An Analysis and Enhancement of the Gap Heuristic for the Pancake Puzzle

Richard Valenzano
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
rvalenzano@cs.toronto.edu

Danniel Sihui Yang
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
dannielyang1996@gmail.com

Abstract

The pancake puzzle is a standard benchmark domain used to test search algorithms, and the gap heuristic is the state-of-the-art heuristic function most often used in such tests. In this work, we analyze the accuracy of this heuristic and identify ways to enhance it. We begin by showing that in the worst-case, the amount that the gap heuristic underestimates the optimal cost of a pancake puzzle state can be linear in the number of pancakes in the stack. However, empirical analysis suggests that it is extremely rare that the gap heuristic underestimates the optimal cost by more than two. We then identify several simple methods that can be used to generate large sets of problems on which the gap heuristic underestimates the optimal cost by a larger amount than it typically does on random permutations. In doing so, we provide new pancake puzzle test sets that can be used to evaluate how search algorithms behave when the heuristic is inaccurate.

We also formally characterize states according to the size of the heuristic plateaus around them. This characterization allows us to efficiently compute a two-step lookahead of the gap heuristic on any state, which we can use alongside a state's dual to further improve heuristic accuracy. These enhancements substantially improve the performance of an IDA*-based pancake problem solver on both the existing benchmarks and the new ones proposed in this paper.

1 Introduction

The **pancake puzzle** (Gates and Papadimitriou 1979) is a classic combinatorial problem that is related to a number of applications, including routing in a parallel computer (Qiu, Meijer, and Akl 1991) and the calculation of genome similarity (Hayes 2007). In this problem, a chef must sort a stack of pancakes in increasing size from top to bottom, given a spatula that can be used to flip some portion of the top of the stack, using as few flips as possible. The pancake puzzle has been extensively studied by the algorithms community (Gates and Papadimitriou 1979; Heydari and Sudborough 1997; Chitturi et al. 2009; Fischer and Ginzinger 2005; Bulteau, Fertin, and Rusu 2015), and has become a standard benchmark for comparing and analyzing search algorithms (Bouzy 2015; Lippi, Ernandes, and Felner 2016; Zahavi et al. 2008). This is due in part to the problem's simplicity, and that it has a higher branching factor than other standard domains like the sliding tile puzzle.

The heuristic function most often used in such work is the **gap heuristic** (Helmert 2010), which is generally viewed as being very accurate. In this paper, we analyze this heuristic in an effort to increase our understanding of its properties and thereby better equip researchers who are using the pancake puzzle to evaluate a search algorithm. Our empirical analysis shows that the gap heuristic is accurate on a very high percentage of states, and rarely underestimates the optimal cost of a randomly generated state by more than two. However, the gap heuristic can be much more inaccurate, which we show by using existing results on the diameter of the pancake puzzle to prove that the worst-case heuristic error of any state with N pancakes grows linearly with N. We then define several methods for generating large sets of states with a higher average heuristic error than random permutations. In doing so, we have provided new benchmarks that can be used to evaluate search algorithms on problems with a high branching factor on which the heuristic is not almost always providing nearly perfect estimates.

We have also formally studied the local search topology of the gap heuristic. In particular, we classify states according to their distance from the nearest state with a lower heuristic value and show that there are no heuristic local minima. We then identify that the topology of the gap heuristic allows us to efficiently compute the value of a one or two step lookahead from every state, and thereby improve the admissible heuristic estimates being used by the search. This efficient lookahead can be combined with methods for exploiting a state's dual, and the resulting enhancements are shown to substantially decrease search time when used on both the existing and new benchmark problems.

2 Background

In this section, we provide background on the pancake problem and define the notation used in the rest of the paper.

Sequences and Permutations. We represent a **sequence** σ of k elements from some set as $\sigma = \langle e_1, ..., e_k \rangle$. We use $\sigma[i]$ to refer to the i-th element of σ (*i.e.* $\sigma[i] = e_i$). Notice that the first element in the permutation is at location 1, which is the convention in the pancake puzzle literature. If $\sigma' = \langle g_1, ..., g_{k'} \rangle$ is a second sequence, then $\sigma \circ \sigma'$ denotes the **concatenation** of these sequences. This means that $\sigma \circ \sigma' = \langle e_1, ..., e_k, g_1, ..., g_{k'} \rangle$.

A **permutation** π of size N is a sequence of the natural numbers from 1 to N, such that each element in the sequence is unique. The **dual** or **inverse** of a permutation π, is defined as the permutation π^D where for every $1 \leq i, j \leq N$, if $\pi[i] = j$ then $\pi^D[j] = i$. For example, $\langle 3, 1, 2 \rangle$ is the dual of $\langle 2, 3, 1 \rangle$. Intuitively, $\pi[i]$ refers to the natural number at location i of π, and $\pi^D[j]$ refers to the location of j in π.

The Pancake Puzzle. An N**-pancake puzzle state** is a stack of N different sized pancakes, which is represented by a permutation of size N. The natural number i in this permutation refers to the i-th smallest pancake, and the order of the numbers in the permutation corresponds to the order of the pancakes in the stack from top to bottom. For example, $\langle 2, 1, 4, 3 \rangle$ represents a 4-pancake state in which the second smallest pancake is at the top of the stack.

In any N-pancake state π, there are $N - 1$ applicable **actions** or **moves**, given by M_2, M_3, ..., M_N, and we use $M_i(\pi)$ to denote the permutation that is the result of applying action M_i to π. Each action M_i reverses the order of the first i values in the stack. Formally, this means that $M_i(\pi)[j] = \pi[i-j+1]$ for every $j \leq i$, and $M_i(\pi)[j] = \pi[j]$ for every $j > i$. For example, $M_3(\langle 2, 1, 4, 3 \rangle) = \langle 4, 1, 2, 3 \rangle$.

Definition 1. *Given N-pancake state π_{init}, the N-**pancake puzzle task** is to find the shortest or **optimal** sequence of actions that transforms π_{init} into state $\pi_{\text{goal}} = \langle 1, 2, ..., N \rangle$.*

The N-pancake puzzle task has been shown to be NP-hard (Bulteau, Fertin, and Rusu 2015). For any N, the **diameter** of the N-pancake puzzle — defined as the longest optimal solution to any N-pancake state — is known to be at least $\lfloor \frac{15}{14} \cdot N \rfloor$ (Heydari and Sudborough 1997) and no more than $\lfloor \frac{18}{11} \cdot N \rfloor$ (Chitturi et al. 2009). A 2-approximation algorithm has also been given by Fischer and Ginzinger (2005).

State-Space Search, Heuristics, and IDA*. From the perspective of state-space search, the pancake puzzle task involves finding the lowest cost solution from state π_{init} to state π_{goal} in a **unit-cost** state-space with $N!$ states. The state-space is also **undirected** since applying the same flip twice in a row merely undoes the effect of the original flip.

