Computational Biology (MAT4930/MAT5932)

Lecture 5: Greedy Algorithms

Abdulmelik Mohammed

University of South Florida

01/26/2021

Recall the UsChange algorithm

Input: An amount of money M, in cents.
Output: The smallest number of quarter q, dimes d, nickels n and pennies p whose value adds to M.

1 $r \leftarrow M$
2 $q \leftarrow r/25$; #Integer division
3 $r \leftarrow r - 25 \cdot q$
4 $d \leftarrow r/10$
5 $r \leftarrow r - 10 \cdot d$
6 $n \leftarrow r/5$
7 $r \leftarrow r - 5 \cdot n$
8 $p \leftarrow r$
9 **return** (q, d, n, p)

Algorithm 1: UsChange algorithm.

- For $M = 77$, the algorithm returns $(3, 0, 0, 2)$.

Greedy algorithms

- The UsChange algorithm is a type of *greedy algorithm*.
- In the UsChange algorithm, we take whatever maximum number of possible maximum denomination we can take from the remaining amount.
- That is, instead of considering various possible combinations of denominations, the greedy algorithm only considers what is the most promising in the short term.
- For US denominations, the greedy approach works fine.

In other cases, greed leads to a non-optimal solution

Input: An amount of money M and an array d of denominations $\mathbf{c} = (c_1, \ldots, c_d)$ in decreasing order of value

Output: A list of d integers $i_1, \ldots, i_d$ such that $\sum_j^d c_j i_j = M$ and $\sum_j^d i_j$ is minimized.

```
1 r ← M
2 for k ← 1 to d do
3 |   i_k ← r/c_k
4 |   r ← r − c_k · i_k
5 end
6 return (i_1, ..., i_d)
```

Algorithm 2: BetterChange

- Consider the input $M = 40$ and $\mathbf{c} = (25, 20, 10, 5)$.

Characteristics of greedy algorithms

- Greedy algorithms often lead to suboptimal results to optimization problems.
- However, they are usually very fast.
- There are some greedy algorithms which guarantee optimal results (e.g. Kruskal's algorithm for finding minimum spanning trees).
- In this lecture, we see an application of greedy algorithm for a genome rearrangement problem.

Waardenburg's syndrome

Figure: Image by K. Marlise 2014.

- Leads to hearing loss and pigmentary abnormalities.

Genome rearrangements

- The gene implicated in Waardenburg's syndrome is in human chromosome 2.
- Breeding experiments in mice showed that a mutant named splotch had pigmentary abnormalities similar to those in humans with Waardenburg's syndrome.
- Gene mapping studies identified the chromosome in which the splotch gene is found in mice.
- Further gene mapping experiments showed that a group of genes in mice appear in the same order as they do in humans.
- More generally, the human genome contains about 300 large genomic fragments called *synteny blocks* from a mouse genome that have been rearranged and pasted together.

Rearrangement steps from synteny blocks in mouse X chromosome to those in human X chromosome

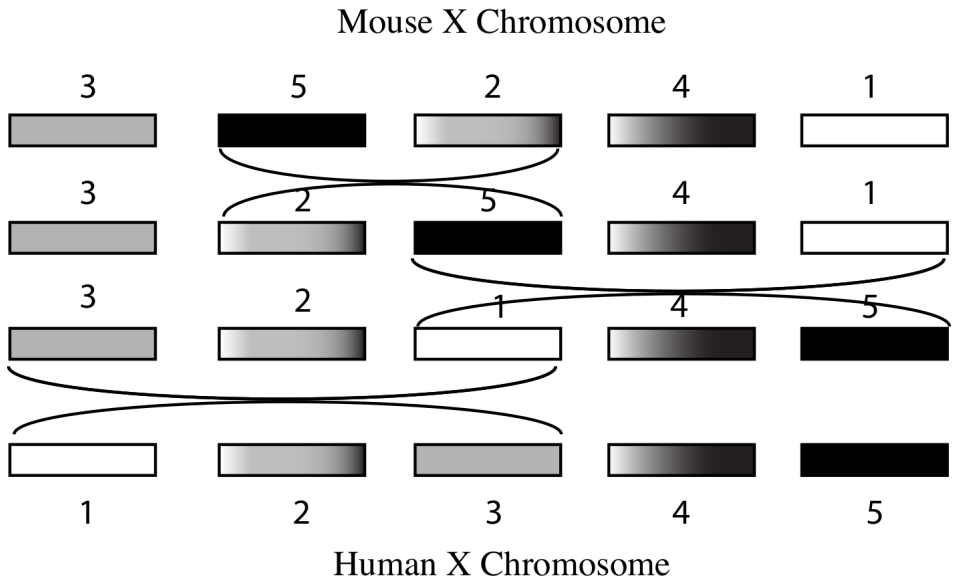

Figure: Image reprinted from Jones and Pevzner 2004.

