Analysis of the Server Problem on the Cycle

R&D Project Final Report

by

Aamod Kore
110050004

Guide: **Prof. Sundar Vishwanathan**

Department of Computer Science and Engineering
Indian Institute of Technology – Bombay
Mumbai 400076, India

May 5, 2014

Contents

Abstract

The k-server problem, is the problem to move k servers through a metric space serving a sequence of requests, making decisions in an online manner. It is perhaps the most influential online problem. It is a generalization of the paging problem and the weighted cache problem, which have widespread practical applications. The k-server conjecture, posed over two and a half decades ago, is still a open problem and is a major driving force in developing online algorithms.

We study a specific case of the server problem – the 3-server problem on the cycle. We analyse the performance of online algorithms, specifically the work function algorithm, on the problem as compared to the optimal solution. For this we implement both the work function algorithm and the (offline) algorithm that gives the optimal solution. We develop different strategies for generating request sequences, in patterns that would provide insight on the general k-server conjecture. We also take a look at other parameters of the algorithm and try to formulate results based on the same.

Chapter 1

Introduction and Overview

1.1 The k-server problem

The k-server problem can be stated informally as the problem of moving around k servers, in a metric space or a weighted graph, to service requests that appear online at points in the metric sapce or at nodes of the weighted graph. A formal definition can be as follows.

Let M be a metric space and $d : M \times M \to \mathbb{R}$ be the distance function such that d is non-negative, symmetric and follows the triangle inequality. For simplicity, we allow distinct points in M to be at zero distance. Thus M is a *pseudometric* rather than a metric. We call an ordered set of k points in M to be a configuration. Let M^k denote the set of all possible configurations in M.

We extend the notion of d from M to M^k. Let $\mathsf{d} : M^k \times M^k \to \mathbb{R}$ denote the distance between configurations in M. For $C_1, C_2 \in M^k$, $\mathsf{d}(C_1, C_2)$ is the value of the minimum-weight perfect matching between the points of C_1 and C_2. In short, it is the minimum distance travelled by k servers to change their configuration from C_1 to C_2.

The server problem $\mathsf{S} = (k, \mathsf{M}, C_0, \mathsf{r})$ is defined by the number of servers k, a metric $\mathsf{M} = (M, d)$, an initial configuration $C_0 \in M^k$ and a sequence of requests $\mathsf{r} = (r_1, \ldots, r_m)$ where each $r_i \in M$ is a point in the metric. The solution is given by a seqence of configurations $(C_1, \ldots, C_m)$, where each $C_i \in M^k$, such that for all $t = 1, \ldots, m$, $r_t \in C_t$. The objective is to minimize the cost of the solution, given by $\sum_{t=1}^{m} \mathsf{d}(C_{t-1}, C_t)$, which is the total distance travelled by the servers.

An ***online algorithm*** computes each configuration C_t based on only the past, that is only on $r_1, \ldots, r_t$ and $C_1, \ldots, C_{t-1}$. An ***offline algorithm*** may also use knowledge of future requests $r_{t+1}, \ldots, r_m$. Thus an offline algorithm knows the entire request sequence before computing the solution.

1.1.1 Competitive Ratio

For a given initial configuration C_0 and a sequence of requests $\mathsf{r} = (r_1, \ldots, r_m)$, let $\mathsf{COST}_A(C_0, \mathsf{r})$ denote the cost of an algorithm A and let $\mathcal{O}(C_0, \mathsf{r})$ denote the cost of an optimal solution. Then we say that algorithm A has ***competitive ratio*** ρ if for every C_0 and r,

$$\mathsf{COST}_A(C_0, \mathsf{r}) \leq \rho \cdot \mathcal{O}(C_0, \mathsf{r}) + \Phi(C_0),$$

where the term $\Phi(C_0)$ depends only on the initial configuration C_0 and is independent of the request sequence r.

The competitive ratio of an alorithm can also be called its approximation ratio. If an algorithm has competitive ratio ρ, it is said to be ρ-competitive.

