A linear-time algorithm for finding optimal vehicle refueling policies

Shieu-Hong Lin *, Nate Gertsch, and Jennifer R. Russell

Department of Mathematics and Computer Science
Biola University, 13800 Biola Avenue, La Mirada, California 90639, USA.

Abstract

We explore a fixed-route vehicle refueling problem as a special case of the inventory-capacitated lot-sizing problem, and present a linear-time greedy algorithm for finding optimal refueling policies.

Key words: algorithms, capacitated lot-sizing problem with inventory bounds

1 Introduction

In the *inventory-capacitated lot-sizing problem* (ICLSP), the demand for a single commodity in each of the consecutive n stages of time must be met by production in that stage or from the accumulated inventory, and the computing task is to determine the amounts of production over the stages to minimize the total setup, production, and inventory holding cost subject to inventory capacity bounds over the stages [7] [1]. ICLSP is solvable by dynamic programming in $O(n^3)$ time for cases with zero setup costs and concave cost functions for production and inventory holding [7], and can be solved by linear programming for cases with setup costs and linear cost functions for production and inventory satisfying the Wagner-Whitin nonspeculative property [1]. ICLSP is different from the extensively studied *production-capacitated lot-sizing problem* (PCLSP) where the computing task is to minimize the total setup, production, and inventory holding cost subject to production capacity bounds over

* Corresponding author. Email: shieu-hong.lin@biola.edu

the stages of time, instead of inventory capacity bounds in ICLSP. Compared to ICLSP, PCLSP is NP-hard [5] [2], but is solvable in polynomial time for cases with a stationary production capacity [2] [3] [8] [4] [6]. Assumptions like zero setup costs [6], zero inventory holding costs [2], and linear cost functions [6] have been exploited in some of these special cases of PCLSP.

In this paper, we are interested in a special case of ICLSP where we have a stationary inventory capacity, zero setup costs, zero inventory holding costs, and linear cost functions for production. This special case of ICLSP has an application in minimizing the vehicle refueling cost for the cargo transportation business, where the vehicle travels along a fixed route with a series of fuel stations on it. The movement of the vehicle is composed of stages moving from one fuel station to the next, and fuel is the commodity in demand in these stages. Varying amounts of fuel are needed in these stages due to the differences of terrains and mileage between adjacent stations. Fuel must be purchased (produced) in each stage with varying prices from the fuel stations, and the capacity of the fuel tank imposes a stationary upper bound on the inventory of fuel. The computing task is to determine the amounts of fuel purchased at these stations to reach the destination with minimal total fuel cost. In the remainder of the paper, we describe this special case of ICLSP as the *simple fixed-route vehicle refueling problem*. We develop a linear-time greedy algorithm to solve the problem, which is an improvement over the previous $O(n^3)$ dynamic-programming algorithm for ICLSP with dynamic inventory capacity bounds, zero setup costs, and concave cost functions for production and inventory [7].

2 The Simple Fixed-Route Refueling Problem

Consider a vehicle with a fixed fuel tank capacity operates along a fixed route with a series of fuel stations $\mathbf{S} = \langle S_0, S_1, \ldots, S_{n-1} \rangle$. The vehicle starts with some U_0 units of fuel at fuel station S_0 in the beginning of the route and progressively move from current station S_i to next station S_{i+1} for $0 \leq i < n$ until ending in the destination station S_n in the end of the route. We refer to i as the *index* of station S_i. The vehicle can refuel any non-negative units of fuel at each fuel station as long as it does not exceed the capacity of the fuel tank.

Definition 1. (Problem Instances) A problem instance of the simple fixed-route refueling problem is a five-tuple $\langle C, U_0, \mathbf{S}, \mathbf{P}, \mathbf{G} \rangle$ regarding a vehicle initially with U_0 units of fuel in a fuel tank of capacity C along a fixed route with a series of fuel stations $\mathbf{S} = \langle S_0, S_1, \ldots, S_{n-1} \rangle$ and two vectors $\mathbf{P} = \langle P_0, P_1, \ldots, P_n \rangle$ and $\mathbf{G} = \langle G_0, G_1, \ldots, G_{n-1} \rangle$ where P_i is the unit fuel price charged at fuel station S_i and G_i is the number of fuel units consumed

to move the vehicle from station S_i to station S_{i+1}, $C \in \mathbb{R}^+$, $U_0 \in \mathbb{R}^+ \cup \{0\}$, and for $0 \leq i < n$, $P_i \in \mathbb{R}^+$ and $G_i \in \mathbb{R}^+$. The amount of fuel left when arriving at the destination station S_n is of no concern here, and we always artificially set the fuel price P_n at S_n to zero.

