Improved Prediction of IDA*'s Performance via ϵ-Truncation

Levi Lelis
Department of Computing Science
University of Alberta
Edmonton, AB, Canada T6G 2E8
santanad@cs.ualberta.ca

Sandra Zilles
Department of Computer Science
University of Regina
Regina, SK, Canada S4S 0A2
zilles@cs.uregina.ca

Robert C. Holte
Department of Computing Science
University of Alberta
Edmonton, AB, Canada T6G 2E8
holte@cs.ualberta.ca

Abstract

Korf, Reid, and Edelkamp launched a line of research aimed at predicting how many nodes IDA* will expand with a given cost bound. This paper advances this line of research in three ways. First, we identify a source of prediction error that has hitherto been overlooked. We call it the "discretization effect". Second, we disprove the intuitively appealing idea that a "more informed" prediction system cannot make worse predictions than a "less informed" one. More informed systems are more susceptible to the discretization effect, and in several of our experiments the more informed system makes poorer predictions. Our third contribution is a method, called "ϵ-truncation", which makes a prediction system less informed, in a carefully chosen way, so as to improve its predictions by reducing the discretization effect. In our experiments ϵ-truncation rarely degraded predictions; in the vast majority of cases it improved predictions, often substantially.

Introduction

Tree search is a popular technique for solving combinatorial problems (Pearl 1984). A frequent impediment of the application of tree searching algorithms is the inability to quickly predict the running time of an algorithm on a particular problem instance. While one instance of a problem might be solved in a blink of an eye, another instance of the same problem might take centuries. Korf, Reid, and Edelkamp (2001) launched a line of research aimed at creating a method to predict exactly how many nodes IDA* would expand on an iteration with cost bound d given a particular heuristic function. This was in contrast with the traditional approach to search complexity analysis, which focused on "big-O" complexity typically parameterized by the accuracy of the heuristic (Dinh, Russell, and Su 2007; Gaschnig 1979; Pearl 1984; Pohl 1977).

Korf, Reid, and Edelkamp developed a prediction formula, KRE, for the special case of consistent heuristics, proved that it was exact asymptotically (in the limit of large d), and experimentally showed that it was extremely accurate even at depths of practical interest. Zahavi, Felner, Burch, and Holte (2010) generalized KRE to work with inconsistent heuristics and to account for the heuristic values of the start states. Their formula, CDP, is described below.

d	IDA*	Coarse	Refined	ϵ-Coarse	ϵ-Refined
18	14.5	0.72	1.16	0.79	**1.02**
19	22.2	0.73	1.19	0.74	**1.01**
20	27.4	0.74	1.24	0.82	**0.99**
21	43.3	0.74	1.28	0.77	**0.99**
22	58.5	0.75	1.34	0.84	**0.96**
23	95.4	0.76	1.39	0.81	**0.98**
24	135.7	0.76	1.45	0.85	**0.93**
25	226.7	0.77	1.52	0.84	**0.98**
26	327.8	0.77	1.58	0.86	**0.92**
27	562.0	0.77	1.64	0.86	**0.98**

Table 1: 8-puzzle, Inconsistent Heuristic. Signed Error.

The present paper advances this research in three ways. First, we identify a source of prediction error that has hitherto been overlooked. We call it the "discretization effect". Second, we disprove the intuitively appealing idea, specifically asserted to be true by Zahavi *et al.*, that a "more informed" system cannot make worse predictions than a "less informed" system.[1] The possibility of this statement being false follows directly from the discretization effect, because a more informed system is more susceptible to the discretization effect than a less informed one. We will show several cases of this statement being false and use the phrase "informativeness pathology" to refer to this situation. Our final contribution is a method for counteracting the discretization effect, which we call "ϵ-truncation". One way to view ϵ-truncation is that it makes a prediction system *less informed*, in a carefully chosen way, so as to improve its predictions by reducing the discretization effect. In our experiments ϵ-truncation rarely degraded predictions; in the vast majority of cases it improved predictions, often substantially.

These contributions are illustrated in Table 1 for the 8-puzzle using the inconsistent heuristic defined by Zahavi *et al.* (see their Table 10). Column "d" shows the cost bound for which the prediction is being made and column "IDA*" shows how many nodes IDA* actually expanded, on average, on its iteration with cost bound d for the given set of start states. The remaining columns represent the quality of the predictions of four prediction systems (defined below under different labels). A perfect score is 1.0 (we explain how this score is calculated in the experimental section of

[1] "More informed" is defined formally in Definition 3 below.

this paper). The *Refined* system is "more informed" than the *Coarse* system; ϵ-*Coarse* and ϵ-*Refined* are the resulting systems when ϵ-truncation is applied to *Coarse* and *Refined*, respectively. The number given is the ratio of the average predicted number of nodes expanded to the average number of nodes actually expanded. For example, the entry in the bottom row of the *Coarse* column being 0.77 means that for $d = 27$, the prediction by the *Coarse* system was 77% of the actual IDA* value (*Coarse* predicted 432.7 nodes would be expanded, on average, whereas in reality 562.0 were). *Coarse* is the $\texttt{CDP}_2$ system of Zahavi *et al.* as it was applied to the 8-puzzle and the *Coarse* column here exactly reproduces the $\texttt{CDP}_2$ column in their Table 10. These are among the least accurate predictions Zahavi *et al.* report for CDP, and they offer no explanation for these poor results. Even though *Refined* is "more informed" than *Coarse*, Table 1 shows that its predictions are worse than *Coarse*'s for $d \geq 21$. The last two columns show the effect of applying ϵ-truncation: in both cases the predictions are substantially improved, and the more informed system ends up outperforming the less informed system.

The CDP Prediction Framework

Here we briefly sketch the CDP system using our own notation. The reader is referred to the original paper (Zahavi et al. 2010) for full explanations and illustrative examples. For reference, Table 2 summarizes the notation introduced in this section.

Let S be the set of states, $E \subseteq S \times S$ the set of directed edges over S representing the parent-child relation in the underlying state space, and $h : S \to \mathbb{N}$ the heuristic function.

Definition 1 *$T = \{t_1, \ldots, t_n\}$ is a type system for (S, E) if it is a disjoint partitioning of E. For every $(\hat{s}, s) \in E$, $T(\hat{s}, s)$ denotes the unique $t \in T$ with $(\hat{s}, s) \in t$.*

Definition 2 *Let $t, t' \in T$. $p(t'|t)$ denotes the probability that a node s with parent $\hat{s}$ and $T(\hat{s}, s) = t$ generates a node c with $T(s, c) = t'$. b_t denotes the average number of children generated by a node s with parent $\hat{s}$ with $T(\hat{s}, s) = t$.*

Parent-pruning is an easy-to-implement enhancement of IDA* that avoids re-expanding the parent of a node. IDA* with parent-pruning will not generate a node $\hat{s}$ from s if $\hat{s}$ is the parent of s. In making the prediction of the number of nodes expanded on an iteration of IDA* we are interested in estimating the number of nodes in the subtree below a given node. Because of parent-pruning the subtree below a node differs depending on the node from which it was generated. Like Zahavi *et al.*, to account for this effect of parent-pruning we define types over node pairs instead of just nodes.[2]

All type systems considered in this paper have the property that $h(s) = h(s')$ if $T(\hat{s}, s) = T(\hat{s}', s')$. We assume this property in the formulae below, and denote by $h(t)$ the value $h(s)$ for any $s, \hat{s}$ such that $T(\hat{s}, s) = t$.

