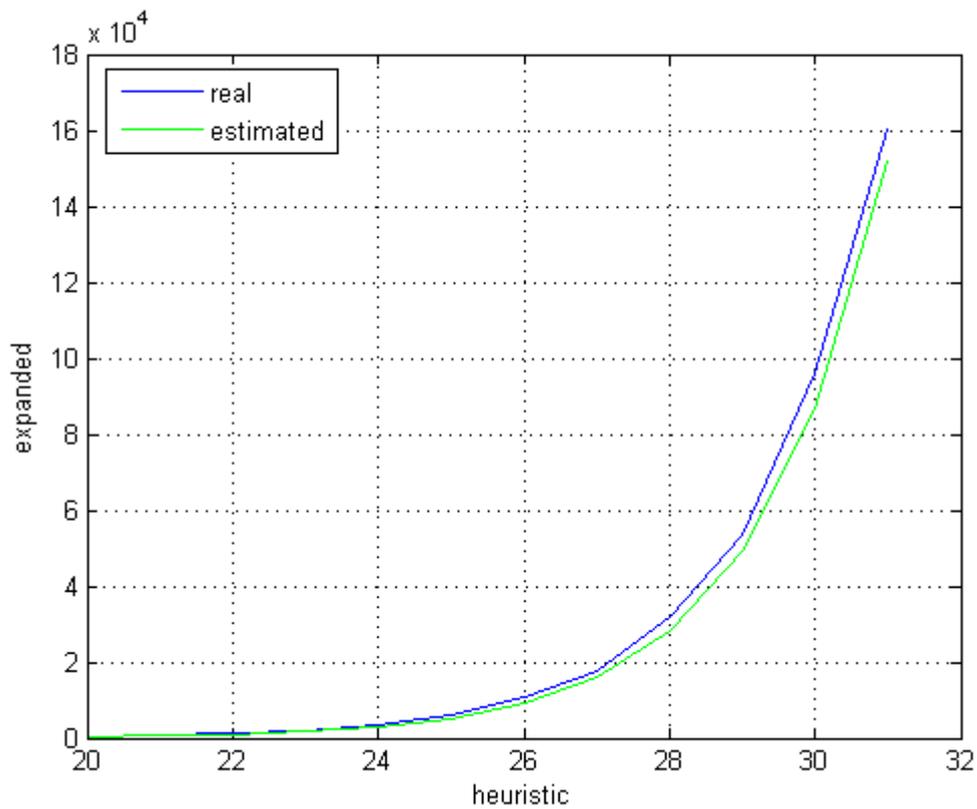


Figura 3.26.: Nodi espansi mediamente per l'8-puzzle; valore stimato e reale

Il risultato è una buona approssimazione della curva, se si tiene anche in conto il fatto che il metodo calcola di base i nodi espansi separatamente per ogni cluster. Di seguito la stima della distanza media per la stessa matrice:

