Average Case Optimal Pancake Sorting

Team CANEPKA

Coleman Link

Nik Olivieri
Agenda

- Brief review of pancake sorting, terminology, and our investigation
- High level overview of our solution
  - Inner vs. outer loops, algorithms, etc.
- Demo
- Future plans
Pancake Sorting - Background

- Given a list: [3, 1, 2, 4, 5]
- A “prefix reversal” of that list is taking some sequential portion of the list, that must start with the first element, and reversing its order
- E.g. [3, 1, 2, 4, 5] → [2, 1, 3, 4, 5]
  - Prefix reversal of length 3
- The terms “prefix reversal” and “flip” are used interchangeably
Pancake Sorting - Background

- Any list can be sorted using prefix reversals
- E.g. \([3, 1, 2, 4, 5] \rightarrow [2, 1, 3, 4, 5] \rightarrow [1, 2, 3, 4, 5]\)
- This is referred to as “Pancake Sorting”
  - The way you would sort a stack of pancakes by flipping some number of them with a spatula
Pancake Sorting - Background
Pancake Sorting - Background

- Each specific sequence has some minimum number of ‘flips’ (prefix reversals) that it can be sorted in
- “Pancake Number”
  - Largest number of flips that may be necessary to sort a list of a given size
  - Considered for permutations of sequential, unique elements (ie: numbers from 1 to N for a list size of N)
Our Investigation

- Compute an estimate for the average case number of prefix reversals to sort a list of a given length
  - Do this by generating many random lists of that length, computing the minimum number of reversals to sort each, then averaging this between all of the generated lists

- NP-Hard Sub-Problem: Find the minimum number of prefix reversals necessary to sort a given list
Our Implementation - Overview

- Fundamentally broken up into two parts:
  - “Inner Portion”: Computing the minimum number of prefix reversals to sort a given list
  - “Outer Portion”: Repeating inner portion on many random lists of a given length
Our Implementation - Overview

- “Inner Portion” (Parallel and Sequential):
  - Iterative Deepening A* using breakpoints as heuristic
  - Runs sequentially
  - Application of results from literature investigation
  - Initially implemented an A* only approach
    - Proved to be less efficient
Background: Breakpoints

- “Breakpoint”:
  - Location in pancake stack where two pancakes not adjacent in the sorted list come into contact
  - One also occurs when the bottom pancake is not the largest
  - Described in “Pancake Flipping is Hard” and comparable to H_gap in “Landmark Heuristics for the Pancake Problem”
Background: Breakpoints

- Example:
- [3, 1, 2, 5, 4]
  - Red characters show breakpoints
Background: A*

- Optimal “Path Finding” algorithm
- Start from some beginning state, adding its neighbors to a priority queue
- Queue is ordered by the sum of:
  - Distance from beginning state to a node
  - Heuristic estimation of distance from that node to the goal
Background: A*

- Continue removing nodes and adding their neighbors/predecessors until we pop goal state off queue
- Key attribute for optimality: “Admissible” heuristic
  - One that never overestimates the distance to the goal
  - Because the lowest valued states come off the queue first, this guarantees that a node is necessarily considered before any others with a higher actual value
Background: Iterative Deepening

- Idea: We have a search algorithm whose depth we can limit
  - Run the search algorithm with the depth limited to 1
  - Then with the depth limited to 2
  - Then 3, then 4, ...

- Useful when search has a large branching factor

- We apply this to a depth-limited A* in our implementation
Our Implementation - Sequential

- “Outer Portion”:
  - Generate a random list (by shuffling ordered list)
  - Run the “inner portion” on it
  - Repeat this for some number of iterations, keeping track of the total number of flips needed to sort the list
    - Use this to compute the average afterwards
Our Implementation - Parallel

- “Outer Portion”:
  - Do in parallel
    - Generate a random list (by shuffling ordered list)
    - Run the “inner portion” on it
    - Keep track of the total number of reversals in each thread
  - Reduce these together to get final result
Our Implementation - Rationale

- Using A*
  - NP Complete problem, so a heuristic approach is necessary for an exact solution
  - Some of the papers we read in our literature investigation reported getting positive results with it
Our Implementation - Rationale

- Parallelizing outer loop only
  - Some literature on parallel A* implementations exists, however:
    - It would add significant complexity to the implementation
    - For our purposes, having a decent number of trials is more important than the speed of any single trial
Our Implementation - Rationale

- Using Iterative deepening
  - Experimentally determined that it seems to scale better than simply using A* alone
Our Implementation - Demo
Future Plans

- Increasing speed
  - See if we can squeeze more efficiency out of A*
    - Can we model states without using objects?
    - Will this make a significant difference?
  - Can we replace JCF data structures with our own?
    - Will ours be noticeably faster than those in JCF?
Future Plans

- Performance Measurement
  - Ties in with both increasing speed and part 4 of project
  - Maybe look into some basic profiling (if time permits)
Sources (1/3)

- “Pancake Flipping Is Hard”
  - Laurent Bulteau, Guillaume Fertin, and Irena Rusu
  - 37th International Symposium on Mathematical Foundations of Computer Science
    - Pages 247-258 of proceedings
  - August 27-31, 2012
Sources (2/3)

- “Pancake Flipping with Two Spatulas”
  - Mahfuza Sharmin, Rukhsana Yeasmin, Masud Hasan, Atif Rahman, and M. Sohel Rahman
  - Electronic Notes in Discrete Mathematics
    - Volume 36, Pages 231-238
  - August 1, 2010
  - http://www.dcs.gla.ac.uk/~pat/cpM/jchoco/slabDesign/zzz.pdf
Sources (3/3)

- “Landmark Heuristics for the Pancake Problem”
  - Malte Helmert
  - AAAI Publications Third Annual Symposium on Combinatorial Search
  - August 25, 2010
Questions?