別解問題の計算量と完全性、およびパズルへの応用

八登 崇之
東京大学大学院理学系研究科
情報科学専攻
yato@is.s.u-tokyo.ac.jp

瀬田 剛広
東京大学大学院情報理工学系研究科
コンピュータ科学専攻
seta@is.s.u-tokyo.ac.jp

概 要

問題 Π に対する別解問題 (ASP) というのは、Π のインスタンス x とそれに対する 1 つの解 s が与えられた時に、x の s 以外の解を求める問題のことである。(新しい問題クラスとしての ASP の概念は Ueda と Nagao による。) 本論文では n 個の解が与えられた時にもう 1 つの解を求める問題 (n-ASP) について考察する。特に、対応する解への変換も多項式時間で行えるような多項式時間 parsimonious 還元に関する完全性である ASP 完全性について考察する。

応用として、本論文では 3 つの有名なパズル、スリザーリンクとカックロ (クロスサム) とナンバープレース (数独) についてその ASP 完全性 (NP 完全性を含意する) を示す。

Complexity and Completeness of Finding Another Solution and Its Application to Puzzles

Takayuki YATO
Department of Information Science
Graduate School of Science
The University of Tokyo
yato@is.s.u-tokyo.ac.jp

Takahiro SETA
Department of Computer Science
Graduate School of Information Science and Technology
The University of Tokyo
seta@is.s.u-tokyo.ac.jp

Abstract

The Another Solution Problem (ASP) of a problem Π is the following problem: for a given instance x of Π and a solution s to it, find a solution to x other than s. (The notion of ASP as a new class of problems was first introduced by Ueda and Nagao.) In this paper we consider n-ASP, the problem to find another solution when n solutions are given. In particular we consider ASP-completeness, the completeness with respect to the parsimonious reductions which allow polynomial-time transformation of solutions.

As an application, we prove the ASP-completeness (which implies NP-completeness) of three popular puzzles: Slither Link, Cross Sum, and Number Place.

1 Introduction

There are some cases where we want to know whether a given problem has any solution other than a given one. In this paper we study the complexity of this sort of problem, called "Another Solution Problem (ASP)."

The notion of ASP as a new class of problems, with its name, was first introduced by Ueda and Nagao [8], although the ASP of a few individual problems had been studied before them, such as Hamiltonian circuit problems [5]. In addition they have pointed out that ASP has a close relation to designing puzzles. For many sorts of puzzles, the uniqueness of solution is desired, and thus puzzle designers have to check whether the designed problem has no solutions other than the intended one. This work is exactly an instance of ASPs. Besides puzzles ASP has many applications. For example, in the restoration of fossils and historical materials, verifying the uniqueness of solution is necessary to claim that the true figure is restored.

Note that ASP sometimes has a different complexity from that of the original problem. One interesting example is the Hamiltonian circuit problem for cubic graphs. Although the problem itself is NP-complete [4], its ASP is trivial because a cubic graph with a Hamiltonian circuit always has another (see [5]).

Ueda and Nagao [8] also pointed out that parsimonious reductions which allow transformation of solutions in polynomial time (what we call *ASP reduction* in this paper) can be used to derive the NP-completeness of ASP of a certain problem from that of ASP of another. Following this approach they proved the NP-completeness of ASP of Nonogram (a kind of puzzle) by showing such a reduction from 3-dimensional matching (3DM) to Nonogram.

In this paper we consider the problem to find another solution when n solutions are given (we call such problems *n-ASP*). Moreover we provide a formalization of n-ASP and investigate the characteristics of n-ASPs and ASP reductions. In particular we consider the completeness with respect to ASP reduction (we call it *ASP-completeness*). As is shown later, ASP-completeness implies NP-completeness of n-ASP for any n.

In addition we prove the ASP-completeness of three popular pencil puzzles: Slither Link, Cross Sum, and Number Place.[1] Since ASP-completeness implies NP-completeness these results also add new items to the list of NP-complete puzzles. (Note also that ASP-completeness implies #**P**-completeness of counting the solutions.)

2 Another Solution Problem (ASP)

2.1 Preliminaries

In this section we state a theory of Another Solution Problem (ASP). First of all we use the following existing formalization of function problems in order to facilitate the argument:

Definition 2.1 Let Π be a triple (D, S, σ) satisfying the following:

- D is the set of the instances of a problem.
- S is a set which includes all that can be a solution.[2]
- σ is a mapping from D to 2^S. For an instance $x \in D$, $\sigma(x)$ ($\subseteq S$) is called the *solution set* to x, and an element of $\sigma(x)$ is called an *solution* to x.

