Two NP-Complete Problems in Software Security

Boting Yang, Philip W. L. Fong

Department of Computer Science, University of Regina, Regina, Canada
{boting, pwlfong}@cs.uregina.ca

Abstract

A secure programming environment must offer protection mechanisms for regulating the interaction between mutually suspicious code units. Discretionary Capability Confinement (DCC) is a static type system recently proposed for expressing and enforcing access control policies in Java programs. This paper considers the computational complexity of the type reconstruction problem for DCC, that is, the automatic inference of DCC annotations for legacy Java code base. We model a representative subproblem of the type reconstruction problem as a graph-theoretic optimization problem. We demonstrate that this problem is NP-Complete. To strengthen the result, we further eliminate the asymmetry in the problem statement to obtain an elegant reformulation. We show that the reformulated problem is still NP-Complete. These results help identify the complexity core of the type reconstruction problem, and suggest that practical type reconstruction tool must adopt a heuristic and/or approximation approach.

1. Introduction

Software systems in the Internet era carry out computation on behalf of multiple principals. Mobile code, scriptable applications, and plug-in architectures all involve the execution of untrusted program fragments alongside trusted ones. More than ever, we find, within a single process, multiple code units that are mutually suspicious of one another. In layman's language, the left hand does not trust the right.

Discretionary Capability Confinement (DCC) is a static type system recently proposed [2, 3] for expressing and enforcing access control policies in Java programs. Programmers express their access control policies in the form of program annotations. The DCC typing discipline ensures that the access control policies of different code units are consistent with one another, and with the semantics of the Java programming model. When mutually suspicious code units are dynamically linked together at the time of class loading [6], a static verifier checks that the annotations of the code units comply to the typing rules of DCC. It has been proven that the DCC typing rules guarantees a number of confinement properties. A DCC type checker has been fully implemented in a standard Java platform.

One objection to the current design of DCC is that it is applicable only to new programming projects that are built from scratch. For legacy systems, one will have to retrofit DCC annotations back into the source code, which can be a tedious and labor-intensive undertaking. Therefore, it is desirable to have an algorithm for automatically inferring a reasonable set of DCC annotations from a legacy Java code base. The resulting annotations, though not necessarily perfect, serve as a starting point for further, manual annotations. This automatic inference of DCC annotations is called the *type reconstruction problem* of DCC.

In this paper, we specify two simplified versions of the type reconstruction problem, and demonstrate that they are NP-Complete. The simplified problems represent the complexity core of the type reconstruction problem. Such results help us to better understand the computational nature of DCC type reconstruction, and call for the adoption of heuristic and/or approximation approaches when type reconstruction tools are to be developed.

We will model two versions of the type reconstruction problem as optimization problems in digraphs. A *digraph* (or directed graph) G consists of a finite non-empty set $V(G)$ of vertices and a finite set $A(G)$ of ordered pairs of distinct vertices, called *arcs*. We denote the arc from x to y by (x, y). A *directed path* on n vertices has an n-element vertex set $\{v_1, v_2, \ldots, v_n\}$ and arcs (v_i, v_{i+1}) for $1 \leqq i \leqq n-1$. A *directed cycle* on n vertices has an n-element vertex set $\{v_1, v_2, \ldots, v_n\}$ and arcs (v_i, v_{i+1}) for $1 \leqq i \leqq n-1$, as well as the arc (v_n, v_1). A digraph is *acyclic* if it contains no directed cycles. An acyclic digraph is an analogue of a tree in undirected graphs. A digraph G is *strong* if there exists a directed path from vertex x to vertex y for all possible choices of vertices x and y in G. A *strong component* of G is a

Research was supported in part by NSERC.

$(\mathcal{DCC}1)$: A can invoke a static method declared in B only if $B \triangleright A$.
$(\mathcal{DCC}2)$: (i) A can generate[a] a reference of type C only if $C \triangleright A$. (ii) B can share[b] a reference of type C with A only if $C \triangleright A \vee A \bowtie B$.
$(\mathcal{DCC}5)$: A can be declared as a subtype of B only if $B \triangleright A$.
$(\mathcal{DCC}6)$ (ii) & (iii): Method $B.n$ can be overridden by method $B'.n'$ only if the following hold: (ii) If the method return type is C, then $C \triangleright B \vee B \bowtie B'$; (iii) For every formal parameter type C, $C \triangleright B' \vee B \bowtie B'$.
Remarks: The typing rules $(\mathcal{DCC}3)$, $(\mathcal{DCC}4)$, $(\mathcal{DCC}6)$ (i), and $(\mathcal{DCC}7)$ as specified in [2, 3] have been omitted from this core set.

[a] *A generates* a reference of type C when one of the following occurs: (1) A creates an instance of C; (2) A dynamically casts a reference to type C; (3) an exception handler in A with catch type C catches an exception.
[b] *B shares* a reference of type C with A when one of the following occurs: (1) A invokes a method declared in B with return type C; (2) A reads a field declared in B with field type C; (3) B writes a reference into a field declared in A with field type C.

Figure 1: A core subset of DCC typing rules that are modeled in this work.

maximal induced subdigraph of G that is strong. A single vertex is taken to be a strong subdigraph. If G_1, G_2, ..., G_m are all the strong components of G, then $V(G_1)$, $V(G_2)$, ..., $V(G_m)$ form a partition of $V(G)$.

This paper is organized as follows. In Section 2, we formulate two versions of the type reconstruction problem as graph-theoretic optimization problems. In Sections 3 and 4, we show that both optimization problems are NP-complete respectively. Finally, we conclude this paper by discussions in Section 6.

2. Formulation of the type reconstruction problem

DCC is an attempt to integrate into the Java programming language the support for an access control paradigm known as *capabilities* [1]. A capability is an unforgeable pair consisting of (i) the designation of a resource (e.g., memory address, pointer, reference), and (ii) an access control interface specifying a constrained view of the underlying resource. In a capability-based programming system, the possession of a capability is the necessary and sufficient condition for access to the underlying resource. In a type-safe object-oriented programming language such as Java, a statically-typed object reference can be seen as a capability: (i) the object reference is an unforgeable designation to the underlying object; (ii) the static type of the reference, which is guaranteed to be a supertype of the actual run-time type of the underlying object, presents a constrained view through which the underlying object can be accessed by program code. The DCC typing rules specify a collection of conditions under which the annotated Java code is consistent with a capability-based programming model inspired by the above reference-as-capability metaphor.

