Burnt Pancake Problem: New Lower Bounds on the Diameter and New Experimental Optimality Ratios

Bruno Bouzy

LIPADE-TR-N° 3

May 2016

Technical Report

LIPADE

Laboratoire d'Informatique PAris DEscartes

Burnt Pancake Problem: New Lower Bounds on the Diameter and New Experimental Optimality Ratios

Bruno Bouzy[1]

[1]*LIPADE, Paris Descartes University, France*

Abstract

In this paper, we present new findings on the burnt pancake problem. First, while previously exact values of $g(-I_N)$ were known for $N \leq 20$, we provide new exact values for $N \leq 27$. It results new lower bounds on the diameter for $N \leq 27$. These results are obtained with IDA* with the number of breakpoints as the heuristic function. Furthermore, by means of a new heuristic function proven to be admissible on stacks of size at most 15, we provide upper bounds on $g(-I_N)$ for $N \leq 30$. While the exact value of the diameter is known for $N \leq 17$, with this new heuristic, we uncover some hard positions, different from $-I_N$ for $N \leq 22$. Our new heuristic function uses the number of breakpoints, the number of anti-adjacencies, and a new feature, the number of wrong adjacencies. Secondly, on stacks with a very high size ($N \leq 256$), we give experimental optimality ratio obtained with nested MCS using Cohen and Blum's algorithm as basic simulations.

Keywords: *Burnt Pancake Problem, Planning, Heuristic Search, Monte-Carlo Search*

1. Introduction

The *burnt* pancake problem is described as follows. A chef prepares a stack of pancakes that come out all of different sizes on a plate. Each pancake has a *burnt* side. The goal of the server is to order them with decreasing sizes, the largest pancake touching the plate, the smallest pancake being at the top, and the *burnt* side being *down*. The server can insert a spatula below a pancake and flip the substack situated above the spatula.

Email address: `bruno.bouzy@parisdescartes.fr` (Bruno Bouzy)

He may repeat this action as many times as necessary. In the particular version, the goal of the server is to sort a particular stack with a minimum number of flips. In the global version, the question is to determine the maximum number of flips $g(N)$ [1] to sort any stack of N pancakes optimally. In the unburnt pancake problem, there is no burnt side, and the orientation of pancakes has no importance.

The pancake problem is a puzzle, or a one-player game well-known in artificial intelligence and in computer science under the name of sorting by prefix reversals (SBPR). Its importance is caused by its similarity with the sorting by reversals (SBR) problem which is fundamental in biology to understand the proximity between genomes of two different species. For example, the SBR distance between a cabbage and a turnip is three [1]. The SBR problem has been studied in depth [2] for the last twenty years. The unburnt pancake problem is NP-hard [3]. The complexity of the burnt pancake problem is not known except for specific sub-class of problems for which it is polynomial [4].

Our goal is to augment the state of the art on the burnt pancake problem with new findings. First, while previous values on $g(-I_N)$[2] were known for $N \leq 20$ [5], we provide exact values of $g(-I_N)$ for $N \leq 27$. This result is simply obtained with IDA* [6] and the number of breakpoints [7] as heuristic function. Consequently, while previous exact values on the diameter were known for $N \leq 17$ [5], and lower bounds on the diameter were known for $N \leq 20$, we provide lower bounds on the diameter for $N \leq 27$. Furthermore, we used a new heuristic function simply using the number of breakpoints and the number of anti-adjacencies. This heuristic is proven admissible on a test set for $N \leq 15$. It speeds up the searches by a factor greater than 2, compared to the number of breakpoints heuristic. With this new heuristic, we uncover some hard positions, different from $-I_N$ for $N \leq 22$. This shed some light on the knowledge of the diameter for $N \leq 22$.

Until now, the best theoretical R-approximation on the burnt pancake problem is 2 [8]. In the current work, we give experimental optimality ratio obtained with Monte-Carlo Search (MCS) [9] with nested levels using Cohen and Blum's algorithm as basic simulations. Under time constraints in which IDA* solves random positions for $N \leq 15$

[1]$g(N)$ denotes the diameter of the N burnt pancake problem, and not the cost of a partial plan in the A* meaning.

[2]$-I_N$ and $g(-I_N)$ are defined in section 2.

only, and obtains an experimental optimality ratio of 1.25, MCS solves random positions with a very high size, with an experimental optimality ratio of 1.30 for $N \leq 32$, between 1.30 and 1.50 for $N \leq 64$, between 1.50 and 1.60 for $N \leq 128$, between 1.60 and 2.0 for $N \leq 256$. These results extend the results concerning experimental optimality ratio in the unburnt pancake problem MCS [10].

The paper is organized as follows. Section 2 defines the burnt pancake problem and some useful notations. Section 3 relates work performed on the burnt pancake problem. Section 4 presents our work and its experimental results. Section 5 concludes.