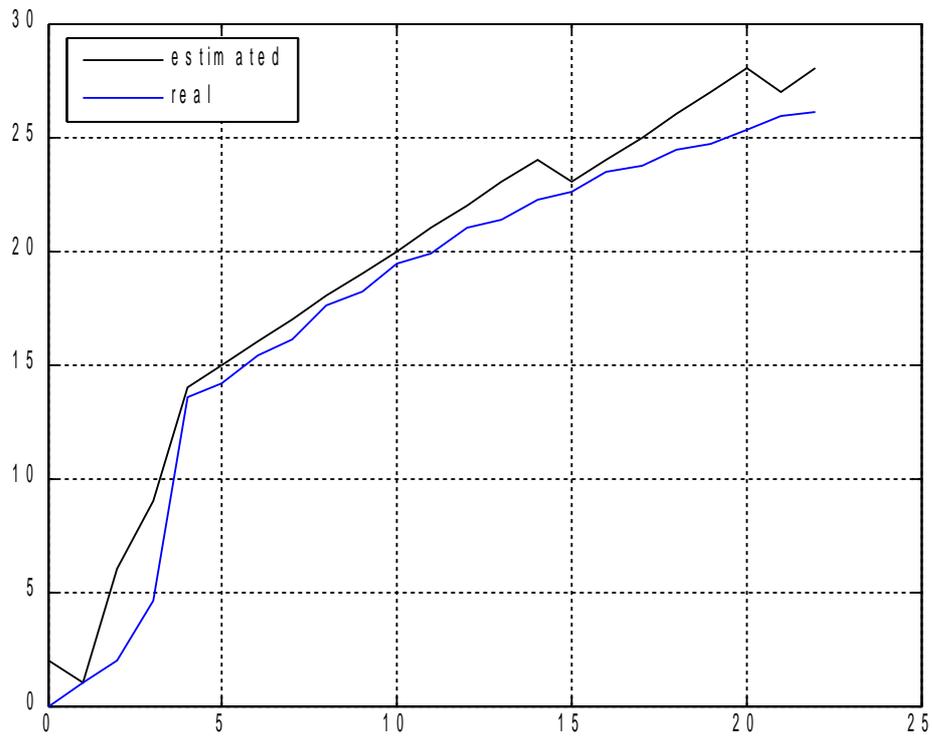


Figura 3.27.: Distanza media della soluzione stimata – 8-puzzle

I risultati sono certamente incoraggianti, ma resta ancora molto lavoro da svolgere: è stato dovuto usare un gran numero di campioni, e il campionamento è stato dovuto realizzare per tentativi. Si potrebbe, per migliorare la precisione, formare la matrice con delle tecniche ibride in modo da determinare con maggiore le transizioni dei cluster meno rappresentati.

4 CONCLUSIONI E SVILUPPI FUTURI

Questa tesi contiene uno studio sulle modalità di analisi e valutazione di funzioni euristiche, con la finalità di rilevare la loro efficacia nell'indirizzare un algoritmo di ricerca verso la soluzione di un problema.

Studiare il comportamento delle funzioni euristiche comporta la necessità di porsi in un contesto situato tra il ragionamento rigoroso e l'intuizione. Il tentativo fatto in questa sede è quello di mettere il primo al servizio della seconda, piuttosto che ricorrere all'approccio inverso solitamente usato in questo tipo di analisi. Tale compito è stato reso particolarmente difficile da diversi aspetti correlati, tra i quali i motivi noti sulla difficoltà nello stimare la distribuzione dell'errore, e per l'enorme impatto delle diverse implementazioni di algoritmi ed euristiche sull'efficienza della ricerca.

La finalità principale per gli studi in questo ambito è quella di mettere a disposizione del ricercatore metriche capaci di determinare automaticamente quale fra una collezione di euristiche sia migliore delle altre in relazione a uno specifico problema o tipologia di problemi. Oltre a fornire indicazioni utili per la finalità di cui sopra, tali metriche possono rivelarsi vantaggiose anche per comprendere i punti di debolezza di euristiche esistenti, fornendo così lo spunto per definire euristiche maggiormente informate o algoritmi in grado di rilevare le situazioni in cui l'euristica associata diventa localmente meno informativa (plateau).

Dopo averli brevemente introdotti, nel corso della tesi sono stati studiati numerosi

algoritmi di ricerca e sono state definite tecniche innovative per valutare la bontà delle funzioni euristiche utilizzate per esplorare lo spazio di ricerca associato a un'istanza di problema. Lo studio è stato affiancato da un corposo corredo di risultati sperimentali, con l'obiettivo di dare anche al lettore la possibilità di fornire una propria interpretazione, stimolando nel contempo la possibilità da parte sua di proporre qualche nuova idea o miglioramento.

