3

Privacy-Preserving Local Search for the Traveling Salesman Problem

Jun Sakuma[1] and Shigenobu Kobayashi[2]
[1]*University of Tsukuba*
[2]*Tokyo Institute of Technology*
Japan

1. Introduction

In this chapter, we specifically examine distributed traveling salesman problems in which the cost function is defined by information distributed among two or more parties. Moreover, the information is desired to be kept private from others.

As intuitive situations in which distributed private information appears in combinatorial optimization problems, we take problems in supply chain management (SCM) as examples. In SCM, the delivery route decision, the production scheduling and the procurement planning are fundamental problems. Solving these problems contributes to improvement of the correspondence speed to the customer and shortening the cycle time(Vollmann, 2005; Handfield & Nichols, 1999). In the process of forming the delivery route decision and production schedule decision, the combinatorial optimization plays an important role.

When the SCM is developed between two or more enterprises, information related to the stock, the production schedule, and the demand forecast must be shared among enterprises. Electronic Data Interchange (EDI), the standardized data exchange format over the network, is often used to support convenient and prompt information sharing[1]. Information sharing apparently enhances the SCM availability; however, all information related to the problem resolution must be disclosed to all participants to lay the basis for global optimization. Such information is often highly confidential and its disclosure would be impossible in many cases. As more concrete examples, two scenarios are presented. These scenarios pose situations that appear to be unsolvable unless private information is shared.

Scenario 1: Let there be a server E_A that manages a route-optimization service and a user E_B who tries to use this service. The user's objective is to find the optimal route that visits points $F_1,...,F_n$ chosen by himself. The user, however, does not like to reveal the list of visiting points to the server. The server manages a matrix of cost for traveling between any two points. The server does not like to reveal the cost matrix to the user, either. How can the user learn the optimal route without mutually revelation of private information?

Note that this problem is obviously solved as the Traveling Salesman Problem (TSP) if either of traveling cost or visiting points is shared. As more complicated examples, a multi-party situation is described next.

Scenario 2: Let there be two shipping companies E_A and E_B in two regions A and B. Client E_C requests that E_A deliver freight to point $F_1^A,...,F_n^A$ in region A and also requests E_B to deliver

[1]United Nations Economic Commission for Europe, http://www.unece.org/trade/untdid/welcome.htm

freight to point $F_1^B,...,F_n^B$ in region B, separately. Now E_A,E_B and E_C are involved in the business cooperation and try to unify the delivery route to reduce the overall delivery cost. To search for the optimum unified route and to estimate the reduced cost, E_A and E_B must reveal their costs between any two points, but they would not reveal their delivery cost because they seek mutual confidentiality. How can these companies search for the optimal unified delivery route without revealing their confidential information?

In these scenarios, costs and visiting points are confidential information and participants would not reveal them. As shown, the difficulty of private information sharing sometimes thwarts problem resolution.

In our study, we specifically investigate a privacy-preserving local search to solve the traveling salesman problem. The easiest way to converting existing metaheuristics to privacy-preserving metaheuristics is to introduce an entity called a trusted third party (TTP). A TTP is an entity that facilitates interactions between two parties who both trust the TTP. If a TTP exists, then all parties can send their private information to the TTP; the TTP can find a local optimum using the existing metaheuristics and can return the optimized solution.

This idea works perfectly. However, preparation of a TTP is often quite difficult mainly in terms of cost. Needless to say, a protocol that works only between participants in the standard network environment (e.g. TCP/IP network) is preferred.

Secure function evaluation (SFE) (Yao, 1986; Goldreich, 2004) is a general and well studied methodology for evaluating any function privately, which allows us to convert any existing metaheuristics into privacy-preserving metaheuristics. However, the computational cost of SFE is usually quite large. The time complexity of SFE is asymptotically bounded by the polynomial of the size of the Boolean circuit of the computation. If the computation is primitive, SFE works practically; however, it can be too inefficient for practical use, particular when the large-scale computation is performed or large amount of datasets are taken as inputs and outputs.

In solving the traveling salesman problem by means of metaheuristics, not only the input size but the number of iterations can be quite large. Therefore, in our protocol, in order to solve TSP in a privacy-preserving manner, we make use of a public-key cryptosystem with homomorphic property, which allows us to compute addition of encrypted integers without decryption. Existing SFE solutions are used only for small portions of our computation as a part of a more efficient overall solution.

2. Problem definition

In this section, we introduce distributed situations of the traveling salesman problem (TSP). Then the privacy in the distributed traveling salesman problem is defined.

Let $G = (V,E)$ be an undirected graph and $|V| = n$ be the number of cities. For each edge $e_{i,j} \in E$, a cost connecting node i and node j, $\alpha_{i,j}$, is prescribed. Tours are constrained to be a Hamilton cycle. Then the objective of TSP is to find a tour such that the sum of the cost of included edges is as low as possible.

