An Experimental Investigation on the Pancake Problem

Bruno Bouzy

LIPADE, Université Paris Descartes, FRANCE,
`bruno.bouzy@parisdescartes.fr`

Abstract. In this paper, we present an experimental investigation on the pancake problem. Also called sorting by prefix reversals (SBPR), this problem is linked to the genome rearrangement problem also called sorting by reversals (SBR). The pancake problem is a NP-hard problem. Until now, the best theoretical R-approximation was 2 with an algorithm which gives a 1.22 experimental R-approximation on stacks with a size inferior to 70. In the current work, we used a Monte-Carlo Search (MCS) approach with nested levels and specific domain-dependent simulations. First, in order to sort large stacks of pancakes, we show that MCS is a relevant alternative to Iterative Deepening Depth First Search (ID-DFS). Secondly, at a given level and with a given number of polynomial-time domain-dependent simulations, MCS is a polynomial-time algorithm as well. We observed that MCS at level 3 or 4 gives a 1.04 experimental approximation, which is a breakthrough. With a low nested level, our MCS solver solves stacks of size 512 with an experimental R-approximation value of 1.20.

1 Introduction

The pancake problem is described as follows. A chef prepares a stack of pancakes that come out all different sizes on a plate. The goal of the server is to order them with decreasing sizes, the largest pancake touching the plate, and the smallest pancake being at the top. The server can insert a spatula below a pancake and flip the substack situated above the spatula. 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 $f(n)$ - the diameter - to sort any stack of n pancakes.

This 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 [24]. The SBR problem has been studied in depth [25] for the last twenty years. The SBR problem can be signed when the signs of the genes are considered, or unsigned otherwise. Similarly, the pancakes can be burnt on one side, or not. This brings about four domains: one with unburnt pancakes, one with burnt pancakes, one with unsigned genes and one with signed genes.

In the unburnt pancake problem, Gates and Papadimitriou [23] gave the first bounds of the diameter in 1979, and Bulteau has shown that the problem is NP-hard in 2013

[12]. Very interesting work have been done between 1979 and today. The goal of the current work is to provide an experimental contribution to the unburnt pancake problem. More specifically, we show the pros and cons of two planning algorithms used in computer games: IDDFS [31] and MCS [14]. Besides, we define several domain-dependent algorithms: Efficient Sort (EffSort), Alternate Sort (AltSort), Basic Random EFficient algorithm (BREF), Fixed-Depth Efficient Sort (FDEffSort), and Fixed-Depth Alternate Sort (FDAltSort), and we re-use the Fischer and Ginzinger's algorithm (FG) [22]. FG was proved to be a 2-approximation algorithm that also reaches a 1.22 approximation experimentally. We show how to use the algorithms above in the MCS framework. We obtain an experimental approximation of 1.04, which is a significant reduction.

The paper is organized as follows. Section 2 defines the SBR and SBPR problems. Section 3 sums up the related work in the four domains. Section 4 presents our work and its experimental results. Section 5 concludes.

2 Definitions

Let N be the size of a permutation π and

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

the representation of π. The problem of sorting a permutation by reversals consists in reaching the identity permutation

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

by applying a sequence of reversals. A reversal $\rho(i, j)$ with $i < j$ is an action applied on a permutation. It transforms the permutation

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

into

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

The effect of a reversal is reversing the order of the numbers between the two cuts. A cut is located between two numbers of the permutation. In the above example, the first cut is between $i-1$ and i and the second one is between j and $j+1$.

In the pancake problem, each number $\pi(i)$ corresponds to the size of the pancake situated at position i in a stack of pancakes, and the permutation problem is seen as a pancake stack to be sorted by decreasing size. One cut is fixed and corresponds to the top of the stack. The other cut corresponds to the location of a spatula inserted between two pancakes so as to reverse the substack above the spatula. For example, the permutation

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

has its top on the left and its bottom on the right. After a flip $\rho(i)$ between i and $i+1$, the permutation becomes

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

In addition, a permutation can be signed or not. In the signed case, a sign is associated to each number, i.e. the integers of the permutation can be positive or negative. When performing a reversal, the sign of the changing numbers changes too. For example, after the reversal $\rho(i, j)$,

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

becomes

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

The burnt pancake problem is the signed version of the pancake problem. In the burnt pancake problem, the pancakes are burnt on one side, and a flip performs the reversal and changes the burnt side. The goal is to reach the sorted stack with all pancakes having their burnt side down.

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

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

The reversal distance of a permutation π is the length of the shortest sequence of reversals that sorts the permutation.

