2003

An Approach to Identifying Beneficial Collaboration Securely in Decentralized Logistics Systems

Richard Cho

Chris Clifton
Purdue University, clifton@cs.purdue.edu

Ananth V. Ilyer

Wei Jiang

Murat Kantarioglu

Report Number:
03-038

AN APPROACH TO IDENTIFYING BENEFICIAL COLLABORATION SECURELY IN DECENTRALIZED LOGISTICS SYSTEMS

Richard Cho
Chris Clifton
Ananth V. Iyer
Wei Jiang
Murat Kantarcioglu
Jaideep Vaidya

CSD TR #03-038
December 2003

An Approach to Identifying Beneficial Collaboration Securely in Decentralized Logistics Systems

Richard Cho • Chris Clifton • Ananth V. Iyer • Wei Jiang • Murat Kantarcıoğlu • Jaideep Vaidya

Krannert School of Management and Department of Computer Sciences, Purdue University, 403 West State Street, West Lafayette, Indiana 47907-2056

rcho@mgmt.purdue.edu • clifton@cs.purdue.edu • aiyer@mgmt.purdue.edu • wjiang@cs.purdue.edu • kanmurat@cs.purdue.edu • jsvaidy@cs.purdue.edu

Sharing capacity can lead to significant logistics improvements. Sharing the information needed to determine if capacity can be shared poses problems. We present a method that addresses both issues, specifically finding opportunities to swap loads without revealing any information except the loads swapped. The paper includes proofs of the security of this method, as well as an analysis of the cost reduction based on real-world transportation data.

1. Introduction

In a competitive market for operational capacity, there are numerous instances where individual companies would like to swap tasks or loads to gain operational efficiencies. Trucking companies face the problem of inefficiencies deriving from contracts that require them to deliver to all locations required by a shipper. Paper companies facing the need to do special setups or required to deliver to out-of-the-way locations would prefer to trade loads to improve the efficiency of production and delivery of rolls of paper. Steel companies face similar problems of dealing with surplus inventories with specific properties (Kalagnanam, Trumbo & Lee 2000). In such markets, the broker plays the role of market maker by collecting and distributing loads. Often, barter companies act as independent brokers to facilitate these transactions across competitors.

One reason for use of such a broker is that it prevents disclosure of proprietary information to competitors. However, this information must still be revealed to the broker. The cryptographic community has shown that a trusted third party is not required – it is possible to compute functions without either disclosing private data to any other party(Yao 1986, Goldreich, Micali & Wigderson 1987). The result is that no party learns more than they would

if a broker arranged the transactions, *and no broker is required.* Our goal is to apply state of the art techniques from data encryption to logistics problems to automate the task performed by the broker. Companies learn no more than with an honest broker. However, the broker is eliminated. In fact, for the specific problem and solution given in this paper, we prove that no party learns more than the minimum they must know to accomplish the desired efficiency gains. The benefit to shipping companies and shippers is the ability to reduce the costs associated with collaborating and thus improve the efficiency of the overall system.

In order to provide a complete treatment of our approach, we examine a specific problem context involving truck routing. Truck transport is a $481 billion industry in the US. However, the industry is extremely fragmented with the largest company accounting for less than 5% of the market. The main source of inefficiency in this industry is the "deadhead" miles or miles driven empty. The primary reason for this inefficiency is the spatial nature of this industry i.e., for a truck to pickup a load, it physically has to be at that location. When the truck is done, it ends up at the physical drop off point and then has to travel to the required location to be useful. Intuitively, if transport companies swap some of their loads, there is the potential for Pareto improving savings (i.e., neither company faces a higher cost and at least one company faces a lower cost).

However, attempts to collaborate and thus swap loads to get more efficient routes are often discouraged by the potential desire by individual companies to "share only if beneficial". In addition, legal restrictions, dealing with anti-trust issues, frown upon information sharing and collaboration that can be potentially anti-competitive. However anti-trust issues do permit competing firms to engage in limited information sharing and collaboration that is clearly efficiency enhancing. As reported in the Wall Street Journal, "It is permissible (for shipping companies) to cooperate in certain ways. For instance, if two of them carry chemicals for a given producer on the same route, they may pool their capacity for the purpose of operational efficiency. ... But cooperating to divide up markets or affect prices would fall outside these permitted arrangements" (Bandler 2003).

If transport companies resort to using a broker to swap loads, then the first step is for each company to independently identify loads it would like to swap. These potential loads are provided to a broker who now sees all available loads. As a result, parties will only make things available that they view as likely to be picked up in a swap. *The key difference in our approach is that all available loads are provided by all companies. The algorithm is executed in a distributed manner by each company and at no time is the data entrusted to any third*

party. In addition, the secure protocols used in the algorithm guarantee that no information other than the swapped loads that improve efficiency are revealed to each company, and all such efficiency-improving swaps are made. This results in both lower information disclosure and higher efficiency than a traditional broker intermediated model.

Why would companies want such an approach? One motivation comes from the transport companies themselves as part of their desire to protect proprietary information while achieving maximum efficiency. However, another motivation may come from shippers who could demand such a protocol be used by their associated carriers to ensure that efficiency is enhanced while preventing any collusion regarding data that is not explicitly required to be shared and that may reduce competitiveness of the carrier market.

Thus, in this paper, we do the following:

1. We provide an algorithm that ensures that sharing takes place only if each company sees its costs reduced and that the sharing scheme ensures that all potential players can engage to identify cost reducing swaps, while
   - ensuring no information is shared other than what can be concluded from the final swapped points, and
   - honesty on the part of the collaborators is ensured by guaranteeing that either it can be detected that one participant is cheating (and thus gets thrown out of future collaborations) or that the cheating is not incentive compatible, i.e., the cheater is worse off.
2. We apply the algorithm proposed to an empirical dataset from a transportation company that provided us with 12 weeks of pickup and delivery data. This empirical data suggests the delivered value of the algorithm. Empirical results suggest the potential to reduce costs by over 30% based on application of the algorithm.
3. We show that the algorithm, implemented in a decentralized manner, affords the global optimum split of loads for a specific setting. Empirical results show its role in practice when pickups and deliveries are considered.

While the algorithm proposed is a heuristic in the context of vehicle routing, it uses the state of the art techniques in data encryption and secure multi-party computation techniques that *guarantee* that the security requirements are met. It thus brings up an interesting issue for

algorithm design for managing information sharing i.e., how does one choose the tradeoff between an optimal algorithm for routing that might provide information leakage against a heuristic algorithm that prevents leakage but is a heuristic in the problem domain.

One goal with this paper is to suggest the application of cryptographic techniques as a enabler of operational efficiency as a potentially fruitful area of research in operations management. To further this goal, we also included summaries of the literature in data security and several illustrative examples for the proposed algorithm.

1.1 Problem Description

Given N independent transport companies, each with m points located in two dimensions that have to be served by a truck, identify a sequence of enquiries and swaps between pairs of companies that results in:

1. a set of points that when swapped between the companies guarantees that no company is worse off,

2. no information is shared other than what can be concluded from the final swapped points,

3. the algorithm is polynomial in running time, and

4. any cheating by either party during the execution of the algorithm can either be detected by the other party, or results in a less efficient solution for the cheating party, thus providing the incentive to truthfully follow the algorithm.

We first map the initial two dimensional problem to one dimension using a space filling curve. The basic idea of using a space filling curve to develop heuristics for combinatorial problems is described in Bartholdi III & Platzman (1988). Since there is a one to one mapping from two dimensions to one dimension, points identified for swap in one dimension provide a unique pointer to the corresponding original point location in two dimensions. Bartholdi and Platzman show that the heuristic has (a) 25% worse than optimal worst case performance for planar traveling salesman problems (Bartholdi III & Platzman 1982) and (b) generates solutions within one second that has a gap of less than 34% more than the best approach for large problems using 2 months of computing time (see http://www.isye.gatech.edu/~jjb/mow/mow.html) and (c) is used in many implemented logistics packages such as the ARC/Info Geographical Information System, the CAPS Logistics

Toolkit of Baan Systems, and other commercial systems managing 2-dimensional data (see http://www.isye.gatech.edu/~jjb/mow/mow.html). Note that our approach will be to incorporate the encryption processes within this heuristic.

Why map to one dimension? In general this is an NP-hard optimization problem, even without worrying about privacy/security. While proper choice of heuristics may give good solutions directly on the two dimensional problem, choosing those heuristics requires an understanding of the characteristics of the data – and sharing this information violates goal 2. However, in one dimension the optimal solution is tractable: search for the best solution requires a logarithmic number of steps. We will show that such search can be done in a way that reveals no information not obvious from the results (loads swapped), using secure comparison and oblivious transfer as cryptographic primitives.

The basic idea of the algorithm is that the parties use a space-filling curve to map this to a one-dimensional problem. In one dimension, the parties perform a binary search for the maximum number of points to swap such that each party benefits, i.e., all points received are closer than points given. We give a simple example based on $\triangle$'s point of view. Refer to Figure 1, and assume $\triangle$ wants all of its points to be located on the right. Figures 1(a)-1(c) reflect $\triangle$'s independent view: Its initial route, a space-filling curve mapping the points to one dimension, and the one-dimensional view of its initial points. Figure 1(d) is its view at the completion of the algorithm: It has given up its three most distant points, and received three that are closer. We will now describe the "magic" that allows $\triangle$ to determine which points it gives and receives, without learning (or revealing) anything else.