The permutation representation or the edge representation is often used to describe tours. In this chapter, we introduce the scalar product representation with indicator variables for our solution. Let $x = (x_{1,2},...,x_{1,n},x_{2,3},...,x_{2,n},...,x_{n-1,n})$ be a tour vector where $x_{i,j}$ are indicator variables such that

$$x_{i,j} = \begin{cases} 1 & e_{i,j} \text{ is included in the tour,} \\ 0 & \text{otherwise.} \end{cases}$$

The cost can be written as an instance vector $\alpha = (\alpha_{1,2},...,\alpha_{n-1,n})$ similarly. The number of elements of the tour vector x and the cost vector α are $d = n(n-1)/2$. For simplicity, we respectively describe the i-th element of x and α as x_i and α_i. Then, using this representation, the objective function of TSP is written in the form of the scalar product:

$$f(x,\alpha) = \sum_{i=1}^{d} \alpha_i x_i = \alpha \cdot x. \tag{1}$$

The constraint function of the TSP is defined as

$$g(x;V) = \begin{cases} 1 & \text{If } x \text{ is a Hamilton cycle of } V, \\ 0 & \text{otherwise.} \end{cases}$$

Next, we consider distributed situations of the TSP. The instance of the TSP consists of city set V and cost vector α.
First, the simplest and typical two-party distributed situation is explained. Let there be two parties $P(1)$ and $P(2)$. Assume that city set V is publicly shared by $P(1)$ and $P(2)$. In such a case, $P(1)$ (referred to as *searcher*) arbitrarily chooses a city subset $V' \subseteq V$ and privately holds it. Here, V' represents the searcher's private list of visiting cities. The searcher can generate tour x that includes all cities in V' and aims to minimize the total cost of the tour.
In addition, $P(2)$ (referred to as *server*) privately holds cost vector α for all cities in V. The server works to support the optimization of the searcher.
We call this problem $(1,1)$-TSP or *one-server one-searcher TSP*. Here, $(1,1)$-TSP corresponds to the formal description of scenario 1 described in section 1.
Multi-party cases are explained as the extension of $(1,1)$-TSP. Assume a situation in which the cost vector α is distributed among k servers. Let $\alpha(i)$ be a vector that is owned by the i-th server such that $\alpha = \sum_{i=1}^{k} \alpha(i)$. As in the case of $(1,1)$-TSP, V' is chosen by the searcher. We designate this distributed TSP as $(k,1)$-TSP, which corresponds to the formal description of scenario 2 presented in section 1.
Next we explain $(1,k)$-TSP. Let $\{V'(1),...,V'(k)\}$ be city subsets that are chosen independently by k searchers independently. Let $V' = \cup_{i=1}^{k} V'(i)$. The server privately manages α.
See Fig. 1 for the partitioning patterns of these distributed TSPs. Apparently, $(1,1)$-TSP is a special case of these cases. The cost function is represented as the scalar product of two vectors in all situations. The constraint function is written as $g(x;V')$, which is evaluable by the searcher in $(1,1)$ or (k,1)-TSP . However, $g(x;V')$ cannot be evaluated by any single party in $(1,k)$-TSP.
In our protocol presented in latter sections, we require that constraint function g is evaluable by a single party. For this reason, we specifically investigate $(1,1)$-TSP and $(k,1)$-TSP in what follows.

3. Our approach

In this section, we explain our approach for solving distributed TPSs with private information by means of the local search. For the convenience of description, we specifically examine $(1,1)$-TSP in the following sections. The extension to $(k,1)$-TSP is mentioned in Section 6.
Let $N(x)$ be a set of neighborhoods of solution x. Let $\in_r$ denote an operation which chooses an element form a given set uniformly at random. Then, the algorithm of local search without privacy preservation is described as follows:

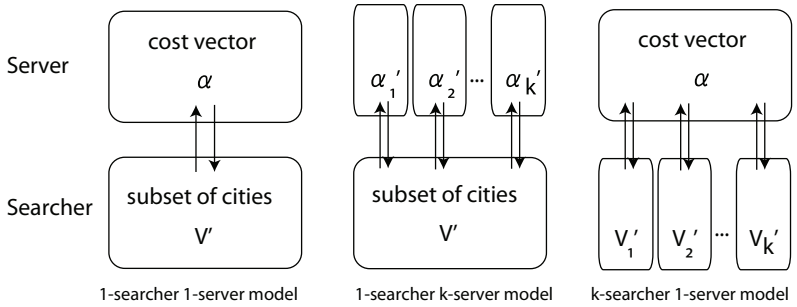


Fig. 1. Privacy model of the distributed TSP

[Local search]

1. Generate an initial tour x_0
2. $x \in_r N(x_0), N(x_0) \leftarrow N(x_0) \setminus \{x\}$.
3. If $f(x) < f(x_0), x_0 \leftarrow x$
4. If some termination conditions are satisfied, output x_0 as x^*. Else, go to step 2.

In the process of local search, cost values are evaluated many times. From the perspective of privacy preservation, if cost values are shared at each iteration, information leakage is likely to arise. For example, in $(1,1)$-TSP, the searcher might infer some elements of the server's private distance vector from a series of tours and cost values.
Fortunately, for many rank-based metaheuristics algorithms, including local search, cost values need not always be evaluated; the evaluation of a paired comparison of two cost values is sometimes sufficient. This fact is convenient for privacy preservation in optimization because the risk of information leakage from the result of paired comparison would be much smaller than the cost value itself.
Considering the matters described above, we consider a protocol that solves privacy-preserving optimization through a combination of local search and a cryptographic protocol that privately compares a pair of scalar products.
First, we define the private paired comparison of the scalar product. Let $x_1, x_2, \alpha \in Z_m^d (= [0, ..., m-1]^d)$. Also assume the the following inequalities.

