Parallel Algorithms for Vehicle Routing Problems

K. Jeevan Madhu* Sanjeev Saxena
Computer Science & Engineering,
Indian Institute of Technology, Kanpur,
INDIA-208 016

Abstract

In a complete directed weighted graph there are jobs located at nodes of the graph. Job i has an associated processing time or handling time h_i, and the job must start within a prespecified time window $[r_i, d_i]$. A vehicle can move on the arcs of the graph, at unit speed, and that has to execute the jobs within their respective time windows. We consider three different problems on the CREW PRAM.
(1) Find the minimum cost routes between all pairs of nodes in a network. We give an $O(\log^3 n)$ time algorithm with $n^4/\log^2 n$ processors.
(2) Services all locations in minimum time. The general problem is $\mathcal{NP}$-complete but $O(n^2)$ time algorithms are known for a special case; for this case we obtain an $O(\log^3 n)$ time parallel algorithm using $n^4/\log^2 n$ processors and a linear time optimal parallel algorithm.
(3) Minimize the sum of waiting times at all locations. The general problem is $\mathcal{NP}$-complete but $O(n^2)$ time algorithm are known for a special case; for this case, we obtain an $O(\log^2 n)$ time algorithm with $n^3/\log n$ processors and also a linear time optimal parallel algorithm.

1. Introduction

Vehicle routing problems involve the navigation of one or more vehicles through a network of locations with each location being serviced. We follow the terminology of Gupta and Krishnamurti[6] extensively in this paper. The underlying network can be represented as a graph $G(V, E)$ where the set of nodes V are locations and the set of edges E are links between locations. Locations have associated handling times as well as time windows during which they are active. *Handling time* $h(v)$ will denote the time required for vehicle to service node v. The closed interval $[r(v), d(v)]$ is the *time window* of v here $r(v)$ is the *release time* and $d(v)$ is the *deadline time*; these are the earliest and latest times the node can be serviced. Further, we assume $0 \leq r(v) \leq d(v) < \infty$ for all nodes v and that all weights are positive and integral. The vehicle can pass through the node on the way to some other node on the route or actually handle the node during its time window; route is a sequence of locations. The arcs connecting locations have time costs associated with them; edge (u, v) has a weight $t(u, v)$ the *travel time*, it is the time required for the vehicle to traverse this edge. The length of route is the sum of handling times of each node on the route and the travel times along each edge on the route; a vehicle arriving at a node before release time must wait till the release time; these wait times are also added to routes length. If the vehicle arrives at a node after the deadline then route is not feasible. A route is *feasible* if its cost is defined; otherwise it is *infeasible*. The cost of route at a particular starting time is defined if the vehicle can traverse through the nodes (in the route) with every node being serviced before its deadline time. The cost of route is *optimal* if there is no route with smaller cost. It is a *covering* if every node of G appears in the route [6].

In this paper we develop parallel algorithms for All pair routing, Single vehicle routing problem and Travelling repairman problem. In the first problem, the all pair routing problem, we compute the shortest route between all pairs of locations. Our algorithm runs in time $O(\log^3 n)$ time on a CREW PRAM using $n^4/\log^2 n$ processors. The best known previous algorithm given by Gupta and Krishnamurti [6] runs in $O(\log^3 n)$ time using n^4 processors on a CREW PRAM. Formally, the *all pairs routing problem* is to determine for each pair of nodes u and v in G, the optimal route between u and v for each possible starting time of the vehicle starting from node u and servicing all the nodes in the route within their respective time windows.

A network is said to be a line if all locations in the network lie on a line i.e., if nodes can be ordered as $v_1, v_2, \ldots, v_n$ such that there is an edge from v_i to v_{i+1} for $1 \leq i < n$ and there are no other edges [6].

In the second problem, the single vehicle routing problem, we compute the shortest route involving all locations starting from a particular location called Depot. Here the

*Now with Motorola India Electronics Ltd, Bangalore - 560 042

network is a line and all the locations will have either release time or deadline time. Our algorithm takes $O(\log^3 n)$ time using $n^4/\log^2 n$ processors on a CREW PRAM. The best known previous algorithms given by Gupta and Krishnamurti runs in $O(\log^3 n)$ time with n^8 processors on a CREW PRAM[1]. Formally, the *single vehicle routing problem with time window constraints*(SVRPTW) on G finds the optimal covering route of G that starts at depot($S(G)$) at time 0 and ends at $T(G)$, where $T(G)$ can be any node. SVRP is $\mathcal{NP}$-hard even for $h(v), r(v) = 0$ and $d(v) = \infty$ for every $v \in V(G)$ [6]. In this paper we only consider SVRPTW problem for which $h(v) = 0$ with underlying network is a line. Further, we discuss only SVRPTW-line problems having only release times (i.e. $d(v) = \infty \; \forall v$) and SVRPTW-line problems having deadlines (i.e. $r(v) = 0; \forall v$) These are called rSVRPTW-line and dSVRPTW-line respectively.

The third problem, the travelling repairman problem, is similar to the second but, here we try to find a route involving all locations such that sum of waiting times at all locations is to be minimum. In this problem, the network is a line and locations don't have any time windows associated with them. We show that this problem can be reduced to the shortest path algorithm for layered graph and give an $O(\log^2 n)$ time parallel algorithm with $n^3/\log n$ processors on a CREW PRAM. Formally, the *travelling repairman problem*(TRP) on G is to find the covering route of G that starts at depot($S(G)$) at time 0 and ends at $T(G)$ having sum of waiting times of nodes minimum. The waiting time of node is the difference between release time and the actual time at which vehicle services the node. In this paper we discuss TRP-line problem for which all nodes in the network have zero handling time and there is no time window associated with them.

These problems have applications in service sector (garbage collection and postal delivery), commercial sector (transportation of goods through road and rail), industrial sector (material handling systems in manufacturing) and experimental applications (like automatic vehicle routing and robot arm movement)[6]. Vehicle navigation problem arises in various situations (see [6] for details).

