UNIVERSITA' DEGLI STUDI DI FIRENZE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Esercitazioni per l'esame di
"Intelligenza Artificiale"
Prof. G. Soda

*WIDA**

di Sauro Menchetti

A.A.1998-99

Introduzione

L'algoritmo WIDA* (Weighted Iterative Deepening A*) è una modifica dell'algoritmo di ricerca IDA* in cui il peso relativo di g e h viene controllato ponendo:

$$f(n) = g(n) + w\ h(n)$$

dove $w \geq 0$ è un peso associato alla funzione h. Grandi valori di w enfatizzano il potere euristico, mentre piccoli valori di w enfatizzano il carattere Breadth First dell'algoritmo. L'idea che porta all'introduzione del peso w, è quella di riuscire a trovare la soluzione in modo più veloce: questo porta purtroppo a dover rinunciare all'ottimalità dell'algoritmo nel caso in cui l'euristica h sia ammissibile.

1.1 Caratteristiche generali dell'algoritmo WIDA*

L'algoritmo di ricerca WIDA* esegue gli stessi passi dell'algoritmo IDA*; l'unica differenza riguarda la funzione $f(n)$:

$$f(n) = g(n) + h(n) \text{ in IDA*}$$
$$f(n) = g(n) + w\ h(n) \text{ in WIDA*}$$

dove $f(n)$ rappresenta una stima del costo del percorso vincolato a passare dal nodo n, $g(n)$ rappresenta un valore esatto (distanza dalla radice al nodo n) e $h(n)$ rappresenta una stima della distanza dal goal (è la funzione euristica). In generale, il fattore di correzione w fa perdere l'ammissibilità all'algoritmo di ricerca che quindi non garantisce più di trovare la soluzione ottimale del problema.

1.1.1 L'algoritmo WIDA*

L'algoritmo può essere riassunto nei seguenti passi, in cui la funzione euristica indica i nodi da scegliere all'interno di una ricerca Depth First:

1) *Inizializza il valore di taglio C;*
2) *Poni nello stack S i nodi iniziali;*
3) *Se lo stack S è vuoto, incrementa il valore di taglio C e vai al passo 2), altrimenti sia n il primo nodo di S;*
4) *Se n è un nodo goal, stop e restituisci il percorso dal nodo iniziale a n;*
5) *Altrimenti rimuovi n da S, poni sul top dello stack S ogni figlio m di n per cui $f(m) \leq C$ e ritorna al passo 3).*

1.2 Implementazione dell'algoritmo WIDA*

L'algoritmo illustrato lascia alcuni margini di scelta quando si passa all'implementazione: ad esempio, in che modo deve essere incrementato il valore di taglio *C* e in che ordine i figli di un nodo *n* devono essere inseriti sul top dello stack *S*. Un'altra questione aperta riguarda come evitare di generare degli stati già generati in precedenza per evitare eventuali cicli infiniti.

1.2.1 Evitare la ripetizione degli stati

La tecnica usata per evitare la ripetizione degli stati implementa un metodo per non creare cammini che abbiano cicli. La funzione di espansione (o l'insieme degli operatori) deve rifiutare di generare qualsiasi successore del nodo che coincida con un suo qualsiasi antenato (non si fa un controllo su tutti i nodi generati in precedenza ma solo sugli antenati per avere un utilizzo lineare e non esponenziale della memoria). Questo viene realizzato memorizzando i nodi che costituiscono il cammino dalla radice al nodo corrente in uno stack *T* ed usando un insieme di riferimenti tra lo stack *S* e lo stack *T*. L'occupazione di spazio rimane così lineare.

1.2.2 Incremento del valore di taglio

Un modo semplice ma non molto efficiente per incrementare il valore di taglio è quello di aumentarlo ad ogni ripartenza di un valore costante fissato a priori, ad esempio uno. Esiste un altro modo di più difficile implementazione ma di maggiore efficacia: si incrementa il valore di taglio di una quantità dipendente dalla ripartenza a cui siamo giunti. Per determinare questa quantità, si fa uso di una variabile ausiliaria `min_cut_off` che contiene, per ogni ripartenza, il valore della funzione *f(n)* più piccolo fra tutti quelli generati superiore al valore di taglio corrente. Osservando poi che un goal avrà sempre un valore di *f(n)* intero, si incrementa il valore di taglio della quantità `min_cut_off` arrotondata all'intero immediatamente superiore. Questo accorgimento permette di evitare inutili ripartenze e quindi di diminuire il numero di nodi espansi a vantaggio di una maggiore velocità dell'algoritmo.

1.2.3 Inserimento dei figli sul top dello stack

Ci sono vari modi per inserire i figli in cima allo stack: si può stabilire un ordine prefissato in cui applicare gli operatori ed inserire in quest'ordine i figli oppure si possono inserire casualmente i figli sul top dello stack. Nell'algoritmo realizzato, i figli vengono inseriti in modo che quello con il più piccolo valore di *f(n)* si trovi in cima allo stack. L'idea si basa sul fatto che se l'euristica fosse perfetta, il valore di *f(n)* calcolato sulla radice sarebbe uguale a quello calcolato sul goal e su tutti i nodi che si trovano sul cammino che porta dalla radice al goal: quindi, nell'espandere i nodi, dovrei scegliere quello con il valore di *f(n)* uguale al padre, che non è altro che quello con il valore più piccolo di *f(n)*. Anche questo piccolo accorgimento apporta qualche lieve miglioramento all'algoritmo.