– $\alpha \cdot x_2 - \alpha \cdot x_1 \geq 0$ be I_+
– $\alpha \cdot x_2 - \alpha \cdot x_1 < 0$ be I_-

Then, the problem can be stated as follows:

Statement 1 *(Private scalar product comparison) Let there be two parties: Alice and Bob. Alice has two private vectors x_1, x_2 and Bob has a private vector α. At the end of the protocol, Alice learns one correct inequality in $\{I_-, I_+\}$ and nothing else. Bob learns nothing.*

We call this problem *private scalar product comparison*. Assuming that there exist protocols that solve this private scalar product comparison, private scalar product comparison allows us to perform local search in a privacy-preserving manner as shown below:
In step one, the searcher generates an initial tour x_0. In step two, the searcher chooses a tour in neighborhood of x_0, $N(x_0)$, uniformly at random. These two steps can be executed by the

searcher solely. At step three, we make use of private scalar product comparison. Recall that $f(x) = \alpha \cdot x$ and $f(x_0) = \alpha \cdot x_0$. What we need here is to learn whether or not $\alpha \cdot x_0 - \alpha \cdot x < 0$. This problem is readily solved by means of private scalar product comparison without sharing the searcher's x, x_0 and the server's α. In step four, the searcher can terminate the computations after generating a prescribed number of individuals, or can terminate the serarch when he finds $N(x) = \emptyset$. In both cases, step four can be executed by the searcher solely.
As shown, assuming the existence of a protocol for private scalar product comparison, TSPs including private information can be securely solved. In next section, we introduce cryptographic building blocks required for solving private scalar product comparison. Then in Section 5, a protocol for solving the private scalar product comparison is presented.

4. Cryptographic building blocks

In this section, three cryptographic building blocks are introduced.

4.1 Homomorphic public-key cryptosystem

For our protocol, we use a public-key cryptosystem with a homomorphic property. A public-key cryptosystem is a triple (Gen, Enc, Dec) of probabilistic polynomial-time algorithm for key-generation, encryption, and decryption, respectively. The key generation algorithm generates a valid pair (s_k, p_k) of secret and public keys. The secret key and public key are used only for decryption and encryption. Then $Z_p = \{0, 1, ..., p-1\}$ denotes the plain text space. The encryption of a plain text $t \in Z_p$ is denoted as $\mathrm{Enc}_{p_k}(t;r)$, where r is a random integer. The decryption of a cipher text is denoted as $t = \mathrm{Dec}_{s_k}(c)$. Given a valid key pair (p_k, s_k), $\mathrm{Dec}_{s_k}(\mathrm{Enc}_{p_k}(t;r)) = t$ for any t and r is required.
A public key cryptosystem with additive homomorphic property satisfies the following identities.

$$\begin{aligned} \mathrm{Enc}(t_1;r_1) \cdot \mathrm{Enc}(t_2;r_2) &= \mathrm{Enc}(t_1 + t_2 \mod p; r_1 + r_2) \\ \mathrm{Enc}(t_1;r_1)^{t_2} &= \mathrm{Enc}(t_1 t_2 \mod p; r_1) \end{aligned}$$

In those equations, $t_1, t_2 \in Z_p$ are plain texts and r_1, r_2 are random numbers. These random numbers are used to introduce redundancy into ciphers for security reasons; encrypted values of an integer with taking difference random numbers are represented differently. These properties enable the addition of any two encrypted integers and the multiplication of an encrypted integer by an integer. A public-key cryptosystem is *semantically secure* when a probabilistic polynomial-time adversary cannot distinguish between random encryptions of two elements chosen by herself. Paillier cryptosystem is known as a semantically secure cryptosystem with homomorphic property(Pailler, 1999). We use the Paillier cryptosystem in experiments in section 6.

4.2 Secure function evaluation

As mentioned in the introductory section, secure function evaluation (SFE) is a general and well studied cryptographic primitive which allows two or more parties to evaluate a specified function of their inputs without revealing (anything else about) their inputs to each other (Goldreich, 2004; Yao, 1986).
In principle, any private distributed computation can be securely evaluated by means of SFE. However, although polynomoially bounded, naive implementation of local search using SFE can be too inefficient. Therefore, in order to achieve privacy-preserving local search efficiently,

we make use of existing SFE solutions for small portions of our computation as a part of a more efficient overall solution.

4.3 Random shares

Let $x = (x_1,...,x_d) \in Z_N^d$. When we say A and B have *random shares* of x, A has $x^A = (x_1^A,...,x_d^A)$ and B has $x^B = (x_1^B,...,x_d^B)$ in which x_i^A and x_i^B are uniform randomly distributed in $\mathbb{Z}_N$ with satisfying $x_i = (x_i^A + x_i^B) \mod N$ for all i. Random shares allow us to keep a private value between two parties without knowing the value itself. Note that if one of the party pass the share to the other, the private value is readily recovered.