1.2 Solution Overview

In one dimension, the problem of determining which points to swap reduces to determining *how many* to swap. Each party is given one end of the line; the goal is to determine how many points to swap to partition the line.

To demonstrate this, we "open up" the view in Figure 2, so the reader (but not $\triangle$) can see both sides. The idea of the algorithm is to check and see if it is okay to swap a certain number of points, without either side learning anything except if that many points will be a beneficial swap. This check is done securely, so that the parties learn only if swapping a certain number of points is beneficial, not where the points are. We describe this using *oblivious transfer*. $\bigcirc$ builds a set of boxes, each containing either $>$ or $\leq$. The boxes

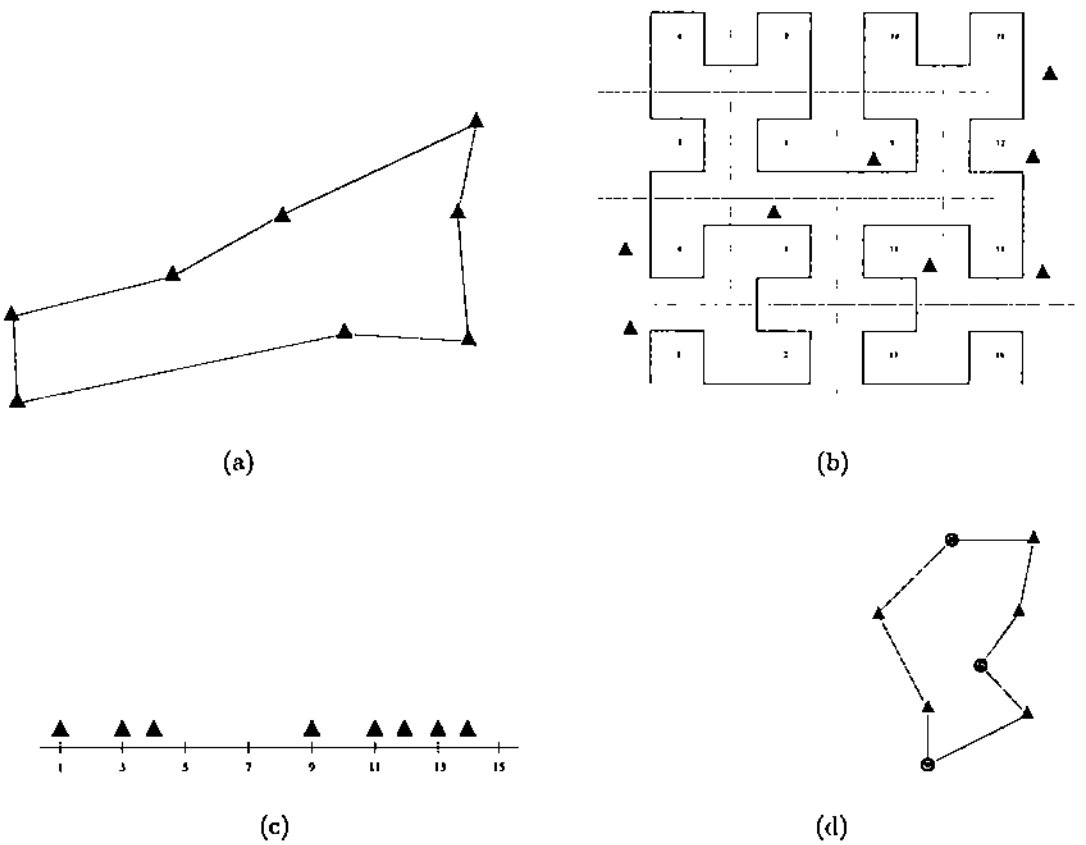

Figure 1: A feasible approach from Δ's point of view

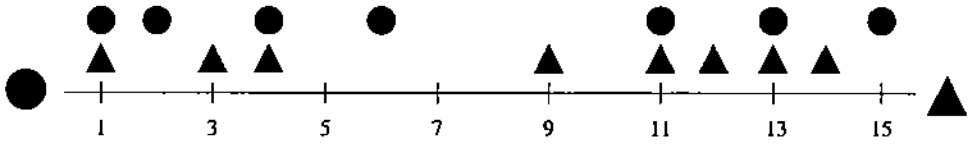

Figure 2: Illustration of OSROD Execution

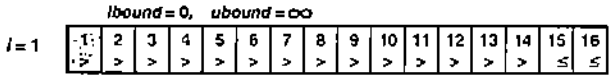

Figure 3: Example of execution

correspond to positions on a line, a box entry has a $<$ if ◯ will benefit from swapping the given number of points if every one it receives is at or to the left of that point.

Figure 3 shows the values of the boxes for checking if swapping a single point is okay. If ◯ receives points to the left of 15, it benefits (since it gives up a point at 15.) However, if it were to receive a point at 15 or 16, it would be better not to swap.

If it were to swap one point, △ would give up its point at position 1. It therefore opens the box at position 1 (its leftmost point) - since this is $>$, ◯ (and △) will gain from swapping one point.

The key to the security of the process is that ◯ doesn't know which box △ opened, and △ only learns the value in one box. (This is accomplished through a cryptographic protocol described on page 8.) Thus the only thing learned from the protocol is the value "$>$". Since the final result (Figure 1(d)) shows △ involves swapping three points, both parties could figure out at the end that swapping one point is okay. While they have learned something new *at this point*, since it will be obvious once the final swap occurs, it doesn't really reveal anything.

This process is repeated to find the right number of points to swap. Figure 4 shows the test for $i = 2$; △ looks at position 3 and finds it is okay to swap two points. Figure 5 shows $i = 4$; looking at position 9 △ finds that four points given by ◯ would include at least one to the left of this point, so it isn't a beneficial swap. All that is now left is testing a swap

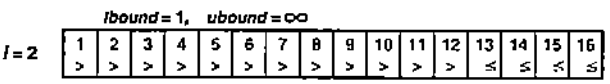

Figure 4: Example of execution

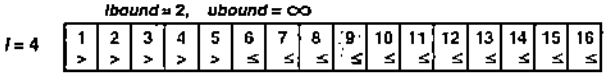

Figure 5: Example of execution

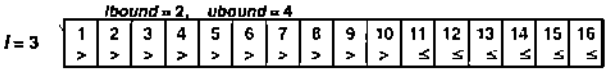

Figure 6: Example of execution

of 3 points - this is shown in Figure 6. Since this is okay ($<$), and four points doesn't work, the final result is to swap three points.

As we will proven later, nothing is learned from this protocol that ◯ and △ wouldn't learn from giving all of their data to an honest broker. This is accomplished without the need for an honest broker. For example, from what △ sees during the execution of the protocol (the shaded boxes only), △ knows that ◯ has fewer than four points to the right of position 9, and at least three to the right of position four. But these would be obvious even with an honest broker: △ learns of the os at 11, 13, and 15 from the swap, and knows that there cannot be another o to the right of 9 or it would have been swapped as well. Thus the algorithm ensures that sharing of beneficial information is identified by circle and triangle with no other information being revealed in the process.

We still need to make sure that only one of the electronic "boxes" is opened by △, and that ◯ doesn't learn which box is opened. We now show how to do this, *even if one party tries to cheat.* The cryptography community refers to this problem as 1 *out of N oblivious transfer*; it has been the subject of extensive research. Here, we will describe a simple 1 out of N oblivious transfer (OT_1^N) protocol from Naor & Pinkas (2001) and Naor & Pinkas (1999).

First, we describe two cryptographic definitions that are used in the protocol.

Computational Diffie-Hellman Assumption: Assume that p is a very large prime number and g is the generator of its multiplicative group (i.e. every number between $1 \ldots p-1$ can be written as $g^k \bmod p$ for some k between $1 \ldots p-1$). The computational Diffie-Hellman Assumption states that given $g^a \bmod p$ and $g^b \bmod p$ (Note that a and b is not given), there is no efficient way to compute $g^{ab} \bmod p$. This assumption is the basis for the Diffie-Hellman key exchange protocol; if it does not hold many

cryptographic techniques would be breakable.

Random Oracle Assumption: In the construction of the protocol, we will use a cryptographic Hash function H. We assume that this function is known to all parties (e.g., SHA) and it maps its input to what appears to be a random output. Again, this is a common cryptographic tool used in many protocols.

Now using the Diffie-Hellman Assumption and a hash function H, we can implement a 1 out of n oblivious transfer that discloses no information even if one of the parties tries to deviate from the protocol.

For simplicity, we will describe 1 out of 2 oblivious transfer (OT_1^2) first, then show a method to extend any OT_1^2 protocol to arbitrary OT_1^n using $log(n)$ OT_1^2 operations. In the following protocol, let σ be the index of the box that Δ wants to open. (Note that in OT_1^2, we have only two boxes), and that B_0 and B_1 are the contents of boxes 0 and 1 respectively. Also note that every operation except evaluating the function H and $\oplus$ (exclusive or) is done mod p.

