Inconsistent Heuristics

Uzi Zahavi
Computer Science
Bar-Ilan University
Ramat-Gan, Israel 92500
zahaviu@cs.biu.ac.il

Ariel Felner
Information Systems Engineering
Ben-Gurion University
Be'er-Sheva, Israel 85104
felner@bgu.ac.il

Jonathan Schaeffer and Nathan Sturtevant
Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{jonathan,nathanst}@cs.ualberta.ca

Abstract

In the field of heuristic search it is well-known that improving the quality of an admissible heuristic can significantly decrease the search effort required to find an optimal solution. Existing literature often assumes that admissible heuristics are *consistent*, implying that consistency is a desirable attribute. To the contrary, this paper shows that an inconsistent heuristic can be preferable to a consistent heuristic. Theoretical and empirical results show that, in many cases, inconsistency can be used to achieve large performance improvements. The conventional wisdom about inconsistent heuristics is wrong.

Introduction

Heuristic search algorithms such as A* and IDA* are guided by the cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the actual distance from the initial state to state n and $h(n)$ is a heuristic function estimating the cost from n to a goal state. If $h(s)$ is "admissible" (i.e., is always a lower bound) these algorithms are guaranteed to find optimal paths.

It is usually assumed that admissible heuristics are *consistent*, implying that consistency is a desirable attribute. In their popular AI textbook *Artificial Intelligence: A Modern Approach*, Russell and Norvig write that "one has to work quite hard to concoct heuristics that are admissible but not consistent" (Russell & Norvig 2005). Many researches work using the assumption that "almost all admissible heuristics are consistent" (Korf 2000). The term "inconsistent heuristic" is portrayed negatively; as something that should be avoided. Part of this is historical: early research discovered that inconsistency can lead to poor A* performance, however the issue has never been fully investigated, and was not re-considered after the invention of IDA*.

The goal of this paper is to show that many of the preconceived notions about inconsistent heuristics are wrong. We first show that while there can be drawbacks to using inconsistent heuristics with A*, these do not affect IDA*. We then show that inconsistent (admissible) heuristics are easy to create. Finally, we show that there are many benefits for using inconsistent heuristics and provide a number of techniques to do that.

Experimental results show that new and recent techniques together provide a significant reduction in search effort for IDA*-based search applications. For A*, the issue is not as clear; inconsistency can be an asset or a liability, but the application-dependent properties that determine this are a matter of future research.

Results are demonstrated primarily using permutation puzzles, however the ideas are more general, and can be applied in any single-agent search domain. Pattern databases (Culberson & Schaeffer 1998) (PDBs) are heuristics in the form of lookup tables which store solutions to instances of subproblems. We use PDBs since they provide current state-of-the-art heuristics for many applications but this research is not specific to PDBs.

Background

An admissible heuristic h is *consistent* if for any two states, x and y, $|h(x) - h(y)| \leq dist(x, y)$ where $dist(x, y)$ is the shortest path between x and y. In particular, for neighboring states the h-value never changes by more than the change in the g-value. An admissible heuristics h is *inconsistent* if for at least some pairs of nodes x and y, $|h(x) - h(y)| > dist(x, y)$. For example, if a parent node p has $f(p) = g(p) + h(p) = 5 + 5 = 10$, then (since the heuristic is admissible) any path from the start node to the goal node that passes through p has a cost of at least 10. If the heuristic is inconsistent, then for some child c of p, the heuristic could return, e.g., $h(c) = 2$. If operators all have cost 1, the total cost of getting to the goal through c will be at least $f(c) = g(c) + h(c) = (5 + 1) + 2 = 8$. This lower bound, 8, is weaker than the lower bound from the parent. Thus the information provided by evaluating c is inconsistent with the information from its parent p.

Pathmax (PMX) is one approach to correcting inconsistent heuristics (Mero 1984). It propagates heuristic values from a parent node p to its child c as follows. $h(p) - dist(p, c)$ is a lower bound on $dist(c, Goal)$ and therefore can be used instead of $h(c)$ if it is larger. In this case c inherits its f-cost from p. Note that PMX is not needed for IDA*, because if the parent node already exceeds the threshold than the child node will not even be generated. Otherwise, if the parent does not exceed the threshold then PMX will never lift the f-cost of the child above that of its parent, which is required for a cut-off to occur in IDA*.