A state π' is said to be a **neighbour** of state π if there exists an action M_i such that $M_i(\pi) = \pi'$. We also use $h^*(\pi)$ to denote the cost of the optimal solution path starting at π. A **node** n is given by a state, denoted by $n.\text{state}$, and a sequence of flips, called a **path**, that leads to π from π_{init}. The cost of this path is referred to by $g(n)$. Node n' is called a **child** of node n if there is an action M_i such that $n'.\text{state} = M_i(n.\text{state})$ and the path to n' is given by the path to n with the addition of action M_i.

A **heuristic function** h is a function from the set of states to the set of non-negative real numbers. A heuristic function h is **admissible** if for every state π, $h(\pi) \leq h^*(\pi)$. In an undirected unit-cost state-space, h is **consistent** if for every pair of neighbouring states π and π', $|h(\pi) - h(\pi')| \leq 1$. If admissible h, we also define the **absolute heuristic error** or **AHE** of h on state π as $h^*(\pi) - h(\pi)$.

IDA* (Korf 1985) is an algorithm which iteratively performs a sequence of threshold-limited depth-first searches. Given heuristic h, the initial threshold is set as $h(\pi_{\text{init}})$. During each iteration, node n is pruned if its f**-cost**, defined as $f(n) = g(n) + h(n.\text{state})$, is larger than the current threshold. The threshold for iteration $i + 1$ is set as the minimum f-cost of all nodes pruned during iteration i. IDA* is guaranteed to return optimal solutions if h is admissible.

The Gap Heuristic. We define the **gap** heuristic (Helmert 2010), which we denote by h^G, using the **extended** permutation π^e of π. Permutation π^e is defined as $\pi \circ \langle N+1 \rangle$. The value $N+1$ can be thought of as the plate below the pancake stack, though we often refer to it as the $N + 1$-st pancake. Moreover, due to the one-to-one correspondence between π and π^e we often refer to $\pi[N + 1]$, the "$N + 1$-st pancake" of π, or "location $N + 1$" in π.

For any j where $1 \leq j \leq N$, an **adjacency** is said to occur in π^e between locations j and $j + 1$, or between pancakes $\pi^e[j]$ and $\pi^e[j + 1]$, if $|\pi^e[j] - \pi^e[j + 1]| = 1$. A **gap** is said to occur between those locations (or those pancakes) if an adjacency does not occur. The value of $h^G(\pi)$ is then given by the count of the number of gaps in π^e:

$$h^G(\pi) = |\{j \mid 1 \leq j \leq N, |\pi^e[j] - \pi^e[j + 1]| > 1\}|$$

Since any action can only add or remove at most one gap and there are no gaps in π_{goal}, h^G is admissible and consistent.

If action M_i removes a gap when applied to state π (*i.e.* $h^G(M_i(\pi)) = h^G(\pi) - 1$), then M_i is called a **gap decreasing** move in π. Similarly, M_i is a **gap increasing** move if it introduces a gap, while if it replaces one gap with another or one adjacency with another, M_i is a **gap neutral** move.

We note that action M_i will move $\pi[1]$ on top of $\pi[i + 1]$, and so determining if M_i is a gap decreasing, increasing, or neutral move merely involves checking if $|\pi[1] - \pi[i + 1]| = 1$, and if there is a gap between pancakes $\pi[i]$ and $\pi[i + 1]$. This allows for a constant time computation of the difference between $h^G(M_i(\pi))$ and $h^G(\pi)$. Given π and $h^G(\pi)$, we can therefore efficiently calculate $h^G(M_i(\pi))$ without generating $M_i(\pi)$. This incremental computation of h^G, which we use in our experiments, is a well-known optimization that has previously shown to be important in the sliding tile puzzle (Korf 1985; Burns et al. 2012).

Observe that there are always at most two gap decreasing moves in any state. This is because M_i can only resolve a gap (if one exists) between locations i and $i + 1$ if $\pi[1]$ is adjacent to $\pi[i+1]$ in π_{goal}, and this is only true if $\pi[i+1] = \pi[1] + 1$ or $\pi[i + 1] = \pi[1] - 1$. However, in many states there are no gap decreasing moves. These states are said to be **locked**. In such states, we can show the following:

Lemma 2.1. *There is at least one gap neutral action in any non-goal state that is locked.*

This holds because we can always replace one gap or adjacency with another in such states. A complete proof can be found in a technical report (Valenzano and Yang 2017).

3 The Accuracy of the Gap Heuristic

In this section, we demonstrate that while the gap heuristic is very accurate on random permutations, it can be much more

	AHE of h^G			
N	0	1	2	3
16	369	581	49	1
20	335	621	44	0
24	332	642	26	0
28	338	647	15	0
40	340	650	10	0
50	386	609	5	0
60	363	636	1	0

(a) AHE of h^G on $1,000$ random permutations per N.

	Heuristic Function				
AHE	h^G	LD	LD^D	2LD	2LD^D
0	$205,330,493$	$216,267,458$	$224,031,821$	$221,584,129$	$231,096,110$
1	$246,800,263$	$241,319,635$	$237,261,313$	$238,902,035$	$233,210,974$
2	$26,213,570$	$21,050,960$	$17,482,806$	$18,289,424$	$14,566,568$
3	$648,977$	$360,630$	$224,202$	$224,908$	$127,456$
4	$8,216$	$2,906$	$1,457$	$1,103$	491
5	80	10	0	0	0

(b) AHE of different heuristics over all 12-pancake states.

Table 1: The number of states with different AHE values on random permutations and the 12-pancake puzzle.

inaccurate. We also formally and experimentally investigate the worst-case AHE of h^G for any pancake problem size N.

3.1 Accuracy on Random Permutations

We begin by analyzing the accuracy of h^G on random permutations, since this is the standard method for constructing pancake puzzle test sets. We generated and solved $1,000$ random permutation for seven values of N ranging from 16 to 60. For each tested N, Table 1a shows a count of the number of the $1,000$ states with each AHE. No permutation had an AHE of more than 3. For all tested N, h^G was perfect for between 33% and 39% of the problems, and off by no more than one for between 95% and 99.9% of the problems. The standard deviation of the AHE also decreases as N increases, from 0.57 when $N = 16$ to 0.48 when $N = 60$.

While we never encountered a random permutation with an AHE of h^G over 3, an exhaustive search of the 12-pancake puzzle shows that the AHE of h^G can be higher. This can be seen in column two of Table 1b, which shows the number of states with each AHE value encountered. The table shows that the AHE can get as high as 5 in the 12-pancake puzzle. However, states with an AHE of 3 or more are exceedingly rare, consisting of only 0.13% of states.