Rather unexpectedly, it is likely that the competitve ratio of an algorithm is independent of the metric space, provided it has more than k distinct points. This is given by the k-server conjecture [11].

Conjecture 1 (k-server conjecture) *For every metric space with more than k distinct points, the competitive ratio of the k-server problem is exactly k.*

The k-server conjecture is supported by all results on the problem since it was first stated. The conjecture has neither been proved nor disproved and is open, but has been proved for $k = 2$ and some special metric spaces.

We specially consider the metric space of the cycle.

1.1.2 The Cycle Metric Space

The cycle metric space on p points $\mathcal{C}_p$ is an undirected cyclic graph that consists of p nodes connected cyclically. Thus formally, consider the undirected graph $G = (V, E)$ where $V = \{a_1, \ldots, a_p\}$ and for $j = 1, \ldots, p-1$, $\{a_j, a_{j-1}\} \in E$, and $\{a_p, a_1\} \in E$. Let the distance function $d : V \times V \to \mathbb{N}$ be such that, $d(a, a) = 0$, for all $a \in V$, and $d(a, b)$ denotes the length of the shortest path between the nodes a and b, for $a, b \in V$ and $a \neq b$.

The metric space denoted by $\mathcal{C}_p = (V, d)$ is the cycle metric space on p points.

For the cycle, given a point a, we denote the point furthest away from a (the diametrically opposite point) as $\overline{a}$.

$$\overline{a} = \arg\max_{x \in V} d(a, x)$$

In this report, we mainly focus on finite cycles. It has been proven for all metric spaces that the competitive ratio for $k = 2$ servers is 2 [11]. We take a look at the case for $k = 3$ servers.

1.2 Related Work

The server problem was first defined by Manasses, McGeogh and Sleator [11] in 1988. It was a special case of the online metrical task systems problem stated by Borodin et al. [3, 4] earlier. Manasse et al. showed few important results – They showed that no online algorithm can have competitive ratio less than k, as long as the metric space has more than k distinct points. Further, they showed that the competitive ratio is exactly 2 for for the special case of $k = 2$ and that it is exactly k for all metric spaces with $k + 1$ points. With this evidence, they posed the k-server conjecture (Conjecture 1).

Computer experiments on small metric spaces verified the conjecture for $k = 3$. The conjecture was shown to hold for the line (1-dimensional Euclidean space) [5] and for tree metric spaces [6]. An optimal algorithm for the offline problem was also established [5]. In 1994, a dramatic improvement was shown by Koutaoupias et al. [10] which established the work function algorithm and showed that it has competitive ratio $2k - 1$. This remains the best known bound [9] and there has been limited progress on the server problem. In a recent survey [9], Koutsoupias analyses some major results about the problem, specially concerning the 1-dimensional Euclidian metric, tree metrics and metric spaces with $k + 1$ points.

Two special cases of the server problem we are interested in are the 3-server problem [7, 2] and the k-server problem on a cycle [8]. Any progress on these problems may lead to new paths to attack the k-server conjecture. For both these cases, nothing better than the $2k - 1$ bound is known.

Chapter 2

Algorithms for the Server Problem

2.1 Optimal Algorithm

We take a look at at on offline strategy to find a solution to the k-server problem, provided the entire request sequence is given in advance. The dynamic programming algorithm of Manasse et al. [11] is especially suited for cases where the number of requests dramatically exceeds the number of points in the metric. It has $O(nm\binom{m}{k})$ running time and at least $O(\binom{m}{k})$ space usage for n requests and m points in the metric. Later in 1991, Chrobak et al. gave an $O(kn^2)$ algorithm for the k-server problem to serve a sequence of n requests known in advance on a metric [5].