The Vehicle routing problem (VRP) involves the design of a set of minimum cost routes, originating and terminating at a central depot, for a fleet of vehicles which services a set of customers with known demands [12]. Each customer is serviced exactly once and furthermore, all the customers must be assigned to vehicles such that the vehicle capacities are not exceeded. Bodin *et. al.* [3] provide a comprehensive survey of the VRP and its variations which also describes the many practical occurrences of these problems.

[1]Gupta and Krishnamurti [6] they have wrongly stated that their algorithms for the All pair routing problem and the Single vehicle routing problems run in $O(\log^2 n)$ time. They assume that composition of two tables or finding minimum of two tables requires $O(1)$ time. But, these operations require $O(\log n)$ time.

In VRP with time windows (VRPTW), the above issues have to be dealt with under the added complexity of time windows. Time windows specify the deadlines and earliest service times of each customer. Servicing of a customer can begin within the time window defined. These time windows can be soft or hard. In case of hard windows the service has to start within the time window. However, in case of soft windows the service time can violate time window with some penalty. Solomon *et. al.* [12] provide a survey of time constrained routing and scheduling problems.

SVRP is a case of VRP when there is only one vehicle and it is $\mathcal{NP}$-complete even if the inter point distance metric is restricted to Euclidian [8]. Introducing time constraints on the problem (SVRPTW) can only make it harder [10]. There are some polynomial time algorithms when we restrict the underlying network to a straight line and there are no capacitity constraints for the vehichle (see table below for details).

	Zero processing times	General processing times
No release times or deadlines	Trivial	Trivial
Release times only	$O(n^2)$[9]	$\mathcal{NP}$-complete[13]
Deadlines only	$O(n^2)$[13]	?
General time windows	Strongly $\mathcal{NP}$-complete[13]	Strongly $\mathcal{NP}$-complete[5]

The travelling repairman problem(TRP) is a variation of the well known travelling salesman problem, in which instead of minimizing the total completion time for salesman tour, one tries to minimize the sum of the waiting times of locations or customers [1]. TRP captures the waiting costs of a service system from the customers point of view and it can be used to model numerous types of service systems. While the general problem is $\mathcal{NP}$-complete [1], some progress has been made when the network is straight line. Afrati *et. al.* [1] have given a $O(n^2)$ time algorithm when the handling time of location is zero and there are no time bounds associated with locations[13]. We call this as TRP-line problem (see table below for summary).

	Zero processing times	General processing times
No release times or deadlines	$O(n^2)$[1]	?
Release times only	?	strongly $\mathcal{NP}$-complete[7]
Deadlines only	$O(n^2)$[13]	$\mathcal{NP}$-complete[13]
General time windows	Strongly $\mathcal{NP}$-complete[13]	Strongly $\mathcal{NP}$-complete[5]

In Section 2 we discuss the all pairs routing problem. In Section 3 we describe algorithms for rSVRPTW-line and dSVRPTW-line problems. In Section 4 we give an algorithm for TRP-line problem.

2. The all pairs routing problem

The *handling time* term can be eliminated by simply adding $h(v)$ of node v to the travel time on each edge out of v and assume that the networks do not have handling times [6]. In section 2.1 we describe a sequential algorithm for the problem of finding a shortest route from a node S to a node T using Fibonacci heap data structure given by Fredman and Tarjan [4]. This in-fact finds the shortest route from node S to all other nodes in G. This is called S-T routing problem [6]. The algorithm described takes $O(m+n\log n)$ time, where m is the number of edges and n is the number of nodes in the network. This is an improvement over the $O(n^2)$ time algorithm given by Gupta and Krishnamurti [6]. In Section 2.2 we give an parallel algorithm for all pair shortest route problem.

2.1. Sequential algorithm for all pair routing problem

Let $G(V,E)$ be a network with time windows specified at each node $u \in V$ and travel times specified on edge $(u,v) \in E$. Let n be number of nodes and m be the number of edges. Our algorithm proceeds in the manner analogous to that of Fibonacci heap (F-heap) implementation of Dijkstra's algorithm.

1. Un-label all vertices $v \in G$.
2. Cost function T is defined as follows:
The cost function for S is $T(S) = 0$ and for all other vertices u is $T(u) = \infty$, $prev(u) = S$
3. for all neighbors v of S, we update T as follows: $T(v) = \max\{r(v), T(S) + t(S,v)\}$, and if $T(S) + t(S,v) > d(v)$ then $T(v) = \infty$
4. while($\exists v \in V$ which is un-labeled)

(a) Select an un-labeled vertex v, having minimum $T(v)$, Mark it.
(b) For each edge (v,w),
$T(w) = \min\{T(w), \max\{r(w), T(v) + T(v,w)\}\}$. If $T(v) + T(v,w) > d(w)$ then $T(w) = \infty$.

Lemma 1 *Sequential algorithm for S-T routing problem takes $O(m + n\log n)$ time*

Proof: If all un-labeled vertices are kept in F-heap then step 4(a) requires a delete minimum operation, and step 4(b) require a decrease key operation; in that case we also make $prev(w) = v$. Moreover, for every edge (u,v), step 4(b) is performed exactly once. Thus step 4(b) will be performed exactly m times. Moreover, step 4(a) will be performed exactly once for each vertex. Thus, we have n minimum deletion operations and m decrease key operations on F-heap data structure. A delete minimum operation takes $O(\log n)$ amortised time[2] and a decrease key operation requires $O(1)$ amortised time [4]. Thus the total time taken by our algorithm is $O(m + n\log n)$. ∎

Theorem 1 *Sequential algorithm for all pair routing problem takes $O(nm + n^2\log n)$ time*

Proof: The proof of the theorem follows from the fact that all pair routing problem can be solved by performing S-T routing problem once for each vertex v in G i.e., n times. ∎

2.2. Parallel Algorithm for the all pair shortest routing problem

Gupta and Krishnamurti [6] associate a function $\mathcal{T}_{u,v}$ with every pair (u,v) of vertices; $\mathcal{T}_{u,v}$ is the cost of an optimal u-v route starting at time t. $\mathcal{P}_{u,v}(t)$ will denote optimal cost route from u to v when the vehicle starts from u at time t. The cost vector of the optimal route from u to v is a function $\mathcal{T}_{u,v} : N \to N$ such that $\mathcal{T}_{u,v}(t)$ is the cost of $\mathcal{P}_{u,v}(t)$. The cost vector is monotonic in t since the vehicle must wait at nodes whose time windows are not yet open.