Definition 2. (The Simple Fixed-Route Refueling Problem) Given a problem instance $\mathbf{I} = \langle C, U_0, \mathbf{S}, \mathbf{P}, \mathbf{G} \rangle$, a valid refueling policy for $\mathbf{I}$ is a vector $\mathbf{F} = \langle f_0, f_1, \ldots, f_{n-1} \rangle$, $f_i \in \mathbb{R}^+ \cup \{0\}$ for $0 \leq i < n$ such that by purchasing f_i units of fuel at each station S_i the vehicle starting with a tank of U_0 units of fuel at the starting fuel station S_0 can progressively move from current station S_i to next station S_{i+1} for $0 \leq i < n$ until ending in the destination station S_n, never out of fuel and never exceeding the tank capacity C. The computing task of the simple fixed-route refueling problem is to determine an optimal valid refueling policy $\mathbf{F} = \langle f_0, f_1, \ldots, f_{n-1} \rangle$ that minimizes the total fuel cost $\sum_{0 \leq i < n} P_i * f_i$.

Given a problem instance $\mathbf{I} = \langle C, U_0, \mathbf{S}, \mathbf{P}, \mathbf{G} \rangle$, we can define two functions $FAR(.)$ and $LOW(.)$ later needed in the description of the greedy algorithm. $FAR(i)$ is the index of the farthest station that the vehicle can reach from S_i if starting with a full tank of fuel at fuel station S_i moving toward the destination station S_n without any refueling afterward. $LOW(i)$ is the index of the first station with a lower fuel price that the vehicle would encounter when moving from S_i toward S_n. The following are the formal definitions of the functions.

Definition 3. (The FAR Function) Given a series of n stations $\mathbf{S} = \langle S_0, S_1, \ldots, S_{n-1} \rangle$ and a problem instance $\mathbf{I} = \langle C, U_0, \mathbf{S}, \mathbf{P}, \mathbf{G} \rangle$, we define the FAR function as $FAR : \{0, 1, \ldots, n-1\} \rightarrow \{0, 1, \ldots, n\}$ where $FAR(i)$ is the unique index satisfying: (i) $\sum_{i \leq k < FAR(i)} G_k \leq C$ and (ii) $\sum_{i \leq k < j} G_k > C$ for every j in $[FAR(i), n-1]$.

Definition 4. (The LOW Function) Given a series of n stations $\mathbf{S} = \langle S_0, S_1, \ldots, S_{n-1} \rangle$ and a problem instance $\mathbf{I} = \langle C, U_0, \mathbf{S}, \mathbf{P}, \mathbf{G} \rangle$, we define the LOW function as $LOW : \{0, 1, \ldots, n-1\} \rightarrow \{0, 1, \ldots, n\}$ where $LOW(i)$ is the unique index satisfying: (i) $LOW(i) \in [i+1, n]$ and the station with the index $LOW(i)$ has a lower fuel price than that of S_i, and (ii) no station with an index in the range of $[i+1, LOW(i)]$ has a lower fuel price than that of S_i.

The following are two linear-time algorithms for the computation of the functions together with proofs of algorithmic correctness and computational efficiency. Note that the amount of fuel left when arriving at the destination station S_n is of no concern here, and for the convenience of computation we always artificially set $P_n = 0$ as the fuel price at S_n in the problem instance.

Algorithm 1. Computing $FAR(.)$
Input: a problem instance $\mathbf{I} = \langle C, U_0, \mathbf{S}, \mathbf{P}, \mathbf{G} \rangle$

regarding n stations $S_0, S_1, \ldots, S_n$ in **S**
Output: an array of n integers $FAR(i), 0 \leq i < n$
Variables: *tank*, i, *here*
1. initialize *tank* to C;
   initialize i and *here* to 0;
2. while $(i < n)$ // Calculate $FAR(i)$ for station S_i
   { 2.1. // Check the reachability of S_{here+1} with a full tank from S_i
   while ($tank \geq G_{here}$ and $here < n$)
   { $tank = tank - G_{here}$; $here = here + 1$; }
   2.2. // S_{here} is the farthest reachable from S_i
   $FAR(i) = here$;
   2.3. // Decide the tank level when reaching S_{here} from S_{i+1}
   if ($here > i$) // able to reach S_{i+1} from S_i
   $tank = tank + G_i$;
   else // unable to even reach S_{i+1} from S_i
   { $tank = C$; $here = i + 1$; }
   2.4. // Ready to calculate $FAR(i + 1)$ for S_{i+1} next
   $i = i + 1$;
   }
3. return the array of values $FAR(i), 0 \leq i < n$;

