Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

ROBERT CLAUSECKER

The Quality of Heuristic Functions for IDA*

ZIB Report 20-17 (Juni 2020)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

The Quality of Heuristic Functions for IDA*

Robert Clausecker
<clausecker@zib.de>
Zuse Institute Berlin

May 2020

Abstract

The aim of this thesis is to deepen our understand of how IDA* heuristics influence the number of nodes expanded during search. To this end, we develop Korf's formula for the number of expanded nodes [6] into a *heuristic quality* η which expresses the quality of a heuristic function as a constant factor on the number of expanded nodes, independent of a particular problem instance.

We proceed to show how to compute η for some common kinds of heuristics and how to estimate η by means of a random sample for arbitrary heuristics. Using the value of η for some concrete examples, we then inspect for which parts of the search space the values of $h(v)$ are particularly critical to the performance of the heuristic, allowing us to build better heuristics for future problems.

This report originally appeared as a master thesis at Humboldt University of Berlin.

1 Notation

We write $A{:}= B$ for A *is defined by* $A = B$. Some other relational symbol may stand in place of $=$ occasionally; for example, we write $x{:}\in S$ to indicate that x is defined to be some element of S. The natural numbers

$$\mathbf{N}{:}= \{0, 1, 2 \ldots\} \tag{1.1}$$

start at 0.

Let $G = (V, E)$ with edge set $E \subset V \times V$ be an undirected graph. In this work, it is assumed that all graphs we consider are connected. Then we write $\sim$ for the *edge relation* such that $u \sim v$ iff $\{u, v\} \in V$. This relation is reflexive and symmetric. The *(open) neighborhood*, denoted by

$$N(v){:}= \{\, u \mid u \in V,\ u \sim v,\ u \neq v \,\}, \tag{1.2}$$

is the set of vertices adjacent to v. It does not contain v itself. Likewise, we write

$$N[v]{:}= \{\, u \mid u \in V,\ u \sim v \,\} = N(v) \cup \{v\} \tag{1.3}$$

for the *closed neighborhood* which is the set of vertices adjacent to v including v. The *closed k-neighborhood* of v with $k \in \mathbf{N}$ is denoted by

$$\begin{aligned} N^0[v]&{:}= \{v\} \\ N^{k+1}[v]&{:}= \{\, u \mid u \in N^k[w],\ w \in N[v] \,\} \end{aligned} \tag{1.4}$$

and is the set of all vertices reachable within k steps from v. This leads us to the definition of the *distance function*

$$d(u, v){:}= \min\{\, n \mid n \in \mathbf{N},\ v \in N^n[u] \,\} \tag{1.5}$$

which yields the length of the shortest path from u to v. As G is assumed to be connected, such a path always exists. Similarly, we define the *open k-neighborhood* of v as

$$N^k(v){:}= \{\, u \mid u \in V,\ d(u, v) = k \,\}, \tag{1.6}$$

which is the set of vertices with distance k to v.

2 Background

In this thesis, we consider the *(single pair) shortest path problem* which is the problem of finding the shortest path v_0, v_1, …, $v_{d(v,z)}$ between two vertices $v = v_0$ and $z = v_{d(v,z)}$ of some graph $G = (V, E)$. Depending on the use case, we are some times interested in finding all shortest paths between the two vertices, or just any one of them.

2.1 Heuristic Functions

The goal vertex z (from German *Ziel*) is generally fixed, allowing us to build powerful *heuristic functions*. These heuristics give us extra information about the distance from an arbitrary vertex w to z. Specifically, a heuristic is a function $h : V \to \mathbf{N}$ such that

$$h(w) \leq d(w, z) \quad \text{and} \quad |h(w) - h(w')| \leq 1 \tag{2.1}$$

for any vertices w and any neighbor $w' \sim w$. The first condition is called *admissability* and states that the value of the heuristic (or *h-value*) is a lower bound for the true distance from w to z. This property tells us when a possible path from v to z going over w would take too long to arrive at z, allowing us to disregard it in the search. The second property, called *consistency,* states that the heuristic does not make sudden jumps in value when following a path. Consistency is a property strictly stronger than admissability in that every consistent heuristic for which $h(z) = 0$ holds is also admissible [7].

Most useful or interesting heuristics are naturally consistent. While inconsistent heuristics can be used with IDA* and enjoy popularity for some applications, they pose additional challenges in their performance analysis. In the rest of this work, we thus assume that all heuristic functions are both admissible and consistent. The focus of this thesis lies on analysing the effectiveness (or *quality*) of such heuristic functions; how they are constructed is out of scope.

2.2 The IDA* Algorithm

In this thesis, the *iterative-deepening A** or IDA* algorithm is used to solve the shortest path problem. IDA*, invented in 1985 by Richard Korf [8], combines the idea of an *iterative-deepening search* (IDS) with the heuristic pruning used by A*.

Like IDS, IDA* performs a sequence of depth-first searches up to a limit d that is incremented after each search until a solution is found. Like in A*, this is combined with a heuristic function $h(v)$: for each node u, we compute the total node cost

$$f(u) = g(u) + h(u) \tag{2.2}$$

which is the sum of the number of steps $g(u)$ taken to reach u and the h-value of u. If this cost exceeds d, no solution starting with the current path will be found within a total of d steps and u needs not be expanded. A description of IDA* for unweighted undirected graphs obtains:

Algorithm 2.1 *The IDA* algorithm [8]. Find a shortest path from v to z using heuristic h.*

1 $found \leftarrow \bot$, $d \leftarrow 0$
2 **until** *found* **do**
  2.1 $found \leftarrow search(v, 0)$
  2.2 $d \leftarrow d + 1$

function $search(v, g)$
Search depth-first for the shortest path from v to z up to distance d having already gone g steps.

1 **if** $v = z$ **then return** $\top$
2 $f \leftarrow g + h(v)$
3 **if** $f > d$ **then return** $\bot$
4 **for each** $v' \in N(v)$ **do**
  4.1 **if** $search(v', g + 1)$ **then return** $\top$
5 **return** $\bot$

As a common optimisation, d can be started at $d = h(v)$ instead of $d = 0$ to eliminate the first few useless iterations. For bipartite graphs, we can replace step 2.2 with $d \leftarrow d + 2$ to ignore iterations that will not land in the correct partition after setting up d to have the correct parity initially. Neither optimisation has an effect on the asymptotic runtime of the algorithm and we do not consider them further.

2.3 Statistical Definitions

For the analysis of IDA*, we need to introduce some statistical definitions. The *equilibrium distribution* of a graph $G = (V, E)$ is the distribution of vertices obtained by random walks through G of infinite length. It is proportional to an eigenvector for the largest eigenvalue of the state transition matrix corresponding to G [6]. We define the *equilibrium distribution weight function* $w(v) : V \to \mathbf{R}^+$ as

$$w(v) := |V| \; \mathrm{P}[\, u = v \,] \quad \text{with} \quad \sum_{v \in V} w(v) = |V|. \tag{2.3}$$

for an equilibrium-distributed random vertex u. This function assigns to each $v \in V$ a weight proportional to the probability of encountering it after an infinite random walk through the graph such that the average weight of each vertex is 1.

Using $w(v)$ we then define for $h(v)$ the *probability mass* and *distribution functions*

$$p(i) := \mathrm{P}[\, h(u) = i \,] = \frac{1}{|V|} \sum_{h(v)=i} w(v) \tag{2.4}$$

$$\text{and} \quad P(i) := \mathrm{P}[\, h(u) \le i \,] = \frac{1}{|V|} \sum_{h(v) \le i} w(v) = \sum_{j=0}^{i} p(j). \tag{2.5}$$

Lastly, we define the *search front size* N_i which is the number of paths of i steps starting in some vertex v. In IDS or IDA* with a trivial heuristic $h(v) = 0$ for all v, this is equal to the number of vertices encountered at depth i.

2.4 Time Complexity of IDA*

Predicting the number of nodes an IDA* search expands, that is, the number of vertices encountered for which $g(v) + h(v) \le d$, was a long-standing problem. The currently best estimate is due to Korf et al. [6] who show that in a search to depth d, the heuristic reduces the search front sizes N_i by a factor of $P(d-i)$ compared to uninformed IDS. Summed up over all i, the number of nodes $E(v, d, P)$ expanded in one round of IDA* searching for the shortest path of up to d steps from v to z is then predicted as

$$E(v, d, P) = \sum_{i=0}^{d} N_i \, P(d - i). \tag{2.6}$$

This estimate holds in the limit of large d when the distribution of nodes at the fringe of the search tree approaches the equilibrium distribution $w(v)$ introduced above. A simple and intuitive proof is given:

Theorem 2.2 *In the limit of large d, the number of expanded nodes $E(v, d, P)$ in one iteration of IDA* searching for the shortest path from v to z up to a length of d using a heuristic with distribution $P(v)$ is as given by Eq. 2.6.*

Proof (following [6]) Consider the search tree of an uninformed depth-first search from v to distance d. By definition of N_i, this tree contains N_i nodes of distance i and for each such node u, we have $g(u) = i$. Of these nodes, all those for which

$$g(u) + h(u) \le d, \quad \text{equally} \quad h(u) \le d - i \tag{2.7}$$

holds are also expanded in the IDA* search tree to depth d: if u was not expanded in the IDA* search tree, there must have been some ancestor w in the path from v to u that was not expanded, as u would have been expanded had it been reached due to Eq. 2.7. Let w be the last such node and let $w' \sim w$ be its successor. We then have

$$g(w') + h(w') \le d \quad \text{but} \quad g(w) + h(w) > d \tag{2.8}$$

with $g(w') = g(w) + 1$ by construction. Subtracting the left inequation from the right one gives us

$$g(w) - g(w') + h(w) - h(w') > d - d \quad \text{and thus} \quad h(w) - h(w') > 1, \tag{2.9}$$

contradicting the consistency of our heuristic h. Hence, w cannot exist and all nodes on the path from v to u are expanded in the IDA* search tree.

As i grows, the distribution of nodes at distance i approaches the equilibrium distribution as the walk of i steps simulates a random walk. Therefore, the probability that Eq. 2.7 holds for any given node at distance i approaches $P(d - i)$ and the number of nodes at distance i that are part of the IDA* search tree approaches $N_i \, P(d - i)$. Summing over all distances then gives us Eq. 2.6 as desired. *q. e. d.*

3 A Measure of Heuristic Quality

The formula by Korf et al. accurately predicts the number of expanded nodes but is specific to a particular vertex v and search depth d and doesn't tell us a lot about the heuristic's impact on the number of expanded nodes in general.