1. $\bigcirc$ publishes a random number C between $1, \ldots, p-1$ along with g and p.
2. Δ picks a random number k between $1, \ldots, p-1$, sets $P_\sigma = g^k$ and $P_{1-\sigma} = C/P_\sigma$, and sends P_0 to $\bigcirc$.
3. $\bigcirc$ finds P_1 by evaluating C/P_0, creates $E_0 = (g^{r_0}, H((P_0)^{r_0}) \oplus B_0)$, $E_1 = (g^{r_1}, H((P_1)^{r_1}) \oplus B_1)$, by randomly choosing r_0, r_1 between $1, \ldots, p-1$, and sends E_0, E_1 to Δ.
4. Δ computes $H((P_\sigma)^{r_\sigma}) = H((g^{r_\sigma})^k))$ to find B_σ.

In the above protocol the choice of Δ (σ) is not revealed because all $\bigcirc$ receives is either g^k or C/g^k, where k is chosen randomly. Since operations are done in mod p, both g^k or C/g^k values are uniformly distributed between $0, \ldots, p-1$. Therefore, $\bigcirc$ does not see anything more than a random number. Δ learns nothing by receiving the random C, or (because of the random oracle hash function) from inspecting E_0 or E_1. While Δ can decrypt E_σ to obtain the final result, by the original Diffie-Hellman assumption it cannot determine $(g^{r_{\sigma-1}})^k$ to decrypt the other box. If *bigtriangleup* could decrypt both E_0 and E_1, it means that it knows a efficient way to find t such that $C = g^t \bmod p$ for given randomly chosen C. This would contradict with the Diffie-Hellman assumption, because given $g^a \bmod p$ and $g^b \bmod p$, Δ can calculate a and b to get $g^{ab} \bmod p$.

We now give an overview of how to use the OT_1^2 protocol to create a OT_1^N. Instead of H above, we use an encryption function E (e.g., DES). For simplicity, assume that ◯ has $B_1 \ldots B_{16}$ and △ wants to learn B_7.

1. ◯ generates 4 key pairs

$$(K_1^0, K_1^1), (K_2^0, K_2^1), (K_3^0, K_3^1), (K_4^0, K_4^1)$$

   where each K_j^i is a randomly chosen key for E. For $1 \leq i \leq 16$, with binary representation (i_1, i_2, i_3, i_4), ◯ creates $EB_i = B_i \oplus E_{K_1^{i_1}}(i) \oplus E_{K_2^{i_2}}(i) \oplus E_{K_3^{i_3}}(i) \oplus E_{K_4^{i_4}}(i)$

2. Since 7 (the box number △ wants to open) has binary representation 0111, using four OT_1^2, △ learns $K_1^0, K_2^1, K_3^1, K_4^1$.

3. ◯ sends all EB_i to △.

4. △ retrieves B_7 by calculating $EB_7 \oplus E_{K_1^0}(7) \oplus E_{K_3^1}(7) \oplus E_{K_4^1}(7)$

It can easily be proven that if the encryption scheme and OT_1^2 are secure then the above algorithm is secure. Since it only retrieves one key from each pair, △ accurately decrypt at most one message.

Those who are following closely will note a problems: the space and communication cost is linear in the possible choices for comparison. We really just need to compare two numbers, the cryptography community has shown how to do this in time $O(\log n)$. This will be discussed further in Appendix A.

The next section gives a formal treatment of the algorithm, along with proofs that it achieves a one-dimensionally optimal result, that nothing is disclosed that is not obvious from the result, and that cheating is either caught or detrimental to the cheater. In Section 2.4 we show that the algorithm can be used among multiple parties, and will converge on a globally optimal solution. Section 3 analyzes the improvement that would result from using this approach on a set of real shipping transactions. In Appendix A we give more background on secure comparison, showing how to handle the efficiency issues with the above simple approach.

2. Secure Relative Outlier Detection (SROD) Algorithms

This section gives the detail of the algorithms that have been discussed above. We start off by introducing the notation used in the rest of the section, and also give some necessary definitions. Subsection 2.3 describes the one dimensional secure relative outlier detection (SROD) algorithm and also gives a proof of correctness, a proof of security and a proof of honesty for the algorithm.

2.1 Notations

$S^k = \{s_1^k, s_2^k, \cdots, s_m^k\}$, a set of m k-dimensional points
$O^k = \{o_1^k, o_2^k, \cdots, o_n^k\}$, a set of n k-dimensional points
$T_{s^k} = \{s_i^k | s_i^k \in S^k\}$
$T_{o^k} = \{o_i^k | o_i^k \in O^k\}$
$SF \equiv$ A space filling curve
$[a, b] \equiv$ The range of SF

2.2 Formal Definition of Relative Outliers

Definition 1: (Relative outliers) T_{s^k} and T_{o^k} are relative outliers with respect to S^k and O^k if:

- $|T_{s^k}| = |T_{o^k}|$
- Cost of the optimal Hamiltonian cycles related to both S^k and O^k is minimized after the memberships of all points in T_{s^k} are swapped with those of all points in T_{o^k}
- T_{s^k} and T_{o^k} are smallest such sets

Definition 2: (Extreme Points with respect to X and d_x)

- X is a set of points, d_x is a direction (i.e. left or right) related to X
- An extreme point is a point in X that is the furthest away from d_x
- i_{th} extreme point is a point in X that is the i_{th} furthest away from d_x

Definition 3: $(Extreme\_POS(X, d_x, i))$

- X is a set of points, d_x is a direction (i.e. left or right) and i is an integer
- The function returns the position of the i_{th} extreme points with respect to X and d_x
- If $i > |X|$, the function returns a position beyond the range limits (e.g., $-\infty$, $+\infty$) in the direction d_x.

Formally, $Extreme\_POS(X, d_x, i)$ is:

Reorder X in ascending order according to position
if d_x =right **then**
Return the position of ith item of X ($+\infty$ if $i > |X|$)
else
Return the position of the $(|X| + 1 - i)$th item of X ($-\infty$ if $i > |X|$)
end if

To illustrate, in Figure 7 $Extreme\_POS(S^1, left, 2)$ returns 12, and $Extreme\_POS(S^1, left, 6)$ returns $-\infty$.

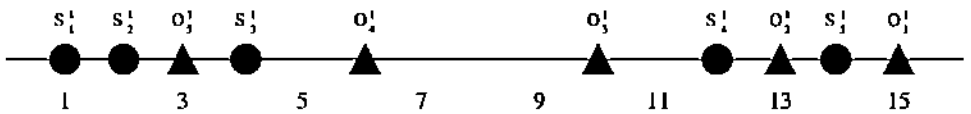

Figure 7: Illustration of extreme points

2.3 One-dimensional Secure Relative Outlier Detection (SROD) Algorithm

We gave a brief example of the way our protocol works in the introduction. The basic idea of the protocol is that the parties perform a binary search for the maximum number of beneficial points to swap. Given a number of points k, the parties compare their k^th extreme points. If the locations don't cross, k is a lower bound and the parties try $2k$. Once the locations cross, k is an upper bound, and the search continues between the upper and lower bounds until the right number of points is found. The set of extreme points become the relative outlier set at the end of the execution of the protocol.

The detailed protocol is given in Algorithm 1. Line 2 through line 6 determines initial directions to start the protocol. Because both parties may request the same direction, if one

Algorithm 1 SROD: One Dimensional Secure Relative Outlier Detection

Require: S^1, d_s, O^1, d_o

1: {Line 2 through 6 determines initial direction}
2: **if** $|S^1| \geq |O^1|$ **then**
3: $\quad d_o \leftarrow \overline{d}_s$
4: **else**
5: $\quad d_s \leftarrow \overline{d}_o$
6: **end if**
7: $lbound \leftarrow 0$
8: $ubound \leftarrow \infty$
9: $i \leftarrow 1$
10: {Linc 11 through 18 determines the maximal size of i}
11: **while** $(ubound - lbound > 1)$ **do**
12: $\quad$ **if** $Extreme\_POS(S^1, l, i) > Extreme\_POS(O^1, r, i)$ **then**
13: $\quad\quad lbound \leftarrow i$
14: $\quad$ **else**
15: $\quad\quad ubound \leftarrow i$
16: $\quad$ **end if**
17: $\quad i \leftarrow min\left(i * 2, \left\lfloor \frac{lbound+ubound}{2} \right\rfloor\right)$
18: **end while**
19: return i

party has more points than the other it gets its choice of direction (this ensures an optimal result.) Lines 11 through 18 finds the maximal size of the relative outlier set.

The only communication occurs in lines 2 and 12, each of which is a comparison. We have given a brief idea of how this is done, the details (from (Yao 1986)) are given in the Appendix. To make lines 2-6 secure (only revealing the direction, so that if both get their choice neither learns who has the larger set), we need a slightly more complex protocol. As the secure comparison method builds on securely evaluating a boolean circuit, it is straightforward to extend this to returning the direction rather than the comparison result. The only requirement is a simple circuit that takes the desired of each and the comparison result as input, and outputs the result direction. Merging and evaluating both circuits as one (as described in the Appendix) allows these to be compared securely.

2.3.1 Illustration of SROD Execution

Let ◯ and △ be two parties, with original routes shown in Figure 8(a). Figures 8(b) and 8(c) present a space transformation process via a Hilbert curve. We now show the execution of the algorithm from ◯'s point of view.