A basic and central concept in SBR problems is the breakpoint. For $1 \leq i \leq N+1$, a breakpoint is situated between i and $i-1$ when $|\pi(i) - \pi(i-1)| \neq 1$. In the following, $\#bp$ is 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 reversal distance. In the planning context, $\#bp$ is a simple and admissible heuristic. In the pancake problem, the possible breakpoint between the top pancake and above is not taken into account. In the signed permutation problem or in the burnt pancake problem a breakpoint is situated between i and $i-1$ when $\pi(i) - \pi(i-1) \neq 1$.

3 Related work

Since the four domains are closely linked, this section presents them in order: signed permutation, unsigned permutation, unburnt pancakes, and burnt pancakes.

3.1 Signed permutations

The best overview of the genome rearrangement problem to begin with is by Hayes [25]. Since the genes are signed, the genome rearrangement problem is mainly connected with the signed permutation problem, but also to the unsigned permutation problem. In 1993, Bafna and Pevzner [5] introduced the cycle graph. In 1995, Hannenhalli and Pevzner [24] devised the first polynomial-time algorithm for signed permutations. Its complexity was in $O(n^4)$. The authors introduced the breakpoint graph, and the so called hurdles. This work is the reference. The follow up consists in several refinements.

In 1996, Berman and Hannenhalli [7], and Kaplan, Shamir and Tarjan [28] in 1998, enhanced the result with an algorithm whose complexity was in $O(n^2)$. The concept

of fortress was new. In 2001, Bader and colleagues [4] found out an algorithm that finds the reversal distance in $O(n)$, but without giving the optimal reversal sequence. In 2002, GRIMM [39], a web site, was developed to implement the above theories. In 2003, [29] described efficient data structures to cope with the problem. Then, in 2005, Anne Bergeron [6] introduced a simple and self-contained theory, that does not use the complexities of the previous algorithms, and that solves signed permutation problems in quadratic time as well. In 2006, [38] and [37] are subquadratic improvements.

3.2 Unsigned permutations

The basic work in the unsigned permutation problem is [30] in 1992, by Kececioglu and Sanker. This problem was proved to be NP-hard [13] by Caprara in 1997. In 1998, Christie [17] described the 3/2-approximation algorithm. Reversal corresponding to red nodes are relevant. Furthermore, the Christie's thesis [18] described many approaches for other classes of permutation problems. In 1999 and 2001, [9] and [8] contain complexity results. In 2003, [3] and [1] describe evolutionary approaches to the unsigned permutation problem. Particularly, the work of Auyeung and Abraham [3], performed in 2003, consists in finding out the best signature of an unsigned problem, with a genetic algorithm. The best signature is the signature such that the signed permutation reversal distance is minimal. Computing this distance is performed in linear time [4].

3.3 Unburnt pancakes

The unburnt pancake problem is the most difficult of the four domains [12]. Related work focused on the diameter of the pancake graph. In 2004, a pancake challenge was set up. Tomas Rokicki won the challenge and gave explanations to solve and generate difficult pancake problems [35]. Fischer and Ginzinger published their 2-approximation algorithm [22].

Bounds on the diameter A focus is to bound the diameter of the graph of the problem in N the size of the pancake stack. The first bounds on the pancake problem diameter were found by Gates and Papadimitriou in 1979 [23]: $(5n+5)/3$ is the upper bound and $(17/16)n$ is the lower bound. To prove the upper bound, [23] exhibits an algorithm with several cases. They count the number of actions corresponding to each case and obtain inequalities. They formulate a linear program whose solution proves the upper bound. To prove the lower bound, they exhibit a length-8 elementary permutation that can be used to build length-n permutations with solutions of length $(18/16)n$ on average but bounded by below by $(17/16)n$. The Gates and Papadimitriou's sequence is

$$GP = [1, 7, 5, 3, 6, 4, 2, 8].$$

The $(15/14)n$ lower bound was found by Heydari and Sudborough in 1997 [27].

In 2006, the diameter of the 17-pancake graph [2] was computed. In 2009, a new upper bound was found on the diameter: $(18/11)n$ [16]. In 2010, in the planning context, the breakpoint heuristic $\#bp$ was explicitly used in a depth-first-search [26]. In 2011,

[19] Josef Cibulka showed that $17n/12$ flips were necessary to sort stacks of unburnt pancakes on average over the stacks of size n. Josef Cibulka mentioned a list of interesting concepts: deepness, surfaceness, biggest-well-placed, second-biggest-well-placed, smallest-not-on-top.