One of the access control challenge that DCC is attempting to address is that of capability confinement. Once a capability is propagated from one program unit to another, how does one ensure that the latter will not further propagate the capability to an untrusted third party? Behind the typing discipline of DCC are two key concepts, namely, trust and confinement domains. Implicitly induced by the user-supplied annotations is a binary *trust* relation between declared types (i.e., Java classes and interfaces). Intuitively, declared type A is said to trust declared type B, written $A \triangleright B$, whenever B is allowed to freely acquire an object reference of type A. Otherwise, A is considered to be a capability type from the perspective of B, and thus the DCC typing rules impose restrictions on the ability of B to acquire a type-A object reference. The trust relation is a preorder (i.e., reflexive and transitive). If both $A \triangleright B$ and $B \triangleright A$, then we write $A \bowtie B$. Obviously, $\bowtie$ is an equivalence relation. Equivalence classes induced by $\bowtie$ are called *confinement domains*. Intuitively, the propagation of an instance of a declared type within a confinement domain $\mathcal{D}$ is freely allowed, but the typing rules impose constraints on when a capability type may escape from a confinement domain.

Figure 1 specifies a core subset of DCC typing rules that are modeled in this paper. A discussion of how such a subset is chosen is deferred to the concluding section. For now, it suffices to notice that the DCC typing rules are *monotonic*, in the sense that adding back the remaining typing rules (or a subset of which) only removes programs from consideration, but never make new programs admissible. While most of the rules in Figure 1 are technical in nature, $(\mathcal{DCC}2)$ stands out as the cornerstone of capability propagation control. Intuitively, it says two things: (i) the forging of capability references is categorically forbidden; (ii) retrieving and storing capabilities across the boundary of a confinement domain are not allowed. The rule implies that capabilities can only escape from a confinement domain through argument passing — an event that is guarded by the conscious discretion of the programmer (thus Discretionary Capability Confinement).

Given the bytecode of a compiled Java program, the type reconstruction process consists of two steps:

1. Scan through the bytecode to obtain a set of type constraints as mandated by the DCC typing rules. Each constraint is in the form of either $(B \triangleright A)$ or $(C \triangleright A \vee A \bowtie B)$, where A, B and C are concrete declared type identifiers.

2. Find an assignment of declared types to confinement domains, so that all the constraints obtained in Step 1 are satisfied, while maintaining the reflexivity and transitivity of the $\triangleright$ relation.

Note that the type reconstruction problem shall not be formulated as a satisfying problem. There is always a degenerate annotation that satisfies all the DCC typing rule: assign all declared type to the same confinement domain[1]. Instead, we seek to formulate the type reconstruction problem as an *optimization* problem. What then is a reasonable objective function? By allocating two declared types to distinct confinement domains, we limit the interactions between the two. Therefore, to promote security, we maximize the number of confinement domains[2].

We now formulate the above problem as a graph-theoretic optimization problem. We use vertices to represent declared type identifiers, and use arcs to represent trust relationships between declared type identifiers. Then we obtain a digraph G, in which directed cycles of length 2 correspond to relation $\bowtie$, and strong components correspond to confinement domains. Specifically, in the first step of the type reconstruction process, each constraint in the form of $(B \triangleright A)$ corresponds the arc (b, a) in the digraph G, and each constraints in the form of $(C \triangleright A \vee A \bowtie B)$ corresponds to a pair $((c, a), (a, b, a))$, where a, b and c are vertices in G represent the declared type identifiers A, B and C respectively, and (a, b, a) is a directed cycle of length 2.

Definition 2.1 *Given a digraph $G(V, A)$ and its complement $\overline{G}(V, \overline{A})$, let L be a set of (arc, 2-cycle) pairs such that each pair has the format $((c, a), (a, b, a))$, where $a, b, c \in V$ and (c, a), (a, b), $(b, a) \in A \cup \overline{A}$. We call this L an AC-set of $G(V, A)$.*

Notice that for each element $((c, a), (a, b, a)) \in L$, the head of the arc is a vertex in the directed 2-cycle, and (a, b), (b, c) and (c, b) are not necessarily in A.

Given a digraph $G(V, A)$ and an AC-set L of $G(V, A)$, for each (arc, 2-cycle) pair in L, select at least one element from the pair and add it to G. How to arrange the selections and additions such that the new digraph has the maximum number of strong components? We call this problem the *Maximum Strong Components problem* (MAX-SC).

We postulate that the hardness of MAX-SC arises not from the asymmetric structure of the constraints, but rather the disjunctive nature of the constraints. To demonstrate this, we study a variant of MAX-SC. Specifically, we eliminate the asymmetric structure of a constraint and simplify each member of an AC-set to a pair of arcs. Then we have another version of the type reconstruction problem: Given a digraph $G(V, A)$ and a set L of pairs of arcs in $\overline{A}$ ($\overline{A}$ is the edge set in the complement of G), we want to find a set of arcs $S \subseteq \overline{A}$ such that for each element (i.e., pair of arcs) in L, at least one arc in the element belongs to S and the digraph $G^*(V, A \cup S)$ has the maximum number of strong components. We call this problem the *Simplified* MAX-SC *problem* (S-MAX-SC).

3. MAX-SC problem is NP-complete

The decision version of MAX-SC problem can be described as follows.

Problem: Maximum Strong Components (MAX-SC)
Instance: A digraph $G(V, A)$, an AC-set L of $G(V, A)$, and a positive integer K.
Question: Is there a set S of arcs and 2-cycles from the elements in L such that (1) for each pair (arc, 2-cycle) in L, the arc, or the 2-cycle, or both of them belong to S, and (2) the digraph obtained by adding all arcs and 2-cycles in S to G has at least K strong components?

In order to show the NP-hardness of MAX-SC problem, we need first introduce the 1-IN-3 SAT problem.

Problem: 1-IN-3 SAT
Instance: A Boolean formula ϕ in conjunctive normal form such that each clause contains three positive literals.
Question: Is there a satisfiable assignment for ϕ such that each clause in ϕ has exactly one true literal?

Schaefer showed that 1-IN-3 SAT problem is NP-complete by a reduction from 3SAT [7, 8]. In order to prove MAX-SC problem is NP-complete, we define a variant of 1-IN-3 SAT problem as follows:

Problem: 1-IN-(2,3) SAT
Instance: A Boolean formula ϕ in conjunctive normal form such that each clause contains either three positive literals or one positive literal and one negative literal, and each variable either appears only once as a positive literal in a 3-literal clause, or appears exactly 3 times, once as a negative literal in a 2-literal clause, and twice as a positive literal in a 3-literal

[1]The design of DCC is backward compatible to the original Java programming language. If no further annotations are provided, all declared types are assigned to the same domain by default. The DCC typing discipline is designed to render such a program admissible.

[2]There is a trivial upper bound to this number — each declared type belonging to its own domain.

clause and a 2-literal clause respectively.
Question: Is there a satisfiable assignment for ϕ such that each clause in ϕ has exactly one true literal?

We can show that 1-IN-(2,3) SAT problem is NP-complete by a reduction from 1-IN-3 SAT problem (the proof is omitted due to the space limit).