Then the problem which finds a solution to an instance $x \in D$ (or simply Π itself) is called a *function problem.*

Under this formalization, the class **FNP** is described as follows:

Definition 2.2 **FNP** is a class consisting of function problems $\Pi = (D, S, \sigma)$ such that the following holds:

- There exists a polynomial p such that $|s| \leq p(|x|)$ holds for any $x \in D$ and any $s \in \sigma(x)$.
- For any $x \in D$ and $y \in S$, the proposition $y \in \sigma(x)$ can be decided in polynomial time.

[1]Pencil puzzles are those offered as some figure on the paper and solved by drawing on the figure with a pencil. Nonogram is a pencil puzzle. Although many results are known about the computational complexity of combinatorial games and puzzles (see the survey by Demaine [3]), few are about pencil puzzles.

[2]S may include extra things. When we think on the basis of Turing machines, S may be the set of all the strings.

Definition 2.3 Let $\Pi = (D, S, \sigma)$ be a function problem. $Y = \{x \in D \mid \sigma(x) \neq \emptyset\}$ $\Pi_{\mathrm{d}} = (D, Y)$ (that is, the pair consist of the set of all the instances and the set of all the yes-instances) is called the *decision problem induced by* Π or simply the decision problem of Π.

Note that it follows from the characteristic of **NP** that for any decision problem $\hat{\Pi} \in$ **NP** there exists a function problem $\Pi \in$ **FNP** satisfying $\Pi_{\mathrm{d}} = \hat{\Pi}$. (Of course, $\Pi \in$ **FNP** implies $\Pi_{\mathrm{d}} \in$ **NP**.)

2.2 Formalization of ASP

Now we are ready to define ASPs.

Definition 2.4 Let $\Pi = (D, S, \sigma)$ be a function problem. For Π and a nonnegative integer n, we call the function problem $\Pi_{[n]} = (D_{[n]}, S, \sigma_{[n]})$ constructed as follows the *n-another solution problem* of Π (or shortly n-ASP).

$$D_{[n]} = \{(x, S_x) \mid S_x \subseteq \sigma(x),\ |S_x| = n\},$$
$$\sigma_{[n]}(x, S_x) = \sigma(x) - S_x$$

We call the decision problem of an n-another solution problem *n-another solution decision problem.*

We use the notation Π_{d} and $\Pi_{[n]}$ as the meaning mentioned above throughout this paper.

The next definition is polynomial-time ASP reduction. This is parsimonious reduction which allows polynomial-time transformation of solutions and is introduced by Ueda and Nagao [8]. (Recall that parsimonious reductions are used for proving #**P**-completeness of counting problems.)

Definition 2.5 Let $\Pi_1 = (D_1, S_1, \sigma_1)$ and $\Pi_2 = (D_2, S_2, \sigma_2)$ be function problems. We call the pair $\varphi = (\varphi_{\mathrm{D}}, \varphi_{\mathrm{S}})$ satisfying the following *polynomial-time ASP reduction* from Π_1 to Π_2:

- φ_{D} is a polynomial-time computable mapping from D_1 to D_2.
- For any $x \in D_1$, φ_{S} is a polynomial-time computable *bijection* from $\sigma_1(x)$ to $\sigma_2(\varphi_{\mathrm{D}}(x))$.

If there is a polynomial-time ASP reduction from Π_1 to Π_2, Π_1 is called to be *polynomial-time ASP reducible* to Π_2 (denoted by $\Pi_1 \preceq_{\mathrm{ASP}} \Pi_2$).

By definition, all the ASP reductions are parsimonious. Although the converse does not always hold, many parsimonious reductions involve concrete transformation of solutions and thus are ASP-reductions.

The relation $\preceq_{\mathrm{ASP}}$ is transitive, like other reducibility relations. If $\Pi_1 \preceq_{\mathrm{ASP}} \Pi_2$, then $\Pi_{1\mathrm{d}}$ is polynomial-time reducible (as a decision problem) to $\Pi_{2\mathrm{d}}$.

The relation of ASP reducibility is invariant with respect to "taking an ASP" operation.

Proposition 2.1 *Let Π_1 and Π_2 be function problems. If $\Pi_1 \preceq_{\mathrm{ASP}} \Pi_2$, then $\Pi_{1[n]} \preceq_{\mathrm{ASP}} \Pi_{2[n]}$ for any nonnegative integer n.*

Proof Immediate from the definition. ■

The next proposition shows that n-ASP of m-ASP is $(m+n)$-ASP (of the original problem).