Più che voler trarre conclusioni definitive sull'argomento, i risultati esposti in questa tesi potrebbero costituire un punto di partenza per future applicazioni delle tecniche proposte. Sebbene ognuna di esse possa già costituire un utile supporto all'analisi, sicuramente ancora molti passi avanti dovranno essere fatti affinché si possa avere uno strumento automatico e affidabile di valutazione delle funzioni euristiche. Nonostante il fatto che lo stato dell'arte non sia ancora in grado di dare una risposta sufficientemente generale a questo problema, la speranza è che comunque questa tesi abbia dato un piccolo contributo in tale direzione.

Il lavoro futuro potrebbe essere rivolto ad aspetti sia teorici sia pratici. In particolare si potrebbe investigare la possibilità di ottenere euristiche più informate correggendo localmente il valore di euristiche note sfruttando alcune delle tecniche proposte, ma anche sperimentare su varie tipologie di problemi noti (e significativi per la loro intrinseca complessità) la bontà delle metriche definite e studiate nell'ambito di questa tesi. Resta comunque inteso che lo sbocco più naturale per le tecniche indagate sarà quello di poter selezionare euristiche da un insieme generato automaticamente (ad esempio sfruttando metodi di look-ahead o di perimetro parametrici oppure ricorrendo a generatori automatici di problemi rilassati).

APPENDICE A: CAMPIONAMENTO DELLO SPAZIO DEGLI STATI

Come si è visto il campionamento dello spazio degli stati è una fase fondamentale nell'analisi delle funzioni euristiche, in particolare ha un'importanza cruciale in presenza di spazi di dimensioni tanto elevate da non permettere l'enumerazione esaustiva. Qui di seguito si propone una classificazione e un'implementazione di diverse possibili tecniche di campionamento.

- *Enumerate sample*: sono campionati una e una sola volta tutti i nodi dello spazio degli stati. L'operazione non è in realtà banale come potrebbe apparire: è necessario trovare un algoritmo che fornisca in sequenza tutte le possibili combinazioni risolubili, perché appartenenti tutte allo stesso insieme connesso della soluzione. Nel caso del N-puzzle è sufficiente calcolare tutte le permutazioni di $N+1$ numeri da 0 a N ed eliminare dalla collezione le istanze non risolubili mediante il test di parità (formula 1.1).
- *Random sample*: si tratta di restituire un campione preso casualmente dallo spazio degli stati. Si è usata nel framework sperimentale la funzione shuffle del modulo random di Python, che restituisce direttamente una permutazione casuale di una lista secondo una distribuzione uniforme. Viene accettata la prima combinazione che non fallisce il test 1.1. Questo tipo di campionamento, sebbene statisticamente il più

sensato, ha il difetto di non riuscire a cogliere categorie di campioni meno rappresentate (ad esempio i nodi con piccolo valore euristico per l'*N*-puzzle).

- *Tree sample*: si campionano i nodi a profondità determinata d , in M alberi costruiti intorno a M campioni casuali. Un campionamento di questo tipo potrebbe cogliere la distribuzione dei nodi in profondità nell'albero di ricerca.
- *Enhanced sample*: più che un campionamento è un'aggiunta a qualsiasi altro campionamento: si aggiungono i nodi a partire da alberi costruiti a partire al più piccolo e/o al più grande valore campionato in precedenza, per cercare di ovviare alla lacuna di nodi per valori euristici alti e bassi, solitamente statisticamente meno rappresentati.
- *Random Walk sample*: si campionano i nodi generati in un cammino casuale (un cammino guidato dall'applicazione di operatori in cui passo per passo è scelto un figlio a caso).
- *Random Walk pick sample*: viene preso solo l'ultimo nodo di un cammino casuale. È pensato per rispecchiare la distribuzione all'equilibrio.

