New Results on the k-Truck Problem*

Weimin Ma[1,2], Yinfeng Xu[1], Jane You[2], James Liu[2], and Kanliang Wang[1]

[1] School of Management, Xi'an Jiaotong University, Shaanxi 710049, PRC
[2] Dept. of Computing, Hong Kong Polytechnic Uinversity, Hung Hom, Kowloon, Hong Kong
cswmma@comp.polyu.edu.hk

Abstract. In this paper, some results concerning the k-truck problem are produced. First, the algorithms and their complexity concerning the off-line k-truck problem are discussed. Following that, a lower bound of competitive ratio for the on-line k-truck problem is given. Based on the *Position Maintaining Strategy* (PMS), we get some new results which are slightly better than those of [1] for general cases. We also use the *Partial-Greedy Algorithm* (PG) to solve this problem on a special line. Finally, we extend the concepts of the on-line k-truck problem to obtain a new variant: *Deeper On-line k-Truck Problem* (DTP).

1 Introduction

On-line problem and their *competitive analysis* have received considerable interest for about twenty years. S. Albers and S. Leonardi [2] coined out a comprehensive survey of this domain. On-line problems had been systematically investigated only when Sleator and Tarjian [3] suggested comparing an on-line algorithm to an optimal off-line algorithm and Karlin, Manasse, Rudolph and Sleator [4] coined the term *competitive analysis*. The task system, the k-server problem, and on-line/off-line games ([5], [6] and [7]) all attempt to model on-line problems and algorithms. In this paper, we first discussed the algorithms and its complexity concerning the off-line k-truck problem. Following that, a lower bound of competitive ratio for the on-line k-truck problem is given. Especially, based on the PMS, we get some new results for the general cases. In addition, we also use the PG to solve this problem on a special line and prove that PG is a $(1 + (n - k)/\theta)$ -competitive algorithm for this case. Finally, we extend the concepts of the on-line k-truck problem to obtain a new variant: DTP.

2 Preliminaries

The k-truck problem can be stated as follows. We are given a metric space M, and k trucks which move among the points of M, each occupying one point of M.

* The authors would like to acknowledge the support of Central Research Grant GV-975 of the Hong Kong Polytechnic University and Research Grant from NSF of China. No.19731001

O.H. Ibarra and L. Zhang (Eds.): COCOON 2002, LNCS 2387, pp. 504–513, 2002.

Repeatedly, a request (a pair of points $x, y \in M$) appears. To serve a request, an empty truck must first move to x and then move to y with goods from x. How to minimize the total cost of all trucks? Obviously, the k-truck problem aims at minimizing the cost of all trucks. Because the cost of trucks with goods is different from that of trucks without goods on the same distance, the total distance cannot be considered as the objective to be optimized. For simplicity, we assume that the cost of a truck with goods is θ times that of one without goods on the same distance. We can then take $(1+\theta)$ times of the empty loaded distant as the objective of optimization.

The Model. Let $G = (V, E)$ denote an edge weighted graph with n vertices and the weights of edges satisfying the triangle inequality, where V is a metric space consisting of n vertices, and E is the set of all weighted edges. We assume that the weight of edge (x, y) is denoted by $d(x, y)$ and the weights are symmetric, i.e., for all $x, y, d(x, y) = d(y, x)$. We assume that k trucks occupy a k-vertexes which is a subset of V. A service request $r = (a, b), a, b \in V$ implies that there are some goods on vertex a that must be moved to vertex b (for simplicity, we assume that the weight of the goods is same all the time). A service request sequence R consists of some service request in turn, namely $R = (r_1, ..., r_m)$, where $r_i = (a_i, b_i), a_i, b_i \in V$. All discussion is based on the following essential assumptions: (1) Graph G is connected; (2) When a new service request occurs, k trucks are all free; (3) All trucks have the same load weight and the cost of a truck with goods is θ times that of one without goods on the same distance, and $\theta \geq 1$. For a known sequence $R = (r_1, ..., r_m)$, let $C_{\mathrm{OPT}}(R)$ be the optimal total cost after finishing it. For a new service request r_i, if scheduling algorithm A can schedule without information regarding the sequence next to r_i, we call A an on-line algorithm. For on-line algorithm A, if there are constants α and β satisfying

$$C_{\mathrm{A}}(R) \leq \alpha \cdot C_{\mathrm{OPT}}(R) + \beta,$$

then for any possible R, A is called a competitive algorithm, where $C_{\mathrm{A}}(R)$ is the total cost with algorithm A to satisfy sequence R.