The algorithm, given by Chrobak et al., reduces the k-server problem to a minimum-cost maximum flow problem in a directed acyclic graph (DAG). If there are k servers $s_1, \ldots, s_k$ and a sequence of n requests $r_1, r_2 \ldots, r_n$, we create a DAG as follows –

- The vertex set is $V = \{s, s_1, \ldots, s_k, r_1, r_1, r'_1, r_2, r'_2, \ldots, r_n, r'_n, t\}$
- The node s is the source and t is the sink
- For every $i \in \{1, \ldots, k\}$, there is an edge of cost 0 from s to s_i and an edge of cost 0 from s_i to t
- For every $j \in \{1, \ldots, n\}$, there is an edge of cost 0 from r'_j to t

- For each pair $(i, j) \in \{1, \ldots, k\} \times \{1, \ldots, n\}$, there is an edge from s_i to r_j of cost equal to the distance between the location of the ith server in the initial configuration and the location of the jth request
- For every $i < j$ there is an endge from r'_i to r_j of cost equal to the distance between the ith and the jth requests
- For every $i \in \{1, \ldots, n\}$, there is an edge from r_i to r'_i of cost $-K$, where K is an extremely large real number.

It is shown [5] that the maximum flow in this graph is k and a maximum flow with minimum cost can be found in $O(kn^2)$ time. Further, the flow can be decomposed into k edge-disjoint $s \to t$ paths, the ith path passing through s_i. In the optimal solution for the problem, the ith server will serve exactly those requestscontained in the $s \to t$ path passing through s_i.

2.2 Work Function Algorithm

In 1994, Koutsoupias et al. [10] established a natural online algorithm based on the dynamic programming approach, called the ***work function algorithm*** (WFA). For metric M, an initial configuration C_0 and a sequence of requests $\mathsf{r} = (r_1, \ldots, r_t)$, we define for every configuration $X \in M^k$, the function $w(C_0, (r_1, \ldots, r_t), X)$ to be the cost of the optimal solution which starts at the configuration C_0, passes through (serves the requests) $r_1, \ldots, r_t$ (in that order) and ends at the configuration X. Therefore,

$$w(C_0, (r_1, \ldots, r_t), X) = \min\{\sum_{i=1}^{t+1} \mathsf{d}(C_{i-1}, C_i) \mid C_i \in M^k \wedge r_i \in C_i \wedge C_{t+1} = X\}$$

For a given server problem, we have a fixed C_0 and fixed $\mathsf{r} = (r_1, \ldots, r_n)$ and w becomes a real function of M^k. Such a function is called the *work function* and is denoted as $w_{C_0,(r_1,\ldots,r_t)}$ or simply as w_t, since C_0 and $(r_1, \ldots, r_t)$ are fixed. The value of the work function $w_t(X) = w(C_0, (r_1, \ldots, r_t), X)$ can be computed in a dynamic programming approach as,

$$w_i(X) = \min_{Z \in M^k, r_i \in Z} \{w_{i-1}(Z) + \mathsf{d}(Z, X)\}$$

using the base values $w_0(X) = \mathsf{d}(C_0, X)$.

To service a request r_t, with the current configuration C_{t-1}, the work function algorithm moves to the configuration C_t such that $r_t \in C_t$ and such

that the quantity $w_t(C - t) + \mathsf{d}(C_{t-1}, C_t)$ is minimized.

It has been proven [10, 9] that the work function algorithm for the 2-server problem has competitive ratio 2, and that the work function algorithm for the general k-server problem has competitive ratio of at most $2k - 1$ [9].

Chapter 3

Implementation, Tests and Results

3.1 Implementation of Algorithms

The implementation of the algorithms and test cases is done in Python 3.2.3. In the sections that follow k represents the number of servers (mostly and unless specified otherwise we consider $k = 3$), p represents the number of points on the cycle, n represents the length of the request sequence and the approximation ratio is represented as $\alpha = \frac{\mathsf{COST}_{\mathsf{WFA}}(C_0,r)}{\mathcal{O}(C_0,r)}$.

3.1.1 Optimal Algorithm

We implemented the optimal offline algorithm (Section 2.1). For this we had to implement an algorithm to evaluate the minimum cost maximum flow of an directed acyclic network. For the minimum cost maximum flow problem, we use minimum cost augmentation [12] using successive shortest paths [1] based on the cost of edges. The flow along the minimum cost path is successively augmented to the flow of the network and the residual network is again iteratively augmented, until no path from the source to the sink remains.