We can obtain a result more useful for this purpose by first following Korf's observation [1] that independently of v, the search front size N_i grows exponentially in i for many interesting search spaces. The base b of this exponential growth is called the *branching factor* of the problem space. Setting

$$N_i := b^i, \tag{3.1}$$

we can then manipulate equation (1.4) into a more useful form:

$$\begin{aligned} E(b,d,P) &= \sum_{i=0}^{d} b^i\, P(d-i) \\ &= \sum_{i=0}^{d} b^{d-i}\, P(i) \\ &= b^d \sum_{i=0}^{d} \frac{P(i)}{b^i} \\ &= b^d \sum_{i=0}^{\infty} \frac{P(i)}{b^i} - b^d \sum_{i=d+1}^{\infty} \frac{P(i)}{b^i}. \end{aligned} \tag{3.2}$$

Using $0 < P(i) \leq 1$ for $i \geq 0$, bounds for the subtrahend obtain:

$$0 < b^d \sum_{i=d+1}^{\infty} \frac{P(i)}{b^i} \leq b^d \sum_{i=d+1}^{\infty} \frac{1}{b^i} = \frac{1}{b} \sum_{i=0}^{\infty} \frac{1}{b^i} = \frac{1}{b-1}, \tag{3.3}$$

and thus establish bounds for $E(b,d,P)$ that show how the number of expanded nodes is equal to b^d with the heuristic merely contributing a constant factor, regardless of what heuristic is used:

$$b^d \sum_{i=0}^{\infty} \frac{P(i)}{b^i} - \frac{1}{b-1} \leq E(b,d,P) < b^d \sum_{i=0}^{\infty} \frac{P(i)}{b^i}. \tag{3.4}$$

For further use, it is useful to express (3.4) in terms of $p(i)$ instead of $P(i)$ as $p(i)$ is often readily available while $P(i)$ needs to be computed by evaluating an extra sum. Using $P(-1) = 0$, we get:

$$\begin{aligned} \frac{b-1}{b} \sum_{i=0}^{\infty} \frac{P(i)}{b^i} &= \sum_{i=0}^{\infty} \frac{P(i)}{b^i} - \sum_{i=0}^{\infty} \frac{P(i)}{b^{i+1}} \\ &= \sum_{i=0}^{\infty} \frac{P(i) - P(i-1)}{b^i} + \frac{P(-1)}{b^0} \\ &= \sum_{i=0}^{\infty} \frac{p(i)}{b^i} \end{aligned} \tag{3.5}$$

and define the *heuristic quality* η, the constant factor by which a given heuristic h reduces the number of expanded nodes compared to an uninformed iterative deepening search, as

$$\eta := \sum_{i=0}^{\infty} \frac{p(i)}{b^i} = \frac{b-1}{b} \sum_{i=0}^{\infty} \frac{P(i)}{b^i} = |V|^{-1} \sum_{v \in V} \frac{w(v)}{b^{h(v)}}. \tag{3.6}$$

We then restate equation (3.4) in terms of η:

$$\frac{b}{b-1} b^d \eta - \frac{1}{b-1} \leq E(b,d,P) < \frac{b}{b-1} b^d \eta. \tag{3.7}$$

Intuitively, b^d is the number of nodes at depth d of an uninformed depth-first search to depth d, $\frac{b}{b-1}$ accounts for the inner nodes in the search tree and η is the constant factor by which the heuristic reduces the number of expanded nodes. We decided to keep the factor $\frac{b}{b-1}$ separate from η to keep η normalised in the sense that $\eta = 1$ for a trivial heuristic h with $h(v) = 0$ everywhere. Another way to see η is obtained by viewing the heuristic $h(v)$ as reducing the depth of the search problem from d to an *effective depth*

$$d' = d - \log_b(1/\eta). \tag{3.8}$$

3.1 General Properties

3.1.1 Sums of Heuristics Given two stochastically independent heuristics h_a and h_b with qualities η_a and η_b, the quality η_{a+b} of their sum $h_{a+b}(v) = h_a(v) + h_b(v)$ can be derived using the Cauchy product.

$$\begin{aligned}\eta_{a+b} &= \sum_{i=0}^{\infty} \frac{p_{a+b}(i)}{b^i} \\ &= \sum_{i=0}^{\infty} \sum_{j=0}^{i} \frac{p_a(j)\, p_b(i-j)}{b^i} \\ &= \left(\sum_{i=0}^{\infty} \frac{p_a(i)}{b^i}\right) \left(\sum_{i=0}^{\infty} \frac{p_b(i)}{b^i}\right) \\ &= \eta_a \eta_b\end{aligned} \tag{3.9}$$

Note that this identity is of limited use as the h values given by different heuristics for the same configuration are generally far from independent. For example, the pattern databases forming partitioning (a) from Fig. 5.1, have qualities 2.164×10^{-6} (top left) and 1.871×10^{-6} (others). The product of these four qualities is 1.417×10^{-23}, missing the actual quality $\eta = 4.562 \times 10^{-22}$ by a factor of 32.2.

3.1.2 Maxima of Heuristics While the quality of a heuristic $h_{\max(a,b)}(v) = \max\big(h_a(v), h_b(v)\big)$ defined as the maximum of two stochastically independent heuristics h_a and h_b cannot be computed from just η_a and η_b, we can observe that

$$P_{\max(a,b)} = \mathrm{P}[\, \max\big(h_a(v), h_b(v)\big) \leq i\,] = \mathrm{P}[\, h_a(v) \leq i \wedge h_b(v) \leq i\,] = P_a(i)\, P_b(i), \tag{3.10}$$

allowing us to compute $\eta_{\max(a,b)}$ using Eq. 3.6, knowing just P_a and P_b.

$$\eta_{\max(a,b)} = \frac{b-1}{b} \sum_{i=0}^{\infty} \frac{P_a(i)\, P_b(i)}{b^i} \tag{3.11}$$

However, the same caveat as in §3.1.1 applies: heuristics are generally not statistically independent, generally causing this formula to underestimate $\eta_{\max(a,b)}$ significantly.

3.1.3 Partial Heuristics Some heuristics can be modelled as giving the value of some heuristic $\hat{h}$ for some part $\hat{V} \subset V$ of the search space and a fixed value k everywhere else. Let $\hat{\eta}$ be the quality of $\hat{h}$ computed on $\hat{V}$ and $\alpha = |A|/|V|$ be the share of vertices in $\hat{V}$, then the quality of

$$h(v) = \begin{cases} \hat{h}(v) & \text{if } v \in \hat{V} \\ k & \text{otherwise} \end{cases} \tag{3.12}$$

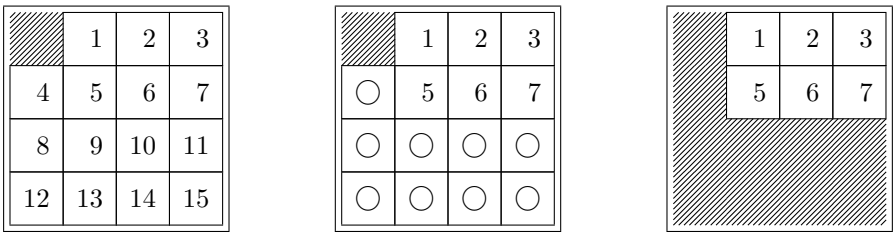
is given by

$$\eta = \alpha\hat{\eta} + (1-\alpha)b^{-k}. \tag{3.13}$$

A useful example of such a partial heuristic is the heuristic

$$h(v) = \begin{cases} d(v,z) & \text{if } d(v,z) \leq k \\ k & \text{otherwise} \end{cases} \tag{3.14}$$

that returns the true distance to the goal z for a k-neighbourhood around it and k for every vertex outside. If this neighborhood is very small compared to the size of the search space, we have $\eta \approx b^{-k}$, showing that such a heuristic reduces the complexity of the search to that of an uninformed search of a vertex k nodes closer to the target.



15 puzzle non-additive PDB additive PDB

Figure 3.1 *the solved configuration of the 15 puzzle as seen by the non-additive and the additive* $\{1, 2, 3, 5, 6, 7\}$ *pattern database. While the non-additive PDB admits* 2 *moves out of this configuration, the additive PDB has* 5 *moves, reflecting its higher branching factor* $B > b$.

3.2 The Quality of Pattern Databases

Of particular interest are *pattern database* or PDB heuristics [2][3]. The general idea behind PDBs is to abstract the problem into a simpler problem by ignoring some parts of its state. This yields a problem with a search space so small that the distance from each abstracted vertex v' to the abstracted solved configuration z' can be tabulated. If the abstraction is chosen such that each legal move in the original problem corresponds to a legal move in the abstracted problem, these distances form an admissible and consistent heuristic.

As an example, consider Fig. 3.1 for two ways to abstract the *15 puzzle* [9], a common benchmark problem for heuristic search. On the left, you can see the solved state of the 15 puzzle. Following the original Culberson/Schaeffer paper [2], the puzzle is abstracted by ignoring the identity of tiles 4 and 8–15, reducing the search space by a factor of 9! This approach was improved by Korf and Felner [3] into an *additive PDB* shown on the right, where we pretend the tiles 4 and 8–15 do not exist, allowing any moves into the hatched space. While this generally leads to a worse heuristic quality, we gain *additivity*, the ability to add the h values of PDBs for disjoint tile sets, leading to much better h values with acceptable memory consumption through the combination of several small additive PDBs.

3.2.1 Definitions Glossing over the various kinds of pattern databases, we define a *table-based heuristic* of size s as a pair of an *index function* $idx(v) : V \to \{0, 1, \ldots, s-1\}$ projecting the vertex set V to table entries and a *lookup table* $tbl[e] : \{0, 1 \ldots, s-1\} \to \mathbf{N}$ such that $h(v) = tbl[idx(v)]$ is a heuristic function. We also define the *inverse index function*

$$idx^{-1}(e) = \{\, v \mid v \in V,\ idx(v) = e \,\} \tag{3.15}$$

mapping a table index back to the set of vertices it describes.

3.2.2 Computing the Quality of Table-Driven Heuristics Computing η is easy for table-based heuristics as $p(i)$ can be found by taking an appropriately weighted histogram over tbl. To do so, we first define the *index weight* $w_{idx}(e)$ which is the sum of the equilibrium-distribution weights of all $v \in V$ for which $idx(v) = e$ is again normalised such that $w_{idx}(e)$ for all $e \in \{0, 1, \ldots, s-1\}$ sum to s.

$$w_{idx}(e) = \frac{s}{|V|} \sum_{v \in idx^{-1}(e)} w(v) \quad \text{with} \quad \sum_{e=0}^{s-1} w_{idx}(e) = s \tag{3.16}$$

With this in hand, we can compute the probability mass function

$$p(i) = s^{-1} \sum_{tbl[e]=i} w_{idx}(e) \tag{3.17}$$

or obtain η directly:

$$\eta = s^{-1} \sum_{e=0}^{s-1} \frac{w_{idx}(e)}{b^{tbl[e]}}. \tag{3.18}$$

3.2.3 Non-additive PDBs In a performance analysis due to Korf [1], *pattern database* (PDB) heuristics are modeled as subspaces of the problem space where each vertex in the subspace corresponds to an equal number of vertices in the problem space. Furthermore, the subspace described by the PDB is modeled as having the same branching factor b as the problem space such that the subspace is made up of $k+1$ classes of vertices with distances $i = 0, 1, \ldots, k$ and b^i vertices each.

