Sorting a list of numbers with modified cost

First, this was one of the four problems we had to solve in a project last year and I couldn't find a suitable algorithm so we handle in a brute force solution.

Problem: The numbers are in a list that is not sorted and supports only one type of operation. The operation is defined as follows:

Given a position i and a position j the operation moves the number at position i to position j without altering the relative order of the other numbers. If i > j, the positions of the numbers between positions j and i - 1 increment by 1, otherwise if i < j the positions of the numbers between positions i+1 and j decreases by 1. This operation requires i steps to find a number to move and j steps to locate the position to which you want to move it. Then the number of steps required to move a number of position i to position j is i+j.

We need to design an algorithm that given a list of numbers, determine the optimal (in terms of cost) sequence of moves to rearrange the sequence.

Attempts: Part of our investigation was around NP-Completeness, we make it a decision problem and try to find a suitable transformation to any of the problems listed in Garey and Johnson's book: Computers and Intractability with no results. There is also no direct reference (from our point of view) to this kind of variation in Donald E. Knuth's book: The art of Computer Programing Vol. 3 Sorting and Searching. We also analyzed algorithms to sort linked lists but none of them gives a good idea to find de optimal sequence of movements.

Note that the idea is not to find an algorithm that orders the sequence, but one to tell me the optimal sequence of movements in terms of cost that organizes the sequence, you can make a copy and sort it to analyze the final position of the elements if you want, in fact we may assume that the list contains the numbers from 1 to n, so we know where we want to put each number, we are just concerned with minimizing the total cost of the steps.

We tested several greedy approaches but all of them failed, divide and conquer sorting algorithms can't be used because they swap with no cost portions of the list and our dynamic programing approaches had to consider many cases.

The brute force recursive algorithm takes all the possible combinations of movements from i to j and then again all the possible moments of the rest of the element's, at the end it returns the sequence with less total cost that sorted the list, as you can imagine the cost of this algorithm is brutal and makes it impracticable for more than 8 elements.

Our observations:

- n movements is not necessarily cheaper than n+1 movements (unlike swaps in arrays that are O(1)).
- There are basically two ways of moving one element from position i to j: one is to move it directly and the other is to move other elements around i in a way that it reaches the position j.
- At most you make n-1 movements (the untouched element reaches its position alone).
- If it is the optimal sequence of movements then you didn't move the same element twice.

algorithm sorting optimization

edited Sep 26 '12 at 0:19
Bill the Lizard ♦
161k 107 390 671

asked Jan 14 '11 at 23:53
David
60 5

Would you mind giving a small example of the "operation"? I don't completely understand it the way it's described. – orlp Jan 15 '11 at 0:41

@nightcracker: To me it looks like a simple `insert(remove(i), j)` , so that the position of that element will change from i to j, with the cost of `abs(i-j)` – ruslik Jan 15 '11 at 0:43

It would help if you posted an example, or the (pseudo) code for the best solution so far. – Apalala Jan 15 '11 at 5:03

Input: 4,1,2,3 output: pos 0 to 3 cost: 3 – David Jan 15 '11 at 6:31

Input: 1,3,4,2 output: pos 3 to 1 cost: 5 – David Jan 15 '11 at 6:34

show **1** more comment

1 Answer

This problem looks like a good candidate for an approximation algorithm but that would only give us a good enough answer. Since you want the optimal answer, this is what I'd do to improve on the brute force approach.

Instead of blindly trying every permutations, I'd use a backtracking approach that would maintain the best solution found and prune any branches that exceed the cost of our best solution. I would also add a transposition table to avoid redoing searches on states that were reached by previous branches using different move permutations.

I would also add a few heuristics to explore moves that are more likely to reach good results before any other moves. For example, prefer moves that have a small cost first. I'd need to experiment before I can tell which heuristics would work best if any.

I would also try to find the longest increasing subsequence of numbers in the original array. This will give us a sequence of numbers that don't need to be moved which should considerably cut the number of branches we need to explore. This also greatly speeds up searches on list that are almost sorted.

I'd expect these improvements to be able to handle lists that are far greater than 8 but when dealing with large lists of random numbers, I'd prefer an approximation algorithm.

By popular demand (1 person), this is what I'd do to solve this with a genetic algorithm (the meta-heuristique I'm most familiar with).

First, I'd start by calculating the longest increasing subsequence of numbers (see above). Every item that is not part of that set has to be moved. All we need to know now is in what order.

The genomes used as input for the genetic algorithm, is simply an array where each element represents an item to be moved. The order in which the items show up in the array represent the order in which they have to be moved. The fitness function would be the cost calculation described in the original question.

We now have all the elements needed to plug the problem in a standard genetic algorithm. The rest is just tweaking. Lots and lots of tweaking.

edited Jan 31 '11 at 7:11

answered Jan 15 '11 at 8:58

Ze Blob
1,517 7 10

Sometimes an approximation algorithm is a good option too. If you think something can work here, it would be great if you add it ;) – Oscar Mederos Jan 31 '11 at 1:01

@Oscar I added a genetic algorithm solution. I just used what I knew so there might be something else that converges much faster for this problem (see all 3 billion algorithms in the meta-heuristique wikipedia article). – Ze Blob Jan 31 '11 at 7:17

add a comment

Not the answer you're looking for? Browse other questions tagged algorithm | sorting | optimization **or ask your own question.**