Since edges may have negative costs, we cannot use Dijkstra's algorithm. We need to find shortest path in a network where edges have negative weights, which calls for use of the Bellman-Ford algorithm. However, Bellman-Ford algorithm would take worst case $O((k+n)^2 n)$ time complexity, for k servers and a sequence of n requests. Instead we establish node potentials and reduced costs for edges using Bellman-Ford algorithm. Since

reduced costs are non-negative and do not change shortest paths between nodes, we use Dijkstra's algorithm for finding shortest path. After augmenting the flow along the found path, the node potentials and reduced costs need to be updated.

`NetworkFlow.py` contains the implementation for the minimum cost maximum flow algorithm. The `FlowNetwork` class creates the directed acyclic network. Functions `add_vertex` and `add_edge` help build the required network. The function `max_flow` and `min_cost_max_flow` find the maximum flow and the maximum flow with minimum cost respectively.

Then, we create a directed acyclic network based on the number of servers and request sequence and compute the optimal service strategy based on the maximum flow with minimum cost of the created network. The `ServerSpace` class is created for these functionalities. It takes input the initial configuration of the servers and the distance function of the metric. A sequence of requests can then be added and the function `process_requests` processes these requests and produces the output strategy.

3.1.2 Work Function Algorithm

The work function algorithm [10] (desribed in Section 2.2) is implemented in `WorkFunction.py`. We use the dynamic programming approach to find the appropriate server to service the oncoming request. The already computed work function values are stored in the table `stored`, which is a dictionary type data structure. The dictionary data structure ensures fast $O(1)$ access to associations and also makes sure that rows in the table and elements in the rows are created only when needed. Thus, space in memory is occupied by only values of the function that are computed (i.e. there are no empty cells), giving efficient usage of memory.

While an object of the `WorkFunction` class is created, it takes as input the distance function of the metric space and the initial configuration C_0 of the servers. Further, `add_request` and `delete_request` can be used to add or remove requests to or from the end of the request sequence. The function `value` returns the value of the work function given the appropriate arguments. `process_request` takes an integer i and processes the request at the ith position in the sequence based on the work function algorithm and produces the server that serves the request and updates the configuration of the system to after that of the request has been served.

3.2 The Cycle Metric Space

For all tests we considered a cycle with small number of points (mostly $p = 20$ or $p = 36$). Implementing the cycle as a metric space was simple. For the cycle with p points, each point was taken as an integer from 0 to $p-1$. The distance between two points in the metric is calculated based on their difference modulo p. If the difference (modulo p) is greater than $p/2$ we take the shorter distance along the other direction.

Code 3.1: Code for distance function of a cycle with p points

```
def cycle_metric(a,b) :
        d = (b-a)%p
        if d > p//2 :
                return (p-d)
        else :
                return d
```

3.3 Generating Requests

The request sequence to be generated for the problem holds an important place. A good strategy to produce subsequent requests could have potential ways to attack the general k-server conjecture.

3.3.1 Random Generation

An initial strategy (tried out mostly to check the implementation) was to generate requests randomly at one of the points in the metric. This did not produce very interesting results.

As the sequence was random, both the work function algorithm and optimal strategy give similar performance. The ratio $\alpha = \frac{\mathsf{COST}_{\mathsf{WFA}}(C_0,r)}{\mathcal{O}(C_0,r)}$ (henceforth referred to as the approximation ratio of the algorrithm is quite close to 1.0 (See Table 3.1).