- The elementary step here is the *reversal* or *inversion* of a series of consecutive synteny blocks.

Genome rearrangement and evolution

- Genome rearrangement studies attempt to find a series of transformations leading from one genome to another.
- The transformations could be of different types, but we focus on reversals.
- Rearrangement studies can give us clues about the evolutionary history.
- Biologist believe that the architecture of the X chromosome in the human-mouse ancestor is about the same as the architecture of the human X chromosome, and thus, for instance, by studying the shortest possible rearrangement from the mouse X chromosome to the human X chromosome, we can get a lower bound on the number of rearrangements that might have occured during evolution.

Computational modeling of genome rearrangements

- The order of synteny blocks can be represented by a *permutation*.
- A permutation of a set S is a bijective (one-to-one and onto) function from S to itself.
- For our genome rearrangement modeling, we use permutations of $[n] = \{1, 2, \ldots, n\}$.
- The permutation $\pi : [3] \rightarrow [3]$ with $\pi(1) = 1$, $\pi(2) = 3$ and $\pi(3) = 2$ is one such example.
- We use $(\pi_1, \ldots, \pi_n)$ to represent the permutation $\pi : [n] \rightarrow [n]$, where $\pi_i = \pi(i)$.

Computational modeling of genome rearrangements

- We can use $(1, 2, \ldots, n)$ to represent the order of synteny blocks in the one chromosome, such as $(1, 2, 3, 4, 5)$ for those of the human X chromosome.
- The order of the synteny blocks in the mouse X chromosome can then be represented by the permutation $(3, 5, 4, 2, 1)$.
- A reversal of consecutive synteny blocks from i to j is modeled by a permutation $\rho(i, j)$ that transforms a permutation

$$\pi = (\pi_1, \ldots, \pi_{i-1}, \underrightarrow{\pi_i, \ldots, \pi_j}, \pi_{j+1} \ldots \pi_n)$$

to a permutation

$$\pi \cdot \rho(i, j) = (\pi_1, \ldots, \pi_{i-1}, \underleftarrow{\pi_j, \ldots, \pi_i}, \pi_{j+1}, \ldots, \pi_n).$$

Reversal Distance Problem

- We can then model the shortest rearrangement steps from one permutation of synteny blocks to another using the Reversal Distance Problem.

Input: Permutations π and σ.

Output: The shortest series of reversals $\rho_1, \rho_2, \ldots, \rho_t$ such that $\pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_t = \sigma$.

Problem 3: Reversal Distance Problem.

- We call $d(\pi, \sigma) := t$ the *reversal distance* between π and σ.

Sorting By Reversals

- We can set the second genome order σ as $(1, 2, \ldots, n)$. This yields the Sorting By Reversals problem.

Input: Permutation π.

Output: The shortest series of reversals $\rho_1, \rho_2, \ldots, \rho_t$ such that $\pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_t = (1, 2, \ldots, n)$.

Problem 4: Sorting By Reversals.

- We call $d(\pi) := t$ the *reversal distance* of π.

A greedy approach to sorting by reversals

- Consider the example $\pi = (1, 2, 3, 6, 4, 5)$.
- At first sight, or being greedy, it would appear that there is no point in reversing a prefix of a permutation if that prefix is already sorted. Thus, one greedy approach is to fix these elements, i.e. elements 1,2,3 in π.
- We can thus proceed by trying to maximize the prefix which have been sorted. In our example, this would entail reversing $(6, 4)$ to get $(1, 2, 3, 4, 6, 5)$.
- Continuing in the same manner, the greedy approach to reversal yields:

$$(1, 2, 3, 6, 4, 5) \rightarrow (1, 2, 3, 4, 6, 5) \rightarrow (1, 2, 3, 4, 5, 6).$$

- The general idea is thus to reverse from the first out of order element up to the location of that element.