Lemma 1. *Algorithm 1 determines the value of every $FAR(i), 0 \leq i < n$ all together in $O(n)$ time correctly.*
Proof. To prove the algorithm runs in $O(n)$ time note that (1) for each iteration of step 2 as a whole the value of *here* is monotonically increased at least by one either in step 2.1 or step 2.3, (2) through each iteration of the *while* loop in step 2.1, the value of *here* is monotonically increased by one, and (3) the value of *here* is never updated to be greater than n. Therefore there are no more than n iterations of the *while* loop in step 2.1 throughout the entire execution of the algorithm, which gives us $O(n)$ total running time.

To prove the correctness of the algorithm, first we show that in the iteration of the outer *while* loop in step 2 where the value of variable i equals i' $(0 \leq i' < n)$ in the very beginning of the iteration, step 2.2 can correctly determine $FAR(i')$ if the value of variable *tank* in the very beginning of that iteration equals the amount of fuel left in the tank when the vehicle reaches S_{here} given a full tank of fuel from station $S_{i'}$ without refueling afterward. Let's refer to the statement above as Proposition 1. Note that as long as the destination station S_n is not yet reached and there is enough fuel left in the tank to move from current station S_{here} to next station S_{here+1}, the inner *while* loop in step 2.1 simply iteratively (1) subtracts G_{here}, the amount of fuel consumed when moving from current station S_{here} to next station S_{here+1}, from the current value of *tank* and (2) increases the value of *here* by one. Therefore, when the precondition of Proposition 1 is true, by the time step 2.2 is executed after the while loop in

step 2.1 is finished, the value variable *here* is the index of the farthest station the vehicle can reach given a full tank of fuel from station $S_{i'}$ without refueling afterward, and this index is correctly recorded as $FAR(i')$ in step 2.2. This establishes Proposition 1.

Second, we prove by induction on i' for $0 \leq i' < n$ that in the iteration of the outer *while* loop in step 2 where the value of variable i equals i' $(0 \leq i' < n)$ in the very beginning of the iteration, the value of variable *tank* in the very beginning of that iteration always equals the amount of fuel left in the tank when the vehicle reaches S_{here} given a full tank of fuel from station $S_{i'}$ without refueling afterward. Let's refer to the statement above as Proposition 2. For the inductive base, we have $i' = 0$ and Proposition 2 is obviously true in this case since step 1 initializes *tank* to the full tank capacity C and sets both i and *here* to zero before entering the first iteration of the outer *while* loop in step 2. For the inductive step, assuming that Proposition 2 is true when the value of variable i equals some i' $(0 \leq i' < n-1)$ in the very beginning of an iteration of the outer *while* loop in step 2, it is easy to see that the inner *while* loop in step 2.2 maintains the value of *tank* to be equal to the amount of fuel left in the tank when the vehicle reaches S_{here} given a full tank of fuel from station $S_{i'}$ without refueling afterward. Then entering into step 2.3 in the same iteration of the outer *while* loop, we have two cases to consider. When the logic condition in step 2.3 is true, the vehicle is able reach station $S_{i'+1}$ or beyond from $S_{i'}$ given a full tank of fuel. In this case, step 2.3 compensates $G_{i'}$ units of fuel (consumed when moving from $S_{i'}$ to $S_{i'+1}$) back to the tank, and this makes the value of variable *tank* now equal the level of fuel left in the tank when the vehicle reaches S_{here} given a full tank of fuel from station $S_{i'+1}$. When the logic condition in step 2.3 is false, the vehicle cannot even reach station $S_{i'+1}$ from $S_{i'}$ with a full tank of fuel. In this case the value of *tank* is reset to C the full capacity, the value of *here* is updated to $i'+1$, and the value of variable *tank* also equals the level of fuel left in the tank when the vehicle reaches S_{here} given a full tank of fuel from station $S_{i'+1}$. In both cases, Proposition 2 is still true when entering the next iteration of the outer *while* loop in step 2 where the value of variable i equals $i+1'$ in the very beginning of that iteration. This completes the proof of Proposition 2 by induction. Proposition 1 and Proposition 2 together show that the algorithm correctly determines $FAR(i')$ for $0 \leq i' < n$. □