This accurately describes some kinds of *non-additive PDBs* like those for sliding tile puzzles described by Culberson and Schaeffer [2] but fails for *additive PDBs* such as those for sliding tile puzzles [3] or gained by operator partitioning [4] due to the differing branching factors.

Following this model, a pattern database with size

$$s = \sum_{i=0}^{k} b^i = \frac{b^{k+1} - 1}{b - 1}, \tag{3.19}$$

$$\begin{aligned} k &= \log_b\big(s(b-1)+1\big) - 1 \\ &= \log_b s + \log_b(b-1) - 1 + \mathcal{O}(s^{-1}) \end{aligned} \tag{3.20}$$

has probability mass function

$$p(i) = \begin{cases} b^i/s & \text{if } 0 \le i \le k \\ 0 & \text{otherwise,} \end{cases} \tag{3.21}$$

and thus quality

$$\begin{aligned} \eta &= \sum_{i=0}^{k} \frac{b^i/s}{b^i} = \frac{k+1}{s} \\ &= \frac{\log_b\big(s(b-1)+1\big)}{s} \\ &= \frac{\log_b s + \log_b(b-1)}{s} + \mathcal{O}(s^{-2}) \end{aligned} \tag{3.22}$$

showing that the pruning power of a PDB following this model is proportional to its size by the logarithm of its size. Korf's paper gets the slightly different result

$$\eta = \frac{\log_b s + \log_b(b-1) + (b-1)^{-1}}{s} \approx \frac{\log_b s + 1}{s} \tag{3.23}$$

where the extra $(b-1)^{-1}$ term is an artifact of some coarser approximations in his derivation.

3.2.4 Additive PDBs As an extension to Korf's model, it is useful to consider PDBs with a subspace branching factor B strictly larger* than the search space branching factor b. This gives a heuristic with

$$s = \sum_{i=0}^{k} B^i = \frac{B^{k+1} - 1}{B - 1}, \tag{3.24}$$

$$\begin{aligned} k &= \log_B\big(s(B-1)+1\big) - 1 \\ &= \log_B s + \log_B(B-1) - 1 + \mathcal{O}(s^{-1}), \end{aligned} \tag{3.25}$$

$$\text{and} \quad p(i) = \begin{cases} B^i/s & \text{if } 0 \le i \le k \\ 0 & \text{otherwise.} \end{cases} \tag{3.26}$$

* $B < b$ cannot occur for admissible heuristics.

The quality is somewhat messy to derive. Note that as $B > b$, we have $\log_b B > 1$.

$$\begin{aligned}
\eta &= \sum_{i=0}^{k} \frac{B^i/s}{b^i} = \frac{1}{s} \sum_{i=0}^{k} \left(\frac{B}{b}\right)^i \\
&= \frac{(B/b)^{k+1} - 1}{(B/b-1)s} = \frac{b}{(B-b)s}\left(\left(\frac{B}{b}\right)^{k+1} - 1\right) \\
&= \frac{b}{(B-b)s}\left(\left(\frac{B}{b}\right)^{\log_B\left((B-1)s+1\right)} - 1\right) \\
&= \frac{b}{(B-b)s}\left(\left((B-1)s+1\right)^{\log_B(B/b)} - 1\right) \\
&= \frac{b}{(B-b)s}\left(\left((B-1)s+1\right)^{1-1/\log_b B} - 1\right) \\
&= \frac{b}{(B-b)s}\left(\frac{(B-1)s+1}{\sqrt[\log_b B]{(B-1)s+1}} - 1\right) \\
&= \frac{b}{B-b}\left(\frac{B-1+s^{-1}}{\sqrt[\log_b B]{(B-1)s+1}} - \frac{1}{s}\right) \\
&= \frac{b(B-1)}{B-b}\frac{1}{\sqrt[\log_b B]{(B-1)s+1}} - \mathcal{O}(s^{-1})
\end{aligned} \tag{3.27}$$

Crucially, this shows that the pruning power of an additive PDB grows proportional to some root of its size, giving us diminished returns when increasing the PDB size that become meagerer the larger B grows.

4 Sampling Heuristics

While counting table entries is a good choice for heuristics driven by a single table, we'd really like to have a general method to estimate η for arbitrary heuristics. An obvious choice is to take a uniform sample of the heuristic over the search space and then compute η using the measured $p(i)$ values. If we see η as the expected value

$$\eta = \mathrm{E}[b^{-h(v)}], \tag{4.1}$$

of an equilibrium-distributed vertex v, we can estimate η by uniformly sampling random vertices from the graph. Using this method η has variance

$$\begin{aligned}\sigma^2 &= \frac{1}{|V|}\sum_{v\in V} w(v)\,(b^{-h(v)} - \eta)^2 \\ &= \sum_{i=0}^{\infty} p(i)\,(b^{-i} - \eta)^2 \\ &= \sum_{i=0}^{\infty} \frac{p(i)}{b^{2i}} - \eta^2.\end{aligned} \tag{4.2}$$

Plugging in $p(i)$ as observed in our uniform random sample and corrected for the equilibrium distribution, we can then compute the standard error

$$\sigma_\eta = \sqrt{\sigma^2/n} \tag{4.3}$$

for a random uniform sample with n samples, telling us if we gathered enough samples to estimate η to a satisfying degree.

It turns out that for large search spaces such as the 24 puzzle's with $|V| = 7.76\times10^{24}$, no satisfying estimate is gained even after taking more than a billion samples. For example, for the heuristic derived from partitioning (a) of Fig. 5.1, we get $\eta = 4.562\times10^{-22}$ with a standard deviation of $\sigma = 4.590\times10^{-13}$. To estimate that quality to just one significant digit (i. e. a relative error of less than 10 % at 95 % confidence), one would need unreasonable $4\sigma^2/(10\,\%\cdot\eta)^2 = 4.050\times10^{20}$ samples, making a direct uniform sample infeasible in practice.

Intuitively, this is because $\mathrm{E}[b^{-h(v)}]$ is dominated by the necessarily small h values of the vertices around z. A uniform sample is very unlikely to ever hit this region and thus gets a skewed result that can be wrong by orders of magnitude.

4.1 Sphere Stratified Sampling

As a solution to this problem, we observe that h value strongly correlates with distance to z and estimate η by means of a *stratified sample* where each stratum $V_0, V_1, \ldots, V_{L-1}$ is a *sphere*

$$V_k := \{\, v \mid v \in V,\ d(v,z) = k \,\} = N^k(z) \tag{4.4}$$

centered around z containg the vertices whose shortest path to z is k steps long. As the aforementioned critical neighborhood around z is captured in the nearest few spheres, we can increase the sample size for their strata to estimate η precisely.

4.1.1 Statistical Properties Let η_k be the expected value and σ_k^2 be the variance of $\mathrm{E}[b^{-h(v)}]$ in stratum V_k. We can find η and σ^2 as

$$\eta = \sum_{k=0}^{L-1} \frac{|V_k|}{|V|}\eta_k \tag{4.5}$$

$$\text{and}\quad \sigma^2 = \sum_{k=0}^{L-1} \frac{|V_k|}{|V|}\big(\sigma_k^2 + (\eta_k - \eta)^2\big). \tag{4.6}$$

If we draw n_k samples from each V_k, giving us $n = n_0 + n_1 + \ldots + n_{L-1}$ samples in total, the squared standard error is then given by

$$\sigma_\eta^2 = \sum_{k=0}^{L-1} \frac{|V_k| - n_k}{|V_k| - 1}\left(\frac{|V_k|}{|V|}\right)^2 \frac{\sigma_k^2}{n_k}. \tag{4.7}$$

Ideally, each sample size n_k should thus be chosen in proportion to σ_k^2.

4.2 Sampling Spheres with Random Walks

For small k, samples can be taken from V_k by enumerating all vertices in V_k by means of a breadth-first search and then taking a uniform random sample from those vertices. For larger k, this method becomes first inconvenient and then quickly impossible as the size of V_k rapidly exceeds available storage. Fortunately, a different approach allows us to take samples from V_k without having to enumerate the vertices in the sphere.

4.2.1 Taking Samples To take a sample from V_k we first perform a random walk of k steps starting out from $v_0 = z$. At each step i of the random walk, we draw a uniform random vertex

$$\begin{aligned} v_0 &:= z \\ v_i &:\in N'(v_0, v_1, \ldots, v_{i-1}) \end{aligned} \tag{4.8}$$

where $N'(v_0, v_1, \ldots, v_{i-1})$ is the neighborhood of v_{i-1} pruned in some way such that at least one path to each $v_i \in V_i$ remains. A detailed discussion of the pruning methods used is found in Appendix A.

Once the final vertex v_k of the walk has been picked, we compute $d(z, v)$ (e. g. using IDA*) and accept $v = v_k$ as a sample if $d(z, v_k) = k$. If at any point in the random walk $N'(v_0, v_1, \ldots, v_i)$ is empty (i. e. we entered a cul-de-sac, making completion of the walk impossible) or if $d(z, v_k) < k$, the sample is instead rejected. We keep track of the yield

$$y := \mathrm{P}[\, d(z, v_k) = k \,] = \frac{n_{\text{accepted}}}{n_{\text{samples}}}, \tag{4.9}$$

which is the measured probability of a vertex v_k being accepted as a sample, for future use. The yield falls as k grows because at each v_i, there is a chance to pick a vertex $v_i \notin V_i$ causing rejection, and the longer the walk is, the more likely it is for this to happen. This effect can be counteracted by an effective pruning rule $N'(v_0, \ldots, v_i)$, keeping the yield at an acceptable level.

4.2.2 Computing the Bias While this method allows us to rapidly generate samples, the samples are far from being unbiased. Luckily, the exact bias of each sample can be determined by information we already possess, allowing us to compensate for the bias when processing the samples.

Let $S(v, k) \subset V^{k+1}$ be the set of all paths $v_0, v_1, \ldots, v_k$ from $v_0 = z$ to some sample $v = v_k$ with $d(z, v) = k$ by which we could have reached v according to the method explained in the previous section, i. e.

$$S(v, k) = \{\, (v_0, v_1, \ldots, v_k) \mid v_0 = z,\ v_k = v,\ v_i \in N'(v_0, \ldots, v_{i-1}) \text{ for } i = 1, 2, \ldots, k \,\}. \tag{4.10}$$

This set is obtained as a side effect of computing $d(z, v)$ during the accepting step by taking the set of all shortest paths from z to v computed by IDA* and removing all paths from it that would have been pruned by $N'(v_0, \ldots, v_i)$.

