Privacy-preserving Tabu Search for Distributed Graph Coloring

Yuan Hong[§], Jaideep Vaidya[§], Haibing Lu[‡], and Basit Shafiq[§]

[§]MSIS Department and CIMIC, Rutgers University, NJ, USA, {yhong, jsvaidya, basit}@cimic.rutgers.edu
[‡]OMIS, Santa Clara University, CA, USA, hlu@scu.edu

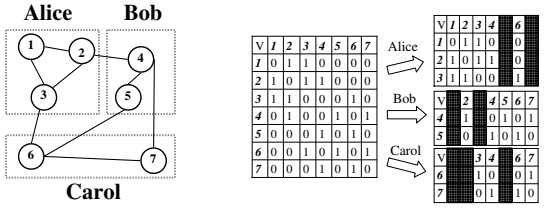
***Abstract*—Combinatorial optimization is a fundamental problem found in many fields. In many real life situations, the constraints and the objective function forming the optimization problem are naturally distributed amongst different sites in some fashion. The typical approach is to collect all of this information together and centrally solve the problem. However, this requires all parties to completely share their information, which may lead to serious privacy issues. Privacy-preserving techniques need to be developed to enable distributed optimization with limited information disclosure. A further complicating factor is that combinatorial optimization problems are typically NP-hard, requiring approximation algorithms or heuristics to provide a practical solution. In this paper, we focus on a very well-known hard problem – the distributed graph coloring problem, which has been utilized to model many practical problems in scheduling and resource allocation. We propose an efficient protocol that securely solves this problem based on the tabu search metaheuristic. Specifically, our solution uses a distributed local search algorithm to find a good solution. We analyze the security of our approach and experimentally demonstrate the effectiveness of our approach.**

I. Introduction

Optimization is a fundamental problem found in many fields, and it has many applications in operational research, artificial intelligence, machine learning and mathematics. In many real life situations, the constraints and the objective function forming the optimization problem are naturally distributed amongst different sites in some fashion. The typical approach is to collect all of this information together and centrally solve the problem. For example in supply chain management, since the delivery trucks are not always fully loaded (i.e. empty 25% of the time), a collaborative logistics service (Nistevo.com) can be utilized to merge loads from different companies bound to the same destination. Huge savings were realized in such collaborative transportation (freight costs were cut by 15%, for an annual savings of $2 million[20]), which can be modeled as a typical distributed linear programming problem.

However, this requires all parties to completely share their information, which may lead to serious privacy issues simply because each party's portion of the optimization problem typically refers to its private information. i.e. In constrained optimization, the private constraints often describe the limitations or preferences of a party, and each party's share of the global solution represents its private output in the global best decision or assignments. Even without privacy constraints, solving an optimization problem typically is computationally very expensive. Thus, an open question is whether we can solve such problems both securely and efficiently? This challenging issue has been investigated only in the context of a few optimization problems. Specifically, Vaidya [21] securely solved the collaborative linear programming problem, assuming that one party holds its private objective function while the other party holds its private constraints. Sakuma et al. [16] proposed a genetic algorithm for securely solving two-party distributed traveler salesman problem (TSP), assuming that one party holds the cost vector while the other party holds the exact cities that should be traveled. Atallah et al.[1] proposed protocols for secure supply chain management. However, none of these can be used to solve general problems such as distributed scheduling, or distributed network resource allocation, etc. [13]. In this paper, we focus on this class of problems. For instance, consider a simple multi-party job scheduling problem:

Example 1: Alice, Bob and Carol have jobs $\{1,2,3\}$,$\{4,5\}$ *and* $\{6,7\}$ *respectively; as depicted in Fig. 1(a) (edges represent conflicts), some jobs cannot be assigned into the same time slot due to resource conflicts (i.e. sharing the same machine); Since each job is held by a specific party, they do not want to reveal their set of jobs and corresponding assigned time slots to each other. Given this, is it possible to securely assign the* 7 *jobs into* k *time slots?*



(a) Partitioned Graph (b) Partitioned Adjacency Matrix

Fig. 1. Distributed Graph Coloring Problem

Intuitively, the above scheduling problem can be modeled as a distributed (among 3 parties) k-coloring problem (the decision problem of graph coloring). Specifically, given k different colors (k time slots), the undirected graph $G = (V, E)$ represents the jobs and their conflicts (as shown in Fig. 1(a)) – a vertex represents a job, and an edge between two vertices represents a conflict between the corresponding jobs. Now, each vertex should be assigned one out of k colors

(representing the time assignment for a job), with no two vertices connected by an edge having the same color (no two jobs having a conflict should be scheduled into the same time slot). Thus, a feasible k-coloring solution represents a feasible scheduling solution.

Note that G is partitioned such that each party owns a subgraph (i.e, holds some vertices and the edges related to its own vertices). As shown in Fig. 1(a), Alice, Bob and Carol hold 3, 2 and 2 jobs, respectively. The adjacency matrix of the graph is thus partitioned as shown in Fig. 1(b). When formulating and solving this problem, Alice, Bob and Carol do not want to reveal their partition in the graph and the colors of their vertices to each other. Hence, the primary goal of this paper is to propose a privacy-preserving solution for the distributed graph coloring problem.

Note that, many combinatorial optimization problems (including graph-coloring), are NP-hard. This means that exhaustive search is simply not feasible, and no polynomial time algorithm exists. Thus, securely solving the graph coloring problem is extremely challenging due to the computational complexity. Several meta-heuristics such as tabu search have been proposed to solve the centralized graph coloring problem by improving the performance of local search with memory structures [5][6][9]. While meta-heuristics cannot guarantee an optimal results, they are generally considered to be quite effective in solving combinatorial optimization problems and obtain near-optimal outputs in an acceptable running time. Therefore, in this paper, we propose an efficient secure algorithm to enable distributed tabu search.