3.3.2 Generation on the Longest Arc

We consider only the case of $k = 3$. The 3 servers divide the cycle into 3 arcs. For a given configuration, we take the longest of the 3 arcs and place the new requests at the mid-point of the arc (i.e. point equidistant from both servers at the end of the arc). If two points are in the mid of the arc, we randomly choose one of the two.

p	n	$\mathsf{COST}_{\mathsf{WFA}}(C_0, r)$	$\mathcal{O}(C_0, r)$	α
20	100	202	174	1.161
20	100	184	169	1.089
20	100	186	152	1.224
20	100	191	173	1.104
20	100	211	189	1.116
36	100	380	330	1.152
36	100	362	306	1.183
36	100	370	337	1.098
36	100	320	294	1.088
36	100	339	300	1.130

Table 3.1: Approximation Ratio of WFA for Random Generation

Table 3.2 represents the values for the approximation ration of the work function algorithm for this request generation strategy. For more data of the tests refer Section 3.5.

p	n	$\mathsf{COST}_{\mathsf{WFA}}(C_0, r)$	$\mathcal{O}(C_0, r)$	α
20	100	500	174	2.874
20	100	497	171	2.906
20	100	499	171	2.918
20	50	248	89	2.787
20	50	253	87	2.908
20	50	253	90	2.811
20	50	252	88	2.864
36	100	896	310	2.890
36	100	899	313	2.872
36	100	901	313	2.879
36	50	450	163	2.761
36	50	451	154	2.929
36	50	453	167	2.713
36	50	449	154	2.916

Table 3.2: Approximation Ratio of WFA for Generation on Longest Arc

3.3.3 Worst Cost Generation

The worst cost generation generates the request for which the immediate cost for processing the request based on the work function algorithm would be maximum.

$$r_{t+1} = \arg\max_{r \in \mathcal{C}_p}(\mathsf{COST}_{\mathsf{WFA}}(C_0, (r_1, \ldots, r_t, r)) - \mathsf{COST}_{\mathsf{WFA}}(C_0, (r_1, \ldots, r_t))$$

As it turns out worst cost generation is much similar to generartion in the middle of the longest arc. Thus it produces very similar results to the previous request sequence generation strategy. (See Section 3.5, Table 3.5). For both these strategies, the ratio is very close to 3.0 but always less than 3.0.

3.3.4 Repeated Shifting

Repeated shifting involves shifting requests between adjacent locations until both of them are occupied by servers. We generate the request in phases. In each phase we take the longest arc formed by the server configuration and take the mid-point of the arc and one adjacent location to this mid-point. Then we alternately generate requests at these 2 locations until servers occupy both these locations. Then we move on to the next phase. The following code gives an easy implementation of this strategy.

Code 3.2: Code for single phase of repeated shifting

```
mid = generate_middle(longest_arc(config))
next = (mid+1)%p
while True :
        if mid not in config :
                add_request(mid)
        elif next not in config :
                add_request(next)
        else :
                break
        i = i+1
```

3.4 Evaluating Performance

For evaluating performance further (specifically for the repeated shifting strategy), we also consider some other parameters.

We call a configuration C_i ***terminal*** if a particular server in the configuration does not move after this configuration. Formally, we say that C_i is a terminal configuration if there exists $s \in \{1, \ldots, k\}$ such that, for all $t > i$, $C_t[s] = C_i[s]$, where $C_i[j]$ represents the jth element (i.e. position of the jth server) in the ordered set configuration. C_i.

Also, for a point on the cycle r, the point $\overline{r}$, as described in Section 1.1.2, represents the diametrically opposite point on the cycle.

We are particularly interested in the following performance parameters–

- $\mathcal{D} = \sum_{i=1}^{n}(w_i(C_{i-1}) - w_{i-1}(C_{i-1}))$
- $\mathcal{A} = \sum_{i=1}^{n}(w_i(\overline{r_i}^k) - w_{i-1}(\overline{r_i}^k))$
- $\mathcal{T} = \sum_{C_i \text{ is terminal}} w_i(C_i)$

p	n	$\overline{\mathcal{D}}$	$\overline{\mathcal{A}}$	$\overline{\mathcal{T}}$
20	100	260	270	213
20	200	520	532	454
20	250	648	658	574
20	500	1270	1367	1151
20	1000	2576	2588	2386
20	2000	5056	5077	4628
20	5000	12735	12772	11776
20	10000	25147	25158	23308
36	100	281	299	201
36	200	558	572	437
36	500	1343	1362	1120
36	1000	2654	2675	2271
36	2000	5292	5332	4577
36	5000	13093	13170	11315
36	10000	26301	26399	22921