Using $S(v, k)$, we first compute the a priori probability of having reached v in a random walk with k steps from z as the sum of the probabilities of having reached v through any of the paths in $S(v, k)$:

$$\mathrm{P}[\, v_k = v \,] = \sum_{\substack{(v_0, \ldots, v_k) \\ \in S(v,k)}} \prod_{i=0}^{k-1} |N'(v_0, \ldots, v_i)|^{-1}. \tag{4.11}$$

Then, we divide by y to get the probability $p_k(v)$ of having accepted v as a sample of V_k:

$$p_k(v) := \mathrm{P}[\, v_k = v \mid d(z, v_k) = k \,] = \frac{\mathrm{P}[\, v_k = v \cap d(z, v_k) = k \,]}{\mathrm{P}[\, d(z, v_k) = k \,]} = \frac{\mathrm{P}[\, v_k = v \,]}{y}. \tag{4.12}$$

After computing the size of V_k as shown in the next section, we multiply with that size to find the sampling bias weight $\beta(v)$:

$$\beta(v) := |V_k|\, p_k(v) = \frac{|V_k|}{y} \sum_{\substack{(v_0, \ldots, v_k) \\ \in S(v,k)}} \prod_{i=0}^{k-1} |N'(v_0, \ldots, v_i)|^{-1}. \tag{4.13}$$

4.2.3 Estimating the Stratum/Sphere Size Both for computing $\beta(v)$ and for weighting the individual strata when computing the expected value of η, the number of elements in each V_k is needed. As with sampling values from V_k, the size can only be computed for small k by enumerating all vertices. As k grows, memory constraints rapidly render this method impossible.

Luckily, we can easily compute $|V_k|$ using the random walks we sampled. Recall that as $p_k(v)$ is the probability mass function of having chosen v out of V_k, its arithmetic mean

$$\bar{p}_k = \frac{\sum_{v \in V_k} p_k(v)}{|V_k|} = \frac{1}{|V_k|} \tag{4.14}$$

is equal to the reciprocal of the desired quantity. As our samples are biased, we cannot directly compute $\bar{p}_k$ by averaging $p_k(v)$ for the samples we drew. Instead, we observe that the bias $\beta(v)$ is proportional to the quantity $p_k(v)$ we are interested in, making our sample a *size biased sample*. Applying a theorem by Cox [11], the expected value of the reciprocal of a size-biased quantity is equal to the reciprocal of the arithmetic mean of the quantity. Thus we get

$$\mathrm{E}[p_k(v)^{-1}] = \sum_{v \in V_k} p_k(v)\, p_k(v)^{-1} = \sum_{v \in V_k} 1 = |V_k|, \tag{4.15}$$

allowing us to estimate $|V_k|$ as the expected value of $p_k(v)^{-1}$.

4.2.4 Estimating the Quality within a Stratum Given a biased sample of a stratum $\tilde{V}_k \subseteq V_k$ with known sampling bias $\beta(v)$, we can estimate η_k and σ_k, the expected value and standard deviation of η within $\tilde{V}_k$. To compensate for the sampling bias introduced by sampling from random walks, each sample $v \in \tilde{V}_k$ is weighted with $\beta(v)^{-1}$:

$$\eta_k = \frac{1}{|\tilde{V}_k|} \sum_{v \in \tilde{V}_k} \frac{b^{-h(v)}}{\beta(v)} \tag{4.16}$$

$$\text{and} \quad \sigma_k^2 = \frac{1}{|\tilde{V}_k|} \sum_{v \in \tilde{V}_k} \frac{\left(\eta_k - b^{-h(v)}\right)^2}{\beta(v)}. \tag{4.17}$$

4.3 The Rest of the Graph

As k grows, drawing samples from V_k becomes more and more difficult due to the falling yields (cf. appendices A and B) and exponentially growing cost of finding the shortest paths from v_k back to z. At the same time, the influence of V_k on the value of η falls.

To avoid having to draw samples from these distant spheres, it has proven to be useful to use spheres for the first $L-1$ strata for some L with the last stratum V_{L-1} being the rest of the graph.

$$V_k \colon = \begin{cases} N^k(z) & \text{if } k < L-1 \\ V_{\geq k} & \text{otherwise} \end{cases} \tag{4.18}$$

$$\text{where} \quad V_{\geq k} \colon = V \setminus \bigcup_{i=0}^{k-1} N^k(z) \tag{4.19}$$

Samples can be drawn from $V_{\geq k}$ by uniformly drawing a sample v from all of V and then proving that its distance to z is no less than k by running a partial IDA* search on v until $f \geq k$, proving that no solution with a length of less than $f \geq k$ exists. The size of $V_{\geq k}$ is simply determined by

$$|V_{\geq k}| = |V| - \sum_{i=0}^{k-1} |N^k(z)|. \tag{4.20}$$

4.4 Algorithm Summary

From the ideas outlined in the previous sections, we distill an algorithm for estimating η.

First, samples are drawn from the graph according to the methods outlined in §4. For the first few strata, the entire stratum is enumerated using a breadth-first search and samples are drawn using an inside-out Fisher-Yates algorithm [12]. Samples for the next strata til the penultimate one are drawn using the random walk approach from §4.2. This sampling method is summarised in Algorithm 4.1. Finally, samples for V_{L-1}, the rest of the graph, are drawn uniformly from the entire graph as explained in §4.3, rejecting those that would fall into earlier strata. Being uniform samples, each sample drawn by the breadth-first search method and the method from §4.3 within a stratum V_k has the same sampling probability $p_k(v) = |V_k|^{-1}$.

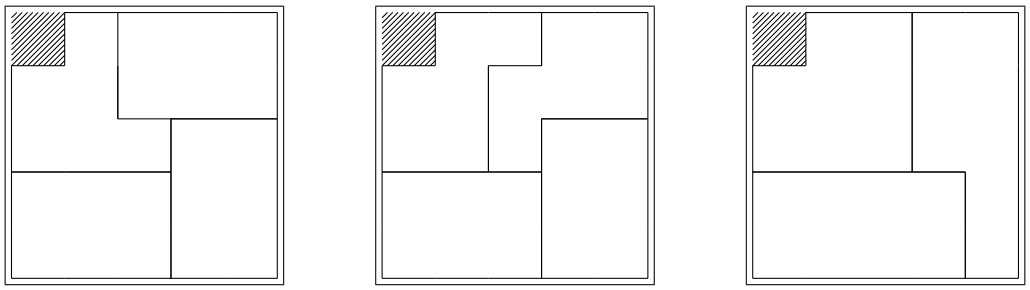
Algorithm 4.1 *Using a pruning function $N'(v_0, \ldots, v_k)$, sample n vertices from V_k with known bias values $\beta(v)$. Return a multiset of samples $\tilde{V}_k$ with n elements, a bias function $\beta : \tilde{V}_k \to \mathbf{R}^+$ and an estimation $\widetilde{|V_k|}$ of $|V_k|$.*

1 $\tilde{V}_k \leftarrow \{\}, \quad v_0 \leftarrow z$
2 **do** n **times**
  2.1 **for** $i = 1, \ldots, k$ **do** uniformly pick $v_i \in N'(v_0, \ldots, v_{i-1})$ — see §4.2.1
  2.2 find all shortest paths from v_0 to v_k
  2.3 **if** $d(v_0, v_k) \neq k$ **then** continue with the next iteration — reject $v_k \notin V_k$
  2.4 compute $S(v_k, k)$ by pruning the shortest paths from step 2.2 with $N'(v_0, \ldots, v_i)$ — see Eq. 4.10
  2.5 $\tilde{V}_k \leftarrow \tilde{V}_k \cup \{v_k\}$
  2.6 $P(v_k) \leftarrow \sum_{(v_0,\ldots,v_k)\in S(v_k,k)} \prod_{i=0}^{k-1} \left|N'(v_0, \ldots, v_i)\right|^{-1}$ — see Eq. 4.11
3 $\widetilde{|V_k|} \leftarrow n^{-1} \sum_{v\in\tilde{V}_k} P(v)^{-1}$ — see Eq. 4.15
4 $y \leftarrow |\tilde{V}_k|/n$ — see Eq. 4.9
5 **for each** $v \in \tilde{V}_k$ **do**
  5.1 $p_k(v) \leftarrow P(v)/y$ — see Eq. 4.12
  5.2 $\beta(v) \leftarrow \widetilde{|V_k|}\, p_k(v)$ — see Eq. 4.13
6 **return** $(\tilde{V}_k, \beta(v), \widetilde{|V_k|})$

Once a set of samples are computed, we can compute η for any heuristic $h(v)$. The stratum quality η_k and its corresponding standard deviations σ_k are evaluated for each stratum $k = 0, 1, \ldots, L-1$ according to Eq. 4.16 and Eq. 4.17. These values are then combined according to Eq. 4.5, Eq. 4.6, and Eq. 4.7 to find the quality η, the standard deviation σ and the standard error σ_η, allowing us to ascertain the precision of the estimate.

5 Results

In this section, the heuristic quality η and the sphere sampling process are illustrated using the *24 puzzle* [9] as a model problem and heuristics based on *ZPDB heuristics* [7], an improved variant of *additive pattern databases* [3] as model heuristics.



(a) Korf et al. (b) Clausecker (c) Döbbelin et al.

Figure 5.1 *Three partitonings of the 24 puzzle's tiles into pattern databases. (a) from Korf et al. [3], (b) from my previous work [7], and (c) from Döbbelin et al. [14].*

5.1 Examples

To illustrate the effectiveness of various heuristic schemes, Table 5.1 shows the quality η of some interesting heuristics for the 24 puzzle. The heuristics shown are the Manhattan heuristic [8], ZPDB heuristics [7] based on two 6-6-6-6 partitionings [3][7] and one 8-8-8 partitioning [14] as well as the two ZPDB catalogues (from [7], reproduced in Appendix C). Where applicable, the effectiveness of transposition on the quality of the heuristic is shown as well. With transposition, a heuristic is computed both for the configuration and its transposition along the main diagonal, taking the maximum of the two values.

The quality was determined using the sphere stratified sampling procedure from §4.1 with up to 10^7 samples for each of the spheres V_0 to V_{64} and 10^8 samples for sphere $V_{\geq 65}$, representing the rest of the graph. For the Manhattan heuristic, sphere $V_{\geq 65}$ is covered by 10^9 samples instead. As seen in Fig. 5.3, this is needed to reduce the error from about 20 % to 9.08 % because our sphere samples don't go far enough to capture the entire critical region of this heuristic, leaving a large part of η to $V_{\geq 65}$.

