Mathematical Insights in Computing

Day 16: NP-Completeness Reductions

Problem 1. 3-Coloring is NP-Complete

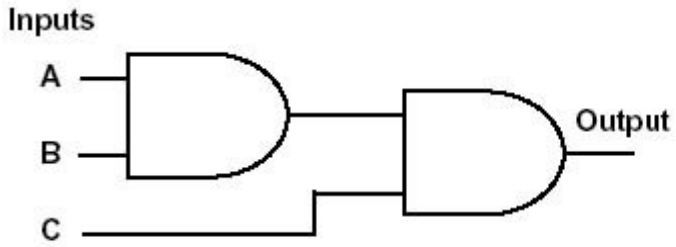
Having invented a time machine, you are walking the streets of an outdoor marketplace... *in the future*. It's a kind of bizarre place, and there are bizarre things for sale.

"Graph 3-coloring service! Only $5 per graph!" proclaims one salesrobot. "I'll color any 3-colorable graph as fast as I can! Polynomial-time guaranteed, or your money back!"

As far as you know, the P versus NP problem is *still* unsolved in the future, so naturally you are skeptical of this robot's claims. Let's try to trick this robot by creating some difficult 3-coloring graphs, shall we?

You remember that boolean circuit satisfiability, or CIRCUITSAT, is reducible to the 3-coloring problem, or 3COLORING. That is, CIRCUITSAT ≤ 3COLORING.

Let's say you want to devise a 3-coloring problem which will mimic the following simple circuit made from two AND gates:



A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

a.) Which values of A, B, and C will **satisfy** this circuit? That is, which inputs will cause the output to be 1 (True)?

Refer to the chalkboard lecture to see what the 3-coloring widgets are.

b.) Using these widgets, draw a graph to represent the boolean circuit from (a).

- The graph should have three nodes labelled **A**, **B**, and **C**, and a node labelled **output.**
- The graph might have other nodes in addition to those four.
- The graph should be 3-colorable *if and only if* the circuit is satisfiable.
- When the graph is successfully 3-colored, you should be able to read out the values of A, B, and C from the colors of their nodes.
- The three colors are “True”, “False”, and “Neither”.

Draw the graph below. Don't color it, just write nodes and edges.

You hand your graph to the robot and pay $5.

c.) Pretend you are the robot: how could you color in the above graph so that it 3-colors successfully? Remember, no two nodes that touch via an edge may have the same color!

(Now you can color the graph.)

Excerpts from

Classic Nintendo Games are Computationally Hard

by Demaine et al

Start and Finish. The Start and Finish gadgets contain the starting point and goal for the character, respectively. In most of our reductions, these gadgets are trivial, but in some cases we need the player to be in a certain state throughout the construction, which we can force by making the Finish accessible only if the player is in the desired state, and the desired state may be entered at the Start. For example, in the case of Super Mario Bros., we need Mario to be big throughout the stage, so we put a Super Mushroom at the start and a brick at the Finish, which can be broken through only if Mario is big.

Variable. Each Variable gadget must force the player to choose one of two paths, corresponding to the variable x_i or its negation *not-*x_i being chosen as the satisfied literal, such that once a path is taken, the other path can never be traversed. Each Variable gadget must be accessible from and only from the previous Variable gadget, independent of the choice made in the previous gadget, in such a way that entering from one literal does not allow traversal back into the negation of the literal.

Clause and Check. Each Clause gadget must be accessible from (and initially, only from) the literal paths corresponding to the literals appearing in the clause in the original Boolean formula. In addition, when the player visits a Clause gadget in this way, they may perform some action that \unlocks" the gadget. The Check path traverses every Clause gadget in sequence, and the player may pass through each Clause gadget via the Check path if and only if the Clause gadget is unlocked. Thus the Check path can be fully traversed only if all the Variable gadgets have been visited from literal paths. If the player traverses the entire Check path, they may access the Finish gadget.

Crossover. The Crossover gadget must allow traversal via two passages that cross each other, in such a way that there is no leakage between the two.