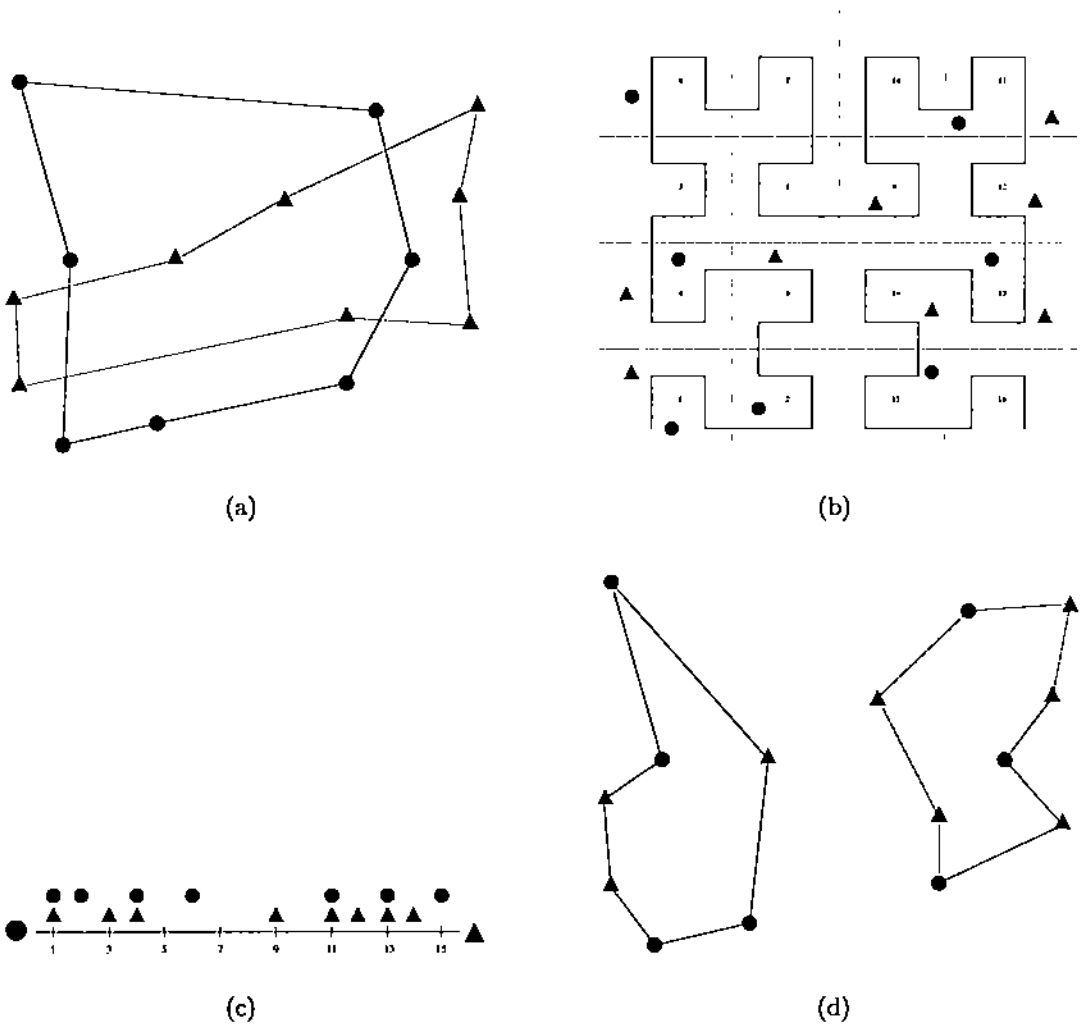

Figure 8: Execution of SROD protocol between two parties

Initially, *lbound* $\leftarrow 0$, *ubound* $\leftarrow \infty$ and $i \leftarrow 1$. Since the difference between *lbound* and *ubound* is greater than 1, the execution enters the *while* loop. $Extreme\_POS(\bigcirc, left, 1)$ returns 15, which is greater than the index returned from $Extreme\_POS(\triangle, right, 1)$ (1), so *lbound* $\leftarrow 1(i)$ and $i \leftarrow 2$. In the next iteration, $Extreme\_POS(\bigcirc, left, 2) = 13$ is still greater $Extreme\_POS(\triangle, right, 2) = 3$, so *lbound* $\leftarrow 2$ and $i \leftarrow 4$. $Extreme\_POS(\bigcirc, left, 4) = 6 < Extreme\_POS(\triangle, right, 4) = 9$, so *ubound* $\leftarrow 4$ and $i \leftarrow 3$. $Extreme\_POS(\bigcirc, left, 3) = 11 > Extreme\_POS(\triangle, right, 3) = 4$, so *lbound* $\leftarrow 3$ and $i \leftarrow 3$. since $ubound - lbound = 1$, the execution exits with 3 as the number of relative outliers, or points to be swapped

Figure 8(d) shows the resulting routes after each party gives up their worst three points.

2.3.2 Proof of Correctness

A simple inductive proof demonstrates that the process terminates with the optimal (in one dimension) results.

If *ubound* $\geq$ *lbound* then the following hold

- *lbound* $\leq i \leq$ *ubound*
- since only way to change *ubound* and *lbound* is to set to i, *ubound* $\geq$ *lbound* holds (induction step)

If $Extreme\_POS(S, l, i) > Extreme\_POS(O, r, i)$ then i is not greater than

- *target_result*,
- *lbound* gets set to i, otherwise *lbound* is not changed, so
- *lbound* $\leq$ *target_result*

If $Extreme\_POS(S, l, i) \leq Extreme\_POS(O, l, i)$ then

- $i >$ *target_result*
- *ubound* gets set to i, otherwise *ubound* not changed, so
- *ubound* $>$ *target_result*

If $ubound - lbound = 1$ then

- *ubound* $>$ *target_result*

- $lbound \leq target\_result$
- so $lbound = target\_result$

If $ubound - lbound = 1$ then

- i set to $lbound$

Therefore, at the end, $i = lbound = target\_result$.

One caveat: If both parties want the same end, it is possible (but not certain) that the smaller party will end up with a longer tour than it started with. The global cost cannot increase: the benefit accruing to the large party will be at least as great as the loss faced by the smaller party. If this is a problem (i.e., parties will only participate if they are guaranteed not to have a longer tour), steps 2-6 can be modified to abort the protocol if both parties choose the same direction. If the protocol is aborted, the only information revealed is that the two parties have their business concentrated on the same end of the line.

2.3.3 Proof of Security

Theorem 2.1. *Algorithm 1 is secure under the definitions of Secure Multiparty Computation.*

Proof. Communication occurs from line 2 to line 6 and line 12. To prove the protocol is secure, we only need to show these two parts of the protocol can be computed securely. This is done with a simulation approach. We can build a simulator such that given one party's inputs, the number of relative outliers, and a direction, we are able to simulate what that party sees during every step of its execution. Since directions are parts of the final results, we can simulate the first part directly from the final results. A party may know the other party has a larger dataset if the returned direction is different from the party's requested direction. However, this additional information is not considered as an information leak because it is inferred from the final results. Therefore, this first part of the protocol is secure. Let $S\_SROD(S^1, d_s, n)$ be simulator of line 12 of the SROD protocol where n is the number of relative outliers and d_s is a direction. Without loss of generality, assume $d_s = left(l)$. The simulator is given in Algorithm 2.

The basic idea of the simulator is the following: let n be the number of points to swap, and t be a temporary size of the relative outlier set of S^1. If t is less than n, S^1 knows t has yet to be maximal, and the protocol will double the size of t. On the other hand, if t is

Algorithm 2 Simulator for SROD Protocol

Require: S^1, d_{S^1}, n

1: $i \leftarrow 1$
2: $lbound \leftarrow 0$
3: $ubound \leftarrow \infty$
4: **while** $(i \neq n)$ **do**
5: **if** $i < n$ **then**
6: $Known : Extreme\_POS(S^1, l, i) > Extreme\_POS(O^1, r, i)$
7: $lbound \leftarrow i$
8: $i \leftarrow i * 2$
9: **else**
10: $Known : Extreme\_POS(S^1, l, i) < Extreme\_POS(O^1, r, i)$
11: $ubound \leftarrow i$
12: $i \leftarrow \left\lfloor \frac{lbound+ubound}{2} \right\rfloor$
13: **end if**
14: **end while**
15: $Known : Extreme\_POS(S^1, l, n) = Extreme\_POS(O^1, r, n)$

greater than n, S^1 knows t is too big to be maximal, and the protocol will decrease the size of t to a value that is in the middle of its upper and lower bounds. If t is the same as n, t becomes the size of the relative outlier set of S^1, and the protocol terminates its execution.

From the simulator, it is oblivious that the shared results from the protocol plus a party's input are sufficient to precisely simulate each execution of the protocol. Therefore, because the simulation process and execution of the protocol are computationally indistinguishable, the SROD protocol is secure. □

2.3.4 Proof of Honesty

To show that the protocol is incentive compatible, we define an objective function, and then prove that by cheating:

- the cheating party gets caught
- the cheating party loses due to the decrease in its objective function
- cheating results in no change in the true solution (solution without cheating)

The objective function for a party is defined as the tour length for its points. Formally, the cost for a party P, $Cost(P)$ is

{Let S represent the list of points owned by P}

Reorder S in ascending order according to position

$Cost = Position(S_{|S|}) - Position(S_1)$

The following discussion assumes that P is the cheating party, and Q is the honest party. We use the following definitions:

- S_p (resp. S_q) represents $P's$ (resp. $Q's$) original set of points.
- $Swap_{pt}$ (resp. $Swap_{qt}$) represents the set of points of P (resp. Q) that are swapped in a true run of the protocol (i.e., if P was honest).
- $Swap_{pc}$ (resp. $Swap_{qc}$) represents the set of points of P (resp. Q) that are swapped after a "cheating" run of the protocol.
- $Cost(P)_{true}$ represents $P's$ final cost after correctly executing the protocol.
- $Cost(P)_{cheat}$ represents $P's$ final cost after "cheating" in the protocol.
- $Pos(X)$ represents the position of a point X.
- $nswap_{true} = |Swap_{pt}|$ represents the number of points to swap in a correct execution of the protocol.
- $nswap_{cheat} = |Swap_{pc}|$ is the number of points to swap achieved by "cheating" in the protocol in some way.
- $|S_p| = n$ is the total number of points originally owned by P.