2. Definitions

Let N be the size of a permutation π and

$$[\pi(1), \pi(2), ..., \pi(N-1), \pi(N)]$$

be the representation of π. The SBPR problem or pancake problem consists in reaching the identity permutation

$$I_N = [1, 2, ..., N-1, N]$$

by applying a sequence of prefix reversals. A prefix reversal, or a flip, $\rho(i)$ transforms the permutation

$$[\pi(1), ..., \pi(i), \pi(i+1), ..., \pi(N)]$$

into

$$[\underline{\pi(i), ..., \pi(1),} \pi(i+1), ..., \pi(N)].$$

Note that the top of the stack is on the left, and the bottom is on the right. Each number $\pi(i)$ corresponds to the size of the pancake situated at position i. The cut corresponds to the location of the spatula inserted between two pancakes.

In the burnt version, a sign is associated to each pancake (number). When performing a prefix reversal, the sign of the changing numbers changes too. After the reversal $\rho(i)$,

$$[\pi(1), ..., \pi(i), \pi(i+1), ..., \pi(N)]$$

becomes

$$\underline{-\pi(i), ..., -\pi(1),} \pi(i+1) ..., \pi(N).$$

In the literature, the permutations are often extended with two numbers, $N+1$ after $\pi(N)$, and 0 before $\pi(1)$, and the extended representation of permutation π is

$$[0, \pi(1), ..., \pi(N), N+1].$$

A basic and central concept in pancake problems is the breakpoint. In the unburnt pancake problem, a breakpoint is situated between i and $i-1$ when $|\pi(i) - \pi(i-1)| \neq 1$. In the burnt pancake problem, a breakpoint is situated between i and $i-1$ when $\pi(i) - \pi(i-1) \neq 1$. Note that in the pancake problems, the possible breakpoint between the top pancake and above is not taken into account.

In the following, $\#bp$ names the number of breakpoints. Since each breakpoint must be removed to obtain the identity permutation, and since one reversal removes at most one breakpoint, $\#bp$ is a lower bound of the length of optimal solutions [7]. In the planning context, $\#bp$ is a simple and admissible heuristic [11]. In the pancake problem literature, $g(N)$ names the diameter of the graph associated to the problems of size N. $g(p)$ names the length of an optimal solution of problem p, and not the cost of a partial plan in the A* meaning. $g(-I_N)$ is the length of an optimal solution of $-I_N$ which is a well-known stack in the burnt pancake literature. Known stacks which are hard to solve are [8], [5]:

$$-I_N = [-1, -2, ..., -(N-1), -N],$$

$$J_N = [\mathbf{+1}, -2, ..., -(N-1), -N],$$

$$Y_N = [-1, -2, ..., -(N-2), \mathbf{+(N\text{-}1)}, -N].$$

We define additional stacks to define the test sets $TestSet(N)$ used in our work. These stacks are generalizations of I_N, $-I_N$, J_N and Y_N. We call them $H_{N,M}$ where M is a non-negative integer with N bits. We define

$$H_{N,M} = [sgn(1, M) \times 1, ..., sgn(i, M) \times i, ..., sgn(N, M) \times N]$$

with

$$sgn(i, M) = \begin{cases} +1 & \text{if the ith most significant bit of M is 1} \\ -1 & \text{if the ith most significant bit of M is 0} \end{cases}$$

To illustrate this definition, we give some examples with $N = 6$.

$$H_{6,0} = [-1, -2, -3, -4, -5, -6] = -I_6,$$

$$H_{6,1} = [-1, -2, -3, -4, -5, \mathbf{+6}],$$

$$H_{6,2} = [-1, -2, -3, -4, \mathbf{+5}, -6] = Y_6,$$

$$H_{6,3} = [-1, -2, -3, -4, \mathbf{+5}, \mathbf{+6}],$$

$$H_{6,4} = [-1, -2, -3, \mathbf{+4}, -5, -6],$$

$$H_{6,5} = [-1, -2, -3, \mathbf{+4}, -5, \mathbf{+6}],$$

$$H_{6,32} = H_{6,2^5} = [\mathbf{+1}, -2, -3, -4, -5, -6] = J_6,$$

$$H_{6,33} = H_{6,2^5+1} = [\mathbf{+1}, -2, -3, -4, -5, \mathbf{+6}],$$

$$H_{6,34} = H_{6,2^5+2} = [\mathbf{+1}, -2, -3, -4, \mathbf{+5}, -6],$$

$$H_{6,63} = H_{6,2^6-1} = I_6.$$

3. Related work

First, this section presents the most important references of the SBR and SBPR problems. Secondly, it describes Cibulka's heuristic function, useful for our work. Thirdly, it presents Cohen and Blum's algorithm. Finally, it sums up the known values on the diameter $g(N)$ of the burnt pancake problem for stacks of size N.

3.1. Background

The best overview of the genome rearrangement problem is [2]. [1] devised the first polynomial-time algorithm for signed permutations. The unsigned permutation problem remains NP-hard [12]. The unburnt pancake problem is NP-hard [3]. The first bounds on the unburnt pancake problem diameter were found by [7]. The $(15/14)N$ lower bound was found by [13]. In 2009, a new upper bound was found on the diameter: $(18/11)N$ [14]. In 2010, in the planning context, the breakpoint heuristic $\#bp$ was explicitly used in a depth-first-search [11].