If there is no limit for the R and θ, the on-line truck problem is called P. In problem P, if for any $r_i = (a_i, b_i), a_i, b_i$ and $\theta > 1$ holds, the problem is called $P1$. In problem P, if there is no limit for any $r_i = (a_i, b_i)$, but if $\theta = 1$, the problem is $P2$. In $P2$, if $d(a_i, b_i) > 0$, namely $a_i = b_i$, the problem is called $P3$. In problem P, if $d(a_i, b_i) = 0$, namely $a_i = b_i$, it is called $P4$.

Lemma 1. *[9] There exists an on-line algorithm for the k-server problem with the competitive ratio* $2k - 1$.

Lemma 2. *[1] Letting OPT be an optimal algorithm for an request sequence* $R = (r_1, ..., r_m)$, *then we have* $C_{OPT}(R) \geq C_{OPT}(\sigma) + \sum_{i=1}^{m} (\theta - 1) \cdot d(a_i, b_i)$, *where* $\sigma = ((a_1, a_1), ..., (a_m, a_m))$ *and* $r_i = (a_i, b_i)$.

Lemma 3. *[1] For any algorithm A for a request sequence* $R = (r_1, ..., r_m)$, *we have* $C_A(R) \geq \sum_{i=1}^{m} \theta \cdot d(a_i, b_i)$, *and* $C_{OPT}(R) \geq \sum_{i=1}^{m} \theta \cdot d(a_i, b_i)$.

Lemma 4. *[10] There exists an on-line algorithm for the k-server problem on a real line with the competitive ratio k.*

Position Maintaining Strategy (PMS) [8]

For the present request $r_i = (a_i, b_i)$, after a_i is reached, the truck reaching a_i must move from a_i to b_i with the goods to complete r_i. When the service for r_i is finished, the PMS moves the truck at b_i back to a_i (empty) before the next request arrives.

3 Off-Line Problem

In this section, two solutions for the off-line k-truck problem are discussed.

Definition (Configuration) *On the metric space M, a possible position of k trucks is called a configuration. That is, a configuration is a special k-multiset whose elements consist of at least one and at most k points of space M. Here, the special means that in the multiset the same node can be repeated from one to k times.*

3.1 Dynamic Programming (DP) Solution

In [6], a DP solution was given for the famous k-server problem. Similarly, we can develop a DP solution for the k-truck problem.

Lemma 5. *On a given graph G with n nodes, the number of possible configurations of all k trucks is $\binom{n+k-1}{n-1}$, where $k \leq n$.*

Proof. Assume that all k trucks and all n nodes line up along a line from left to right, thus there are $n + k$ locations on which there is either a truck or a node. Following that, we move all trucks between two nodes i and j (assuming that node i is right to node j and that there are not any other nodes between them) to node i. If there are not trucks between the two nodes, the meaning of this operation is that no truck is moved on to node i. In addition, in order to move all trucks on some nodes according to the above rules, we need to let the extreme right location be a node. The final task is to choose $n-1$ locations, on which we will arrange the remaining $n-1$ nodes, from the $n+k-1$ locations. Obviously, we have $\binom{n+k-1}{n-1}$ choices. □