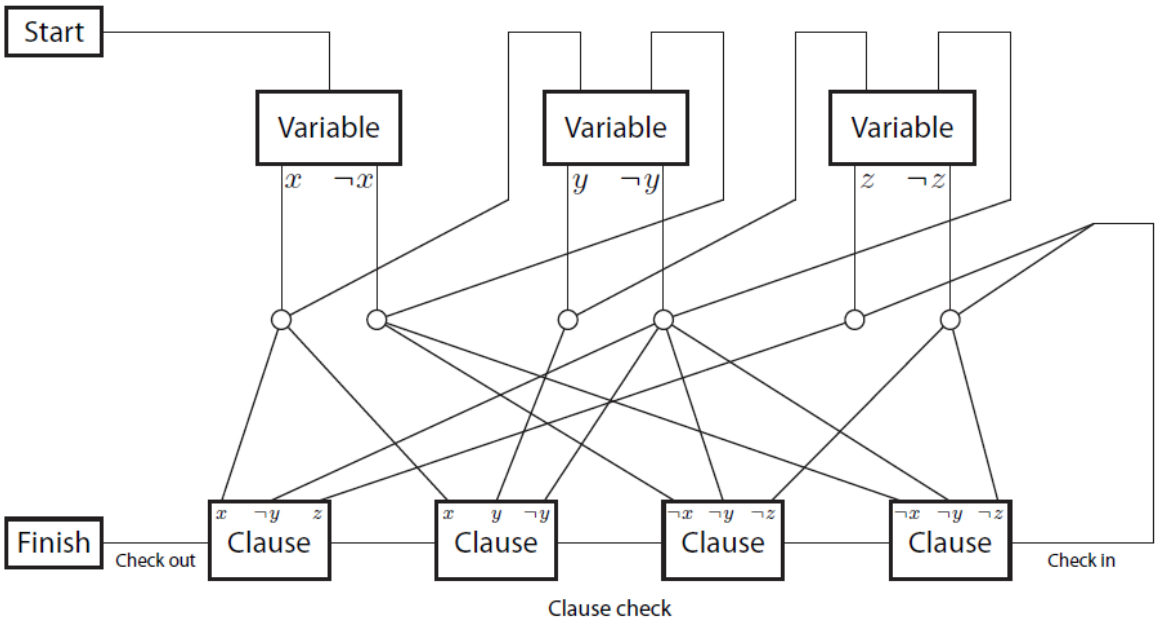


Figure 1: General framework for NP-hardness

Theorem 3.1. It is NP-hard to decide whether the goal is reachable from the start of a stage in generalized Super Mario Bros.

Proof. When generalizing the original Super Mario Bros., we assume that the screen size covers the entire level, because the game forbids Mario from going left of the screen. This generalization is not needed in later games, because those games allow Mario to go left.

We use the general framework provided in Section 2, so it remains only to implement the gadgets. The **Start and Finish gadgets** are straightforward. The **Start gadget**, shown in Figure 8, includes an item block containing a Super Mushroom which makes Mario into Super Mario. The Super Mushroom serves two purposes. First, Super Mario is two tiles tall, which prevents him from fitting into narrow horizontal corridors, a property essential to our other gadgets. Second, Super Mario is able to destroy bricks whereas normal Mario cannot. In order to force the player to take the Super Mushroom in the beginning, we block Mario's path with bricks in the **Finish gadget**, shown in Figure 9.

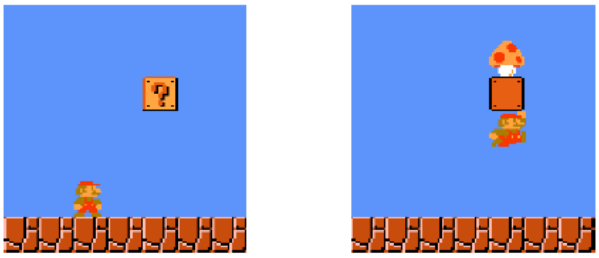


Figure 8: Left: Start gadget for Super Mario Bros. Right: The item block contains a Super Mushroom