Table 3.3: Performance Parameters for Repeated Shifting (Mean Data, for individual case data see Section 3.5)

For all test cases (Table 3.3), it is observed that, $\mathcal{T} < \mathcal{D} < \mathcal{A} <$. Thus,

$$\sum_{C_i \text{ is terminal}} w_i(C_i) < \sum_{i=1}^{n}(w_i(C_{i-1}) - w_{i-1}(C_{i-1})) < \sum_{i=1}^{n}(w_i(\overline{r_i}^k) - w_{i-1}(\overline{r_i}^k))$$

3.5 Test Results

This section covers the major aspects of the results. However, this is not the exhaustive cover of all the values obtained during the testing. The complete set of results can be found along with the implementation (in the `data` folder).

p	n	$\mathsf{COST}_{\mathsf{WFA}}$	$\mathcal{O}$	α
20	100	202	174	1.161
20	100	184	169	1.089
20	100	186	152	1.224
20	100	191	173	1.104
20	100	211	189	1.116
20	100	190	178	1.067
20	100	177	165	1.073
20	100	198	162	1.222
20	100	211	179	1.179
20	100	194	170	1.141
36	100	391	313	1.249
36	100	396	328	1.207
36	100	337	308	1.094
36	100	402	350	1.149
36	100	397	313	1.268
36	100	380	330	1.152
36	100	362	306	1.183
36	100	370	337	1.098
36	100	320	294	1.088
36	100	339	300	1.130
90	50	423	363	1.165
90	50	418	343	1.219
90	50	403	389	1.036
90	50	473	403	1.174
90	50	388	388	1.000
90	50	501	419	1.196
90	50	425	379	1.121
90	50	481	393	1.224
90	50	404	374	1.080
90	50	449	387	1.160

p	n	$\mathsf{COST}_{\mathsf{WFA}}$	$\mathcal{O}$	α
90	100	832	794	1.048
90	100	872	796	1.095
90	100	1010	834	1.211
90	100	855	751	1.138
90	100	866	786	1.102
90	100	991	843	1.176
90	100	861	737	1.168
90	100	848	788	1.076
90	100	913	755	1.209
90	100	926	804	1.152
100	50	440	390	1.128
100	50	381	375	1.016
100	50	539	453	1.190
100	50	407	386	1.054
100	50	444	432	1.028
100	50	532	452	1.177
100	50	523	455	1.149
100	50	537	442	1.215
100	50	523	453	1.155
100	50	549	455	1.207
100	100	900	840	1.071
100	100	980	860	1.140
100	100	1155	907	1.273
100	100	1022	870	1.175
100	100	1017	867	1.173
100	100	953	907	1.051
100	100	1060	910	1.165
100	100	1123	947	1.186
100	100	1075	949	1.133
100	100	1048	933	1.123