5. Scalar product comparison

Before describing the protocol for scalar product comparison, we introduce a protocol which privately computes scalar products from privately distributed vectors. Goethals et al. proposed a protocol to compute scalar products of two distributed private vectors without revealing them by means of the homomorphic public-key cryptosystem (Goethals et al., 2004). For preserving the protocol generality, parties are described as Alice and Bob in this section. The problem of private scalar product is stated as follows:

Statement 2 *(Private scalar product) Let there be two parties: Alice and Bob. Alice has a private vector $x \in Z_\mu^d$; Bob also has a private vector $\alpha \in Z_\mu^d$. At the end of the protocol, both Alice and Bob learn random shares of scalar product $x \cdot \alpha$ and nothing else.*

Let Z_p be the message space for some large p. Set $\mu = \lfloor \sqrt{p/d} \rfloor$. In what follows, the random number used in encryption function Enc is omitted for simplicity. Then, the protocol is described as follows.

[Private scalar product protocol]

- Private Input of Alice: $\alpha \in Z_\mu^d$
- Private Input of Bob: $x \in Z_\mu^d$
- Output of Alice and Bob: $r_A + r_B = x \cdot \alpha \mod p$
  (Alice and Bob output r_A and r_B, respectively)

1. Alice: Generate a public and secret key pair (p_k, s_k).
2. Alice: For $i = 1,...,d$, compute $c_i = \mathrm{Enc}_{p_k}(\alpha_i)$ and send them to Bob.
3. Bob: Compute $w \leftarrow (\prod_{i=1}^d c_i^{x_i}) \cdot \mathrm{Enc}_{p_k}(-r_B)$ where $r_B \in_r Z_p$ and send w to Alice.
4. Alice: Compute $Dec(w) = r_A (= x \cdot \alpha - r_B)$

In step two, Alice sends the ciphertext of her private vector $(c_1,...,c_d)$ to Bob. Bob does not possess the secret key. Therefore, he cannot learn Alice's vector from received ciphertexts. However, in step three, he can compute the encrypted scalar product based on homomorphic

properties without knowing Alice's x:

$$\begin{aligned} w &= (\prod_{i=1}^{d} c_i^{x_i}) \cdot \mathrm{Enc}_{p_k}(-r_B) = (\prod_{i=1}^{d} \mathrm{Enc}_{p_k}(\alpha_i)^{x_i}) \cdot \mathrm{Enc}_{p_k}(-r_B) \\ &= \mathrm{Enc}_{p_k}(\alpha_1 x_1) \cdots \mathrm{Enc}_{p_k}(\alpha_d x_d) \cdot \mathrm{Enc}_{p_k}(-r_B) \\ &= \mathrm{Enc}_{p_k}(\sum_{i=1}^{d} \alpha_i x_i - r_B) = \mathrm{Enc}_{p_k}(\alpha \cdot x - r_B) \end{aligned}$$

Then, in step four, Alice correctly obtains a random share of $\alpha \cdot x$ by decrypting w using her secret key: Bob's r_B is a random share of $\alpha \cdot x$, too. Assuming that Alice and Bob behave semi-honestly[2], it can be proved that the scalar product protocol is secure (Goethals et al., 2004). In the discussions of subsequent sections, we assume that all parties behave as semi-honest parties.

A protocol for the private scalar product comparison appears to be obtained readily using the private scalar product protocol. The difference of two scalar products $\alpha \cdot x_2 - \alpha \cdot x_1$ can be computed as

$$\prod_{i=1}^{d} c_i^{x_{2,i}} \cdot \prod_{i=1}^{d} c_i^{-x_{1,i}} = \mathrm{Enc}_{p_k}(\alpha \cdot x_2 - \alpha \cdot x_1). \tag{2}$$

By sending this to Alice, Alice learns $\alpha \cdot x_2 - \alpha \cdot x_1$. Equation 2 appears to compare two scalar products successfully and privately. However, it is not secure based on statement 1 because not only the comparison result but also the value of $\alpha \cdot x_1 - \alpha \cdot x_2$ is known to Alice. In the case of the TSP, tour vectors are $x_1, x_2 \in \{0,1\}^d$. Therefore, Bob's x_1 and x_2 are readily enumerated from the value of $\alpha \cdot x_2 - \alpha \cdot x_1$ by Alice. To block Alice's enumeration, Bob can multiply some positive random value r_B to the difference of two scalar products,

$$\prod_{i=1}^{d} c_i^{r_B x_{2,i}} \cdot \prod_{i=1}^{d} c_i^{-r_B x_{1,i}} = \mathrm{Enc}_{p_k}(r_B(\alpha \cdot x_2 - \alpha \cdot x_1)).$$

By sending this to Alice, Alice learns $r_B(\alpha \cdot x_2 - \alpha \cdot x_1)$. Since $r_B > 0$, Alice can know whether or not $\alpha \cdot x_2 > \alpha \cdot x_1$ from this randomized value; however, this is not secure, either. r_B is a divider of $r_B(\alpha \cdot x_2 - \alpha \cdot x_1)$ and is readily enumerated again. Alice can also enumerate the candidate of Bob's x_1 and x_2 for each r_B in polynomial time.