Let function $C_{\text{OPT}}(R, S)$ denote the cost of the minimum cost algorithm that handles request sequence R and ends up in configuration S. As in paper [6], we can compute this function recursively as follows, assuming that the trucks are initially in configuration S_0

$$C_{\text{OPT}}(\varepsilon, S) = \begin{cases} 0 & \text{if } S = S_0 \\ \text{undefine} & \text{otherwise} \end{cases}$$

$$C_{\text{OPT}}(Rr_i, S) = \begin{cases} \min_T (C_{\text{OPT}}(R, T) + d(T, \theta \cdot (a_i, b_i), S)) & \text{if } S = S_0 \\ \text{undefine} & \text{otherwise} \end{cases}$$

where $d(T, \theta \cdot (a_i, b_i), S)$ is the cost of transition from configuration T to configuration S and the last operation of transition is $a_i \rightarrow b_i$ (satisfying the request r_i at cost $\theta \cdot (a_i, b_i)$), T and S denote the configurations at time $i-1$ and time i, respectively, and ε denotes the empty request sequence.

Theorem 1. *The above optimal off-line algorithm for the k-truck problem can give an optimal solution with time proportional to* $m \cdot \binom{n+k-1}{n-1}^2$*, where m is the length of the request sequence (the number of requests).*

Proof. Let $|R| = m$, we can develop a table-building method according to the above discussion. Build a table with $|R| + 1$ rows, each of which implies a subsequence of request sequence R, and $\binom{n+k-1}{n-1}$ columns each of which denote a possible configuration of trucks. Namely, the entry in row i and column j is $C_{\mathrm{OPT}}(R_i, S_j)$, where R_i is the subsequence of R of length i. Each row of the table can be built from the previous one within time $\binom{n+k-1}{n-1}^2$. Furthermore, only $|R| = m$ rows need these computations. The proof is completed. □

3.2 Minimum Cost Maximum Flow (MCMF) Solution

In [11], MCMF was used to resolve the off-line k-server problem. Our objective is to find an optimal strategy to serve a sequence of m requests with k trucks, if the request sequence is given in advance. Assume that the k-trucks initially occupy one point, the origin. And denote the i-th request by the binary-tuple (a_i, b_i). If there are m requests, the inputs to our problem are the superdiagonal entries of an $(m+1) \times (m+1)$ matrix, whose $(0, j)$ entry is the sum of cost from the original to the location of j-request start a_j (empty) and then to the request destination b_j (with the goods), $j = 1, 2, ..., m$, and whose (i, j) entry is the sum of cost from the location of i-request destination to the location of j-request start and then to the relevant destination with goods, $j, 1 \leq i < j \leq m$.

Theorem 2. *There is an* $O(km^2)$*-time off-line algorithm to find an optimal schedule for k trucks to serve a sequence of m requests (whether or not the triangle inequality holds).*

Proof. We can resolve the off-line the k-truck problem (with or without triangle inequality) by reducing it to the problem of finding a minimum cost flow of maximum value in an acyclic network. Suppose that there are k trucks $t_1, ..., t_k$ and m requests $r_1, ..., r_m$, where $r_i = (a_i, b_i)$, and $i = 1, ..., m$, we can build the following $(2 + k + 3m)$-node acyclic network: the vertex set is $V = \{s, s_1, ..., s_k, a_1, b_1, b'_1, ..., a_m, b_m, b'_m, t\}$. In that vertex set, nodes s and t are the source and sink, respectively. Each arc of our network has a capacity one. There is an arc of cost 0 from s to each s_i, an arc of cost 0 form each b'_i to t, as well as an arc to t from each s_i, of cost 0. ¿From each s_i, there is an arc to a_j of cost equal to the distance from the origin to the location of a_j. ¿From each a_j, there is only an arc to b_j of cost equal to $\theta \cdot d(a_i, b_i)$. For $i < j$, there is an arc from b'_i to a_j of cost equal to the distance between b_i to a_j. Moreover, form b_i to b'_i there is an arc of cost $-K$, where K is an extremely large positive real. The constructing of the network is completed.