The rest of this paper is organized as follows. Section II reviews the related work in privacy-preserving optimization. Section III formally defines the problem and reviews the tabu search method for graph coloring. Section IV introduces a secure and efficient protocol to solve distributed graph coloring problem, assuming that all parties are honest-but-curious (obey the protocol), and provides a formal security analysis. Section V experimentally validates the performance of our approach. Section VI concludes the paper and discusses some directions for future work.

II. Related Work

There is prior work both in distributed optimization as well as in secure optimization. Silaghi and Rajeshirke [18] show that a secure combinatorial problem solver must necessarily pick the result randomly among optimal solutions to be really secure. Silaghi and Mitra [17] propose arithmetic circuits for solving constraint optimization problems that are exponential in the number of variables for any constraint graph. A significantly more efficient optimization protocol specialized on generalized Vickrey auctions and based on dynamic programming is proposed by Suzuki and Yokoo [19]. Yokoo et al. [24] also propose a scheme using public key encryption for secure distributed constraint satisfaction. Atallah et al.[1] propose protocols for secure supply chain management. However, none of these can be used to solve the general class of distributed scheduling problems, which is what we focus on.

Recently, privacy-preserving linear programming problem [21][12][22][4][3][10] has been well studied to securely solve the distributed LP problems among multiple parties. However, little work has been made on securing distributed NP-hard problems due to the challenging issue of computational complexity. Sakuma et al. [16] proposed a genetic algorithm for securely solving two-party distributed traveling salesman problem (NP-hard). They consider the case that one party holds the cost vector while the other party holds the tour vector (this is a special case of multi-party distributed combinatorial optimization). The distributed TSP problem that is completely partitioned among multiple parties has been discussed but not solved in [16]. We consider a more complicated scenario in another classic combinatorial optimization model (graph coloring) that is completely partitioned among multiple parties.

III. Problem Definition

In graph theory, the **k-coloring** problem represents the following NP-complete decision problem[15] – does graph G have a proper vertex coloring with k colors. The (NP-hard) optimization version of this problem simply seeks the minimum k that can properly color the given graph. The minimum k is known as the chromatic number of the graph. Since, seeking the chromatic number can be reduced to a set of k-coloring problems, we only consider the k-coloring problem on distributed graphs in this paper. We first formally define the graph coloring (decision problem), review tabu search for graph coloring, and then introduce our solution approach.

A. Graph Coloring Problem Formulation with Scalar Products

When seeking the chromatic number k or solving the k-coloring problem, any solution requires that the number of adjacent vertices having the same color should be 0. We thus denote any two adjacent vertices that have the same color as a "conflict" in a colored graph. Thus, given k different colors, if we let μ represent the total number of conflicts in a colored G, we thus have: if $\min(\mu) = 0$, G is k-colorable.

Given k colors, we can represent the color of any vertex in the graph using a k-dimensional boolean vector x (domain $\Phi(k)$: all the k-dimensional boolean vectors include only one "1" and $k-1$ "0"s). If the ith number in the boolean vector is 1 (all the remaining numbers should be 0), the current vertex is colored with the ith color ($i \in [1, k]$). Thus, we let $x = \{x_i, \forall v_i \in G\}$ be a coloring solution in a k-coloring problem where x_i represents the coloring boolean vector for vertex v_i. In addition, given the coloring boolean vectors of two adjacent vertices $x_i, x_j \in x$ ($i \neq j$ and $v_i, v_j \in G$), we can represent the color conflict of v_i and v_j in solution x as the **scalar product** of x_i and x_j:

$$\mu_{ij}(x) = \begin{cases} 0 & \text{if } x_i \cdot x_j = 0 \text{ (v_i and v_j are not conflicted)} \\ 1 & \text{if } x_i \cdot x_j = 1 \text{ (v_i and v_j are conflicted)} \end{cases} \quad (1)$$

where $\mu(x) = \sum_{\forall e_{ij} \in G} \mu_{ij}(x)$. We thus formulate the k-coloring problem as $\{\min : \sum_{\forall e_{ij} \in G} x_i \cdot x_j,\ s.t. \forall v_i, v_j \in$

$G, x_i, x_j \in \Phi(k)\}$. If and only if the optimal value is 0, the problem is k-colorable. When seeking the chromatic number k, we can solve the above problem using different k and choose the minimum colorable k.

B. Tabu Search for Solving Graph Coloring Problems

Tabu search uses a memory structure to enhance the performance of a local search method. Once a potential solution has been determined, it is marked as "taboo". So the algorithm does not visit that possibility repeatedly. Tabu search has been successfully used for graph coloring. Specifically, given a graph $G = (V, E)$ and k colors, a popular tabu search algorithm for graph coloring – Tabucol [9] (summarized using our notation) is described as follows:

1) initialize a solution $x = (\forall v_i \in G, x_i \in \phi(k))$ by randomly selecting colors.
2) iteratively generate neighbors x' of x until $\mu(x') < \mu(x)$ by changing the color of an endpoint of an arbitrary conflicted edge $v_i : x_i \rightarrow x_i'$ (if no x' is found with $\mu(x') < \mu(x)$ in a specified number of iterations, find a best x' in current set of neighborhoods where (v_i, x_i') is not in the tabu list, and let x' be the next solution).
3) if (v_i, x_i') is not in the tabu list, make current x' (best so far) as the next solution $x = x'$ and add the pair (v_i, x_i) into the tabu list; If (v_i, x_i') is in the tabu list and $\mu(x')$ is still smaller than $\mu(x)$, make x' as the next solution anyway; Otherwise, discard x'.
4) repeat step 2,3 until $\mu(x') = 0$ or the maximum number of iterations is reached.