As shown, multiplying a random number does not contribute to hinder Alice's guess, either. In our protocol, we use the SFE for private comparison with scalar product protocol. Private comparison is stated as follows:

Statement 3 *(Private comparison of random shares) Let Alice's input be x^A and Bob's input be x^B, where x_i^A and x_i^B are random shares of x_i for all i. Then,* private comparison of random shares *computes the index i^* such that*

$$i^* = arg\max_i(x_i^A + x_i^B). \tag{3}$$

[2] A semi-honest party is one who follows the protocol properly with the exception that the party retains a record of all its intermediate observations. From such accumulated records, semi-honest parties attempt to learn other party's privte information (Goldreich, 2004).

After the protocol execution, Alice knows i and nothing else: Bob learns nothing.*

When Alice has a cost vector α and Bob has two tour vectors x_1, x_2, the protocol for private scalar product comparison is obtained using SFE as follows:
[Private scalar product comparison]

- Private Input of Alice : $\alpha \in Z_m^d$
- Private Input of Bob : $x_1, x_2 \in Z_m^d$
- Output of Bob : An inequation $I \in \{I_-, I_+\}$ (Alice has no output)

1. Alice: Generate a private and public key pair(p_k, s_k) and send p_k to Bob.
2. Alice: For $i = 1, ..., d$, Alice computes $c_i = \mathrm{Enc}_{p_k}(\alpha_i)$. Send them to Bob.
3. Bob: Compute $w \leftarrow \prod_{i=1}^{d} c_i^{x_{2,i}} \cdot \prod_{i=1}^{d} c_i^{-x_{1,i}} \cdot \mathrm{Enc}_{p_k}(-r^B)$ and send w to Alice where $r \in_r Z_N$
4. Alice: Compute $r^A = \mathrm{Dec}_{s_k}(w) (= x_2 \cdot \alpha - x_1 \cdot \alpha - r^B)$.
5. Alice and Bob: Jointly run a SFE for private comparison. If $((r^A + r^B) \mod N) \geq 0$, I_+ is returned to Bob. Else, I_- is returned to Bob.

First, Alice encrypts her cost vector and send all elements to Bob. Then, Bob computes the encrypted difference of two scalar products with randomization as follows:

$$
\begin{aligned}
w &= \prod_{i=1}^{d} c_i^{x_{2,i}} \cdot \prod_{i=1}^{d} c_i^{-x_{1,i}} \cdot \mathrm{Enc}_{p_k}(-r^B) && (4)\\
&= \mathrm{Enc}_{p_k}\left(\sum_{i=1}^{d} \alpha_i x_{2,i} - \sum_{i=1}^{d} \alpha_i x_{1,i} - r^B \right) && (5)\\
&= \mathrm{Enc}_{p_k}\left(\alpha \cdot x_2 - \alpha \cdot x_1 - r^B \right). && (6)
\end{aligned}
$$

At step four, Alice obtains

$$
r^A = x_2 \cdot \alpha - x_1 \cdot \alpha - r^B, \tag{7}
$$

where r^A and r^B are random shares of $x_2 \cdot \alpha - x_1 \cdot \alpha$ over Z_N; both do not learn any from random shares. Then at the last step, Both jointly run SFE for private comparison to evaluate whether or not $(r^A + r^B) \mod N$ is greater than zero, which is the desired output.
In what follows, we describe the execution of this protocol as $(\alpha, (x_1, x_2)) \longrightarrow_{\mathrm{SPC}} (\emptyset, I)$.
Note that the input size for SFE is p regardless of the vector size m and dimension d. In principle, the computational load of SFE is large particularly when the input size is large. Although the computation complexity of this protocol in step two and step three is still $O(d)$, we can reduce the entire computational cost of private scalar product comparison by limiting computation of SFE only to comparison of random shares.

Theorem 1 *(Security of private scalar product comparison) Assume Alice and Bob behave semi-honestly. Then, private scalar product comparison protocol is secure in the sense of Statement 1.*

The security proof should follow the standardized proof methodology called simulation paradigm (Goldreich, 2004). However, due to the limitation of the space, we explain the

security of this protocol by showing that parties cannot learn nothing but the output from messages exchanged between parties.
The messages Bob receives from Alice before step five is $\mathrm{Enc}_{p_k}(\alpha_1),...,\mathrm{Enc}_{p_k}(\alpha_d)$. Because Bob does not have Alice's secret key, Bob learns nothing about Alice's vector. The information Alice receives from Bob before step five is only a random share $r^A = \alpha \cdot x_2 - \alpha \cdot x_2 - r^B$, from which she cannot learn anything, either. At step five, it is guaranteed that SFE reveals only the comparison result (Yao, 1986). Consequently, the overall protocol is secure in the sense of Statement 1.