Un altro aspetto con cui ci si scontra nel momento di campionare spazi degli stati di dimensioni elevate è il fatto che i campioni possono rapidamente occupare tutta la memoria disponibile. Per questo motivo è necessario utilizzare delle strutture di tipo stream in grado di iterare sui campioni senza mantenerli in memoria, ossia generandoli o caricandoli in tempo reale. Questo è possibile memorizzando preventivamente i campioni in file, oppure controllando la generazione pseudocasuale tenendo memoria del seme utilizzato, in modo da poter riprodurre lo stesso campionamento in iterazioni consecutive.

APPENDICE B:

CLUSTERIZZAZIONE DELLO

SPAZIO DEGLI STATI

Clusterizzare significa individuare una suddivisione dello spazio in classi il più possibile omogenee tra loro in base a determinati criteri. Nello specifico ci interessa suddividere i nodi dello spazio degli stati in cluster omogenei dal punto di vista del comportamento della ricerca e della funzione euristica.

In un problema di clusterizzazione è fondamentale il compromesso tra precisione e sinteticità: si vogliono cogliere il numero più possibile elevato di elementi simili, e al contempo mantenere un numero di classi che riduca le dimensioni dello spazio degli stati.

La clusterizzazione più ovvia e sintetica da questo punto di vista è quella basata sul valore euristico: i nodi con uguale valore euristico spesso hanno anche distanza dalla soluzione simile, ma molto meno di quanto potrebbe apparire a prima vista, e molti altri particolari non vengono colti.

Korf nel calcolare la equilibrium distribution [KorfReidEdelkamp2001] ha proposto un raggruppamento dei nodi basato sul valore euristico e sul branching factor. Considerare il branching factor oltre che il solo valore euristico permette di fare un'importante distinzione tra nodi che corrispondono a situazioni simili nella ricerca. Ad esempio i nodi del N-Puzzle vengono divisi in base al branching factor in tre categorie fundamentalmente distinte: nodi

all'angolo (corner, $b=1$), nodi laterali (side, $b=2$) e centrali ($b=3$)¹. Sempre in [KorfReidEdelkamp2001] è dimostrato come la probabilità di incontrare uno dei tre tipi di nodi sia differente dal rapporto delle loro numerosità (vedi la sezione sul metodo di Korf-Reid, 2.3.4).

Un ulteriore affinamento di questa suddivisione è quella che chiamiamo clusterizzazione tau: a euristica e branching factor si aggiunge il numero di figli con valore euristico inferiore al padre: questo permette di distinguere i nodi più o meno promettenti di avere un valore euristico accurato. Il parametro è stato identificato con c , e dunque un cluster con la tripla $\tau(h, b, c)$. questa clusterizzazione è stata usata per la matrice di transizione per il metodo di calcolo d'errore per l'algoritmo Hill-Climbing basato su catene di Markov.

In questo documento si è fatto inoltre riferimento al concetto di clusterizzazione *perfetta*, nel senso di una classificazione che riesca a cogliere tutti i fenomeni di interesse: una clusterizzazione perfetta suddividerà perfettamente lo spazio degli stati in base al comportamento, ad esempio nella ricerca. Dal punto di vista della ricerca potrebbero essere considerati perfettamente equivalenti dei nodi che abbiano lo stesso valore euristico, lo stesso numero di figli e le cui transizioni portino a nodi anch'essi equivalenti.

Questi semplici esempi non esauriscono assolutamente l'argomento della clusterizzazione, sul quale molto lavoro ancora può essere fatto; molto materiale potrebbe essere riadattato da tecniche classiche e generali di clusterizzazione.

¹ I branching factor indicati considerano il fatto che non possa essere generato il genitore di un nodo.