It is easy to know that the value of the maximum flow in this network is k. Using minimum-cost augmentation [12], we can find an integral min-cost flow of value k in time $O(km^2)$, because all capacities are integral and the network is acyclic. An integral $s \to t$ flow of value k can be decomposed into k arc-disjoint $s \to t$ paths, the ith one passing through s_i. Obviously, this flow saturates all of the (b_i, b_i') arcs, and hence corresponds to an optimal schedule for serving the requests, the ith server serving exactly those requests contained in the $s \to t$ path that passes through s_i, because $-K$ is so small. □

4 A Lower Bound

In this section we will give a lower bound of competitive ratio for the k-truck problem on a symmetric metric space. In other words, any general on-line algorithm for this problem, either a deterministic or a randomized algorithm, must have a competitive factor of at least $(\theta + 1) \cdot k/(\theta \cdot k + 2)$. In fact, we have actually proven a slightly more general lower bound on the competitive ratio. Suppose we wish to compare an on-line algorithm with k servers to an off-line one with $h \leq k$ servers. Naturally, the factor decreases when the on-line algorithm gets more servers than the off-line algorithm. We get the lower bound as $(\theta + 1) \cdot k/((\theta + 2) \cdot k - 2h + 2)$. A similar approach was taken in [6], where the lower bound and matching upper bound are given for the traditional k-server problem.

Theorem 3. *Let A be an on-line algorithm for the symmetric k-truck problem on a graph G with at least k nodes. Then, for any $1 \leq h \leq k$, there exist request sequences $R_1, R_2, ...$ such that: (1) For all i, R_i is an initial subsequence of R_{i+1}, and $C_A(R_i) < C_A(R_{i+1})$; (2) There exists an h-truck algorithm B (which may start with its trucks anywhere) such that for all i,$C_A(R_i) > (\theta+1) \cdot k \cdot C_B(R_i)/((\theta+2) \cdot k - 2h + 2)$.*

Proof. Without loss of generality, assume A is an on-line algorithm and that the k trucks start out at different nodes. Let H be a subgraph of G of size $k + 2$, induced by the k initial positions of A's trucks and two other vertexes. Define R, A's nemesis sequence on H, such that $R(i)$ and $R(i-1)$ are the two unique vertexes in H not covered by A and a request $r_i = d(R(i), R(i-1))$ occurs at time i, for all $i \geq 1$. Then

$$C_{\mathrm{A}}(R_t) = \sum_{i=1}^{t} (d(R(i+1), R(i)) + \theta \cdot d(R(i), R(i-1))) =$$
$$(1+\theta) \cdot \sum_{i=1}^{t-1} d(R(i+1), R(i)) + d(R(i+1), R(i)) + \theta \cdot d(R(1), R(0)),$$

because at each step R requests the node just vacated by A.

Let S be any h-element subset of H containing $R(1)$ but not $R(0)$. We can define an off-line h -truck algorithm $A(S)$ as follows: the trucks finally occupy the vertices in set S. To process a request $r_i = d(R(i), R(i-1))$, the following rule is applied: If Scontains $R(i)$, move the truck at node $R(i)$ to $R(i-1)$ with goods to satisfy the request, and update the S to reflect this change. Otherwise move the truck at node $R(i-2)$ to $R(i)$ without goods and then to $R(i-1)$ with goods, also to satisfy the request, and update S to reflect this change.

It is easy to see that for all $i > 1$, the set S contains $R(i-2)$ and does not contain $R(i-1)$ when step i begins. The following observation is the key to the rest of the proof: if we run the above algorithm starting with distinct equal-sized sets S and T, then S and T never become equal, for the reason described in the following paragraph.

Suppose that S and T differ before $R(i)$ is processed. We shall show that the versions of S and T created by processing $R(i)$, as described above, also differ. If both S and T contain $R(i)$, they both move the truck on node $R(i)$ to node $R(i-1)$, on which there is exactly not any truck. The other nodes have no changes, so S and T are still different and both S and T contain $R(i-1)$. If exactly one of S or T contains $R(i)$, then after the request exactly one of them contains $R(i-1)$, so they still differ. If neither of them contains $R(i)$, then both change by dropping $R(i-2)$ and adding $R(i-1)$, so the symmetric difference of S and T remains the same (non-empty).

Let us consider simultaneously running an ensemble of algorithms $A(S)$, starting from each h-element subset S of H containing $R(1)$ but not $R(0)$. There are $\binom{k}{h-1}$ such sets. Since no two sets ever become equal, the number of sets remains constant. After processing $R(i)$, the collection of subsets consists of all the h element subsets of H which contain $R(i-1)$.