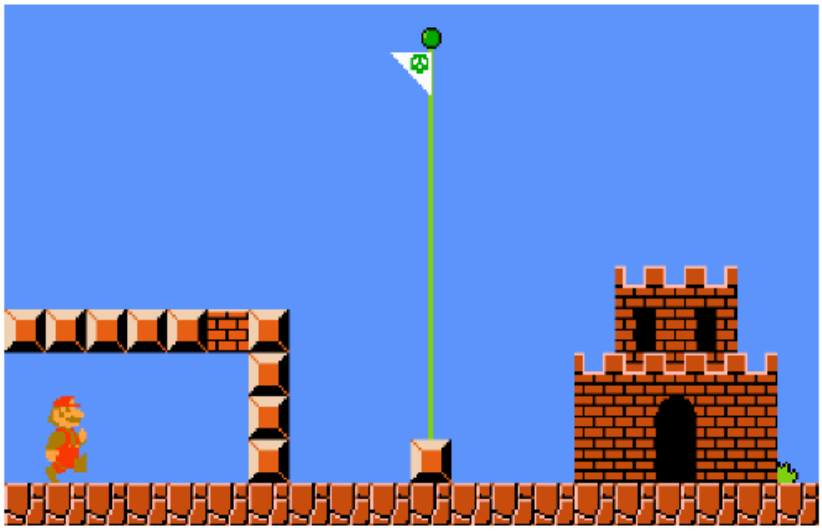


Figure 9: Finish gadget for Super Mario Bros.

Next, we implement the **Variable gadget**, illustrated in Figure 10. There are two entrances, one from each literal of the previous variable (if the variable is xi, the two entrances come from xi⊙1 and :xi⊙1). Once Mario falls down, he cannot jump back onto the ledges at the top, so Mario cannot go back to a previous variable. In particular, Mario cannot go back to the negation of the literal he chose. To choose which value to assign to the variable, Mario may fall down either the left passage or the right.

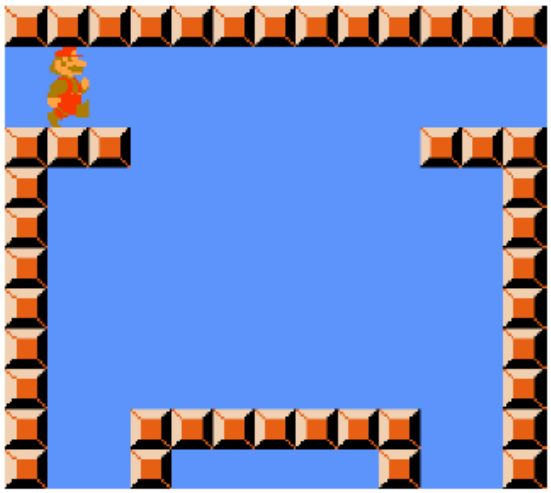


Figure 10: Variable gadget for Super Mario Bros.

Now we present the **Clause gadget**, illustrated in Figure 11. The three entrances at the bottom correspond to the three literals that are in the clause. When the Clause is visited, Mario hits the item block, which releases a Star into the area above, which will bounce around until Mario comes through the Check path to pick it up, allowing Mario to traverse through the Firebars safely. Without a Star, Mario cannot get through the Firebars without getting hurt.

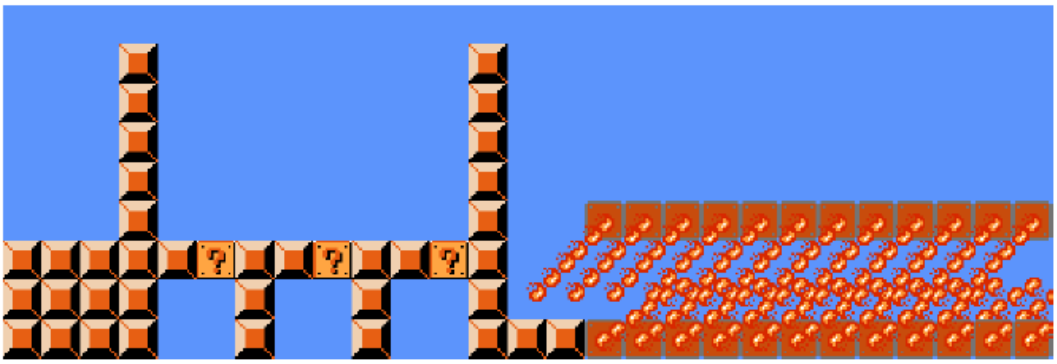


Figure 11: Clause gadget for Super Mario Bros.

Finally, we implement the **Crossover gadget**, illustrated in Figure 12. This unidirectional gadget enables Mario to traverse left-to-right or bottom-to-top, which suffices by Remark 2.1. Mario cannot enter this Crossover initially from the top or right.

Approaching from the left, Mario may get hurt by the Goomba and revert back to his small state, then walk through the narrow corridor and pick up a Super Mushroom from the item block on the other side, allowing him to break through the brick. Initially, there is no leakage to the bottom nor leakage to the top because small Mario cannot break bricks. Furthermore, there is not enough space for Mario to run and crouch-slide through the gap.