There are two ways that a party can cheat.

1. Falsifying the input to the protocol:
   (a) Withholding points from the protocol entirely,
   (b) Withholding points from execution, but adding them to swap, or
   (c) Adding fake points
2. Modifying or causing aberrations in the execution of the protocol:
   (a) Aborting the protocol,
   (b) Lying / Modifying step 12 of protocol 1, or

While it would appear there are other ways of cheating (e.g., lying in steps 2-6), any such cheating is equivalent to one of the above. For example, any cheating in the secure protocol computing steps 2-6 will give an execution of those steps equivalent to some falsified input.

With one exception (cheating at Step 2, discussed later), each form of cheating leads either to a suboptimal result from the cheater's point of view, or the honest party is able to detect that cheating has occurred. We will prove this on a case-by-case basis. Without loss of generality, we assume that P holds the left direction while Q holds the right direction.

Case 1a: Here a party withholds points from the protocol entirely. Assume that P withholds a point X from the protocol. Either $X \notin Swap_{pt}$ or $X \in Swap_{pt}$.

$$\begin{aligned}
X \notin Swap_{pt} &\Rightarrow Swap_{pt} = Swap_{pc} \Rightarrow Swap_{qt} = Swap_{qc} \\
&\Rightarrow \text{Protocol is unaffected} \\
X \in Swap_{pt} &\Rightarrow \text{either } (\exists Y \in S_q | Y \in Swap_{qt} \wedge Y \notin Swap_{qc} \wedge pos(Y) < pos(X)) \\
&\quad \text{or } (\exists Y \in S_p | Y \in Swap_{pc} \wedge Y \notin Swap_{pt} \wedge pos(Y) < pos(X)) \\
&\Rightarrow Cost(P)_{cheat} > Cost(P)_{true}
\end{aligned}$$

In this case, either the points would not have been swapped anyway, and thus do not affect the protocol, or they would have been swapped. In the latter case, again there are two possibilities. The first is that there were closer points that P could have gained, so the cheating party ends up with a longer travel distance than if it had behaved correctly. The other is that P could have given up some of these points instead of some of the points it will now have to swap. But even in this case, the points withheld have to be farther for P than the points swapped, thus it still ends up with a longer travel distance. Both of these could in fact be considered legitimate - one party may choose to withhold some loads from the ones that may be swapped.

Case 1b: Falsifying input can also occur through withholding points during the execution of the protocol, then adding them to the swap (instead of legitimate points). Assume that P withholds a point X from the execution of the protocol but adds it to the swap instead of some point Z. i.e. $X \notin S_p \wedge$X substituted for some point Z $\in Swap_{pc}$. Again, either $nswap_{true} =$

$nswap_{cheat} + 1$ or $nswap_{true} = nswap_{cheat}$ (Note: $nswap_{true}$ *cannot* be $nswap_{cheat} - 1$).

$$
\begin{aligned}
nswap_{true} = nswap_{cheat} + 1 \quad \Rightarrow \quad & X \in Swap_{pt} \Rightarrow \exists Y \in Swap_{qt} | Y \notin Swap_{qc} \wedge \\
& (pos(Y) < pos(X) \vee pos(Y) < pos(Z)) \\
& \Rightarrow Cost(P)_{cheat} > Cost(P)_{true} \\
nswap_{true} = nswap_{cheat} \quad \Rightarrow \quad & \text{either } X \in Swap_{pt} \Rightarrow (\exists W \in S_p | W \in Swap_{pc} \wedge W \notin Swap_{pt}) \\
& \text{if } Z = W \\
& \Rightarrow \text{Protocol is unaffected} \\
& \text{if } Z \neq W \Rightarrow pos(Z) < pos(W) \\
& \Rightarrow Cost(P)_{cheat} > Cost(P)_{true} \\
& \text{or } X \notin Swap_{pt} \Rightarrow Swap_{pt} = Swap_{pc} \Rightarrow Z \in Swap_{pt} \\
& \Rightarrow pos(Z) < Pos(X) \\
& \Rightarrow Cost(P)_{cheat} > Cost(P)_{true}
\end{aligned}
$$

The +1 case, where the withheld point would have been swapped, P is keeping a point that it could have traded for a better point from Q, giving P a higher cost. If the withheld points would *not* have been swapped, there are two possibilities:

- Withheld points are further away (costlier) than the points swapped by Q. In this case, Q detects the cheating at the time of the swap.
- Withheld points are not further, but are not as close as alternative points In this case, though cheating is not detected, P party ends up swapping some points which are closer for it than some of the points it would have swapped had it not cheated, so P loses from cheating.

Case 1c: Alternatively, P can add "fake" points to the execution or the protocol, and then swap real points instead (swapping fake points would clearly be caught, when the honest party goes to pick up the non-existent load.) In either of these cases, the honest party will get points that are further than some it gave up, or the cheating will be detected.

Formally, assume that P adds a point X to the protocol, where $X \notin S_p$. Either $X \notin Swap_{pc}$ or $X \in Swap_{pc}$.

$$
\begin{aligned}
X \notin Swap_{pc} \quad \Rightarrow \quad & Swap_{pc} = Swap_{pt} \Rightarrow Swap_{qc} = Swap_{qt} \\
& \Rightarrow \text{Protocol is unaffected} \\
X \in Swap_{pc} \quad \Rightarrow \quad & \text{either } nswap_{cheat} = nswap_{true} + 1 \\
& \quad \Rightarrow \text{Cheating is detected} \\
& \text{or } nswap_{cheat} = nswap_{true} \\
& \Rightarrow \text{either } (\exists Y \in S_p | Y \in Swap_{pt} \wedge \text{Y substituted for X in } Swap_{pc}) \\
& \quad \Rightarrow \text{Protocol is unaffected} \\
& \quad \text{or } (\exists Y \in S_p | Y \notin Swap_{pt} \wedge \text{Y substituted for X in } Swap_{pc}) \\
& \quad \Rightarrow \text{Cheating is detected}
\end{aligned}
$$

Case 2a: If the protocol is aborted by either party, the other party immediately knows this and can demand an explanation. If the explanation is unsatisfactory, clearly the party is cheating.

Case 2b: The effect of lying at step 12 causes the two parties to exchange a suboptimal number of points. (i.e. $nswap_{cheat} \neq nswap_{true}$) If this number is more than the optimal number, this will be found out at the exchange phase and P will be caught. If the number is less than the optimal number, the only result is that both parties have a longer than optimal tour. This increases the cost for both parties (though possibly to an unequal extent), but since P will have a higher than optimal cost, and cannot know for certain if Q or itself will be hurt the most, there is no incentive for P to cheat.

$$\begin{aligned} nswap_{cheat} > nswap_{true} \Rightarrow\ & (\exists Y \in Swap_{pc} | Y \notin Swap_{pt} \wedge \\ & (\exists Z \in Swap_{qc} | Z \notin Swap_{qt} \wedge pos(Z) > pos(Y))) \\ & \Rightarrow \text{Cheating is detected} \\ nswap_{cheat} < nswap_{true} \Rightarrow\ & (\exists Y \in Swap_{pt} | Y \notin Swap_{pc} \wedge \\ & (\exists Z \in Swap_{qt} | Z \notin Swap_{qc} \wedge pos(Z) < pos(Y))) \\ & \Rightarrow Cost(P)_{cheat} > Cost(P)_{true} \end{aligned}$$

Combination of the above A cheating party can indulge in any combination of the above to confound the other parties. However, the cost effect of each individual cheating is additive; there is no interdependence on the penalties associated with cheating. Thus a combination of any of the above simply implies a combination of the individual penalties and therefore, the cheating party will either be caught, have a higher cost than by being honest, or the protocol will be unaffected.

Choice of direction: A party can successfully lie in steps 2-6 without detection. The effect is to enable P to force Q to conform to its choice for direction. This is equivalent to falsified input: a large number of fake points at the end of the scale in P's chosen direction. Q cannot distinguish this from the true case where P has these points; if the points were real they would never be swapped anyway. If Q party gets its choice of direction, or P really has more points, then the protocol is unaffected. However, if the protocol is affected, one party will end up with a higher cost than in an honest execution, the other with a lower cost. While P cannot know which party will gain, it can make a reasonable guess based on the distribution of its own points: If its points are already close together, then it is more likely to lose if it is forced to swap all of these and take the other direction. As has been

pointed out, this is a problem even if both parties are honest; allowing abort after step 6 is one solution.

2.4 Execution of SROD among Multiple Parties

Define the optimal result from the execution of SROD among multiple parties to be the perfect partitioning among all parties' one-dimensional datasets. In other words, any two of parties' ranges of their one-dimensional datasets are disjoint. This results in the minimal global tour (in one dimension).