Concerning the burnt pancake problem, [8] presented the first bounds on the diameter: $3N/2$ is a lower bound and $2(N-1)$ a upper bound. Furthermore, they conjectured that the solution length of stack $-I_N$ equals the diameter. A polynomial-time algorithm on a specific sub-class of burnt pancake problems [4] was published in 2011. [5] showed that $7N/4$ flips were necessary to sort stacks of burnt pancakes on average over the stacks of size N. This work also disproved the conjecture by [8].

3.2. Cibulka's Heuristic Function

Let c be a stack of burnt pancakes, $-c$ denotes the stack with the reverse sign for every pancakes. [5] defines function $v(c)$ with equation (1).

$$\begin{aligned} v(c) &= a(c) - a(-c) + l(c) - l(-c) \\ &+ 1/3(o(c) - o(-c) + ll(c) - ll(-c) \\ &- (b(c) - b(-c))) \end{aligned} \tag{1}$$

$v(c)$ reflects the quality of stack c. The higher $v(c)$, the nearer c to the solution. We have equation (2).

$$v(I_N) = N + 2/3 \tag{2}$$

$a(c)$ is the number of adjacencies. An adjacency occurs between two neighbouring pancakes when their size difference is 1, and when they are oriented correctly relatively to each other (the burnt side of the smallest pancake faces the unburnt side of the largest pancake). $a(c)$ and $\#bp$ are linked by equation 3.

$$N = a(c) + \#bp. \tag{3}$$

$b(c)$ is the number of deep blocks. A block is a maximal subset of adjacent pancakes. A deep block does not contain the topmost pancake. $a(-c)$ is the number of anti-adjacencies. An anti-adjacency occurs between two neighbouring pancakes when their size difference is 1, and when the burnt side of the largest pancake faces the unburnt side of the smallest pancake. $b(-c)$ is the number of deep clans. A clan is a maximal subset of anti-adjacent pancakes. A deep clan does not contain the topmost pancake.

$o(c)$ is 1 if pancake -1 is not in a block on top or if pancake 1 is in a block, 0 otherwise.

$l(c)$ is 1 if N is lowest pancake, 0 otherwise.

$ll(c)$ is 1 if $l(c)$ is 1 and if $N - 1$ is the second lowest pancake.

$o(-c)$ is 1 if pancake 1 is not in block on top or if pancake -1 is in a block, 0 otherwise.

$l(-c)$ is 1 if $-N$ is lowest pancake, 0 otherwise.

$ll(-c)$ is 1 if $l(-c)$ is 1 and if $-(N - 1)$ is the second lowest pancake.

We do not use $b(c)$, $b(-c)$, $l(c)$, $ll(c)$, $l(-c)$, $ll(-c)$, $o(c)$ and $o(-c)$ in our work. [5] aims at obtaining a heuristic for the burnt pancake problem that is admissible, admissibility being proven mathematically. $c2$ being a stack obtained by performing a flip on stack c, Cibulka demonstrates that inequality (4) holds.

$$v(c) - v(c2) \leq 4/3 \tag{4}$$

Thus, he defines the admissible heuristic function h_C with equation (5).

$$h_C(c) = \left\lceil \frac{3}{4}(v(I_N) - v(c)) \right\rceil \tag{5}$$

However, h_C has a low quality in the most frequent stacks c of the pancake problem: the stacks with no adjacency and no anti-adjacency. For such stacks c, $h_C(c) = \frac{3}{4}\#bp < \#bp$ where $\#bp$ is admissible. h_C strictly underestimates the cost to go in many stacks. Furthermore, because $h_C(c)$ uses features such as $b(c)$ or $b(-c)$, $h_C(c)$ is slow to compute.

3.3. Cohen and Blum's algorithm

[8] designed an algorithm to sort a stack of N burnt pancakes in at most $2N$ moves. It distinguishes between Case 1 and Case 2. In Case 1, at least one pancake is rightside up. Let p be the largest such pancake. If $p = N$, then it is possible to move p down to the bottom position in two flips. Otherwise $p < N$ and $-(p+1)$ is upside down. It is possible to create an adjacency between p and $p+1$ in two moves. In Case 2, all pancakes are upside down. Then, there are two sub-cases: (2a) and (2b). In case (2a), at least one pancake $-p$ is strictly below $-(p+1)$. Two moves create the adjacency between them. In Case (2b), the stack is $-I_N$. In such case, [8] presents an algorithm that sorts $-I_N$ in at most $2N$ moves. Overall, any stack of burnt pancakes can be sorted in at most $2N$ moves.

3.4. Known values of $g(N)$

Table 1 yields the known values of $g(N)$ and $g(-I_N)$. [8] found the values of $g(N)$ for $n \leq 10$ and the values of $g(-I_N)$ for $N \leq 18$. [15] found $g(11)$ and $g(12)$. Cibulka found the values of $g(N)$ for $N \leq 17$ and the values of $g(-I_N)$ for $N \leq 20$. Furthermore, for any n such that $N \equiv 3$ (mod 4), Cibulka demonstrated equation 6 for $N \geq 15$.

Table 1: Known values of $g(N)$ and $g(-I_N)$.