Approaching from the bottom, Mario may break the right brick on the lower level, jump onto the left brick of the lower level, break the left brick on the upper level, and jump onto the right brick of the upper level to proceed upwards. Mario must be big while traversing vertically in order to break the bricks, and because there is a one-way passage leading to the bottom entrance, Super Mario cannot break the bricks first, then return later as small Mario. Because Mario must be big this entire time, he cannot t through the narrow gaps on the sides and so there is no horizontal leakage.

There may be leakage from the horizontal path to the vertical path after traversing the vertical path. This is because, once the bricks have been broken, even small Mario can exit from the top. Fortunately, Remark 2.2 says that such a leakage can safely be allowed. It suffices that no leakage can occur from the vertical path to the horizontal one.

QED: Super Mario Bros is NP-hard.

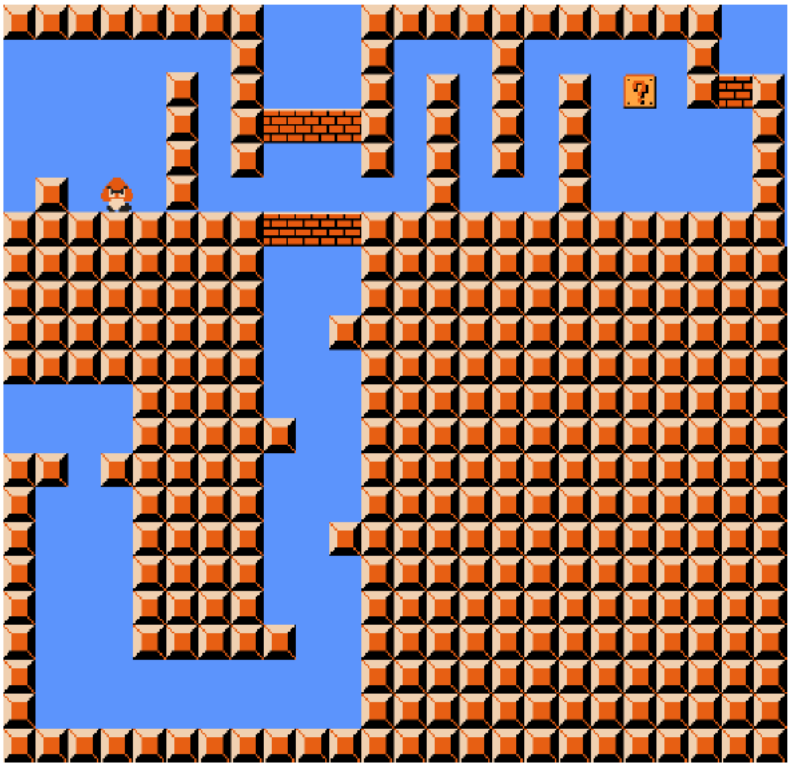


Figure 12: Crossover gadget for Super Mario Bros.

Theorem 7.1. It is NP-hard to decide whether a given target location is reachable from a given start location in generalized Pokemon.

Proof. We briefly describe the mechanism for enemy Trainer encounters in the Pokemon games. Each enemy Trainer has an orientation (which direction they are facing), a range of sight, and a set of Pokemon. If the player walks into the line of sight of the Trainer (and if such a line of sight is not occluded by some other element, e.g., another Trainer), the player is forced to stop, the Trainer walks toward the player until they are adjacent, and then battle ensues. Additionally, if the player approaches a Trainer from outside the Trainer's line of sight (i.e., from behind or the sides), the player may talk to the Trainer, activating the battle. If the player wins the battle, the Trainer will not move or attack again.

We prove NP-hardness by using the framework in Section 2. **Start and Finish gadgets** are trivial, hence it suffices to implement the Variable, Clause, and Crossover gadgets.
In our implementations, we use three kinds of objects. **Walls**, represented by dark grey blocks, cannot be occupied or walked through. Trainers' lines of sight are indicated by translucent rectangles. We have two types of **Trainers**. **Weak Trainers**, represented by red rectangles, are Trainers whom the player can defeat with certainty without expending any effort, i.e., without consuming PP or taking damage. **Strong Trainers**, represented by blue rectangles, are Trainers against whom the player will always lose.