Table 3.4: Data for Random Generation for Large Number of Points

p	n	$\mathsf{COST}_{\mathsf{WFA}}$	$\mathcal{O}$	α	p	n	$\mathsf{COST}_{\mathsf{WFA}}$	$\mathcal{O}$	α
100	100	3080	1277	2.412	90	100	2768	1153	2.401
100	100	3088	1279	2.414	90	100	2771	1160	2.389
100	100	3092	1294	2.389	90	100	2772	1123	2.468
100	100	3092	1305	2.369	90	100	2782	1160	2.398
100	100	3103	1309	2.371	90	100	2790	1161	2.403
100	100	3104	1311	2.368	90	100	2812	1184	2.375
100	100	3109	1289	2.412	90	100	2816	1196	2.355
100	100	3110	1312	2.370	90	100	2819	1186	2.377
100	100	3110	1317	2.361	90	100	2821	1190	2.371
100	100	3115	1310	2.378	90	100	2825	1188	2.378
100	200	6146	2577	2.385	90	200	5511	2296	2.400
100	200	6171	2561	2.410	90	200	5622	2376	2.366
100	200	6198	2599	2.385	90	200	5627	2376	2.368
100	200	6199	2594	2.390	90	200	5628	2371	2.374
100	200	6202	2597	2.388	90	200	5638	2391	2.358
100	50	1543	653	2.363	90	50	1384	578	2.394
100	50	1553	670	2.318	90	50	1385	585	2.368
100	50	1555	656	2.370	90	50	1385	590	2.347
100	50	1557	638	2.440	90	50	1390	572	2.430
100	50	1559	664	2.348	90	50	1407	591	2.381
100	50	1560	653	2.389	90	50	1411	594	2.375
100	50	1561	652	2.394	90	50	1414	615	2.299
100	50	1564	654	2.391	90	50	1415	591	2.394
100	50	1565	665	2.353	90	50	1416	604	2.344
100	50	1568	636	2.465	90	50	1423	606	2.348
100	75	2311	967	2.390	90	75	2076	873	2.378
100	75	2317	968	2.394	90	75	2083	871	2.392
100	75	2319	964	2.406	90	75	2089	873	2.393
100	75	2323	972	2.390	90	75	2099	883	2.377
100	75	2325	988	2.353	90	75	2105	893	2.357
100	75	2326	979	2.376	90	75	2111	903	2.338
100	75	2330	974	2.392	90	75	2114	915	2.310
100	75	2333	985	2.369	90	75	2117	893	2.371
100	75	2335	997	2.342	90	75	2120	915	2.317
100	75	2340	984	2.378	90	75	2121	899	2.359

Table 3.5: Data for Worst Cost Generation for Large Number of Points

Case No.	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{T}$
1	250	259	205
2	269	278	212
3	250	259	205
4	250	259	205
5	254	261	203
6	259	270	214
7	247	259	203
8	247	259	203
9	260	270	214
10	254	261	203
11	247	262	203
12	260	272	214
13	250	264	209
14	263	281	213
15	255	264	206
16	266	277	214
17	284	293	238
18	285	293	238
19	284	294	241
20	261	270	216

Table 3.6: Repeated Shifting ($p = 20$, $n = 100$)

Case No.	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{T}$
21	520	531	464
22	498	511	436
23	530	542	469
24	520	530	442
25	529	539	468
26	518	533	464
27	502	509	436
28	514	536	436
29	529	542	462
30	495	505	433
31	510	522	440
32	527	539	466
33	531	542	467
34	522	535	460
35	518	526	442
36	530	539	467
37	505	518	436
38	522	543	441
39	550	564	492
40	535	543	468

Table 3.7: Repeated Shifting ($p = 20$, $n = 200$)

Case No.	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{T}$
111	12687	12701	12047
112	12489	12761	11100
113	12702	12711	11526
114	12714	12728	12074
115	12698	12708	11527
116	12688	12701	12052
117	12729	12740	11555
118	12335	12345	11491

Table 3.8: Repeated Shifting ($p = 20$, $n = 5000$)

Case No.	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{T}$
121	24669	24679	23006
122	25427	25437	23097
123	24667	24678	23006
124	26290	26300	24406
125	24686	24698	23029

Table 3.9: Repeated Shifting ($p = 20$, $n = 10000$)

Case No.	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{T}$
126	157	171	83
127	148	165	86
128	139	158	81
129	132	150	76
130	151	167	87
131	138	157	78
132	162	174	84
133	135	157	79
134	158	171	85
135	187	206	124
136	139	157	78
137	162	173	85
138	151	165	80
139	141	161	81
140	140	156	80
141	148	163	82
142	154	167	83
143	151	168	88
144	156	171	84
145	147	164	80

Table 3.10: Repeated Shifting ($p = 36$, $n = 100$)

Case No.	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{T}$
146	276	293	193
147	305	328	230
148	262	280	187
149	266	284	187
150	271	287	189
151	279	294	193
152	317	336	238
153	307	330	230
154	274	287	189
155	275	290	193
156	317	338	238
157	266	284	188
158	274	287	191
159	265	284	190
160	315	333	235
161	262	280	187
162	264	280	187
163	277	295	199
164	268	282	185
165	281	299	196

Table 3.11: Repeated Shifting ($p = 36$, $n = 200$)