The 2004 pancake challenge In 2004, a pancake challenge was organized to focus on the resolution of specific problems. In a first stage, the entrants had to submit pancake problems. In the second stage, the entrants had to solve the submitted problems. The entry of the winner of the pancake challenge was the one of Tomas Rokicki [35]. Its entry is described and gives really interesting ideas.

The inverse problem and backward solutions Considering π^{-1} the inverse permutation of the original problem π can be helpful. For example, if

$$\pi = [5, 3, 6, 1, 4, 2]$$

then

$$\pi^{-1} = [4, 6, 2, 5, 1, 3].$$

π^{-1} and π correspond to two different problems. The one is the forward problem yielding a forward solution, and the other is the backward problem yielding a backward solution (BS). Because $\pi\pi^{-1} = Id$, the backward solution is the reverse sequence of the forward solution. The enhancement consists in solving the two problems simultaneously and comparing the lengths of the two solutions, and comparing the times to get them. We call it the BS enhancement. For IDDFS, the two solutions are optimal and share the same length, but the times to solve them can be very different. For MCS or approximate algorithms, the lengths of the two sequences can be different, and the idea consists in keeping the shortest solution. BS works in practice. See Table 3 compared to Table 2.

Difficult positions In the diameter estimation context, Tomas Rokicki exhibited two elementary sequences:

$$S5 = [1, 3, 5, 2, 4]$$

and

$$L9 = [1, 5, 8, 3, 6, 9, 4, 7, 2].$$

Then, he built L9-based (resp. S5-based) permutations by repeating the L9 (resp. S5) permutation shifted by 9 (resp. by 5). For example,

$$L9(2) = [1, 5, 8, 3, 6, 9, 4, 7, 2, 10, 14, 17, 12, 15, 18, 13, 16, 11].$$

These sequences $L9(x)$ and $S5(y)$ were used to attempt to prove a $11/10$ ratio lower bound on the diameter. Unfortunately, this approach did not work. However, these adhoc sequences are hard to solve, and we consider them as hard problems in the following.

The 2-approximation algorithm of Fischer and Ginzinger Fischer and Ginzinger designed FG, an algorithm that is a 2-approximation polynomial algorithm [22]. It means that the length L_{FG} of the solution found by FG is inferior to two times the length of the optimal solution. Since the best lower bound known today is the number of breakpoints $\#bp$, it means that L_{FG} is proved to be inferior to $2 \times \#bp$. In practice, Fischer and Ginzinger mention an approximation ratio $Rapprox = 1.22$. The idea is to classify moves in four types. Type 1 moves are the ones that remove a breakpoint in one move. Type 2 and type 3 moves lead a pancake to the top of the stack so as to move it to a correct place at the next move. Type 4 moves corresponds to the other cases. Fischer and Ginzinger proves that type 2, 3, 4 moves removes a breakpoint in less than 2 moves, and that type 1 moves remove a breakpoint in one move.

Pancake flipping is hard In 2012, Laurent Bulteau and his colleagues proved that the pancake flipping problem is NP-hard [12] [11]. He did this by exhibiting a polynomial algorithm that transforms a pancake problem into a SAT problem and vice-versa. He gave an important clue to solve the pancake problem. He considered sequence of type 1 moves only, i.e. moves removing one breakpoint. He defined efficiently sorted permutations, i.e. permutations that can be sorted by type 1 moves, or "efficient" moves. He defined deadlock permutations without type 1 move. A sequence of type 1 moves reaches either the identity permutation and the permutation is efficiently sorted, or deadlock permutations only, and the permutation is not efficiently sorted. To see whether a permutation is efficiently sortable, a binary tree must be developed. Bulteau made a polynomial correspondence between the efficiently sortable permutation problem and the SAT problem, proving by this translation that the former problem is NP-hard.

Miscellaneous [34] contains results about the genus of pancake network.

3.4 Burnt pancakes

Here again, the focus was to bound the diameter too. In 1995, the first bounds on the diameter and a conjecture [20] were presented: $3n/2$ is a lower bound and $2(n-1)$ a upper bound. The second bounds on the diameter were proved in 1997 [27]. A polynomial-time algorithm on "simple" burnt pancake problems [32] was published in 2011. In 2011, Josef Cibulka showed that $7n/4$ flips were necessary to sort stacks of burnt pancakes on average over the stacks of size n [19]. He also disproved the conjecture by Cohen and Blum [20]. Josef Cibulka mentioned interesting concepts: anti-adjacency and clan.