Test sull'algoritmo WIDA*

I test sull'algoritmo hanno come obiettivo la riduzione del numero dei nodi espansi[1] e quindi del tempo di ricerca, pur cercando di trovare soluzioni vicine a quelle ottimali. Sono testate varie funzioni per *w*, evidenziandone pregi e difetti.

2.1 Test

Il test viene eseguito su un campione rappresentativo di 1000 configurazioni del gioco dell'otto generate a caso e distinte l'una dall'altra. Tali configurazioni sono salvate su di un file e vengono caricate in memoria centrale quando è necessario. Per ogni configurazione iniziale vengono valutati alcuni parametri ed i dati riportati sulle tabelle sono la somma di questi parametri per tutte e 1000 le configurazioni; quindi, se si è interessati ai valori medi, basta dividere per 1000 i valori riportati nelle tabelle.

2.1.1 La distanza Manhattan

L'euristica usata è la distanza Manhattan: è un'euristica ammissibile, in quanto non sopravvaluta mai la distanza effettiva dal nodo *n* al goal più vicino. Poiché la distanza Manhattan è sempre minore od uguale alla distanza effettiva, l'idea di base è quella di moltiplicarla per un fattore maggiore od uguale ad uno, in modo da avvicinarsi al numero reale di passi che mancano per arrivare al goal. Il rischio è quello di sopravvalutare la distanza effettiva dal goal e quindi di non trovare una soluzione ottimale: infatti, come *w* cresce al di sopra di uno, cresce anche la lunghezza della soluzione, mentre diminuisce il numero di nodi generati. Inoltre, quando $w \geq 1$, la funzione *f(n)* perde la sua monotonicità. Un teorema ci fornisce un limite superiore all'incremento della lunghezza rispetto alla soluzione ottimale:

Teorema: Se $w \geq 1$, la soluzione trovata da WIDA* non può eccedere quella ottima di più di un fattore *w*.

La sovrastima della distanza effettiva implica l'utilizzo di maggiori informazioni provenienti dal problema: anche se questo va a scapito dell'ammissibilità, si possono risolvere gli stessi problemi in meno tempo.

[1] Un nodo si dice espanso se è un nodo goal o se sono stati generati tutti i suoi successori (può anche non avere successori e in tal caso è una foglia).

2.1.2 Obiettivo del test

Data la funzione euristica *h(n)* e data la distanza effettiva dal goal *d(n)*, si vuole determinare una funzione *w* tale che:

$$d(n) = w\, h(n) \text{ con } w \geq 1.$$

Se si riuscisse a trovare una tale funzione *w*, essa garantirebbe l'espansione del numero minimo di nodi, mantenendo anche l'ammissibilità: il problema è che la funzione *d(n)* non è nota a priori.

2.2 Misure di prestazioni

Per verificare la bontà di una funzione per *w* rispetto ad un'altra, vengono misurati vari parametri che tengono conto sia della riduzione del tempo di ricerca, sia dell'aumento del costo della soluzione.

La prima riga di ogni tabella contiene la somma dei costi dei cammini trovati dall'algoritmo di ricerca WIDA*. Questo valore permette di verificare di quanto le soluzioni trovate si discostino da quelle ottime: in altre parole, è una misura di ammissibilità dell'algoritmo. La seconda riga contiene la somma dei nodi espansi in totale dall'algoritmo ed è quindi un valore direttamente proporzionale al tempo impiegato per risolvere il problema. La terza riga contiene la somma dei nodi espansi durante l'ultima ripartenza e serve a caratterizzare quanto tempo viene impiegato in quest'ultima fase rispetto al tempo totale. La quarta riga contiene la somma del numero di ripartenze ed indica quante volte l'algoritmo ritorna sui propri passi. La quinta e la sesta riga contengono la somma delle penetranze. La penetranza è definita come il rapporto tra la lunghezza della soluzione trovata ed il numero di nodi generati. Nelle tabelle viene riportata la somma delle penetranze calcolata utilizzando i nodi espansi all'ultima ripartenza ed i nodi espansi in totale. La prima misura ha anche un'interpretazione grafica e sta ad indicare quanto velocemente l'algoritmo va verso il goal: chiamerò tale misura penetranza reale, mentre l'altra verrà chiamata penetranza totale. La penetranza sia reale sia totale, è un numero strettamente maggiore di zero e minore od uguale ad uno: se la funzione di successione è così precisa da generare un nodo alla volta, allora la penetranza è uguale ad uno. Valori prossimi allo zero indicano un numero molto elevato di nodi espansi rispetto a quelli che costituiscono la soluzione trovata.

2.3 Prestazioni dell'algoritmo IDA*

Vengono di seguito riportate le prestazioni dell'algoritmo IDA* sulle 1000 configurazioni distinte e generate a caso del gioco dell'otto, illustrando i miglioramenti apportati usando un incremento del valore di taglio dipendente dall'iterazione (ripartenza) a cui siamo giunti e inserendo il figlio con il valore di *f(n)* più piccolo sul top dello stack. L'algoritmo IDA* è un caso particolare dell'algoritmo WIDA* in cui $w = 1$.