Algorithm 2. Computing $LOW(.)$
Input: a problem instance $\mathbf{I} = \langle C, U_0, \mathbf{S}, \mathbf{P}, \mathbf{G} \rangle$
regarding n stations $S_0, S_1, \ldots, S_n$ in $\mathbf{S}$
Output: an array of n integers $LOW(i), 0 \leq i < n$
Variables: i, *next*
1. initialize i to $n-1$;
2. while $(i \geq 0)$ // Calculate $LOW(i)$ for station S_i
{ 2.1. if ($P_{i+1} < P_i$)

{ $LOW(i) = i + 1$; goto step 2.4; }
else
$next = LOW(i + 1)$
2.2. while ($P_{next} \geq P_i$) $next = LOW(next)$;
2.3. $LOW(i) = next$;
2.4. $i = i - 1$;
}
3. return the array of values $LOW(i), 0 \leq i < n$;

Lemma 2. *Algorithm 2 determines the value of every* $LOW(i), 0 \leq i < n$ *all together in* $O(n)$ *time correctly.*
Proof. There is one iteration of step 2 for each i starting from $n - 1$ down to 0. By induction, we can show that each iteration correctly determines the value of $LOW(i)$ for i. For the inductive base, note that $P_n = 0$ is lower than any price elsewhere and step 2.1 correctly determines $LOW(n - 1)$ as n. For the inductive step, consider two cases. First, if the fuel price at station S_{i+1} is lower than that at S_i, the condition of the *if* statement in step 2.1 is false, and $LOW(i)$ is correctly computed as $i + 1$. Otherwise, the *while* loop in step 2.2 would iteratively search for and compare the next fuel price lower than P_{next} until it finds a fuel price lower than P_i. If the values of $LOW(j)$'s for $n - 1 \geq j > i$ have been correctly determined in previous iterations, the index of the station with that fuel price is correctly set as $LOW(i)$ in step 2.3.

To prove the algorithm runs in $O(n)$ time, we need to establish the proposition that if the variable *next* in the *while* loop of step 2.2 once had a particular value k where $LOW(i) > k > i$ when probing for a fuel price lower than P_i of station S_i, then *next* will never resume this particular value k again later in the *while* loop of step 2.2 when probing for a fuel price lower than $P_{i'}$ of station $S_{i'}$ for some $i' < i$. Based on the proposition above, the total number of iterations of step 2.2 when running the algorithm is $O(n)$ since the values of *next* can only be in the range of $[1, n]$ and thus the total number of iterations through step 2.2 is $O(n)$. Let's assume we have $LOW(next)$ equals k for some *next* in the range of $[i' + 1, i]$ sometime when probing for a fuel price lower than $P_{i'}$ of station $S_{i'}$ in the *while* loop of step 2.2. There are two cases to consider. On the one hand, if P_{next} is greater than P_i, then $LOW(next)$ must be less than or equal to i. So $LOW(next)$ could not equal k, which is greater than i. This is a contradiction. On the other hand, if P_{next} is less than or equal to P_i, then P_{next} is also less than P_k because P_k must be greater than P_i according to the setting of step 2.2. So $LOW(next)$ could not equal k. This too is a contradiction. □

3 A Linear-Time Greedy Algorithm

In the following, we present Algorithm 3 to determine the existence of valid refueling policies and returns an optimal refueling policy based on the $FAR(.)$ function and the $LOW(.)$ function defined according to the given problem instance. This algorithm essentially simulates the movement of the vehicle from the starting fuel station toward the destination station by repeatedly going through a series of greedy decision-making steps along the way until the destination station is reached. The following is the highlight of the greedy decision-making rules along with a brief explanation of the variables and the notation used in the algorithm.