We need some more notation [6]. If $\mathcal{P}$ is an optimal route in G in time interval $\mathcal{I} = [a,b]$ and if $W > 0$ is the sum of the waiting times when a vehicle follows $\mathcal{P}$ starting from u at time a and if $b > a+W$, then $\mathcal{P}$ makes a transition at time $a + W$: before this time, a vehicle following $\mathcal{P}$ is forced to wait at one or more nodes but after this time there are no waits. $\mathcal{P}$ is called *transition route* and $a + W$ its *transition time*. In the transition route $\mathcal{P}$, there is at least one node w such that if the vehicle starts at any time $t < a + W$, it must wait at w; w is called *bottle-neck* node of $\mathcal{P}$ and $bn(\mathcal{P})$

[2] Amortised complexity of an operation is $O(g(n))$ if for a sequence k (sufficiently large, $k \geq n$) operations, the total time required by these operations is $O(kg(n))$ [2]

will denote the set of such nodes [6]. Suppose $\mathcal{P}_1$ and $\mathcal{P}_2$ are two transition routes from u to v that are optimal during time intervals $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively. If $\mathcal{I}_1$ occurs strictly before $\mathcal{I}_2$ then $bn(\mathcal{P}_1) \cap bn(\mathcal{P}_2) = \emptyset$ [6].

The cost vector $\mathcal{T}_{u,v}$ is represented by smallest table satisfying certain properties [6]. Each row r of the table consists of $interval(r)$ and $cost(r)$, where $interval(r)$ is of the form $[a, b]$ or $[a, \infty)$. If $r_1 < r_2$ then all elements of $interval(r_1)$ precede all elements of $interval(r_2)$. Further, the union of $interval(r)$ over all rows r is the interval$[0, \infty)$; $cost(r)$ is a function of the form α or $\tau + \alpha$ where α is a constant and τ a variable. If $cost(r) = \alpha$ (respectively $\tau + \alpha$) then for all $t \in interval(r)$, $\mathcal{P}_{u,v}(t)$ has cost α (respectively $t + \alpha$). There is one route $\mathcal{P}_r$ associated with each r such that $\mathcal{P}_r$ is an optimal u-v route when the vehicle starts form u at any time $t \in interval(r)$.

The *complexity* of a cost vector $\mathcal{T}_{u,v}$, $complex(\mathcal{T}_{u,v})$, is the minimum number of rows required to represent it as a table $complex(\mathcal{T}_{u,v}) \in O(n)$ [6]. In fact, for any two nodes u and v, *complex*$(\mathcal{T}_{u,v}) \leq 4n$ [6].

High level description of parallel algorithm of Gupta and Krishnamurti [6] is:
1. For every $(u, v) \in E(G)$ in parallel compute $\mathcal{T}_{u,v}$
2. For every non-edge (u, v), let $\mathcal{T}_{u,v}$ be one row table with interval $[0, \infty)$ and cost ∞.
3. Loop $\log n$ rounds
In parallel for every pair $(u, v) \in E(G)$ do
(a) $A = \{w \mid w \in V(G); (u, w), (w, v) \in E(G)\}$
(b) $\forall w \in A$, compose $\mathcal{T}_{u,w}$, $\mathcal{T}_{w,v}$ to form $\mathcal{T}^w_{u,v}$
(c) Let $\mathcal{T}_{u,v} = \min\{\mathcal{T}_{u,v}, \min\{\mathcal{T}^w_{u,v} \mid w \in A\}\}$
(d) If (u, v) is not in $E(G)$ then add edge (u, v) to $E(G)$