Proposition 2.2 *For any function problem Π and nonnegative integers m, n, $(\Pi_{[m]})_{[n]} \preceq_{\mathrm{ASP}} \Pi_{[m+n]}$.*

Proof Let $\Pi = (D, S, \sigma)$. An instance $\tilde{x}$ of $(\Pi_{[m]})_{[n]}$ is of the form

$$((x, \{s_1, \ldots, s_m\}), \{t_1, \ldots, t_n\})$$
$$(x \in D;\ s_1, \ldots, s_m, t_1, \ldots, t_n \in \sigma(x)).$$

For this we set $\varphi_{\mathrm{D}}(\tilde{x})$ to be $(x, \{s_1, \ldots, s_m, t_1, \ldots, t_n\})$. Then the solution set of $\tilde{x}$ is equal to that of $\varphi_{\mathrm{D}}(\tilde{x})$, and thus φ_{S} can be set to be the identity function. ■

Combining the two propositions, we obtain the following important result.

Theorem 2.3 *Let Π be a function problem. If $\Pi \preceq_{\mathrm{ASP}} \Pi_{[1]}$, then for any nonnegative integer n $\Pi_{[n]} \preceq_{\mathrm{ASP}} \Pi_{[n+1]}$. (Therefore $\Pi \preceq_{\mathrm{ASP}} \Pi_{[n]}$.)*

Proof From $\Pi \preceq_{\mathrm{ASP}} \Pi_{[1]}$ and Proposition 2.1 we obtain $\Pi_{[n]} \preceq_{\mathrm{ASP}} (\Pi_{[1]})_{[n]}$, and from Proposition 2.2 we obtain $(\Pi_{[1]})_{[n]} \preceq_{\mathrm{ASP}} \Pi_{[n+1]}$. Thus the theorem holds because of the transitivity of $\preceq_{\mathrm{ASP}}$. ■

3 ASP-completeness

ASP-completeness is completeness with respect to ASP reductions, and defined as follows:

Definition 3.1 A function problem Π is *ASP-complete* if and only if $\Pi \in \mathbf{FNP}$, and $\Pi' \preceq_{\mathrm{ASP}} \Pi$ for any $\Pi' \in \mathbf{FNP}$.

Proposition 3.1 *Let Π and Π' be function problems. If Π is ASP-complete, $\Pi' \in$ **FNP** and $\Pi \preceq_{\mathrm{ASP}} \Pi'$, then Π' is ASP-complete.*

Our first ASP-complete problem is SAT. Here SAT represents the function problem of satisfiability. (The decision problem is denoted by $\mathrm{SAT}_{\mathrm{d}}$.)

Theorem 3.2 SAT *is ASP-complete.*

Proof Cook's reduction is an ASP reduction. ■

An important property of ASP-completeness is that it implies the **NP**-completeness of n-ASPs for any n. We first show this property as to SAT.

Theorem 3.3 *1.* $\mathrm{SAT} \preceq_{\mathrm{ASP}} \mathrm{SAT}_{[1]}$.

2. For any nonnegative integer n, $\mathrm{SAT}_{[n]\mathrm{d}}$ is **NP**-*complete.*

Proof 1. We construct a polynomial-time ASP reduction $\varphi = (\varphi_{\mathrm{D}}, \varphi_{\mathrm{S}})$ from SAT to $\mathrm{SAT}_{[1]}$.

For a given instance of SAT ψ (a CNF formula), we construct a CNF formula ψ' as follows:

- We introduce a new variable w, and for each clause $l_1 \vee \cdots \vee l_r$ in ψ make a clause $l_1 \vee \cdots \vee l_r \vee w$ and add it to ψ'.
- For each variable x we add a clause $x \vee \bar{w}$ (CNF of $w \Rightarrow x_i$) to ψ'.

Then define $\varphi_{\mathrm{D}}(\psi)$ to be $(\psi', \{g\})$, where g is the assignment in which all variables are true. For a solution h to ψ, let $\varphi_{\mathrm{S}}(h)$ be h with assignment $w =$ false added.

2. From 1 and Theorem 2.3, we obtain $\mathrm{SAT} \preceq_{\mathrm{ASP}} \mathrm{SAT}_{[n]}$ for any nonnegative integer n. Thus $\mathrm{SAT}_{\mathrm{d}}$ is polynomial-time reducible to $\mathrm{SAT}_{[n]\mathrm{d}}$ and the theorem holds. ■

Using this result and Proposition 2.1 the property is shown for all ASP-complete problems.

Theorem 3.4 *For any ASP-complete function problem Π and any nonnegative integer n, $\Pi_{[n]\mathrm{d}}$ is* **NP**-*complete.*

Proof Since Π is ASP-complete, $\mathrm{SAT} \preceq_{\mathrm{ASP}} \Pi$ holds, thus it follows from Proposition 2.1 that $\mathrm{SAT}_{[n]} \preceq_{\mathrm{ASP}} \Pi_{[n]}$. This implies that $\mathrm{SAT}_{[n]\mathrm{d}} \preceq \Pi_{[n]\mathrm{d}}$. Thus $\Pi_{[n]\mathrm{d}}$ is shown to be **NP**-complete from Theorem 3.3. ■

Here follow the proofs of ASP-completeness of some logic problems, which are used in the later sections.

