Speeding up the optimal off-line algorithm for solving the $k$-server problem

Tomislav Rudec · Alfonzo Baumgartner · Robert Manger

1 Introduction

In the $k$-server problem [6] one has to decide how $k$ mobile servers should serve a sequence of requests. Each server occupies a location in a fixed, possibly infinite, metric space. Repeatedly, a new request appears at some location $x$. To serve the request, a corresponding algorithm must move a server to $x$ unless it already has a server at that location. Whenever the algorithm moves a server from a location $x$ to a location $y$, it incurs a cost equal to the distance between $x$ and $y$. The goal of good serving is not only to serve requests, but also to minimize the total distance moved by all servers.

It is usually required that the solution to the $k$-server problem is produced in on-line fashion [4], so that each request is served before the next request arrives. However, for various reasons, it is also useful to consider off-line solutions. Contrary to any on-line procedure, whose serving is never quite satisfactory due to lack of information on future requests, an off-line procedure knows the whole input in advance and can therefore provide optimal serving.

2 Aim of the paper

In this paper we are concerned with the optimal off-line algorithm for solving the $k$-server problem - let us call it OPT. It is well known [2] that OPT can be implemented quite efficiently by network flow techniques. The aim of this paper is to propose certain modifications to the original network flow implementation of OPT. Our modifications improve the original version in terms of speed. The aim of the paper is also to evaluate experimentally the obtained improvements.

Tomislav Rudec
Faculty of Electrical Engineering, University of Osijek, Trpimirova 2b, 31000 Osijek, Croatia
E-mail: Tomislav.Rudec@etfos.hr

Alfonzo Baumgartner
Faculty of Electrical Engineering, University of Osijek, Trpimirova 2b, 31000 Osijek, Croatia
E-mail: Alfonzo.Baumgartner@etfos.hr

Robert Manger
Department of Mathematics, University of Zagreb, Bijenička 30, 10000 Zagreb, Croatia
E-mail: Robert.Manger@math.hr
3 Original implementation

As it has been described in [2], OPT finds the optimal way of serving a given sequence of \( n \) requests by \( k \) servers with given initial positions. The solution is obtained by computing the minimal-cost maximal flow on a suitably constructed network with \( 2n + k + 2 \) nodes. Actual computation of the optimal flow is accomplished by the flow augmentation method [1], which reduces to exactly \( k \) steps. Each of those steps reduces to a single-source shortest path problem in a so-called displacement network consisting again of \( 2n + k + 2 \) nodes. With some preprocessing [3], any shortest path problem can be solved by Dijkstra’s procedure [5].

4 Modified implementation

Our modified implementation of OPT is obtained from the original implementation by modifying each of its steps separately. All modifications are based on special properties of the involved networks. Here are some details.

– In each step arcs or nodes are identified which cannot influence the remaining part of the algorithm. Such arcs or nodes are removed from the network.
– In the first step it is observed that the sought shortest path is almost completely determined in advance, only one arc has yet to be chosen. Consequently, the shortest path is constructed directly, without any general path-finding procedure, by considering \( k \) possibilities and finding the minimum among certain \( k \) values.
– In the second step it is observed that the displacement network is acyclic. Thus the shortest path is found by simple scanning of nodes in topological order, which is several times faster than with Dijkstra.
– In the third, fourth, or any of the remaining steps the shortest path is found by a slightly customized version of Dijkstra. Moreover, if it happens that the shortest path has length 0, then the whole algorithm is immediately stopped since the current solution cannot change anymore.

Our modifications produce substantial improvement of the first two steps of OPT. The improvement of the remaining steps is not so dramatic. Still, thanks to constant removal of arcs and nodes, the network in each step becomes simpler, thus enabling faster execution.

5 Experimental results

Our experiments have been designed to measure the overall effect of all modifications applied to OPT. We have tested both versions of OPT on the same set of \( k \)-server problem instances, with the request sequence length \( n \) ranging from 1000 to 3000, and the number of servers \( k \) ranging from 2 to 20. Some results of our experiments are presented in Table 1, where each entry corresponds to one particular problem instance with particular values of \( n \) and \( k \). According to the table, the modified version of OPT turns out to be considerably faster than the original version, achieving the speedup between 5 and 13. The experiments have also shown that for a fixed \( n \) the speedup becomes larger as \( k \) becomes smaller.
Table 1 Speedup of the modified OPT vs. the original OPT

<table>
<thead>
<tr>
<th>k</th>
<th>n = 1000</th>
<th>n = 1500</th>
<th>n = 2000</th>
<th>n = 2500</th>
<th>n = 3000</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10.097</td>
<td>9.026</td>
<td>8.865</td>
<td>7.744</td>
<td>7.563</td>
</tr>
<tr>
<td>5</td>
<td>8.565</td>
<td>7.742</td>
<td>7.179</td>
<td>6.795</td>
<td>6.864</td>
</tr>
</tbody>
</table>

6 Conclusion

The optimal off-line algorithm for the k server problem is important, because it provides benchmarks for on-line computation, and because it serves as a building block for certain complex on-line algorithms that rely on solving auxiliary off-line subproblems. In this paper we have described how the conventional network-flow implementation of the optimal off-line algorithm can be improved in terms of speed. According to our experimental results, the improved version of the algorithm is indeed considerably faster than the original version, specially if the number of servers k is relatively small.

References