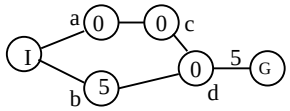

Figure 1: Reopening of nodes with A*

The adjective "inconsistent" has negative connotations, implying that it is something to be avoided. The most important drawback of inconsistent heuristics is that the same

node can be expanded more than once when using A*, even when pathmax is employed (Martelli 1977). With a consistent heuristic, the first time a node is expanded by A* it always has the minimal g value. By contrast, with inconsistent heuristics the search might re-expand nodes when they are reached via a shorter path, especially if many small cycles exist. This operation is sometimes referred to as the *reopening* of nodes, since nodes from the *closed list* are reopened and moved back to the *open list*. An example is shown in Figure 1. Assume that all edges have a weight of 1 except edge (d, G) which has weight 5. Nodes are labeled with their h-values. With A* (even with PMX) nodes a ($f(a) = 1$) and b ($f(b) = 6$) are generated first. Next, node a is expanded, then c, leading to the expansion of node d ($f(d) = 3$). There is no path to the goal of cost less than or equal to 6 so A* returns to node b. Expanding b results in reopening node d (with a new cost of $f(d) = 6$). Now the g-value of d is the minimal value. The optimal path to the goal has been found, but d was expanded twice.

With IDA*, d will be expanded twice, once for each of the paths, regardless of whether the heuristic is consistent or inconsistent. Thus the problem of re-expanding nodes already exists in IDA*, and using inconsistent heuristics will not make this behavior worse. This paper concentrates on IDA*, returning to the issues surrounding A* at the end.

In most previous work on admissible heuristics, research concentrated on improving the quality of the heuristic assessment. A heuristic h_1 is considered to be more informed (better quality) than h_2 if it usually returns higher values for arbitrary states. For a state s, a more informed heuristic generally improves the $f(s)$ value and increases the chance of a cutoff in the search. In the 15-puzzle, for example, there have been massive performance gains seen through the development of more informed heuristics (20 years of research have led a reduction of four orders of magnitude in the search effort needed).

A *de facto* standard usually used by researchers (e.g., (Korf 1997; Korf & Felner 2002; Felner *et al.* 2004)) for comparing the "informedness" of heuristics is to compare the average values of a given heuristic over the entire domain space or, if not practical, over a large sample of states of the domain. This paper demonstrates that, while the average heuristic value is important, there are other considerations that can influence the effectiveness of the heuristic.

Achieving Inconsistent Heuristics

As illustrated earlier, there is a perception that inconsistent admissible heuristics are hard to create. However, inconsistent heuristics have been used effectively in a number of applications, including puzzles (Zahavi *et al.* 2006), pathfinding (Likhachev & Koenig 2005), learning real-time A* and Moving Target Search (Shimbo & Isida 2000). Furthermore, with PDBs it turns out to be very easy to generate inconsistent heuristics. Three examples follow. The first is the most general. It is new and is explored further in this paper. The other two have been recently used to solve the permutation search space applications used in this paper. These are by no means the only ways of creating inconsistent heuristics.

1: Random selection of heuristics: A well-known method for overcoming pitfalls of a given heuristic is to consult a number of heuristics and use their maximum. Of course, there is a tradeoff for doing this—each heuristic calculation increases the time it takes to compute $h(s)$. Additional heuristic consultations provide diminishing returns, in terms of the reduction in the number of nodes generated, so it is not always best to use them all.

Given a number of heuristics one could alternatively select which heuristic to use randomly. One benefit is that only a single heuristic will consulted at each node. Random selection between heuristics will produce inconsistent heuristic values if there is no (low) correlation between the heuristics.

Multiple heuristics often arise from domain-specific geometrical symmetries, meaning that there are no additional storage costs associated with these extra heuristics.