APPENDICE C: RISULTATI SPERIMENTALI: TEMPO E NODI ESPANSI

In questa sezione sono riportati alcuni valori calcolati empiricamente di tempo e nodi espansi: sono state risolte diverse istanze dei diversi problemi utilizzati nelle sperimentazioni all'interno della tesi, utilizzando l'algoritmo IDA*. L'analisi di questi dati sperimentali può essere utile a confrontare le analisi predette dai vari metodi studiati con le prestazioni di una ricerca reale. Per questione di tempo sono state risolte totalmente solo le istanze del N-puzzle fino all'8. Per quanto riguarda il 15-puzzle sono state considerate le prime venti (le più semplici) istanze della suite di 100 esempi utilizzata da Korf in [Korf1985].

Si è utilizzata l'applicazione Python descritta al nella sezione 2.1.1, su un laptop equipaggiato con processore Pentium M a 1.5 Ghz, e con sistema operativo Windows XP H.E.. I dati sui tempi devono essere presi con la dovuta cautela trattandosi di un singolo tentativo; si considerino le misure con una tolleranza del 10-20%¹.

Si può notare come il comportamento della stessa euristica varia per le diverse dimensioni del puzzle; le euristiche più informate portano a un vantaggio maggiore per le istanze più difficili. Il tempo scende addirittura di 10 volte per il sample set del 15-puzzle utilizzando

¹ In particolare gli esperimenti più lunghi risentono di un utilizzo della macchina non completamente dedicato all'applicazione di test.

Appendice C: Risultati sperimentali: tempo e nodi espansi

conflict-deduction piuttosto che manhattan, e i nodi espansi sono 22 volte in meno. Sorprendono inoltre le prestazioni dell'euristica all-corrections, nel quale tutte le correzioni sono sommate non badando a possibili sovrapposizioni: i nodi espansi sono quasi sempre a favore della somma non-ammissibile, confermando che le non ammissibilità sono sì da considerarsi errori, ma accettare questi errori possono essere accettati quando necessari a consentire di aggiungere maggiore informatività alla ricerca e/o a migliorare l'efficienza di calcolo dell'euristica. Lo stesso si può affermare dell'euristica raw-conflicts rispetto a conflicts.

L'euristica tiles-out è stata esclusa dalla maggioranza degli esperimenti per via del tempo eccessivo necessario a completare l'operazione di ricerca con questa euristica decisamente inadeguata.

heuristic	Time	Avg. h	Avg. d.	Gen.	Exp.	Exp/s
manhattan	4.02	6.833	12.622	62689	46232	11513
linearConflict	3.22	7.717	12.622	45452	33270	10348
nonLinearConflict	3.82	7.211	12.622	54291	39448	10337
cornerTiles	2.73	7.744	12.622	41084	30147	11047
lastMoves	3.19	7.494	12.622	48898	35702	11177
tilesOut	9.05	4.167	12.622	140336	102596	11332
cornerDeduction	3.36	7.883	12.689	45889	33665	10029
conflicts	3.26	8.000	12.622	39135	28110	8613
conflictDeduction	2.80	8.567	12.622	29829	21672	7735
lastMovesLinearConflict	2.71	8.378	12.633	35934	26074	9610
rawConflicts	3.08	8.094	12.622	37850	27129	8813
allCorrections	2.89	9.806	13.344	31962	22938	7939

Tabella C.1. Prestazioni sperimentali 5-Puzzle con diverse euristiche

heuristic	Time	Avg. h	avg. d	Gen.	Exp.	exp/s
manhattan	6642	12.0	22.678	109577531	73283624	11033
linearConflict	3762	13.693	22.678	53294813	35477846	9430
nonLinearConflict	6445	12.375	22.678	91667397	60843834	9440
cornerTiles	3748	13.058	22.678	58229389	39283533	10478
lastMoves	4999	12.750	22.678	78798758	52255978	10452
cornerDeduction	3893	13.155	22.678	54218103	36671904	9419
conflicts	3966	13.830	22.678	44262069	29355828	7400
conflictDeduction	3546	14.464	22.678	30415860	20258923	5712
lastMovesLinearConflict	3672	14.443	22.690	39209437	25934562	7062
rawConflicts	4608	14.068	22.678	41347536	27314844	5926
allCorrections	2368	15.973	23.068	20973449	13926189	5878