- We use the variable i to keep the index of the station the algorithm is exploring at the moment. S_i and f_i denote the station the algorithm is exploring currently and the amount of fuel to fill in at S_i. We use the vector of variables $\mathbf{F} = \langle f_0, f_1, \ldots, f_{n-1} \rangle$ to record the refueling decisions. We use the variable $fuel$ to record the amount of fuel currently in the tank when the algorithm simulates the vehicle movement.
- If $FAR(i)$ equals (i), we are not able to move from the current fuel station S_i to the immediately next station even with a full tank of fuel. In this case, there is no valid refueling policy.
- If $LOW(i)$ is greater than $FAR(i)$, no fuel station reachable within the range of a full tank of fuel from the current fuel station S_i has a lower fuel price. In this case, we greedily fill the tank to full capacity at the current fuel station and then move on to the next station.
- If $LOW(i)$ is less than $FAR(i)$, the closest fuel station with a lower fuel price is reachable within the range of a full tank of fuel from the current fuel station S_i. In this case, we stingily fill the minimum amount of fuel needed at the current fuel station to reach that closest station with a lower unit fuel price.

Algorithm 3. Finding an optimal refueling policy
Input: a problem instance $\mathbf{I} = \langle C, U_0, \mathbf{S}, \mathbf{P}, \mathbf{G} \rangle$
regarding n stations $S_0, S_1, \ldots, S_n$ in $\mathbf{S}$
Output: an optimal refueling policy as a vector
Variables: $here$, $fuel$, $\mathbf{F} = \langle f_0, f_1, \ldots, f_{n-1} \rangle$
1. determine $FAR(i)$ and $LOW(i)$ for every i in the range of $[0, n-1]$
using Algorithm 1 and Algorithm 2;
initialize $\mathbf{F} = \langle f_0, f_1, \ldots, f_{n-1} \rangle$ to $\langle 0, 0, \ldots, 0 \rangle$;
initialize $fuel$ to U_0 and initialize i to 0;
2. while $(i < n)$ // Make a fueling decision at fuel station S_i
{ 2.1 if ($FAR(i)$ equals i)
stop and report that no valid refueling policy exists;
2.2 if ($LOW(i) > FAR(i)$)

{ // Fill in the entire tank to the full capacity
$f_i = (C - fuel)$;
// Move to the next station
$fuel = C - G_i$; $\quad i = i + 1$;
}

2.3 if $(LOW(i) \leq FAR(i))$
{ // Fill in the bare minimum to reach S_{low}
if $(\sum_{i \leq j < LOW(i)} G_j) > fuel$;
$f_i = (\sum_{i \leq j < low} G_j) - fuel$;
else
$f_i = 0$;

// Move all the way to station S_{low} without further refueling
$fuel = fuel + f_i - \sum_{i \leq j < LOW(i)} G_j$;
$i = LOW(i)$;
}
}

3. return the refueling vector $\mathbf{F} = \langle f_0, f_1, \ldots, f_{n-1} \rangle$ as the fueling policy

Lemma 3. *Algorithm 3 runs in $O(n)$ time to determine the existence of valid refueling policies and returns a valid refueling policy if one does exist.*

Proof. First, if the condition of step 2.1 is true in any iteration, obviously there is no way to reach the next station S_{i+1} from S_i even with the full tank of fuel. In this situation, the algorithm stops in step 2.1 to report this conclusion; otherwise the refueling vector $\mathbf{F}$ returned in step 3 is a valid refueling policy since it guarantees the vehicle has enough fuel to move through the series of fuel stations to the destination. According to Lemma 1 and Lemma 2, the computation in step 1 takes $O(n)$ time. On each iteration of step 2, either step 2.1 or step 2.2 (but not both) is executed. If step 2.1 is executed, i is increased by one and step 2.1 takes constant time to finish. If step 2.2 is executed, i is increased to $LOW(i)$ while step 2.2 takes time proportional to $LOW(i) - i$ to calculate $\sum_{i \leq j < LOW(i)} G_j$. Since i is monotonically increased from 0 to n in the end, Algorithm 3 runs in $O(n)$ time in total. □

Lemma 4. *The refueling policy returned by step 3 of Algorithm 3 is an optimal refueling policy.*

Proof. Let $\mathbf{F} = \langle f_0, f_1, \ldots, f_{n-1} \rangle$ be the policy returned by Algorithm 3 in step 3, let i_0 be zero, and let i_k be the value of variable i by the end of the kth iteration through step 2. To prove the lemma, we show in the following by induction on k (for k greater than or equal to zero and less than or equal to the total number of iterations through step 2) that there always exists an optimal refueling policy $\mathbf{F}^* = \langle f_0^*, f_1^*, \ldots, f_{n-1}^* \rangle$ where $f_j^* = f_j$ for $0 \leq j \leq i_k - 1$. Note that when the kth iteration is the very last iteration through step 2 before entering step 3, i_k is n and the statement above therefore asserts the

optimality of the refueling policy $\mathbf{F} = \langle f_0, f_1, \ldots, f_{n-1} \rangle$.