2: Dual heuristics: In permutation spaces, for each state s there exists a dual state s^d which shares many important attributes with s, such as the distance to the goal (Felner *et al.* 2005; Zahavi *et al.* 2006). Therefore, any admissible heuristic applied to s^d is also admissible for s. In the permutation space puzzles used in this paper, the role of locations and objects (values) can be reversed; the "regular" state uses a set of objects indexed by their current location, while the "dual" state has a set of location indexed by the objects they contain. Both mappings can be looked up in the same PDB and return an admissible value. Performing only regular PDB lookups for the states generated during the search produces consistent values. However, the values produced by performing the dual lookup can be inconsistent because the identity of the objects being queried can change dramatically between two consecutive lookups.

3: Compressed PDBs: Larger PDBs tend to be more accurate than smaller PDBs and thus reduce the number of generated nodes. Lossy PDB compression, by storing the minimum of a group of PDB entries in one entry, proved to preserve most of the information of the larger (more accurate PDB) but with less space (Felner *et al.* 2004). Heuristic values obtained by compressed PDBs might be inconsistent even if the original PDB heuristic is consistent, because the PDB values for two neighboring nodes can be compressed into two different entries in the compressed PDB.

In summary, there are two easy ways to generate inconsistency for a given domain: 1) the use of multiple different heuristics (e.g., symmetries, dual states), and 2) using a heuristic that has some values missing or degraded (e.g., lossy compression). This list is not exhaustive.

Benefits of Inconsistency

Consider a consistent heuristic that is being applied to state s in a region of the search space where the heuristic is a poor estimator of the true distance to the goal. Since the heuristic is consistent, each of the children of s have a value that differs from that of s by at most c, where c is the cost of the operator used to reach them. In other words, the value of a node and its children are correlated. A search algorithm will incur significant costs before it is able to escape this region of poor heuristic values.

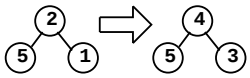

Figure 2: Bidirectional pathmax

Consider using an inconsistent heuristic. Heuristics that arise from random or dual lookups might have no correlation or only little correlation between the heuristic value of s and that of the children of s. Thus when reaching a state with a poor heuristic estimation, a child of s might have a much larger heuristic value – possibly large enough to escape from this region of the search space. This can be done with *bidirectional pathmax* (BPMX) which we introduced in (Felner *et al.* 2005) which further improves the original pathmax method. This is illustrated in Figure 2. The left side of the figure shows the (inconsistent) heuristic values for a node and its two children. When the left child is generated, its heuristic ($h = 5$) can propagate up to the parent and then down again to the right child. To preserve admissibility, each propagation reduces h by the cost of traversing that path (1 in this example). This results in $h = 4$ for the root and $h = 3$ for the right child. Note that BPMX is only applicable for undirected graphs. When using IDA*, BPMX can cause many nodes to be pruned that would otherwise be expanded. For example, suppose the current IDA* threshold is 2. Without the propagation of h from the left child, both the root node ($f = g + h = 0 + 2 = 2$) and the right child ($f = g+h = 1+1 = 2$) would be expanded. Using BPMX, the left child will improve the parent's h value to 4, resulting in a cutoff without even generating the right child.

In summary, inconsistent heuristics are valuable when the values are not highly correlated between neighboring nodes. This greatly reduces the chance of entering a region of the search space where all the heuristic values are low. Because heuristic values can be propagated with PMX and BPMX, a single node with a good heuristic may be sufficient to direct the search into better parts of the search space.

Static Distribution and Korf's Formula

The distribution of values from a heuristic function can be used to measure the "informedness" of the function. Typically this distribution is *statically* computed over the space of all possible states or, if impractical, a large random sample of states. Doing this for admissible heuristics will usually show that if a heuristic is more informed then the distribution of values will be higher, as will be the average value.