Case No.	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{T}$
238	13063	13214	11209
239	13200	13219	11577
240	12932	13018	10993
241	13062	13213	11215
242	13061	13213	11209
243	12942	13026	10996
244	13145	13162	11530
245	12946	13026	10993

Table 3.12: Repeated Shifting ($p = 36$, $n = 5000$)

Case No.	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{T}$
246	26152	26443	22499
247	26325	26342	23134
248	26820	26838	23830
249	25872	26018	22009
250	26336	26354	23136

Table 3.13: Repeated Shifting ($p = 36$, $n = 10000$)

Chapter 4

Conclusions

4.1 Summary of Current Progress

We have looked at the k-server problem, rather specifically on the 3-server problem on the cycle. We have tried a new aspect of the server problem and tried to approach the k-server conjecture accordingly. We generated requests with different patterns and strategies to analyse different aspects of the server problem. Large number of tests were performed corresponding to each of these strategies and separated raw data and consolidated data for every case has been catalogued accordingly. All results of the tests performed support the k-server conjecture.
The implementation for the algorithms is complete in Python (3.2.3). The algorithms implemented (as of April 2014) are,

- Minimum-cost maximum flow algorithm [12, 1],
- Optimal offline algorithm for the general k-server problem, when the request sequence is known in advance [5],
- Work function algorithm for the general k-server problem [10].

The data gives new insights on the k-server problem, though formulation of strong results based on the data remains to be done.

4.2 Challenges and Future Work

A major challenge faced was usage of memory. The work function algorithm for k-server problem on the cycle with p points and a n request sequence,

in the worst case uses $O(p^k n)$ space. For the case we see for $k = 3$ servers, the space usage is $O(p^3 n)$. For large number of points in the metric space, the space requirements increase drastically. However for a fixed number of points, the requirement increases linearly with the size of the request sequence. Thus on a standard machine with memory of the order of hundreds of megabytes to few gigabytes, this is typically not much of a concern, since we can easily handle tens of millions of requests.

Storing only those values of the work function which are computed and not allocating space for empty blocks (which is what is done in the implementation), drastically reduces memory space use. An alternate approach for very large request sequences is to destroy unused values in the table, which were previously created, and create them again when necessary. This will reduce the space complexity, but will increase the running-time complexity (since it defeats the purpose of dynamic programming). A trade-off between these two can be attempted to get the best of both worlds.

On the other hand, the optimal algorithm the space requirement is independent of the size of the metric, but rather depends on n and k. For fixed $k = 3$, the space requirement, assuming that the corresponding directed network formed for the minimum-cost-maximum-flow is stored in edge-list or adjacency-list form, will be $O(n^2)$. Thus, the space requirement is quadratic with respect to the request sequence size. This is the best we can have, since there is no better way to fit the formed graph. This limits the size of the request sequence that can be processed on astandard machine (to few thousands of requests).

Also, the time complexity for computing the optimal strategy can be reduced if heap realization is used instead of simple realization in Dijkstra's algorithm for computing the maximum flow. However, both space and time complexities for getting the optimal solution seem highly constrained, unless a better algorithm is established.

Bibliography

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

[2] Wolfgang W. Bein, Marek Chrobak, and Lawrence L. Larmore. The 3-server problem in the plane. *Theor. Comput. Sci.*, 289(1):335–354, 2002.

[3] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal online algorithm for metrical task systems. In *STOC*, pages 373–382, 1987.

[4] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[5] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.

[6] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991.

[7] Marek Chrobak and Lawrence L. Larmore. Generosity helps or an 11-competitive algorithm for three servers. *J. Algorithms*, 16(2):234–263, 1994.

[8] Amos Fiat, Yuval Rabani, Yiftach Ravid, and Baruch Schieber. A deterministic $\mathcal{O}(k^3)$-competitive algorithm for the circle. 1991. Manuscript.

[9] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

[10] Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. In *FOCS*, pages 394–400, 1994.

[11] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *STOC*, pages 322–333, 1988.

[12] Robert Endre Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.