3.2 Bounds on the Worst-Case AHE of h^G

Let us now consider just how inaccurate the gap heuristic can get, by providing bounds on the worst-case AHE.

Theorem 3.1. *The maximum AHE over all N-pancake states is no smaller than $\lfloor \frac{1}{14} \cdot N \rfloor$ and no larger than $\lfloor \frac{9}{11} \cdot N \rfloor$.*

Proof. For the lower bound, recall that Heydari and Sudborough (1997) identified a family of states, where the state of size N has an optimal cost of at least $\lfloor \frac{15}{14} \cdot N \rfloor$. Since these states have N gaps, they guarantee the existence of a state with an AHE of at least $\lfloor \frac{15}{14} \cdot N \rfloor - N = \lfloor \frac{1}{14} \cdot N \rfloor$.

For the upper bound, let π be the N-pancake state with maximum AHE. By Chitturi et al.'s bound (2009) on the diameter of the the N-pancake puzzle, $h^*(\pi) \leq \lfloor \frac{18}{11} \cdot N \rfloor$. Since Fischer and Ginzinger's 2-approximation algorithm (2005)always finds a solution in at most $2 \cdot h^G(\pi)$ moves, $h^*(\pi) \leq 2 \cdot h^G(\pi)$. As such, if $h^G(\pi) = j$, then the AHE of π is at most $\min(2 \cdot j, \lfloor \frac{18}{11} \cdot N \rfloor) - j$. This expression hits its maximum at the largest value of j for which $2 \cdot j \leq \lfloor \frac{18}{11} \cdot N \rfloor$, at which point the expression has a value of $\lfloor \frac{9}{11} \cdot N \rfloor$. $\square$

While Theorem 3.1 guarantees that the worst-case AHE of h^G grows linearly with N, there is still a discrepancy between the upper and lower bounds provided. To better understand the actual worst-case AHE, we performed an exhaustive search on all states for $N \leq 12$. These experiments showed that the worst-case AHE is exactly $\lfloor \frac{N}{2} \rfloor - 1$ for every $N \leq 12$. For larger values of N, we use a particular state, denoted by FG^2_N, to provide lower bounds on the worst-case AHE. For an even N, $FG^2_N = \langle 2, 1, 4, 3, ..., N, N-1 \rangle$. We have solved this problem, which has $N/2$ gaps, up to size 28. For every $N \leq 18$, $h^*(FG^2_N) = N - 1$ and so the worst-case AHE of h^G for any N where $13 \leq N \leq 19$ is at least $\lfloor \frac{N}{2} \rfloor - 1$.[1] For $20 \leq N \geq 28$, this pattern does not hold, as $h^*(FG^2_N) < N - 1$ in this range. However, our experiments with FG^2_N does show that the worse-case AHE of h^G is at least 8 for $20 \leq N \leq 23$ and at least 9 for $24 \leq N \leq 28$.

We note that FG^2_N is based on a burnt pancake state often denoted by $-I_N$. This puzzle is a variant of the standard pancake puzzle, in which each pancake p has an orientation of either $+p$ or $-p$, and a pancake's orientation changes when it is flipped (Gates and Papadimitriou 1979). The burnt pancake state $-I_N$, which is given by $\langle -1, -2, ..., -N \rangle$, was conjectured to have the longest optimal solution cost of any N burnt pancake state (Cohen and Blum 1995), though this has been shown to be false (Heydari and Sudborough 1997). FG^2_{2N} can be constructed from $-I_N$ by replacing each $-i$ entry in $-I_N$ with two entries $\langle 2i+1, 2i \rangle$. However, the optimal cost for FG^2_{2N} can be lower than for $-I_N$.

4 Generating Harder Pancake Problems

We now identify three methods for generating states on which h^G usually has a higher AHE than it does on random permutations. In doing so, we provide ways of constructing large problem sets that are difficult when using h^G.

[1]For odd-valued N, this follows since for any π, $\pi \circ \langle N+1 \rangle$ has the same optimal cost and number of gaps as π.

4.1 Problem Generation Methods

We had two main objectives when developing new state generation methods. First, we wanted these methods to generate states for which h^G has a higher average AHE than typically seen with random permutations. Secondly, we wanted these methods to easily allow for the generation of large test sets of any size. While Gates and Papadimitriou (1979), Heydari and Sudborough (1997), and Rockicki (2004) have all manually constructed problems that are considered "hard," together they represent only a handful of states for any given N. In contrast, our methods can be used to generate large test sets of states with higher AHE values, and are thus more conducive to studies regarding how an algorithm's behaviour scales with problem size, large-scale and systematic experimentation, or investigations that require separate training and test data. These methods are described below.

Self-Inverses. A permutation π is said to be a **self-inverse** permutation if for all i, $\pi[i] = \pi^D[i]$. This is equivalent to requiring that for all i and j, if $\pi[i] = j$ then $\pi[j] = i$. Notice that this condition allows for an integer to be **mapped** to itself (*i.e.* $\pi[i] = i$). For example, $\pi = \langle 1, 4, 3, 2\rangle$ is a self-inverse state since 1 and 3 are mapped to themselves, $\pi[2] = 4$, and $\pi[4] = 2$.

As shown below, self-inverse states typically have a higher AHE of h^G than random permutations. To generate a self-inverse state π of size N, we use the following iterative procedure. We begin with set $S = \{1, ..., N\}$. On each iteration, there are two possibilities. With probability 0.5, we randomly select and remove two distinct elements e_1 and e_2 from S, and set $\pi[e_1] = e_2$ and $\pi[e_2] = e_1$. Otherwise, we randomly select and remove a single element e from S and set $\pi[e] = e$. This process then continues until S is empty.

Short Cycles. A subset of elements $\{e_1, ..., e_k\}$ is in a k-**cycle** in permutation π with order $\langle e_{(1)}, ..., e_{(k)}\rangle$ if for any j in $1 \le j \le k-1$, $\pi[e_{(j)}] = e_{(j+1)}$, and $\pi[e_{(k)}] = e_{(1)}$. A well-known fact from group theory is that any permutation can be expressed as a set of disjoint cycles. For example, $\langle 6, 3, 5, 2, 4, 1\rangle$ consists of a 2-cycle with order $\langle 1, 6\rangle$ and a 4-cycle with order $\langle 2, 3, 5, 4\rangle$.

Our second state generation method focuses on problems with cycles that satisfy two specific conditions that were found to lead to problems with high AHE. Specifically, each cycle has a size of 4 or less, and each cycle consists of a set of consecutive integers, though these integers do not necessarily appear consecutively in the cycle order. For example, state $\langle 2, 4, 1, 3, 5, 7, 8, 6\rangle$ satisfies these conditions, since the cycles in this state are $\langle 1, 2, 4, 3\rangle$, $\langle 5\rangle$, and $\langle 6, 7, 8\rangle$.