(Korf, Reid, & Edelkamp 2001) suggested that the number of nodes expanded by IDA* with a *consistent* admissible heuristic is $N(b, c, P) = \sum_{i=0}^{d} b^i P(c - i)$ where b is the brute-force branching factor, c is the depth of the search and P is the static distribution function of heuristics. They first showed that the expected number of all nodes n such that $f(n) = g(n) + h(n) \leq c$ is equal to $N(b, c, P)$. These nodes have the *potential to be expanded*. They then proved that all the potential nodes will eventually be expanded by IDA* as follows. Assume that n is a potential node. Since the heuristic is *consistent* then any ancestor of n, p, must also have $f(p) \leq c$ and is also a potential node. Then, by a simple induction they showed that the entire branch from the root to n will be expanded since all the nodes of the branch are potential nodes.

For inconsistent heuristics this is not necessarily true. Compare, for example, the dual (or random) PDB lookup to the regular consistent lookup of the same PDB. Since exactly the same PDB is used, all heuristics will have the same static distribution of values. Thus, according to Korf's formula, the number of potential nodes (i.e., their $f(n) \leq c$) will again be $N(b, c, P)$. However, Korf's proof that given a potential node n all its ancestors must also be potential nodes is not true. Due to inconsistency there might exist an ancestor p with $f(p) > c$. Once IDA* visits this node the entire subtree below it is pruned and n will not even be generated. A potential node will be expanded only if all its ancestors are also potential nodes. This is guaranteed for consistent heuristics but not for inconsistent heuristics. Thus, for inconsistent heuristic $N(b, c, P)$ is only an upper bound on the number of generated nodes.

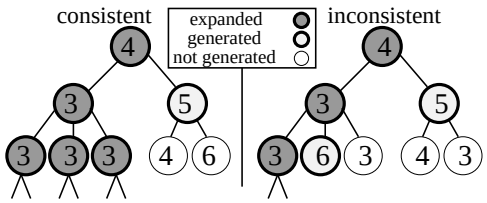

Figure 3: Consistent versus inconsistent heuristics

An example of the difference between consistent and inconsistent heuristics is shown in Figure 3. Nodes are marked with their h-value. Observe that in both cases we have exactly the same h-value distribution in the various levels of the tree. For the consistent case, if the current IDA* threshold is 5, all 3 nodes at depth 2 with h-value of 3 have $f = g + h = 2 + 3 = 5$ and will be expanded. The right subtree is pruned because the f-value of the right node at level 1 is $f = g + h = 1 + 5 = 6 > 5$. For the inconsistent case, however, only one node at depth 2 will be expanded – the leftmost node. The second node with h-value of 6 will be generated but not expanded because its f-value is 8 and exceeds the threshold. Due to BPMX, its value will be propagated to its parent and its right sibling (a potential node with $h = 3$) will not even be generated. The right subtree will be pruned again and, due to PMX, the rightmost node – with h-value of 3 – will not be generated.

Dynamic Distribution of Heuristic Values

There is no guarantee that the static distribution of values in a heuristic will have the same distribution as the values actually considered during search. What makes a heuristic effective is not its overall *static distribution* but the *dynamic distribution* of the values generated during search.

The idea of static and dynamic distributions of heuristic values is not new; it has been previously used to explain why the maximum of several weak heuristics can outperform one stronger heuristic (Holte *et al.* 2006). In this paper we examine the distributions generated by inconsistent heuristics and analyze their influence on the search.

The Rubik's Cube puzzle is used to illustrate the impact of inconsistent heuristics on the search. The puzzle has

#	No	Lookup	Nodes	Time
One PDB lookup				
1	1	Regular	90,930,662	28.18
2	1	Dual	19,653,386	7.38
3	1	Dual + BPMX	8,315,116	3.24
4	1	Random	9,652,138	3.30
5	1	Random + BPMX	3,829,138	1.25
Maxing over multiple PDB lookups				
6	2	Regular	13,380,154	7.85
7	4	Regular	10,574,180	11.60
8	2	Random + BPMX	1,902,730	1.14
9	4	Random + BPMX	1,042,451	1.20