N	$g(N)$		$g(-I_N)$	
2	4	CB95	4	CB95
3	6	CB95	6	CB95
4	8	CB95	8	CB95
5	10	CB95	10	CB95
6	12	CB95	12	CB95
7	14	CB95	14	CB95
8	15	CB95	15	CB95
9	17	CB95	17	CB95
10	18	CB95	18	CB95
11	19	K2008	19	CB95
12	21	K2008	21	CB95
13	22	C2011	22	CB95
14	23	C2011	23	CB95
15	25	C2011	**24**	CB95
16	26	C2011	26	CB95
17	28	C2011	28	CB95
18			29	CB95
19			30	C2011
20			32	C2011

Moreover, Cibulka disproved Cohen and Blum's conjecture by showing that $g(15) = 25$ and $g(-I_{15}) = 24$.

$$g(-I_N) = \lceil \frac{3N+3}{2} \rceil \tag{6}$$

3.5. Two complementary algorithms for solving stacks

In the planning community, there are two basic and domain-independent algorithms that we can use to solve burnt pancake stacks: IDA* [6] and Monte-Carlo Search (MCS) [9]. When IDA* uses an admissible heuristic and completes, the solution found is optimal.

At any time, IDA* yields a lower bound on the optimal length. However, N being the size of a stack, for N higher than a threshold, IDA* becomes useless actually. MCS is a simulation-based algorithm[3].

MCS is launched at a given level L and performs a level L simulation, the levels being nested. To perform a level $L+1$ simulation, at each point of the $L+1$ simulation, for any legal action, MCS performs the action, and launches a level L simulation. MCS backs up the result with the action. Then MCS chooses the action with the best result, and plays it within the level $L+1$ simulation. At any level, the best simulation played so far is memorized. The level 0 simulations of MCS are domain-dependent simulations that may contain more or less randomness. The higher the level of the simulation, the smarter the simulation quality.

MCS is recent but it has proved good results in several domains such as general game playing [17], expression discovery [18], morpion solitaire [19], weak Schur numbers [20], cooperative path-finding [21], and recently on the unburnt pancake problem [10]. In the following, IDA* is used with our heuristic function so as to compute values relevant to the diameter of the burnt pancake graph. MCS is used to obtain experimental optimality ratio.

4. Our work

Our contribution is twofold. The first contribution consists in new results (values of $g(-I_N)$ for N up to 27, and the resulting lower bounds on $g(N)$ for N up to 27), a new heuristic function, admissible for $N \leq 15$, enabling IDA* to find hard positions for $N \leq 22$. The second contribution consists in new experimental optimality ratio results by using MCS and Cohen and Blum's algorithm.

4.1. New results

In a first set of experiments, we use IDA* and the reference heuristic function $\#bp$. We use a desktop computer under Linux with one core Intel(R) Xeon(R) CPU X5690 running at 3.47GHz and few days of computations. We compute $g(-I_N)$ for N as high as

[3]MCS has a name that looks like MCTS (Monte-Carlo *Tree* Search), but MCS is different from MCTS [16].

possible. The second leftmost column of Table 2 gives the values of $g(-I_N)$ for $N \leq 27$ obtained in this simple setting. Because $\#bp$ is admissible, these values are exact. As another result, we have lower bounds of $g(N)$ for $N \leq 27$. These results are new, and improve Cibulka's results significantly. There are two explanations. First, while proven admissible, Cibulka's heuristic has a bad quality on all the random stacks with $\#bp = N$ and $a(-c) = 0$ which are frequent, due to the $\frac{3}{4}$ factor in equation 5. Secondly, Cibulka's heuristic uses many features whose computations slow down the search.

4.2. A new heuristic function

Following the work of Cibulka, we kept two features only: $a(c)$ and $a(-c)$. In addition, we define a wrong adjacency between two neighbouring and size-adjacent pancakes in the following case. The burnt sides of the two pancakes are stuck to each other, or the unburnt sides of the two pancakes are stuck to each other. $w(c)$ denotes the number of wrong adjacencies in a stack c. We seek for an admissible heuristic fonction $h_B(c)$ following equation (7)

$$h_B(c) = \#bp + \lambda a(-c) + \mu w(c) \tag{7}$$

λ (respectively μ) reflects the weight of the overhead brought about by anti-adjacencies (respectively wrong adjacencies) considered as breakpoints. $\#bp$ and $a(-c)$ are known state-of-the-art features. $w(c)$ is somewhat new but derived from $a(-c)$ and $\#bp$. Considering that the lower bound of the diameter of the burnt pancake problem is $\frac{3}{2}n$, we expect to find positive values of λ and μ such that h_B is admissible. We want to speed up the running time at most as possible. The computations of the features are computed incrementally in $O(1)$.

For each N, we define $TestSet(N)$, the test set of stacks of size N with their exact values obtained with $\lambda = 0$. $TestSet(N)$ contains positions such that $-I_N$, J_N, Y_N, $H_N,_M$ for specific values of M. The specific values of M are 0, 2, 4, 10, 20, 42, and so on, corresponding to stacks ordered in the unburnt version, but alternating positive pancakes and negative pancakes starting from the second pancake from the bottom. The specific values of M also correspond to stacks ordered in the unburnt version, but alternating positive pancakes and negative pancakes starting from from the top.