4 Our work

First, this section presents the domain-independent algorithms used in our work. Secondly, it presents the domain dependent algorithms designed in the purpose of our work. Thirdly, it presents the settings of the experiments. Then, this section yields the results of the experiments in order.

4.1 Domain Independent Algorithms

This section describes the algorithms we used to solve pancake problems as efficiently as possible. There are two basic and general algorithms:

– Iterative Deepening Depth-First Search (IDDFS) [31],
– Monte-Carlo Search (MCS) [14].

We consider IDDFS as an exponential-time algorithm in N [31]. When it completes, the solution found is optimal. However, for N superior to a threshold, IDDFS needs to much time, and becomes unuseful actually. However, before completion, IDDFS yields a lower bound on the optimal length.

MCS [14] is a simulation-based algorithm that gave very good results in various domains such as general game playing [33], expression discovery [15], morpion solitaire [36], weak Schur numbers [21] and cooperative path-finding [10]. MCS is used with a level L. At any time, MCS stores its best sequence found so far, thus it yields an upper bound on the optimal solution. When used at a given level L, MCS is a polynomial-time algorithm in N. Let assume that the level 0 simulations are polynomial-time algorithms. Let T_0 be the time used to perform a complete level 0 simulations. Let us bound T_1 the time to complete a level 1 MCS simulation. To move one step ahead in a level 1 simulation, MCS launches at most N level 0 simulations, which costs $N \times T_0$. Since the length of level 0 simulation is bounded by $2N$, we have $T_1 \leq 2N^2 T_0$. If T_0 is polynomial in N then T_1 is polynomial as well. By induction, a level L MCS is polynomial-time. The higher L, the higher the polynom degree.

The threshold effect observed for IDDFS does not appear for a polynomial-time algorithm. If you obtain solutions for N in time T, and if d is the degree of the polynom, you may obtain solutions for $N + 1$ in time $T \times (N + 1)^d / N^d$ which is just a little bit more expensive than T. Therefore, we get two tools to work with: one is costly but optimal when it completes, IDDFS, and the other one is approximate but its cost is polynomial-time, MCS.

4.2 Domain Dependent Algorithms

We have designed several pancake problem dependent algorithms: Efficient Sort (EffSort), Alternate Sort (AltSort), Basic Random EFficient algorithm (BREF), Fischer and Ginzinger algorithm (FG) [22], Fixed-Depth Efficient Sort (FDEffSort), Fixed-Depth Alternate Sort (FDAltSort). We have implemented each of them and we describe them briefly here.

Since a position has at most two efficient moves, EffSort searches within a binary tree to determine whether a permutation is efficiently sortable [12] or not. If the permutation is efficiently sortable, the sequence of efficient moves is output, and the output permutation is the identity. Otherwise, the longest sequence of efficient moves is output, and the output permutation is a deadlock (i.e. a permutation without efficient move).

When a position is a deadlock, the solver has to perform an inefficient move, i.e. a move that does not lower $\#bp$. A waste move is a move that keeps the $\#bp$ constant. We define two kinds or waste moves: hard or soft. A waste move is hard if it creates

a breakpoint while removing another one. A waste move is soft otherwise (the set of breakpoints is unchanged). We define a destroying move as a move that increases $\#bp$.

We designed AltSort. While the output permutation is not sorted, AltSort iteratively calls EffSort and performs a soft waste move if the output permutation is a deadlock. When EffSort is successful, AltSort stops. AltSort and EffSort are inferior to IDDFS. They are exponential time algorithms.

We designed BREF, that iteratively chooses and performs an efficient move if possible. A position may have 0, 1 or 2 efficient moves. On a position with one efficient move, this move is selected with probability 1. On a position with two efficient moves, one of them is selected with probability 0.5. Otherwise, BREF chooses and performs a soft waste move defined above. Most of the times, a position without efficient moves has a lot of soft waste moves available. In this case, the move is chosen at random with uniform probability. At the end of a simulation, the reward is the length of the simulation. BREF is a randomized version of AltSort. BREF and FG are polynomial-time algorithms. They can serve as level 0 simulation for MCS.

FDEffSort is the fixed-depth version of EffSort. With depth D, FDEffSort becomes a polynomial-time algorithm. FDAltSort is the version of AltSort using FDEffSort. FDAltSort is a polynomial-time algorithm. It can be used as a level 0 simulation for MCS.

4.3   Settings of experiments

The experiments show the effect of using:

– IDDFS or MCS,
– FG and BREF within MCS,
– backward solutions BS in addition to original solutions,
– FDAltSort within MCS.