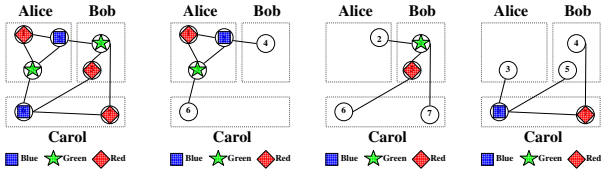
C. Privacy-preserving Tabu Search for DisGC

Recall that Example 1 can be modeled by a distributed graph coloring problem on a graph G whose adjacency matrix is partitioned among m parties (each vertex belongs to only one party). We thus define it as follows:

Definition 1 (Distributed Graph Coloring (DisGC)): Given k colors and a partitioned graph $G = (V, E)$ where each vertex $v_i \in V$ belongs to only one party, does a k-coloring of G exist?

In the distributed graph, we define two different types of edges: 1. **internal edge:** any edge e_{ij} whose endpoints (v_i and v_j) are owned by the same party; 2. **external edge:** any edge e_{ij} whose endpoints (v_i and v_j) belong to two different parties. Similarly, we define two categories of vertices: 1. **inner vertex**: a vertex that does not have any external edge (e.g.,. v_1 in Fig. 1(a)) 2. **border vertex**: a vertex that has at least one external edge (e.g., v_2 in Fig. 1(a)).

When m different parties want to collaboratively solve the DisGC problem, each party does not want to reveal its partition in G and the colors of its vertices to others. For instance, suppose we color the graph in Fig. 1(a) using $k = 3$ colors, the parties should only learn the relevant information. For example, Fig. 2(a) gives a 3-colored global solution, corresponding to which Alice, Bob and Carol would only know the information shown in Fig. 2(b), 2(c) and 2(d), respectively. Thus, each party only sees the colors assigned to the vertices in their local graph and cannot even see the color of the border vertices of other parties that share an external edge with this party (the vertex existence is known anyway). For example, in Alice's view of Bob's graph, only vertex 4 which is directly connected to vertex 2 in Alice's graph is visible to Alice, though its color is not known to Alice.



(a) 3-Colored (b) Alice (c) Bob (d) Carol

Fig. 2. A Secure Solution to the DisGC Problems

Thus, we have two privacy requirements when solving the DisGC problem: first, protect each party's subset of the graph structure including its vertices and the internal edges (the external edges and their endpoints are inevitably known to the two parties sharing them); second, protect the color assignment for each party's vertices (including the endpoints of the external edges) in every iteration. A direct method to securely solve the DisGC problem is by using a trusted third party to collect partitioned data from all parties, centrally solving the problem with Tabucol algorithm [9] and distributing the color of every vertex to the vertex owner. However, if no trusted third party can be found, a secure protocol is necessary. Therefore, the goal of this paper is to present a privacy-preserving tabu search method for solving DisGC without a trusted third party, assuming that all parties are honest-but-curious. The honest-but-curious assumption is realistic, and is commonly assumed in the literature, and the solution can be generalized to fully malicious adversaries (though at added cost).

IV. Privacy-preserving Tabu Search

As mentioned in Section III-B, a typical tabu search algorithm for graph coloring problem includes numerous repeated computations such as: computing the total number of conflicts for a solution, searching better neighborhoods for a specific solution, verifying a new vertex color pair with all the taboos and updating the tabu list if necessary. We now look at how all of this can be securely done, starting with some of the fundamental cryptographic building blocks that will be used to provide a solution.

A. Fundamental Cryptographic Building Blocks

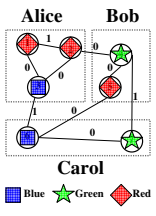
1) Secure Scalar Product for Secure Conflict Computation: For any potential solution (a colored graph) to the DisGC problem, the total number of conflicts corresponding to that solution must be computed to figure out whether it improves the objective. As introduced in Section III-A, the total number of conflicts in a solution x is denoted by $\mu(x)$. $\mu(x)$ can be represented as the sum of all the scalar products of every two adjacent vertices' coloring vectors. To securely compute the scalar product of two endpoints on every external edge, we use Goethals et al.'s approach [8] which is based on a public-key homomorphic cryptosystem. This is both simple and provably secure. The problem is defined as follows: P_a has a n-dimensional vector $\vec{X}$ while P_b has a n-dimensional vector

$\vec{Y}$. At the end of the protocol, P_a should get vector vector $\vec{X}\cdot\vec{Y}+r_b$ where r_b is a random number chosen from a uniform distribution and is known only to P_b. The key idea behind the protocol is to use a homomorphic encryption system such as the Paillier cryptosystem [14]. A homomorphic cryptosystem is a semantically-secure public-key encryption system which has the additional property that given any two encryptions $E(A)$ and $E(B)$, there exists an encryption $E(A*B)$ such that $E(A)*E(B)=E(A*B)$, where $*$ is either addition or multiplication (in some abelian group). The cryptosystems mentioned above are additively homomorphic (thus the operation $*$ denotes addition). Using such a system, it is quite simple to create a scalar product protocol. If P_a encrypts its vector $\vec{X}$ and sends $Enc_{pk}(\vec{X})$ with the public key pk to P_b, P_b can use the additive homomorphic property to compute the encrypted sum of scalar product and its random number: $Enc_{pk}(\vec{X}\cdot\vec{Y}+r_b)$. Finally, P_b sends $Enc_{pk}(\vec{X}\cdot\vec{Y}+r_b)$ back to P_a. Thus, P_a decrypts and obtain $\vec{X}\cdot\vec{Y}+r_b$ (r_b and $\vec{X}\cdot\vec{Y}$ are unknown to P_a). Since P_b cannot decrypt $Enc_{pk}(\vec{X}\cdot\vec{Y}+r_b)$ with a public key, it cannot learn the scalar product either.