Operator for composing $\mathcal{T}_1$ and $\mathcal{T}_2$ can be viewed graphically as follows: For each row r in $\mathcal{T}_1$, calculate the corresponding cost at both ends of the time interval for that row. This cost interval computed becomes the time interval of $\mathcal{T}_2$. Then copy the graph for $\mathcal{T}_2$ to get the graph for $\mathcal{T}_{12}$ for that interval. Putting it more formally, we have the following algorithm for composing $\mathcal{T}_1$ and $\mathcal{T}_2$.
For each row r in $\mathcal{T}_1$, find rows s in $\mathcal{T}_2$ such that cost interval of r overlaps with time interval of s. For each pair r in $\mathcal{T}_1$, and s in $\mathcal{T}_2$ (cost(r) overlaps with interval(s)) first compute the cost function by substituting the cost(r) in cost(s) and then create a row in $\mathcal{T}_{12}$ with newly computed cost function. Finally Merge the rows in $\mathcal{T}_{12}$ which have identical cost function and route; Algorithm for finding the overlapped intervals is:
For each row $r \in \mathcal{T}_1$ calculate the corresponding cost at both ends of the time interval. Now we can see that table is of four columns with first two columns representing the time interval (say $\mathcal{T}_1(1), \mathcal{T}_1(2)$) and last two representing the cost interval (say $\mathcal{T}_1(3), \mathcal{T}_1(4)$). Let $\mathcal{T}_2(1), \mathcal{T}_2(2)$ are the columns representing lower and upper end of time intervals of $\mathcal{T}_2$. Merge arrays $\mathcal{T}_2(1)$ and $\mathcal{T}_1(3)$ and call it $\mathcal{T}_l$. Form an array T_l where, if $\mathcal{T}_l(i) \in \mathcal{T}_2$ then $T_l(i) =$ rank of $\mathcal{T}_l(i)$ in $\mathcal{T}_2$ else $T_l(i) = 0$. For every $T_l(i)$ such that $\mathcal{T}(i) \in \mathcal{T}_1$ $(T_l(i) = 0)$, find the the leftmost time interval of $\mathcal{T}_2$ in which the cost interval of $\mathcal{T}_1$ falls (find the nearest larger on left in T_l). Say it r_l. In other words, r_l gives the index of the first row in $\mathcal{T}_2$, which overlaps with cost interval of $\mathcal{T}_1$. Merge arrays $\mathcal{T}_1(4)$ and $\mathcal{T}_2(2)$ and call it $\mathcal{T}_r$. Form an array T_r where, if $\mathcal{T}_r(i) \in \mathcal{T}_2$ then $T_r(i) =$ rank of $\mathcal{T}_r(i)$ in $\mathcal{T}_2$ else $T_r(i) = 0$. For every $T_r(i)$ where $\mathcal{T}(i) \in \mathcal{T}_1$ $(T_r(i) = 0)$ find the rightmost time interval of $\mathcal{T}_2$ in which the the cost interval of $\mathcal{T}_1$ falls (find nearest larger on right in T_l). Say it r_r. In other words, r_r gives the index of the last row in $\mathcal{T}_2$, which overlaps with cost interval of $\mathcal{T}_1$. For every r in $\mathcal{T}_1$, find the value $r_o = r_r - r_l + 1$. It represents the number of rows in $\mathcal{T}_2$, whose time interval overlap with cost interval of r. Since the cost functions are monotonic and time intervals in tables do not overlap, the total size of the composed table is sum of tables that are being composed. If we do a prefix sum of r_o for every row (excluding its own r_o from prefix sum), to get the indices of the each row in the final table constructed.

Lemma 2 [6] *$\mathcal{T}^w_{u,v}$ can be described by a table using at most $complex(\mathcal{T}_{u,w}) + complex(\mathcal{T}_{w,v})$ rows.*

$\mathcal{T}^w_{u,v}$ is a cost vector in a sub graph of G, hence $complex(\mathcal{T}^w_{u,v})$ is at most $4n$; thus some rows of $\mathcal{T}^w_{u,v}$ can be merged.

Lemma 3 *The composition operation $\{\forall w \in A \mid \mathcal{T}^w_{u,v} = \mathcal{T}_{u,w} + \mathcal{T}_{w,v}$ can be performed in $O(\log n)$ time using $n^2/\log n$ processors on a CREW PRAM.*

Proof: Algorithm for composition of two tables is based on finding prefix sum, merging two arrays and finding nearest larger in an array of length $O(n)$ all these operations take $O(\log n)$ time using $n/\log n$ processors on a CREW PRAM Since we compose n pairs of tables, we need $n^2/\log n$ processors. ∎

Problem of finding minimum of $\mathcal{T}_1$ and $\mathcal{T}_2$ is finding the time intervals of $\mathcal{T}_1$ which overlaps with a time interval of $\mathcal{T}_2$ for every row and then finding the minimum cost of the intervals overlapped. Putting it more formally we have the following algorithm for finding minimum of two tables $\mathcal{T}_1$ and $\mathcal{T}_2$.
For each row r in $\mathcal{T}_1$, find rows s in $\mathcal{T}_2$ such that time interval of r overlaps with time interval of s. For each pair r in $\mathcal{T}_1$, and s in $\mathcal{T}_2$ (such that interval(r) overlaps with interval(s)) first compute the cost function by taking minimum of cost(r), cost(s) and then create a row in $\mathcal{T}_{12}$ with newly computed cost function. Merge the rows in $\mathcal{T}_{12}$ which have identical cost function and route. Overlapped intervals are found as before.

Lemma 4 *The minimization operation* $\mathcal{T}_{u,v} = \min\{\mathcal{T}_{u,v}, \min\{\mathcal{T}^w_{u,v} \mid w \in A\}\}$ *can be performed in* $O(\log^2 n)$ *time using* $n^2/\log^2 n$ *processors on a CREW PRAM.*

Proof: Algorithm for finding minimum of table $\mathcal{T}_{uw}$ and table $\mathcal{T}_{wv}$, is similar to that for composing two tables. ∎

Lemma 5 [6] *After first* k *iterations of Step 3 the correct value of* $\mathcal{T}_{u,v}$ *has been computed for every pair of nodes* u *and* v *when only routes of length at most* 2^k *are taken into account*

Theorem 2 *All pair* S-T *routing problem can be solved in* $O(\log^3 n)$ *time using* $n^4/\log^2 n$ *processors on a CREW PRAM.*

Proof: The correctness follows from Lemma 5. Steps 1 and 2 take $O(1)$ time using n^2 processors. In Step 3(a), we check all nodes w to see if they are candidates for A; this takes $O(1)$ time with n processors. Step 3(b) takes $O(\log n)$ time with $n^2/\log n$ processors by Lemma 3, or $O(\log^2 n)$ time with $n^2/\log^2 n$ processors (Since CREW PRAM is self-simulating). Step 3(c) takes $O(\log^2 n)$ time with $n^2/\log^2 n$ processors by Lemma 4. Finally Step 3(d) takes $O(1)$ time using one processor. Since we have to look at all pairs u and v in parallel, we need $n^4/\log^2 n$ processors. ∎