There are different classes of test positions: positions randomly generated for a given size, and difficult positions mentioned by related work, mainly [23], [27], [35]. IDDFS may easily solve easy positions of size 60 randomly generated in a few seconds. However, IDDFS cannot solve some hard positions of size 30. Whatever the size and the problem difficulty, MCS always finds a preliminary solution quickly. This solution is refined as time goes on to become near-optimal or optimal.

We mention three indicators to evaluate our algorithms.

– Finding the minimal length $Lmin$ of an optimal solution for a given problem.
– Finding $Rapprox$ as low as possible with a polynomial time algorithm averaged over a set of 100 problems.
– Limiting the running time of an experiment with 100 problems to one or two hours.

$Rapprox$ is the ratio of the length L of the actual solution found over $Lmin$. Since $Lmin$ is unknown in practice, $Lmin$ is replaced by $\#bp$. $Rapprox = L/\#bp$. The standard deviation of the $Rapprox$ value that we observed for one problem generated at random 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 $Rapprox$ given below are 0.01 correct with probability 0.95. On average, the time to solve one problem is inferior to one minute.

Table 1: IDDFS: In practice, how $Rapprox$ varies in N. L is the average length of solutions. T is the average time in seconds to sort one stack.

N	L	$Rapprox$	T
8	6.5	1.09	0
16	14.5	1.05	0
32	31.0	1.03	0
64	63.0	1.02	5

Table 2: MCS+BREF: How $Rapprox$ varies in N and Level. $L(x)$ is the average length of solutions at level x. $T(x)$ is the average time in seconds to sort one stack at level x.

N	L(0)	R(0)	T(0)	L(1)	R(1)	T(1)	L(2)	R(2)	T(2)	L(3)	R(3)	T(3)	L(4)	R(4)	T(4)
8	7.5	1.15	0	7.5	1.19	0	7.0	1.15	0	6.7	1.12	0	6.5	1.08	0.01
16	18	1.30	0	18	1.27	0	16.5	1.18	0	16.0	1.14	0.01	15.5	1.09	0.05
32	38	1.30	0	37.5	1.26	0	37	1.22	0.02	36.5	1.21	0.6			
64	82	1.32	0	79.5	1.28	0.01	76	1.24	1.2						
128	165	1.30	0.01	163	1.29	0.23									
256	339	1.34	0.05	333	1.31	9									

4.4 MCS and IDDFS

The first experiment consists in assessing IDDFS and MCS under reasonable time constraints: at most one hour. MCS uses BREF as level 0 simulations. When using IDDFS, table 1 shows how $Rapprox$ varies in N. First, although IDDFS gives an optimal result, $Rapprox$ is not 1. This happens because $\#bp$ is not the length of optimal solutions but a lower bound only. Secondly, Table 1 shows that IDDFS cannot give results in reasonable time for $N > 64$. For $N = 30$, IDDFS does not solve some difficult positions in less than few hours. Thirdly, Table 2 shows how $Rapprox$ varies in N and in level with MCS. Level 0 simulations can be launched easily with $N = 256$. $Rapprox$ is 1.30 for level 0 simulations, 1.28 for level 1 simulations. Then, as the level increases, $Rapprox$ decreases. Level 2 simulations yields $Rapprox = 1.22$ which equals the value mentioned in [22]. For higher levels, $Rapprox = 1.10$ showing that high levels of MCS give good results in reasonable time, and that MCS is a viable alternative to IDDFS even for difficult positions. When compared to $Rapprox = 1.22$ of [22], $Rapprox = 1.08$ is a first breakthrough.

4.5 MCS + BREF + BS

Table 3 shows how $Rapprox$ varies in N and in the MCS level when level 0 simulations are the best of one forward simulation and one backward simulation. We call this the BS enhancement. Table 3 must be compared to Table 2. One can observe that the BS enhancement is effective at level 0 indeed, and also at level 1 and level 2. However, its effect is less visible at higher levels of MCS: $Rapprox = 1.09$.

Table 3: How $Rapprox$ varies in N and MCS Level with the trick of Backward Solutions (BS). $L(x)$ is the average length of solutions at level x. $T(x)$ is the average time in seconds to sort one stack at level x.

