Landmark Heuristics for the Pancake Problem

Malte Helmert
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee 52
79110 Freiburg, Germany
helmert@informatik.uni-freiburg.de

Abstract

We describe the *gap heuristic* for the pancake problem, which dramatically outperforms current abstraction-based heuristics for this problem. The gap heuristic belongs to a family of *landmark heuristics* that have recently been very successfully applied to planning problems.

Introduction

The pancake problem is a famous search problem (e. g., Dweighter 1975; Gates and Papadimitriou 1979; Heydari and Sudborough 1997) where the objective is to sort a sequence of objects (*pancakes*) through a minimal number of prefix reversals (*flips*).

A state of the n*-pancake problem* represents a stack of n pancakes of different size, commonly given as a permutation in sequence notation. To ease notation later on, we represent pancake stacks as sequences over $\{1, \ldots, n+1\}$, where the last sequence element is always $n+1$, representing the "plate" on which the n pancakes are arranged. Successor states are obtained by flipping $k \in \{2, \ldots, n\}$ pancakes at the top of the stack (a k*-flip*, denoted by F_k), i. e., by reversing the order of the first k sequence elements. The goal is to transform a given state into the identity permutation with as few flips as possible. Figure 1 shows a 6-pancake instance with initial state $\langle 3, 2, 5, 1, 6, 4, 7\rangle$. An optimal solution is given by the flip sequence $\langle F_5, F_6, F_3, F_4, F_5\rangle$.

The prevalent approach for the pancake problem in the heuristic search literature is the use of *pattern database* (PDB) heuristics; the most important representatives are the nonadditive PDB heuristics of Zahavi et al. (2008) and the additive PDB heuristics of Yang et al. (2008). The literature does not provide experimental results for these approaches for instances with more than 17 pancakes. Our own experiments with the solver of Zahavi et al. suggest that it does not reliably scale to instances of size beyond 20 within usual memory constraints and a one-day timeout. According to Yang et al. (personal communications), neither does their approach. Here, we describe an alternative heuristic, not based on abstraction, that optimally solves instances with up to 60 pancakes in a matter of seconds for most cases and minutes for hard cases.

Figure 1: A 6-pancake instance ($s = \langle 3, 2, 5, 1, 6, 4, 7\rangle$).

The Gap Heuristic

Let $s = \langle s_1, \ldots, s_{n+1}\rangle$ be an n-pancake state. Its heuristic value is the number of stack positions for which the pancake at that position is *not* of adjacent size to the pancake below:

$$h^{\text{gap}}(s) := |\{i \mid i \in \{1, \ldots, n\}, |s_i - s_{i+1}| > 1\}|.$$

We get $h^{\text{gap}}(s) = 5$ for the state in Fig. 1 because there are 5 *gaps* in this pancake stack, namely below positions 2, 3, 4, 5, and 6. For example, there is a gap below position 2 because the 2nd and 3rd pancake in the sequence differ in size by more than 1, and there is a gap below position 6 because the 6th pancake and the "plate" differ in size by more than 1. The only place *without* a gap is below position 1, since the two first pancakes in the sequence are of adjacent size. A pancake problem goal state has no gaps at all, and hence its heuristic value is 0. It is easy to see that a k-flip can reduce the number of gaps by at most 1: the only gap it can potentially "heal" is the one between positions k and $k + 1$. Hence, h^{gap} is a consistent and admissible heuristic.

As far as we know, the gap heuristic has not been previously proposed in the academic literature. However, it was used by Tom Rokicki in his winning entry to a 2004 programming contest for finding short (possibly suboptimal) solutions for the pancake problem.[1] Our discovery of the heuristic is independent from Rokicki's and was inspired by the article by Gates and Papadimitriou (1979), the first academic paper on the pancake problem. Gates and Papadimitriou show that the n-pancake state space ($n \geq 4$) has a diameter of at least n because there exist arrangements with n gaps, and flips can only eliminate one gap at a time. (Instead of *gaps*, they speak of *adjacencies*, the absence of gaps.) We call the heuristic h^{gap} because it counts gaps and because it is based on an idea of **G**ates **a**nd **P**apadimitriou.)

[1] See http://tomas.rokicki.com/pancake/.

Evaluation

We implemented the gap heuristic within a standard IDA* algorithm and evaluated it on instances with 2–60 pancakes. The results in Tab. 1 show that the heuristic scales much better than previous approaches. Further improvements should be possible, e. g. by making the heuristic computation incremental and by ordering gap-removing moves first.