Therefore, we can adopt the above cryptographic building block to securely compute the total number of conflicting edges $\mu(x)$ in solution x (Secure Conflict Computation). Note: while computing $\mu(x)$, it is not necessary to securely compute the number of each party's conflicting internal edges in x. Alternatively, we can let each party independently count them, and sum the number with its shares in secure scalar product computation. The detailed steps are shown in Algorithm 1.

Algorithm 1 Secure Conflict Computation

Input: $G=(V,E)$ is colored as $x\in\Phi(k)$ where the color of $\forall v_i\in G$ is denoted as $x_i\in x$.
Output: each party P_a's share $\mu_a(x)$ in $\mu(x)$ where $\mu(x)=\sum_{\forall a}\mu_a(x)$;
1: **for** each party P_a **do**
2: count the total number of its conflicting internal edges as $\mu_a(x)$;
3: **for all** external edges $e_{ij}\in G$ **do**
4: $***$ Suppose that v_i and v_j belong to P_a and P_b, respectively. $***$
5: W.o.l.g. P_a creates a public and private key pair (pk,sk);
6: P_a encrypts its color vector x_i as $x_i'=Enc_{pk}(x_i)$ and sends x_i' and pk to P_b;
7: P_b generates a random integer r_{ij}, encrypts the scalar product as: $x_i''=Enc_{pk}(x_i\cdot x_j)*Enc_{pk}(r_{ij})$ with the public key pk, and sends x_i'' back to P_a;
8: P_a decrypts x_i'' with its private key sk and obtain $s_{ij}=x_i\cdot x_j+r_{ij}$;
9: **for** each party P_a **do**
10: $\mu_a(x)\leftarrow\mu_a(x)+\sum\forall s_{ij}$ (received from other parties)$-\sum\forall r_{ij}$ (generated by P_a);



(a) Conflicts in G

Edges (i or e)	*Alice*	*Bob*	*Carol*
e_{12} (i), $\mu_{12}(x)=1$	1	N/A	N/A
e_{13} (i), $\mu_{13}(x)=0$	0	N/A	N/A
e_{23} (i), $\mu_{23}(x)=0$	0	N/A	N/A
e_{24} (e), $\mu_{24}(x)=0$	0+45	-45	N/A
e_{36} (e), $\mu_{36}(x)=1$	1+99	N/A	-99
e_{45} (i), $\mu_{45}(x)=0$	N/A	0	N/A
e_{47} (e), $\mu_{47}(x)=1$	N/A	1+77	-77
e_{56} (e), $\mu_{56}(x)=0$	N/A	0-98	+98
e_{67} (i), $\mu_{67}(x)=0$	N/A	N/A	0
Shares of the Sum	146	-65	-78

(b) Each Party's Share in $\mu(x)$=3

Fig. 3. Example for Secure Conflict Computation

Fig. 3(a) and 3(b) provide a simple example of Secure Conflict Computation. For each external edge, such as e_{24}, e_{36}, e_{47} and e_{56}, the scalar product computation is carried out between the two involved parties. As shown in Fig. 3(b), the returned value in secure scalar product computation is the sum of $\mu_{ij}(x)$ and a random number r_{ij} generated by the other party (Note: the scalar product of every external edge is securely computed only once). To generate each party's output, each party then sums all the returned values with its number of conflicting internal edges, and finally subtracts all its generated random numbers (the sum of all parties' outputs should be $\mu(x)$). This protocol is provably secure.

2) Secure Comparison: Given two solutions x and x', we can let each party P_a generate its shares $\mu_a(x)$ and $\mu_a(x')$ in $\mu(x)$ and $\mu(x')$ respectively, ensuring that $\mu(x)=\sum_{\forall a}\mu_a(x)$ and $\mu(x')=\sum_{\forall a}\mu_a(x')$. However, $\mu(x)$ and $\mu(x')$ cannot be revealed to any party because they may lead to additional privacy breach in generating neighborhoods.

To securely compare $\mu(x')$ and $\mu(x)$ using all parties' partitioned shares, we can adopt Yao's secure comparison protocol [23] that compares any two integers securely. Yao's generic method is one of the most efficient methods known. Recently, FairplayMP [2] is developed for secure multiparty computation where m untrusted parties with private inputs $x_1,\ldots,x_m$ wish to jointly compute a specific functions $f_i(x_1,\ldots,x_m)$ and securely obtain each party's output. Specifically, FairplayMP is implemented by writing functions with a high-level language – Secure Function Definition Language (SFDL) 2.0 and describing the participating parties using a configuration file. The function is then complied into a Boolean circuit, and a distributed evaluation of the circuit is finally performed without revealing anything. In our secure comparison, each party has the shares in $\mu(x)$ and $\mu(x')$ respectively; the function should be $\mu(x')-\mu(x)$ (computed by all m parties); the output $\mu(x')<\mu(x)$ or $\mu(x')\geq\mu(x)$ is generated for designated parties. Finally, we have to compare $\mu(x)$ and 0 to decide whether to terminate the optimization or not. With the same protocol, we can let all parties securely compare $\mu(x)$ and 1 with inputs from all parties (the shares of $\mu(x)$ and 1). The output $\mu(x)<1$ or $\mu(x)\geq 1$ is similarly generated for corresponding parties.

B. Privacy-preserving Tabu Search

1) Distributed Tabu List: In each iteration of Tabucol algorithm, a set of neighborhoods is generated for current solution x in local search. While generating a new neighborhood x', if the new vertex color in x' has already been considered as a taboo in previous iterations, x' should be abandoned except in the case that that $\mu(x')<\mu(x)$ (aspiration criteria). Thus, checking whether the tabu list T includes a move $[x\rightarrow x']$ (we denote the color change for only one vertex as a move) is a key step in privacy-preserving tabu search.