3. Parallel algorithms for dSVRPTW-line and rSVRPTW-line Problems

Tsitsiklis [13] and Psaraftis *et. al.* [9] have used dynamic programming technique to obtain quadratic algorithms for dSVRPTW-line and rSVRPTW-line problems respectively. Our algorithms for these problems make use of the parallel algorithm for the shortest path problem for layered directed acyclic graph (LDAG) combined with the dynamic programming formulation of Tsitsiklis and Psaraftis *et. al.* As parallel algorithm for rSVRPTW-line is similar, it is not discussed any further. We can fix any instance G of dSVRPTW-line to be ordered in the form $a_m, a_{m-1}, \ldots, a_1, D, b_1, \ldots, b_p$. we assume that nodes a_0 and b_0, both refer to same node, the central depot D from where the vehicle originates and terminates. Cost of an edge is time taken for the vehicle to travel between the two locations [6]. If $S = v_1, \ldots, v_{m+p+1}$ is an optimal feasible route that covers G, then S is *uniform* if for any $k \leq m+p+1$, $v_1, \ldots, v_k$ is an optimal feasible route that covers the subgraph of G induced by $a_i, a_{i-1}, \ldots, a_1, D, b_1, \ldots, b_j$ for some i and j such that $k = i + j + 1$. If there is a feasible route then there is an optimal uniform feasible route [6, 13].

If $v_1, \ldots, v_{m+p+1}$ is a uniform route in G, then the subsequence $v_1, \ldots, v_k$ can be represented by the ordered pair (i, j) where a_i and b_j both appear in this subsequence and $i + j + 1 = k$; subsequence $v_1, \ldots, v_k, v_{k+1}$ is represented by either $(i+1, j)$ or $(i, j+1)$ depending on whether v_{k+1} is a_{i+1} or b_{j+1}.

In general, for $0 \leq i \leq m$ and $0 \leq j \leq p$, let $\mathcal{C}(i, j; L)$ (respectively $\mathcal{C}(i, j; R)$) be the cost of uniform route on network induced by the subgraph $a_i, \ldots, a_1, D, b_1, \ldots, b_j$ in which a_i is the last element of the sequence (b_j is the last element respectively). Then, $\mathcal{C}(0, 0; L) = \mathcal{C}(0, 0; R) = 0$ and for $i, j > 0$, $\mathcal{C}(i, j; L) = \min\{\mathcal{C}(i-1, j; L) + t(a_{i-1}, a_i), \mathcal{C}(i-1, j; R) + t(b_j, a_i)\}$ and $\mathcal{C}(i, j; R) = \min\{\mathcal{C}(i, j-1; R) + t(b_{j-1}, b_j), \mathcal{C}(i, j-1; L) + t(a_i, b_j)\}$. Gupta and Krishnamurti [6] construct a layered directed acyclic graph (LDAG) called configuration network with nodes representing uniform routes in G and edges representing extensions of one route to another by the addition of one new node. We can then use shortest path algorithms for LDAG (see Section 3.1) to find the shortest route in configuration network.

The vertices in configuration network are $\mathcal{V} = \{(i, j, R), (i, j, L) : 1 \leq i \leq m, 1 \leq j \leq p\} \cup \{(0, 0, D)\} \cup \{(m+1, p+1, D)\}$ with source as $S(\mathcal{G}) = (0, 0, D)$ and sink as $T(\mathcal{G}) = (m+1, p+1, D)$ respectively. All the nodes have zero handling time, the deadline of $(0, 0, D)$ is 0 and the deadline of $(m+1, p+1)$ is ∞, deadlines of (i, j, L) is $d(a_i)$ and of (i, j, R) is $d(b_j)$. There are edges from (m, p, L) and (m, p, R) to $(m+1, p+1, D)$ with zero travel time and from $(0, 0, D)$ to $(1, 0; L)$ and $(0, 1, R)$ with travel time as $t(a_1, D)$ and $t(D, b_1)$ respectively. Finally, for $1 \leq i \leq m$ and $1 \leq j \leq p$ there are edges from (i, j, L) to $(i+1, j, L)$ with travel time $t(a_i, a_{i+1})$; from (i, j, L) to $(i, j+1, R)$ with travel time $t(a_i, b_{j+1})$; from (i, j, R) to $(i, j+1, R)$ with travel time $t(b_j, b_{j+1})$; and from (i, j, R) to $(i+1, j, L)$ with travel time $t(b_j, a_{i+1})$[6]. A cost vector or cost table $\mathcal{T}_{u,v}$ is associated with each $(u, v) \in \mathcal{E}$. The node (i, j, L) denotes the optimal route covering $[a_i, b_j]$ with a_i being the last node visited. Similarly the node (i, j, R) denotes the optimal route covering $[a_i, b_j]$ with b_j being the last node visited. Node $(0, 0, D)$ and $(m+1, p+1, D)$ denote the null route and optimal route respectively [6].

There are $n+1$ layers in the configuration network $\mathcal{G}$, where $n = m + p + 1$; $(0, 0, D)$ will be in first layer and $(m+1, p+1)$ in the last layer; (i, j, L) and (i, j, R) will be in $(i+j+1)^{th}$ layer. If $b = \min(m, p)$, then the maximum number of nodes in any layer will be $2b+1$. The number of nodes in layer k will be

$$\begin{cases} 2(i-1) & 2 \leq i \leq b+1 \\ 2b+1 & b+2 \leq i \leq n-b \\ 2(n-i+1) & n-b+1 \leq i \leq n \end{cases}$$

The first layer will have only the source node and the last layer will have only sink node.

We are required to find the shortest path from source to sink.

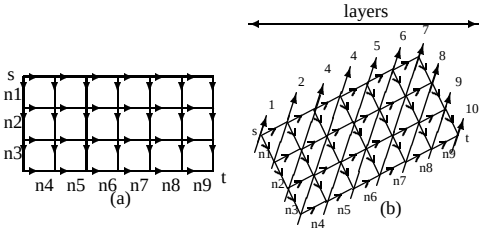

Figure 1. An example Layered graph

.

Lemma 6 [6] *For $\mathcal{G}$ the configuration network of a line graph G, the optimal $S(\mathcal{G})$-$T(\mathcal{G})$ route corresponds to an optimal uniform covering route of G.*

3.1. Shortest path algorithm for Layered Directed Acyclic Graph

An n Layered directed acyclic graph (LDAG) is a graph with vertices lying on layers; the edges will be from nodes at layer i to $i+1$ only. A simple example of a layered graph is the grid graph as shown in Figure 3.1(a).