Theorem 2.2. *Execution of SROD among k-parties eventually reaches optimal: k partitions (where $k \geq 2$.*

Proof. This is proven through strong induction on the number of parties.

Base case: $k = 2$. From the previous examples, the base case is obviously true.

Inductive step: Let integer $t > 2$, and assume the claim is true for all $k < t$. We need to prove that the claim is true when $k = t$.

Let $k = t - 1$. From the induction hypothesis, the execution among the k parties creates k partitions. Then let another party (say l) join the *k-parties* and i be any party among the *k-parties* or k partitions. If we leave i alone and execute the protocol among the rest of the k parties plus l. Then we have a new set of k partitions.

Considering i with the other parties, if the protocol is executed between i and any one of the parties whose dataset range not overlapping that of i, the protocol does nothing. Let p be the number of parties whose dataset ranges overlapping that of i.

If $p < k$, the execution of the protocol among these p parties plus i eventually leads to a set of disjoint partitions. These partitions plus the previously disjoint partitions with i consist $k + 1$ disjoint partitions.

If $p = k$, first we need to point out the fact that either the right or left most partition eventually belongs to the largest party (the one with the most points). During the execution among the $k + 1$ parties, if the largest party is selected with any party that overlap with it, the partition of the largest party will shrink. Since there are finite number of points for each party, the number of times the partition can shrink is finite. Therefore, eventually, no party's partition overlaps with that of the largest party. Then the execution among the rest of the parties will lead to k partitions. Adding the partition of the largest party, we have $k + 1$ disjoint partitions. As a result, the claim is true when $k = t$.

Because the base case and inductive step are true, the claim is true for all $k \geq 2$. □

3. Experimental Analysis

We evaluated this approach on a data set obtained from a trucking company representing loads during the first quarter in 2003. It contains information regarding when and where to pick up and deliver as well as full Truckload (TL) or Less Than Truckload (LTL). Since we view this approach as most likely for LTL sharing, we used only the LTL data. The data was separated into eleven data set with respect to the pickup week of each LTL data. We randomly assigned some pick-up points to company A and the rest to company B. Based on pick-up point, data in each week was divided into company A or B.

Within a given weekly data set, we modeled a single truck being used for all pick-up and then all delivery in turn, regardless of an exact pick-up and delivery date. Actual distance being dependent on the starting point of each company, we used the left-most pickup point in one dimension as the starting point of company A and the largest for B.

Before swapping the points, we obtained a pre-swap tour length (in two dimensions) of the pick-up and delivery for each company using the 2-opt algorithm (Lin & Kernighan 1973) to order the points. We also used the same optimization method to calculate the tour length after swapping.

One problem is that the algorithm picks *points* to swap, whereas the two-truck and different base assumption means that what we really need to swap is *loads*.[1] We considered three swapping methods for our analysis. One is Pair Swapping (PS), where we swap only if both pick-up and delivery points are beyond a certain point. Another is Average Swapping (AS), where companies swap data based on the average of pick-up and delivery points. As the number of delivery points is more than three times the number of pick-up points, we also consider Delivery Swapping (DS) based on the delivery point. Figure 9 gives the saving in total tour length by week. The result shows that DS performs best, because the distance in our data set depends mainly on the delivery points. This long tour thus has the most opportunity for improvement. Note that Pair Swapping sometimes gives a worse tour length – since this effectively withholds data points from the swapping, it demonstrates the cost of not participating honestly in the protocol. Although none of our swapping method

[1]Point-swapping is a correct model if multiple trucks return to a central warehouse, such as the model used by parcel carriers; the purpose of the experiment is to evaluate the effectiveness of the approach under more difficult conditions.

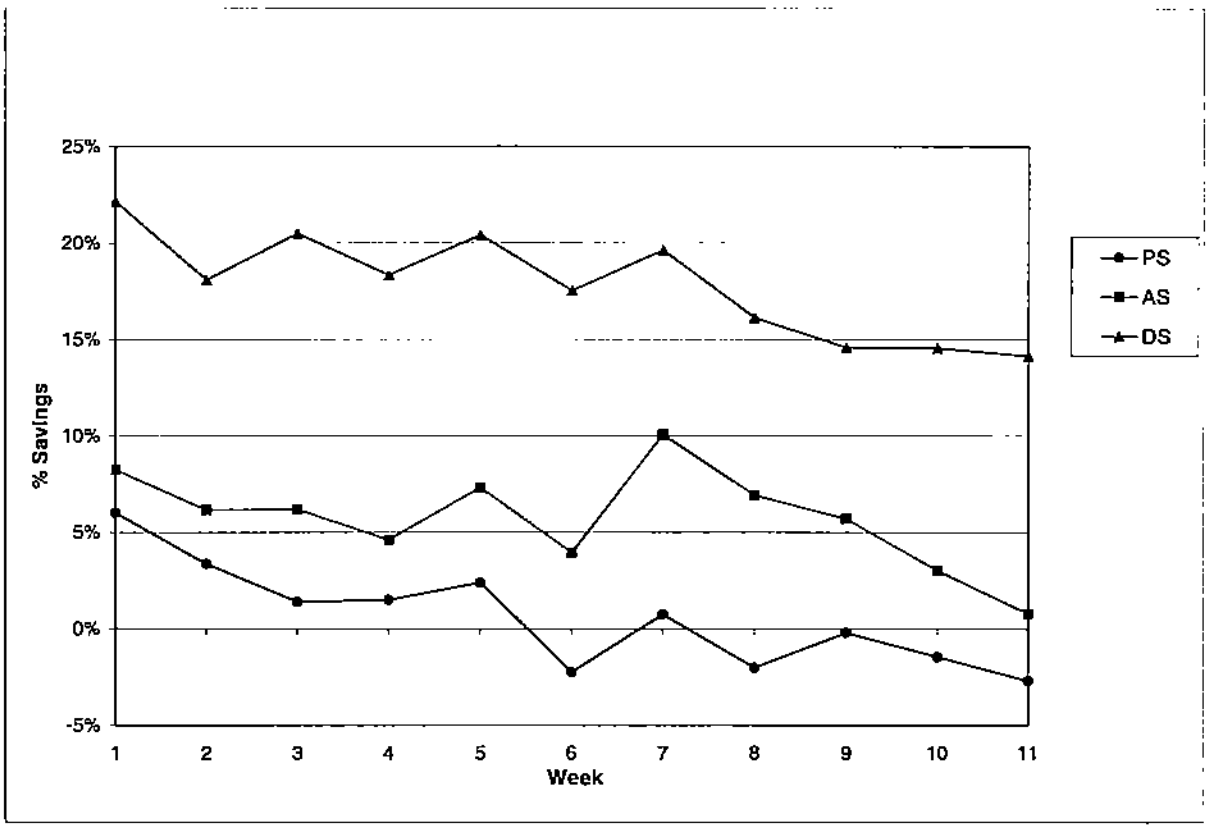

Figure 9: Savings in Total Distance

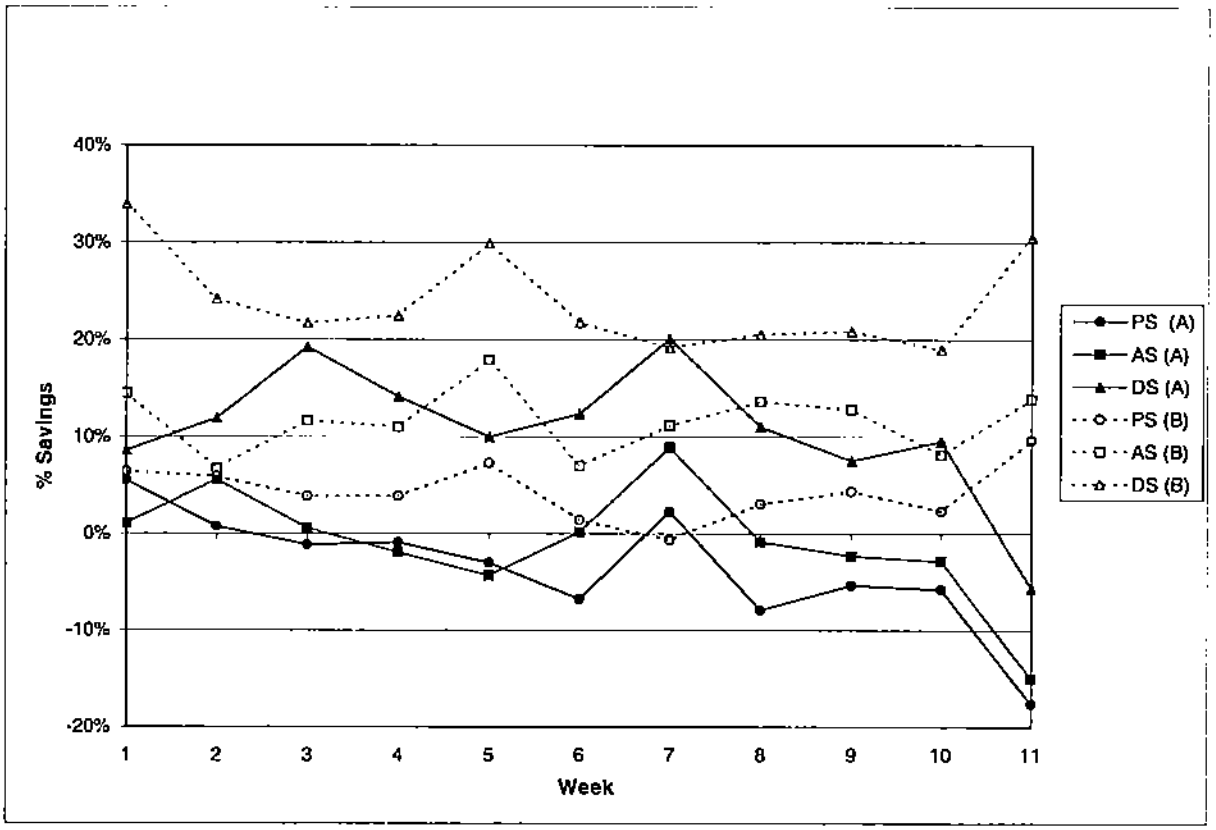

Figure 10: Savings in Individual Distance

guarantees positive savings, the average savings in total distance are 0.6% for PS, 5.7% for AS and 17.8% for DS.