By our choice of starting configuration, step 1 just costs $\theta \cdot d(R(1), R(0))$. At step i (for $i \geq 2$), each of these algorithms either moves the truck at node $R(i)$ to $R(i-1)$ (if S contains $R(i)$), at cost $\theta \cdot d(R(i), R(i-1))$, or moves the truck at node $R(i-2)$ to $R(i)$ and then to $R(i-1)$ (if S does not contain $R(i)$), at cost $d(R(i-2), R(i)) + \theta \cdot d(R(i), R(i-1))$. Of the $\binom{k}{h-1}$ algorithms being run, $\binom{k-1}{h-1}$ of them (the ones which contain $R(i-2)$ but not contain either $R(i)$) incur the cost of $d(R(i-2), R(i)) + \theta \cdot d(R(i), R(i-1))$. The remaining $\binom{k-1}{h-2}$ of algorithms incur the cost of $\theta \cdot d(R(i), R(i-1))$. Thus, for step i, the total cost incurred by all of the algorithms is

$$\binom{k}{h-1} \cdot \theta \cdot d(R(i), R(i-1)) + \binom{k-1}{h-1} \cdot d(R(i-2), R(i)).$$

The total cost of running all of these algorithms up to and including R(t) is

$$\sum_{i=1}^{t} \binom{k}{h-1} \cdot \theta \cdot d(R(i), R(i-1)) + \sum_{i=2}^{t} \binom{k-1}{h-1} \cdot d(R(i-2), R(i)).$$

Thus the expected cost of one of these algorithms chosen at random is

$$C_{\text{EXP}}(R_t) = \theta \cdot \sum_{i=1}^{t} d(R(i), R(i-1)) + \frac{\binom{k-1}{h-1}}{\binom{k}{h-1}} \cdot \sum_{i=2}^{t} d(R(i-2), R(i)) \leq \frac{(\theta+2)k-2h+2}{k} \cdot \sum_{i=1}^{t-1} d(R(i), R(i+1)) + \frac{(\theta+1)k-h+1}{k} \cdot d(R(1), R(0)) - \frac{k-h+1}{k} \cdot d(R(t-1), R(t))$$

This inequality holds for the triangle inequality and expending of the binomial coefficients. Recall that the cost to A for the same steps was

$$C_{\text{A}}(R_t) = (1+\theta) \cdot \sum_{i=1}^{t-1} d(R(i+1), R(i)) + d(R(i+1), R(i)) + \theta \cdot d(R(1), R(0)),$$

Because the distances are symmetric, the two summations of the $C_{\text{EXP}}(R_t)$ and $C_{\text{A}}(R_t)$ are identical, except that both of the costs include some extra terms, which are bounded as a constant. Therefore, after some mathematical manipulation (e.g., let $t \to \infty$), we obtain

$$\frac{C_{\mathrm{A}}(R_t)}{C_{\mathrm{EPT}}(R_t)} \geq \frac{(\theta+1)\cdot k}{(\theta+2)\cdot k-2h+2}.$$

Finally, there must be some initial set whose performance is often no worse than the average of the costs. Let S be this set, and $A(S)$ be the algorithm starting from this set. Let R_i be an initial subsequence of R, for which $A(S)$ does no worse than average. □

Corollary 1. *For any symmetric k-truck problem, there is no c-competitive algorithm for* $c < (\theta+1)\cdot k/(\theta \cdot k+2)$.

Corollary 2. *For any symmetric k-taxi problem, there is no c-competitive algorithm for* $c < 2k/(k+2)$.

5 Competitve Ratios

5.1 Position Maintaining Strategy Solution

In [1], with the PMS, the case under which $\theta > (c+1)/(c-1)$ was studied, and a c-competitive algorithm was found to exist for the k-truck problem. In fact, we can get a somewhat better result for general cases.