To generate a permutation π of size N of this kind, we begin by uniformly selecting the size of the first cycle from 1, 2, 3, or 4. If k is the value generated, this means that 1, 2, ..., k will appear in π as part of some k-cycle. If $k = 1$, then we set $\pi[1] = 1$. Otherwise, we generate a random ordering $\langle e_1, ..., e_k\rangle$ of 1 to k. We then set $\pi[e_1] = e_2$, ..., $\pi[e_{k-1}] = \pi[e_k]$, and $\pi[k] = e_1$, thus assigning the first cycle. The remaining cycles are set analogously.

Bootstrapping. Our final problem generation method involves concatenating together smaller problems that are known to have a high AHE. For example, consider two "hard" N-pancake puzzle states π and π', where $\pi \ne \pi'$, and let $(\pi + N)$ denote the sequence given by incrementing each entry of π by N (*i.e.* $(\langle 3, 1, 2\rangle + 3) = \langle 6, 4, 5\rangle$). Then four potentially difficult pancake problems of size $2N$ are $\pi\circ(\pi+N')$, $(\pi+N')\circ\pi$, $(\pi+N)\circ\pi'$, and $\pi'\circ(\pi+N)$. More generally, given a set S of N-pancake states and a set S' of N' pancake states, we can generate states of size $N + N'$ by sampling a state from each of S and S', and using one of the four possible combinations of these two states.[2] We call this method **bootstrapping** from **seed sets** S and S'.

Below, we experiment with test sets for the 16, 20, 24, and 28-pancake puzzles, each containing $1,000$ states. For the initial seed sets, we used exhaustive search to find the 50 8-pancake states with the highest AHE values, breaking ties in favour of higher optimal cost. We did the same to get a set of 50 12-pancake problems. We denote these sets as S_8 and S_{12}, respectively. To construct the 16-pancake test set, S_8 was used for both seed sets. For the 20-pancake set, S_8 and S_{12} were used. We then built set S_{16} consisting of 50 of the $1,000$ 16-pancake states generated using bootstrapping, using the same selection criteria as was used for S_8 and S_{12}. The 24-pancake set was built using S_{16} and S_8 as seed sets, while S_{12} and S_{16} were used for the 28-pancake test set.

We note that the 24-pancake test set could have also been constructed using two copies of S_{12} as the seed sets. The decision to do otherwise was made arbitrarily, and was not revisited because the generated problems were already found to have a high AHE for h^G. This highlights a weakness of the bootstrapping approach: many decisions need to be made for any N and it can be difficult to make them consistently for different values of N, as is desirable when testing how techniques scale with N. However, it is still simple to build large sets using this method, and as we will see, they will contain the hardest problems of all those considered.

4.2 Difficulty of New Benchmarks

We evaluated the new problem generation methods by building 16 test sets, one for each combination of four pancake sizes — 16, 20, 24, and 28 — and four problem generation methods, including the use of randomly generated permutations. Each test set contains $1,000$ problems. The benchmarks used and the state generators can be found at `http://bit.ly/2pGEEt0`. Table 2 shows the average, median, maximum, and standard deviation of the AHE of h^G over the states in these sets. The table shows that the new generation methods lead to higher AHE values than is seen with random permutations, and that on these new test sets, the average, median, and maximum AHE generally increase with N. Self-inverse states have the lowest average AHE of the new methods, while bootstrapping has the highest.

To evaluate how the increased AHE impacts runtime, we ran IDA* using h^G on all 16 test sets. The average number of node generations for each combination of N and state generation method is shown in Figure 1. Notice that the vertical axis is in log-scale. The figure shows that the new gen-

[2]If the two sampled states are the same, then there are only two possible unique combinations.

	Problem Generation Method															
	Random				Self-Inverse				Short Cycles				Bootstrapping			
N	Av	Med	Max	SD	Av	Med	Max	SD	Av	Med	Max	SD	Av	Med	Max	SD
16	0.68	1	3	0.57	1.59	2	5	0.79	2.12	2	5	1.06	3.69	4	6	0.85
20	0.72	1	2	0.54	1.71	2	4	0.79	2.69	3	6	1.16	4.99	5	7	0.87
24	0.69	1	2	0.51	1.79	2	5	0.78	3.21	3	7	1.21	5.59	6	8	0.83
28	0.68	1	2	0.50	1.87	2	5	0.80	3.63	4	7	1.18	6.61	7	9	0.86

Table 2: AHE of h^G on 16 different test sets. The table shows the average (Av), median (Med), maximum (Max), and standard deviation (SD) over the $1,000$ problems generated for each generation method and problem size (N) considered.

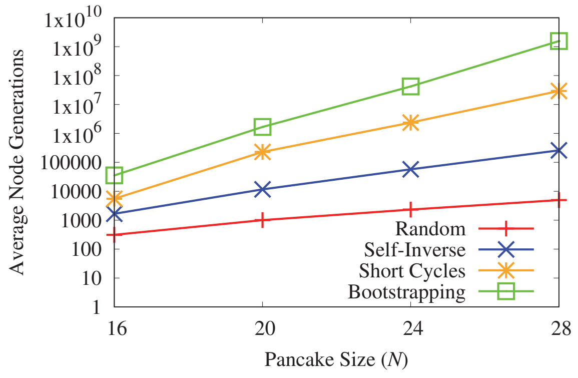

Figure 1: IDA* scaling behaviour on different benchmarks.

eration methods yield substantially harder problems for an IDA* guided by h^G than just randomly generating permutations. This is not due to an increase in average optimal cost, as the opposite is true. For example, the average optimal cost for random permutations when $N = 28$ is 26.8, while it is 24.2, 21.1, and 21.4 for the self-inverse, short cycle, and bootstrapped states, respectively.

These results show that the new generation methods can be used to construct pancake puzzle test tests on which the gap heuristic is less accurate than on randomly generated permutations. As such, these methods provide an alternative way to test how algorithms are affected by heuristic accuracy in a domain with a high branching factor like the pancake puzzle, other than purposefully degrading the heuristic as has been done in other work (Holte et al. 2016).

5 Topology of the Gap Heuristic

In this section, we extend the work of Fischer and Ginzinger (2005) by classifying pancake states according to the size of the plateaus around them. To simplify this analysis, we assume that in all states, there is gap between locations N and $N + 1$. Doing so removes the postfix of a state if it is already sorted, since this portion of the state will have no impact on the number of gaps or the optimal solution cost. For example, where $\pi = \langle 2, 1, 4, 3\rangle$ and $\pi' = \langle 2, 1, 4, 3, 5, 6, 7\rangle$, clearly $h^G(\pi) = h^G(\pi')$ and $h^*(\pi) = h^*(\pi')$.