Lemma 3.1 1-IN-(2,3) SAT *problem is NP-complete.*

PROOF. It is easy to see that 1-IN-(2,3) SAT problem belongs to NP. We will show that 1-IN-(2,3) SAT problem is NP-hard by transforming 1-IN-3 SAT to 1-IN-(2,3) SAT. Let ϕ be an instance of 1-IN-3 SAT problem, i.e., a Boolean formula in conjunctive normal form such that each clause contains three positive literals. For any variable x appearing $k(>1)$ times in ϕ, we replace the first occurrence of x by a new variable x_1, replace the second occurrence of x by a new variable x_2, and replace the kth occurrence of x by a new variable x_k. In order to force $x_1, x_2, \ldots, x_k$ take the same truth value, we add k new clauses to ϕ: $(\overline{x}_1 \vee x_2) \wedge (\overline{x}_2 \vee x_3) \wedge \cdots \wedge (\overline{x}_k \vee x_1)$. After we replace each variable appearing at least two times in ϕ, we obtain a new Boolean formula ψ in conjunctive normal form such that each clause contains either three positive literals or one positive literal and one negative literal, and each variable either appears only once as a positive literal in a 3-literal clause, or appears exactly 3 times, once as a negative literal in a 2-literal clause, and twice as a positive literal in a 3-literal clause and a 2-literal clause respectively. It is easy to see that this transformation can be done in polynomial time.

If there exists a truth assignment for ϕ such that each clause in ϕ has exactly one true literal, then for each variable x in ϕ, we assign the truth value of x in ϕ to every replacement variable x_i, $1 \leq i \leq k$, in ψ. Thus, there exists a truth assignment for ψ such that each clause in ψ has exactly one true literal. Conversely, suppose that there exists a truth assignment for ψ such that each clause in ψ has exactly one true literal. Consider the sub-formula $(\overline{x}_1 \vee x_2) \wedge (\overline{x}_2 \vee x_3) \wedge \cdots \wedge (\overline{x}_k \vee x_1)$. Since this sub-formula is true, we know that each variable x_i, $1 \leq i \leq k$, has the same truth value. We assign this truth value to the variable x in ϕ. Therefore, there exists a truth assignment for ϕ such that each clause in ϕ has exactly one true literal. ■

We now prove the main result of this section from Lemma 3.1.

Theorem 3.2 MAX-SC *problem is NP-complete.*

PROOF. It is easy to see that MAX-SC belongs to NP. We will show that MAX-SC problem is NP-hard by proving a reduction of 1-IN-(2,3) SAT to MAX-SC. Let $\psi = C_1 \wedge C_2 \wedge \cdots \wedge C_m \wedge C'_1 \wedge C'_2 \wedge \cdots \wedge C'_{m'}$ be an instance of 1-IN-(2,3) SAT problem such that ψ consists of $m + m'$ clauses and n variables in conjunctive normal form such that C_i, $1 \leq i \leq m$, contains three positive literals and C'_i, $1 \leq i \leq m'$, contains one positive literal and one negative literal, and each variable x_i, $1 \leq i \leq n$, either appears only once as a positive literal in a 3-literal clause, or appears exactly 3 times, once as a negative literal in a 2-literal clause, and twice as a positive literal in a 3-literal clause and a 2-literal clause respectively. Let n' be the number of variables that appears exactly 3 times. Without loss of generality, suppose that each x_i, $1 \leq i \leq n'$, appears exactly 3 times, and each x_i, $n'+1 \leq i \leq n$, appears only once. Let $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$, $1 \leq i \leq m$, and $C'_i = l_{i1} \vee l_{i2}$, $1 \leq i \leq m'$.

We now construct an instance of MAX-SC problem, that is, a digraph $G = (V, A)$, an AC-set L of $G(V, A)$ and a positive integer K. For each clause C_i, $1 \leq i \leq m$, we define

$$V_i = \{t_{i1}, h_{i1}, t_{i2}, h_{i2}, t_{i3}, h_{i3}\},$$

$$A_i = \{(t_{i1}, h_{i1}), (t_{i2}, h_{i2}), (t_{i3}, h_{i3})\},$$

$$L_i = \{((h_{i3}, t_{i1}), (t_{i1}, h_{i1}, t_{i1})), ((h_{i1}, t_{i2}), (t_{i2}, h_{i2}, t_{i2})), ((h_{i2}, t_{i3}), (t_{i3}, h_{i3}, t_{i3}))\}.$$

For each clause C'_i, $1 \leq i \leq m'$, we define

$$V'_i = \{p_{i1}, q_{i1}, p_{i2}, q_{i2}\},$$

$$A'_i = \{(p_{i1}, q_{i1}), (p_{i2}, q_{i2})\},$$

$$L'_i = \{((q_{i2}, p_{i1}), (p_{i1}, q_{i1}, p_{i1})), ((q_{i1}, p_{i2}), (p_{i2}, q_{i2}, p_{i2}))\}.$$

Without loss of generality, we can assume that no clause in ψ contains a literal and its negation. We now consider each variable x_i, $1 \leq i \leq n'$, that appears exactly 3 times, once as a negative literal in some 2-literal clause, and twice as a positive literal in some 3-literal clause and 2-literal clause respectively. Let $(a_1, b_1) \in A_j$, $1 \leq j \leq m$, and $(a_2, b_2) \in A'_{j'}$, $1 \leq j' \leq m'$, be the two arcs corresponding to x_i, and $(c, d) \in A'_k$, $1 \leq k \leq m'$, be the arc corresponding to $\overline{x}_i$. As illustrated in Figure 2, we define

$$V''_i = \{a_1, b_1, a_2, b_2, c, d, a'_1, a'_2, c', c''\},$$

$$A''_i = \{(a_1, a'_1), (a'_1, d), (a_2, a'_2), (a'_2, d), (c, c'), (c', b_1), (c, c''), (c'', b_2)\},$$

$$L''_i = \{((d, a_1), (a_1, b_1, a_1)), ((b_1, c), (c, d, c)), ((d, a_2), (a_2, b_2, a_2)), ((b_2, c), (c, d, c))\}.$$

Finally, we define

$$V = (\bigcup_{i=1}^{m} V_i) \cup (\bigcup_{i=1}^{m'} V'_i) \cup (\bigcup_{i=1}^{n'} V''_i),$$

$$A = (\bigcup_{i=1}^{m} A_i) \cup (\bigcup_{i=1}^{m'} A'_i) \cup (\bigcup_{i=1}^{n'} A''_i),$$

$$L = (\bigcup_{i=1}^{m} L_i) \cup (\bigcup_{i=1}^{m'} L'_i) \cup (\bigcup_{i=1}^{n'} L''_i).$$

We now finish the construction of $G(V, A)$ and L with $|V| = 6m + 4m' + 4n'$, $|A| = 3m + 2m' + 8n'$ and $|L| = 3m + 2m' + 4n'$. Since each 2-cycle in L contains an arc in A that corresponds to a literal, the 2-cycle can be considered also corresponding to this literal and the variable in this literal. In the following proof, we will see that each (arc, 2-cycle) pair in L can be considered as a truth assignment pair such that selecting "2-cycle" for S corresponds to assign *true* to the corresponding literal, and not selecting "2-cycle" corresponds to assign *false* to the corresponding literal. We now finish the reduction by setting $K = 5m + 3m'$. Figure 2 illustrates the construction of G and L. It is easy to see that we can construct the above G and L in polynomial time.