Theorem 4. *For the on-line k-truck problem and a given graph G, if there is a c-competitive on-line algorithm for the k-server problem on G, then: (1) If* $\theta > (c+1)/(c-1)$*, then PMS is a c-competitivealgorithm; (2) If* $1 \leq \theta \leq (c+1)/(c-1)$*, then PMS is a* $(c/\theta + 1/\theta + 1)$*-competitive algorithm.*

Proof. For any $R = (r_1, ..., r_m)$, where $r_i = (a_i, b_i)$, considering the k-server problem's request sequence $\sigma = (a_1, ..., a_m)$, let A_σ be a c-competitive algorithm for the on-line k-server problem on graph G to satisfy the sequence. We design algorithm A as follows. For current service request $r_i = (a_i, b_i)$, first schedule a truck to a_i using algorithm A_σ, then complete the r_i with PMS. Thus total cost of A is

$$C_{\mathrm{A}}(R) = \sum_{i=1}^{m} C_{\mathrm{A}}(r_i) = \sum_{i=1}^{m} [C_{\mathrm{A}}(a_i) + (\theta+1)\cdot d(a_i,b_i)] =$$
$$C_{A_\sigma}(\sigma) + (1+1/\theta)\cdot \sum_{i=1}^{m} \theta \cdot d(a_i,b_i)$$

where θ is defined above and $\theta \geq 1$. From lemma 2 and algorithm A_σ, we have

$$C_{A_\sigma}(\sigma) \leq c\cdot C_{\mathrm{OPT}}(\sigma) + \beta \leq c\cdot [C_{\mathrm{OPT}}(R) - \sum_{i=1}^{m}(\theta+1)\cdot d(a_i,b_i)] + \beta$$

Then we get

$$C_{\mathrm{A}}(R) \leq c\cdot C_{\mathrm{OPT}}(R) + [1 + 1/\theta - c\cdot(\theta-1)/\theta]\cdot \sum_{i=1}^{m} \theta \cdot d(a_i,b_i)] + \beta$$

If $\theta > (c+1)/(c-1)$, we get $C_{\mathrm{A}}(R) \leq c\cdot C_{\mathrm{OPT}}(R) + \beta$; if $1 \leq \theta \leq (c+1)/(c-1)$, and with lemma 3, $C_{\mathrm{OPT}}(R) \geq \sum_{i=1}^{m} \theta \cdot d(a_i,b_i)$,we have $C_{\mathrm{A}}(R) \leq (c/\theta + 1/\theta + 1)\cdot C_{\mathrm{OPT}}(R) + \beta$, where c and β are some constants. □

Combining Theorem 4 and Lemma 1, the following corollary holds.

Corollary 3. *For the on-line k-truck problem on a given graph G, if* $\theta > (c+1)/(c-1)$*, holds, then there exists a* $(2k-1)$*-competitive algorithm; if* $1 \leq \theta \leq (c+1)/(c-1)$*, then there exists a* $(2k/\theta+1)$*-competitive algorithm.*

5.2 Comparison of Two Algorithms

In [1], an algorithm B, here we called it the PG, is given for the problem $P1$. The competitive ratio of algorithm B is $1+\lambda/\theta$, where $\lambda = d_{\max}/d_{\min}$, $d_{\max} = \max d(v_i, v_j)$, and $d_{\min} = \min d(v_i, v_j), i \neq j, v_i, v_j \in V$. We denote the PMS algorithm of subsection 5.1 by algorithm A. We may be confronted with the problem of choosing one algorithm from A' and B in different contexts. Respectively the competitive ratios of algorithms A and B are $c_{\mathrm{A}} = \begin{cases} 2k-1 & \text{if } \theta > (c+1)/(c-1) \\ 2k/\theta+1 & \text{if } 1 \leq \theta \leq (c+1)/(c-1) \end{cases}$ and $c_{\mathrm{B}} = 1+\lambda/\theta$. Letting $c_{\mathrm{A}} = c_{\mathrm{B}}$, we can get a k that makes the algorithm A and B equal as follows

$$k = \begin{cases} 1+\lambda/(2\theta) & \text{if } \theta > (c+1)/(c-1) \\ \lambda/2 & \text{if } 1 \leq \theta \leq (c+1)/(c-1) \end{cases}$$