[2]Zahavi *et al.* call the type definition over pairs a "two-step" model.

CDP samples the state space in order to estimate $p(t'|t)$ and b_t for all $t, t' \in T$. We denote by $\pi(t'|t)$ and $\beta(t)$ the respective estimates thus obtained. The predicted number of nodes expanded by IDA* (with parent pruning) for start state $\hat{s}^*$, cost bound d, heuristic h, and type system T is

$$\texttt{CDP}(\hat{s}^*, d, h, T) = \sum_{(\hat{s}^*, s^*) \in E} \sum_{i=1}^{d} \sum_{t \in T} N(i, t, (\hat{s}^*, s^*), d)\,.$$

Here $N(i, t, (\hat{s}^*, s^*), d)$ is the number of pairs $(\hat{s}, s)$ with $T(\hat{s}, s) = t$ and s at level i of the search tree rooted at s^*. It is computed recursively as follows.

$$N(1, t, (\hat{s}^*, s^*), d) = \begin{cases} 0 & \text{if } T(\hat{s}^*, s^*) \neq t\,, \\ 1 & \text{if } T(\hat{s}^*, s^*) = t\,, \end{cases}$$

and, for $i > 1$, the value $N(i, t, (\hat{s}^*, s^*), d)$ is given by

$$\sum_{u \in T} N(i-1, u, (\hat{s}^*, s^*), d)\pi(t|u)\beta_t P(t, i, d) \qquad (1)$$

where $P(t, i, d)$ = 1 if $h(t) + i \leq d$, and is 0 otherwise.

According to the formulae above, in order to predict the number of nodes IDA* expands with a cost bound d, for every level $i \leq d$, CDP predicts how many instances *of each type* will be generated; *i.e.*, it predicts a vector $(N[1], \ldots, N[|T|])$ of numbers of instances of each type on a level.[3] We will call such a vector a *type allocation vector*. The type allocation vector for the first level of prediction is computed by verifying the type of the children of the start state (the $i = 1$ base case of the recursive calculation shown above). Once the allocation vector is calculated for the first level, the vector for the next level is estimated according to Equation 1. At level i, for each type t such that $h(t) + i$ exceeds the cost bound d, the corresponding entry in the type allocation vector, $N[t]$, is set to zero to indicate that IDA* will prune nodes of this type from its search.[4] The prediction continues to deeper and deeper levels as long as there is an entry in the type allocation vector greater than zero.

As our basic type system, T_h, we use Zahavi *et al.*'s basic "two-step" model, defined (in our notation) as $T_h(\hat{s}, s) = (h(\hat{s}), h(s))$. Two new domain-independent type systems we will also use, which are "more informed" than T_h, are:

$T_c(\hat{s}, s) = (T_h(\hat{s}, s), c((\hat{s}, s), 0), \ldots, c((\hat{s}, s), H))$, where $c((\hat{s}, s), k)$ is the number of children of s, considering parent pruning, whose h-value is k, and H is the maximum h-value observed in the sampling process;

$T_{gc}(\hat{s}, s) = (T_c(\hat{s}, s), gc((\hat{s}, s), 0), \ldots, gc((\hat{s}, s), H))$, where $gc((\hat{s}, s), k)$ is the number of grandchildren of s, considering parent pruning, whose h-value is k.

The intuitive concept of one type system being "more informed" than another is captured formally as follows.

[3]We use $N[t]$ to denote $N(i, t, (\hat{s}^*, s^*), d)$ when i, $(\hat{s}^*, s^*)$ and d are clear from context.

[4]That is why we ensure all nodes mapped to a type have the same heuristic value, as mentioned above.

Notation	Meaning
T	type system
$T(\hat{s}, s)$	type of a pair $(\hat{s}, s) \in E$
$p(t'\|t)$	probability of type t generating type t'
b_t	average branching factor of node pairs of type t
$\pi(t'\|t)$	approximation of $p(t'\|t)$
β_t	approximation of b_t
$P(t, i, d)$	pruning function
$N(i, t, (\hat{s}^*, s^*), d)$	number of node pairs of type t at level i
$T_h(\hat{s}, s)$	type defined as $(h(\hat{s}), h(s))$
$T_c(\hat{s}, s)$	type defined by T_h and the h-values of s's children
$T_{gc}(\hat{s}, s)$	type defined by T_c and the h-values of s's grandchildren
$T_1 \prec T_2$	T_1 is a refinement of T_2

Table 2: Notation used in the CDP prediction framework.

Definition 3 *Let T_1, T_2 be type systems. T_1 is a refinement of T_2, denoted $T_1 \prec T_2$, if $|T_1| > |T_2|$ and for all $t_1 \in T_1$ there is a $t_2 \in T_2$ with $\{(\hat{s}, s)|T_1(\hat{s}, s) = t_1\} \subseteq \{(\hat{s}, s)|T_2(\hat{s}, s) = t_2\}$. If $t_1 \in T_1$ and $t_2 \in T_2$ are related in this way, we write $T_2(t_1) = t_2$.*

Note that $T_{gc} \prec T_c \prec T_h$, and so, by transitivity, $T_{gc} \prec T_h$.

Intuitively, if $T_1 \prec T_2$ one would expect predictions using T_1 to be at least as accurate as the predictions using T_2, since all the information that is being used by T_2 to condition its predictions is also being used by T_1 ((Zahavi et al. 2010), p. 59). However, our experiments show that this is not always true (*e.g.* Table 1, where *Refined*$\prec$*Coarse*). The underlying cause of poorer predictions by T_1 when $T_1 \prec T_2$ is the discretization effect, which we will now describe.

The Discretization Effect

Consider for example the situation depicted in Figure 1. Here type t_1 generates type t_2 with probability $p(t_2|t_1) = p$ and type t_3 with probability $p(t_3|t_1) = 1 - p$. Let $b_{t_1} = 1$, *i.e.*, one instance of type t_1 generates 1 new instance on average. Assuming, for simplicity, that the estimates $\pi(t'|t)$ and $\beta(t)$ are accurate, CDP predicts that p instances of type t_2 and $(1 - p)$ instances of type t_3 will be generated by one instance of t_1. If level $i - 1$ consists only of one instance, which is of type t_1, CDP's prediction $(N[1], N[2], N[3]) = (0, p, (1 - p))$ for level i minimizes the mean squared error (MSE) compared to the actual expected type allocation vector (a_1, a_2, a_3) on level i. Note that the ultimate task of CDP is to minimize the error in predicting the number of nodes expanded by IDA*. There is no guarantee that minimizing MSE for type allocation vectors at each level will actually minimize the overall MSE of the predicted number of nodes expanded, as will be illustrated below.