Table 1: Rubik's Cube results for 100 instances

been used extensively for benchmarking the performance of search algorithms. In Rubik's Cube there are 20 movable cubes (or "cubies"); 8 are corners and 12 are edges. The 8-corners PDB (first used by (Korf 1997)) cannot be used here because it is always consistent since all 8 corners are always examined. We experimented with a large variety of other PDBs where inconsistency can be achieved. For example, we added a single edge cubie to the 8-corner PDB resulting in a large PDB with 9 cubies. Similarly, we have experimented with PDBs for the TopSpin puzzle. For compatibility, we chose to report the results for the same 7-edge PDB that we used in (Felner *et al.* 2005; Zahavi *et al.* 2006) but similar tendencies were observed in our other experiments. There are 24 lines of geometrical symmetries, which arise from different ways to rotate and reflect the cube. For our 7-edge PDB, each of these symmetries considers a different set of edges, and thus results in a different PDB lookup. The traditional ("regular") and dual state ("dual") PDB lookups implemented in (Felner *et al.* 2005) was used as the basis for comparison.

Table 1 shows the average number of generated nodes and the average running time over the same set of 100 depth-14 Rubik's cube instances taken from (Felner *et al.* 2005). Column No gives the number of PDB lookups used. The following PDB lookups were used for lines $1-5$:

Regular: The regular PDB lookup. This heuristic is consistent because the same set of cubies is used for the PDB lookup of both parent and child nodes.

Dual: For each node, the dual state is calculated and is looked up in the PDB. This will produce inconsistent heuristic values because the dual lookup of the parent might consult different cubies than the dual lookup of the child.

Random: Randomly select one of the different 24 possible symmetric PDB lookups for the given node. This is inconsistent because the set of cubies that are used for the parent are not necessarily the same as for the child.

The table shows that a random lookup (with BPMX) is much faster than either one regular lookup (a factor of 24) or one dual lookup (a factor of 2). Lines 6-9 shows the results of maximizing over a number of regular and random lookups. It is interesting to note that even one random lookup (line 5) generates just one third of the nodes than with 4 regular lookups (line 7). Note that performing more PDB lookups can only decrease the number of nodes but there is a diminishing return in terms of time because each PDB lookup incurs overhead. The best time for regular

#	No	Lookup	Nodes	Time
One PDB lookup				
1	1	Regular	136,289	0.081
2	1	Dual + BPMX	247,299	0.139
3	1	Random + BPMX	44,829	0.029
Maxing over multiple PDB lookups				
4	2	Regular*	36,710	0.034
5	2	2 Randoms + BPMX	26,863	0.025
6	3	3 Randoms + BPMX	21,425	0.026
7	4	Regular* + Dual* + BPMX	18,601	0.022

Table 2: Random lookups on the 15-puzzle

PDB lookups (2 regular lookups, line 6) was improved by one random lookup (line 5) by factor of 5.

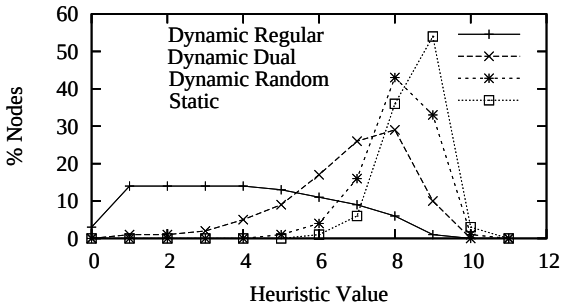

Figure 4: Rubik's Cube value distributions

Figure 4 shows the distribution of the heuristic values seen during the searches of Table 1. We make the following observations from these results. First, note the dramatic difference between the static distribution of values and the dynamic distribution for the regular (consistent) heuristic. As pointed out by others, the static distribution is a poor approximation of the dynamic distribution (Holte *et al.* 2006). Second, it is easy to recognize that the heuristic that performed the best also had a higher distribution of heuristic values. Note that all these versions used exactly the same PDB with the same overall static distribution of values. Third, the regular heuristic did not gain from the potential of the static distribution because it is consistent; when the heuristic for a state is low, the children of that state will also have low values. The inconsistent heuristics do not have this problem; a node can receive any value, meaning that the distribution of values seen is closer to the potential of the static distribution. Finally, inconsistency has the effect of improving the dynamic runtime distribution towards that of the static distribution. The greater the level of inconsistency, the closer the dynamic distribution approaches the static distribution.