These results give a number of interesting insights into the various heuristics. First, it shows how the Manhattan heuristic's quality is 220 times worse than that of partitioning (a), the next best heuristic considered. Thus, we can expect partitioning (a) to expand on average 220 times less vertices than the Manhattan heuristic when solving difficult problem instances.

We also see how use of transposition generally improves the quality by more than a factor of 2, offsetting the need for the double lookup. An exception to this is partitioning (c) whose almost symmetrical structure likely causes the transposed puzzle's h-value to be very close to the original puzzle's h-value.

Lastly, we can see how the small catalogue of 7 partitionings and 14 ZPDBs outperforms the 8-8-8 partitioning slightly without transposition and much more strongly with transposition. This is contrasted with the large catalogue of 14 partitionings made of 20 ZPDBs which doesn't manage to be all that better than the small catalogue, highlighting the diminished returns from adding more partitionings to a catalogue.

heuristic	*without transposition*		*with transposition*	
Manhattan	$9.926 \times 10^{-20} \pm 9.013 \times 10^{-21}$	(9.08 %)	—	
partitioning (a)	$4.562 \times 10^{-22} \pm 3.335 \times 10^{-24}$	(0.73 %)	$1.359 \times 10^{-22} \pm 5.342 \times 10^{-25}$	(0.39 %)
partitioning (b)	$4.391 \times 10^{-22} \pm 9.346 \times 10^{-24}$	(2.13 %)	$1.611 \times 10^{-22} \pm 9.133 \times 10^{-25}$	(0.57 %)
partitioning (c)	$1.097 \times 10^{-22} \pm 5.430 \times 10^{-25}$	(0.49 %)	$6.780 \times 10^{-23} \pm 2.260 \times 10^{-25}$	(0.33 %)
small catalogue	$8.548 \times 10^{-23} \pm 4.254 \times 10^{-25}$	(0.50 %)	$3.787 \times 10^{-23} \pm 1.316 \times 10^{-25}$	(0.35 %)
large catalogue	$6.751 \times 10^{-23} \pm 4.768 \times 10^{-25}$	(0.71 %)	$3.208 \times 10^{-23} \pm 7.784 \times 10^{-26}$	(0.24 %)

Table 5.1 *The quality of some interesting heuristics, determined with and without transposition search. For each heuristic, a 95% confidence interval is given.*

5.2 Heuristic Quality By Sphere

In their 2004 paper *Multiple Pattern Databases*, Holte et al. propose that eliminating small h values is most important in improving the quality of heuristics and that using the maximum of many small pattern databases is more effective than using few large ones because while large PDBs provide higher h values for difficult configurations, collections of small PDBs are able to provide consistently good h values for easy configurations [13].

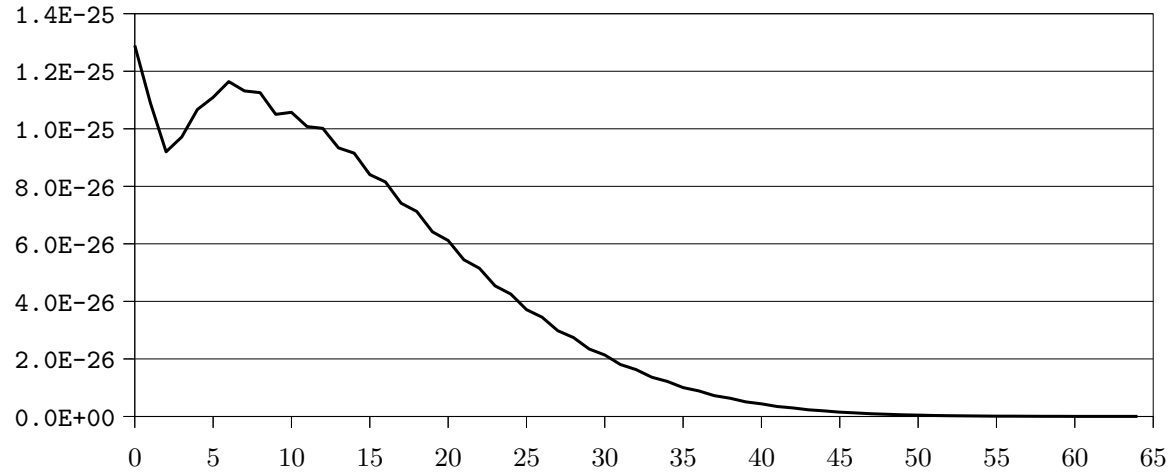


Figure 5.2 *Quality histogram of contribution to the perfect heuristic h_{perfect} by distance from the solved configuration; lowest achievable values.*

Sphere stratified sampling provides us with the tools needed to test and expand this idea. By plotting how much each sphere of the search space contributes to η, we obtain a *quality histogram* that tells us which parts of the search space contribute how much to the total number of expanded vertices. As a baseline, we use the quality of the hypothetical perfect heuristic $h_{\text{perfect}}(v) = d(v, z)$ given by

$$\begin{aligned} \eta_{\text{perfect}} &= \sum_{i=0}^{\infty} \frac{p(i)}{b^i} \\ &= \sum_{i=0}^{\infty} \frac{|V_i|/|V|}{b^i} \\ &= 2.5063 \times 10^{-24}. \end{aligned} \tag{5.1}$$

Fig. 5.2 shows η_{perfect} plotted by the contribution of each sphere. As no consistent heuristic can surpass h_{perfect} in accuracy, no heuristic's histogram can dip below the baseline given by Fig. 5.2, making it a useful reference for possible room of improvement.

The shape of this curve can be explained by $|V_i|$ growing slower than b^i as not all moves possible in a given configuration bring us further away from z. The initial spike in the graph is the consequence of the solved configuration being a single state with the blank tile in the corner and thus starting with a lower than average branching factor. As the distribution of states within a sphere gradually approaches the equilibrium distribution, the curve becomes smooth.

In Fig. 5.3, 5.4, and 5.5, the quality histograms of some of the heuristics listed in Tbl. 5.1 are shown. For sufficiently good heuristics, it can be seen how the *critical region* between V_{30} and V_{60} accounts for most of the value of η. Intuitively, this can be explained through the interaction of two behaviours: (a) the closer we are to z, (i. e. the lower the sphere number is), the better our heuristic is able to approximate h_{perfect}. Hence, the contribution to η rises as the sphere number rises. (b) the farther we are from z, the more does the b^{-i} weighting pull down our curve. As b^i grows faster than the sphere size $|V_i|$ (cf. Fig. 5.2), more so once the sphere size growth stagnates as we move towards the middle of the graph, the contribution to η falls as the sphere number rises. Between these two processes, there is an interval where the contribution peaks, falling off to both sides.

From this insight, we conclude that when optimising heuristics under memory usage or other constraints, particular attention should be paid to the h values of vertices in this critical region. For example, one could build pattern databases that provide large tables for partial configurations likely to belong to vertices in the critical region and small tables for configurations that are likely too strongly permuted to lie in the critical region.

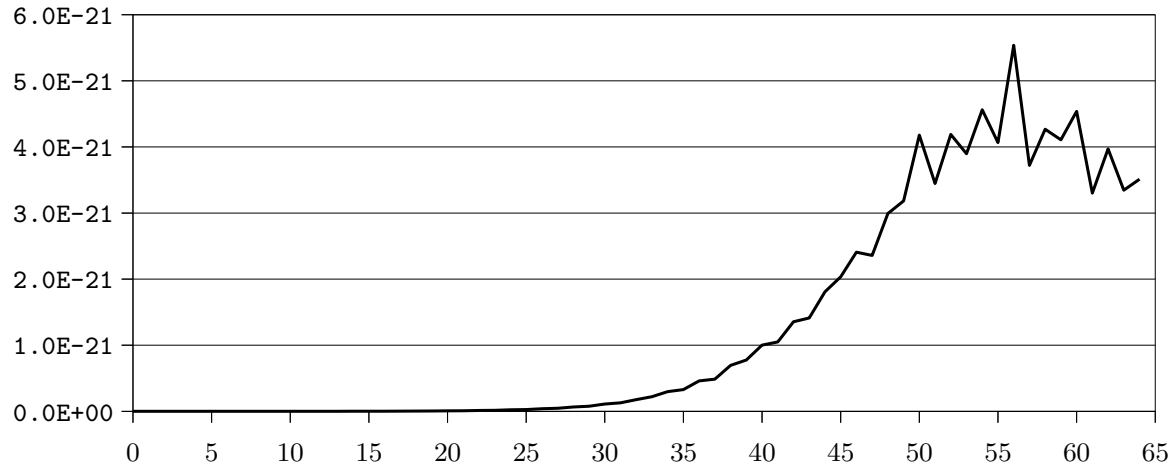


Figure 5.3 *Manhattan heuristic quality histogram.* $\eta_{\geq 65} = 2.364 \times 10^{-20}$

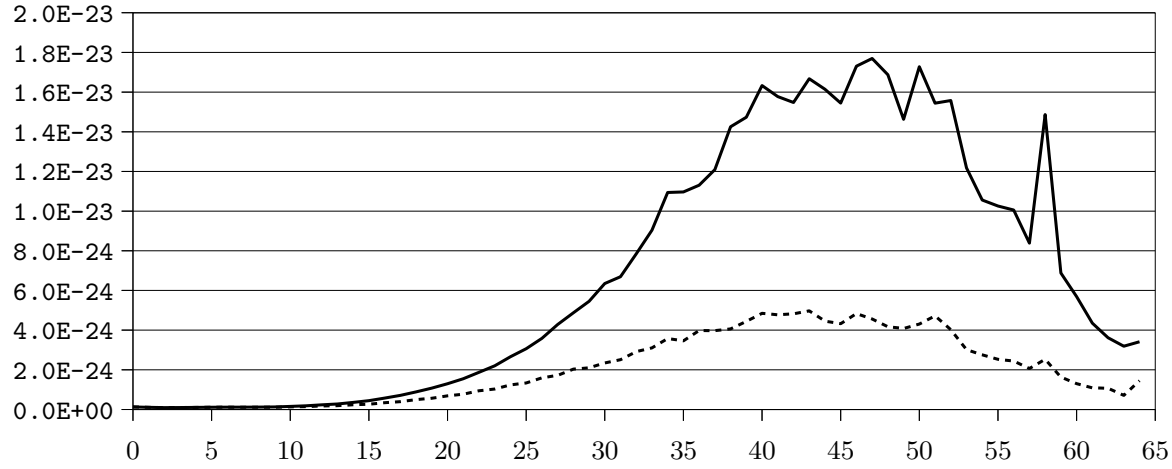


Figure 5.4 *Partitioning (a) quality histogram without (solid line,* $\eta_{\geq 65} = 1.098 \times 10^{-23}$*) and with (dashed line,* $\eta_{\geq 65} = 2.573 \times 10^{-24}$*) transposition.*