Theorem 3.5 3SAT *is ASP-complete.*

Proof We show a polynomial-time ASP reduction from SAT. Let ψ be a CNF formula given as an instance of SAT. We modify ψ as follows:

- Introduce new variables t_1, t_2, t_3 and add these 7 clauses: $t_1 \vee t_2 \vee t_3$, $t_1 \vee t_2 \vee \bar{t}_3$, $t_1 \vee \bar{t}_2 \vee t_3$, $t_1 \vee \bar{t}_2 \vee \bar{t}_3$, $\bar{t}_1 \vee t_2 \vee t_3$, $\bar{t}_1 \vee t_2 \vee \bar{t}_3$, $\bar{t}_1 \vee \bar{t}_2 \vee t_3$. (All of t_1, t_2, t_3 must be true.)
- Add $\bar{t}_1$ to each clause with two literals, and add $\bar{t}_1$ and $\bar{t}_2$ to each clause with one literal,
- Divide each clause with 4 or more literals $l_1 \vee l_2 \vee l_3 \vee \cdots \vee l_r$ as follows: $l_1 \vee l_2 \vee \bar{d}$, $\bar{l}_1 \vee d$, $\bar{l}_2 \vee d$ (these are CNF of $(l_1 \vee l_2) \equiv d$), $d \vee l_3 \vee \cdots \vee l_r$, where d is a new variable. While the last clause has 4 or more literals, repeat the division.

Let ψ' denote the modified formula. Then ψ' is satisfiable if and only if ψ is satisfiable. Moreover, for each truth assignment h satisfying ψ, there is only one assignment that can be obtained by extending h and satisfy ψ'. ■

1-in-3 SAT is the following problem: given a 3-CNF formula, find a truth assignment in which each clause contains exactly one true literal.

Theorem 3.6 *1-in-3 SAT is ASP-complete.*

Proof To prove the ASP-completeness of 1-in-3 SAT, we first show a polynomial-time ASP reduction from 3SAT to Just-one SAT. Here Just-one SAT means the same problem as 1-in-3 SAT except that a CNF formula is given instead of 3-CNF formula.

Let ψ be a 3-CNF formula with n variables $v_1, v_2, \ldots, v_n$ given as an instance of 3SAT.

Then we construct a CNF ψ' which is an instance of Just-one SAT as follows:

- Introduce new variables $v_{i,j,k,l}$ for all $1 \le i < j < k \le n$, $0 \le l \le 7$. (l represents the assignments of v_i, v_j, v_k in binary.)
- For all $i < j < k$, add following clauses to ψ':
  $v_{i,j,k,0} \vee v_{i,j,k,1} \vee v_{i,j,k,2} \vee v_{i,j,k,3} \vee v_i$,
  $v_{i,j,k,4} \vee v_{i,j,k,5} \vee v_{i,j,k,6} \vee v_{i,j,k,7} \vee \bar{v}_i$,
  $v_{i,j,k,0} \vee v_{i,j,k,1} \vee v_{i,j,k,4} \vee v_{i,j,k,5} \vee v_j$,
  $v_{i,j,k,2} \vee v_{i,j,k,3} \vee v_{i,j,k,6} \vee v_{i,j,k,7} \vee \bar{v}_j$,
  $v_{i,j,k,0} \vee v_{i,j,k,2} \vee v_{i,j,k,4} \vee v_{i,j,k,6} \vee v_k$,
  $v_{i,j,k,1} \vee v_{i,j,k,3} \vee v_{i,j,k,5} \vee v_{i,j,k,7} \vee \bar{v}_k$.
  (Make the assignment to v_i, v_j, v_k determine that of $v_{i,j,k,l}$.)
- For each clause in ψ, add to ψ' the clause which rejects unsatisfying assignment to ψ. For example, for $v_i \vee \bar{v}_j \vee \bar{v}_k$ in ψ ($v_i = 0, v_j = 1, v_k = 1$ is the unsatisfying assignment) add $\bar{v}_{i,j,k,3}$ to ψ'.

It is easily confirmed that this is an ASP reduction.