The LDAG shown in Figure 3.1(a) can be re-drawn as in Figure 3.1(b). The graph has l rows, $(n-l)$ columns and $n-1$ layers and maximum number of nodes in any layer is $b = min(l, n-l)$. Edges from nodes at layer i lead to nodes at layer $i+1$ only. If adjacency list is used for storing the edges, the edges to next level can be stored in an array of length $2b$. An element (i, j) will be in layer $(i+j)$ and it is in j^{th} position in array if $1 \leq i+j \leq l$ and in position $l-i$, otherwise.

For finding the shortest path between s and t the vertices in alternate layers will be removed and the number of levels in each iteration reduces by half. Hence, the algorithm will take $O(\log n)$ iterations. If there is an edge from nodes 'x' at level $i-1$ to node 'y' at level i, and if there is another edge from node 'y' to node 'z' at level $i+1$, then the two edges can be replaced by a single edge from node 'x' to node 'z' with new weight as sum of the weights of original edges. We can find the shortest path in $O(\log n * \log b)$ time, with $O(nb^2/\log b)$ processors on a CREW PRAM and in $O(\log n * \log\log b)$ time with $O(nb^2/\log\log b)$ processors on a common-CRCW PRAM [11].

3.2. Parallel algorithms for dSVRPTW-line problem

An $O(n)$ time Optimal parallel algorithm when the vehicle starts at time 0 can be obtained by first constructing the LDAG for the given problem instance and then for $n+1$ iterations, serially in turn removing all nodes in the second layer (the nodes adjacent to source).

For removing nodes at second layer, we allocate one processor to every node in third layer and find the shortest path from source node to that node. Since, the in-degree of any node in LDAG is 2, the number of paths from source node to any node in third layer is 2. So, we can find the shortest path from source node to any node in third layer in $O(1)$ time. Since, there can be at most $n+1$ nodes in third layer in any iteration (the maximum number of nodes in any layer before first iteration is $n+1$), we need n processors in each iteration. Since, there are $n+1$ iterations it we will take $O(n)$ time. Thus, we can find an optimal schedule for coalescing operations when number of jobs is two in $O(n)$ time with $O(n^2)$ work.

We next describe an $O(\log^2 n)$ time Parallel algorithm. This algorithm computes the optimal path for all vehicle starting times in interval $[0, \infty)$. The algorithm is:
1. Construct the LDAG for the given problem instance
2. for t = 0 to $\log n + 1$ do /* $\log n$ iterations */
for i = 2 to $(n+1)/2^t - 1$ pardo
    if i is even then remove nodes at layer i;

The algorithm for removing all nodes at layer i is given below. Here, d is the maximum in-degree (or out-degree) of any node in the current iteration. Clearly, $d \leq n+1$.
for each node r of layer $i+1$ do /* at most $2b+1$ */
for each node v at layer i incident from r do /* at most d */
for each node p of layer $i-1$ incident from v do /* at most d */
parbegin
  (a) $\mathcal{T}^v_{r,p} = compose(\mathcal{T}_{r,v}, \mathcal{T}_{v,p})$
  (b) $\mathcal{T}_{r,p} = \min_v(\mathcal{T}^v_{r,p})$
parend;

Lemma 7 *The maximum degree of any node before k^{th} iteration will be* $\min(2^k, 2b+1)$

Proof: Any node in the graph has degree less than or equal to the degree of source node. This is because, the source node satisfies the same constraints as any other nodes in the graph. So, it is enough if we prove that the degree of source as $\min(2^k, 2b+1)$ in k^{th} iteration. It is evident from algorithm that after every iteration the number of layers decreases by a factor of 2. After every iteration layer i will become layer $(i+1)/2$ (if i is odd). Therefore, before k^{th} iteration the layer adjacent to first layer (second layer) will be same as the layer $2^{k-1}+1$ before the first iteration. (This can be proved by simple induction). The number of nodes layer i before first iteration is $2(i-1)$. Therefore, the number nodes in second layer after k iterations will be $2(2^{k-1}) = 2^k$. It is evident from description of our graph that every node is reachable from the source (this implies that every node in the second layer is connected to source) and the maximum number of nodes in any layer is at most

$2b+1$ so, the degree of a node before any iteration can not exceed $2n+1$. Therefore, the degree of source node is $\min(2^k, 2b+1)$ in k^{th} iteration. Thus, the result follows. ∎

Lemma 8 *For removing nodes at layer i in k^{th} iteration, our algorithm on a CREW PRAM requires $O\left(\log(\min(2b+1,2^k)*(\log n))\right)$ time using $(2b+1)*\left(\frac{\min(2b+1,2^k)^2*n}{\log\min(2b+1,2^k)*\log n}\right)$ processors.*

Proof: For removing nodes at layer i we need to find $\mathcal{T}_{r,p}$ where, $r \in$ layer $i-1$ and $p \in$ layer $i+1$. For finding $\mathcal{T}_{r,p}$ we need to find minimum of $\mathcal{T}^v_{r,p}$ where, $v \in$ layer i and r is incident on v, v incident on p. If d is the degree of r then we need to compose d tables and find minimum among these d tables. By Lemma 3 and 4, we can compose two tables or find minimum of two tables in $O(\log n)$ time with $n/\log n$ processors. We can compose d pairs of tables into d tables in $O(\log d * \log n)$ time with $d*n/(\log d * \log n)$ processors. We can also find minimum of d tables in $O(\log d * \log n)$ time with $d*n/(\log d * \log n)$ processors. Therefore, we can find $\mathcal{T}_{r,p}$ in $O(\log d * \log n)$ time with $d*n/(\log d * \log d)$ processors on a CREW PRAM. Since, there can be d nodes in layer $i+1$ associated with v and at most $(2b+1)$ nodes in layer $i-1$ the number of processors for removing nodes at layer i is $(2b+1)*d*(d/\log d)*(n/\log n)$. We have by Lemma 7 that the degree of any node before k^{th} iteration is $\min(2b+1,2^k)$. Therefore, we need $O(2b+1)*\left((\min(2b+1,2^k)^2/\log\min(2b+1,2^k))*(n/\log n)\right)$ processors and $O\left(\log(\min(2b+1,2^k)*\log n)\right)$ time for k^{th} iteration. ∎