In the example above, if $p = 0.05$, CDP predicts that 0.05 instances of type t_2 and 0.95 instances of type t_3 will be generated, and thus minimizes the expected MSE of the type allocation vector. In an actual search tree, only integer values are possible, hence it is most likely that 1 instances of type t_3 and no instances of type t_2 are generated. Intuitively, the case that an instance of type t_2 is generated happens so rarely in practice that it might be better to predict that it never happens. That does not mean that fractional predictions in the type allocation vectors should be avoided in general, but in practice it might be advantageous to avoid very small fractions.

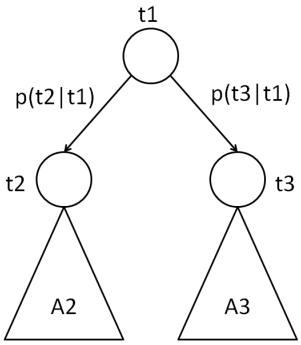

Figure 1: A hypothetical situation in CDP prediction.

To illustrate that it is at least possible that minimizing MSE for type allocation vectors does not minimize MSE of the number of nodes predicted to be expanded, consider again Figure 1. Let us denote by A_2 and A_3 the actual average number of nodes expanded below an instance of type t_2 and type t_3, respectively, and by E_2 and E_3 CDP's predictions for the number of nodes in the subtrees below an instance of type t_2 and an instance of type t_3, respectively. The expected MSE for CDP's prediction would be

$$p(pE_2+(1-p)E_3-A_2)^2+(1-p)(pE_2+(1-p)E_3-A_3)^2\,. \quad (2)$$

The MSE for predicting only 1 instance of type t_3 (and no instances of type t_2) of a system estimating A_3 with E'_3 would be

$$p(E'_3 - A_2)^2 + (1 - p)(E'_3 - A_3)^2\,. \quad (3)$$

It is mathematically possible for the value of (2) to be larger than that of (3), even if $E_3 = E'_3$, as can be verified by a simple algebraic calculation, which we skip because of space constraints. This by itself does not justify changing the way CDP makes predictions, but it shows at least that minimizing MSE for type allocation vectors does not guarantee minimizing MSE in the overall prediction.

Hence we would like to examine the possibility that ignoring very rare types, *i.e.*, in the example above, predicting that type t_2 will not be generated at all if p is very small, might in practice lead to better predictions—we call this phenomenon the discretization effect. Empirical evidence for the discretization effect would also explain what we observed in the introduction, namely that more refined type systems may lead to worse CDP predictions. The reason is that more refined type systems sometimes contain many very small type generation probabilities $p(t'|t)$. For example, for the 8-puzzle using the inconsistent heuristic defined by Zahavi *et al.*, 7.3% of the types generated with probability greater than zero are generated with probability lower

Level (i)	10	11-12	13	14	15	16-17	18-19	20-23	24
ϵ_i	0.05	0.07	0.08	0.07	0.05	0.04	0.03	0.01	0.00

Table 3: ϵ_i values for the 8-puzzle with Manhattan Distance.

than 1% for the more refined type system. This percentage is only 0.3% for the less refined type system.

The discretization effect hence would *(i)* explain why more refined type systems may behave poorly, and *(ii)* suggest that ignoring rare types may improve predictions overall. The goal of this paper is to show that our intuition is correct and to provide a method for systematically ignoring rare types in CDP, thus improving CDP predictions substantially in practice.

The ϵ-Truncation Prediction Method

Our approach to avoiding the discretization effect, which we call ϵ-truncation, can be summarized as follows.

1. As before, sample the state space to obtain $\pi(t|u)$.
2. Compute a cutoff value ϵ_i for each i between 1 and d.
3. Use ϵ_i to define $\pi^i(t|u)$, a version of $\pi(t|u)$ that is specific to level i. In particular, if $\pi(t|u) < \epsilon_i$ then $\pi^i(t|u) = 0$; the other $\pi^i(t|u)$ are set by scaling up the corresponding $\pi(t|u)$ values so that the $\pi^i(t|u)$ sum to 1.
4. In computing CDP use $\pi^i(t|u)$ at level i instead of $\pi(t|u)$.

The key step in this process is the calculation of the ϵ_i values. A full description of this calculation is given in the Appendix. It requires computing CDP predictions for a set of start states and, for each level i in each of these prediction calculations, solving a set of small linear programs (one for each "supertype" at that level, as defined in the Appendix). Table 3 shows the ϵ_i values calculated using 10,000 randomly generated start states for the 8-puzzle with Manhattan Distance. In practice, useful ϵ_i values can be computed using a much smaller number of start states. Also, as observed in Table 3 and in all the experiments in this paper, the ϵ_i values converge to zero as i gets larger.

Experimental Results

This section presents the results of experiments showing that: (a) refining a type system often reduces prediction accuracy; and (b) ϵ-truncation often substantially improves predictions. Each experiment will use two type systems, a basic one and a refinement of the basic one, and will compare the predictions made by CDP with each type system and with ϵ-truncation applied to the refined type system. We omit the results of applying ϵ-truncation to the basic type system for lack of space; as in Table 1 it never produced large improvements and never produced predictions that were more accurate than those of the refined system with ϵ-truncation. Our experiments are run on two domains: the sliding-tile puzzle, which has a small branching factor and deep solutions, and the pancake puzzle, which has a large branching factor and shallow solutions. We use at least two sizes for each domain: one that is small enough that the entire reachable portion of the state space can be enumerated and used in lieu of "sampling", and one that is large enough to be of practical interest. The small domains are an important element of the experiments because phenomena witnessed in them cannot be attributed to sampling effects. For each domain we use at least one consistent heuristic and one inconsistent one.

The choice of the set of start states will be described in the specific sections below, but we always applied the same principle as Zahavi *et al.* (2010): start state s is included in the experiment with cost bound d only if IDA* would actually have used d as a cost bound in its search with s as the start state. Unlike an actual IDA* run, we count the number of nodes expanded in the entire iteration for a start state even if the goal is encountered during the iteration.