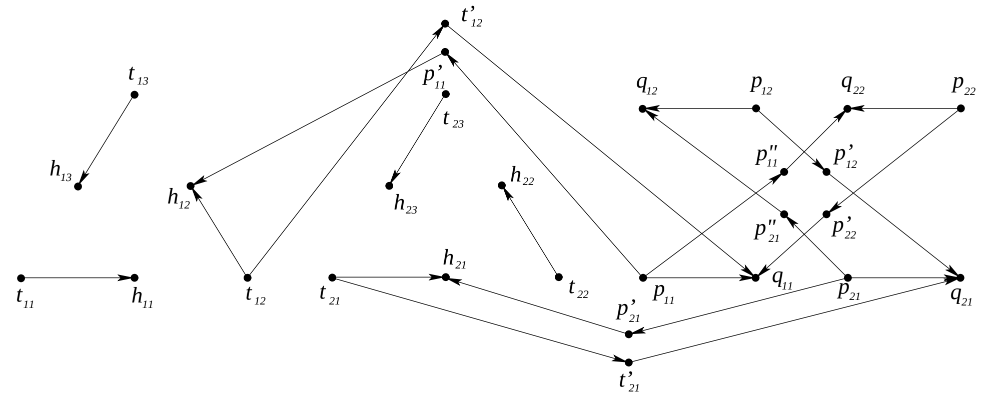


Figure 2: The graph $G(V, A)$ constructed from an instance of 1-IN-(2,3) SAT: $(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_4 \vee x_2)$.

We claim that there exists a satisfying truth assignment for ψ such that each clause in ψ has exactly one true literal if and only if there exists a set S of arcs and 2-cycles satisfying the following three conditions: (1) Each arc or 2-cycle in S is contained in some pair in L; (2) for each (arc, 2-cycle) pair in L, either the arc or the 2-cycle must belong to S; and (3) the digraph G^* obtained by adding S to G has at least K strong components. Let $\langle V_i \rangle$, $\langle V'_i \rangle$ and $\langle V''_i \rangle$ be subgraphs of G^* induced by V_i, V'_i, V''_i, respectively. So $\langle V_i \rangle$ corresponds to clause C_i for $1 \le i \le m$, $\langle V'_i \rangle$ corresponds to clause C'_i for $1 \le i \le m'$, and $\langle V''_i \rangle$ corresponds to variable x_i for $1 \le i \le n'$.

We first suppose that there exists a satisfying truth assignment for ψ such that each clause in ψ has exactly one true literal. For each true literal in ψ, we put its corresponding 2-cycle into S (suppose S is empty initially). For each (arc, 2-cycle) pair in L, if there is no true literal in ψ corresponding to this 2-cycle, then we put the arc in this pair into S. It is easy to see that this S satisfies conditions (1) and (2). We now check the number of strong components in G^*. For each C_i, $1 \le i \le m$, it contains exactly one true literal. Thus, $\langle V_i \rangle$ consists of a 2-cycle and a directed path of length 4 leaving from a vertex in the 2-cycle. Thus $\langle V_i \rangle$, $1 \le i \le m$, has 5 strong components. For each C'_i, $1 \le i \le m'$, it also contains exactly one true literal. Thus, $\langle V'_i \rangle$ consists of a 2-cycle and a directed path of length 2 leaving from a vertex in the 2-cycle. Thus $\langle V'_i \rangle$, $1 \le i \le m'$, has 3 strong components. For each x_i, $1 \le i \le n'$, since it appears exactly 3 times, once as a negative literal in some 2-literal clause, and twice as a positive literal in some 3-literal clause and 2-literal clause respectively, let $(a_1, b_1) \in A_j$,$1 \le j \le m$, and $(a_2, b_2) \in A'_{j'}$, $1 \le j' \le m'$, be the two arcs corresponding to x_i, and let $(c, d) \in A'_k$, $1 \le k \le m'$, be the arc corresponding to $\overline{x_i}$ (see Figure 3(a)). We have two cases regarding to the truth value of x_i.

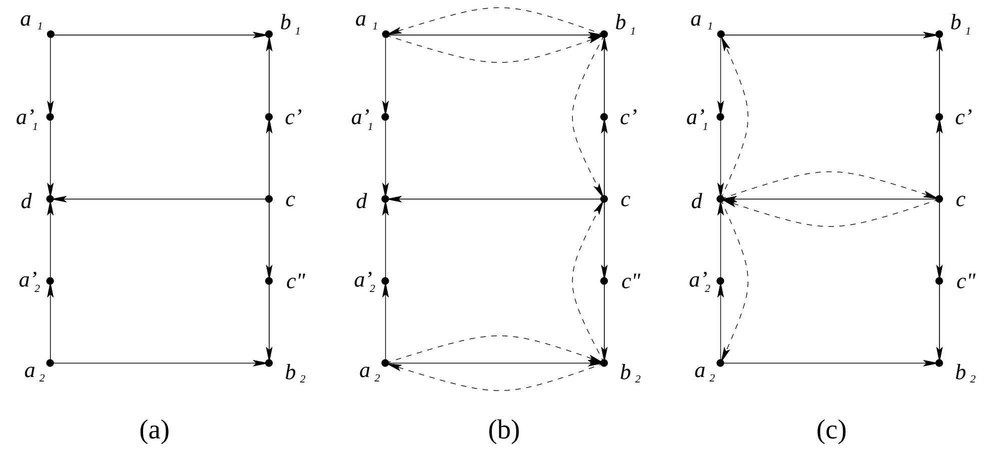


Figure 3: (a) The subgraph of G induced by $\{a_1, b_1, a_2, b_2, c, d, a'_1, a'_2, c', c''\}$. (b) When x_i is *true*, $\langle V''_i \rangle$ contains 4 strong components. (c) When x_i is *false*, $\langle V''_i \rangle$ contains 5 strong components.

CASE 1. If x_i is *true*, then $\langle V''_i \rangle$ contains 4 strong components: $\{d\}$, $\{a'_1\}$, $\{a'_2\}$ and the subgraph induced by $\{a_1, b_1, a_2, b_2, c, c', c''\}$ (see Figure 3(b)). In this case, 2-cycle (a_1, b_1, a_1) contributes one strong component to $\langle V_j \rangle$, 2-cycle (a_2, b_2, a_2) contributes one strong component to $\langle V'_{j'} \rangle$, and $\{c\}$ and $\{d\}$ contribute two strong components to $\langle V'_k \rangle$.