Next, we show a polynomial-time ASP reduction from Just-one SAT to 1-in-3 SAT and finish the proof. The reduction is just like that from SAT to 3SAT. Let ψ be a CNF formula given as an instance of Just-one SAT. Then, we modify ψ to ψ' as follows:

- Introduce new variables t_1, f_1, f_2, f_3 and add these 3 clauses: $t \vee f_1 \vee f_2$, $t \vee f_2 \vee f_3$, $t \vee f_3 \vee f_1$ (t must be true, and all of f_1, f_2, f_3 must be false.)
- Add f_1 to each clause with two literals, and add f_1 and f_2 to each clause with one literal,
- Divide each clause with 4 or more literals $l_1 \vee l_2 \vee l_3 \vee \cdots \vee l_r$ as follows: $l_1 \vee l_2 \vee \bar{d}$, $d \vee l_3 \vee \cdots \vee l_r$, where d is a new variable. While the last clause has 4 or more literals, repeat the division.

Again it is easily confirmed that this is an ASP reduction. ■

4 Some ASP-complete Puzzles

Here we prove the ASP-completeness of three modern pencil puzzles widely played around the world.

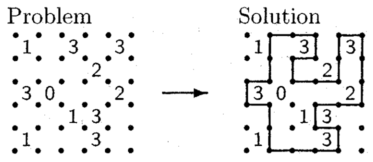

Figure 1: A problem of Slither Link.

4.1 Slither Link

The rule of Slither Link is as follows:

- Each problem is given as a rectangular lattice. The length of sides of the rectangle (as to the unit length of lattice) is called the *size* of the problem.
- A 1×1 square surrounded by four points is called a *cell*. A cell may have a number out of 0, 1, 2 or 3.
- The goal is to make a loop which does not intersect or branch by connecting adjacent dots with lines, so that a number on the cell is equal to the number of lines drawn around it.

An example of problems of Slither Link is shown in Fig. 1.

Here we prove the ASP-completeness of Slither Link. To construct an ASP reduction, we firstly prove the following lemma.

Lemma 4.1 *To find a Hamiltonian circuit for a given planar graph with degree at most 3 is ASP-complete.*

Proof See [7]. ■

We also use the following known fact (see [1]):

Lemma 4.2 *Any planar graph with degree at most 3 with n vertices can be embedded in an $\mathcal{O}(n) \times \mathcal{O}(n)$ grid in polynomial time in n.*

Now we are ready to state the proof.

Theorem 4.3 *To find a solution to a given instance of Slither Link is ASP-complete.*

Proof The membership in **FNP** is immediate. We construct a polynomial-time ASP reduction from the restricted Hamiltonian circuit problem to the Slither Link problem.

Using Lemma 4.2, we can transform a given instance (graph) G of the restricted Hamiltonian circuit problem into a graph G' on the grid. Note that G' has lattice points which do not correspond to any vertex of G and thus need not be visited when considering Hamiltonian circuits of G'. The degree of such points is two.

First, the gadget for a lattice point which need not be visited is the 6×6 board A shown in Fig. 2. Here we assume that no lines are drawn around this gadget. (This assumption will be satisfied later.) Then there are only four points, n, s, w

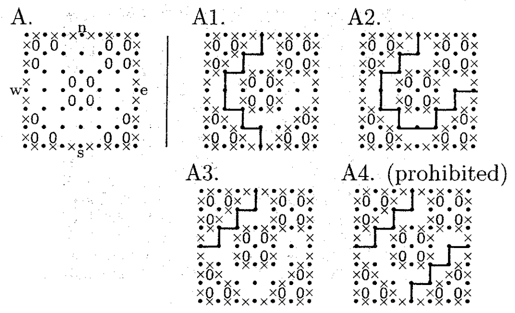

Figure 2: The gadget for a point in G' which does not correspond to a vertex of G, and local solutions to it.

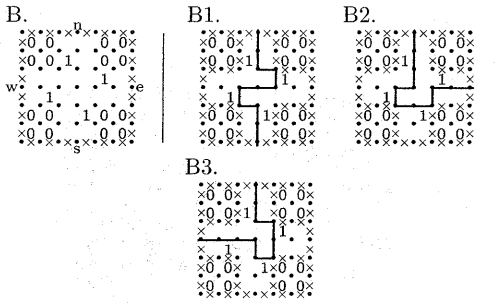

Figure 3: The gadget for a vertex of G' which corresponds to a vertex of G, and local solutions to it.

and e, through which a line goes out of the gadget. Thus, the local solution is either (i) that no lines are drawn, or (ii) that a path connecting two points out of n, s, w and e is drawn as shown in A1, A2 and A3. (In fact two paths can be drawn as in A4. But this case will be prohibited later.) Note that for any two points there is a unique way to connect them.

Next, the gadget for a lattice point which must be visited is the board B shown in Fig. 3. Again we assume no lines around the gadget. This time a path connecting two points on the boundary *must* be drawn, as in B1, B2 and B3, because this gadget has some 1's inside. Note that for any two points there is a unique way to connect them.