We can implement weak and strong Trainers as follows, due to Istvan Chung. Weak Trainers each hold a Level 100 Electrode with maximum Speed and equipped with only the Self Destruct move. Strong Trainers each hold two Snorlaxes, with Speed of 30. The player has no items, and only one Pokemon in his team. For Generation I and II games (Red/Blue/Yellow and Gold/Silver/Crystal versions respectively), the player holds a Gastly which has learned Self Destruct using TM36, and its PP for its other moves have all been expended, so it can only use Self Destruct in battle. When the player encounters a weak Trainer, the enemy Electrode will move first and use Self Destruct, which deals no damage to Gastly since Self Destruct is a Normal type attack and Gastly is Ghost type, so the weak Trainer immediately loses. When the player encounters a strong Trainer, Gastly moves first and uses Self Destruct, causing the player to lose (even if it defeats the enemy Snorlax, the opponent holds another one). This implementation only works in Generations I and II since TM36 exists only in Generation I and the Time Capsule feature in Generation II allows a Gastly with Self Destruct to be traded from Generation I to Generation II. In Generations III, IV, and V, Gastly can be replaced by Duskull, which is allowed to learn the move Memento, which serves the same purpose as Self Destruct, via breeding.

We now implement each gadget. The **Variable gadget**, illustrated in Figure 33, must force the player to choose either an a-to-b traversal or an a-to-c traversal. We show that this is the case. The player enters through a. If the player wants to exit through b, they walk into the Trainer's line of sight, luring the Trainer down and opening up the passage to b, but closing the passage to c. On the other hand, if the player wants to exit through c, they walk up to the Trainer and talk to him, disabling the Trainer, so that they may walk around and exit through c.

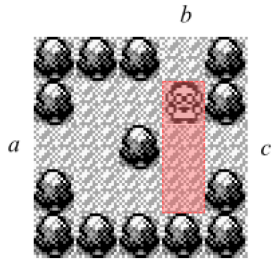


Figure 33: Variable gadget for Pokémon

The **Clause gadget**, depicted in Figure 34, is unlocked whenever the player enters from one of the top paths and talks to one of the three weak Trainers, disabling him. When the Check path is traversed, every weak guard who has not been disabled is lured down by the player, thus getting out of the line of sight of the rightmost strong Trainer. Eventually the player must cross that line of sight, and is reached by the strong Trainer if and only if no weak Trainer blocks him, which happens if and only if the Clause gadget is still locked. The leftmost strong Trainer prevents leakage from the Check path to the literals, in case the leftmost weak Trainer has been lured down.

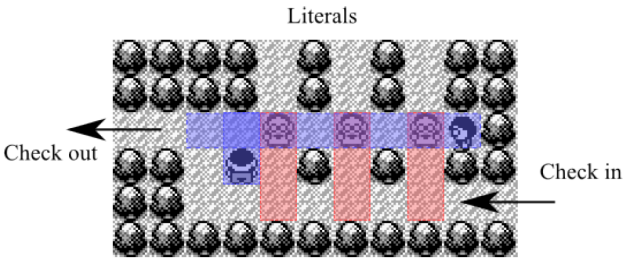


Figure 34: Clause gadget for Pokémon

Before presenting the **Crossover gadget**, we introduce the **Single-use path**, a path that can only be traversed once, and only in one direction. This is implemented by the gadget in Figure 35, which can be traversed only once, from a to b. Clearly, the player cannot enter via b, because that lures the weak Trainer to block the passage. Suppose the player enters through a. They can safely walk to b, because the weak Trainer is blocking the bottom strong Trainer's line of sight. However, to reach b, the player must lure the weak Trainer out of the line of sight of the strong Trainer, hence the player may never return in the reverse direction. Also, there is a strong Trainer above the weak Trainer, preventing the player from disabling the weak Trainer by entering from a.

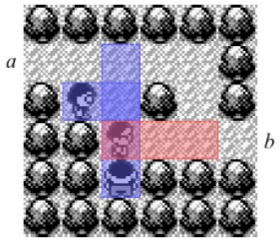


Figure 35: Single-use path for Pokémon

Finally, the **Crossover gadget** is shown in Figure 36. Each of its two passages, x-to-x_0 and y-to-y_0, is unidirectional and may be traversed only once, which suffices due to Remarks 2.1 and 2.2.