CASE 2. If x_i is *false*, then $\langle V''_i \rangle$ contains 5 strong components: $\{b_1\}$, $\{b_2\}$, $\{c'\}$, $\{c''\}$ and the subgraph induced by $\{a_1, a_2, c, d, a'_1, a'_2\}$ (see Figure 3(c)). In this case, 2-cycle (c, d, c) contributes one strong component to $\langle V'_k \rangle$, $\{a_1\}$ and $\{b_1\}$ contribute two strong components to $\langle V_j \rangle$, and $\{a_2\}$ and $\{b_2\}$ contribute two strong components to $\langle V'_{j'} \rangle$.

From the above two cases, we only need to count the number of strong components in $\langle V_i \rangle$, $1 \le i \le m$, and $\langle V'_i \rangle$, $1 \le i \le m'$. Hence, the number of strong components in G^* is $5m + 3m'$. Therefore, S satisfies conditions (3) (see Figure 4).

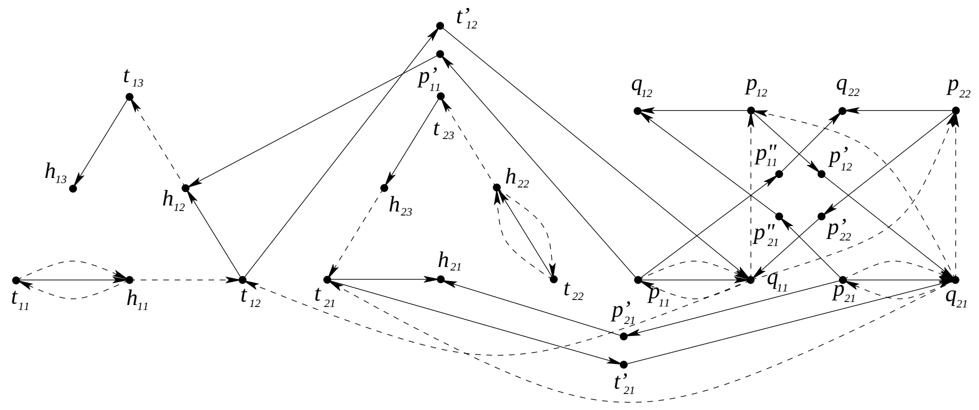


Figure 4: Suppose x_1 and x_5 are *True* and all other variables are *False* for $(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_4 \vee x_2)$. The dashed arcs and 2-cycles belong to S. G^* has 16 strong components.

Conversely, suppose that there exists a set S of arcs satisfying the above three conditions. For each subgraph $\langle V_i \rangle$, $1 \le i \le m$, it follows from conditions (1) and (2) that $\langle V_i \rangle$ has at most 5 strong components. Similarly, each $\langle V'_i \rangle$, $1 \le i \le m'$, has at most 3 strong components. For each $\langle V''_i \rangle$, $1 \le i \le n'$, let $(a_1, b_1) \in A_j$,$1 \le j \le m$, and $(a_2, b_2) \in A'_{j'}$, $1 \le j' \le m'$, be the two arcs corresponding to x_i, and let $(c, d) \in A'_k$, $1 \le k \le m'$, be the arc corresponding to $\overline{x}_i$. We have six cases:

CASE 1. If $(a_1, b_1, a_1), (a_2, b_2, a_2), (c, d, c) \in S$, then $\langle V''_i \rangle$ itself is a strong graph. But (a_1, b_1, a_1) contributes one strong component to $\langle V_j \rangle$, (a_2, b_2, a_2) contributes one strong component to $\langle V'_{j'} \rangle$, and (c, d, c) contributes one strong component to $\langle V'_k \rangle$.

CASE 2. If $(a_1, b_1, a_1), (a_2, b_2, a_2), (c, d, c) \notin S$, then $\langle V''_i \rangle$ itself is a strong graph. But $\{a_1\}$ and $\{b_1\}$ contribute 2 strong components to $\langle V_j \rangle$, $\{a_2\}$ and $\{b_2\}$ contribute 2 strong components to $\langle V'_{j'} \rangle$, and $\{c\}$ and $\{d\}$ contribute 2 strong components to $\langle V'_k\rangle$.

CASE 3. If $(a_1,b_1,a_1),(c,d,c) \in S$ and $(a_2,b_2,a_2) \notin S$ (resp. (a_2,b_2,a_2), $(c,d,c) \in S$ and $(a_1,b_1,a_1) \notin S$), then $\langle V''_i\rangle$ contains 3 strong components. Similar to CASE 1 and 2, $(a_1,b_1,a_1),(c,d,c)$, $\{a_2\}$ and $\{b_2\}$ (resp. $(a_2,b_2,a_2),(c,d,c)$,$\{a_1\}$ and $\{b_1\}$) contribute 4 strong components to $\langle V_j\rangle$, $\langle V'_{j'}\rangle$ and $\langle V'_k\rangle$.

CASE 4. If $(a_2,b_2,a_2) \in S$ and $(a_1,b_1,a_1),(c,d,c) \notin S$ (resp. $(a_1,b_1,a_1) \in S$ and (a_2,b_2,a_2), $(c,d,c) \notin S$), then $\langle V''_i\rangle$ contains 1 strong components. Similar to CASE 1 and 2, (a_2,b_2,a_2), $\{a_1\}$,$\{b_1\}$,$\{c\}$ and $\{d\}$ (resp. (a_1,b_1,a_1),$\{a_2\}$, $\{b_2\}$,$\{c\}$ and $\{d\}$) contribute 5 strong components to $\langle V_j\rangle$, $\langle V'_{j'}\rangle$ and $\langle V'_k\rangle$.

CASE 5. If $(a_1,b_1,a_1),(a_2,b_2,a_2) \in S$ and $(c,d,c) \notin S$, then $\langle V''_i\rangle$ contains 4 strong components. In this case, $(a_1,b_1,a_1),(a_2,b_2,a_2)$,$\{c\}$ and $\{d\}$ contribute 4 strong components to $\langle V_j\rangle$, $\langle V'_{j'}\rangle$ and $\langle V'_k\rangle$.

CASE 6. If $(c,d,c) \in S$ and $(a_1,b_1,a_1),(a_2,b_2,a_2) \notin S$, then $\langle V''_i\rangle$ contains 5 strong components. In this case, (c,d,c),$\{a_1\}$,$\{b_1\}$,$\{a_2\}$ and $\{b_2\}$ contribute 5 strong components to $\langle V_j\rangle$, $\langle V'_{j'}\rangle$ and $\langle V'_k\rangle$.