La tabella seguente riporta i valori ottenuti eseguendo un incremento costante ed uguale ad uno del valore di taglio ed inserendo i figli del nodo espanso in cima allo stack secondo un ordine fissato a priori:

IDA*	w = 1.0
Somma dei cammini	22027
Somma dei nodi espansi in totale	3508711
Somma dei nodi espansi all'ultima ripartenza	864841
Somma delle ripartenze	7924
Somma delle penetranze reali (ultima iterazione)	126
Somma delle penetranze totali (nodi espansi in totale)	40

Tabella 1 Incremento costante del valore di taglio ed ordine prefissato di inserimento dei figli

Si può subito notare l'elevato numero di nodi espansi in totale rispetto a quelli espansi durante l'ultima iterazione: questo è dovuto in gran parte all'elevato numero di ripartenze senza successo che l'algoritmo esegue. Si notano anche dei piccoli valori per le due penetranze.

La seguente tabella riporta invece i valori ottenuti usando un incremento del valore di taglio dipendente dall'iterazione ed inserendo i figli nel top dello stack secondo un ordine determinato in precedenza:

IDA*	w = 1.0
Somma dei cammini	22027
Somma dei nodi espansi in totale	2186776
Somma dei nodi espansi all'ultima ripartenza	864841
Somma delle ripartenze	3962
Somma delle penetranze reali (ultima iterazione)	126
Somma delle penetranze totali (nodi espansi in totale)	54

Tabella 2 Incremento variabile del valore di taglio ed ordine prefissato di inserimento dei figli

Come si può vedere, il numero di ripartenze è esattamente dimezzato, provocando una diminuzione notevole del numero di nodi espansi in totale ed un incremento del valore della penetranza totale; gli altri valori rimangono invariati. Una giustificazione di questo dimezzamento può essere questa: ho notato che la differenza tra la lunghezza del cammino ottimo e la distanza Manhattan è sempre un numero pari. Quindi l'incremento variabile del valore di taglio sarà dato dal più piccolo numero pari diverso da zero, che è ovviamente due, in quanto la distanza minima tra due configurazioni è proprio due.

La seguente tabella riporta infine i valori ottenuti applicando entrambi i miglioramenti all'algoritmo di ricerca:

IDA*	w = 1.0
Somma dei cammini	22027
Somma dei nodi espansi in totale	2155960
Somma dei nodi espansi all'ultima ripartenza	834025
Somma delle ripartenze	3962
Somma delle penetranze reali (ultima iterazione)	135
Somma delle penetranze totali (nodi espansi in totale)	54

Tabella 3 Incremento variabile del valore di taglio ed inserimento ordinato dei figli nello stack

L'inserimento ordinato dei figli provoca un'ulteriore diminuzione dei nodi espansi, incrementando la penetranza reale; gli altri valori rimangono invariati.

Notazione

In tutte le tabelle che seguiranno, verranno utilizzate le seguenti abbreviazioni:
scw sta per somma del costo dei cammini;

snt sta per somma dei nodi espansi in totale;
snu sta per somma dei nodi espansi nell'ultima ripartenza;
sr sta per somma del numero delle ripartenze;
spu sta per somma delle penetranze reali (considerando l'ultima iterazione);
spt sta per somma delle penetranze totali (considerando i nodi espansi in totale).

2.4 Utilizzo di varie funzioni per *w* in WIDA*

Per cercare di migliorare le prestazioni dell'algoritmo IDA*, ho testato varie funzioni per *w*: i risultati ottenuti indicano che si può ottenere una soluzione in modo più veloce solo a scapito dell'ottimalità. Ci sono comunque funzioni che si comportano meglio di altre, nel senso che a parità di nodi espansi garantiscono di trovare una soluzione più vicina a quella ottimale oppure che a parità di incremento della lunghezza della soluzione, espandono meno nodi: lo scopo è quindi di trovare tali funzioni.

2.4.1 Funzione costante

La prima funzione testata per *w* è la funzione $w(h) = a$ dove $a \geq 1$ è un valore costante (se $a = 1$ si ottiene l'IDA*). Tale funzione è indipendente dall'altezza *h* a cui è giunto l'algoritmo: è quindi di facile implementazione. Il grafico di *w(h)* è il seguente:

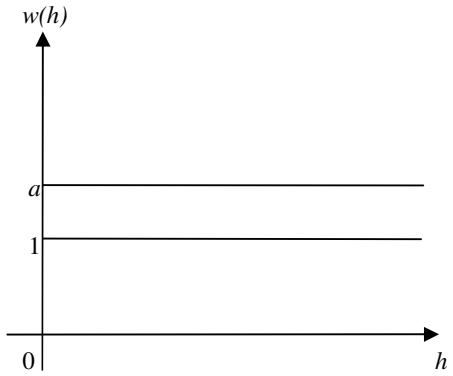

Figura 1 Funzione costante per *w(h)*

dove la variabile indipendente *h* indica l'altezza dell'albero di ricerca a cui siamo giunti e $y = 1$ è il valore sotto il quale la funzione *w(h)* non deve mai scendere. Tale funzione fa perdere l'ammissibilità all'algoritmo, in quanto moltiplica i valori di *h(n)* per un fattore costante indipendente dall'altezza: alcuni dei valori di *wh(n)* saranno certamente superiori a quelli di *d(n)* che rappresenta la vera distanza dal goal. Ecco i dati ottenuti al variare di *a* nell'intervallo [1.1, 2.5]:

a = *cost*	*a* = 1.1	*a* = 1.2	*a* = 1.3	*a* = 1.4	*a* = 1.5	*a* = 1.6
scw	22027	22105	22245	22513	22759	23099
snt	2842260	2510346	2233035	1955538	1862938	1642561
snu	664815	635779	609016	540400	524287	478205
sr	6952	6876	6597	6264	5942	5707
spu	157	164	172	182	182	189
spt	47	52	57	65	68	74

a = *cost*	*a* = 1.7	*a* = 1.8	*a* = 1.9	*a* = 2.0	*a* = 2.1	*a* = 2.5
scw	23319	23795	23947	24571	25179	28971
snt	1570199	1424117	1347613	1225261	1121676	938312
snu	488167	442678	451711	413923	409832	408783
sr	5383	5125	4921	4587	4249	3615
spu	184	188	188	200	200	221
spt	75	80	85	100	109	143

Tabella 4 Funzione *w(h) = a = cost*

I risultati evidenziano una forte diminuzione del numero di nodi espansi all'aumentare di *a*: purtroppo si perde l'ottimalità della soluzione. Anche le due penetranze subiscono un elevato aumento, mentre il numero di ripartenze è superiore (tranne per *a* = 2.5) a quelle di IDA*. Un buon compromesso tra velocità è ammissibilità è dato da *a* = 1.8 oppure *a* = 1.9.
Nota: la funzione *w(h) = a = cost* ottiene dei valori molto bassi per quanto riguarda la somma dei nodi espansi all'ultima iterazione, e dei valori molto alti per la somma delle penetranze (soprattutto per quella reale), difficilmente raggiungibili con altre funzioni.

2.4.2 Funzioni concave del tipo *w(h) = 1 + a / h^(n / m)*

Il passo successivo è stato quello di testare delle funzioni inversamente proporzionali ad una qualche potenza di *h*: quando *h* cresce, tali funzioni tendono ad uno ed per *h* = 1 assumono il valore 1 + *a*. L'andamento del grafico è il seguente:

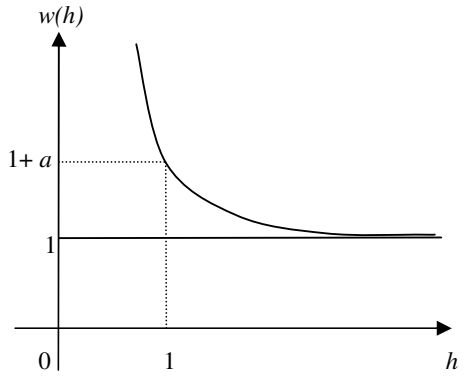

Figura 2 Funzione concava inversamente proporzionale ad *h^(n / m)*

Funzione *w(h) = 1 + a / h*
Ponendo *n* = *m* = 1, si ottiene una funzione per *w(h)* inversamente proporzionale all'altezza *h*: ecco i dati ottenuti al variare di *a*:

	a = 0.5	*a* = 0.7	*a* = 0.85	*a* = 1.0
scw	22569	23841	25289	26605
snt	2649400	2043299	1744680	1519284
snu	907132	1028816	1063353	1046026
sr	3306	2124	1644	1454
spu	122	127	134	140
spt	72	95	114	125

Tabella 5 Funzione *w(h) = 1 + a / h*

I risultati ottenuti sono peggiori di quelli ottenuti con un valore di *w(h)* costante: questo è dovuto al fatto che tale funzione è piuttosto elevata inizialmente (se *a* = 1

e $h = 1$, allora $w(h) = 2$), mentre poi vale pressoché 1. La somma dei nodi espansi all'ultima iterazione è persino peggiore di quella di IDA*.

Funzione *w(h) = 1 + a / h^2*

Il test seguente serve ad evidenziare che una potenza maggiore di uno per *h* porta a dei risultati praticamente uguali a quelli della tabella 5.

	$a = 0.5$	$a = 0.7$	$a = 0.85$	$a = 1.0$
scw	22569	23841	25295	26629
snt	1932378	1681313	1524483	1384784
snu	922306	1054551	1083889	1079510
sr	2427	1694	1394	1259
spu	119	126	131	138
spt	78	98	115	126

Tabella 6 Funzione *w(h) = 1 + a / h^2*

Alcuni parametri migliorano (somma dei nodi totali, somma ripartenze, somma penetranze totali), altri peggiorano (somma dei nodi espansi all'ultima iterazione che è maggiore che in IDA*, somma penetranze reali), rimanendo stabile la situazione complessiva.