As in Table 1 for each prediction system we will report the ratio of the predicted number of nodes expanded, averaged over all the start states, to the actual number of nodes expanded, on average, by IDA*. This ratio will be rounded to two decimal places. Thus a ratio of 1.00 does not necessarily mean the prediction is perfect. This ratio we call the (average) signed error. It is the same as the "Ratio" reported by Zahavi *et al.* (2010) and is appropriate when one is interested in predicting the total number of nodes that will be expanded in solving a set of start states. It is not appropriate for measuring the accuracy of the predictions on individual start states because errors with a positive sign cancel errors with a negative sign. If these exactly balance out, a system will appear to have no error (a ratio of 1.00) even though there might be substantial error in every single prediction. To evaluate the accuracy of individual predictions, an appropriate measure is absolute error. For each instance one computes the absolute value of the difference between the predicted and the actual number of nodes expanded, divides this difference by the actual number of nodes expanded, adds these up over all start states, and divides by the total number of start states. A perfect score according to this measure is 0.0. Appendix B shows the corresponding results when root mean squared error (RMSE) is chosen as a measure (recall that CDP minimizing MSE and thus RMSE for type allocation vectors does *not* imply that CDP minimizes RMSE for the overall prediction; the numbers in the appendix clearly demonstrate that CDP is outperformed by ϵ-truncation in terms of RMSE).

Zahavi *et al.* (2010) introduced a method for improving predictions for single start states. Instead of directly predicting how many nodes will be expanded for cost bound d and start state s, all states, S_r, at depth $r < d$ are enumerated and one then predicts how many nodes will be expanded for depth bound $d - r$ when S_r is the set of start states. We applied this technique in all our experiments. The value of r for each experiment is specified below.

The number of start states used to determine the ϵ_i is closely related to the r-value that will be used in the experiment. For example, the number of states (and thus the number of linear programs) at level 10 of the 8-puzzle is expected to be much lower than the number of states at level 25 of the 15-puzzle. Therefore, in order to find a suitable ϵ_{10}-value for the 8-puzzle we have to use more start states than

is required to determine the ϵ_{25}-value for the 15-puzzle. The number of states used to determine the ϵ_i is stated below for each experiment.

Small Domains

The 10-pancake puzzle has 10! states and a maximum optimal solution depth of 11. We used 100 random start states to determine the ϵ_i and 5,000 to measure prediction accuracy. The heuristic used was a pattern database (PDB (Culberson and Schaeffer 1996)) based on the smallest four pancakes. Heuristics defined by PDBs are admissible and consistent. We used T_h and T_{gc} as the type systems. The results of this experiment are shown in the upper part of Table 4. The lower part of Table 4 shows the same experiment when the heuristic is multiplied by 1.5, which makes it inconsistent and inadmissible (*e.g.,* there is a d=12 row even though the maximum optimal solution depth is 11). The bold entries in this and all other tables of results indicate the best predictions. In both these experiments the refinement T_{gc} substantially improves on T_h and ϵ-truncation automatically detects that T_{gc} contains almost no rare types and therefore makes little change to its predictions. For all the type systems the absolute error is substantially higher than the signed error suggests it should be, indicating that a combination of over- and under-estimation is occurring.

		Signed Error			Absolute Error		
d	IDA*	T_h	T_{gc}	ϵ-T_{gc}	T_h	T_{gc}	ϵ-T_{gc}
Admissible and Consistent Heuristic							
7	1,710.8	1.01	**1.00**	**1.00**	0.33	**0.02**	0.03
8	12,082.4	1.03	**1.00**	**1.00**	0.27	**0.03**	**0.03**
9	79,650.5	1.08	1.01	**1.00**	0.25	**0.03**	0.04
10	507,640.4	1.14	1.01	**1.00**	0.24	**0.03**	0.04
11	3,449,158.0	1.20	1.01	**1.00**	0.22	**0.02**	0.03
The heuristic above multiplied by 1.5							
7	528.5	1.03	**1.01**	**1.01**	0.68	**0.13**	**0.13**
8	3,740.3	1.06	**1.01**	**1.01**	0.51	**0.11**	**0.11**
9	13,924.7	1.13	**1.02**	**1.02**	0.57	**0.12**	**0.12**
10	64,611.8	1.22	1.04	**1.03**	0.50	**0.11**	**0.11**
11	356,366.1	1.36	1.06	**1.05**	0.46	**0.10**	**0.10**
12	2,397,748.6	1.36	1.09	**1.05**	0.39	**0.09**	**0.09**

Table 4: 10-pancake puzzle. r=1.

For the sliding-tile puzzle we ran experiments on two small sizes, the (3x3)-puzzle (the 8-puzzle) and the (3x4)-puzzle. We used the same type system as Zahavi *et al.* (2010), which is a refinement of T_h we call $T_{h,b}$. $T_{h,b}$ is defined by $T_{h,b}(s, s') = (T_h, blank(s), blank(s'))$ where $blank(s)$ returns the kind of location (corner, edge, or middle) the blank occupies in state s.[5] $T_{gc,b}$ is defined analogously. For square versions of the puzzle T_{gc} is exactly the same as $T_{gc,b}$ and therefore $T_{gc} \prec T_{h,b}$. However, for the (3x4)-puzzle, T_{gc} and $T_{gc,b}$ are not the same. For the 8-puzzle we used 1,000 random start states to determine the ϵ_i and every solvable state in the space to measure prediction accuracy. Table 5 shows the results for Manhattan Distance, which is admissible and consistent, with r=10. Here we see the informativeness pathology: T_{gc}'s predictions are worse than $T_{h,b}$'s, despite its being a refinement of $T_{h,b}$. Applying ϵ-truncation substantially reduces T_{gc}'s prediction error.

[5]For the (3x4)-puzzle there are two kinds of edge locations that $blank(s)$ needs to distinguish—edge locations on the short side (length 3) and edge locations on the long side (length 4).

		Signed Error			Absolute Error		
d	IDA*	$T_{h,b}$	T_{gc}	ϵ-T_{gc}	$T_{h,b}$	T_{gc}	ϵ-T_{gc}
18	134.4	1.01	1.01	**1.00**	0.03	**0.01**	**0.01**
19	238.4	1.01	1.01	**1.00**	0.04	**0.02**	**0.02**
20	360.1	1.01	1.02	**0.99**	0.06	**0.03**	**0.03**
21	630.7	1.01	1.02	**0.99**	0.06	**0.03**	**0.03**
22	950.6	1.02	1.04	**0.99**	0.08	0.05	**0.04**
23	1,649.5	**1.01**	1.04	**0.99**	0.08	0.06	**0.04**
24	2,457.5	**1.01**	1.07	0.98	0.09	0.08	**0.06**
25	4,245.5	**1.00**	1.08	0.98	0.09	0.08	**0.05**
26	6,294.4	**1.00**	1.10	0.97	0.10	0.11	**0.07**
27	10,994.9	**0.99**	1.11	0.97	0.11	0.11	**0.06**

Table 5: 8-puzzle, Manhattan Distance. r=10.

The inconsistent heuristic we used for the 8-puzzle is the one defined by Zahavi *et al.* (2010). Two PDBs were built, one based on tiles 1-4, and one based on tiles 5-8. The locations in the puzzle are numbered left-to-right and top-to-bottom and the first PDB is consulted for states having the blank in an even location and the second PDB is consulted otherwise. Since the blank's location changes parity every time it moves, we are guaranteed that the heuristic value of a child node will be drawn from a different PDB than its parent. Again we used 1,000 random start states to determine the ϵ_i and every solvable state in the space to measure prediction accuracy. The results of this experiment, with r=1, are shown in Table 6. The Signed Error columns of Table 6 reproduce the results shown in Table 1. As discussed in the Introduction, they exhibit the informativeness pathology and demonstrate that ϵ-truncation can substantially reduce prediction error. The pathology and the ϵ-truncation improvements are also observed when absolute error is the measure.