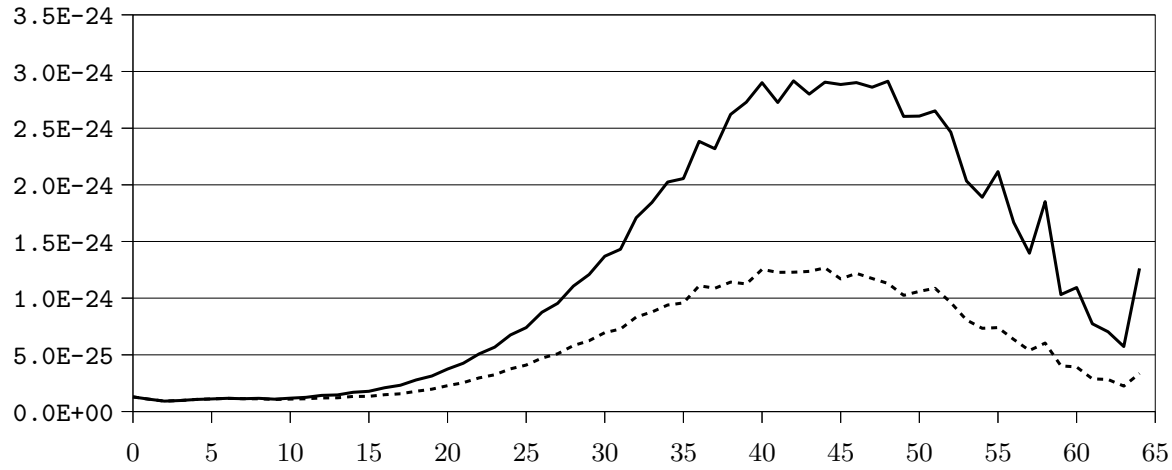


Figure 5.5 *Small catalogue quality histogram without (solid line,* $\eta_{\geq 65} = 2.008 \times 10^{-24}$*) and with (dashed line,* $\eta_{\geq 65} = 2.623 \times 10^{-25}$*) transposition.*

5.3 Predicted vs. Actual Expanded Nodes

As with Korf's formula Eq. 2.6, the definition η is based on, care must be taken to understand what it actually measures. Correct predictions are only obtained when the distribution of start states follows the equilibrium distribution. Zahavi et al. [15] illustrate this problem by showing how restricting the selection of start states for an IDA* search to some distance n to vertices with $d(v, z) \geq n$, i. e. vertices for which an IDA* search to distance n does not overshoot the goal, is already sufficient to render Eq. 2.6 inaccurate. They address this problem by introducing a more sophisticated prediction formula that uses conditional distributions to take the location of the start state into account.

Even accounting for this, measurement proved to be more challenging than the author assumed. Up to 10^6 solvable puzzle configurations have been chosen uniformly at random. On each of these, IDA* was then ran to a distance of up to 80 regardless of the actual difficulty of the problem and the number of vertices expanded at each distance were noted. A high fluctuation of expanded nodes by up to a factor of 24 was observed between different test runs. Similar to the estimation of η by uniform random sample, this is due to the high variation between individual puzzle instances, causing the number of expanded nodes to be dominated by outliers. A bigger sample or an adaption of the sphere sampling scheme could have been able to alleviate these issues, but was not attempted due to a lack of time.

5.4 Distribution of Vertices in the Search Space

Apart from generating samples for the computation of the heuristic quality, the sampling procedure developed in §4.2 also allows us to empirically determine the distribution of vertices with respect to their distances from the solved configuration (or any arbitrary configuration) in arbitrary search spaces.

This method is limited by the necessity to determine if the random walk did indeed reach the desired distance and thus by the computation time needed to solve the many configurations encountered, with computation time generally growing exponentially with distance. It is therefore a feasible method to map the size of the first few spheres, but it breaks down as distance advances. For example, computing the size of sphere V_{64} to a relative error of about 3.15 % (95 % confidence) took about 2.5 days of wall clock time, keeping all 80 threads of a compute server* busy.

A plot of the 24 puzzle's measured vertex distribution can be found in Appendix B.

5.5 Conclusion

The heuristic quality η allows us to determine the influence of the heuristic function on an IDA* search as a constant factor and is a convenient tool to compare different heuristic functions for pruning power. Its elegant mathematical definition allows us to derive interesting theoretical results about the pruning power of classes of heuristics, both making the proof of well-known result easier and enabling us to estimate to find new ones.

By means of random walks, we can effectively draw samples from spheres of vertices equidistant to a given vertex z. As a side product, this method gives us a powerful tool to estimate the sphere sizes, i. e. the number of vertices at each such distance.

Using the samples drawn from random walks, we can effectively estimate η for arbitrary heuristics without having to know any details about the heuristic's construction. Plotting η by contribution of each sphere, we can see how the h values of a critical region in the search space contribute most to the number of expanded nodes.

* 2 × Intel Xeon Gold 6138 CPU @ 2.00 GHz

A Effective Pruning

The pruning method used in this thesis is based on *finite state machine pruning* [5]. In this method, a finite state machine (FSM) is used to record path segments that can be replaced by other path segments of equal or shorter length. If the step we want to take next would cause any such segment to appear in the path we took, we know that there is another path of equal or shorter length leading to the same vertex and perform a different step instead; i. e. not include that vertex in $N'(v_0, \ldots, v_i)$.

Modelling the state space of the 24 puzzle as a groupoid with generators of the form $m_{i,j}$ for adjacent grid locations i and j, meaning *move the blank from location i to location j*, we can easily match path segments as sequences of generators in the FSM. For example, we can use the simple pruning rule of rejecting all path segments that undo the previous move, in other words rejecting all segments of the form $m_{i,j}m_{j,i}$ as $m_{j,i}$ is the inverse of $m_{i,j}$. This yields the simple pruning rule

$$\begin{aligned} N'(v_0) &:= N(v_0) \\ N'(v_0, \ldots, v_{i-1}, v_i) &:= N(v_i) \setminus \{v_{i-1}\}. \end{aligned} \tag{A.1}$$

In the actual sampling process, finite state machines are used that contain all path segments up to a certain length n such that for any pair of vertices $u, v \in V$ with $d(u, v) \leq n$, out of all paths of length at most k, exactly one path from u to v of length $d(u, v)$ is not matched by the finite state machine.

We furthermore keep track of *moribund states*. A FSM state is 0-moribund if it is matching and $k + 1$-moribund if all outgoing edges lead to states that are k or less moribund. Thus, a k-moribund state is one from which no random walk of k of more steps would escape being matched by the FSM; ending in a cul-de-sac is inevitable (hence the name "moribund state").

Using this information, we omit moves leading to k-moribund states from $N'(v_0, \ldots, v_i)$ when there are k or more steps left to go. This way $N'(v_0, \ldots, v_i)$ being inhabited is guaranteed and no random walk is rejected because it cannot be finished, further improving the yield.

To illustrate the effectiveness of a good pruning rule, Fig. A.1 shows the yield y for random walks of lengths up to 59, comparing the simple pruning rule from Eq. A.1 with a finite state machine matching paths up to length $n = 20$ with and without tracking moribund states. Tracking moribund states improved the yield by up to 58 % compared to conventional FSM pruning and up to 457 % compared to the pruning rule from Eq. A.1. This shows the considerable effect on sampling yield coming from the application of finite state machine pruning.

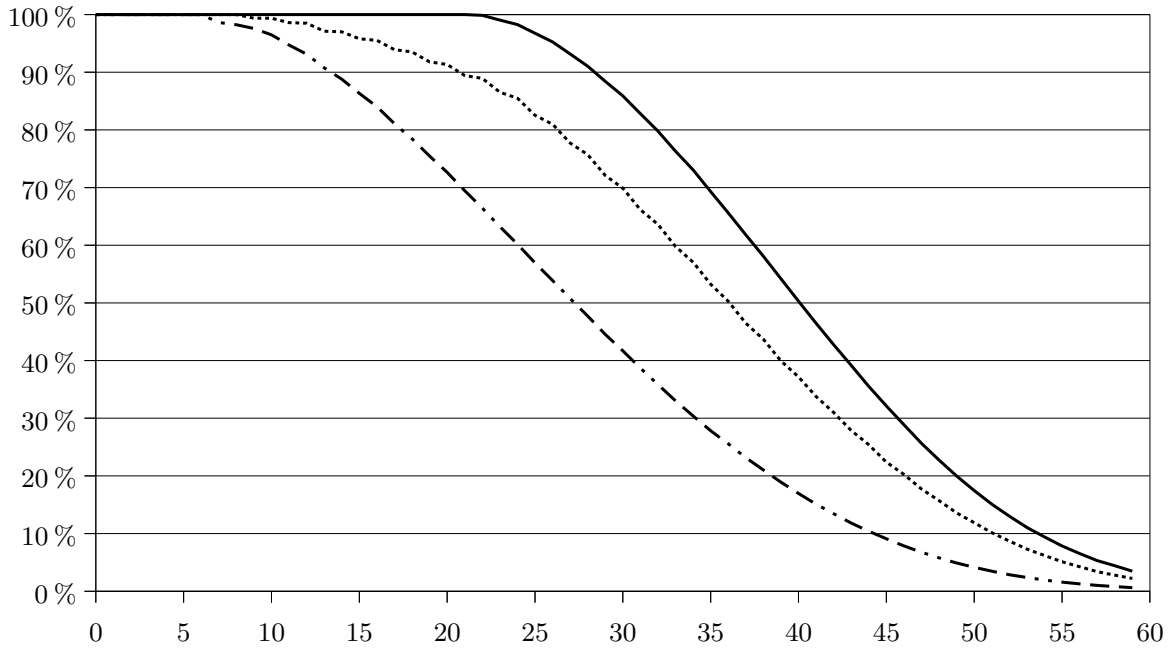


Figure A.1 *Sampling yield by distance using Eq. A.1 (dashed and dotted line) and a finite state machine for $n = 20$ without (dashed line) and with (solid line) moribund state tracking.*