We now use the following method to count the number of strong components in G^*: (1) Construct a new graph $G' = (\bigcup_{i=1}^{m}\langle V_i\rangle)\cup(\bigcup_{i=1}^{m'}\langle V'_i\rangle)$. Count the number of strong components in G' which is the sum of the strong components in $\langle V_i\rangle$ and $\langle V'_i\rangle$. (2) For each $\langle V''_i\rangle$, $1 \le i \le n'$, add it to G' and count the strong components in the current G'. For CASE 1, 2, 3 and 4, the current total number is reduced; and for CASE 5 and 6, the current total number dose not change. Since each $\langle V_i\rangle$ has at most 5 strong components and each $\langle V'_i\rangle$ has at most 3 strong components, the number of strong components in G^* is at most $5m + 3m'$. By condition (3), we know that G^* has at least $5m + 3m'$ strong components. Thus, each $\langle V_i\rangle$ has exactly 5 strong components, each $\langle V'_i\rangle$ has exactly 3 strong components, and each $\langle V''_i\rangle$ must satisfy CASE 5 or 6. Hence, each $\langle V_i\rangle$ contains exactly one 2-cycle. For each clause C_i, if the 2-cycle contained in $\langle V_i\rangle$ corresponds to variable x in C_i, then we assign *true* to x and assign *false* to the remaining two literals in C_i. Since each variable occurs exactly once in $C_1 \wedge C_2 \wedge \cdots \wedge C_m$, we know that each varible has a truth value and each clause C_i in ψ has exactly one true literal. Since each $\langle V'_i\rangle$ has exactly 3 strong components, we know that each clause C'_i in ψ has exactly one true literal. Since each $\langle V''_i\rangle$ satisfies CASE 5 or 6, each variable and its negation must been assigned distinct truth values. Therefore, there exists a satisfying truth assignment for ψ such that each clause in ψ has exactly one true literal. ■

4. S-MAX-SC problem is NP-complete

The decision version of S-MAX-SC problem can be described as follows.

Problem: Simplified MAX-SC problem (S-MAX-SC)
Instance: A digraph $G(V,A)$ and its complement $\overline{G}(V,\overline{A})$, a set L of pairs of arcs in $\overline{A}$, and a positive integer K.
Question: Is there a subset $S \subseteq \overline{A}$ such that for each element (i.e., pair of arcs) in L, at least one arc in the element belongs to S and the number of strong components in the digraph $G^*(V, A\cup S)$ is at least K?

In order to prove the NP-hardness of S-MAX-SC, we introduce another problem, called the *Acyclicity Maintenance problem.*

Problem: Acyclicity Maintenance (AM)
Instance: An acyclic digraph $G(V,A)$ and its complement $\overline{G}(V,\overline{A})$, a set L of pairs of arcs in $\overline{A}$.
Question: Is there a subset $S \subseteq \overline{A}$ such that for each element in L, at least one arc of the element belongs to S and the digraph $G^*(V, A\cup S)$ has no directed cycle?

We now show that AM is NP-complete by a reduction from 1-IN-3 SAT.

Theorem 4.1 AM *problem is NP-complete.*

PROOF. It is easy to see that AM belongs to NP. We will show that the AM problem is NP-hard by proving a reduction of 1-IN-3 SAT to AM. Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be an instance of 1-IN-3 SAT problem, i.e., a Boolean formula in conjunctive normal form in which each clause contains three positive literals. Let $C_i = x_{i1} \vee x_{i2} \vee x_{i3}$, $1 \le i \le m$.

We now construct an acyclic digraph $G = (V,A)$ and a set L of pairs of arcs (refer to Figure 5). We first create $V = \{v_1, v_2, \ldots, v_{6m}\}$ such that for each i $(1 \le i \le m)$ and j $(1 \le j \le 3)$, the arc $a_{ij} = (v_{6(i-1)+2j-1}, v_{6(i-1)+2j})$ corresponds to the literal x_{ij}. We define arc $\overline{a}_{ij} = (v_{6(i-1)+2j}, v_{6(i-1)+2j-1})$. Notice that arcs a_{ij} and $\overline{a}_{ij}$ are not in A. We then create

$$A = \{(v_{6i-4}, v_{6i-3}), (v_{6i-2}, v_{6i-1}), (v_{6i}, v_{6i-5}) : 1 \le i \le m\}.$$

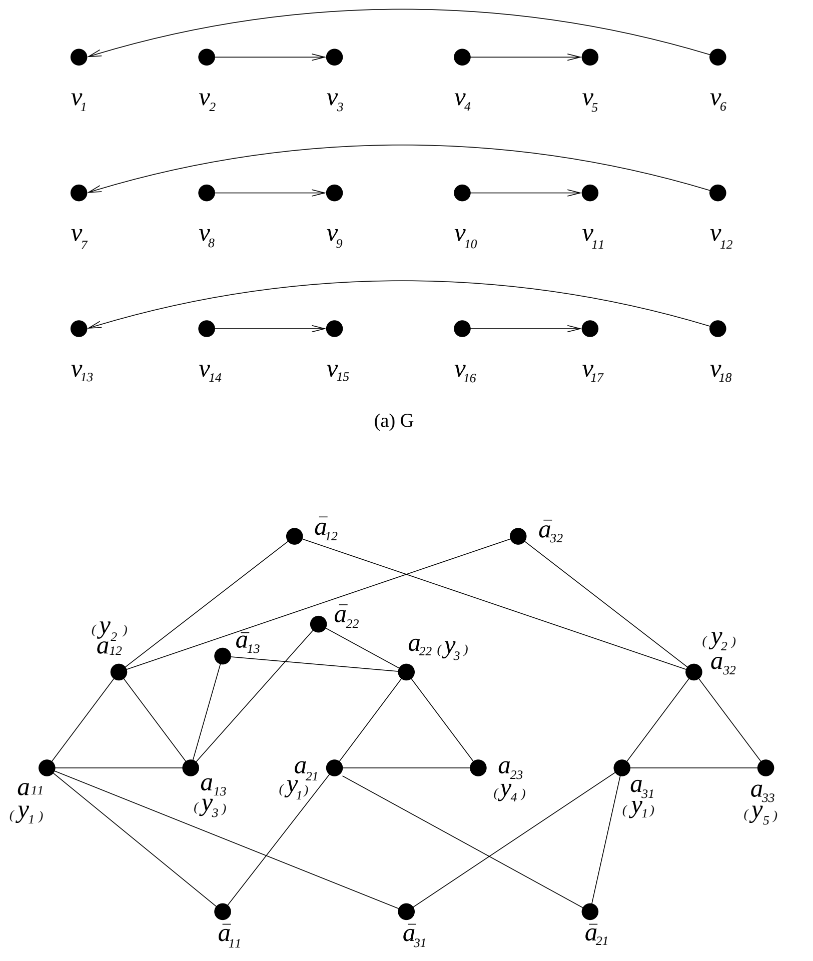
We next construct a set L of pairs of arcs. We first create $3m$ pairs of arcs for L as follows: For each clause $C_i = x_{i1} \vee x_{i2} \vee x_{i3}$, $1 \le i \le m$, we create a set A_i of 3 pairs of arcs