For instance with $N = 6$, we give some of the most useful stacks.

$$H_{6,0} = [-1, -2, -3, -4, -5, -6] = -I_6,$$

$$H_{6,2} = [-1, -2, -3, -4, \mathbf{+5}, -6] = Y_6,$$

$$H_{6,4} = [-1, -2, -3, \mathbf{+4}, -5, -6],$$

$$H_{6,10} = [-1, -2, \mathbf{+3}, -4, \mathbf{+5}, -6],$$

$$H_{6,20} = [-1, \mathbf{+2}, -3, \mathbf{+4}, -5, -6],$$

$$H_{6,42} = [\mathbf{+1}, -2, \mathbf{+3}, -4, \mathbf{+5}, -6],$$

$$H_{6,32} = [\mathbf{+1}, -2, -3, -4, -5, -6] = J_6,$$

$$H_{6,40} = [\mathbf{+1}, -2, \mathbf{+3}, -4, -5, -6].$$

The particularity of these stacks is to contain many anti-adjacencies and many wrong adjacencies. Consequently, they were very useful to maximize λ and μ.

In a first set of experiments, we set $\mu = 0$ and use $TestSet(15)$. On these stacks, we determined the maximal value of λ keeping h_B admissible on these stacks. Our best value was $\lambda = 0.44$. As shown by Table 2, $\lambda = 0.44$ enabled our program to find the exact values of $g(-I_N)$ for $N \leq 27$ again, and to uncover new estimates of $g(-I_N)$ for $N \leq 30$. These estimates are upper bounds only, since $h_B(\lambda = 0.44)$ is not proven admissible for $N = 28, 29, 30$. The speed up ratio between $\lambda = 0.44$ and $\lambda = 0$ is more than 2. For $\lambda > 0.44$, for some N, the values found were superior to the values found for $\lambda = 0$, showing that h_B is not admissible in such case.

In a second set of experiments, we determined the best values of μ given few values of λ. We found out that $\mu = 0.18$ was the best value for $\lambda \leq 0.33$. $\mu = 0.04$ was the best value for for $\lambda = 0.44$.

A recent trend in optimal planning is to build heuristic functions based on Linear Programming (LP) techniques [22]. Our technique looks like potential heuristics optimization [23] in that we maximize a linear combination of weights associated to features while respecting a set of inequalities corresponding to the admissibility of the heuristic function on a specific set of positions. However, the domain of burnt pancakes is specific, we have two weights and, at this point of our work, we did not really need LP to obtain λ and μ.

4.3. Seeking for hard positions

The classical way of computing the specific value of $g(N)$ for a specific N lies on seeking for hard stacks and on observation $O1$ [5].

4.3.1. Observation $O1$

Stack B of size $N-1$ being given, $2(N-1)$ stacks A of size N can be built starting with B. To build one of them, add a pancake at the bottom of B, either $+N$ or $-N$ (2 possibilities). Then, perform the N-flip that moves the added pancake to the top, and perform a flip to move the added pancake to any position ($N-1$ possibilities).

By applying observation $O1$, stack A is at most at distance $d(B)+2$ from I_N, where $d(B)$ is the distance between B and I_N. A search on stack A computes the actual distance $d(A)$ which can be $d(B)+2$ or $d(B)+1$.

4.3.2. The sets C_N^M

Let us call C_N^M the set of stacks of size N requiring M flips to be sorted [5]. If we know $C_{N-1}^{g(N-1)}$, the set of stacks at distance $g(N-1)$, then, by the mean of observation $O1$, we may build a set of candidate stacks that can reach $g(N-1)+2$. We perform the searches on this set. If at least one stack reaches $g(N-1)+2$, then $g(N)=g(N-1)+2$ and we are almost done. We may perform the searches on the remaining stacks of the set of candidates, and obtain $C_N^{g(N)}$. Otherwise, we have $g(N)=g(N-1)+1$. In this case, if we want to get $C_N^{g(N)}$, we have to build stacks and perform the searches starting with $C_{N-1}^{g(N-1)-1}$, the set of stacks at distance $g(N-1)-1$, which can take a long time, but is possible. This process can be iterated.

Since the classical process to compute $g(n)$ is heavy, we design a new light process that seeks for hard positions. Rather than using observation $O1$, we define observation $O2$.

4.3.3. Observation $O2$

Considering a stack s of size $N-1$, reverse the burnt side of one pancake in s (there are $N-1$ possibilities) or do nothing, and add pancake $+N$, or $-N$, at the bottom of s. This results in $2N$ stacks of size N.

4.3.4. Sets B_N

Our light process computes sets called B_N. B_N is the set of stacks of size N that are maximal according to the length of the solutions, and that are found by our light process below.

4.3.5. Light Process

Our light process starts with $B_2 = \{-I_2\}$. Then, iteratively, for $i > 2$, for each stack s in B_i, it builds all the candidate stacks sc by using observation $O2$; it searches the stack sc, and updates B_{i+1} according to the result of the search.