We begin with some additional notation. Following Hoffmann (2005), a **plateau** for h is a connected set of one or more states that all have the same heuristic value. An **exit** from a plateau with heuristic value ℓ is a state π such that $h(\pi) = \ell$ and there is some neighbour π' of π such that $h(\pi') < h(\pi)$. The **exit distance** of h from a state π is the minimum number of actions needed to reach an exit. Notice that this means that any exit has an exit distance of 0.

Consecutive locations $i, i + 1, ..., i + j$ in a permutation π forms a **strip** of size $j + 1$ if there are no gaps between the pancakes in those locations, and that sequence of locations is maximal (*i.e.* on either side of the strip there is a gap or the end of the permutation). A strip of size 2 or more is **descending** if $\pi[i] > \pi[i + 1] > ... > \pi[i + j]$, and **ascending** otherwise. Two strips from i to $i + j$ and i' to $i' + j'$ where $i \leq i + j < i' \leq i' + j'$ are **in order** if the pancakes in the strip from i to j are smaller than the pancakes in the strip from i' to j'. The **rightmost** strip is the one ending at location N. For example, $\langle 1, 2, 3, 5, 4\rangle$ has two strips: an ascending strip of size 3 from locations 1 to 3, and a descending strip of size 2 from location 4 to 5. The latter strip is the rightmost and the two strips are in order.

We now define the following family of states:

Definition 2. *π is a **Fischer-Ginzinger (FG)** state if and only if π has at least two strips, and all strips in π are descending, have a size of at least two, and are in order.*

For example, $\langle 3, 2, 1, 5, 4\rangle$ is an FG state, while $\langle 1, 2, 4, 3\rangle$ and $\langle 2, 1, 3, 5, 4\rangle$ are not FG states since they have an ascending strip and strip of size 1, respectively. The FG^2_N states in Section 3.2 are also FG states in which all strips have size 2. Notice that all FG states are locked.

We can now characterize states according to their exit distance. First, notice that for any state in which there is a gap decreasing move, the exit distance is 0. For locked states, consider the following corollary of Lemma 5 from Fischer and Ginzinger (2005):

Corollary 5.1. *The exit distance of h^G for any locked state that is not an FG state is 1.*

Fischer and Ginzinger proved this by providing appropriate sequences of actions that could decrease the number of gaps for all possible cases of non-FG locked states. However, their result does not characterize the exit distance of FG states. To do so, we define an **easy FG state** as an FG state with exactly 2 strips such that the rightmost strip has a size of 2. The remaining FG states are called **hard FG states**. We can now show the following:

Theorem 5.2. *The exit distance of h^G is 1 for any easy FG state and 2 for any hard FG state.*

Proof Sketch. The proof of this statement can be found in its entirety in a technical report (Valenzano and Yang 2017). The following is a sketch of this proof.

If π is an easy FG state, it has the following form $\langle N-2, ..., 1, N, N-1\rangle$. As such, π has two gaps, one of which can be removed by applying M_{N-1} and then M_N to reach state $\pi' = \langle N-1, N-2, ..., 1, N\rangle$.

If π is a hard FG state π, let $\ell \geq 2$ be the size of the rightmost strip. Then we can easily verify that the following action sequence removes a gap: M_N, M_ℓ, and then M_N. Thus, the exit distance is no more than 2. We show that the exit distance is greater than 1 by contradiction. If it is 1, then there exists a move M_i such that $M_i(\pi)$ is not locked. Since π is an FG state, π is locked, and so M_i is either gap increasing or gap neutral. If it is gap increasing, the consistency of h^G guarantees that the exit distance is at least 2. Otherwise, $M_i(\pi)$ can be shown to be locked in all cases, which is done in the technical report. This contradiction ensures that the exit distance of π cannot be 1. □

The results above show that the exit distance of any pancake state is at most 2. In fact, for any locked state, the actions that lead to the nearest exit consist solely of gap neutral moves. This means that the gap heuristic does not have any **heuristic local minima**, and every state π can be solved suboptimally along a path that does not include a gap increasing move. Such a solution can also be shown to be found by Fischer and Ginzinger's two-approximation algorithm (2005).

However, there are states that cannot be solved optimally without using a gap increasing moves, which we verified experimentally using an IDA* instance that pruned such moves and occasionally was found to return suboptimal solutions.

6 Enhancing the Gap Heuristic

In this section, we use the local topology analysis from Section 5 to develop several enhancements for the gap heuristic.

6.1 Heuristic Lookahead

We begin by defining a d**-step heuristic lookahead** of heuristic h on state π. Let us first assume that no goal state can be reached from π along a path of no more than d actions. In that case, the optimal path from π to a goal state must pass through one of the descendants of π at depth d. In a unit-cost domain, this means that the minimum value of $d+h(\pi')$ seen over all descendants at depth d from π will be an admissible estimate of the cost to reach the goal from π. We refer to this value as the d-step lookahead estimate for π. In the case that the goal state can be reached from π in fewer than d steps, then the lookahead must return the cost of the shortest such path to ensure admissibility.

While the estimates provided by a d-step lookahead on h^G should be at least as accurate as h^G alone, a naive implementation of this technique would require the generation of $O((N-1)^d)$ states at depth d. However, we will show that due to the topology of the gap heuristic, we can efficiently compute the lookahead of h^G for depths of one and two. Note, we use $L_1^G(\pi)$ and $L_2^G(\pi)$ to denote the estimates returned by a 1 and 2-step lookahead of h^G on state π.

6.2 Lock Detection

Let us now consider the estimates returned by a 1-step lookahead of h^G on non-goal state π. Assume that M_i is the action that leads to the neighbour of π with the lowest heuristic value. By definition, this means that $L_1^G(\pi) = h^G(M_i(\pi)) + 1$. If M_i is a gap decreasing move, this value will be $h^G(M_i(\pi)) + 1 = h^G(\pi) - 1 + 1 = h^G(\pi)$. If M_i is not a gap decreasing move, then π must be locked. Since there is always a gap neutral move applicable in any non-goal locked state by Lemma 2.1, $L_1^G(\pi) = h^G(M_i(\pi)) + 1 = h^G(\pi) + 1$ in this case. Therefore, the value of $L_1^G(\pi)$ will be $h^G(\pi) + 1$ if π is locked and $h^G(\pi)$ otherwise.

As such, we can determine the value of $L_1^G(\pi)$ with a linear scan of π in search of the locations of pancakes $\pi[1] + 1$ and $\pi[1] - 1$. If there is a gap above either of these pancakes, then there is a gap decreasing move in π and the value of $L_1^G(\pi)$ is $h^G(\pi)$. Otherwise, $L_1^G(\pi) = h^G(\pi) + 1$. We refer to this calculation of L_1^G as **lock detection** or **LD**.