Theorem 5. *For on-line k-truck problem P1, denoting the PMS and PG algorithms by A and B, respectively, at the aspect of the competitive ratio: if* $\theta > (c+1)/(c-1)$ *holds, if* $k \leq 1+\lambda/(2\theta)$ *then A is better than B, and contrarily if* $k > 1+\lambda/(2\theta)$ *then B is better than A; if* $1 \leq \theta \leq (c+1)/(c-1)$ *holds, if* $k \leq \lambda/2$ *then A is better than B, and contrarily if* $k > \lambda/2$ *then B is better than A.*

5.3 Partial-Greedy Algorithm on a Special Line

Let $G = (V, E)$ for the instance of an on-line k-truck problem consisting of a line of n vertices with $n-1$ edges whose lengths are equal to one. More formally, we have that $V = \{v_i | i = 1, ..., n\}$ and $E = \{v_i v_{i+1} | i = 1, ..., n-1\}$. All edge-weights are equal to one. It is natural to assume that no vertex has more than one truck (otherwise, we can get at this situation at most cost of $k \cdot (k+1)/2$). In addition, we assume that $n \geq k+2$ holds (otherwise the fourth case of the following algorithm does not exist).

Partial-Greedy Algorithm. *For the current request* $r_i = (a_i, b_i)$ *from the request sequence* $R = (r_1, ..., r_m)$*, schedule the k-truck problem* $P1$ *on the above special line with the following rules:*

(1) If there is a truck at a_i *and also one at* b_i*, then PG moves the truck at* a_i *to* b_i *complete the request, and at the same time PG moves the truck at* b_i *to* a_i *with an empty load. The cost of PG for the* r_i *is* $(1+\theta) \cdot d(a_i, b_i)$ *and at present no vertex has more than one truck.*

(2) If there is a truck at a_i *and no truck at* b_i*, then PG moves the truck at* a_i *to* b_i *to complete the request. The cost of PG for the* r_i *is* $\theta \cdot d(a_i, b_i)$*, and at present no vertex has more than one truck.*

(3) If there is no truck at a_i and there is a truck at b_i, then PG moves the truck at b_i to a_i first without a load, and after that moves it from a_i to b_i to complete the request. The cost of PG for the r_i is $(1+\theta)\cdot d(a_i, b_i)$ and at present no vertex has more than one truck.

(4) If there is no truck at a_i and b_i, then PG moves the truck which is the closest to a_i (suppose that the truck is located at c_i) with an empty load and then moves to b_i to complete the request. The cost of PG for the r_i is $d(c_i, a_i) + \theta \cdot d(a_i, b_i)$, and again no vertex has more than one truck.

Theorem 6. *PG is a $(1+(n-k)/\theta)$-competitive algorithm for the k-truck problem P1 on the above special line.*

Proof. For cases (1), (2) and (3), the cost of it PG is at most $(1+\theta)$ times the optimal cost for any request. For case (4), the extra cost is $d(c_i, a_i)$. Since c_i is the closest occupied vertex to a_i, we have $d(c_i, a_i) \leq (n-k)\cdot d(a_i, b_i)$. Let $C_{\mathrm{PG}}(R)$ denote the cost of algorithm *PG* for request sequence $R = (r_1, ..., r_m)$, then we have

$$C_{\mathrm{PG}}(R) = \sum_{i=1}^{m}\{max[d(b_i, a_i), d(c_i, a_i)] + \theta \cdot d(a_i, b_i)\} + \beta \leq \sum_{i=1}^{m}\{(n-k)\cdot d(a_i, b_i) + \theta \cdot d(a_i, b_i)\} + \beta = (1+(n-k)/\theta)\cdot \sum_{i=1}^{m} \theta \cdot d(a_i, b_i) + \beta \leq (1+(n-k)/\theta)\cdot C_{\mathrm{OPT}}(R)$$

where β is the cost for preconditioning the truck such that each vertex has at most one truck and it is bounded by a constant related with G. The last inequality holds for the lemma 3. □

Similar to subsection 5.2, combining the lemma 4 and the above theorem 6, we have the following theorem.