Let d_N be the distance between any $b \in B_N$ and I_N.

4.3.6. Results

Table 3 gives the set B_N of hard stacks. Since our heuristic is admissible for $N \leq 15$, we are sure that $d_N = g(N)$ for $N \leq 15$. Above the linecut ($N \leq 17$), $g(N)$ is known from the state of the art. We observe that $d_N = g(N)$ for $N \leq 17$. Therefore, for $N \geq 18$, we only have $d_N \geq g(N)$.

However, we highlight that, for $N \geq 18$, B_N contains the hardest stacks found by our light process, whatever their true solution lengths. T is the global time elapsed to complete the process until iteration N. T grows quickly. However, in a few days, the process completes iteration 21. We interrupted the process during iteration 22 after 25 days. The size of B_N remains small for $N \leq 22$, and can be kept in the memory of the computer. However, $|B_N|$ should grow quickly for $N > 22$. The disproof of Cohen and Blum's conjecture is observable by our process, since Y_{15} and J_{15} reach the maximal value 25, but not $-I_{15}$. For $N \leq 22$, $-I_N$ is in B_N, except for $N = 15$. The positions $-I_N$, J_N, Y_N, $H_{N,M}$, re-generate themselves from one iteration to the next one.

4.4. Experimental Optimality Ratio

The complexity of the burnt pancake problem is still unknown. When solving a problem is time exponential in the size of the problem, it is of interest to design time polynomial and approximate algorithms assessed with R-approximations as low as possible. The R-approximation of a polynomial-time algorithm A is the greatest ratio $R(p) = \frac{L_A(p)}{L_{opt}(p)}$ over all problems p. It corresponds to a worst-case analysis. $L_A(p)$

is the length of a solution output of A on p, and $L_{opt}(p)$ is the length of an optimal solution on p [24]. For the unburnt pancake problem, R-approximation algorithms are known. [25] describes a 2-approximation algorithm. Furthermore, this algorithm reaches a 1.22 “experimental R-approximation”, which corresponds to an average-case analysis. [10] describes a Monte-Carlo Search approach that reaches a 1.04 experimental R-approximation. In the following we call *Experimental Optimality Ratio* (EOR) these “experimental R-approximations”.

We call $EOR(p)$ the experimental optimality ratio of an approximate algorithm on a specific problem p. Since $L_{opt}(p)$ is generally unknown for large stacks, $L_{opt}(p)$ is replaced by the best admissible heuristic value of the literature: $\#bp$. We have $EOR(p) = \frac{L_A(p)}{\#bp}$. We generate several stacks at random (we do not use the hard stacks in B_N here). On this set of stacks, we compute the mean value EOR and the standard deviation. The standard deviation is roughly 0.05. The two-sigma rule says that the standard deviation over 100 problems is $0.05 \times 2/10 = 0.01$. The values of EOR given below are 0.01 correct with probability 0.95. On average, the time to solve one problem is at most one minute.

*4.4.1. IDA**

IDA* is not a polynomial-time algorithm. However, we give the EOR value for IDA* for a calibration purpose. The EOR values for IDA* are the references so as to compare the subsequent EOR values obtained later on. Because $\#bp$ is a strictly sub-optimal value on many stacks, we expect that $EOR > 1$ for IDA*. Table 4 shows the values of EOR in N when using IDA* as a solver. We observe that $EOR > 1.20$.

4.4.2. MCS+Greedy

Greedy performs a depth-one lookahead minimizing the heuristic function $\#bp$. It is used as level 0 simulations in the MCS framework. Table 5 shows the values of EOR in N. L_x is the average length of solutions at level x. T_x is the average time in seconds to sort one stack at level x. At level 0, EOR approximately equals 2. At level 1, EOR is greatly lowered but the maximal size of the stacks processed is decreased down to 64. At level 2, EOR is slightly lowered and the maximal size of the stacks processed is decreased down to 16.

4.4.3. MCS+EfficientCohenBlum

A better simulator is EfficientCohenBlum Cohen and Blum's algorithm CohenBlum enhanced with efficient moves [3]. Cohen and Blum's algorithm works by pair of moves. A simple enhancement consists in adding a first pass performing efficient moves while they exist. An efficient move is a move that creates an adjacency without destroying another one. Table 6 shows the values of EOR in N. At level 0, the improvement is slight only. At levels 1 and 2, EOR is significantly lowered. Levels 2 and 3 enable the solver to find $EOR = 1.3$ for sizes up to 32. Level 1 processes stacks of size 128 with $EOR = 1.6$.