For the inductive base, k is zero, $i_k = i_0$ is zero, and the statement is trivially true. For the inductive step, we show that for a valid kth ($k \geq 1$) iteration through step 2 if there exists an optimal refueling policy $\mathbf{F}^* = \langle f_0^*, f_1^*, \ldots, f_{n-1}^* \rangle$ where $f_j^* = f_j$ for $0 \leq j \leq i_{k-1} - 1$, then there exists an optimal refueling policy $\mathbf{F}^* = \langle f_0^*, f_1^*, \ldots, f_{n-1}^* \rangle$ where $f_j^* = f_j$ for $0 \leq j \leq i_k - 1$. Note that the value of variable i in the very beginning of the kth iteration of step 2 is simply i_{k-1}. There are two cases to consider. Either step 2.1 or step 2.2 (but not both) is executed in the kth iteration, depending on whether $LOW(i_{k-1})$ is greater than $FAR(i_{k-1})$.

First, if $LOW(i_{k-1})$ is greater than $FAR(i_{k-1})$, step 2.1 is executed in the kth iteration to fill the tank full at station $S_{i_{k-1}}$ and then move to station $S_{i_{k-1}+1}$. Therefore the value of variable i in the end of the kth iteration, i_k, is simply $i_{k-1} + 1$, and $f_{i_{k-1}}$ is the amount to fill the tank. Note that in this case the fuel price at $S_{i_{k-1}}$ is no more expensive than any station with indices in the range of $[i_{k-1}, FAR(i_{k-1})]$. Also note that the vehicle definitely needs to fill in more than $f_{i_{k-1}}$ amount of fuel through one or more of the stations with indices in the range of $[i_{k-1}, FAR(i_{k-1})]$ in order to reach station $S_{FAR(i_{k-1})+1}$. If $f_{i_{k-1}}^*$ in an optimal policy is not equal to $f_{i_{k-1}}$ the cost to fill the tank, $f_{i_{k-1}}^*$ must be less than $f_{i_{k-1}}$. In this situation, we can increase the amount of fuel purchase at $S_{i_{k-1}}$ by $f_{i_{k-1}} - f_{i_{k-1}}^*$ and decrease the combined amount of fuel purchased at other stations in the range of $[i_{k-1}+1, FAR(i_{k-1})]$ by $f_{i_{k-1}} - f_{i_{k-1}}^*$ to arrive at another valid refueling policy without increasing the total cost. So we can always tune $f_{i_{k-1}}^*$ to $f_{i_{k-1}}$ without losing optimality and without changing f_j^* for $0 \leq j \leq i_{k-1} - 1$. Since i_k equals $i_{k-1} + 1$ in this case, this means there always exists an optimal refueling policy, $\mathbf{F}^* = \langle f_0^*, f_1^*, \ldots, f_{n-1}^* \rangle$ where $f_j^* = f_j$ for $0 \leq j \leq i_k - 1$. So the inductive step is true in the first case.

Second, if $LOW(i_{k-1})$ is less than or equal to $FAR(i_{k-1})$, step 2.2 is executed in the kth iteration to fill the tank at station $S_{i_{k-1}}$ with the minimum additional amount of fuel needed to move to station $S_{LOW(i_{k-1})}$. Therefore the value of variable i in the end of the kth iteration, i_k, is simply $LOW(i_{k-1})$, $f_{i_{k-1}}$ is the minimum additional amount of fuel needed to move to station $S_{LOW(i_{k-1})}$ from $S_{i_{k-1}}$, and f_j is zero for j in the range of $[i_{k-1} + 1, LOW(i_{k-1}) - 1]$. Note that, $S_{LOW(i_{k-1})}$ is the closest station from $S_{i_{k-1}}$ with a lower fuel price than that of $S_{i_{k-1}}$. All the stations with indices in the range of $[i_{k-1} + 1, LOW(i_{k-1}) - 1]$ have more expensive fuel prices than that of $S_{LOW(i_{k-1})}$. If $f_{i_{k-1}}^*$ in the optimal refueling policy $\mathbf{F}^* = \langle f_0^*, f_1^*, \ldots, f_{n-1}^* \rangle$ is less than the minimum additional amount $f_{i_{k-1}}$ to get to $S_{LOW(i_{k-1})}$, it must add fuel in one or more stations with indices in the range of $[i_{k-1} + 1, LOW(i_{k-1}) - 1]$ for fuel more expensive than that of $S_{i_{k-1}}$ to move beyond $S_{LOW(i_{k-1})}$. Without increasing total cost, we can fine tune the policy $\mathbf{F}^*$ by buying all the minimum additional amount $f_{i_{k-1}}$ to get to $S_{LOW(i_{k-1})}$ at station $S_{i_{k-1}}$, and fill nothing at stations with indices in