Figure 10 give the savings by party – again it is clear that destination swapping, by optimizing the longest tour, gives the best results. It also shows that it is *possible* for a party to get a longer tour, even when the global tour length decreases.

3.1 Empirical Results of N party collaboration

We now consider the observed empirical effect of running the algorithm described in the previous section but for a dataset where we have (a) Individual points in 2 dimensions, (b) pickup and drop off pairs associated with each shipment. The goal of this run is to identify how much of the possible improvements can be generated by the algorithm defined and how that is affected by the number N of transport companies involved.

We also used the first week's data from the dataset described earlier, but ignored the pick-up and delivery pair for each data set. At first, we randomly allocated the same number

of points to each company, named $1, 2, \ldots, n$. These points are converted to one-dimensional data using SF. For every pair of company i and j where $i \neq j$, we applied the algorithm and repeated until equilibrium was reached. After repeating this for 20 initial random allocations, the average savings in total distance and individual distance are shown in Table 1 for several number of participants. The results are summarized in Figure 11. Note that

Table 1: Savings for multiple parties

n	Distance Before Swapping		Distance After Swapping		Individual Savings (%)		Total Savings (%)
2	1	412.4	1	311.8	1	24.4%	27.7%
	2	399.3	2	275.8	2	30.9%	
3	1	332.6	1	255.4	1	23.2%	39.9%
	2	331	2	130.9	2	60.5%	
	3	333.1	3	213.9	3	35.8%	
4	1	283.3	1	214	1	24.5%	45.2%
	2	288.2	2	117.9	2	59.1%	
	3	298.6	3	117.4	3	60.7%	
	4	292	4	189.1	4	35.2%	
5	1	264.7	1	192	1	27.5%	48.5%
	2	263.1	2	113.4	2	56.9%	
	3	264.6	3	72.2	3	72.7%	
	4	263.8	4	149.2	4	43.4%	
	5	263.8	5	155.9	5	40.9%	
6	1	244.5	1	182	1	25.5%	51.4%
	2	243.3	2	107.5	2	55.8%	
	3	256.6	3	62.3	3	75.7%	
	4	240.8	4	71.4	4	70.4%	
	5	244.5	5	145.7	5	40.4%	
	6	239.8	6	147.7	6	38.4%	
10	(Sum) 1986.6		(Sum) 792.3				60.1%
15	(Sum) 2534.6		(Sum) 847.1				66.6%
20	(Sum) 3097.4		(Sum) 856.2				72.4%

as the number of parties increase, the savings increase. This is to be expected, as the size of each partition shrinks as the number of parties grows, whereas the expected tour length of a random distribution of points changes little.

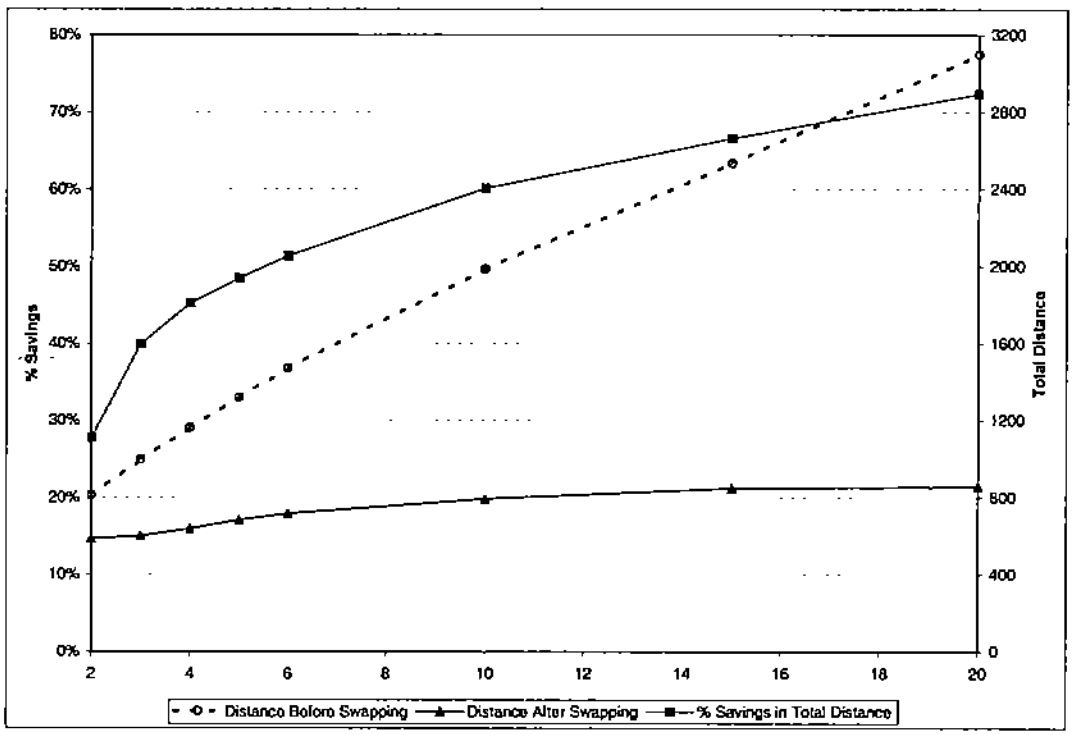

Figure 11: Savings for different number of parties

4. Is the approach pro-competitive?

While we have shown that the method will generally result in an overall cost savings for the hauler, can we expect that some of these cost savings will accrue to the shipper?

What does it mean to say that this collaboration is pro-competitive?

- Decentralized operation of the algorithm generates the same solution for the shipper as centralized operation.
- As N increases, the efficiency increases overall i.e., a viral effect.
- Individual transport companies cannot guess which set of points they would get hence reducing the potential for collusionary pricing.

Consider a set of N independent companies each with m points. Assume that the m points for each company are uniformly distributed along the line. As described in Section 2.4, repeated runs of the protocol will result in continuous improvement in tour length, with each company retaining m points. Eventually equilibrium is reached, with a unique partition of the line into N segments. Thus the decentralized operation of the algorithm generates the same solution as the efficient algorithm offered by a central decision maker.

Note also that as N increases, there is a greater ability to cluster sets of points close together. Thus every possible transport company has the potential to both decrease its own costs as well as decrease costs for all other companies. In that respect, the algorithm presents an open system for potential collaboration and decreases the incentives for collusion.

Now consider how an individual transport company would bid for a supplier's work knowing the existence of the opportunity for collaboration. Given the results described above, an individual transport company cannot guess which segment or which set of points it will end up with. In the absence of this certainty, an individual company is faced with some probability of cost reduction through swaps and some probabilistic description of offered load after swaps. This decreases the ability for individual transport companies to collude to influence shipper prices. We do not explore the price consequences in this paper - we leave such exploration to future research.

5. Conclusion / Future Work

We have described an algorithm that enables independent companies to identify opportunities for collaboration without sharing unnecessary data. The theoretical analysis shows that the algorithm ensures that no information can be inferred except that from the shared data, that truthtelling is incentive compatible and that features of the algorithm make it procompetitive. In addition, application of the algorithm to an empirical dataset indicate a substantial potential impact.

The specific problem described in this paper is a first step towards a greater integration of data encryption techniques and theory along with problems in operations management. An interesting open research question concerns the potential tradeoff between the use of a heuristic approach to problem solution that admits a tight security property vs a closer to optimal algorithm that permits data leakage. We leave such exploration to future research.

A. Secure Multi-party Computation

Substantial work has been done on secure multi-party computation. The key result is that a wide class of computations can be done securely under reasonable assumptions. Any function that can be represented by a polynomial circuit in terms of the number of input bits can be evaluated in reasonable time. We give a brief overview of this work, concentrating on material that is used in the paper. The definitions given here are from Goldreich (1998). For simplicity, we concentrate on the two party case. Extending the definitions to the multi-party case is straightforward.

A.1 Security in the Semi-Honest Model

A semi-honest party follows the rules of the protocol using its correct input, but is free to later use what it sees during execution of the protocol to compromise security. This is somewhat realistic in the real world because parties who want to find out results for their mutual benefit will follow the protocol to get correct results. Also a protocol that is buried in large, complex software can not be easily altered.

A formal definition of private two party computation in the semi-honest model is given below. Computing a function privately is equivalent to computing it securely. The formal proof of this can be found in Goldreich (1998).