5. Conclusion and Future Work

In this paper, we presented a work performed on the burnt pancake problem. We have three sets of results. First, we gave the exact values of $g(-I_N)$ for $N \leq 27$ (previous values were known for $N \leq 20$). Consequently, we obtained lower bounds on $g(N)$ for $N \leq 27$. Secondly, we designed a heuristic function based on adjacencies, anti-adjacencies and wrong adjacencies. This heuristic function is experimentally admissible for $N \leq 15$. Its admissibility is not proved otherwise. This heuristic function is fast to compute. We designed a light process seeking for hard positions for $N \leq 22$. With the light process and our fast heuristic function, we found out hard stacks for $N \leq 22$. Thirdly, we found new experimental R-approximation values, or experimental optimality ratio, by using a MCS approach associated with simulations such as Cohen and Blum's algorithm. Level 0 simulations give EOR in the range 1.8, 2.0 for sizes up to 256. Level 1 simulations give EOR in the range 1.4, 1.6 for sizes up to 128. Level 2 and level 3 simulations give EOR in the range 1.3, 1.4 for sizes up to 16 or 32. These experimental optimality ratio are complementary to the R-approximation values obtained previously on the unburnt pancake problem [10]. It is worth noting here, that we do not improve Cohen and Blum's R-approximation (2.0) which corresponds to the theoretical worst-case analysis. Instead, we provide a new experimental optimality ratio which corresponds to an experiental average-case analysis. This difference is similar to the difference existing between the Fischer and Ginzinger's R-approximation (2.0) and their experimental R-approximation (1.22) observed on average on unburnt pancakes.

We may pursue our research in several directions. A first direction is to apply our ideas on the light process computing the diameter to the unburnt pancake problem. A second one is to see whether other features could be efficiently added to our heuristic function on the burnt pancake problem while ensuring admissibility with LP techniques [22] such as potential heuristics [23]. A third one is more general and would consist in transfering part of our work from the pancake problem toward the genome rearrangement problem, which is another challenging problem.

References

[1] S. Hannenhalli, P. Pevzner, Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals, J. ACM 46 (1) (1995) 1–27.

[2] B. Hayes, Sorting out the genome, American Scientist 95 (2007) 386–391.

[3] L. Bulteau, G. Fertin, I. Rusu, Pancake flipping is hard, in: MFCS, Vol. 7564 of LNCS, 2012, pp. 247–258.

[4] A. Labarre, J. Cibulka, Polynomial-time sortable stacks of burnt pancakes, TCS 412 (2011) 695–702.

[5] J. Cibulka, Average number of flips in pancake sorting, TCS 412 (2011) 822–834.

[6] R. Korf, Depth-first iterative-deepening: An optimal admissible tree search, Artificial Intelligence 27 (1985) 97–109.

[7] W. Gates, C. Papadimitriou, Bounds for sorting by prefix reversal, Discrete Math. 27 (1979) 47–57.

[8] D. Cohen, M. Blum, On the problem of sorting burnt pancakes, DAM (1995) 105–120.

[9] T. Cazenave, Nested Monte-Carlo Search, in: IJCAI, 2009, pp. 456–461.

[10] B. Bouzy, An experimental investigation on the pancake problem, in: IJCAI Computer Game Workshop, 2015.

[11] M. Helmert, Landmark heuristics for the pancake problem, in: SoCS, 2010, pp. 109–110.

[12] A. Caprara, Sorting by reversals is difficult, ICCMB (1997) 75–83.

[13] M. Heydari, H. Sudborough, On the diameter of the pancake problem, Journal of Algorithms 25 (1997) 67–94.

[14] B. Chitturi, W. Fahle, Z. Meng, L. Morales, C. Shields, I. Sudborough, W. Voit, A (18/11)n upper bound for sorting by reversals, TCS 410 (2009) 3372–3390.

[15] R. Korf, Minimizing disk i/o in two-bit breadth-first search, in: D. Fox, C. Gomes (Eds.), AAAI 2008, AAAI press, 2008, pp. 317–324.

[16] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A Survey of Monte-Carlo Tree Search Methods, IEEE TCIAIG 4 (1) (2012) 1–43.

[17] J. Méhat, T. Cazenave, Combining UCT and nested Monte-Carlo search for single-player General Game Playing, IEEE Transactions on Computational Intelligence and AI in Games 2 (4) (2010) 271–277.

[18] T. Cazenave, Nested Monte-Carlo expression discovery, in: ECAI, Lisbon, 2010, pp. 1057–1058.

[19] C. D. Rosin, Nested Rollout Policy Adaptation for Monte Carlo-Tree Search, in: IJCAI, 2011, pp. 649–654.

[20] S. Eliahou, C. Fonlupt, J. Fromentin, V. Marion-Poty, D. Robilliard, F. Teytaud, Investigating Monte-Carlo methods on the weak Schur problem, in: M. Middendorf, C. Blum (Eds.), ECCO, Vol. 7832 of LNCS, 2013, pp. 191–201.

[21] B. Bouzy, Monte-Carlo Fork Search for Cooperative Path-Finding, in: T. Cazenave, M. Winands, H. Iida (Eds.), Workshop on Computer Games, no. 408 in CCIS, 2013, pp. 1–15.

[22] G. Röger, F. Pommerening, Linear Programming for Heuristics in Optimal Planning, in: AAAI workshop on Planning, Search and Optimization, 2015, pp. 69–76.

[23] J. Seipp, F. Pommerening, M. Helmert, New optimization Functions for Potential Heuristics, in: 25th ICAPS, 2015, pp. 193–201.

[24] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, , M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties, Springer, 1999.