Tabella C.2. Prestazioni sperimentali 7-Puzzle con diverse euristiche

Appendice C: Risultati sperimentali: tempo e nodi espansi

heuristic	Time	Avg. h	Avg. d	Gen.	Exp.	Exp/s
manhattan	40448	14.0	21.972	625552787	376292322	9303
linearConflict	21386	15.119	21.972	300330877	180344950	8432
nonLinearConflict	34555	14.403	21.972	486298210	287160366	8310
cornerTiles	22511	14.706	21.972	365530861	218421049	9702
lastMoves	24647	14.889	21.972	388993938	228986675	9290
cornerDeduction	22335	14.774	21.972	340415289	202443331	9063
conflicts	20503	15.422	21.972	228356390	134273213	6549
conflictDeduction	13778	16.213	21.972	143364370	83325431	6047
lastMovesLinearConflict	14207	16.008	21.983	194784698	114687779	8072
rawConflicts	20297	15.522	21.975	219399409	128752567	6343
allCorrections	16685	17.185	22.485	141214119	81491586	4883

Tabella C.3. Prestazioni sperimentali 8-puzzle con diverse euristiche

heuristic	Time	Avg. h	Avg. d	Gen.	Exp.	Exp/s
manhattan	4449	18.395	34.015	72162726	45649743	10258
linearConflict	1400	20.865	34.015	18970136	11903613	8502
nonLinearConflict	4204	18.775	34.015	57981191	36443424	8667
cornerTiles	2342	19.275	34.015	35831886	22971810	9807
lastMoves	3174	19.215	34.015	49161271	30862555	9723
cornerDeduction	2410	19.325	34.015	33646418	21650324	8983
conflicts	1468	20.855	34.015	16036654	10073452	6861
conflictDeduction	1069	21.435	34.015	10458564	6594583	6167
lastMovesLinearConflict	1068	21.685	34.015	13510406	8435493	7891
rawConflicts	1191	21.245	34.015	13534681	8452459	7094
allCorrections	583	22.995	34.325	5931142	3725485	6387

Tabella C.4. Prestazioni sperimentali 9-puzzle (5x2) con diverse euristiche; 200 nodi casuali

Appendice C: Risultati sperimentali: tempo e nodi espansi

heuristic	Time	Avg. h	Avg. d	Gen.	Exp.	Exp/s
manhattan	3570	22.85	34.77	59145252	32501724	9101
linearConflict	975	24.47	34.77	12932839	7152856	7331
nonLinearConflict	3083	23.27	34.77	42275548	22889306	7422
cornerTiles	1632	23.81	34.77	25521951	14096260	8633
lastMoves	2068	23.69	34.77	32775920	17683248	8549
cornerDeduction	1655	23.83	34.77	23679648	13059344	7890
conflicts	894	24.73	34.77	9270514	5062947	5659
conflictDeduction	509	25.63	34.77	4778844	2587509	5080
lastMovesLinearConflict	603	25.31	34.77	7481292	4077110	6760
rawConflicts	759	24.89	34.77	8348260	4548243	5988
allCorrections	338	26.71	35.19	3357722	1807672	5336

Tabella 5. Prestazioni sperimentali 11-puzzle (4x3) con diverse euristiche; 100 nodi casuali

heuristic	Time	Avg. h	Avg. d	Gen.	Exp.	Exp/s
manhattan	3897	35.20	47.00	61909889	31146193	7990
linearConflict	1003	36.40	47.00	12130824	6065033	6043
nonLinearConflict	2756	35.90	47.00	35117827	17481875	6341
cornerTiles	1434	36.00	47.00	21752049	11126762	7754
lastMoves	2100	36.20	47.00	32414946	16010667	7621
cornerDeduction	1605	36.10	47.00	19409295	9944901	6195
conflicts	986	37.10	47.00	6493209	3220928	3266
conflictDeduction	397	38.10	47.00	2818214	1387908	3494
lastMovesLinearConflict	609	37.40	47.00	6127049	3020868	4957
rawConflicts	774	37.10	47.00	6118606	3032589	3913
allCorrections	476	39.00	47.40	3499493	1701587	3567