Definition A.1. (privacy w.r.t. semi-honest behavior):(Goldreich 1998)

Let $f : \{0,1\}^* \times \{0,1\}^* \longmapsto \{0,1\}^* \times \{0,1\}^*$ be probabilistic, polynomial-time functionality, where $f_1(x,y)$(resp., $f_2(x,y)$) denotes the first (resp., second) element of $f(x,y)$) and let Π be two-party protocol for computing f.

Let the view of the first (resp. second) party during an execution of Π on (x,y), denoted $view_1^{\Pi}(x,y)$ (resp., $view_2^{\Pi}(x,y)$) is $(x, r_1, m_1, \ldots, m_t)$ (resp., $(y, r_2, m_1, \ldots, m_t)$) where r_1 represent the outcome of the first (resp., r_2 second) party's internal coin tosses, and m_i represent the i^{th} message it has received.

The output of the first (resp., second) party during an execution of Π on (x,y) is denoted $output_1^{\Pi}(x,y)$ (resp., $output_2^{\Pi}(x,y)$) and is implicit in the party's view of the execution.

Π privately computes f if there exist probabilistic polynomial time algorithms, denoted S_1, S_2 such that

$$\{(S_1(x, f_1(x,y)), f_2(x,y))\}_{x,y\in\{0,1\}^*} \equiv^C \left\{\left(view_1^{\Pi}(x,y), output_2^{\Pi}(x,y)\right)\right\}_{x,y\in\{0,1\}^*} \quad (1)$$

$$\{(f_1(x,y), S_2(x, f_1(x,y)))\}_{x,y\in\{0,1\}^*} \equiv^C \left\{\left(output_1^{\Pi}(x,y), view_2^{\Pi}(x,y)\right)\right\}_{x,y\in\{0,1\}^*} \quad (2)$$

where $\equiv^C$ denotes computational indistinguishability.

The above definition says that a computation is secure if the view of each party during the execution of the protocol can be effectively simulated by the input and the output of the party. This is not quite the same as saying that private information is protected. For example, assume two parties use a secure protocol to compare two positive integers. If A has 2 as its integer and the comparison result indicates that 2 is bigger than equal other site's integer, A can conclude that B has 1 as its input. Site A can deduce this information by solely looking at its local input and the final result - the disclosure is a fault of the problem being solved and not the protocol used to solve it. To prove a protocol is secure, we have to show that anything seen is not giving more information then seeing the final result.

In summary, secure multi-party protocol will not reveal more information to a particular party than the information that can be induced by looking at that party's input and the output.

A.2 Yao's general two party secure function evaluation

Yao's general secure two party evaluation is based on expressing the function $f(x,y)$ as a circuit and encrypting the gates for secure evaluation(Yao 1986). With this protocol any

two party function can be evaluated securely in the semi-honest model, but to be efficiently evaluated the function must have a small circuit representation. To show how secure circuit evaluation is done without going into too much detail, we will present a simple approach here. In real life application where performance is an important issue, other approaches must be used. For comparing any two integers securely, Yao's generic method is one of the most efficient methods known, although other asymptotically equivalent but practically more efficient algorithms could be used as well(Ioannidis & Grama 2003).

The circuit evaluation techniques depends on 1 out of 4 oblivious transfer. Oblivious transfer was described in Section A.2.4. Details of all these techniques can be found in Goldreich (1998).

A.2.1 Evaluating the Logical Gates Securely

It is known fact that every circuit can be built by only using XOR($\oplus$) and AND($\cdot$)gates. If we have a mechanism to evaluate these gates securely and combine the results securely, we will be able to compute any circuit securely.

It must be clear that if we do not want to reveal more than the final result, we must not be revealing anything during the intermediate evaluations. Let assume that after the evaluation of each logic gate, the result of the logic gate divided into two random shares, such that $r = r_1 \oplus r_2$ where r is the result. If Alice has the r_1 and Bob has the r_2, both of them will not be able to predict the result(other part can have anything as the share). The problem is that the output of this particular gate is an input to an another gate, therefore in the evaluation of the next gate, we have to combine r_1 and r_2 privately and again output the random shares of the result. Clearly this process can be repeated to evaluate the entire circuit. Now we will describe how to achieve this for each type of gate.

A.2.2 Evaluating an XOR gate securely

Given the random shares of each input wire, we would like to create random shares of the result. Assume that Alice's part of each random share is (a_1, a_2) and Bob has (b_1, b_2). We would like to create a_r, b_r such that $(a_1 \oplus b_1) \oplus (a_2 \oplus b_2) = (a_r \oplus b_r)$. For XOR gate it is easy to achieve this goal, just set $a_r = (a_1 \oplus a_2)$ and $b_r = (b_1 \oplus b_2)$. This procedure is secure because each party can evaluate its share without exchanging any information with the other party, therefore other party cannot learn anything. Note that XOR of a_r, b_r, will give the result of the gate. Since the initial shares are random, final shares are also random.

Table 2: AND gate evaluation

$(0,1)$	$(0,1)$	$(1,0)$	$(1,1)$
$(a_1 \cdot a_2) \oplus r$	$(a_1 \cdot \bar{a}_2) \oplus r$	$(\bar{a}_1 \cdot a_2) \oplus r$	$\bar{a}_1 \cdot \bar{a}_2) \oplus r$

A.2.3 Evaluating an AND gate securely

Solution for the XOR gate was trivial because we were able use the commutative property of the XOR function. For evaluating the AND gate, we will have to use the oblivious transfer tool. Again Alice's part of each random share is (a_1, a_2) and Bob has (b_1, b_2). Now Alice and Bob want to evaluate $(a_1 \oplus b_1) \cdot (a_2 \oplus b_2) = (a_r \oplus b_r)$. Since Alice knows its inputs, it can evaluate the result of the gate for each of the 4 possibilities. For example, if Bob has (0,0) as its input, the result of the gate will be $a_1 \cdot a_2$. Table A.2.3 shows the creation of such table, one important thing to note is to r. This r will be the random share of the Alice (a_r). In other words, Alice creates its own random share. Now using the 1 out of 4 oblivious transfer, Bob can learn its random share. Assume that Bob has (1,0),then Bob will get only the third element which is $(\bar{a}_1 \cdot a_2) \oplus r$. Clearly $a_r \oplus b_r = \bar{a}_1 \cdot a_2$, the required result.

A.2.4 Combining Everything

With the help of previous constructs, given the random shares of each input wire, we can construct the random shares of the output of each gate. At the start of the evaluation, each party can xor its inputs with some random number and send the random number to other site to create the initial random shares. After this step above procedures for evaluating gates can be used until we get the final result.

Note that constant depth and logarithmic size circuits are known for comparison. Since each gate is evaluated in constant time, this shows that it is possible to avoid the efficiency issues posed by direct oblivious transfer described in Section .

A.3 Extension to Malicious Model

In the semi-honest model, we require that each party follow the protocol exactly. This assumption makes it is easy to develop algorithms for semi-honest model. Although the semi-honest model can be a reasonable in many cases, clearly a solution for parties that do not follow the protocols (i.e., showing malicious behavior) is desirable.

It can be shown that any semi-honest algorithm can be modified to work in malicious model. This can be achieved by using zero knowledge proofs and commitment schemes. The key is making the oblivious transfer protocol proof against a malicious party (as shown in Section 1.2); the rest follows easily.

Acknowledgments

We thank Reha Uzsoy for contributions made to the formulation of this problem in initial discussions of the work.

This work supported by a grant from the Purdue University's e-Enterprise Center at Discovery Park.

References

Bandler, James. 2003. "How Seagoing Chemical Haulers May Have Tried to Divide Market." The Wall Street Journal.

Bartholdi III, John J. & Loren K. Platzman. 1982. "An O(N log N) planar travelling salesman heuristic based on spacefilling curves." *Operations Research Letters* pp. 121–125.

Bartholdi III, John J. & Loren K. Platzman. 1988. "Heuristics Based on Spacefilling Curves for Combinatorial Problems in Euclidean Space." *Management Science* 34(3):157–160.

Goldreich, O., S. Micali & A. Wigderson. 1987. How to Play any Mental Game - A Completeness Theorem for Protocols with Honest Majority. In *19th ACM Symposium on the Theory of Computing*. pp. 218–229.
*http://doi.acm.org/10.1145/28395.28420

Goldreich, Oded. 1998. "Secure Multi-Party Computation.". (working draft).
*http://www.wisdom.weizmann.ac.il/ oded/pp.html

Ioannidis, Ioannis & Ananth Grama. 2003. An Efficient Protocol for Yao's Millionaires' Problem. In *Hawaii International Conference on System Sciences (HICSS-36)*. Waikoloa Village, Hawaii: .

Kalagnanam, J., M. Trumbo & H. S. Lee. 2000. "The Surplus Inventory Matching Problem in the Process Industry." *Operations Research* 48(4).

Lin, S. & Brian W. Kernighan. 1973. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem." *Operations Research* 21(2):498–516.

Naor, Moni & Benny Pinkas. 1999. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing.* Atlanta, Georgia, United States: ACM Press pp. 245–254.

Naor, Moni & Benny Pinkas. 2001. Efficient Oblivious Transfer Protocols. In *Proceedings of SODA 2001 (SIAM Symposium on Discrete Algorithms).* Washington, D.C.: .

Yao, Andrew C. 1986. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science.* IEEE pp. 162–167.