[25] J. Fischer, S. Ginzinger, A 2-approximation algorithm for sorting by prefix reversals, in: ESA, Vol. 3669 of LNCS, 2005, pp. 415–425.

Table 2: Values of $g(-I_N)$ for $N \leq 30$ with running times for $\lambda = 0.44$ and $\lambda = 0$.

N	$g(-I_N)$	h_B	$T(\lambda = 0.44)$	$T(\lambda = 0)$
2	4	2	0.001s	0.002s
3	6	4	0.001s	0.002s
4	8	5	0.002s	0.004s
5	10	7	0.002s	0.004s
6	12	8	0.012s	0.03s
7	14	10	0.15s	0.3s
8	15	11	0.2s	0.5s
9	17	12	2.5s	5s
10	18	14	2.5s	5s
11	19	15	1.5s	5s
12	21	17	14s	1m
13	22	18	10s	40s
14	23	20	4s	15s
15	24	21	17s	30s
16	26	23	1m15s	5m
17	28	24	5m	15m
18	29	25	7m	20m
19	30	27	11m	1h
20	32	28	30m	30m
21	33	30	42m	1h30m
22	35	31	45m	1h30m
23	36	33	6h	14h
24	38	34	6h	15h
25	39	36	9h	28h
26	41	37	10h	20h
27	42	38	1d	2d
28	≤ 44	40	2d	
29	≤ 45	41	5d	
30	≤ 47	43	5d	

Table 3: Values of $g(N)$, d_N, $|B_N|$, T, and B_N.

N	$g(N)$	d_N	$\|B_N\|$	T	B_N
2	4	4	1	0	$-I_2$
3	6	6	2	0	$-I_3$ J_3
4	8	8	2	0	$-I_4$ J_4
5	10	10	2	0	$-I_5$ J_5
6	12	12	1	0	$-I_6$
7	14	14	1	0	$-I_7$
8	15	15	1	0	$-I_8$
9	17	17	1	30s	$-I_9$
10	18	18	1	1m	$-I_{10}$
11	19	19	3	2m	$-I_{11}$ Y_{11} J_{11}
12	21	21	1	4m	$-I_{12}$
13	22	22	3	6m	$-I_{13}$ Y_{13} J_{13}
14	23	23	4	30m	$-I_{14}$ Y_{14} $H_{14,4}$ J_{14}
15	25	25	2	1h	Y_{15} J_{15}
16	26	26	3	1h20	$-I_{16}$ $H_{16,4}$ J_{16}
17	28	28	1	3h	$-I_{17}$
18		29	4	5h	$-I_{18}$ Y_{18} $H_{18,4}$ J_{18}
19		30	9	16h	$-I_{19}$ Y_{19} J_{19} $H_{19,4}$ $H_{19,8}$ $H_{19,10}$ $H_{19,2^{18}+2}$ $H_{19,2^{18}+4}$ $H_{19,2^{18}+8}$
20		32	3	4d	$-I_{20}$ $H_{20,8}$ J_{20}
21		33	10	8d	$-I_{21}$ $H_{21,4}$ $H_{21,16}$ $H_{21,18}$ $H_{21,20}$ J_{21} $H_{21,2^{20}+2}$ $H_{21,2^{20}+4}$ $H_{21,2^{20}+16}$ Y_{21}
22		35	3	>25d	$-I_{22}$ Y_{22} J_{22}

Table 4: IDA*: EOR variations in N. L is the average length of solutions. T is the average time in seconds to sort one stack.

N	L	EOR	T
8	9.7	1.29	0
10	12.1	1.28	0
12	14.6	1.27	0.03
14	17.0	1.26	0.19
16	19.3	1.24	2.0
18	21.9	1.23	2.6
20	23.4	1.21	5.3

Table 5: MCS+Greedy: EOR variations in N and Level l. L_l are the average lengths. T_l are the average times in seconds.

N	L_0	R_0	T_0	L_1	R_1	T_1	L_2	R_2	T_2
8	13.8	1.83	0	10.5	1.39	0	10.1	1.34	0.01
16	30.5	1.96	0	21.4	1.37	0.01	20.0	1.28	0.58
32	64.1	2.04	0	44.1	1.40	0.14			
64	135	2.11	0.01	92.9	1.46	3			
128	263	2.05	0.02						
256	519	2.03	0.04						
512	1037	2.02	0.23						

Table 6: MCS+EfficientCohenBlum: EOR variations in N and Level l. L_l are the average lengths. T_i are the average times in seconds.

N	L_0	R_0	T_0	L_1	R_1	T_1
8	12.9	1.73	0	10.5	1.40	0
16	28.2	1.81	0	21.8	1.41	0
32	59.5	1.89	0	45.8	1.46	0.01
64	122	1.91	0	97.8	1.53	0.05
128	250	1.95	0	203	1.59	0.62
256	505	1.98	0.01			
N	L_2	R_2	T_2	L_3	R_3	T_3
8	10.0	1.33	0.01	10.0	1.33	0.02
16	20.3	1.31	0.03	19.9	1.28	2
32	40.7	1.29	1.2			