Greedy reversal sort

Input: Permutation π.
Output: The shortest series of reversals $\rho_1, \rho_2, \ldots, \rho_t$ such that $\pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_t = (1, 2, \ldots, n)$.

```
1  SimpleReversalSort(π)
2      for i ← 1 to n − 1 do
3          j ← position of element i in π
4          if j ≠ i then
5              π ← π · ρ(i, j)
6              Output π
7          end
8          if π is the identity permutation then
9              return
10         end
11     end
```

Algorithm 5: SimpleReversalSort

Analysis

- The greedy strategy does not necessarily lead to optimal solutions.
- For instance, for input $\pi = (6, 1, 2, 3, 4, 5)$, the algorithm outputs:

$$(\underline{6, 1}, 2, 3, 4, 5) \to (1, \underline{6, 2}, 3, 4, 5) \to (1, 2, \underline{6, 3}, 4, 5) \to (1, 2, 3, \underline{6, 4}, 5) \to (1, 2, 3, 4, \underline{6, 5}) \to (1, 2, 3, 4, 5, 6).$$

- However, $d(\pi) = 2$ as $(\underline{6, 1, 2, 3, 4, 5}) \to (\underline{5, 4, 3, 2, 1}, 6) \to (1, 2, 3, 4, 5, 6)$.

A related problem: pancake flipping

Figure: Reprinted from Jones and Pevzner, 2004.

- In the Pancake Flipping Problem, we are restricted to reversals of prefixes, i.e. reversals of the form $\rho(1, i)$.
- We can sort n pancakes using at most $2(n-1)$ flips. For example:

$$(\underline{1,2,3,6},4,5) \to (\underline{6,3,2,1,4,5}) \to (\underline{5,4,1,2,3},6) \to (\underline{3,2,1},4,5,6) \to (1,2,3,4,5,6).$$

Approximation algorithms

- Pancake Flipping is a computationally hard problem, i.e. it is NP-Hard [1].
- For such problems, we often look for algorithms which, while not giving the optimum, will guarantee that the output is not too far from the optimum.
- An *approximation algorithm* to an optimization problem is an algorithm which outputs a solution whose value is close to the optimum.
- An *approximation ratio of an algorithm $\mathcal{A}$ on an input π* is defined as $\frac{\mathcal{A}(\pi)}{OPT(\pi)}$, where $\mathcal{A}(\pi)$ is the output of $\mathcal{A}$ on π and $OPT(\pi)$ is the optimal solution to π.

Approximation guarantees

- The *approximation ratio* of a minimization problem is a function of the input size n defined as:
$$\max_{|\pi|=n} \frac{\mathcal{A}(\pi)}{OPT(\pi)}.$$
- That is, the approximation ratio at n is the worst possible ratio between the algorithm's output and the optimum output, among all the possible inputs of size n.
- For a maximization problem, the approximation ratio is:
$$\min_{|\pi|=n} \frac{\mathcal{A}(\pi)}{OPT(\pi)}.$$
- The closer the approximation ratio is to 1, the better an approximation algorithm is.

Approximation ratio of SimpleReversalSort

- Consider a generalization $(n, 1, 2, \ldots, n-1)$ of the bad input $(6, 1, 2, 3, 4, 5)$ for SimpleReversalSort.
- SimpleReversalSort takes $n-1$ reversals to reach the identity permutation from $(n, 1, 2, \ldots, n-1)$.
- However, the optimum number of reversals is 2.
- Hence, the approximation ratio is at least $\frac{n-1}{2}$. This is very poor, as ratio approaches infinity as n gets large.
- The best known approximation algorithm for sorting by reversals has an approximation ratio of 1.375 [2].

References

Laurent Bulteau, Guillaume Fertin, Irena Rusu, Pancake Flipping is hard, *Journal of Computer and System Sciences* 81:1556–1574, 2015.

Piotr Berman, Sridhar Hannenhalli, Marek Karpinski, 1.375-Approximation Algorithm for Sorting by Reversals, *European Symposium on Algorithms*, *volume 2461 of LNCS* pp. 200–210, 2002.