6. Local search of TSP using scalar product comparison

Using the protocol for private scalar product comparison, local search is convertible to a privacy-preserving protocol. As an example, we introduce a Privacy Preserving Local Search (PPLS) for TSP using 2-opt neighborhood.
Because the local search described in Section 3 is rank-based, it is readily extended to a privacy-preserving protocol. Using the protocol for private scalar product comparison, PPLS is designed as follows:
[Privacy-Preserving Local Search]

- Private Input of Server: instance vector $\alpha \in Z_m^d$
- Private Input of Searcher: subset of instance $V' \subseteq V$
- Private Output of Searcher: local optimal solution x^*

1. Server: Generate a pair of a public and a secret key (p_k, s_k) and send p_k to the searcher.
2. Server: For $i = 1,...,d$, compute $c_i = \mathrm{Enc}_{p_k}(\alpha_i)$ and send them to the searcher.
3. Searcher: Generate an initial solution x_0 using V'
4. Searcher: $x \in_r N(x_0), N(x_0) \leftarrow N(x_0) \setminus \{x\}$
5. Searcher:
   a) Compute $(\alpha,(x,x_0)) \longrightarrow_{\mathrm{SPC}} (\emptyset, I)$ with probability 0.5. Otherwise, compute $(\alpha,(x_0,x)) \longrightarrow_{\mathrm{SPC}} (\emptyset, I)$.
   b) If I corresponds to $\alpha \cdot x - \alpha \cdot x_0 < 0$, then $x_0 \leftarrow x$
6. Searcher: If $N(x_0) = \emptyset$ or satisfies some termination condition, $x^* \leftarrow x_0$ and output x^*. Otherwise, go to step 4.

Step one and step two can be executed solely by the server. In step three, an solution is initialized. Step three and step four are also executed solely by the searcher.
Step five can be executed privately by means of private scalar product comparison. Note that the order of the inputs of private scalar product comparison is shuffled randomly in step 5(a). The reason is explained in detail in Section 6.2.
Readers can find multi-party expansion of private scalar product protocol in (Goethals et al., 2004). Multi-party expansion of private scalar product comparison is straightforward with a similar manner in the literature. This expansion readily allows us to obtain protocols of PPLS for $(k,1)$-TSP.

6.1 Communication and computation complexity

Communication between the server and the searcher occurs in steps one, two, and five. In $(1,1)$-TSP, we can naturally assume that the cost vector α is not changed during optimization. Therefore, transfer of $(c_1,...,c_d)$ occurs only once in steps one and two. The communication complexity of step 5(a) is $O(1)$. As shown, the communication complexity is not time consuming in this protocol.

The time consuming steps of PPLS are private comparison by SFE (step five of private scalar product comparison) and exponentiation computed by the searcher (step three of private scalar product comparison). The computation time of the SFE is constant for any d and cannot be reduced anymore. On the other hand, there is still room for improvement in the exponentiation step.

Using naive implementation, step three of private scalar product comparison costs $O(d)(= O(n^2))$. To reduce this computation, we exploit the fact that the number of changed edges by 2-opt is much smaller than d.

In vector $x - x_0$, only 2 elements are changed from 1 to 0 and the other 2 elements are changed from 0 to -1 when 2-opt is used as the neighborhood. The remaining elements are all 0 irrespective of the problem size. Since the exponentiation of 0 can be skipped in step three of private scalar product comparison, the time complexity is saved at most $O(1)$ by computing only in changing edges.

6.2 Security

Given a TTP, the ideally executed privacy-preserving optimization is the following.

Statement 4 *(Privacy-preserving local search for $(1,1)$-TSP (ideal)) Let the searcher's input be $V' \subseteq V$. Let the server's input be $\alpha \in Z_m^d$. After the execution of the protocol, the searcher learns the (local) optimal solution x^*, but nothing else. The server learns nothing at the end of the protocol, either.*

Unfortunately, the security of the PPLS is not equivalent to this statement. We briefly verify what is protected and leaked after the protocol execution.

Messages sent from the searcher to the server are all random shares. Thus, it is obvious that the server does not learn anything from the searcher.

There remains room for discussion about what the searcher can guess from what it learns because this point is dependent on the problem domain. The searcher learns a series of the outcome of private scalar product comparison protocol in the middle of PPLS execution. Let the outcome of the t-th private scalar product comparison be $I(t)$.

In the case of TSP, $I(t)$ corresponds to an inequality, $\alpha \cdot x - \alpha \cdot x_0 \leq 0$ or $\alpha \cdot x - \alpha \cdot x_0 > 0$. Although the elements of the private cost vector α are not leaked from this, the searcher learns whether $\alpha \cdot (x - x_0) > 0$ or not from this comparison. This indicates that orders of cost values might be partially implied by the searcher because the searcher knows what edges are included in these solutions.

As long as the server and the searcher mutually interact only through the private scalar product comparison, the server never learns the searcher's V' and the searcher never learns the server's α. However, the security of PPLS is not perfect as a protocol with a TTP. A method to block the guess of the searcher from intermediate messages remains as a challenge for future study.

7. Experimental analysis and discussion

In this section, we show experimental results of PPLS for $(1,1)$-TSP in order to evaluate the computation time and investigate its scalability.

7.1 Setting

Five problem instances were chosen from TSPLIB(Reinelt, 1991). The city size is 195 – 1173. In the distributed TSP, the searcher can choose a subset of cities V' arbitrarily as the input. For our experiments, V' is set to V for all problems because the computation time becomes greatest when $V = V'$. PPLS was terminated when x_0 reaches a local optimum.
As a homomorphic cryptosystem, the Paillier cryptosystem(Pailler, 1999) with 512-bit and 1024-bit key was used. The server and the searcher program were implemented using java. Both programs were run separately on a Xeon 2.8 GHz (CPU) with 1 GB (RAM) PCs with a 100 Mbps Ethernet connection. For SFE, Fairplay (Malkhi et al., 2004), a framework for generic secure function evaluation, was used.
PPLS was repeated for 50, 100, and 300 times with changing initial tours (depicted as 50-itr, 100-itr and 300-itr).
Both the first and the second step of PPLS can be executed preliminarily before choosing city subset V'. Therefore, we measured the execution time from the third step to the termination of the protocol.