Last, we arrange these two sorts of gadgets in accordance with the grid graph G': we join two gadgets as in C of Fig. 4 where G' has a corresponding edge, and as in D where not. Only D allows a path connecting two gadgets. Moreover, both arrangements forbid lines on the boundary of the gadgets. Because all vertices of G' have a degree at most three, not all of n, s, w and e of a gadget A can have a passing line. This restriction prevents two paths passing through a gadget A.

In this way, we obtain the problem of Slither Link corresponding to G' (that is, G), (Drawing lines on the boundary of the entire board is also

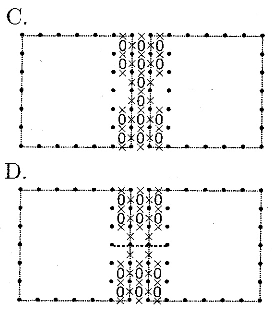

Figure 4: How to join gadgets.

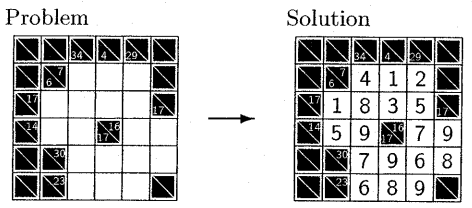

Figure 5: A problem of Cross Sum. The number in black cells represents the sum of integers in the string of cells below (or to the right of) it.

forbidden.) This transformation can be done in polynomial time in the input size, and the solution of Slither Link corresponding to a Hamiltonian circuit of G is unique and also computable in polynomial time. Thus a polynomial time ASP reduction from the restricted Hamiltonian circuit problem to Slither Link is constructed. ■

4.2 Cross Sum

The rule of Cross Sum is as follows:

- A problem is given as a rectangular grid with white (blank) and black cells. The length of sides of the rectangle is called the *size* of the problem. (Usually all the white cells are connected.)
- For each string of two or more (horizontally or vertically) consecutive cells, a *sum* is specified.
- The goal is to fill each blank cell with a integer from 1 through 9 so that for each string of cells (i) the sum of integers filling the cells is equal to the specified value and (ii) no integer occurs more than once.

An example of Cross Sum problems is shown in Fig. 5.

We construct an ASP reduction from the problem of partition into Hamiltonian subgraphs for planar mixed graphs with degree at most 3. (*Mixed* graphs are graphs containing both undirected and directed edges.) As shown in the appendix, this problem is ASP-complete.

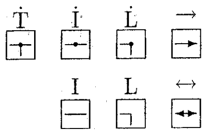

Figure 6: Gadgets needed for the reduction and their illustration.

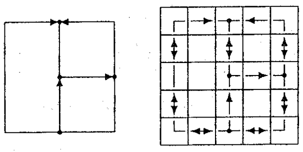

Figure 7: Overall construction of the reduction.

Lemma 4.4 *To find a partition into Hamiltonian subgraphs (and show a Hamiltonian circuit for each subgraph) for a given planar mixed graph with degree at most 3 is ASP-complete.*

Proof See [7] or [6]. ■

The ASP-completeness of Cross Sum results from this lemma.

Theorem 4.5 *To find a solution to a given instance of Cross Sum is ASP-complete.*

Proof The membership in **FNP** is immediate. We construct an ASP reduction from the problem of partition into Hamiltonian subgraphs for planar mixed graphs with degree at most 3.

First, using Lemma 4.2 we embed a given instance (graph) G of this partition problem into a grid of size polynomial in the number of the vertices of G.

Next, we make gadgets for lattice points and edges. There are three kinds of points: a T-intersection (T), a corner (L), and in a straight way (I). They each have two categories: a point without corresponding vertex in G (denoted by a letter shown above) and a point corresponding to a vertex in G (denoted by a dotted letter), except that T-intersection has $\dot{\mathrm{T}}$ only. The edge has two categories: directed ($\rightarrow$) and undirected ($\leftrightarrow$). (See Fig. 6) We will realize gadgets for each category as a 19×19 board later. Once these gadgets are realized, we can perform the whole reduction as shown in Fig. 7.

Next we relate choice of edges in the graph and assignment of numbers to the gadgets as follows:

- In point gadgets, 1 and 3 at a joint cell (a white cell on the boundary) mean that the corresponding edge is chosen, and 2 means not.
- Strings involving joint cells have the specified sum 10. As a result, edge gadgets have 9 or 7 at joint cells if it is chosen, and 8 if not.