B Sphere Sizes

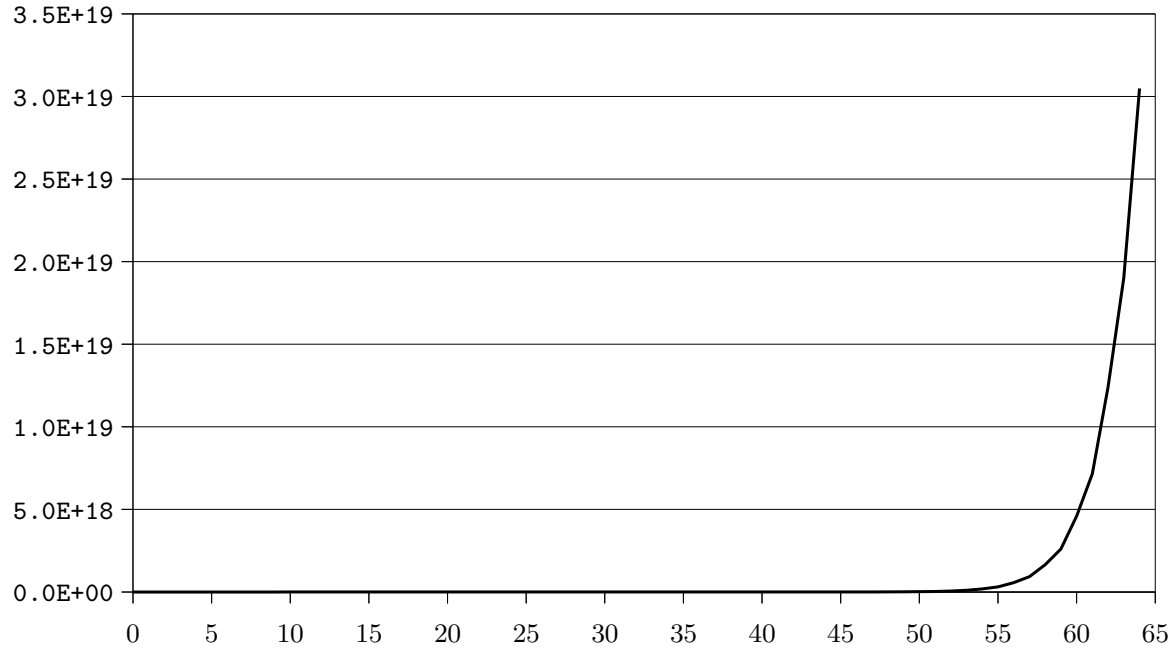


Figure B.1 *Number of vertices of the 24 puzzle's state space by distance from the solved configuration as measured using the sphere sampling method from Sec. 4.2.*

Number of vertices of the 24 puzzle's state space at distance d from the solved configuration. Reference sizes taken from OEIS sequence A090031 [10] are compared with the sizes determined using the sampling procedure from §4. For spheres V_0 to V_{62}, 10^8 attempted samples and a finite state machine with moribund state tracking containing paths up the length $n = 22$ was used. For spheres V_{63} and V_{64}, 5×10^7 attempted samples and a FSM with moribund state tracking up to length $n = 24$ were used instead.

Plotting these measured sphere sizes (cf. Fig. B.1) shows how at 64 steps from the solved configuration, we are far from having reached the largest spheres. At 3.0484×10^{19} vertices, even sphere 64, the largest one measured so far, only accounts for a fraction of about $1 : 250\,000$ of the $25!/2 = 7.7556 \times 10^{24}$ vertices found in the 24 puzzle's search space.

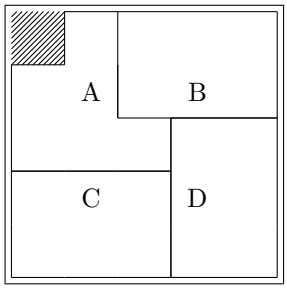
	sphere size			*margin of error*	
d	*reference*	*measured*	*yield*	*95 % confidence*	*actual error*
0	1	$1.0000 \times 10^{+00}$	100.00 %	$0.0000 \times 10^{+00}$ (0.000 %)	$+0.0000 \times 10^{+00}$ (0.000 %)
1	2	$2.0000 \times 10^{+00}$	100.00 %	$0.0000 \times 10^{+00}$ (0.000 %)	$+0.0000 \times 10^{+00}$ (0.000 %)
2	4	$4.0000 \times 10^{+00}$	100.00 %	$0.0000 \times 10^{+00}$ (0.000 %)	$+0.0000 \times 10^{+00}$ (0.000 %)
3	10	$1.0000 \times 10^{+01}$	100.00 %	4.0000×10^{-04} (0.004 %)	$+1.9480 \times 10^{-04}$ (0.002 %)
4	26	$2.6000 \times 10^{+01}$	100.00 %	1.5493×10^{-03} (0.006 %)	-3.2584×10^{-04} (0.001 %)
5	64	$6.3999 \times 10^{+01}$	100.00 %	6.0929×10^{-03} (0.010 %)	-8.2864×10^{-04} (0.001 %)
6	159	$1.5900 \times 10^{+02}$	100.00 %	1.8353×10^{-02} (0.012 %)	$+3.7296 \times 10^{-03}$ (0.002 %)
7	366	$3.6599 \times 10^{+02}$	100.00 %	5.5504×10^{-02} (0.015 %)	-1.0015×10^{-02} (0.003 %)
8	862	$8.6191 \times 10^{+02}$	100.00 %	1.3961×10^{-01} (0.016 %)	-9.3560×10^{-02} (0.011 %)
9	1 904	$1.9039 \times 10^{+03}$	100.00 %	3.6063×10^{-01} (0.019 %)	-1.1341×10^{-01} (0.006 %)
10	4 538	$4.5382 \times 10^{+03}$	100.00 %	9.1175×10^{-01} (0.020 %)	$+1.5343 \times 10^{-01}$ (0.003 %)
11	10 238	$1.0238 \times 10^{+04}$	100.00 %	$2.3368 \times 10^{+00}$ (0.023 %)	$+4.3100 \times 10^{-01}$ (0.004 %)
12	24 098	$2.4098 \times 10^{+04}$	100.00 %	$5.7659 \times 10^{+00}$ (0.024 %)	$+3.8775 \times 10^{-01}$ (0.002 %)
13	53 186	$5.3192 \times 10^{+04}$	100.00 %	$1.4238 \times 10^{+01}$ (0.027 %)	$+6.0750 \times 10^{+00}$ (0.011 %)
14	123 435	$1.2346 \times 10^{+05}$	100.00 %	$3.4538 \times 10^{+01}$ (0.028 %)	$+2.3241 \times 10^{+01}$ (0.019 %)

15	268 416	$2.6840 \times 10^{+05}$	100.00 %	$8.3440 \times 10^{+01}$	(0.031 %)	$-1.5232 \times 10^{+01}$	(0.006 %)
16	616 374	$6.1634 \times 10^{+05}$	100.00 %	$1.9992 \times 10^{+02}$	(0.032 %)	$-3.1437 \times 10^{+01}$	(0.005 %)
17	1 326 882	$1.3270 \times 10^{+06}$	100.00 %	$4.7516 \times 10^{+02}$	(0.036 %)	$+1.5190 \times 10^{+02}$	(0.011 %)
18	3 021 126	$3.0207 \times 10^{+06}$	100.00 %	$1.1280 \times 10^{+03}$	(0.037 %)	$-4.6398 \times 10^{+02}$	(0.015 %)
19	6 438 828	$6.4382 \times 10^{+06}$	100.00 %	$2.6455 \times 10^{+03}$	(0.041 %)	$-6.1112 \times 10^{+02}$	(0.009 %)
20	14 524 718	$1.4525 \times 10^{+07}$	100.00 %	$6.2194 \times 10^{+03}$	(0.043 %)	$-5.9454 \times 10^{+01}$	(0.000 %)
21	30 633 586	$3.0641 \times 10^{+07}$	100.00 %	$1.4391 \times 10^{+04}$	(0.047 %)	$+7.6315 \times 10^{+03}$	(0.025 %)
22	68 513 713	$6.8535 \times 10^{+07}$	100.00 %	$3.3540 \times 10^{+04}$	(0.049 %)	$+2.1325 \times 10^{+04}$	(0.031 %)
23	143 106 496	$1.4304 \times 10^{+08}$	100.00 %	$7.6445 \times 10^{+04}$	(0.053 %)	$-6.5172 \times 10^{+04}$	(0.046 %)
24	317 305 688	$3.1732 \times 10^{+08}$	99.81 %	$1.7652 \times 10^{+05}$	(0.056 %)	$+1.8061 \times 10^{+04}$	(0.006 %)
25	656 178 756	$6.5596 \times 10^{+08}$	98.97 %	$3.9887 \times 10^{+05}$	(0.061 %)	$-2.1555 \times 10^{+05}$	(0.033 %)
26	1 442 068 376	$1.4422 \times 10^{+09}$	98.08 %	$9.1407 \times 10^{+05}$	(0.063 %)	$+1.7247 \times 10^{+05}$	(0.012 %)
27	2 951 523 620	$2.9517 \times 10^{+09}$	96.46 %	$2.0496 \times 10^{+06}$	(0.069 %)	$+1.6952 \times 10^{+05}$	(0.006 %)
28	6 427 133 737	$6.4284 \times 10^{+09}$	94.83 %	$4.6668 \times 10^{+06}$	(0.073 %)	$+1.2967 \times 10^{+06}$	(0.020 %)
29	13 014 920 506	$1.3013 \times 10^{+10}$	92.57 %	$1.0325 \times 10^{+07}$	(0.079 %)	$-1.6665 \times 10^{+06}$	(0.013 %)
30	28 070 588 413	$2.8075 \times 10^{+10}$	90.34 %	$2.3242 \times 10^{+07}$	(0.083 %)	$+4.9061 \times 10^{+06}$	(0.017 %)
31		$5.6194 \times 10^{+10}$	87.57 %	$5.0987 \times 10^{+07}$	(0.091 %)		
32		$1.1994 \times 10^{+11}$	84.80 %	$1.1398 \times 10^{+08}$	(0.095 %)		
33		$2.3783 \times 10^{+11}$	81.59 %	$2.4949 \times 10^{+08}$	(0.105 %)		
34		$5.0202 \times 10^{+11}$	78.35 %	$5.5019 \times 10^{+08}$	(0.110 %)		
35		$9.8134 \times 10^{+11}$	74.74 %	$1.1814 \times 10^{+09}$	(0.120 %)		
36		$2.0533 \times 10^{+12}$	71.17 %	$2.6354 \times 10^{+09}$	(0.128 %)		
37		$3.9619 \times 10^{+12}$	67.29 %	$5.5501 \times 10^{+09}$	(0.140 %)		
38		$8.1914 \times 10^{+12}$	63.49 %	$1.2294 \times 10^{+10}$	(0.150 %)		
39		$1.5588 \times 10^{+13}$	59.45 %	$2.5680 \times 10^{+10}$	(0.165 %)		
40		$3.1794 \times 10^{+13}$	55.53 %	$5.6281 \times 10^{+10}$	(0.177 %)		
41		$5.9655 \times 10^{+13}$	51.49 %	$1.1969 \times 10^{+11}$	(0.201 %)		
42		$1.2043 \times 10^{+14}$	47.57 %	$2.5754 \times 10^{+11}$	(0.214 %)		
43		$2.2266 \times 10^{+14}$	43.60 %	$5.3400 \times 10^{+11}$	(0.240 %)		
44		$4.4187 \times 10^{+14}$	39.82 %	$1.1006 \times 10^{+12}$	(0.249 %)		
45		$8.0513 \times 10^{+14}$	36.07 %	$2.2633 \times 10^{+12}$	(0.281 %)		
46		$1.5792 \times 10^{+15}$	32.54 %	$4.8039 \times 10^{+12}$	(0.304 %)		
47		$2.8374 \times 10^{+15}$	29.09 %	$1.0092 \times 10^{+13}$	(0.356 %)		
48		$5.4745 \times 10^{+15}$	25.91 %	$2.0849 \times 10^{+13}$	(0.381 %)		
49		$9.6311 \times 10^{+15}$	22.82 %	$4.0923 \times 10^{+13}$	(0.425 %)		
50		$1.8397 \times 10^{+16}$	20.03 %	$9.0825 \times 10^{+13}$	(0.494 %)		
51		$3.1874 \times 10^{+16}$	17.38 %	$2.0741 \times 10^{+14}$	(0.651 %)		
52		$5.9307 \times 10^{+16}$	15.02 %	$3.4683 \times 10^{+14}$	(0.585 %)		
53		$1.0118 \times 10^{+17}$	12.82 %	$7.6011 \times 10^{+14}$	(0.751 %)		
54		$1.8691 \times 10^{+17}$	10.90 %	$1.5711 \times 10^{+15}$	(0.841 %)		
55		$3.1078 \times 10^{+17}$	9.14 %	$3.2689 \times 10^{+15}$	(1.052 %)		
56		$5.6396 \times 10^{+17}$	7.63 %	$5.7434 \times 10^{+15}$	(1.018 %)		
57		$9.2789 \times 10^{+17}$	6.28 %	$1.2099 \times 10^{+16}$	(1.304 %)		
58		$1.6497 \times 10^{+18}$	5.14 %	$2.0824 \times 10^{+16}$	(1.262 %)		
59		$2.6358 \times 10^{+18}$	4.15 %	$3.7463 \times 10^{+16}$	(1.421 %)		
60		$4.5888 \times 10^{+18}$	3.33 %	$9.4220 \times 10^{+16}$	(2.053 %)		
61		$7.1572 \times 10^{+18}$	2.63 %	$1.8081 \times 10^{+17}$	(2.526 %)		
62		$1.2410 \times 10^{+19}$	2.07 %	$2.8819 \times 10^{+17}$	(2.322 %)		
63		$1.9020 \times 10^{+19}$	1.91 %	$8.4975 \times 10^{+17}$	(4.468 %)		
64		$3.0484 \times 10^{+19}$	1.47 %	$9.5922 \times 10^{+17}$	(3.147 %)		