N	$L(0)$	$R(0)$	$T(0)$	$L(1)$	$R(1)$	$T(1)$	$L(2)$	$R(2)$	$T(2)$	$L(3)$	$R(3)$	$T(3)$	$L(4)$	$R(4)$	$T(4)$
8	7.0	1.15	0	6.8	1.14	0	6.7	1.12	0	6.7	1.11	0	6.6	1.10	0.01
16	16.5	1.18	0	16.4	1.18	0	15.6	1.12	0	15.4	1.10	0.01	15.3	1.09	0.1
32	37.0	1.22	0	36.4	1.21	0	35.4	1.17	0.05	35.2	1.15	0.8			
64	79.5	1.28	0.01	77	1.24	0.02	75.2	1.20	2						
128	163	1.29	0.02	159	1.26	0.5									
256	335	1.32	0.1												

Table 4: How $Rapprox$ varies in N and MCS level with simulations being FG with the trick of Backward Solutions (BS). $L(x)$ is the average length of solutions at level x. $T(x)$ is the average time in seconds to sort one stack at level x.

N	$L(0)$	$R(0)$	$T(0)$	$L(1)$	$R(1)$	$T(1)$	$L(2)$	$R(2)$	$T(2)$	$L(3)$	$R(3)$	$T(3)$	$L(4)$	$R(4)$	$T(4)$
8	7.73	1.28	0	6.85	1.14	0	6.7	1.10	0	6.7	1.10	0.01	6.7	1.10	0.01
16	18.2	1.30	0	16.0	1.14	0	15.1	1.07	0.01	14.9	1.05	0.06	14.8	1.05	0.6
32	39.7	1.29	0	35.2	1.16	0.01	32.6	1.08	0.12	31.6	1.05	4.2			
64	82.4	1.32	0.02	74.5	1.20	0.06	68.5	1.10	2.5						
128	167	1.33	0.04	155	1.23	0.8									
256	336	1.32	0.2												

4.6   MCS + FG + BS

Table 4 shows how $Rapprox$ varies in N and in the MCS level when level 0 simulations are the forward and the backward FG simulations. So as to see the effect of using FG instead of BREF in MCS, Table 4 must be compared to Table 3. First, the comparison shows that FG is worse than BREF for level 0. FG is on a par with BREF for level 1. For level 2 and higher levels, FG surpasses BREF: $Rapprox = 1.05$.

Table 5 shows how $Rapprox$ varies in N and in the MCS level when level 0 simulations are the forward and the backward *randomized* FG. Randomized FG works as follows. If type 1 moves exist, one of them is chosen at random. Otherwise, if type 2 moves exist, one of them is chosen at random and so on. So as to see the effect of using randomized FG instead of FG, Table 5 must be compared to Table 4. As expected and as shown by the $R(0)$ column of Table 5 and Table 4, randomized FG yields a worse $Rapprox$ than direct FG. Randomized FG is worse than direct FG for level 1. However, when used in higher levels of MCS, $Rapprox$ with the randomized version is slightly inferior to $Rapprox$ with the direct version. At level 4, MCS gives $Rapprox = 1.05$ when using randomized version of FG as basic simulations.

Table 5: How $Rapprox$ varies in N and MCS Level with simulations being randomized FG with the trick of Backward Solutions (BS). $L(x)$ is the average length of solutions at level x. $T(x)$ is the average time in seconds to sort one stack at level x.

N	L(0)	R(0)	T(0)	L(1)	R(1)	T(1)	L(2)	R(2)	T(2)	L(3)	R(3)	T(3)	L(4)	R(4)	T(4)
8	8.4	1.40	0	7.0	1.16	0	6.7	1.10	0	6.7	1.10	0.01	6.7	1.10	0.01
16	19.6	1.40	0	16.1	1.14	0	15.2	1.08	0.01	14.9	1.05	0.08	14.8	1.06	0.6
32	43.8	1.45	0	35.1	1.16	0.01	32.4	1.07	0.2	31.5	1.04	4			
64	90	1.43	0.02	76.5	1.23	0.1	67.5	1.08	6						
128	181	1.43	0.04	160	1.26	1.3									
256	356	1.40	0.3												

Table 6: How $Rapprox$ varies in N and MCS Level with simulations being FDAltSort at depth $D = 10$. $L(x)$ is the average length of solutions at level x. $T(x)$ is the average time in seconds to sort one stack at level x.