Funzione *w(h) = 1 + a / h^(1 / 2)*

Si testano adesso degli esponenti minori di uno per *h*, visto che un incremento non ha portato dei buoni risultati. Ecco i dati per la radice quadrata di *h*:

	$a = 0.5$	$a = 0.7$	$a = 0.85$	$a = 1.0$
scw	22553	23641	24807	25729
snt	2169911	1653315	1402181	1305413
snu	736777	773895	790096	803916
sr	3874	2625	2065	1866
spu	146	147	150	151
spt	75	100	114	122

Tabella 7 Funzione *w(h) = 1 + a / h^(1 / 2)*

Si ottengono dei risultati migliori rispetto alla tabelle 5 e 6 a parità di incremento di lunghezza della soluzione. A differenza delle funzioni che usano un esponente maggiore di 1 per *h*, si ottengono dei miglioramenti rispetto ad IDA*: la somma dei nodi espansi in totale e quella espansi all'ultima iterazione sono peggiori di quelle ottenuti con i valori costanti.

Funzione *w(h) = 1 + a / h^(1 / 3)*

Diminuendo l'esponente di *h*, si misurano i seguenti valori:

	$a = 0.5$	$a = 0.7$	$a = 0.85$	$a = 1.0$
scw	22523	23413	24481	25291
snt	2018800	1664751	1335533	1198130
snu	659817	702680	669020	672910
sr	4363	3134	2504	2318
spu	154	156	161	164
spt	74	98	115	124

Tabella 8 Funzione *w(h) = 1 + a / h^(1 / 3)*

I dati migliorano sensibilmente rispetto alla precedente tabella, pur rimanendo la somma dei nodi espansi in totale e quella dei nodi espansi all'ultima iterazione in-

feriore a quella ottenuta con i valori costanti; i valori sono comunque migliori di quelli ottenuti con IDA*.

Funzione *w(h) = 1 + a / h^(1 / 4)*
L'ultima potenza testata per *h* è 1 / 4 che ci fornisce la seguente tabella:

	a = 0.5	*a* = 0.7	*a* = 0.85	*a* = 1.0
scw	22481	23431	24239	25009
snt	2078834	1544775	1270581	1190663
snu	648714	605391	584502	599847
sr	4693	3512	2916	2682
spu	158	166	172	173
spt	70	100	113	123

Tabella 9 Funzione *w(h) = 1 + a / h^(1 / 4)*

Si ha di nuovo un sensibile incremento delle prestazioni rispetto alla precedente tabella, pur rimando la somma dei nodi espansi all'ultima iterazione superiore a quella ottenuta con i valori costanti: questo conferma il fatto che piccoli esponenti per *h* funzionano meglio rispetto a valori tipo intero.

Nota: le funzioni del tipo *w(h) = 1 + a / h^(n / m)* forniscono dei valori molto bassi per quanto riguarda la somma delle ripartenze, mentre non sono da utilizzare per tutti gli altri parametri.

2.4.2 Funzione lineare *w(h) = a + (1 / a – 1) h / m*

Tale funzione nasce dal fatto che l'euristica *h(n)* = 0 se *n* è uno stato goal; se il goal si trova a profondità *d*, allora *w(d)* = 1 in quanto *w(h)* ≥ 1 qualunque sia *h*. Supponendo che *d* = *am* dove *m* è la distanza Manhattan del nodo iniziale (questa è un'approssimazione che è necessario fare, anche se è piuttosto grossolana) e calcolando l'equazione della retta che passa per i punti (0, *a*) e (*d*, 1), si ottiene tale funzione per *w(h)* di si riporta il grafico:

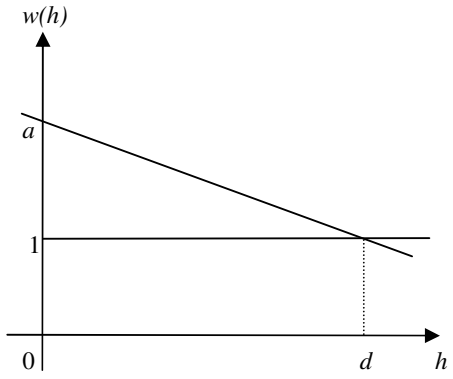

Figura 3 Funzione lineare per *w(h)*

L'idea è quindi di cercare di prevedere la distanza reale dal goal tramite la distanza Manhattan moltiplicata per il fattore *a*. Poiché inizialmente l'altezza *h* è nulla, il fattore *a* rappresenta una stima di quanto in media la distanza Manhattan sottovaluta la distanza effettiva. È molto difficile fornire una stima accurata di quanto in media la distanza Manhattan sottovaluti la distanza effettiva, quindi ho fatto variare il valore *a*, scegliendo poi come migliore stima il valore che fornisce i dati migliori. Ecco la tabella ottenuta al variare di *a*:

	$a = 1.1$	$a = 1.2$	$a = 1.3$	$a = 1.4$	$a = 1.5$
scw	22027	22129	22303	22611	23045
snt	2783385	2432053	2084949	1733426	1484955
snu	723755	689651	654000	600336	579842
sr	6782	6111	5436	4765	4164
spu	148	157	163	165	170
spt	47	53	63	74	81

	$a = 1.6$	$a = 1.7$	$a = 1.8$	$a = 1.9$	$a = 2.0$
scw	23483	24141	24843	25553	26415
snt	1349733	1194500	1058005	974269	950512
snu	571970	564212	529312	551851	575623
sr	3698	3322	3048	2750	2578
spu	176	178	185	180	182
spt	93	103	117	121	133

Tabella 10 Funzione $w(h) = a + (1 / a - 1)\ h / m$

A parità di incremento della soluzione ottimale, si ottengono dei valori migliori di quelli ottenuti con una funzione costante tranne per quanto riguarda la somma dei nodi espansi all'ultima iterazione. La somma delle ripartenze è solo leggermente superiore a quella ottenuta con $w(h) = 1 + a / h^{(n / m)}$.