		Signed Error			Absolute Error		
d	IDA*	$T_{h,b}$	T_{gc}	ϵ-T_{gc}	$T_{h,b}$	T_{gc}	ϵ-T_{gc}
18	14.5	0.72	1.16	**1.02**	0.38	0.44	**0.36**
19	22.2	0.73	1.19	**1.01**	0.45	0.49	**0.39**
20	27.4	0.74	1.24	**0.99**	0.45	0.55	**0.41**
21	43.3	0.74	1.28	**0.99**	0.49	0.57	**0.40**
22	58.5	0.75	1.34	**0.96**	0.47	0.62	**0.41**
23	95.4	0.76	1.39	**0.98**	0.48	0.62	**0.38**
24	135.7	0.76	1.45	**0.93**	0.46	0.66	**0.37**
25	226.7	0.77	1.52	**0.98**	0.44	0.65	**0.34**
26	327.8	0.77	1.58	**0.92**	0.41	0.68	**0.31**
27	562.0	0.77	1.64	**0.98**	0.40	0.67	**0.27**

Table 6: 8-puzzle, Inconsistent Heuristic. r=1.

For the (3x4)-puzzle we used 10 random start states to determine the ϵ_i and 10,000 to measure prediction accuracy. The upper part of Table 7 shows the results for Manhattan Distance. $T_{gc,b}$'s absolute error is very close to $T_{h,b}$'s, so being more informed provides no advantage. ϵ-truncation very substantially improves $T_{gc,b}$'s absolute error. The lower

part of the table is for Manhattan Distance multipled by 1.5, which is inadmissible and inconsistent. Here $T_{gc,b}$'s predictions are considerably more accurate than $T_{h,b}$'s and are substantially improved by ϵ-truncation.

		Signed Error			Absolute Error		
d	IDA*	$T_{h,b}$	$T_{gc,b}$	ϵ-$T_{gc,b}$	$T_{h,b}$	$T_{gc,b}$	ϵ-$T_{gc,b}$
Manhattan Distance							
33	30,461.9	1.02	1.02	**1.01**	0.02	**0.01**	**0.01**
34	49,576.8	1.03	1.03	**1.01**	0.03	0.02	**0.01**
35	80,688.2	1.04	1.04	**1.02**	0.04	0.03	**0.02**
36	127,733.4	1.05	1.06	**1.02**	0.05	0.05	**0.03**
37	201,822.7	1.08	1.09	**1.04**	0.07	0.07	**0.04**
38	327,835.3	1.10	1.11	**1.04**	0.09	0.09	**0.05**
39	478,092.5	1.13	1.16	**1.06**	0.11	0.13	**0.06**
40	822,055.4	1.17	1.20	**1.07**	0.15	0.17	**0.08**
41	1,163,312.1	1.21	1.27	**1.10**	0.18	0.23	**0.10**
42	1,843,732.2	1.27	1.34	**1.13**	0.23	0.30	**0.13**
Weighted Manhattan Distance (w=1.5)							
33	926.2	1.05	1.01	**1.00**	0.02	**0.01**	**0.01**
34	1,286.9	1.07	1.02	**1.01**	0.03	**0.01**	**0.01**
35	2,225.6	1.09	1.03	**1.00**	0.05	**0.02**	**0.02**
36	2,670.7	1.12	1.05	**1.00**	0.06	0.03	**0.02**
37	3,519.5	1.14	1.07	**0.99**	0.08	0.04	**0.02**
38	5,570.8	1.19	1.10	**0.99**	0.13	0.07	**0.03**
39	6,983.8	1.23	1.13	**0.98**	0.15	0.09	**0.04**
40	9,103.3	1.29	1.18	**0.97**	0.20	0.12	**0.05**
41	13,635.3	1.36	1.24	**0.96**	0.28	0.18	**0.06**
42	16,634.2	1.43	1.31	**0.95**	0.33	0.23	**0.07**

Table 7: (3x4)-puzzle. r=20.

Large Domains

For the 15-pancake puzzle, we used 10 random start states to determine the ϵ_i and 1,000 to measure prediction accuracy while using the consistent heuristic and 1,000 while using the inadmissible and inconsistent heuristic. We used T_h and T_c as the type systems. To define $\pi(t|u)$ and β_t, 100 million random states were sampled and, in addition, we used the process described by Zahavi *et al.* (2010) to non-randomly extend the sampling: we sampled the child of a sampled state if the type of that child had not yet been sampled. The results with r=4 and a PDB heuristic based on the smallest eight pancakes are shown in the upper part of Table 8. In both cases T_c outperforms T_h but is also substantially improved by ϵ-truncation.

For the 15-puzzle, we used 5 random start states to determine the ϵ_i and 1,000 to measure prediction accuracy. To define $\pi(t|u)$ and β_t, one billion random states were sampled and, in addition, we used the extended sampling process described for the 15-pancake puzzle. Table 9 gives the results when Manhattan Distance is the heuristic and $T_{h,b}$ and T_{gc} are the type systems. Here again we see the informativeness pathology ($T_{h,b}$'s predictions are better than T_{gc}'s) which is eliminated by ϵ-truncation.

Like for the 8-puzzle, an inconsistent heuristic for the 15-puzzle was created with one PDB based on tiles 1-7, and the other based on tiles 9-15, exactly as Zahavi *et al.* did (see their Table 11). The results with $T_{h,b}$ and T_c as the type systems are shown in Table 10. Here we see that even though

		Signed Error			Absolute Error		
d	IDA*	T_h	T_c	ϵ-T_c	T_h	T_c	ϵ-T_c
Admissible and Consistent Heuristic (PDB)							
9	1,035.2	1.11	1.06	**1.04**	0.23	0.10	**0.09**
10	5,547.9	1.11	1.06	**1.02**	0.36	0.15	**0.11**
11	46,009.3	1.11	1.05	**0.99**	0.51	0.19	**0.13**
12	322,426.7	1.15	1.07	**0.98**	0.58	0.23	**0.14**
13	2,480,436.7	1.27	1.14	**1.02**	0.61	0.25	**0.15**
14	19,583,169.2	1.36	1.17	**1.03**	0.66	0.27	**0.17**
15	133,596,114.2	1.61	1.31	**1.16**	0.74	0.32	**0.20**
Same Heuristic, Weighted (w=1.5)							
12	188,177.1	2.00	1.25	**1.14**	1.51	0.51	**0.38**
13	398,418.8	2.13	1.32	**1.12**	1.62	0.53	**0.39**
14	3,390,387.6	2.31	1.38	**1.11**	1.62	0.51	**0.33**
15	6,477,150.7	2.24	1.28	**0.98**	1.74	0.54	**0.37**
16	16,848,215.1	2.79	1.49	**1.12**	1.98	0.55	**0.37**

Table 8: 15-pancake puzzle. r=4.