LD as Early Checking. We note that LD is not entirely new, as it can be viewed as a slight variant of an often used IDA* enhancement that we call **early checking**. This domain-specific enhancement for IDA* is applicable when using heuristics like h^G, for which we can incrementally calculate the heuristic value of a child state $M_i(\pi)$ given π and $h^G(\pi)$, without actually generating $M_i(\pi)$, . In such cases, we can "check" if the f-cost of a node n is higher than the current IDA* threshold, and thereby prune n "early" without generating it if its $f(n)$ does exceed the threshold. This pre-generation pruning is especially important in the pancake puzzle, where generating $M_i(\pi)$ is $O(N)$.

When using h^G, we refer to this method as **gap early checking**. Gap early checking will actually generate the exact same set of nodes as LD, since neither will generate any neighbours of a locked state π if $f(\pi)$ is equal to the current threshold. The main difference between gap early checking and LD is that LD aggressively checks each move to see if one is gap decreasing, while gap early checking does so "lazily." However, we can also construct an incremental version of LD that allows us to compute $L_1^G(M_i(\pi))$ without generating $M_i(\pi)$, and thus perform LD early checking. The same will be true of the other enhancements discussed below, though we omit the details for the sake of clarity.

LD is also related to the EPEIDA* algorithm (Goldenberg et al. 2014). Using h^G with this IDA* variant essentially involves using the incremental h^G computation to define an **operator selection function (OSF)**, which directly returns the children of a node that satisfy the current IDA* threshold. Both an IDA* using heuristic LD and an EPEIDA* using h^G would examine the same set of nodes, and merely differ in their formulation. That is, the increased pruning seen with the LD formulation is the result of an improved heuristic, while the equivalent pruning in EPEIDA* is the result of using the OSF to perform partial node expansion.

6.3 Two-Level Lock Detection

We now show that due to the topology of the pancake puzzle, we can also compute the estimate returned by a 2-step lookahead of h^G for any state π in linear time and without

```
**TwoLevelLockCheck**(permutation $\pi$):
1: $locked \leftarrow$ TRUE, move index $i \leftarrow 2$
2: while ($locked ==$ TRUE) and $i \leq N$ do
3:     if $|\pi[i+1] - \pi[1]| == 1$ and
           $|\pi[i+1] - \pi[i]| \neq 1$ then
4:         $locked \leftarrow$ LevelTwo($\pi, i, \pi[i]$)
5:     $i \leftarrow i + 1$
6: return $locked$

**LevelTwo**($\pi$, move index $i$, pancake $p$):
7: $locked \leftarrow$ TRUE, location $\ell \leftarrow 1$
8: while ($locked ==$ TRUE) and $\ell \leq N$ do
9:     if $|\pi[\ell] - p| == 1$ then
10:        if ($i > \ell$ and $|\pi[\ell] - \pi[\ell+1]| \neq 1$) or
               ($i < \ell$ and $|\pi[\ell] - \pi[\ell-1]| \neq 1$) then
11:            return FALSE
12:    $\ell \leftarrow \ell + 1$
13: return $locked$
```

Algorithm 1: Linear time check if all gap decreasing moves in π lead to locked states.

generating any neighbours of π. We do so by considering different cases for π. If $h^G(\pi) = 1$, it is easy to show that there is a gap decreasing action in π and $h^*(\pi) = 1$. Therefore, $L_2^G(\pi) = 1$. As such, we now assume $h^G(\pi) > 1$. Without loss of generality we also assume that there is a gap between locations N and $N+1$ of π, as in Section 5.

Let us start with the case that π is locked. If π is a hard FG state, then the minimum heuristic value of any state at depth 2 from π is $h^G(\pi)$ by Theorem 5.2. As such, $L_2^G(\pi) = h^G(\pi) + 2$. Otherwise, π is an easy FG state or an non-FG locked state. In these cases, Corollary 5.1 and 5.2 guarantee that $L_2^G(\pi) = h^G(\pi) + 1$.

Now recall that we showed that locks could be detected with a linear time scan in the previous section. FG detection and classification can also clearly be done during this scan, simply by keeping track of the size, number, and orientation of the encountered strips. As such, we can detect a lock and compute $L_2^G(\pi)$ in time linear in N whenever π is locked.

Let us now show that we can also compute $L_2^G(\pi)$ efficiently when π is not locked. If π is not locked and either gap decreasing move leads to another state that is not locked, then there is a state at depth 2 with a heuristic value of $h^G(\pi) - 2$. As such, $L_2^G(\pi) = h^G(\pi)$. If all gap decreasing moves in π lead to locked states, then $L_2^G(\pi) = h^G(\pi) + 1$. This holds because if M_i is a gap decreasing move and $M_i(\pi)$ is locked, then there is a gap neutral move in $M_i(\pi)$ by Lemma 2.1, and so some neighbour of $M_i(\pi)$ must also have a heuristic value of $h^G(M_i(\pi)) = h^G(\pi) - 1$.

Computing $L_2^G(\pi)$ when π is not locked, thus merely requires us to identify if there is a gap decreasing move in π that does not lead to a locked state. Algorithm 1 provides a linear time method for this purpose. The algorithm begins with a call to ***TwoLevelLockCheck***, which scans π in search of index i such that M_i is a gap decreasing move in π. When such a move is found in line 3, a second function, ***LevelTwo***, is called to check if M_i leads to a locked state. Of note, *LevelTwo* does this in linear time and without generating $M_i(\pi)$.

The parameters given to *LevelTwo* are π, the index i of the gap decreasing move M_i currently being checked, and the pancake p that would be on top of the stack in $M_i(\pi)$. *LevelTwo* scans π looking for the pancakes that should be adjacent to p. When such a pancake $\pi[\ell]$ is found in line 9, the function checks if there is a gap beside $\pi[\ell]$ in $M_i(\pi)$ that can be removed. There are two possible cases to consider.[3] If $i > \ell$, then $\pi[\ell+1]$ would be on top of $\pi[\ell]$ in $M_i(\pi)$. The algorithm therefore checks if there is a gap between $\pi[\ell]$ and $\pi[\ell+1]$ (line 10). If $i < \ell$, then $\pi[\ell-1]$ will be on top of $\pi[\ell]$ in $M_i(\pi)$, so the function checks for a gap between those locations. If a gap is found, then there is a descendant of π at depth 2 with a heuristic value $h^G(\pi) - 2$ and so $L_2^G(\pi) = h^G(\pi)$ (line 11). Otherwise, we have to check the other possible gap decreasing actions.

Because there are at most two gap decreasing moves in π, the linear scan of *LevelTwo* is only done at most twice. As such, $L_2^G(\pi)$ can be computed in time linear in N if π is not locked. Since the same is true for locked states as shown above, $L_2^G(\pi)$ can always be computed without generating the $O(N^2)$ nodes needed to naively do the lookahead.