Table 2 shows similar results for the 15-puzzle based based on the four possible lookups of the 7-8 additive PDBS from (Korf & Felner 2002). In the table, the "*" indicates the use of the state and its reflection. Again, the benefit of random is shown. A single random lookup (line 3) clearly outperforms a single regular lookup (line 1) but it is also faster than the best results of (Korf & Felner 2002) (line 4).

Inconsistency Rate

An inconsistent heuristic is most useful if there is a low correlation between heuristic values in adjacent states. To better understand the behavior of the different heuristics, two new terms are introduced:

1) Inconsistency rate of an edge (IRE): The inconsistency rate of a heuristic h and an edge $e = (m, n)$ is the difference in the heuristic value for the two nodes incident to this edge. $IRE(h, e) = |h(m) - h(n)|$. The IRE of the entire domain is defined as the average IRE over the entire set of edges of the state space.

2) Inconsistency rate of a node (IRN): For a given node, choose the incident edge with the maximum inconsistency rate. $IRN(h, n) = \max_{m \in adj(n)} |h(m) - h(n)|$. The IRN of the entire domain is defined as the average IRN over the entire set of nodes of the state space.

For a consistent heuristic with a uniform edge cost of 1 (as in Rubik's Cube), both the IRN and IRE for all edges and nodes are less than or equal to 1.

The performance of a search can also be characterized by its branching factor. The dynamic branching factor (DBF) is defined to be the average number of children that are generated for each node that is expanded in the search. When the heuristic function is inconsistent and BPMX is employed, the dynamic branching factor can dramatically decrease.

#	IRE	IRN	DBF	Nodes	BPMX Cuts
1	0.434	1.000	13.355	90,930,662	0
2	0.490	1.237	9.389	17,098,875	717,151
3	0.510	1.306	9.388	14,938,502	623,554
5	0.553	1.424	7.152	5,132,396	457,253
8	0.571	1.467	7.043	4,466,428	402,560
12	0.597	1.527	7.036	3,781,716	337,114
16	0.607	1.552	6.867	3,822,422	356,327
20	0.608	1.558	6.852	3,819,699	357,436
24	0.609	1.561	6.834	3,829,139	360,067
dual	0.441	1.358	7.681	8,315,117	796,849

Table 3: Rubik's Cube: random heuristic with BPMX

Table 3 presents results for these new measurements on Rubik's Cube obtained using the 7-edges PDB. There are 24 possible symmetric lookups that could be used. We varied the number of possible symmetries that random could choose from to perform a single lookup. The first column gives this number. The next columns present the IRE and IRN averaged over 100 Million random states. The last three columns show results averaged over the same set instances of table 1 for searches with the particular random heuristic. We report the DBF, the number of nodes that were generated and the number of times that BPMX was used in the search.

In the first row, only one symmetry was allowed and thus the same PDB lookup was performed at all times. This is the benchmark case of a single consistent regular PDB heuristic lookup. Note that the IRE is close to 0.5, implying that the difference in the heuristic for two neighboring nodes was roughly 0 half the time, and 1 the other half. The IRN was exactly 1 showing that for each node in Rubik's Cube there exists at least one neighbor whose heuristic is different by 1. The dynamic branching factor here is equal to 13.355, which is consistent with the results in (Korf 1997).

As the heuristic randomly considers progressively more symmetries, the inconsistency rates increase and the DBF decreases. This results in a significant reduction in the number of generated nodes. It is important to note two issues here. First, the range of heuristic values in Rubik's Cube is rather small, as can be seen in Figure 4. Thus, the potential for a large inconsistency rate is rather modest. However, even in this domain, a rather small IRN of 1.5 caused a dramatic speedup in the search. Second, no extra overhead is needed by these heuristics as only a single PDB lookup is performed at each node. The constant time per node was such that 3.2 million nodes per second were generated in all our runs on a 3.0GHz Pentium 4 machine. Thus, the reduction in nodes is fully reflected in the running times.