		Signed Error			Absolute Error		
d	IDA*	$T_{h,b}$	T_{gc}	ϵ-T_{gc}	$T_{h,b}$	T_{gc}	ϵ-T_{gc}
48	2,958,898.5	1.10	1.11	**1.05**	0.05	0.05	**0.03**
49	5,894,396.1	1.13	1.15	**1.06**	0.07	0.07	**0.04**
50	8,909,564.5	1.16	1.18	**1.08**	0.09	0.10	**0.05**
51	15,427,786.9	1.15	1.19	**1.07**	0.11	0.12	**0.07**
52	28,308,808.8	1.25	1.28	**1.14**	0.14	0.17	**0.09**
53	45,086,452.6	1.23	1.29	**1.13**	0.16	0.20	**0.11**
54	85,024,463.5	1.36	1.41	**1.22**	0.21	0.27	**0.15**
55	123,478,361.5	1.36	1.45	**1.24**	0.24	0.31	**0.17**
56	261,945,964.0	1.44	1.54	**1.30**	0.28	0.39	**0.21**
57	218,593,372.3	1.43	1.57	**1.32**	0.33	0.45	**0.26**
58	531,577,032.2	1.47	1.64	**1.37**	0.35	0.51	**0.30**

Table 9: 15-puzzle. Manhattan Distance. r=25.

$T_{h,b}$ presents a reasonable signed error, it has in fact a very large absolute error, and once again ϵ-truncation produced substantial improvement in prediction accuracy. These prediction results could be improved by increasing the r-value used. However, we wanted our results to be comparable those in Zahavi *et al.*'s Table 11.

Conclusion

In this paper we have identified a source of prediction error that has previously been overlooked, namely, that low probability events can degrade predictions in certain circumstances. We call this the discretization effect. This insight led directly to the ϵ-truncation method for altering the probability distribution used for making predictions at level i of the search tree by setting to zero all probabilities smaller than ϵ_i, an automatically derived threshold for level i. Our experimental results showed that more informed type systems for prediction often suffer more from the discretization effect than less informed ones, sometimes leading to the pathological situation that predictions based on the more informed system are actually worse than those based on the less informed system. In our experiments ϵ-truncation rarely degraded predictions; in the vast majority of cases it improved predictions, often substantially.

		Signed Error			Absolute Error		
d	IDA*	$T_{h,b}$	T_c	ϵ-T_c	$T_{h,b}$	T_c	ϵ-T_c
48	193,396.1	0.36	1.50	**1.00**	232.23	1.13	**1.08**
49	433,915.3	0.35	**1.13**	0.82	210.85	1.21	**1.11**
50	562,708.5	0.55	1.77	**1.20**	537.97	1.29	**1.17**
51	965,792.6	0.70	1.39	**1.04**	812.37	1.32	**1.12**
52	1,438,694.0	0.96	1.68	**1.23**	513.99	1.52	**1.35**
53	2,368,940.3	1.29	1.75	**1.32**	694.34	1.56	**1.26**
54	3,749,519.9	1.64	2.03	**1.54**	647.24	1.77	**1.53**
55	7,360,297.6	1.90	2.07	**1.59**	650.59	1.72	**1.35**
56	12,267,171.0	2.30	2.19	**1.61**	927.71	2.16	**1.86**
57	23,517,650.8	2.69	2.29	**1.78**	600.13	2.02	**1.55**
58	24,607,970.9	5.26	2.54	**1.90**	700.29	2.73	**2.38**

Table 10: 15-puzzle. Inconsistent Heuristic. r=1.

Appendix A: Deriving ϵ_i for $1 \leq i \leq d$

Our aim is to compute a value ϵ_i, for $1 \leq i \leq d$, below which values of $\pi(t|u)$ for $t, u \in T$ are ignored at level i. To do so, we propose an objective function whose minimizer defines a set of π-values that yield predictions with a minimal absolute expected error for the type allocation vectors.

Often a type at level $i+1$ can be generated by several different types at level i. To account for this we define a system of "super types". A *super type system* groups types at level i so that two different super types (at level i) will never generate children of the same type. In Figure 2, the pairs (*tree*$_x$,*tree*$_y$) for $(x, y) \in \{(1,3),(1,4),(2,3),(2,4)\}$ would be in one super type.

Definition 4 *Let T, T' be type systems, $T \prec T'$, and $t' \in T'$. For all i and all $(\hat{s}^*, s^*) \in E$, the super type $st(t', i, \hat{s}^*, s^*)$ over T contains exactly the pairs $(t_1, t_2) \in T \times T$ for which $T'(t_2) = t'$ and t_1 occurs at level i starting the prediction from $(\hat{s}^*, s^*)$. The super type system $ST(i, \hat{s}^*, s^*)$ over T with respect to T' is defined by $ST(i, \hat{s}^*, s^*) = (st(t'_1, i, \hat{s}^*, s^*), \ldots, st(t'_z, i, \hat{s}^*, s^*))$ where $T' = \{t'_1, \ldots, t'_z\}$. We write st instead of $st(t', i, \hat{s}^*, s^*)$ whenever t', i, $\hat{s}^*$, s^* are clear from context.*

In our experiments, for the pancake domain, when $T = T_{gc}$, we used $T' = T_c$; for the sliding tile puzzle, $T = T_{gc,b}$ was used with T' given by T_c augmented with the kind of blank location of the parent of the node.

We adapt CDP to super types by estimating, for state pair $(\hat{s}^*, s^*)$, level i, depth limit d, type t, and super type $st \in ST(i, \hat{s}^*, s^*)$, the probability of generating a node of type t after seeing super type st at level i. We denote the estimate by $\pi^{i,d}_{\hat{s}^*,s^*}(t|st)$, defined as

$$\frac{\sum_{\{t_p|(t_p,t)\in st\}} \pi(t|t_p)\beta_{t_p} N(i, t_p, (\hat{s}^*, s^*), d)}{\sum_{\{t_p|(t_p,t)\in st\}} \beta_{t_p} N(i, t_p, (\hat{s}^*, s^*), d)} .$$

The number of nodes $N(i, st, (\hat{s}^*, s^*), d)$ of a super type $st \in ST$ at a level i of prediction is given by

$$N(i, st, (\hat{s}^*, s^*), d) = \sum_{(t_p,t_c)\in st} N(i, t_p, (\hat{s}^*, s^*), d)\beta_{t_p} .$$

We then reformulate the CDP formula equivalently by computing $N(i, t, (\hat{s}^*, s^*), d)$ as

$$\sum_{st\in ST} N(i-1, st, (\hat{s}^*, s^*), d)\pi^{i-1,d}_{\hat{s}^*,s^*}(t|st)P(t, i, d) .$$

Next, we compute a redistribution of $\pi(t, st)$-values, for each st and each level i, from a solution to the following optimization problem.