Nota: la funzione lineare fornisce della cattive prestazioni per $a = 1.1$ e $a = 1.2$; le prestazioni sono comunque simili a quelle della funzione costante.

2.4.3 Funzione $w(h) = ((1 - b\ (1 - a) / am)\ h - ab) / (h - b)$ con $b = am + i$

L'idea che porta alla individuazione di tale equazione è la stessa di quella che porta all'equazione della funzione lineare. Presa la generica equazione dell'iperbole con quattro parametri liberi:

$$y = (ph + q) / (rh + s)$$

si impone il passaggio per i punti $(0, a)$ e $(d, 1)$.

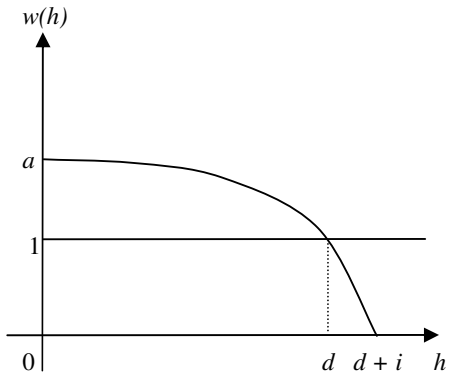

Figura 4 Funzione convessa per $w(h)$

Eseguendo i calcoli, si ottiene:

$$y = ((r + s\ (1 - a) / d)\ h + as) / (rh + s).$$

Dato che l'equazione ha ancora due gradi di libertà, per fissarne la curvatura, si impone l'ulteriore condizione di passaggio per il punto ($d + i$, 0). Dopo aver eseguito i calcoli ed essendosi semplificata l'unica variabile rimasta, si giunge all'equazione non ancora definitiva:

$$y = ((1 - (d + i)(1 - a) / d)\, h - a\,(d + i)) / (h - (d + i)).$$

Fatta l'approssimazione che $d = am$ dove m è la distanza Manhattan della configurazione iniziale, si giunge finalmente alla nostra equazione. Il parametro a assume lo stesso significato dell'equazione lineare, mentre il parametro i individua la curvatura dell'iperbole. Tramite l'approssimazione $d = am$, tale funzione cerca di assumere un valore unitario in corrispondenza dell'altezza della soluzione ottimale. Ecco i valori che si ottengono al variare di a ed i:

$a = 1.3$	$i = 5$	$i = 10$	$i = 15$	$i = 20$	$i = 25$
scw	22403	22509	22459	22445	22413
snt	2859367	2031479	1830115	1845973	1864253
snu	829846	607667	591768	578706	588213
sr	5948	5626	5550	5562	5564
spu	172	176	173	173	173
spt	63	65	65	64	64

Tabella 11 Funzione $w(h) = \{[1 - (am + i)(1 - a) / (am)]\, h - a(am + i)\} / [h - (am + i)]$ con $a = 1.3$

Per $i = 5$ ed $i = 10$ si ottengono delle prestazioni inferiori a quelle ottenute con il valore costante. Per gli altri valori di i, tale funzione fornisce dei valori migliori della funzione costante; per quanto riguarda la somma dei nodi espansi all'ultima iterazione, si riesce a raggiungere la funzione costante. Gli altri valori sono simili a quelli della funzione lineare. Vediamo che cosa succede incrementando a:

$a = 1.5$	$i = 3$	$i = 5$	$i = 8$	$i = 9$	$i = 10$	$i = 11$
scw	23103	23273	23375	23393	23393	23355
snt	2152472	1833276	1495686	1395989	1362903	1350158
snu	687756	622587	503412	494383	484189	495673
sr	5344	5046	4785	4726	4695	4640
spu	180	184	187	185	186	185
spt	73	77	80	80	80	80

$i = 12$	$i = 13$	$i = 14$	$i = 15$	$i = 17$	$i = 20$	$i = 25$
23339	23319	23323	23329	23331	23315	23265
1355169	1370418	1366776	1367543	1361553	1365478	1410895
505746	519273	517507	513539	516662	514404	523751
4588	4590	4572	4577	4568	4523	4464
186	184	184	184	185	185	182
82	82	82	83	83	84	83

Tabella 12 Funzione $w(h) = \{[1 - (am + i)(1 - a) / (am)]\, h - a(am + i)\} / [h - (am + i)]$ con $a = 1.5$

Per $i = 5$ ed $i = 10$ si ottengono sempre delle prestazioni piuttosto scarse; per $i = 10$ si ottengono le migliori prestazioni a parità di incremento della soluzione ottimale. Tutti gli altri valori sono leggermente migliori delle altre funzioni, tranne per quanto riguarda i valori della somma dei nodi espansi all'ultima iterazione e della somma delle penetranze reali ottenuti con la funzione costante. Incrementando ulteriormente a, si ottiene:

$a = 1.7$	$i = 5$	$i = 10$	$i = 15$	$i = 20$	$i = 25$
scw	24075	24275	24281	24233	24245
snt	1490321	1075659	1109007	1116060	1088497
snu	548947	451199	472004	475459	477959
sr	4444	4086	3918	3814	3705
spu	190	192	193	192	192
spt	86	92	97	98	100

Tabella 13 Funzione $w(h) = \{[1 - (am + i)(1 - a) / (am)]\ h - a(am + i)\} / [h - (am + i)]$ con $a = 1.7$

Le prestazioni sono migliori di quelle ottenute con la funzione lineare, ma peggiori anche se non di molto della funzione costante per quanto riguarda la somma dei nodi espansi all'ultima iterazione e la somma delle penetranze reali.