$$A_i = \{\{a_{i1}, a_{i2}\}, \{a_{i2}, a_{i3}\}, \{a_{i3}, a_{i1}\}\}.$$

Then, for each variable that appears more than once in ϕ, say k times, we create a set of $2k$ pairs of arcs for L. Let y be such a variable that appears $k(\geq 2)$ times such that $y = x_{i_1j_1} = x_{i_2j_2} = \cdots = x_{i_kj_k}$, where the subscripts $i_1j_1, i_2j_2, \ldots, i_kj_k$ form a lexicographic order. We create a set B_y of $2k$ pairs of arcs

$$\begin{aligned} B_y = \quad & \{\{a_{i_1j_1}, \quad \overline{a}_{i_1j_1}\}, \{\overline{a}_{i_1j_1}, a_{i_2j_2}\}, \{a_{i_2j_2}, \overline{a}_{i_2j_2}\}, \{\overline{a}_{i_2j_2}, a_{i_3j_3}\}, \ldots, \\ & \{a_{i_kj_k}, \quad \overline{a}_{i_kj_k}\}, \{\overline{a}_{i_kj_k}, a_{i_1j_1}\}\}. \end{aligned}$$

We now finish the construction of L which is the union of A_i $(1 \leq i \leq m)$ and B_y for each variable y appears more than once in ϕ. In order to show the relations between the elements in L, we introduce an undirected graph $\mathcal{L}$, called the graph of L, such that each arc in L corresponds to a vertex in $\mathcal{L}$, and each element (i.e., pair of arcs) in L corresponds to an edge in $\mathcal{L}$. Figure 5 illustrates the construction of G and L. It is easy to verify that every arc in L is not an arc in G. Note that we can construct G and L in polynomial time.



(a) G

(b) The graph of L

Figure 5: The graph G and $\mathcal{L}$ constructed from an instance of 1-IN-3 SAT $(y_1 \vee y_2 \vee y_3) \wedge (y_1 \vee y_3 \vee y_4) \wedge (y_1 \vee y_2 \vee y_5)$.

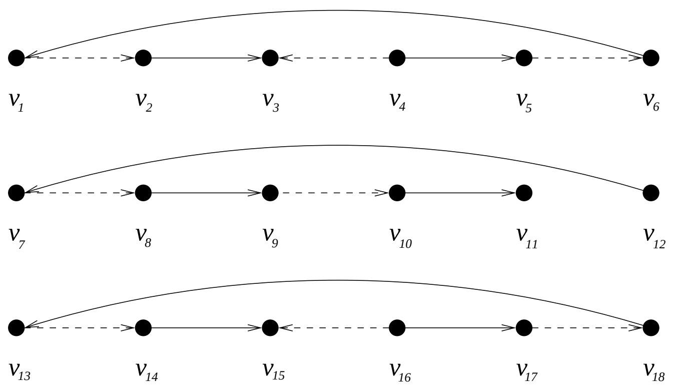
We claim that there exists a satisfying truth assignment for ϕ such that each clause in ϕ has exactly one true literal if and only if there exists a set S of arcs satisfying the following three conditions: (1) Each arc in S is contained in L; (2) for each pair of arcs in L, at least one arc belongs to S; and (3) the digraph $G^*(V, A \cup S)$ does not have any directed cycle. We first suppose that there exists a satisfying truth assignment for ϕ such that each clause in ϕ has exactly one true literal. For each clause C_i, $1 \leq i \leq m$, containing exactly two false literals, say, x_{ij} and $x_{ij'}$, $j \neq j'$, we put the corresponding two arcs a_{ij} and $a_{ij'}$ into S (see Figure 6). Since a_{ij} and $a_{ij'}$ are contained in A_i, condition (1) holds. Notice that each pair of arcs in A_i contains a_{ij} or $a_{ij'}$. For each variable y appearing $k \geq 2$ times in ϕ and its corresponding set B_y, if y is false, then S has contained k arcs $a_{i_1j_1}, a_{i_2j_2}, \ldots, a_{i_kj_k}$; otherwise, we put k arcs $\overline{a}_{i_1j_1}, \overline{a}_{i_2j_2}, \ldots, \overline{a}_{i_kj_k}$ into S. Hence each pair of arcs in B_y has at least one arc in S. After we consider all clauses C_i and variables y, we obtain a set S of arcs which satisfies conditions (1) and (2). We now consider the digraph $G^*(V, A \cup S)$. We can partition $V(G^*)$ into m subsets $V_i = \{v_{6i-5}, v_{6i-4}, \ldots, v_{6i}\}$, $1 \le i \le m$, such that V_i corresponds to C_i. It is easy to see that no arc in G^* links a vertex in V_i and to a vertex in V_j ($j \ne i$). Let $\langle V_i \rangle$ denote the subdigraph of G^* induced by V_i. For any $\langle V_i \rangle$, it is easy to check that this digraph has no directed cycle. For example, let x_{i1} be the true literal in C_i. If variable x_{i1} appears only once in ϕ, then $\langle V_i \rangle$ has the arc set

$$\{(v_{6i-4}, v_{6i-3}), (v_{6i-3}, v_{6i-2}), (v_{6i-2}, v_{6i-1}), (v_{6i-1}, v_{6i}), (v_{6i}, v_{6i-5})\}.$$

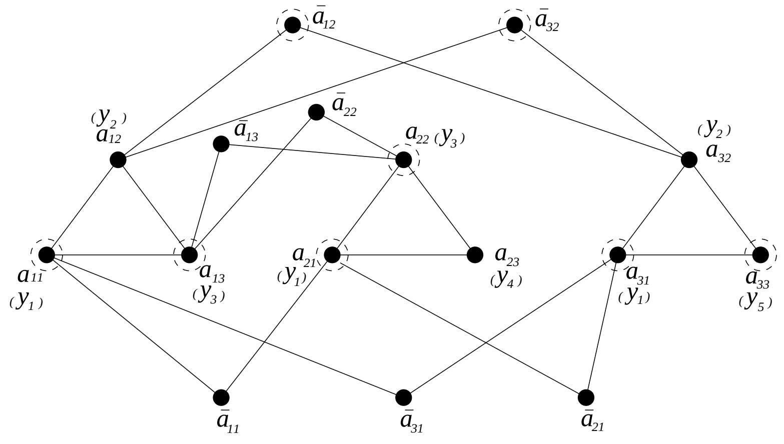
If variable x_{i1} appears more than once in ϕ, then $\langle V_i \rangle$ has the arc set

$$\{(v_{6i-4}, v_{6i-3}), (v_{6i-3}, v_{6i-2}), (v_{6i-2}, v_{6i-1}), (v_{6i-1}, v_{6i}), (v_{6i}, v_{6i-5}), (v_{6i-4}, v_{6i-5})\}.$$

Thus, the digraph G^* has no directed cycle. Hence, condition (3) holds.