Decreasing the Effective Branching Factor

When performing BPMX cutoffs the DBF decreases. As stated previously, we can generate BPMX cutoffs when the heuristic value of a child is larger than the heuristic of the parent by more than 1, assuming an edge cost of 1. Thus, if such a child exists, we would like to generate it as fast as possible. If, in a particular domain, the operators have different inconsistency rates, we can order the application of operators to maximize the chance of a BPMX cutoff.

The operators on Rubiks cube are symmetric, there is no way to order them. Thus, we demonstrate this in the *pancake puzzle* Imagine a waiter with a stack of n pancakes. The waiter wants to sort the pancakes ordered by size. The only available operation is to lift a top portion of the stack and reverse it. Here, a state is a permutation of the values $0...(N-1)$ and has $N-1$ successors, with the k^{th} successor formed by reversing the order of the first $k+1$ elements of the permutation ($1 \leq k < N$). For example, if $N = 4$ the successors of state $< 0, 1, 2, 3 >$ are $< 1, 0, 2, 3 >, < 2, 1, 0, 3 > and < 3, 2, 1, 0 >$. The size of this state space is $N!$ and there is a uniform static branching factor of $N-1$. An important attribute of this domain is that the number of tokens (pancakes) that change their locations is different for each of the operators. As we noted in (Zahavi *et al.* 2006), here, there are no possible geometrical symmetric PDB lookups and the only way to achieve inconsistency is with the dual lookup.

	Regular		Dual	
Op	Max	IRE	Max	IRE
2-10	1	0.370 - 0.397	0	0
11	1	0.396	2	0.613
12	1	0.397	4	0.958
13	1	0.400	6	1.165
14	1	0.401	8	1.291
15	1	0.402	9	1.358
16	1	0.411	10	1.376
17	1	0.216	9	1.321

Table 4: IRE for operators of the 17-pancake

Table 4 shows different measurements on the operators of the 17 pancake puzzle. Operator k refers to the operator that reverses locations $1 \ldots k$. To measure the inconsistency rate of an operator, we first chose a random state, s_1. We then performed the relevant operator on this state, arriving at state s_2, and measured the difference in the heuristic value between s_1 and s_2. This was repeated for 100 million different states. The *Max* column presents the maximal heuristic difference (maximal IRE) found for the specific operator. The IRE column presents the average IRE over all instances.

Similar measurements are reported for the dual PDB lookup.

The regular PDB lookup is consistent. Indeed, the table shows that for all the operators, the maximal IRE was 1 and the average IRE was smaller than 0.5. For the dual PDB lookups the results are more interesting. First, we note that for operators 2-10 all the IRE values were exactly 0. This is an artifact of the particular PDB used for these experiments (which are based on locations 11-17). The dual lookup of this PDB was not changed by operators 2-10. But, for larger operators (13-17), the IRE for the dual lookup is more than 1. It is interesting to note that operator 16 has a larger IRE than operator 17 even though it only changes a smaller number of locations.

#	Lookup	Order	Nodes	DBF
1	Regular	incr.	342,308,368,717	15.00
2	Dual	incr.	27,641,066,268	15.00
3	Dual + BPMX	incr.	14,387,002,121	10.11
4	Regular	IRE	113,681,386,064	15.00
5	Dual	IRE	13,389,133,741	15.00
6	Dual + BPMX	IRE	85,086,120	4.18
Maxing over two PDB lookups				
7	R + D + BPMX	incr.	2,478,269,076	10.45
8	R + D + BPMX	IRE	39,563,288	5.93

Table 5: 17-pancake results.