Like Hertz[9], we define every taboo in T as a visited color for a specific vertex. If all parties share a global tabu list, the taboos in T may reveal private information (each parties' vertices and unallowed colors). To protect such information, we can let each party $P_a, a\in[0,m-1]$ establish a tabu list T_a to record the latest color of one of its vertices that leads to a critical move ($\mu(x)$ reduced move). We thus have

$T = \bigcup_{\forall a} T_a$. Intuitively, since each party manipulates its own tabu list, distributed tabu list does not breach privacy.

2) Synchronous Move in Local Search: Since tabu search is a local search based technique, the computation in local search (generating neighborhoods) is secure. For computing the number of conflicts and comparing results, secure scalar product and secure comparison protocols that employ homomorphic encryption can be used. However, potential inferences may still exist even after applying these secure computation protocols. For example, consider the following scenario:

Example 2 (Inference Scenario): As shown in Fig. 3(a), if Alice changes the color of v_2 from "Red" to "Green", the secure comparison result $\mu(x') - \mu(x) \geq 0$ is learnt by Alice. Thus, Alice can infer that the color of v_4 (belonging to Bob) is "Green".

The above inference results from the output that is revealed in each iteration (the result of comparing $\mu(x)$ and $\mu(x')$). We eliminate such inferences by slightly modifying local search. Specifically, if any border vertex v_i is chosen for move while generating neighborhoods, we allow another party to locally change the color of one of its vertices or opt to *do nothing*. We denote this mechanism as **synchronous move**. The inferences due to conflicting external edges and other parties' border vertex colors (i.e. Example 2) can be eliminated with synchronous move since both parties cannot learn the external edge conflict from the outputs of secure comparison.

With this modification, some generated neighborhoods may include two moves. Since the number of moves is at most 2, the performance of local search does not degrade noticeably – our experimental evaluation in Section V shows that the performance remains close to Tabucol algorithm [9].

To improve the search performance, the vertex (either inner or border vertex) whose color is changed is *preferentially chosen from the vertices that are the endpoints of edges with conflicts*. Since only the conflicting internal edges are known to corresponding parties, we can iteratively let each party change the color of the endpoints of its conflicting internal edges. When moving an inner vertex, the party can independently compare two unknown numbers $\mu(x')$ and $\mu(x)$ (by comparing its shares in $\mu(x')$ and $\mu(x)$) and check its tabu list along with specific operations on its vertices colors in x' (update x and tabu list with x', or abandon x'). On the other hand, moving a border vertex requires secure computation of all scalar products and secure comparison of $\mu(x')$ and $\mu(x)$. Hence, while changing a border vertex color by one party, our privacy-preserving tabu search protocol calls synchronous move, secure conflict computation and secure comparison. The result of secure comparison is only revealed to the two involved parties. These parties use this information to check and update their tabu lists along with specific operations on their vertices' colors in x'.

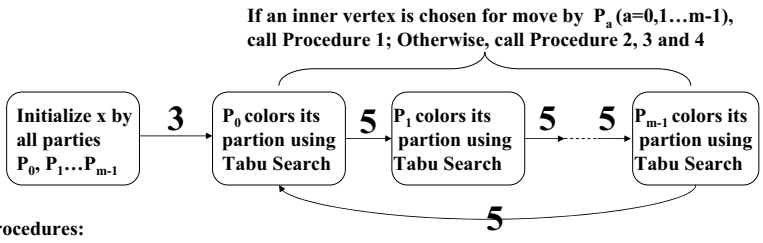
3) Privacy-preserving Tabu Search: In our secure tabu search protocol, each party iteratively executes Tabu search on its local graph to eliminate all the conflicting internal edges. (Each party's local graph is a subgraph of G. Hence, if the local graph is not k-colorable, the protocol can be terminated with output that G is not k-colorable). During local search, each party moves inner or border vertices (with synchronous move). If the party executing tabu search in the given iteration does not have any conflicting internal edge (local k-coloring is attained), this party may move one of its border vertices that triggers synchronous move, or skip its local search and pass control to the next party. In addition, in each move, tabu list is locally checked and aspiration criteria is applied to every solution (locally for inner vertex move or globally for synchronous move in two parties' tabu lists). Finally, in each iteration after a party finishes k-coloring of its local graph, secure comparison protocol is called to check if an optimal solution is obtained (i.e., $\mu(x) = 0$). Since the output of secure comparison is divided into $<$ and $\geq$, the algorithm compares $\mu(x)$ with 1. If $\mu(x) = 0$, the secure tabu search is terminated with k-colorable output. If $\mu(x) \geq 1$ after a given number of iterations, the tabu search is terminated with an output that G is not k-colorable. The detailed steps are presented in Fig. 4 and Algorithm 2.