Theorem 3 *There is a CREW PRAM algorithm for dSVRPTW-line that takes $O(\log^2 n * \log b)$ time using $n^2b^2/\log b * \log n$ processors.*

Proof: The initial number of layers in the graph is $n+1$. In every iteration we will be removing half of layers present. Therefore, there will be $(n+1)/2^k$ layers in k^{th} iteration. Combining this with Lemma 8, If in k^{th} iteration we use $\left(\frac{n+1}{2^k}\right)*(2b+1)*\left(\frac{\min(2b+1,2^k)^2*n}{\log\min(2b+1,2^k)*\log n}\right)$ processors, time will be $O\left(\log\min(2b+1,2^k)\right)$. These expressions will be maximum when $2^k \geq 2b+1$ or when $k \geq \log(2b+1)$. Therefore, the maximum number of processors in any iteration is $n^2b^2/\log b * \log n$. Since each iteration takes at most $O(\log n * \log b)$ time and there are $\log n + 1$ iterations, the time required for our algorithm is $O(\log^2 n * \log b)$. Thus the result follows. ∎

4. Travelling Repairman Problem

Afrati *et. al.* [1] have an $O(n^2)$ algorithm for special case of travelling repairman problem when handling time is zero and all the locations are in a line without having time windows associated with them. (see Tsitsiklis [13]) We first describe a simple $O(n^2)$ time dynamic programming algorithm for this problem.

We can fix any instance G of TRP-line to be ordered in the form $a_m, a_{m-1}, \ldots, a_1, D, b_1, \ldots, b_p$. We assume that nodes a_0 and b_0, both refer to same node, the central depot D from where the vehicle originates and terminates. Since all locations are on a line there are edges between two consecutive locations only. Cost of an edge is time taken for the vehicle to travel between the two locations. If $S = v_1, \ldots, v_{m+p+1}$ is an optimal feasible route that covers G, then S is *uniform* if for any $k \leq m+p+1, v_1, \ldots, v_k$ is an optimal feasible route that covers the subgraph of G induced by $a_i, a_{i-1}, \ldots, a_1, D, b_1, \ldots, b_j$ for some i and j such that $k = i+j+1$.

If $v_1, \ldots, v_{m+p+1}$ is a uniform route in G, then the subsequence $v_1, \ldots, v_k$ can be represented by the ordered pair (i,j) where a_i and b_j both appear in this subsequence and $i+j+1=k$; subsequence $v_1, \ldots, v_k, v_{k+1}$ is represented by either $(i+1, j)$ or $(i, j+1)$ depending on whether v_{k+1} is a_{i+1} or b_{j+1}.

In general, for $0 \leq i \leq m$ and $0 \leq j \leq p$, let $\mathcal{C}(i,j;L)$ (respectively $\mathcal{C}(i,j;R)$) be the cost of uniform route on network induced by the subgraph $a_i, \ldots, a_1, D, b_1, \ldots, b_j$ in which a_i is the last element of the sequence (b_j is the last element respectively) and let $T(i,j;L)$ and $T(i,j;R)$ be the times taken for these uniform routes. The initial conditions for the dynamic programming algorithm is
$\mathcal{C}(0,0;L) = \mathcal{C}(0,0,R) = T(0,0;L) = T(0,0;R) = 0$.
For $i,j > 0$, let $\alpha = T(i,j-1;L) + t(i,j)$, $\beta = T(i,j-1;R) + t(j-1,j)$, $\gamma = T(i+1,j;L) + t(i,i+1)$ and $\delta = T(i+1,j;R) + t(i,j)$. Then
$\mathcal{C}(i,j,R) = \min\{\mathcal{C}(i,j-1,L) + \alpha, \mathcal{C}(i,j-1,R) + \beta\}$
$\mathcal{C}(i,j,L) = \min\{\mathcal{C}(i+1,j,R) + \gamma, \mathcal{C}(i+1,j,L) + \delta\}$

$$T(i,j,R) = \begin{cases} \alpha & \text{if} \quad \mathcal{C}(i,j,R) = \mathcal{C}(i,j-1;L) + \alpha \\ \beta & \text{if} \quad \mathcal{C}(i,j,R) = \mathcal{C}(i,j-1;L) + \beta \end{cases}$$

$$T(i,j,L) = \begin{cases} \gamma & \text{if} \quad \mathcal{C}(i,j;L) = \mathcal{C}(i+1,j;R) + \gamma \\ \delta & \text{if} \quad \mathcal{C}(i,j;L) = \mathcal{C}(i+1,j;L) + \delta \end{cases}$$

Theorem 4 *The special case of Line-TRP in which the handling times of all nodes is zero can be solved in $O(n^2)$ where n is the number of locations.*

Proof: The algorithm proceeds by computing $\mathcal{C}(i,j,L)$ and $\mathcal{C}(i,j,R)$ for $0 \leq i \leq m$ and $0 \leq j \leq p$. Each value of $\mathcal{C}$ can be computed in constant time using above given equations. Since there are $O(n^2)$ values to be computed, the theorem follows. ∎

We construct a configuration network for a given instance graph G with nodes representing uniform routes in G and edges representing extensions of one route to another by the addition of one new node; configuration network is again a LDAG. The construction is very similar to

one in the dSVRPTW-line problem. The vertices in configuration network are $\mathcal{V} = \{(i,j,R),(i,j,L) : 1 \leq i \leq m, 1 \leq p\} \cup \{(0,0,D)\} \cup \{(m+1,p+1,D)\}$ with source as $S(\mathcal{G}) = (0,0,D)$ and sink as $T(\mathcal{G}) = (m+1,p+1,D)$ respectively.