7.2 Results

Fig. 2 shows the estimated computation time required for optimization. With a 512-bit key, PPLS (1-itr) spent 19 (min) and 79 (min) to reach the local optimum of rat195 and rat575. Using a 1024-bit key, PPLS (1-itr) spent 21 (min) and 89 (min) for the same problems.
Table 1 shows the error index (=100× obtained best tour length / known best tour length) of PPLS (average of 20 trials). Note that privacy preservation does not affect the quality of the obtained solution in any way because the protocol does not change the behavior of local search if the same random seed is used.
Earlier, we set $V' = V$ for all problems. Even when the number of cities $|V|$ is very large, the computation time is related directly to the number of chosen cities $|V'|$ because the number of evaluations is usually dependent on the number of chosen cities.
Although the computation time is not yet sufficiently small in large-scale problems, results show that the protocol completes in a practical time in privacy-preserving setting when the number of cities are not very numerous.

8. Related works

A few studies have been made of the *Privacy-Preserving Optimization* (PPO). Silaghi et al. proposed algorithms for distributed constraint satisfaction and optimization with privacy enforcement (MPC-DisCSP)(Silaghi, 2003)(Silaghi & Mitra, 2004). Actually, MPC-DisCSP is based on a SFE technique (Ben-Or et al., 1988). Yokoo et al. proposed a privacy-preserving protocol to solve dynamic programming securely for the multi-agent system (Yokoo & Suzuki, 2002; Suzuki & Yokoo, 2003). (Brickell & Shmatikov, 2005) proposed a privacy-preserving protocol for single source shortest distance (SSSD) which is a privacy-preserving transformation of the standard Dijkstra's algorithm to find the shortest path on a graph. These studies are all based on cryptographic guarantees of security.

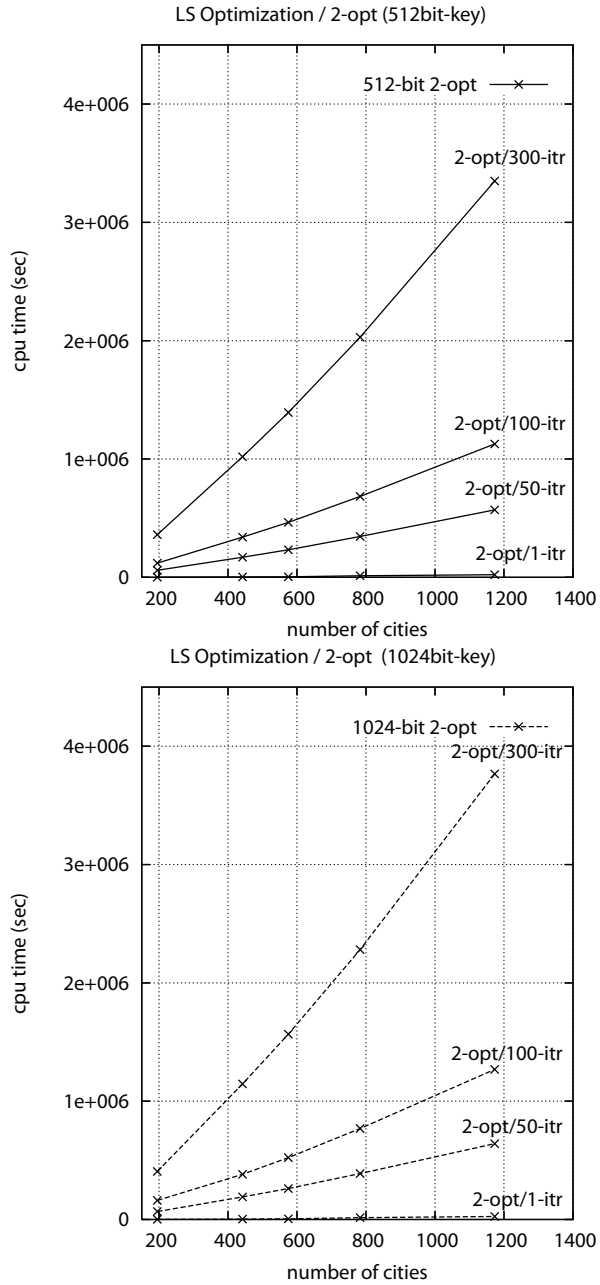


Fig. 2. Computation time of PPLS. Estimated time required for the completion of PPLS optimization (left: 512-bit key, right: 1024-bit key).

	PPLS (2-opt)			
	1-itr.	50-itr.	100-itr.	300-itr.
rat195	13.3	9.16	8.73	8.65
pcb442	16.4	8.88	8.88	8.88
rat575	11.6	9.96	9.96	9.90
rat783	12.2	10.8	10.8	10.3
pcb1173	15.4	12.9	12.9	12.3

Table 1. The error index of PPLS (2-opt).