We call the computation of $L_2^G(\pi)$ by detecting and classifying locked states and using *TwoLevelLockCheck* for unlocked states, as **two-level lock detection** or **2LD**. Note that in our implementation, the detection and classification of locked states is done concurrently with the top-level scan in *TwoLevelLockCheck*.

6.4 Using the Dual State

We now consider two ways for improving performance when using LD or 2LD that exploit a state's dual. While maintaining the dual state during the search will mean that applying each action takes twice as long, the improvements in heuristic accuracy will overcome this extra cost.

Zahavi *et al.* (2008) showed that for any state π, $h^*(\pi) = h^*(\pi^D)$. If h is an admissible heuristic, this means that $\max(h(\pi), h(\pi^D))$ is an admissible — and potentially more accurate — heuristic estimate for π. Unfortunately, this technique does not improve h^G, since $h^G(\pi) = h^G(\pi^D)$ for any π, because inverting a state preserves adjacencies. In particular, if there is an adjacency between pancakes p_1 and p_2 which are in locations j and $j+1$ of π, then $|p_1 - p_2| = 1$. As such, there is an adjacency between pancakes j and $j+1$ in π^D since they are in consecutive locations p_1 and p_2.

However, a lookahead of h^G can return a different value for π and π^D. For example, $\pi = \langle 2,3,1,5,4 \rangle$ has a gap decreasing move M_2, while $\pi^D = \langle 3,1,2,5,4 \rangle$ is locked. As such, we can further improve heuristic accuracy by taking the maximum of either LD or 2LD over π and π^D. We refer to the resulting techniques as **LD**^{D} and **2LD**^{D}.

The dual of a state π can also be used to check if π is locked in constant time. To see this, recall that a gap decreasing move in π must resolve a gap above either $\pi[1]+1$ or $\pi[1]-1$. Moreover, the location of $\pi[1]+1$ in π is given immediately by $\pi^D[\pi[1]+1]$. Finding and checking for the first gap decreasing move can now be done by checking for a gap between locations $\pi^D[\pi[1]+1]$ and $\pi^D[\pi[1]+1]-1$. If $\pi[1] > 1$, finding and checking for a gap above $\pi[1]-1$ can

[3] $\ell \neq i$ since otherwise $p = \pi[\ell]$, which contradicts line 9.

then done similarly. Since the dual of π^D is π, we can also use an analogous approach to check if π^D is locked. Thus, LD^D can be performed in constant time.

The above process can also be extended to perform the *TwoLevelLockCheck* component of 2LD^D on both π and π^D in constant time. This leaves FG detection and classification as the only linear time component of 2LD^D.

6.5 Accuracy of the Enhanced Heuristics

Let us now consider how these enhancements impact heuristic accuracy on all 12! states in the 12-pancake puzzle. Table 1b show counts of the number of states with each possible AHE value when using the standard h^G heuristic, and then when adding these enhancements. The table shows that LD, LD^D, 2LD, and 2LD^D improved the heuristic values of 6.1%, 10.3%, 9.2%, and 14.1%, respectively, of all states on which h^G was not already perfect. The enhancements are particularly effective when h^G is more inaccurate. For example, on states for which h^G has an AHE of 3 or higher, heuristics LD, LD^D, 2LD, and 2LD^D improved the heuristic values of 45.5%, 66.7%, 66.7%, and 81.7%, respectively.

7 Evaluation of the Heuristic Enhancements

In this section, we experiment with the h^G enhancements on the 16 test sets defined in Section 4.2 which range over four different values for N and the four different problem generation methods. All experiments were performed using an IDA* enhanced by early checking (see Section 6.2) on a machine with a Intel Xeon W3550 3.07GHz processor and 8 MB of cache. The C++ code for these experiments can be found at `http://bit.ly/2pGEEt0`.

We begin by considering the performance of IDA* on the 24-pancake puzzle. The average number of nodes generated and the runtime in seconds on each 24-pancake puzzle test set is shown in Table 3. For each metric, the relative improvement made when using each enhancement over using just h^G is shown in parentheses. For example, the bottom-right entry of the table shows that using 2LD^D improved runtime by a factor of 5.4 over h^G.

The table shows that the enhancements substantially decrease the number of nodes generated, even on random permutations where h^G is already quite accurate. The relative improvement increases as the average AHE of h^G increases, with the largest improvement seen on the bootstrapped states. However, runtime does not improve as much as node generations. For example, on the bootstrapped states, 2LD^D generates 9.8 times fewer nodes while runtime decreases by a factor of 5.4. This occurs because the enhancements increase the per-node time needed to compute heuristic values.

Table 3 also shows that LD^D and 2LD lead to similar gains. However, the improvements seen when using lookahead and a state's dual appear to be quite complementary, given how much further improvement is seen with 2LD^D.

The impact on both node generations and time seen when using the heuristic enhancements appears to remain relatively constant as N increases. One exception is with states with short cycles, on which the impact of 2LD^D increases with N. This is seen in Figure 2, which shows the average

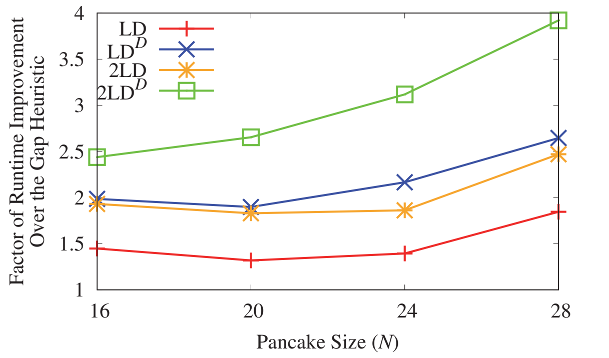

Figure 2: Runtime improvement when using h^G enhancements over h^G alone on problems with short cycles.

runtime improvement over h^G when using each enhancement over different values of N. In contrast, the improvement generally stays similar on the states generated from bootstrapping. For example, these problems take an average of 12.85 minutes each to solve when using IDA* with h^G and early checking when $N = 28$. 2LD^D generated 11.6 times fewer nodesthan h^G, and the runtime improved by a factor of 4.5. These values are similar to the improvements shown in Table 3 that were seen when using 2LD^D on the 24-pancake states generated with bootstrapping.

8 Related Work

Bouzy (2015) evaluated Monte Carlo search methods for suboptimally solving standard and burnt pancake problems. One enhancement considered was a depth-limited search on just the gap decreasing moves, which was used to detect if applying a given action in a given state would lead to only locked states within some horizon. While this technique is related to a d-step heuristic lookahead, it was not used to improve admissible heuristic estimates as we do.