$a = 2.0$	$i = 5$	$i = 10$	$i = 15$	$i = 20$	$i = 25$
scw	25709	26069	26155	26227	26289
snt	1022868	878507	868567	863534	849699
snu	434997	407216	432334	430832	455639
sr	3923	3531	3310	3165	3028
spu	205	206	206	202	198
spt	109	118	122	123	127

Tabella 14 Funzione $w(h) = \{[1 - (am + i)(1 - a) / (am)]\ h - a(am + i)\} / [h - (am + i)]$ con $a = 2.0$

Un valore di $a = 2$ non cambia la precedente situazione.

$a = 2.5$	$i = 5$	$i = 10$	$i = 15$	$i = 20$	$i = 25$
scw	30239	30647	31205	31303	31567
snt	847922	744727	747141	716560	721013
snu	456587	457991	482898	483852	488319
sr	2923	2539	2395	2347	2324
spu	220	216	219	218	221
spt	151	161	165	167	172

Tabella 15 Funzione $w(h) = \{[1 - (am + i)(1 - a) / (am)]\ h - a(am + i)\} / [h - (am + i)]$ con $a = 2.5$

Infine per $a = 2.5$, si ottiene un incremento troppo elevato di mosse rispetto alla soluzione ottimale e dei valori peggiori di quelli raggiunti dalla funzione costante.

2.4.4 Funzione $w(m) = a(m - 10) / 12 + 1$ se $10 \leq m \leq 22$; 1 se $0 \leq m \leq 10$

Dopo aver testato la maggior parte di funzioni in cui w è funzione dell'altezza ed avendo visto quali sono le prestazioni raggiungibili, si testano adesso due funzioni in cui w è funzione della distanza Manhattan m del nodo n. Si parte dall'osservazione che la distanza Manhattan fornisce delle ottime approssimazioni della distanza reale se il valore dell'euristica non è superiore a circa dieci: cioè si può assumere che i valori della distanza Manhattan minori di dieci siano dei valori esatti e che quindi non vadano modificati. Quindi le due funzioni che seguiranno, saranno caratterizzate dal fatto che w è funzione della distanza Manhattan del nodo corrente e non dell'altezza; inoltre se $0 \leq m \leq 10$ circa, allora $w = 1$ indipendentemente da m. La prima funzione esaminata è una funzione lineare a tratti: vi è un primo tratto costante ed uguale ad uno che si raccorda con una retta nel punto (10, 1). Tale retta passa quindi per i punti (10, 1) e (22, 1 + a), dove delle variazioni di a cambiano il valore del coefficiente angolare. Il valore 22 è il valore massimo trovato per la distanza Manhattan. Si riporta il grafico di tale funzione:

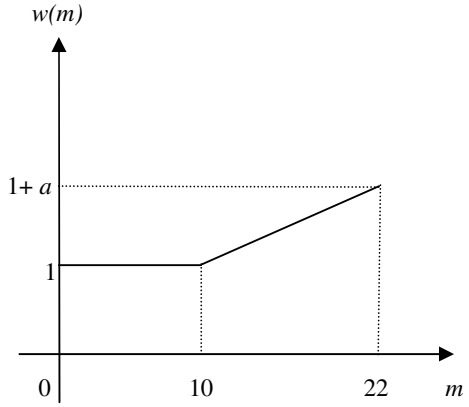

Figura 5 Funzione spezzata per *w(m)*

Ecco i valori che si ottengo per due valori di *a*:

	a = 0.3	*a* = 0.5
scw	22203	22615
snt	2432070	2156233
snu	634313	625190
sr	6544	5800
spu	157	162
spt	52	62

Tabella 16 Funzione $w(m) = a\,(\,m - 10)\,/\,12 + 1$ se $m \geq 10$, se $0 \leq m \leq 10$

Si nota che la somma dei nodi espansi in totale e delle ripartenze è persino peggiore di IDA*: tale funzione non è quindi utilizzabile.