Find a type allocation vector $(a_1, \ldots, a_{|T|})$ that minimizes:

$$\sum_{j=0}^{\lceil N(i,st,(\hat{s}^*,s^*),d)\rceil} \sum_{t\in T} |pr(j,t)(a_t - j)| \qquad (4)$$

$$\textit{s.t.:} \sum_{t\in T} a_t = \lceil N(i, st, (\hat{s}^*, s^*), d)\rceil \text{ and } a_t \geq 0 \text{ for } t \in T .$$

Here $pr(j, t)$ is short for the probability of generating exactly j children of type t from $\lceil N(i, st, (\hat{s}^*, s^*), d)\rceil$ many parents of super type st. The objective corresponds to replacing, for each super type st, each $\pi(t|st)$ by $a_t / \sum_{t\in T} a_t$ so as to minimize the expected *absolute* error of the type vector allocation.

We deliberately chose to minimize the absolute error rather than MSE here, because we expect small fractions in the type allocation vectors to be harmful, yet favoured by MSE minimization. Consider for example a case of only two types, one being generated with probability p and the other being generated with probability $1-p$, *i.e.*, the expected vector would be $(p, 1-p)$. Assume a system predicts a vector $(q, 1-q)$. Then its MSE would be $p\big((1-q)^2 + (0-(1-q))^2\big) + (1-p)\big((0-q)^2 + (1-(1-q))^2\big) = 2q^2 + 2p - 4pq$, which would be minimized by setting $q = p$. Thus MSE minimization would never suggest to ignore a small p value. The absolute error of the same prediction would be $p\big(|1-q| + |0-(1-q)|\big) + (1-p)\big(|0-q| + |1-(1-q)|\big) = 2p(1-q) + 2(1-p)q$. For $p < 0.5$ this error is minimized at $q = 0$ while for $p > 0.5$ error is minimized at $q = 1$ (and thus $1-q = 0$). In this example we assume that $N(i, st, (\hat{s}^*, s^*), d) = 1$. For larger values of $N(i, st, (\hat{s}^*, s^*), d)$ this error is minimized at $q = 0$ for smaller p. Minimizing the absolute error hence encourages the system to ignore types less likely to be generated given an $N(i, st, (\hat{s}^*, s^*), d)$-value, as is our intention.

Solving Equation 4 in every step of the prediction would be too time-consuming. Instead, it plays a role in preprocessing, when we use the modified π-values for *super types* to derive ϵ-cuts for π-values for *types* as follows. First, we solve a small number of instances of Equation (4) for each level i. For every value $\pi(t|st)$ (which is the x-axis in the example in Figure 3) estimated before solving (4), we compute the fraction of times that this $\pi(\cdot|\cdot)$-value was set to zero in the solutions to (4) for the given level i (this is the y-axis in Figure 3). The largest $\pi(t|st)$ that was set to zero in 50% or more of these instances at level i is $\hat{\epsilon}_i$, a potential cutoff value for level i (in Figure 3 this is the x-value where

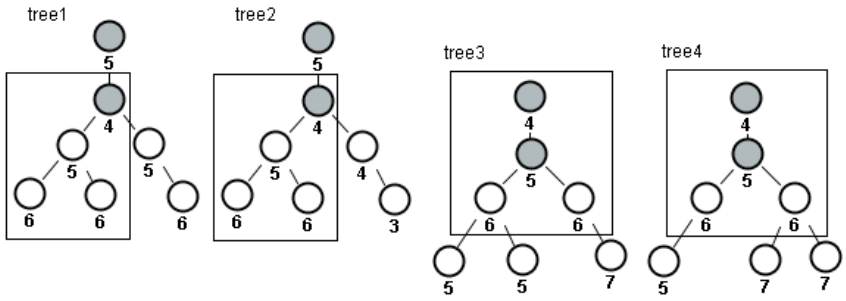

Figure 2: Each of the trees above contains the information necessary to determine the T_{gc} type of the highlighted pair. Numbers denote h-values. The left branch of $tree1$ and $tree2$ can potentially generate children of the same T_{gc} type, illustrated by $tree3$ and $tree4$. Therefore, the pairs (*tree*$_x$,*tree*$_y$) for $(x, y) \in \{(1, 3), (1, 4), (2, 3), (2, 4)\}$ will be in the same super type.

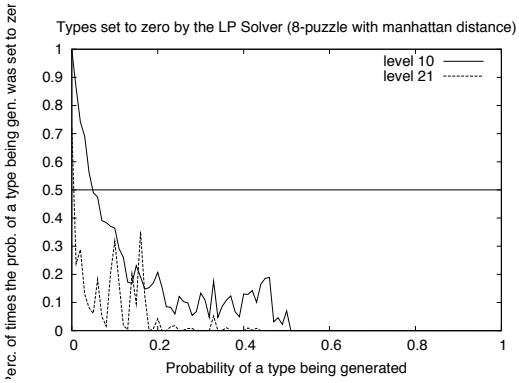

Figure 3: $\hat{\epsilon_i}$ calculation for $i = 10$ and $i = 21$ (8-puzzle with Manhattan distance).

		RRMSE		
d	IDA*	$T_{h,b}$	T_{gc}	ϵ-T_{gc}
17	12.9	0.53	0.44	**0.41**
18	14.5	0.52	0.48	**0.43**
19	22.2	0.51	0.47	**0.40**
20	27.4	0.50	0.50	**0.40**
21	43.3	0.49	0.50	**0.38**
22	58.5	0.48	0.53	**0.37**
23	95.4	0.47	0.54	**0.34**
24	135.7	0.45	0.58	**0.34**
25	226.7	0.44	0.61	**0.30**
26	327.8	0.42	0.65	**0.29**
27	562.0	0.41	0.68	**0.25**
28	818.4	0.39	0.73	**0.24**
29	1,431.7	0.37	0.75	**0.20**

Table 11: 8-puzzle, Inconsistent Heuristic. r=1. RRMSE.

the curve intersects the horizontal y=0.5 line). Note that it can happen that, for some fixed type u, all $\pi(t|u)$-values are below $\hat{\epsilon_i}$. Since we must not set $\pi^i(t|u)$ to zero for all t, we define ϵ_i to be the minimum of $\hat{\epsilon_i}$ and the largest value δ_i such that, for all $u \in T$ there is some $t \in T$ such that $\pi(t|u) > \delta_i$.

Appendix B: Results for RMSE

Tables 11 through 17 show results on the same domains, heuristics, and type systems as used in our experimental section above but in terms of *relative root mean squared error* (RRMSE). The relative root mean squared error is calculated by dividing the sum of the RMSE by the sum of the number of nodes expanded by IDA* for all states in the set of start states. As in the absolute error score, a perfect score for this measure is 0.0. Here we observe the same improvements of ϵ-truncation discussed above in terms of signed and absolute errors.