Algorithm 2 Privacy-preserving Tabu Search

Input: m parties $P_0, \ldots, P_{m-1}$, Graph $G = (V, E)$;
1: initialize a solution x: randomly coloring all vertices by all parties;
2: call Secure Conflict Computation: each party hold a share of $\mu(x)$: $\forall a \in [0, m-1]$, $\mu_a(x)$;
3: **for** party P_a, $a \in [0, m-1]$ **do**
4: **while** the number of P_a's conflicting internal edges is greater than 0 **do**
5: P_a generates a neighborhood x' by changing the color of vertex v_i that is the endpoint of one of its conflicting internal edges: $x_i \rightarrow x'_i$;
6: **if** v_i is an inner vertex **then**
7: P_a locally compare its shares in $\mu_a(x)$ and $\mu_a(x')$ and check its tabu list T_a;
8: **if** $\mu_a(x') < \mu_a(x)$ **then**
9: let $x = x'$, update the tabu list T_a and continue;
10: **else**
11: pick a random party P_b ($b \neq a$) to possibly change the color of vertex v_j: $x_j \rightarrow x'_j$ (x' is generated by synchronous move);
12: call Secure Conflict Computation and Secure Comparison for $\mu(x')$ and $\mu(x)$;
13: **if** $\mu(x') < \mu(x)$ **then**
14: let $x = x'$, update the tabu list T_a, T_b by P_a, P_b respectively and continue;
15: if x's rep neighborhoods are generated (rep is given), choose a best neighborhood x' where $(v_i, x'_i) \notin T_a$, let $x = x'$ and update T_a (v_i is an inner vertex of P_a);
16: if P_a's partition in G cannot be k-colored in maximum number of iterations, terminates protocol with output: G is not k-colorable;
17: if P_a's partition is k-colored, P_a can move its border vertices to generate new k-coloring solutions (call synchronous move, secure conflict computation and secure comparison);
18: call Secure Comparison to compare $\mu(x)$ and 1 among all m parties;
19: if $\mu(x) < 1$, terminates protocol with output: G is k-colorable by solution x;
20: if $\mu(x) \geq 1$ after a given number of iterations, terminates protocol with output: G is not k-colorable;
21: $a = (a + 1) \mod m$;

C. Security Analysis

The exchanged messages of PPTS protocol occurs primarily in two sub-protocols: secure scalar product computation and secure comparison. Thus, the security of our algorithm depends on the security of these two algorithms. Given these secure protocols, the security proof comes down to demonstrating that the output of those protocols can be simulated knowing one's own input and the final results. The composition theorem [7] then enables completing the proof of security.



Procedures:

1. Locally Compare μ(x) and μ(x') by P_a: *if $\mu(x') < \mu(x)$, let $x=x'$ and update tabu list T_a by P_a*
2. Synchronous Move: *randomly choose another party P_b to possibly move one vertex color*
3. Secure Conflict Computation: *$\mu(x)$ or $\mu(x')$ is partitioned among all m parties*
4. Securely Compare μ(x) and μ(x'): *if $\mu(x') < \mu(x)$, let $x=x'$ and update tabu lists T_a, T_b by P_a, P_b*
5. Securely Compare μ(x) and 1*: if $\mu(x)<1$, terminate protocol with k-colorable output*

Fig. 4. Privacy-preserving Tabu Search Protocol

Theorem 1: PPTS privately solves the k-coloring problem where each party P_a learns:

1) Its share in the secure conflict computation
2) The secure comparison result in synchronous moves involving P_a

Proof: Since PPTS protocol is symmetric, proving that the view of one party can be simulated with its own input and output suffices to prove it for all parties. We now show how to simulate the messages received by an arbitrary party $P_a, a \in [0, m-1]$. While P_a is chosen to color its partition (this is iteratively done in PPTS), two cases occur.

First, if P_a changes the color of its inner vertices (Step 6-9), P_a learns only $\mu(x') < \mu(x)$ or $\mu(x') \geq \mu(x)$ by locally comparing its shares in $\mu(x')$ and $\mu(x)$. Other parties learn nothing (the number of iterations to color P_a's partition is also unknown to other parties). Hence, these can be simulated by simply executing those steps.

Second, if P_a moves one of its border vertices, a synchronous move is triggered. P_a now receives the share of secure conflict computation: s_{ij} (w.o.l.g. we let P_a be the receiver of s_{ij} rather than the creator of r_{ij}), and learns the secure comparison result of two solutions (x and x'): $\mu(x') < \mu(x)$ or $\mu(x') \geq \mu(x)$. We now examine P_a's view in Step 10-14 (and possibly include Step 17) and build a polynomial simulator that runs for the known synchronous moves as below:

- The share of every secure scalar product computation $s_{ij} = x_i \cdot x_j + r_{ij}$ can be simulated by generating a random from the uniform probability distribution over $\mathcal{F}$ (Note: we assume that s_{ij} are scaled to fixed precision over a closed field, enabling such a selection). Thus, $Prob[T = t] = Prob[s_{ij} = t] = Prob[r_{ij} = t - s_{ij}] = \frac{1}{\mathcal{F}}$, and the shares can be simulated in polynomial time.
- In every synchronous move, P_a receives $\mu(x') < \mu(x)$ or $\mu(x') \geq \mu(x)$. This cannot be directly simulated unless we know the result of the scalar product of two border vertices: $x_i \cdot x_j$. The simulator can generate any comparison results by changing the color of such border vertex. Since the probability of generating $\mu(x') < \mu(x)$ or $\mu(x') \geq \mu(x)$ based on moving border vertices is deterministic, this output can be simulated in polynomial time as well.

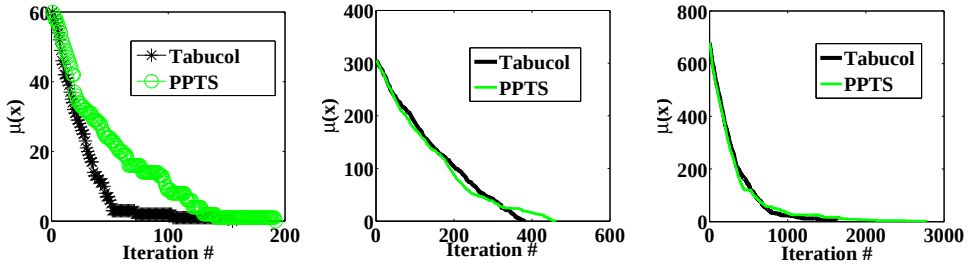
Thus, Theorem 1 has been proven. Note that, effectively, a party can learn at most the color of the border vertex of another party with which it shares an edge. This is useless in the intermediate stages (since the color is not fixed). In the final solution, since typically the number of shared border vertices is significantly smaller than the total number of vertices, the leaked information is not very significant. ■

There are several remaining issues. First, since our simulator is also a meta-heuristic, the worst case running time is not polynomial. This is a problem-dependent issue: if the problem is always solved by tabu search in polynomial time, then the simulator is guaranteed to be polynomial, and the protocol is secure. Second, some additional minor information is probably leaked. For example, in the output of PPTS, given k-colorable solution, the colors of P_a's adjacent vertices (belong to other parties) can be known (to P_a) with a different color from corresponding adjacent vertex. This is unavoidable in the DisGC problem (even if we build a trusted third party, it is learnt by other parties with the output). Moreover, the number of iteratively calling different parties to color its partition and the fact that $\mu(x) > 0$ until the last step are also revealed. These pieces of information are trivial and do not significantly impact the security of the PPTS protocol.