2.4.5 Funzione *w(m) = 1 se 0 ≤ m ≤ l, a se m ≥ l con a ≥ 1*

Si testa adesso una funzione costante a tratti con un gradino a circa metà del dominio, come illustra il seguente grafico, al fine di vedere se è possibile ottenere dei buoni risultati considerando *w* come funzione della distanza Manhattan:

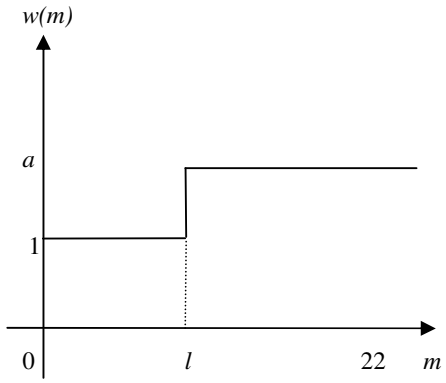

Figura 6 Funzione a gradino per *w(m)*

Il primo tratto è caratterizzato da un valore costante ed uguale ad uno, mentre il secondo tratto da un valore uguale ad *a*: c'è una discontinuità per $m = l$. L'effetto di tale funzione è quello di lasciare invariati i valori per cui $0 \leq m \leq l$, mentre gli altri vengono amplificati del valore costante *a* (è un effetto passa-alto). Si ottengono i seguenti valori al variare dei parametri *a* ed *l*:

$l = 10$	$a = 1.1$	$a = 1.2$	$a = 1.3$	$a = 1.4$	$a = 1.5$
scw	22331	22503	23163	23633	24253
snt	1569353	1968887	1624952	1459873	1306914
snu	665257	520034	508488	469840	493015
sr	4103	5736	4643	4494	3751
spu	155	178	179	185	185
spt	68	66	83	93	105

Tabella 17 Funzione con gradino in $l = 10$ e valore a variabile

A parità di incremento della lunghezza della soluzione trovata, per quanto riguarda la somma dei nodi espansi all'ultima ripartenza, si ottengono dei valori pressappoco uguali a quelli ottenuti con la funzioni costante, mentre gli altri parametri si avvicinano molto a quelli ottenuti con la funzione lineare, migliorando i valori ottenuti con la funzione costante. Per a = 1.1 si ottengono le migliori prestazioni a parità di incremento della lunghezza della soluzione trovata. Quindi tale funzione è una tra quelle che garantiscono delle alte prestazioni. Fissando $a = 1.1$ e facendo variare l, si ottengono i seguenti valori:

$a = 1.1$	$l = 9$	$l = 11$	$l = 12$
scw	22289	22287	22235
snt	2366256	1645761	1737650
snu	637497	709392	726636
sr	6385	4050	3966
spu	162	149	144
spt	55	65	62

Tabella 18 Funzione con $a = 1.1$ e gradino l variabile

I risultati sono peggiori rispetto a quelli riportati nella precedente tabella e indicano di fissare il limite l al valore dieci: piccole variazioni provocano un brusco calo delle prestazioni.

Nota: si verifica una certa instabilità al variare dei parametri in tale tipo di funzione.

2.5 Conclusioni

Facciamo adesso un riepilogo di tutte le funzioni usate, indicando quali parametri riescono ad ottimizzare.

Funzione *w(h) = a = cost*: è la più semplice funzione utilizzabile ma garantisce delle ottime prestazioni: fornisce dei valori ottimi per quanto riguarda le due penetranze e la somma dei nodi espansi all'ultima ripartenza.

Funzioni del tipo *w(h) = 1 + a / h^(n / m)*: è una famiglia di funzioni che non aumenta le prestazioni dell'algoritmo. L'unico parametro che riesce ad ottimizzare è la somma delle ripartente ma non la somma dei nodi espansi in totale.

Funzione lineare *w(h) = a + (1 / a – 1) h / m*: fornisce delle prestazioni leggermente superiori a quelle della funzione costante, eccetto che per la somma dei nodi espansi all'ultima iterazione. Il valore ottenuto per la somma delle ripartenze è simile a quello della precedente famiglia di funzioni.

Funzione convessa ***w(h) = ((1 – b (1 – a) / am) h – ab) / (h – b) con b = am + i*****:** questa classe di funzioni garantisce delle ottime prestazioni per tutti i parametri. Solo la funzione costante fornisce dei valori migliori per quanto riguarda la somma dei nodi espansi all'ultima iterazione.

Funzione ***w(m) = a (m – 10) / 12 + 1 se m ≥ 10, 1 se 0 ≤ m ≤ 10*****:** è un cattivo insieme di funzioni che non fornisce alcun incremento delle prestazioni.

Funzione ***w(m) = 1 se 0 ≤ m ≤ l, a se m ≥ l con a ≥ 1*****:** se fissiamo $l = 10$, otteniamo un'ottima funzione con prestazioni simili a quelle ottenute con le altre migliori funzioni: è forse la migliore funzione trovata.

2.5.1 Quale funzione scegliere?

Non esiste una funzione migliore delle altre in assoluto: ogni funzione ha un proprio campo d'azione in cui garantisce delle prestazioni superiori a quelle ottenute con tutte le altre funzioni. Ad esempio, se siamo disposti a pagare un incremento della lunghezza della soluzione medio di una mossa ogni tre configurazioni risolte, la funzione $w(m) = 1 \text{ se } 0 \leq m \leq l, a \text{ se } m \geq l \text{ con } a = 1.1 \text{ e } l = 10$ garantisce delle prestazioni superiori a tutte le altre; se invece siamo disposti a pagare un incremento medio di 1.3 mosse per configurazione risolta, la funzione $w(h) = ((1 – b (1 – a) / am) h – ab) / (h – b) \text{ con } b = am + i, a = 1.5 \text{ e } i = 10$ fornisce i migliori valori dei parametri.