Tabella C.6. Prestazioni sperimentali 15-Puzzle con diverse euristiche ; 20 istanze

BIBLIOGRAFIA

- [OPENOFFICE_SITE] <http://www.openoffice.org>, Openoffice community.
- [Pearl84] Pearl, J.: *Heuristics*, Addison-Wesley, Reading, MA, 1984.
- [RussellNorvig1998] S.J. Russell, P. Norvig: *Intelligenza Artificiale: un approccio moderno*, UTET, 1998.
- [Johnson1879] W. W. Johnson: Note on the "15" puzzle., *American Journal of Mathematics*, (2), pp. 397-399, 1879.
- [Wilson1974] R.M. Wilson: Graph puzzles, homotopy and alternating group., *J. Combin. Theory (Series B)*, (16), pp. 86-96, 1974.
- [KaindlKainz1997] Kaindl, H., Kainz, G.: Bidirectional Heuristic Search Reconsidered, *J. Artif. Intell. Res. (JAIR)*, 7, 1997.
- [Korf2005] Richard Korf, Weixiong Zhang, IgnacioThayer, and Heath Hohwald: Frontier Search, *Journal of the ACM*, 52 (5), pp. 715-748, 2005.
- [Pohl1970] Pohl: Heuristic search viewed as a path finding in a graph, *Artificial Intelligence*, 1, pp. 193-204, 1970.
- [Gaschnig1979] Gasching: Performance measurement and analysis of certain search algorithms, Tech Report CMU-CS-79-124. CMU, PA, 1979.

- [Korf1985] Korf, R.E.: Depth-First Iterative-Deepening: An Optimal Admissible Tree Search, *Artificial Intelligence*, 27 (1), pp. 97-109, 1985.
- [Korf1993] Korf, R.E.: Linear-space best-first search, *Artificial Intelligence*, 62 (1), pp. 41-78, 1993.
- [Pohl1971] Pohl, I.: Bi-directional search, *Machine Intelligence*, 9, pp. 127-140, 1971.
- [Kwa1989] Kwa, J.: BS*: an admissible bidirectional staged heuristic search algorithm, *Artificial Intelligence*, 38 (2), pp. 95-109, 1989.
- [DeChampeaux1983] de Champeaux: Bidirectional heuristic search again, *J. ACM*, 30 (1), pp. 22-32, 1983.
- [Manzini1995] Manzini, G.: BIDA*: an improved perimeter search algorithm, *Artificial Intelligence*, 75 (2), pp. 347-360, 1995.
- [Korf1990] Korf, R.E.: Real-Time Heuristic Search, *Artificial Intelligence*, 42 (2-3), pp. 189-211, 1990.
- [Shannon1950] Shannon, C. E.: Programming a computer for playing chess, *Philos. Mag.*, 41, pp. 256-275, 1950.
- [HartEdwards1963] Hart T.P., Edwards: The alpha-beta heuristic, MIT Artificial Intelligence Project Memo, 1963.
- [Ishida2003] Ishida, Toru. Shimbo, Masashi.: Controlling the learning process of real-time heuristic search, *Artificial Intelligence*, 146 (1), pp. 1-41, 2003.
- [Thorpe1994] P. E. Thorpe: A hibrid real time search algorithm, Ph.D. Thesis, Computer Science Department, UCLA, 1994.
- [Furcy2000] David Furcy, Sven Koenig: Speeding up the Convergence of Real-Time Search, AAAI, 2000.
- [HernandezMeseguer2005] Carlos Hernández, Pedro Meseguer: Propagating Updates in Real-Time Search: HLRTA*(k), CAEPIA, 2005
- [KorfReidEdelkamp2001] Korf, R. E.; Reid, M.; Edelkamp, S.: Time complexity of Iterative-Deepening-A*, *Artificial intelligence*, 129 (1-2), pp. 199-218, 2001.
- [HoosStutzle2005] Holger H. Hoos and Thomas Stützle: *Stochastic local search: foundations and applications*, Physica Verlag, 2005.
- [Ernandes2003] Ernandes M.: Joining symbolic and subsymbolic sources for p-admissible non memory-based heuristics, *AI*IA*, 4, 2003.
- [Harris1974] LR Harris: The Heuristic Search under Conditions of Error, *Artificial*