A depth-first lookahead was used by Stern et al. (2010) to improve memory efficiency in an A* search. This lookahead examined and pruned states according to f-cost instead of by depth. It was also a domain-independent technique and thus had to generate all relevant descendant states.

Felner et al. (2010) also considered a 1-step lookahead in the pancake puzzle as part of a bi-directional search. However, their lookahead required the generation of all children and was done in the context of a pattern database heuristic.

Several works have analyzed the accuracy and local search topology of different panning heuristics. Examples include the local search topology analysis of the delete relaxation heuristic by Hoffmann (2005), and a comparison of admissible heuristics by Helmert and Mattmüller (2008).

9 Conclusion and Future Work

In this work, we have shown that the gap heuristic is generally very accurate on a large percentage of problems, and thus rarely underestimates the optimal cost of any state by more than two. However, it is more inaccurate on certain problems and we have shown that this worst-case inaccuracy grows with the puzzle size. We then identified methods

	Problem Generation Method							
	Random		Self-Inverse		Short Cycles		Bootstrapping	
Heuristic	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
h^G	$2,323(1.0)$	$0.0009(1.0)$	$57,361(1.0)$	$0.021(1.0)$	$2,361,650(1.0)$	$0.84(1.0)$	$42,037,553(1.0)$	$28.0(1.0)$
LD	$1,756(1.3)$	$0.0008(1.1)$	$39,610(1.4)$	$0.017(1.2)$	$1,339,064(1.8)$	$0.60(1.4)$	$18,894,828(2.2)$	$15.4(1.8)$
LD^D	$1,278(1.8)$	$0.0006(1.5)$	$28,539(2.0)$	$0.013(1.6)$	$847,500(2.8)$	$0.39(2.2)$	$10,069,646(4.2)$	$8.3(3.4)$
2LD	$1,381(1.7)$	$0.0007(1.3)$	$28,289(2.0)$	$0.015(1.4)$	$824,297(2.9)$	$0.45(1.9)$	$9,347,059(4.5)$	$9.4(3.0)$
2LD^D	$898(2.6)$	$0.0005(1.8)$	$18,293(3.1)$	$0.011(1.9)$	$434,467(5.4)$	$0.27(3.1)$	$4,308,345(9.8)$	$5.2(5.4)$

Table 3: Performance of the enhancements on different 24-pancake benchmarks. Values shown are the average over $1,000$ test problems. Time is in seconds. The factor of that each enhancement improves over h^G for each metric is shown in parentheses.

for easily generating large sets of problems that can be used to test how search algorithms behave on those pancake problems with more heuristic error.

We also analyzed the local search topology of the gap heuristic, and classified all states in terms of the size of the plateaus around them. This classification was then used as the basis of several gap heuristic enhancements that efficiently calculate a heuristic lookahead and also exploit a state's dual in order to improve heuristic accuracy. These enhancements were also shown to lead to substantial speedups.

In recent work, it has been shown that the use of bidirectional search and partial node expansions can outperform IDA* on the pancake puzzle (Lippi, Ernandes, and Felner 2016). Given that our enhancements strictly improve on the gap heuristic and that the relative impact of the extra per-node cost of using the enhancements should decrease in that setting, we would expect to see similar trends. However, we leave such an investigation as future work.

Acknowledgements

We thank Sheila McIlraith and the Knowledge Representation group at the University of Toronto for their support on this work. We also thank the anonymous reviewers for their comments, particularly in connecting the work with existing research and helping to make Lemma 2.1 properly precise.

References

Bouzy, B. 2015. An Experimental Investigation on the Pancake Problem. In *Proceedings of the Fourth Workshop on Computer Games(CGW) and the Fourth Workshop on General Intelligence in Game-Playing Agents, (GIGA)*, 30–43.

Bulteau, L.; Fertin, G.; and Rusu, I. 2015. Pancake Flipping is hard. *Journal of Computer and System Sciences* 81(8):1556–1574.

Burns, E. A.; Hatem, M.; Leighton, M. J.; and Ruml, W. 2012. Implementing Fast Heuristic Search Code. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search*.

Chitturi, B.; Fahle, W.; Meng, Z.; Morales, L.; Shields, C.; Sudborough, I. H.; and Voit, W. 2009. An (18/11)n upper bound for sorting by prefix reversals. *Theoretical Computer Science* 410(36):3372–3390.

Cohen, D. S., and Blum, M. 1995. On the Problem of Sorting Burnt Pancakes. *Discrete Applied Mathematics* 61(2):105–120.

Felner, A.; Moldenhauer, C.; Sturtevant, N. R.; and Schaeffer, J. 2010. Single-Frontier Bidirectional Search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 59–64.

Fischer, J., and Ginzinger, S. W. 2005. A 2-Approximation Algorithm for Sorting by Prefix Reversals. In *Proceedings of the 13th Annual European Symposium (ESA)*, 415–425.

Gates, W., and Papadimitriou, C. 1979. Bounds for sorting by prefix reversal. *Discrete Math* 27:47–57.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N. R.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced Partial Expansion A*. *Journal of Artificial Intelligence Research* 50:141–187.

Hayes, B. 2007. Sorting out the genome. *American Scientist* 95:386–391.

Helmert, M., and Mattmüller, R. 2008. Accuracy of admissible heuristic functions in selected planning domains. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence,*, 938–943.

Helmert, M. 2010. Landmark Heuristics for the Pancake Problem. In *Proceedings of the Third Annual Symposium on Combinatorial Search*.

Heydari, M. H., and Sudborough, I. H. 1997. On the Diameter of the Pancake Network. *Journal of Algorithms* 25(1):67–94.

Hoffmann, J. 2005. Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Holte, R. C.; Felner, A.; Sharon, G.; and Sturtevant, N. R. 2016. Bidirectional search that is guaranteed to meet in the middle. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 3411–3417.

Korf, R. E. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence* 27(1):97–109.

Lippi, M.; Ernandes, M.; and Felner, A. 2016. Optimally solving permutation sorting problems with efficient partial expansion bidirectional heuristic search. *AI Communications* 29(4):513–536.

Qiu, K.; Meijer, H.; and Akl, S. G. 1991. Parallel Routing and Sorting of the Pancake Network. In *Proceedings of the International Conference on Computing and Information*, 360–371.

Rockicki, T. 2004. Tom's Pancake Entry. http://tomas.rokicki.com/pancake/.

Stern, R.; Kulberis, T.; Felner, A.; and Holte, R. 2010. Using Lookaheads with Optimal Best-First Search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Valenzano, R. A., and Yang, D. 2017. A Formal Characterization of the Local Search Topology of the Gap Heuristic. *CoRR*.

Zahavi, U.; Felner, A.; Holte, R. C.; and Schaeffer, J. 2008. Duality in permutation state spaces and the dual search algorithm. *Artificial Intelligence* 172(4-5):514–540.