Each (u,v) in the graph denotes optimal cost route between the vertices u and v. We associate three values $d(u,v)$, $a(u,v)$ and $len(u,v)$ with each edge. let $u,u_1,u_2,\ldots,u_l,v$ be the optimal cost route between the vertices then $d(u,v)$ denotes the sum of waiting times of nodes in that path i.e., $d(u,v) = (l+1) * t(u,u_1) + l * t(u_1,u_2) + \ldots + 2 * t(u_{l-1},u_l) + t(u_l,v)$, $a(u,v)$ denotes the travelling time from u to v in the optimal path i.e., $a(u,v) = t(u,u_1) + t(u_1,u_2) + t(u_2,u_3) + \ldots + t(u_{l-1},u_l) + t(u_l,v)$ and $len(u,v)$ denotes the length of the optimal path (i.e., $l+1$). The initial values of $d(u,v)$, $a(u,v)$, are $t(u,v)$, where $t(u,v)$ denotes the travel time between those nodes. The initial values of $len(u,v)$ is 1 if $t(u,v) \neq 0$, else, $len(u,v)$ is 0.

There are edges from (m,p,L) and (m,p,R) to $(m+1,p+1,D)$ and from $(0,0,D)$ to $(1,0;L)$ and $(0,1,R)$. Finally, for $1 \leq i \leq m$ and $1 \leq j \leq p$ there are edges from (i,j,L) to $(i+1,j,L)$, (i,j,L) to $(i,j+1,R)$, (i,j,R) to $(i,j+1,R)$ and from (i,j,R) to $(i+1,j,L)$. The node (i,j,L) denotes the optimal route covering $[a_i,b_j]$ with a_i being the last node visited. Similarly the node (i,j,R) denotes the optimal route covering $[a_i,b_j]$ with b_j being the last node visited. Node $(0,0,D)$ and $(m+1,p+1,D)$ denote the null route and optimal route respectively.

There are $n+1$ layers in the configuration network $\mathcal{G}$, where $n = m+p+1$; $(0,0,D)$ will be in first layer and $(m+1,p+1)$ in the last layer; (i,j,L) and (i,j,R) will be in $(i+j+1)^{th}$ layer. If $b = \min(m,p)$, then the maximum number of nodes in any layer will be $2b+1$. The first layer will have only the source node and the last layer will have only sink node.

An $O(n)$ time Optimal parallel algorithm can again be obtained by first constructing the LDAG for the given problem instance and then for $n+1$ iterations, serially in turn removing all nodes in the second layer. For removing nodes at second layer we use the same algorithm as for dSVRPTW-problem.

An $O(\log^2 n)$ time parallel algorithm for TRP-line problem can again be obtained by first constructing a LDAG for the given problem instance and then for $\log n$ iterations removing nodes in even layers. The algorithm for removing all nodes at layer i is given below. Here, d is the maximum in-degree (or out-degree) of any node in the current iteration. Clearly, $d \leq 2n+1$.

for each node r of layer $i-1$ do /* at most $2b+1$ */
for each node v at layer i incident from r do /* at most d */
for each node p of layer $i+1$ incident from v do /* at most d */
parbegin
(a) $d(r,p) = \min_v (d(r,v) + d(v,p) + len(v,p) * a(r,v))$
(b) Let u be the node that gives rise to the minimum value of $d(r,p)$
(c) $a(r,p) = a(r,u) + a(u,p)$
(d) $len(r,p) = len(r,u) + len(u,p)$
parend;

The correctness proof will be similar to that for dSVRPTW-line problem. The algorithm takes $O(\log n * \log b)$ time with $O(nb^2/\log b)$ processors on a CREW PRAM where $b = \min(m,p)$.

Acknowledgements: We wish to thank anonymous referees for suggesting valuable corrections and for pointing out typos.

References

[1] F. Afrati, A. Cosmadakis, C. H. Papadimitriou, and N. Papakonstantinou. The complexity of the traveling repairman problem. *Theory of Information Applications*, 20(1):79–87, 1986.

[2] R. K. Ahuja, T. L. Magnanti, and J. D. Orlin. *Network Flows*. Prentice-Hall Englewood Cliffs, 1993.

[3] L. Bodin, B. Golden, A. Assad, and M. Ball. Routing and scheduling of vehicles and crews: The state of the art. *Computer Operations Research*, 10:62–212, 1983.

[4] M. L. Fredman and R. E. Trajan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[5] M. R. Garey and D. S. Johnson. Two-processor scheduling with start times and deadlines. *SIAM Journal of Computing*, 6:416–426, 1977.

[6] A. Gupta and R. Krishnamurti. Parallel algorithms for vehicle routing problems. In *Proceedings of High Performance Computing*, pages 144–151. IEEE, 1997.

[7] J. K. Lenstra, A. H. G. R. Kan, and B. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[8] C. H. Papadimitriou. The euclidean traveling salesman problem is $\mathcal{N}P$-complete. *Theoretical Computer Science*, 4:237–244, 1977.

[9] H. Psaraftis, M. Solomon, T. Magnanti, and T. Kim. Routing and scheduling on a shoreline with release times. *Management Science*, 36(2):254–265, 1987.

[10] M. W. P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operation Research*, 4:285–305, 1985/6.

[11] S. Saxena. *Design and Analysis of Some Combinatorial and Computational Geometry Problems for Parallel Execution*. PhD thesis, Comp. Sci. & Engg., IIT Delhi, January 1989.

[12] M. Solomon and J. Desrosiers. Time window constrained routing and scheduling problems: A survey. *Transportation Science*, 22:1–13, 1988.

[13] J. N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22:263–282, 1992.