Table 5 shows the average number of nodes that were generated by IDA* using different heuristics and different operator ordering when optimally solving the same 10 random instances from (Zahavi *et al.* 2006). The *Order* column presents the operator ordering that was used. Rows (1-3) are for the trivial case were the operators are ordered in increasing order, the ordering we used in (Zahavi *et al.* 2006). Rows (4-6) correspond to the operator ordering in the exact decreasing order of IRE imposed by our measures from Table 4. In both cases, there is a dramatic decrease when moving from the regular to the dual lookup and when further adding BPMX. However, operator ordering significantly influences the results. With the operator ordering based on the IRE we get an additional 500 times reduction in nodes expanded over the simple operator ordering. The new state-of the art performance for this domain is obtained by using the maximum of two heuristics (regular and dual) with BPMX and operator ordering. This version outperforms the single regular lookup by 4 orders of magnitude.

Inconsistency with A*

BPMX for A* works as follows. Assume that a node p is expanded and that its k children $v_1, v_2, \ldots, v_k$ are generated. Let v_{max} be the node with the maximum heuristic among all the children and let $h_{max} = h(v_{max})$. Assuming that each edge has a unit cost, we can now propagate h_{max} to the parent node by decreasing 1 and then to the other children by decreasing 1 again. Thus, each of the other children v_i can have a heuristic of

$$h_{BPMX}(v_i) = max(h(v_i), h(p) - 1, h_{max} - 2)$$

For example, assume that the parent node of Figure 2 is the root node and thus its f-value is $f = g + h = 0 + 2 = 2$. When generating the children, the left child has $f = g + h = 1 + 5 = 6$ and the right child has $f = g + h = 1 + max(5 - 2, 2 - 1, 1) = 1 + 3 = 4$. Thus, the right child is inserted to the open list with $f = 4$ instead of $f = 2$.

We have performed experiments with inconsistent heuristics in several domains, including the top-spin puzzle and several different types of pathfinding grids. The results are not conclusive. In top-spin there are large gains from using inconsistent heuristics and BPMX. In pathfinding problems there can be gains, but this depends on the types of maps being used. It is our conjecture that the gains from BPMX and inconsistent heuristics with A* are related to the number and size of cycles within the state space. A detailed exploration of inconsistent heuristics in A* is outside the scope of this paper and left as future work.

Conclusions and Future Work

Historically, inconsistent heuristics have been avoided because of the cost of re-expanding closed nodes with A*, but this is not a concern with IDA*. This paper has demonstrated that inconsistent heuristics are easy to create, and that a large speedup can be achieved when using them with IDA* and BPMX. This represents a significant change to the conventional wisdom for heuristic search.

Exploring the possible potential gain for using inconsistent heuristics with A* is the subject of on-going research.

Acknowledgments

This research was supported by the Israel Science Foundation (ISF) under grant number 728/06 to Ariel Felner. We thank Robert Holte for his help with this manuscript.

References

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Felner, A.; Meshulam, R.; Holte, R.; and Korf, R. 2004. Compressing pattern databases. In *AAAI*, 638–643.

Felner, A.; Zahavi, U.; Holte, R.; and Schaeffer, J. 2005. Dual lookups in pattern databases. In *IJCAI*, 103–108.

Holte, R. C.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence* 170:1123–1136.

Korf, R. E., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134:9–22.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of ida*. *Artificial Intelligence* 129(1-2):199–218.

Korf, R. E. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI*, 700–705.

Korf, R. E. 2000. Recent progress in the design and analysis of admissible heuristic functions. In *AAAI*, 1165–1170.

Likhachev, M., and Koenig, S. 2005. A generalized framework for lifelong planning a*. In *ICAPS*, 99–108.

Martelli, A. 1977. On the complexity of admissible search algorithms. *Artificial Intelligence* 8:1–13.

Mero, L. 1984. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence* 23:13–27.

Russell, S., and Norvig, P. 2005. *Artificial Intelligence, A Modern Approach, Second Edition*. Prentice Hall.

Shimbo, M., and Isida, T. 2000. Towards real-time search with inadmissible heuristics. In *ECAI*, 609–613.

Zahavi, U.; Felner, A.; Holte, R.; and Schaeffer, J. 2006. Dual search in permutation state spaces. In *AAAI*, 1076–1081.