N	L(0)	R(0)	T(0)	L(1)	R(1)	T(1)	L(2)	R(2)	T(2)	L(3)	R(3)	T(3)
8	6.7	1.12	0	6.6	1.09	0	6.6	1.09	0	6.6	1.09	0
16	16.0	1.15	0	14.8	1.08	0	14.5	1.04	0.01	14.5	1.04	0.04
32	37.3	1.23	0	33.3	1.10	0.02	31.8	1.05	0.1	31.3	1.04	0.8
64	80.4	1.29	0	69.7	1.12	0.05	66.4	1.06	1	65.1	1.04	18
128	164	1.30	0.01	145	1.15	0.3	139	1.10	10.5			
256	326	1.28	0.02	300	1.18	1.5						
512	647	1.27	0.06	614	1.20	9						

4.7   MCS + FDAltSort

In a preliminary experiment, not reported here, we assessed MCS using AltSort directly as level 0 simulations. This did not work well on hard positions because AltSort is not a polynomial-time algorithm. Consequently some simulations did not complete quickly. We had to limit the depth at which EffSort searches and we had to create FDAltSort. (FDEffSort determines whether a permutation is efficiently sortable at depth D). FDAltSort can be used as a level 0 simulation in MCS. We assessed MCS using FDAltSort at depth $D = 10$. For each level, Table 6 displays the variations of $Rapprox$ in N. These results must be compared to the results of Table 5. At level 0, FDAltSort is better than randomized FG and on a par with FG. At level 1, level 2 and level 3, FDAltSort is better than the other algorithms. Launching FDAltSort at level 4 was not interesting. However, $Rapprox$ achieves 1.04 at its minimal value. Furthermore, the good point here is that the results are obtained for pancake stack sizes going up to 512 instead of 256 before, and with $Rapprox = 1.20$. This is a significant improvement. FDAltSort as simulations are much more efficient than FG or BREF were. We also tried to incorporate the BS enhancement, but the results were worse.

5 Conclusion and future work

In this work, we summed up the state-of-the-art of the permutation sorting by reversals domain. This domain was studied in depth by researches on the genome. It remains fascinating as underlined by Hayes [25]. The unsigned permutation domain is hard [13] but the signed permutation domain has polynomial-time solver [24]. The pancake problem was less studied. The unburnt pancake problem is difficult [12] while the complexity of the burnt pancake is unknown.

Our contribution is experimental. It shows how MCS extends the results obtained by IDDFS on unburnt pancake stacks. On the one hand, IDDFS can solve some pancake stacks of size 60 [26] in a few minutes but cannot solve some specific hard pancake stacks [35] of size 30 only. On the other hand, MCS can solve pancake stacks of significantly higher sizes and the hard pancake stacks of size 30 not solved by IDDFS. Practically, our MCS solver solves pancake stacks of size up to 512 with a 1.20 R-approximation, under the best configuration. MCS may use BREF, FG or FDAltSort with results that are approximately equal in terms of running time and $Rapprox$ value. Practically, we observed that MCS approximates the best solutions with a $Rapprox$ ratio of 1.04 for size up to 64, which is significantly better than the 1.22 value of [22]. Our MCS solver solves pancake stacks of size 128 or 256 with a R-approximation value roughly situated between 1.10 and 1.25.

In a near future, we want to study the burnt pancake problem. Furthermore, the burnt pancake problem is linked to the unburnt pancake problem. A block of sorted unburnt pancakes can be replaced by one burnt pancake, and the unburnt pancake problem becomes a mixed pancake problem. Solutions to burnt pancake problems could be used to solve ending unburnt pancake problems. We want to investigate in this direction.

To date, the number of breakpoints remains the simplest and the most efficient heuristic to bound the optimal solution length by below. However, this admissible heuristic should be refined to better approximate the optimal solution length. Some hard problems - or stacks - are hard because they contains substacks whose solution lengths are strictly higher than the number of breakpoints. As the permutation problems contain concepts such as hurdles or fortresses [7], and as shown by the work of Josef Cibulka on burnt pancakes, we have to find out the corresponding concepts to design appropriate heuristic functions for the pancake problems.

References

1. Jose Luis Alvarez and Mauricio Ayala-Rincon. A genetic approach with a simple fitness function for sorting unsigned permutations by reversals. 2003.
2. Shogo Asai, Yuusuke Kounoike, Yuji Shinano, and Keiichi Kaneko. Computing the diameter of 17-pancake graph using a PC cluster. In *Euro-par*, number 4128 in LNCS, pages 1114–1124, 2006.
3. Andy Auyeung and Ajith Abraham. Estimating genome reversal distance by genetic algorithm. 2003.
4. David Bader, Bernard Moret, and Mi Yan. A linear-time algorithm for computing inversion distance between signed permutation with an experimental study. In *WADS*, pages 365–376, 2001.