Acknowledgements

This work was supported by the Laboratory for Computational Discovery at the University of Regina. The authors gratefully acknowledge the research support provided by Alberta Innovates - Technology Futures, the Alberta Ingenuity Centre for Machine Learning (AICML), and Canada's Natural Sciences and Engineering Research Council (NSERC).

References

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. *Advances in Artificial Intelligence (LNAI 1081)* 402–416.

Dinh, H. T.; Russell, A.; and Su, Y. 2007. On the value of good advice: The complexity of A* search with accurate heuristics. In *AAAI-07*, 1140–1145.

Gaschnig, J. 1979. *Performance Measurement and Analysis of Certain Search Algorithms*. Ph.D. Dissertation, Carnegie-Mellon University.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of Iterative-Deepening-A*. *Artif. Intell.* 129(1-2):199–218.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison & Wesley.

Pohl, I. 1977. Practical and theoretical considerations in heuristic search algorithms. *Mach. Intell.* 8:55–72.

Zahavi, U.; Felner, A.; Burch, N.; and Holte, R. C. 2010. Predicting the performance of IDA* using conditional distributions. *J. Artif. Intell. Res.* 37:41–83.

		RRMSE		
d	IDA*	T_h	T_{gc}	ϵ-T_{gc}
Admissible and Consistent Heuristic				
7	1,710.8	0.23	**0.02**	**0.02**
8	12,082.4	0.22	**0.02**	0.03
9	79,650.5	0.21	**0.03**	**0.03**
10	507,640.4	0.22	**0.03**	**0.03**
11	3,449,158.0	0.22	**0.02**	0.03
The heuristic above multiplied by 1.5				
7	528.5	0.39	**0.07**	**0.07**
8	3,740.3	0.37	**0.07**	**0.07**
9	13,924.7	0.38	**0.08**	**0.08**
10	64,611.8	0.40	**0.09**	**0.09**
11	356,366.1	0.46	**0.09**	**0.09**
12	2,397,748.6	0.41	0.10	**0.09**

Table 12: 10-pancake puzzle. r=1. RRMSE.

		RRMSE		
d	IDA*	$T_{h,b}$	T_{gc}	ϵ-T_{gc}
18	134.4	0.04	**0.02**	**0.02**
19	238.4	0.05	**0.02**	**0.02**
20	360.1	0.06	**0.03**	**0.03**
21	630.7	0.06	**0.03**	**0.03**
22	950.6	0.07	0.05	**0.04**
23	1,649.5	0.08	0.06	**0.04**
24	2,457.5	0.09	0.08	**0.05**
25	4,245.5	0.09	0.08	**0.05**
26	6,294.4	0.10	0.11	**0.06**
27	10,994.9	0.10	0.12	**0.06**

Table 13: 8-puzzle, Manhattan Distance. r=10. RRMSE.

		RRMSE		
d	IDA*	$T_{h,b}$	$T_{gc,b}$	ϵ-$T_{gc,b}$
Manhattan Distance				
33	30,461.9	0.03	**0.02**	**0.02**
34	49,576.8	0.04	0.03	**0.02**
35	80,688.2	0.05	0.05	**0.03**
36	127,733.4	0.07	0.06	**0.03**
37	201,822.7	0.09	0.09	**0.05**
38	327,835.3	0.11	0.12	**0.05**
39	478,092.5	0.14	0.16	**0.07**
40	822,055.4	0.18	0.21	**0.09**
41	1,163,312.1	0.22	0.27	**0.11**
42	1,843,732.2	0.28	0.34	**0.14**
Weighted Manhattan Distance (w=1.5)				
33	926.2	0.06	0.02	**0.01**
34	1,286.9	0.07	0.03	**0.02**
35	2,225.6	0.10	0.04	**0.02**
36	2,670.7	0.12	0.05	**0.03**
37	3,519.5	0.15	0.07	**0.04**
38	5,570.8	0.20	0.10	**0.05**
39	6,983.8	0.24	0.14	**0.06**
40	9,103.3	0.30	0.19	**0.07**
41	13,635.3	0.37	0.25	**0.08**
42	16,634.2	0.44	0.31	**0.09**

Table 14: (3x4)-puzzle. r=20. RRMSE.

		RRMSE		
d	IDA*	T_h	T_c	ϵ-T_c
Admissible and Consistent Heuristic (PDB)				
9	1,035.2	0.17	0.08	**0.07**
10	5,547.9	0.17	0.10	**0.08**
11	46,009.3	0.19	0.11	**0.09**
12	322,426.7	0.23	0.14	**0.12**
13	2,480,436.7	0.31	0.19	**0.14**
14	19,583,169.2	0.39	0.23	**0.17**
15	133,596,114.2	0.62	0.34	**0.23**
Same Heuristic, Weighted (w=1.5)				
12	188,177.1	1.06	0.36	**0.27**
13	398,418.8	1.14	0.37	**0.22**
14	3,390,387.6	1.31	0.43	**0.27**
15	6,477,150.7	1.27	0.44	**0.32**
16	16,848,215.1	1.82	0.63	**0.41**

Table 15: 15-pancake puzzle. r=4. RRMSE.

		RRMSE		
d	IDA*	$T_{h,b}$	T_{gc}	ϵ-T_{gc}
48	2,958,898.5	0.12	0.11	**0.06**
49	5,894,396.1	0.14	0.15	**0.07**
50	8,909,564.5	0.17	0.18	**0.09**
51	15,427,786.9	0.16	0.19	**0.09**
52	28,308,808.8	0.26	0.28	**0.15**
53	45,086,452.6	0.24	0.29	**0.14**
54	85,024,463.5	0.37	0.41	**0.23**
55	123,478,361.5	0.36	0.45	**0.25**
56	261,945,964.0	0.44	0.54	**0.30**
57	218,593,372.3	0.44	0.57	**0.32**
58	531,577,032.2	0.47	0.64	**0.37**

Table 16: 15-puzzle. Manhattan Distance. r=25. RRMSE.

		RRMSE		
d	IDA*	$T_{h,b}$	T_c	ϵ-T_c
48	193,396.1	1.01	0.69	**0.44**
49	433,915.3	1.01	**0.46**	0.48
50	562,708.5	1.04	0.89	**0.49**
51	965,792.6	1.09	0.62	**0.50**
52	1,438,694.0	1.20	0.81	**0.55**
53	2,368,940.3	1.33	0.86	**0.57**
54	3,749,519.9	1.63	1.08	**0.71**
55	7,360,297.6	1.68	1.14	**0.74**
56	12,267,171.0	2.17	1.23	**0.72**
57	23,517,650.8	2.44	1.31	**0.83**
58	24,607,970.9	4.73	1.57	**0.98**

Table 17: 15-puzzle. Inconsistent Heuristic. r=1. RRMSE.