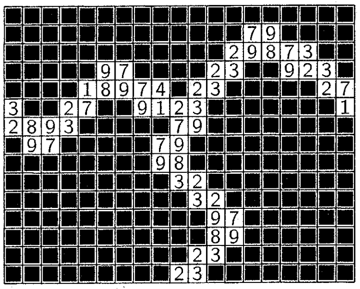

Figure 8: The gadget $\dot{\mathrm{T}}$ (lower part). The numbers in the white cells describe one of the local solutions, and actually these cells are all blank. Although the specified sums were not shown, they can be guessed easily. This local solution represents a path going from left to right. (Downward edge is not chosen. See the digits 1, 2 and 3 at the joint cells. Rearranging 1, 2 and 3 in the center gives other solutions.

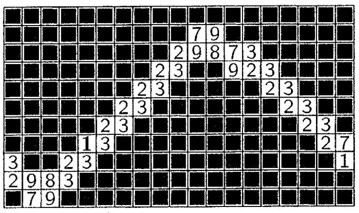

Figure 9: The gadget $\dot{\mathrm{I}}$ (upper part). The local solution in this figure represents a path going from left to right. The specified sum 4 of the two strings involving the 1 written in bold face forces a cycle to pass this gadget, since using $4 = 2 + 2$ is forbidden. By changing this 1 to 2 (and the sum to 5) we obtain the gadget I.

- As to horizontal direction, a cycle is oriented from 3 to 1 (7 to 9). As to vertical direction a cycle is oriented from 1 to 3 (9 to 7).

Figures 8–11 show the realization of each gadget. Note that all the gadgets have the size 19×19, although in the figures some parts consisting of only black cells are omitted.

Now the polynomial-time ASP reduction from the restricted partition into Hamiltonian subgraphs is completed and therefore the ASP-completeness of Cross Sum is proven. ■

The standard CROSS SUM rule allows to use numbers from 1 through 9. However it can be proven that CROSS SUM where numbers are limited to 7 is still ASP-complete. (See [7] for detail.) Whether changing the upper limit to less than 7 preserves ASP-completeness is not known.

4.3 Number Place

The rule of Number Place (also known as *Sudoku* in Japan) is as follows:

- A problem is given as a $n^2 \times n^2$ grid, which is divided into $n \times n$ squares with thick border lines. The value n is called *order*.

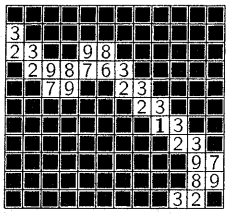
Figure 10: The gadget Ľ (lower-left part). The path goes from left (3 of horizontal position) to down (3 of vertical position). By changing the bold-face 1 to 2 (and incrementing the corresponding sums) we obtain the gadget L.

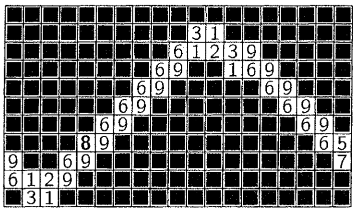
Figure 11: The gadget → (upper part). The digits 9 and 7 at the joint cells indicate that this edge is chosen. Swapping these two digits gives no local solution because of the 8 in bold face. By changing this 8 to 6 we obtain the gadget ↔.

- Some cells are filled with an integer from 1 through n^2.
- The goal is to fill in all the blank cells so that each row, column and $n \times n$ square has each of integers from 1 through n^2 exactly once.

An example of Number Place problems is shown in Fig. 12.

To show the ASP-completeness of this puzzle, we use the result about Latin squares. A *Latin square* of order n is a matrix such that each row and column contains each of integers from 1 through n exactly once. (Similar to Number Place, but lacking of the "small square" condition.) A *partial Latin square* is a matrix with some blank entries such that each row and column contains each of integers from 1 through n at most once. The problem of *partial Latin square completion* is as follows: Given a partial Latin square, make a Latin square by filling in the blanks. The ASP-completeness of this problem is immediate from known results.

Theorem 4.6 *The problem of partial Latin square completion is ASP-complete.*

Proof Colbourn [2] has proved the NP-completeness of ASP of partial Latin square completion by showing a reduction from 1-in-4 SAT to this problem. The reduction he used is what we call ASP reduction here. The ASP-completeness of 1-in-4 SAT is proven in an analogous way to that of 1-in-3 SAT. ∎

More we show the next lemma which shows a relation between the two problems. In the argument

Problem

			4
3		2	
	1		3
4			

$\longrightarrow$

Solution

1	2	3	4
3	4	2	1
2	1	4	3
4	3	1	2

Figure 12: A problem of Number Place

A0	01	02	**B0**	11	12	**C0**	21	22
D0	11	12	**E0**	21	22	**F0**	01	02
G0	21	22	**H0**	01	02	**I0**	11	12
01	02	10	11	12	20	21	22	00
11	12	20	21	22	00	01	02	10
21	22	00	01	02	10	11	12	20
02	10	11	12	20	21	22	00	01
12	20	21	22	00	01	02	10	11
22	00	01	02	10	11	12	20	21