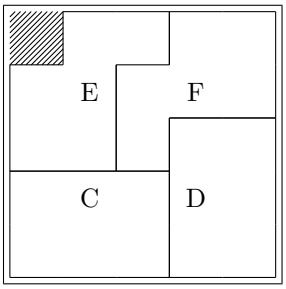
C PDB Catalogues

These two PDB catalogues were developed in my previous work [7] using an empirical method described therein. Provided are a *large catalogue* of 20 pattern databases forming 14 partitionings and as a subset, a *small catalogue* of 14 pattern databases forming just 7 partitionings. For easier identification, a number is assigned to each partitioning and a capital letter to each pattern database. Partitionings 1 and 2 are identical to (a) and (b) from Fig. 5.1.

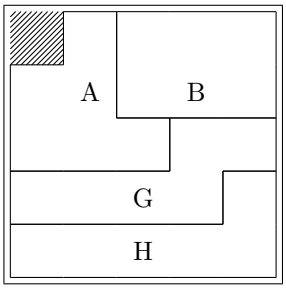
Despite having only half the number of partitionings, Tbl. 5.1 shows that the large catalogue's quality is only slightly better than the small catalogue's. This illustrates the diminished returns obtained from using catalogues comprising too many pattern databases.



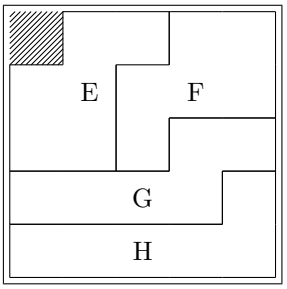
small/large no. 1



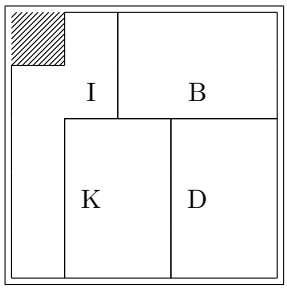
small/large no. 2



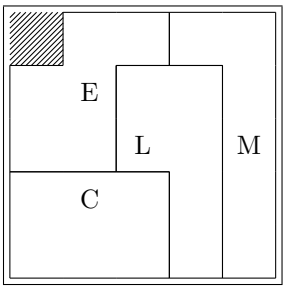
small/large no. 3



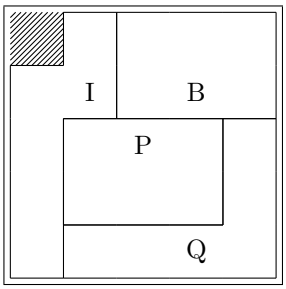
small/large no. 4



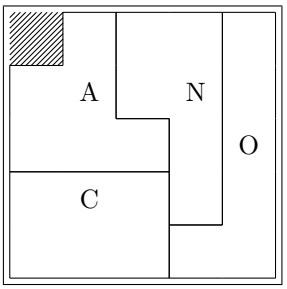
small/large no. 5



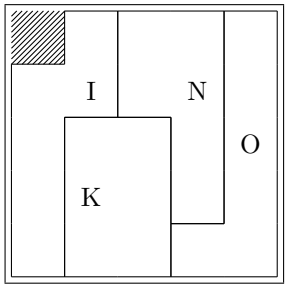
small/large no. 6



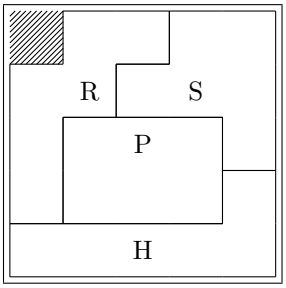
small/large no. 7



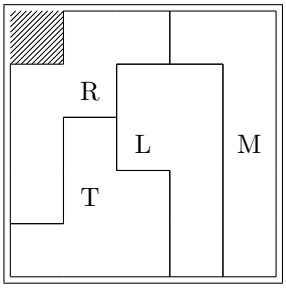
large no. 8



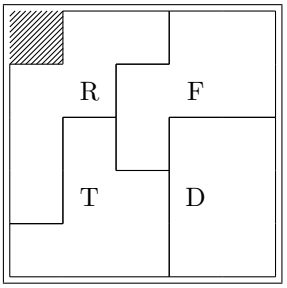
large no. 9



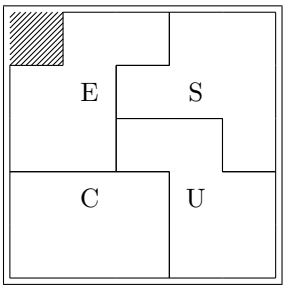
large no. 10



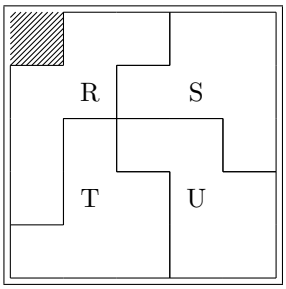
large no. 11



large no. 12



large no. 13



large no. 14

D Literature

[1] Richard E. Korf, *Analyzing the Performance of Pattern Database Heuristics*, Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence, AAAI Press, p. 1164–1170, 2007.

[2] Joseph C. Culberson, Jonathan Schaeffer, *Pattern Databases*, Computational Intelligence, Blackwell, vol. 14, no. 3, p. 318–334, August 1998.

[3] Richard E. Korf, Ariel Felner, *Disjoint Pattern Database Heuristics, Artificial Intelligence*, vol. 134, no. 1, p. 9–22, January 2002.

[4] Florian Pommerening, Malte Helmert, Gabriele Röger, Jendrik Seipp, *From Non-Negative to General Operator Cost Partitioning*, Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI Press, p. 3335–3341, 2015.

[5] Larray A. Taylor, Richard E. Korf, *Pruning Duplicate Nodes in Depth-First Search*, Proceedings of the Eleventh AAAI Conference on Artificial Intelligence, AAAI Press, p. 756–761, 1993.

[6] Richard E. Korf, Michael Reid, Stefan Edelkamp, *Time Complexity of iterative-deepening-A**, Artificial Intelligence, Elsevier, vol. 129, no. 1–2, p. 199–218, 2001.

[7] Robert K. P. Clausecker, *Notes on the Construction of Pattern Databases*, ZIB Report 17-59, Zuse Institute Berlin, 2017.

[8] Richard E. Korf, *Depth-First Iterative-Deepening: An Optimal Admissible Tree Search*, Artificial Intelligence, vol. 27, no. 1, p. 97–109, 1985.

[9] Jerry Slocum, Dic Sonneveld, *The 15 Puzzle*, The Slocum Puzzle Foundation, ISBN 1-890980-15-3, 2006.

[10] Hugo Pfoertner, *Number of configurations of the* 5×5 *variant of sliding block 15-puzzle ("24-puzzle") that require a minimum of n moves to be reached, starting with the empty square in one of the corners*, The On-Line Encyclopedia of Integer Sequences, seq. A090031, published electronically at `https://oeis.org`, 2020.

[11] David R. Cox, *Some sampling problems in technology*, Selected Statistical Papers of Sir David Cox, vol. 1, p. 81–92, 2005.

[12] Wikipedia contributors, *Reservoir Sampling*, Wikipedia, the Free Encyclopedia, page rev. 948434008 of 1 April 2020 00:59 UTC.

[13] Robert C. Holte, Jack Newton, Ariel Felner, Ram Meshulam, David Furcy, *Multiple Pattern Databases*, ICAPS-04 Proceedings, p. 122–131, 2004.

[14] Robert Döbbelin, Thorsten Schütt, Alexander Reinefeld, *Building Large Compressed PDBs for the Sliding Tile Puzzle*, Workshop on Computer Games 2013 Proceedings, p. 16–27, April 2014.

[15] Uzi Zahavi, Ariel Felner, Neil Burch, Robert C. Holte, *Predicting the Performance of IDA* using Conditional Distributions*, Journal of Artificial Intelligence Research, vol. 37, p. 41–83, 2010.