Intelligence, 1974.

[KorfBook] Korf, R.E.: *Heuristic Search*, unpublished book, 2003.

[FikesNilsson1971] Fikes, R., and N. Nilsson: STRIPS: A new approach to the application of theorem proving to problemsolving, *Artificial Intelligence*, 2, pp. 189-208, 1971.

[Prieditis1993] A.E. Prieditis: Machine Discovery of Effective Admissible Heuristics, *Machine Learning*, 12, pp. 117-141, 1993.

[CulbersonSchaeffer1998] Culberson, J., and J. Schaeffer: Pattern Databases, *Computational Intelligence*, 14 (4), 1998.

[KorfFelner2002] Korf, R.E, and A. Felner: Disjoint pattern databases heuristics, *Artificial Intelligence*, 134 (1-2), pp. 9-22, 2002.

[HanssonMayerYung1992] O.r Hansson, A. Mayer, M.i Yung: Criticizing solutions to relaxed models yields to powerful admissibleheuristics, *Information Science*, 63 (3), pp. 207-227, 1992.

[Hoffmann2001] Hoffmann, J: Local Search Topology in Planning Benchmarks: An Empirical Analysis, *IJCAI*, 2001.

[Hoffmann2002] Hoffmann, J: Local Search Topology in Planning Benchmarks: A Theoretical Analysis, *AIPS*, 2002.

[AyyoubMasoud2000] Abdel Elah Al-Ayyoub, Fawaz Ahmed Masoud: Heuristic search revisited, *Journal of Systems and Software*, 55 (2), pp. 103-113, 2000.

[LinaresJunghanns2002] C. Linares and A. Junghanns: Perimeter search performance, *Lecture Notes in Computer Science*, 2883, pp. 345-359, 2003.

[DEEPBLUE_SITE] <http://www.research.ibm.com/deepblue/>, IBM.

[Schaeffer1997] Schaeffer, J.: *One Jump ahead: challenging human supremacy in checkers*, Springer-Verlag, 1997.

[ArmanoPython] Armano, G.: Dispense del corso OOP and Scripting in Python.

[DowneyElkner2002] Allen Downey, Jeffrey Elkner e Chris Meyers: *How to Think Like a Computer Scientist: Learning with Python* , 2002.

[JAVA_SITE] <http://java.sun.org>, Sun Microsystems, Inc.

[PYTHON_SITE] <http://www.python.org/>, Python Software Foundation.

[VISUALWORKS_SITE] <http://www.cincomsmalltalk.com/>, Cincom, inc..

[PSYCO_SITE] <http://psyco.sourceforge.net/>, Open source project.

[pylab] <http://matplotlib.sourceforge.net>, Open source project.

[ECLIPSE_SITE] <http://www.eclipse.org/>, The Eclipse Foundation.

[PYDEV_SITE] <http://pydev.sourceforge.net/>, Open source project.

[CTYPES_SITE] <http://sourceforge.net/projects/ctypes/>, Open source project.

[stewart94] Stewart, W. J.: *Introduction to the Numerical Solution of Markov Chains*, Princeton, New Jersey: Princeton University Press, 1994.