A	B	C
D	E	F
G	H	I

Figure 13: Relation between Number Place and Latin square (in the case $n = 3$): Integers in Number Place are represented in base n. Although the cells with A0, ..., I0 are in actual problems blank, the lower digit of the numbers filling these cells must be 0 from the rule of Number Place. Moreover, the square on the right forms a Latin square.

below, we use integers ranged from 0 (instead of 1) as row and column numbers and contents of cells, and write $S(i,j)$ for the entry at position (i,j) of a square S ($\perp$ means a blank).

Lemma 4.7 *Let S be a Number Place problem of order n such that*

$$S(i,j) = \begin{cases} \perp & \textit{(when } (i,j) \in B\textit{)} \\ ((i \bmod n)n + \lfloor i/n \rfloor + j) \bmod n^2 & \\ & \textit{(otherwise)} \end{cases}$$

where $B = \{(i,j) \mid \lfloor i/n \rfloor = 0$ and $(j \bmod n) = 0\}$. Then a square S' obtained by filling in the blanks of S is a solution to S if and only if

- *For any $(i,j) \in B$, $S'(i,j) \bmod n$ equals 0.*
- *A square L defined by $L(i,j/n) = S(i,j)/n$ for all $(i,j) \in B$ is a Latin square.*

(as shown in Fig. 13)

Proof Derived from the rule of Number Place. ∎

Now we are ready to state the proof.

Theorem 4.8 *To find a solution to a given instance of Number Place is ASP-complete.*

Proof The membership in **FNP** is immediate. We show a polynomial time ASP reduction from the problem of partial Latin square completion to Number Place.

For a given partial Latin square L of order n, we construct a Number Place problem S as follows:

$$S(i,j) = \begin{cases} L(i,j/n) \cdot n & ((i,j) \in B,\ L(i,j/n) \neq \perp) \\ \perp & ((i,j) \in B,\ L(i,j/n) = \perp) \\ ((i \bmod n)n + \lfloor i/n \rfloor + j) \bmod n^2 & \\ & \text{(otherwise)} \end{cases}$$

This construction can be done in polynomial time in the input size. Moreover, from Lemma 4.7, each solution to L has a corresponding solution to S, which is also polynomial-time computable. Thus the desired polynomial-time ASP reduction is obtained. ■

5 Concluding Remarks

We formalized n-ASP, the problem to find another solution when n solutions are given, to facilitate investigations on its complexity. In particular, we introduced ASP-completeness, the completeness with respect to ASP reductions, and proved that ASP-completeness implies the **NP**-completeness of n-ASP for any nonnegative integer n.

It should be noted that in order to show the **NP**-completeness of n-ASP for all n showing ASP-completeness is not a unique way. Let Π be a function problem satisfying the following: (i) Π_{d} is **NP**-complete, and (ii) $\Pi \preceq_{\mathrm{ASP}} \Pi_{[1]}$. Then from Theorem 2.3 $\Pi \preceq_{\mathrm{ASP}} \Pi_{[n]}$ holds for all n. It implies the **NP**-completeness of $\Pi_{[n]\mathrm{d}}$ follows, but the ASP-completeness of Π is not shown directly. Nevertheless we conjecture that such Π is ASP-complete.

Conjecture 5.1 *Let Π be a function problem such that $\Pi_{[n]\mathrm{d}}$ is* **NP***-complete for any nonnegative integer n. Then Π is ASP-complete. (The converse of Theorem 3.4.)*

Moreover, as an application, we proved the ASP-completeness of three popular puzzles. We hope that more ASP-completeness results will appear.

References

[1] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235–284, 1994.

[2] C. J. Colbourn, M. J. Colbourn, and D. R. Stinson. The computational complexity of recognizing critical sets. In *Graph theory, Singapore 1983*, number 1073 in Lecture Notes in Math., pages 248–253. Springer, 1984.

[3] E. D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. Computing Research Repository, arXiv:cs.CC/0106009, 2002. (available at http://arXiv.org/).

[4] M. R. Garey, D. S. Johnson, and R. Endre Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Computing*, 5(4), 1976.

[5] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[6] T. Seta. The complexities of CROSS SUM. *IPSJ SIG Notes AL-84*, 2002.

[7] T. Seta. The complexities of puzzles, CROSS SUM and their another solution problems (ASP). Senior Thesis, Department of Infomation Science, the Faculty of Science, the University of Tokyo, 2002.

[8] N. Ueda and T. Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, 1996.