the range of $[i_{k-1}+1, LOW(i_{k-1})-1]$. This ends in an optimal refueling policy, $\mathbf{F}^* = \langle f_0^*, f_1^*, \ldots, f_{n-1}^* \rangle$ where $f_j^* = f_j$ for $0 \leq j \leq LOW(i_{k-1}) - 1$. Since i_k equals $LOW(i_{k-1})$ in this case, this means there always exists an optimal refueling policy, $\mathbf{F}^* = \langle f_0^*, f_1^*, \ldots, f_{n-1}^* \rangle$ where $f_j^* = f_j$ for $0 \leq j \leq i_k - 1$. So the inductive step is establised in this situation. If $f_{i_{k-1}}^*$ in the optimal refueling policy is more than the minimum additional amount $f_{i_{k-1}}$ to get to $S_{LOW(i_{k-1})}$, we can fine tune the policy without increasing the total cost by reducing the amount of fuel purchase at $S_{i_{k-1}}$ back to the bare minimum of $f_{i_{k-1}}$ to reach $S_{LOW(i_{k-1})}$, adding no fuel at other stations before reaching $S_{LOW(i_{k-1})}$, and then adding fuel at $S_{LOW(i_{k-1})}$ to the level the original refueling policy would end up with when arriving at $S_{LOW(i_{k-1})}$. This ends in an optimal refueling policy, $\mathbf{F}^* = \langle f_0^*, f_1^*, \ldots, f_{n-1}^* \rangle$ where $f_j^* = f_j$ for $0 \leq j \leq LOW(i_{k-1}) - 1$. Since i_k equals $LOW(i_{k-1})$ in this case, this means there always exists an optimal refueling policy, $\mathbf{F}^* = \langle f_0^*, f_1^*, \ldots, f_{n-1}^* \rangle$ where $f_j^* = f_j$ for $0 \leq j \leq i_k - 1$. So the inductive step is also true in this situation.

Theorem 1. *The greedy algorithm depicted in Algorithm 3 always correctly determines in $O(n)$ time whether valid refueling policies exist and returns an optimal refueling policy if valid refueling policies do exist.*
Proof. It follows from Lemma 3 and Lemma 4. □

Acknowledgements

The work described in this paper is part of a research project supported by a faculty development grant 2006-2007 at Biola University. The authors would like to thank all the students in the algorithms class in the fall semester of 2005 at Biola University for their participation in the early stage of the problem solving process.

References

[1] A. Atamtürk and S. Küçükyavuz. Lot sizing with inventory bounds and fixed costs: polyhedral study and computation, *Operations Research*, 53:711-730, 2005.

[2] G. R. Bitran and H.H Yanasse. Computational compexity of the capacitated lot size problem, *Management Science*, 28:1174-1186, 1982.

[3] C.S. Chung and C.H.M. Lin. An $O(T^2)$ algorithm for the NI/G/NI/ND capacitated lot-sizing problem, *Management Science*, 34:420-426, 1988.

[4] M. Florian and M. Klein. Deterministic production planning with concave costs and capacity constraints, *Management Science*, 18:12-20, 1971.

[5] M. Florian, J.K. Lenstra, and A.H.G. Rinnoy Kan. Deterministic production planning: algorithms and complexity, *Management Science*, 26:669-679, 1980.

[6] E. Girlich, M. Hoeding, A.A Zaporozhets, and S. Chubanov. A greedy algorithm for capacited lot-sizing problem, *Optimization*, 52:241-249, 2004.

[7] S. F. Love. Bounded production and inventory models with piecewise concave costs, *Management Science*, 20:313-318, 1973.

[8] C.P.M. van Hoesel and A.P.M. Wagelmans. An $O(T^3)$ algorithm for the economic lot-sizing problem with constant capacities, *Management Science*, 42:142-150, 1996.