D. Communication and Computational Cost Analysis

Given a distributed graph with n vertices and n_e external edges, in PPTS protocol, we assume that m parties are repeatedly called to locally color their partition for c times and ℓ synchronous moves are executed during coloring of the partition by a party. Based on this, we can discuss the computation and computational cost of the PPTS protocol.

1) Communication Cost: Note that only secure conflict computation and secure comparison require communication of messages among parties. Since conflict computation for each external edge requires two communication messages, the total number of messages generated for secure conflict computation is $2c * (n_e * \ell + 1)$, where $O(n_e) = O(n^2)$. Thus, the communication complexity of secure scalar product computation in



(a) #V=100, ρ=10%, k=10 (b) #V=300, ρ=20%, k=30 (c) #V=500, ρ=30%, k=40

Fig. 5. Total Number of Conflicts ($\mu(x)$) in each Iteration

PPTS is $O(cn^2\ell)$. Moreover, every secure comparison requires $O(m)$ parties to compute the output. Since the number of communication messages in every secure comparison equals to the number of computing parties [11] and PPTS includes $c*(\ell+1)$ secure comparison, the communication complexity of secure comparison in PPTS is $O(cm\ell)$.

2) Computation Cost: Since each party triggers ℓ synchronous moves while coloring its partition, $c*\ell+1$ secure conflict computations and $c*(\ell+1)$ secure comparisons are required in PPTS. If the runtime of a single secure scalar product computation and a single secure comparison is denoted by t_s and t_c respectively, the additional cost can be estimated as $(c*\ell*n_e+1)*t_s+c*(\ell+1)*t_c$. Thus, the computational complexity of PPTS is $O(cn^2\ell)$ for secure scalar product and $O(c\ell)$ for secure comparison.

V. Experimental Evaluation

A. Experiments Setup

For the experiments, we randomly generate graphs with the number of vertices N and the density of the graph ρ (probability of the existence of an edge between any two given vertices). Each generated graph is partitioned among 10 parties ($P_0, P_1, \ldots, P_9$), we let each party hold N/10 vertices. In each test, we generate 10 graphs with the same N and ρ, and report the average of the results.

We compare our privacy-preserving tabu search (PPTS) algorithm with Tabucol [9] using the same group of graphs.Both algorithms are terminated if no k-colorable solution can be found in 10^5 iterations. We let each party hold a tabu list with length $N/10$ in PPTS while the overall tabu list in Tabucol is established with length $N/10$. In synchronous move of PPTS algorithm, we assume that P_b (the second party) has 50% chance of skipping a move. Futhermore, in PPTS algorithm, Paillier cryptosystem [14] with 512-bit and 1024-bit key is used for homomorphic encryption (securely computing number of conflicts). For FairplayMP [2], sufficiently large modulus length for 10 parties (i.e. 128 bits) is used for secure comparison. All the experiments are performed on an HP machine with Intel Core 2 Duo CPU 3GHz and 6GB RAM.

Our goal is to validate two hypotheses: 1. the optimal results (chromatic number) of solving DisGC problem using PPTS are close to the results of Tabucol (this indicates that our approach is as accurate as any general meta-heuristic for graph coloring; 2. the computation time of PPTS is reasonable. Thus, for the first goal, we compute the chromatic number of distributed graphs with varying number of vertices #V={100, 200, 300, 500, 1000} and for two different densities ρ=10% and 30% using PPTS and Tabucol (for each pair of #V and ρ, we generate 10 distributed graphs; 512-bit key is used). For the second goal, first, we compute the runtime of PPTS with 512-bit key and with 1024-bit key and compare it with Tabucol for varying #V={100, 200, 300, 500, 1000} and fixed ρ=20% (Fig. 6b); second, we perform the comparison between the exection time of PPTS and Tabucol for varying $\rho=\{2\%, 5\%, 10\%, 20\%, 30\%\}$ and fixed #V=500 (Fig. 6c).

B. Experimental Results

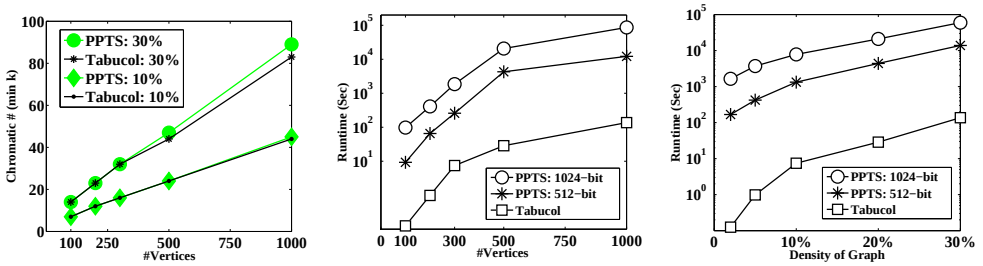
1) PPTS Solution Quality: Fig. 5 depicts the number of total conflicts $\mu(x)$ and the number of iterations required by PPTS and Tabucol for three different graphs (k is set slightly greater than the optimal chromatic number). While solving DisGC on these three graphs, the improvements (reduced $\mu(x)$) in each iteration of both approaches are similar. PPTS generally requires more iterations than Tabucol. This is unavoidable because due to the privacy concern, the algorithm does not know which external edges cause conflict, and therefore cannot improve only upon the conflicting edges. Thus, the search performance is slightly lower, but still acceptable.