To the best of authors' knowledge, this study is the first attempt for privacy-preserving optimization by means of meta-heuristics. As discussed earlier, private scalar product comparison allows us to compare two scalar products. It follows that our PPLS is available for any privacy-preserving optimization problems provided that the cost function is represented in the form of scalar products. In addition, private scalar product comparison can be incorporated into not only local search but more sophisticated meta-heuristics, such as genetic algorithms or tabu search, as long as the target algorithm uses only paired comparison for selection.

9. Summary

We proposed and explained a protocol for privacy-preserving distributed combinatorial optimization using local search. As a connector that combines local search and privacy preservation, we designed a protocol to solve a problem called private scalar product comparison. The security of this protocol is theoretically proved. Then, we designed a protocol for privacy-preserving optimization using a combination of local search and private scalar product comparison. Our protocol is guaranteed to behave identically to algorithms that do not include features for privacy preservation.

As an example of distributed combinatorial optimization problems, we specifically examined the distributed TSP and designed a privacy-preserving local search that adopts 2-opt as a neighborhood operator. The result show that privacy-preserving local search with 2-opt solves a 512-city problem within a few hours with about 10 % error. Although the computation time is not yet sufficiently small in a large-scale problem, it is confirmed that the protocol is completed in a practical time, even in privacy-preserving setting. Both the searcher's and the server's computation can be readily paralleled. The implementation of parallel computation is a subject for future work. Application of PPLS to distributed combinatorial optimization problems such as the distributed QAP, VRP, and Knapsack problem is also a subject for future work.

10. References

Ben-Or, M., Goldwasser, S. & Wigderson, A. (1988). Completeness theorems for non-cryptographic fault-tolerant distributed computation, *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, ACM, pp. 1–10.

Brickell, J. & Shmatikov, V. (2005). Privacy-preserving graph algorithms in the semi-honest model, *Proceedings of ASIACRYPT, LNCS*, Springer-Verlag, pp. 236–252.

Goethals, B., Laur, S., Lipmaa, H. & Mielikäinen, T. (2004). On private scalar product computation for privacy-preserving data mining, *Proceedings of Seventh International Conference on Information Security and Cryptology (ICISC), LNCS*, Springer, pp. 104–120.

Goldreich, O. (2004). *Foundations of Cryptography: Volume 2, Basic Applications*, Cambridge University Press.

Handfield, R. & Nichols, E. (1999). *Introduction to supply chain management*, Prentice Hall Upper Saddle River, NJ.

Malkhi, D., Nisan, N., Pinkas, B. & Sella, Y. (2004). Fairplay - a secure two-party computation system, *Proceedings of the thirteenth USENIX Security Symposium*, pp. 287–302.

Pailler, P. (1999). Public-Key Cryptosystems Based on Composite Degree Residuosity Classes, *Proceedings of Eurocrypt, Lecture Notes in Computer Science*, Vol. 1592, pp. 223–238.

Reinelt, G. (1991). TSPLIB-A Traveling Salesman Problem Library, *ORSA Journal on Computing* **3**(4): 376–384.

Silaghi, M. (2003). Arithmetic circuit for the first solution of distributed CSPs with cryptographic multi-party computations, *Proceedings of IEEE/WIC International Conference on Intelligent Agent Technology (IAT)*, IEEE Computer Society Washington, DC, USA, pp. 609–613.

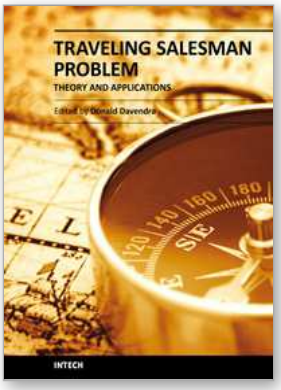
Silaghi, M. & Mitra, D. (2004). Distributed constraint satisfaction and optimization with privacy enforcement, *Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT)*, IEEE Computer Society Washington, DC, USA, pp. 531–535.

Suzuki, K. & Yokoo, M. (2003). Secure Generalized Vickrey Auction Using Homomorphic Encryption, *Proceedings of Seventh International Conference on Financial Cryptography (FC)*, Springer, pp. 239–249.

Vollmann, T. (2005). *Manufacturing Planning and Control Systems for Supply Chaing Management*, McGraw-Hill.

Yao, A. C.-C. (1986). How to generate and exchange secrets, *IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 162–167.

Yokoo, M. & Suzuki, K. (2002). Secure multi-agent dynamic programming based on homomorphic encryption and its application to combinatorial auctions, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, ACM Press New York, NY, USA, pp. 112–119.



Traveling Salesman Problem, Theory and Applications

Edited by Prof. Donald Davendra

ISBN 978-953-307-426-9
Hard cover, 298 pages
Publisher InTech
Published online 30, November, 2010
Published in print edition November, 2010

This book is a collection of current research in the application of evolutionary algorithms and other optimal algorithms to solving the TSP problem. It brings together researchers with applications in Artificial Immune Systems, Genetic Algorithms, Neural Networks and Differential Evolution Algorithm. Hybrid systems, like Fuzzy Maps, Chaotic Maps and Parallelized TSP are also presented. Most importantly, this book presents both theoretical as well as practical applications of TSP, which will be a vital tool for researchers and graduate entry students in the field of applied Mathematics, Computing Science and Engineering.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821