5. Vincent Bafna and Pavel Pevzner. Genome rearrangements and sorting by reversals. In *FoCS*, 1993.
6. Anne Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. *DAM*, 2005.
7. Piotr Berman and Sridhar Hannenhalli. Fast sorting by reversal. In *7th Symp. on Comb. Pattern Matching*, pages 168–185, 1996.
8. Piotr Berman, Sridhar Hannenhalli, and Marek Karpinski. 1.375-approximation algorithm for sorting by reversals. Technical Report 41, DIMACS, 2001.
9. Piotr Berman and Marek Karpinski. On some tighter inapproximability results. Technical Report 23, DIMACS, 1999.
10. Bruno Bouzy. Monte-Carlo Fork Search for Cooperative Path-Finding. In T. Cazenave, M. Winands, and H. Iida, editors, *Workshop on Computer Games*, number 408 in CCIS, pages 1–15, 2013.
11. Laurent Bulteau. *Algorithmic aspects of genome rearrangements*. PhD thesis, Université de Nantes, 2013.
12. Laurent Bulteau, Guillaume Fertin, and Irena Rusu. Pancake flipping is hard. In *MFCS*, volume 7564 of *LNCS*, pages 247–258, 2012.
13. Alberto Caprara. Sorting by reversals is difficult. *ICCMB*, pages 75–83, 1997.
14. Tristan Cazenave. Nested Monte-Carlo Search. In *IJCAI*, pages 456–461, 2009.
15. Tristan Cazenave. Nested Monte-Carlo expression discovery. In *ECAI*, pages 1057–1058, Lisbon, 2010.
16. B. Chitturi, W. Fahle, Z. Meng, L. Morales, C.O. Shields, I.H. Sudborough, and W. Voit. A (18/11)n upper bound for sorting by reversals. *TCS*, 410:3372–3390, 2009.
17. David Christie. A 3/2 approximation algorithm for sorting by reversals. In *9th SIAM symposium on Discrete Algorithms*, 1998.
18. David Christie. *Genome Rearrangement Problems*. PhD thesis, University of Glasgow, 1998.
19. Josef Cibulka. Average number of flips in pancake sorting. *TCS*, 412:822–834, 2011.
20. David Cohen and Manuel Blum. On the problem of sorting burnt pancakes. *DAM*, pages 105–120, 1995.
21. Shalom Eliahou, Cyril Fonlupt, Jean Fromentin, Virginie Marion-Poty, Denis Robilliard, and Fabien Teytaud. Investigating Monte-Carlo methods on the weak Schur problem. In M. Middendorf and C. Blum, editors, *ECCO*, volume 7832 of *LNCS*, pages 191–201. 2013.
22. Johannes Fischer and Simon Ginzinger. A 2-approximation algorithm for sorting by prefix reversals. In *ESA*, volume 3669 of *LNCS*, pages 415–425, 2005.
23. William Gates and Christos Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Math.*, 27:47–57, 1979.
24. Sridhar Hannenhalli and Pavel Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, 46(1):1–27, 1995.
25. Brian Hayes. Sorting out the genome. *American Scientist*, 95:386–391, 2007.
26. Malte Helmert. Landmark heuristics for the pancake problem. In *SoCS*, pages 109–110, 2010.
27. Mohammad Heydari and Hal Sudborough. On the diameter of the pancake problem. *Journal of Algorithms*, 25:67–94, 1997.
28. Haim Kaplan, Ron Shamir, and Robert Tarjan. Faster and simpler algorithm for sorting signed permutationd by reversals. 1998.
29. Haim Kaplan and Elad Verin. Efficient data structures and a new randomized approach for sorting signed permutations by reversals. In LNCS, editor, *CPM*, volume 2676, pages 170–185, 2003.
30. J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals with application to genome rearrangement. *Algorithmica*, 13:180–210, 1992.

31. Richard Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

32. Anthony Labarre and Josef Cibulka. Polynomial-time sortable stacks of burnt pancakes. *TCS*, 412:695–702, 2011.

33. Jean Méhat and Tristan 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):271–277, 2010.

34. Quan Nguyen and Said Bettayeb. On the genus of pancake nerwork. *IAJIT*, 8(3), 2011.

35. Tomas Rokicki. Pancake entry. http://tomas.rokicki.com/pancake/, 2004.

36. Christopher D. Rosin. Nested Rollout Policy Adaptation for Monte Carlo-Tree Search. In *IJCAI*, pages 649–654, 2011.

37. Eric Tannier, Anne Bergeron, and Marie-France Sagot. Advances on sorting by reversals. *DAM*, 2006.

38. Eric Tannier and Marie-France Sagot. Sorting by reversals in subquadratic time. In *SCPM*, 2004.

39. Glenn Tesler. GRIMM: Genome rearrangements web server. *Bioinformatics*, 18(3):492–493, 2002.