Fig. 6(a) shows the chromatic number (min k) for distributed graphs with different size and density (for each pair of #V and ρ, we randomly generate 10 graphs, solve them with two algorithms and different k, and choose the min k). In general, the optimum performance of the two algorithms is similar.

2) PPTS Efficiency: As the communication cost is small compared to the computation cost (each single computation consumes significantly more time), we evaluate the efficiency of PPTS with respect to computation time. As shown in Fig. 6(b) and 6(c), the runtime of PPTS with 512-bit and 1024-bit is not excessive (approximately 30 hours are required to securely solve a 10-party distributed graph coloring problem with 1000 vertices and density 20%). This can also be improved by using cryptographic accelerators or parallelizing the computation.

VI. Conclusion and Future Work

In this paper, we have presented a privacy-preserving tabu search method to solve the distributed graph coloring problem with limited information disclosure. Our solution creates a



(a) Chromatic # (min k) (b) Runtime on Graph Size (c) Runtime on Density

Fig. 6. Optimal Results and Computational Cost Testing

privacy-preserving version of the existing Tabucol algorithm by making use of several cryptographic building blocks in novel ways. It also provides a roadmap for generalizing the solution to other distributed combinatorial optimization problems. Given the iterative nature of the solution, we also identify the possible inference attacks on the output in each iteration, and propose a synchronous move mechanism to resolve such problems. We analyze the computation and communication complexity of our modified tabu search, and experimentally show that the PPTS protocol provides nearly identical optimum performance as Tabucol without sacrificing efficiency for privacy.

There are several interesting future directions. While we examine a specific data partitioning for the DisGC problem, other data partitions are also possible. For example, perhaps a party may hold the information about graph edges rather than graph vertices (this could happen, for example in scheduling). We plan to explore this problem and develop solutions for it. Secondly, while graph coloring has many applications, there are several other combinatorial optimization problems, such as knapsack and vehicle routing, that are naturally distributed. We plan to explore how our privacy-preserving tabu search can be applied in these domains. Finally, while tabu-search is generally quite useful, it does not work well for all applications. There are several other algorithms such as simulated annealing, or genetic algorithms that also enjoy wide usage. We will look at how such algorithms could be made privacy-preserving in the future.

VII. Acknowledgements

This work is supported in part by the National Science Foundation under Grant No. CNS-0746943 and the Trustees Research Fellowship Program at Rutgers, the State University of New Jersey.

References

[1] M. J. Atallah, V. Deshpande, H. G. Elmongui, and L. B. Schwarz. Secure supply-chain protocols. In *Proceedings of the 2003 IEEE International Conference on E-Commerce*, Newport Beach, California, June24-27 2003.

[2] A. Ben-David, N. Nisan, and B. Pinkas. Fairplaymp: a system for secure multi-party computation. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 257–266, New York, NY, USA, 2008. ACM.

[3] O. Catrina and S. de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In *ESORICS*, pages 134–150, 2010.

[4] W. Du. *A Study of Several Specific Secure Two-party Computation Problems*. PhD thesis, Purdue University, West Lafayette, Indiana, 2001.

[5] F. Glover. Tabu search - part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[6] F. Glover. Tabu search - part ii. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[7] O. Goldreich. *The Foundations of Cryptography*, volume 2, chapter Encryption Schemes. Cambridge University Press, 2004.

[8] W. D. Y. S. Han and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *In Proceedings of the 2004 SIAM International Conference on Data Mining*, 2004.

[9] A. Hertz and D. D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4), 1987.

[10] Y. Hong, J. Vaidya, and H. Lu. Efficient distributed linear programming with limited disclosure. In *DBSec*, pages 172–187, 2011.

[11] I. Ioannidis and A. Grama. An efficient protocol for yao's millionaires' problem. In *Hawaii International Conference on System Sciences (HICSS-36)*, pages 205–210, Waikoloa Village, Hawaii, Jan. 6-9 2003.

[12] J. Li and M. J. Atallah. Secure and private collaborative linear programming. In *Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on*, pages 1 –8, 2006.

[13] D. Marx and D. A. Marx. Graph coloring problems and their applications in scheduling. In *in Proc. John von Neumann PhD Students Conference*, pages 1–2, 2004.

[14] P. Paillier. Public key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS 1592*, pages 223–238, 1999.

[15] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[16] J. Sakuma and S. Kobayashi. A genetic algorithm for privacy preserving combinatorial optimization. In *GECCO*, pages 1372–1379, 2007.

[17] M. C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. *iat*, 00:531–535, 2004.

[18] M.-C. Silaghi and V. Rajeshirke. The effect of policies for selecting the solution of a discsp on privacy loss. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1396–1397, 2004.

[19] K. Suzuki and M. Yokoo. Secure generalized vickrey auction using homomorphic encryption. *Lecture Notes in Computer Science*, 2742.

[20] E. Turban, R. K. Rainer, and R. E. Potter. Introduction to information technology, chapter 2nd, information technologies: Concepts and management. *John Wiley and Sons, 3rd edition*.

[21] J. Vaidya. Privacy-preserving linear programming. In *SAC*, pages 2002–2007, 2009.

[22] J. Vaidya. A secure revised simplex algorithm for privacy-preserving linear programming. In *AINA '09: Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications*, 2009.

[23] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, Los Alamitos, CA, USA, 1986. IEEE, IEEE Computer Society.

[24] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *In Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP-2002)*, 2002.