Theorem 7. *For on-line k-truck problem P1 on the special line, denoting the PMS and PG algorithms by A and B, respectively, at the aspect of the competitive ratio: if $\theta > (c+1)/(c-1)$ holds, if $k \leq (n+\theta)/(\theta+1)$ then A is better than B, and contrarily if $k > (n+\theta)/(\theta+1)$ then B is better than A; if $1 \leq \theta \leq (c+1)/(c-1)$ holds, if $k \leq (n-1)/2$ then A is better than B, and contrarily if $k > (n-1)/2$ then B is better than A.*

6 Deeper On-Line *k*-Truck Problem

We call the on-line k-truck problem studied in previous sections, the *Standard On-line k-truck problem* (STP). Here we will discuss another variant of it, the *Deeper On-line k-truck problem* (DTP). We formulate DTP as follows:
Given a metric space M, and k trucks which move among the points of M, each occupying one point of M, repeatedly, a request (a pair of points $x, y \in M$) appears. However, only the node x of request occurring is known when the information of the request is received, and the destination node y will not be known

until a truck has already been on the node of request occurring. To serve a request, an empty truck must first move to x and then move to y with goods from x. How to minimize the total cost of all trucks?

We easily know that the results of the competitive ratio of the PMS still hold for the DTP but these of the PG algorithm do not hold for the DTP.

Theorem 8. *For the DTP on a given graph G, if there is a c-competitive on-line algorithm for the k-server problem on G, then: (1) If* $\theta > (c+1)/(c-1)$, *then PMS is a c-competitive algorithm; (2) If* $1 \le \theta \le (c+1)/(c-1)$, *then PMS is a* $(c/\theta + 1/\theta + 1)$*-competitive algorithm.*

7 Concluding Remarks

Most of the results of this paper can be extended to the relevant cases of the k-taxi problem [8]. Although we get a lower bound of competitive ratio for the k-truck problem, the optimal lower bound of the competitive ratio for it is still open. Furthermore, whether there are some better on-line algorithms than PMS or PG needs further investigation.

References

1. W.M.Ma, Y.F.Xu, and K.L.Wang, On-line k-truck problem and its competitive algorithm. Journal of Global Optimization 21 (1): 15-25, September 2001.
2. S. Albers and S. Leonardi. Online algorithms. ACM Computing Surveys Vol.31. Issue 3 Sept. 1999.
3. D.D.Sleator, R.E.Tarjan, Amortized efficiency of list update and paging rules, Communication of the ACM, 28 (1985) 202-208.
4. A.Karlin, M.Manasse, L.Rudlph and D.D.Sleator. Competitive snoopy caching, Algorithmica, 3:79-119,1988.
5. M.S.Manasse, L.A.McGeoch, and D.D.Sleator, Competitive algorithms for on-line problems. In Proc. 20th Annual ACM Symp. on Theory of Computing, 322-33, 1988.
6. M.S.Manasse, L.A.McGeoch, and D.D.Sleator, Competitive algorithms for server problems, Journal of Algorithms,1990(11),208-230.
7. S.Ben-David, S.Borodin, R.M.Karp, G.Tardos, and A. Wigderson. On the power if randomization in on-line algorithms. In Proc. 22nd Annual ACM Symp. on Theory of Computing, 379-386, 1990.
8. Y.F.Xu, K.L.Wang, and B. Zhu, On the k-taxi problem, Information, Vol.2, No.4, 1999.
9. E.Koutsoupias, C.Papadimitriou, On the k-server conjecture, STOC.,507-511,1994.
10. M.Chrobak, L.Larmore, An optimal algorithm for the server problem on trees, SIAM Journal of Computing 20 (1991)144-148.
11. M.Chrobak, H.Karloff, T.Payne, S.Vishwanathan, New results on the server problem, SIAM Journal on Discrete Mathematics 4 (1991) 172-181.
12. R.Tarjan, Data Structures and Network Algorithms, SIAM, Philadelphia, 1983, 109-111.