(a) G*



(b) The graph of L

Figure 6: Suppose $y_2 = y_4 =$TRUE and $y_1 = y_3 = y_5 =$FALSE for formula $(y_1 \vee y_2 \vee y_3) \wedge (y_1 \vee y_3 \vee y_4) \wedge (y_1 \vee y_2 \vee y_5)$. The dashed arcs in (a) belong to S. The dashed vertices in (b), which correspond to the false literals, also correspond to the arcs in S.

Conversely, suppose that there exists a set S of arcs satisfying the above three conditions. If S contains three different arcs in A_i, then G^* has a directed cycle

$$\{(v_{6i-5}, v_{6i-4}), (v_{6i-4}, v_{6i-3}), (v_{6i-3}, v_{6i-2}), (v_{6i-2}, v_{6i-1}), (v_{6i-1}, v_{6i}), (v_{6i}, v_{6i-5})\}.$$

This contradicts condition (3). If S contains only one arcs in A_i, then there exist a pair of arcs in $A_i \subseteq L$ in which both arcs are not in S. This contradicts condition (2). Thus, S contains exactly two different arcs in A_i, $1 \le i \le m$. We can set FALSE to the two literals corresponding to these two arcs, and set TRUE to the remaining literal in C_i. We now show that any variable y that appears $k \geq 2$ times in ϕ has the same truth assignment. Let $y = x_{i_1 j_1} = x_{i_2 j_2} = \cdots = x_{i_k j_k}$, where the subscripts $i_1 j_1, i_2 j_2, \ldots, i_k j_k$ form a lexicographic order. Assume that not all these literals have the same truth assignment. There must exist two literals $x_{i_p j_p}$ and $x_{i_{p+1} j_{p+1}}$ such that $x_{i_p j_p}$ =FALSE and $x_{i_{p+1} j_{p+1}}$ =TRUE, where $p+1$ is considered as 1 if $p = k$. Thus, $a_{i_p j_p} \in S$ and $a_{i_{p+1} j_{p+1}} \notin S$. Since $(\overline{a}_{i_p j_p}, a_{i_{p+1} j_{p+1}})$ is a pair of arcs in $B_y \subset L$, it follows from condition (2) that $\overline{a}_{i_p j_p} \in S$. Since $a_{i_p j_p}$ and $\overline{a}_{i_p j_p}$ form a directed cycle, this contradicts condition (3). Hence, any variable that appears more than once in ϕ has the same truth assignment. Therefore, there exists a satisfying truth assignment for ϕ such that each clause in ϕ has exactly one true literal. ■

Now we can prove the main result of this section (the proof is omitted due to the space limit).

Theorem 4.2 S-MAX-SC *problem is NP-complete.*

PROOF. Since computing the strong components of a digraph can be performed in linear time [9], it is easy to see that S-MAX-SC belongs to NP.

We now show that the S-MAX-SC problem is NP-hard by proving a reduction of AM to S-MAX-SC. Given an instance of the AM problem, that is, an acyclic digraph G and a set L of pairs of arcs, we construct an instance of S-MAX-SC by setting $\tilde{G} = G$, $\tilde{L} = L$ and $K = |V|$. This construction needs linear time. Let $\tilde{S}$ be a set of arcs from $\tilde{L}$ such that for each pair of arcs in $\tilde{L}$, at least one arc belongs to $\tilde{S}$. Let $\tilde{G}^*$ be the digraph obtained by adding $\tilde{S}$ to $\tilde{G}$. We can set $S = \tilde{S}$ and $G^* = \tilde{G}^*$. It is easy to see that G^* is acyclic if and only if $\tilde{G}^*$ has at least K strong components. ■

5. Discussions

The typing rules listed in Figure 1 do not encompass all the DCC typing rules. Specifically, two features of DCC were not encoded, namely, *capability granting policies* (i.e., $(\mathcal{DCC}3)$, $(\mathcal{DCC}4)$, and $(\mathcal{DCC}6)$ (i)) and *hereditary mutual suspicion* (i.e., $(\mathcal{DCC}7)$). Fortunately, such omissions do not affect the generality of our results. We argue that there is a polynomial-time reduction that takes an instance of the MAX-SC problem, and produces an corresponding Java program not involving the enforcement of capability granting policies and hereditary mutual suspicion. Details of this reduction will be reported in future work.

Confined Types [10] is a type system that partly inspires DCC. It categorically enforces the confinement of package-private object references, rather than imposes a semipermeable confinement boundary as found in DCC. Consequently, its type reconstruction problem is tractable [4, 5].

Although the type reconstruction problem has been demonstrated to be NP-complete, the availability of a tool for inferring DCC annotations for legacy software is of high practical value. To this end, we plan to explore heuristic and/or approximation algorithms. We are currently experimenting with a branch-and-bound approach, and evaluating the applicability of various heuristics.

6. References

[1] Dennis, J. B. and van Horn, E. C., "Programming Semantics for Multiprogrammed Computations", Communications of the ACM, 9(3):143–155, 1966.

[2] Fong, P. W. L., "Discretionary Capability Confinement", Proc. of 11th Euro. Symp. on Research in Computer Security (ESORICS'06), Lecture Notes in Computer Science, 2189, 127–144, 2006.

[3] Fong, P. W. L., "Discretionary Capability Confinement", International J. of Information Security, In press, 2008.

[4] Grothoff, C., Palsberg, J. and Vitek, J., "Encapsulating Objects with Confined Types", Proc. of 16th ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications, 241–253, 2001.

[5] Grothoff, C., Palsberg, J. and Vitek, J., "Encapsulating Objects with Confined Types", ACM Transactions on Programming Languages and Systems, 29(6), 2007.

[6] Lindholm, T. and Yellin, F., The Java Virtual Machine Specification, Addison Wesley, 1999.

[7] Garey, M. and Johnson, D., Computers and Intractability, Freeman, San Francisco, 1979.

[8] Schaefer, T. J., "The complexity of satisfiability problems", Proc. of 10th annual ACM symp. on Theory of computing (STOC'78), 216–226, 1978.

[9] Tarjan, R. T., "Depth-First Search and Linear Graph Algorithms", SIAM Journal on Computing, 1:146–160, 1972.

[10] Vitek, J. and Bokowski, B., "Confined Types in Java", Software - Practice & Experience, 31(6):507–532, 2001.