As mentioned in the introduction, current PDB-based approaches do not reliably scale beyond size 19 or 20. The advantage of h^{gap} can be explained by theoretical considerations: a PDB which distinguishes the k largest pancakes, as considered by Zahavi et al., never gives heuristic values beyond $2k$. For a 60-pancake instance, a size-6 PDB of this form already has at about 36 billion entries, yet its heuristic values are bounded by 12. By contrast, we can show analytically that the expected value of h^{gap} on a random n-pancake state is $n - 2 + \frac{1}{n}$, i. e., about 58.02 for size-60 instances.

Beyond Pancakes

The gap heuristic can be seen in a wider context as a special case of an *admissible landmark heuristic*, a family of heuristics which set the current state of the art in optimal classical planning (Helmert and Domshlak 2009). Indeed, if we define the pancake problem as a STRIPS planning domain in a natural way (as a thought experiment – the representation would be too large in practice), standard polynomial-time techniques based on delete relaxation can prove that disjunctions of the form $\mathit{on}(i, i+1) \vee \mathit{on}(i+1, i)$, expressing that there is no gap between pancakes i and $i+1$, are landmarks (Richter, Helmert, and Westphal 2008). This means that they must be achieved at some point in any solution, and by using these landmarks within the admissible landmark heuristic of Karpas and Domshlak (2009), we can exactly recover h^{gap}. This shows that techniques based on landmarks and delete relaxation can be fruitfully applied to classical permutation puzzles. In the future, we would like to further explore this idea in the context of other puzzles, such as TopSpin.

References

Dweighter, H. 1975. Elementary problem E2569. *The American Mathematical Monthly* 82(10):1010.

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

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, 162–169.

Heydari, M. H., and Sudborough, I. H. 1997. On the diameter of the pancake network. *Journal of Algorithms* 25(1):67–94.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *Proc. IJCAI 2009*, 1728–1733.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. AAAI 2008*, 975–982.

Yang, F.; Culberson, J.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *JAIR* 32:631–662.

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

n	L^*	$h(\mathit{init})$	nodes	time
2	0.515	0.515	2	0.002
3	1.503	1.338	3	0.002
4	2.506	2.249	7	0.002
5	3.545	3.211	13	0.002
6	4.601	4.193	25	0.002
7	5.601	5.144	46	0.002
8	6.677	6.155	87	0.002
9	7.721	7.186	164	0.002
10	8.692	8.093	293	0.002
11	9.732	9.034	517	0.002
12	10.689	9.974	782	0.002
13	11.791	11.078	1456	0.002
14	12.715	11.970	2042	0.002
15	13.735	12.979	3527	0.002
16	14.809	14.085	4264	0.003
17	15.770	15.088	6279	0.003
18	16.673	15.937	10295	0.003
19	17.707	16.999	12824	0.004
20	18.783	18.070	17050	0.005
21	19.707	19.015	24758	0.006
22	20.801	20.083	33120	0.008
23	21.721	21.019	40844	0.009
24	22.749	22.066	58086	0.013
25	23.723	23.000	76054	0.017
26	24.740	24.047	101902	0.022
27	25.741	25.052	116458	0.025
28	26.760	26.081	167192	0.036
29	27.683	27.009	162738	0.036
30	28.730	28.059	225059	0.050
31	29.688	28.965	314542	0.069
32	30.732	30.068	333153	0.074
33	31.684	31.024	455745	0.103
34	32.707	32.045	549673	0.126
35	33.731	33.053	762250	0.175
36	34.751	34.093	926060	0.217
37	35.694	35.040	930314	0.222
38	36.693	36.037	1308683	0.318
39	37.675	36.983	1817656	0.444
40	38.670	38.012	1913381	0.476
41	39.668	39.032	1793336	0.455
42	40.693	40.066	3096624	0.793
43	41.687	41.040	3227706	0.842
44	42.722	42.069	4511476	1.197
45	43.635	43.027	5127214	1.386
46	44.635	43.995	6368582	1.739
47	45.709	45.071	7076578	1.961
48	46.689	46.071	10832404	3.074
49	47.627	46.997	11700248	3.345
50	48.626	48.000	14933748	4.303
51	49.663	49.046	14654504	4.314
52	50.741	50.103	19757127	5.898
53	51.688	51.061	25437072	7.705
54	52.703	52.095	29253338	8.996
55	53.644	53.017	42889898	13.422
56	54.700	54.053	48810889	15.395
57	55.649	55.025	52892442	16.948
58	56.611	55.967	62612914	20.476
59	57.697	57.084	87169465	28.741
60	58.578	57.947	95385185	31.931

Table 1: IDA* + h^{gap} performance. Each row gives averages for 1000 instances selected uniformly randomly. The columns denote problem size, optimal solution length, initial h^{gap} value, generated nodes, and runtime in seconds.