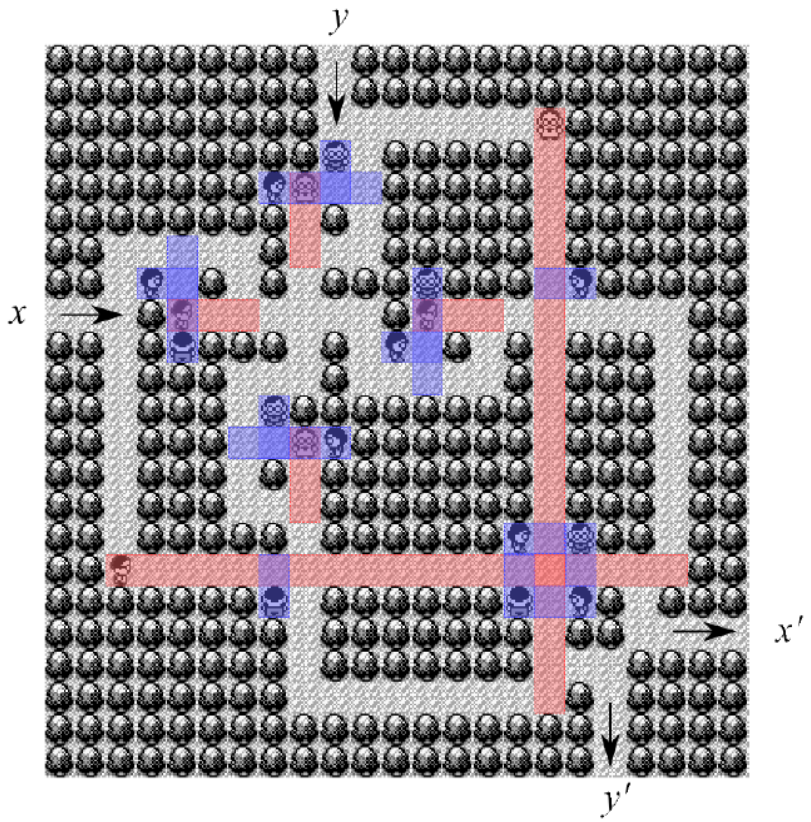


Figure 36: Crossover gadget for Pokémon

Because the Crossover gadget is symmetric, we assume, without loss of generality, that the player attempts to traverse the x-to-x_0 passage first. Upon entering from x, they go down and disable the bottom weak Trainer, then go back up and proceed to the right. Observe that the top-left part of the gadget is a \crossroads" made of four isometric copies of the Single-use path of Figure 35. Upon reaching it, the player is forced to go either down or right. If they go down, they eventually get stuck, because the crossroads is now unreachable due to the just traversed Singe-use path, and the y_0 exit is unreachable too, because the top weak Trainer is lured all the way down to block the player, as soon as they attempt to exit. On the other hand, if the player proceeds to the right upon entering the crossroads, they can safely reach the x_0 exit, because the leftmost weak Trainer has been previously disabled. Note that, by doing so, the player incidentally crosses the line of sight of the top weak Trainer, luring him down and disabling him.

Now, if the player wishes to traverse the y-to-y_0 passage, they must take the vertical Single-use paths of the crossroads, because the other two have already been traversed. At this point, the player has no choice but to exit from y_0, which is safely reachable because the top weak Trainer has already been disabled during the first traversal of the gadget, and is not blocking the way out.

QED: Pokemon is NP-hard.

NP. To show that Pokemon with only enemy Trainers is in NP, note that once a Trainer has been battled, they become inert. Moreover, each actual battle is bounded in length by a constant, because eventually all Pokemon must expend all their PP for their moves and use Struggle, a standard physical attack which hurts the opponent as well as the user. Moreover, each Pokemon only has four different attacks, so the branching factor of the game tree for each battle is constant (and hence the size of the game tree is also constant). Therefore, the player may nondeterministically guess a solution path through the overworld, and for each battle compute the winning strategy in constant time.

QED: Pokemon is NP-complete.

Some other theorems from this paper:

Theorem 4.1. It is NP-hard to decide whether the goal is reachable from the start of a stage in generalized Donkey Kong Country.

Theorem 5.1. It is NP-hard to decide whether a given target location is reachable from a given start location in generalized Legend of Zelda.

Theorem 6.1. It is NP-complete to decide whether a given target location is reachable from a given start location in generalized Metroid.

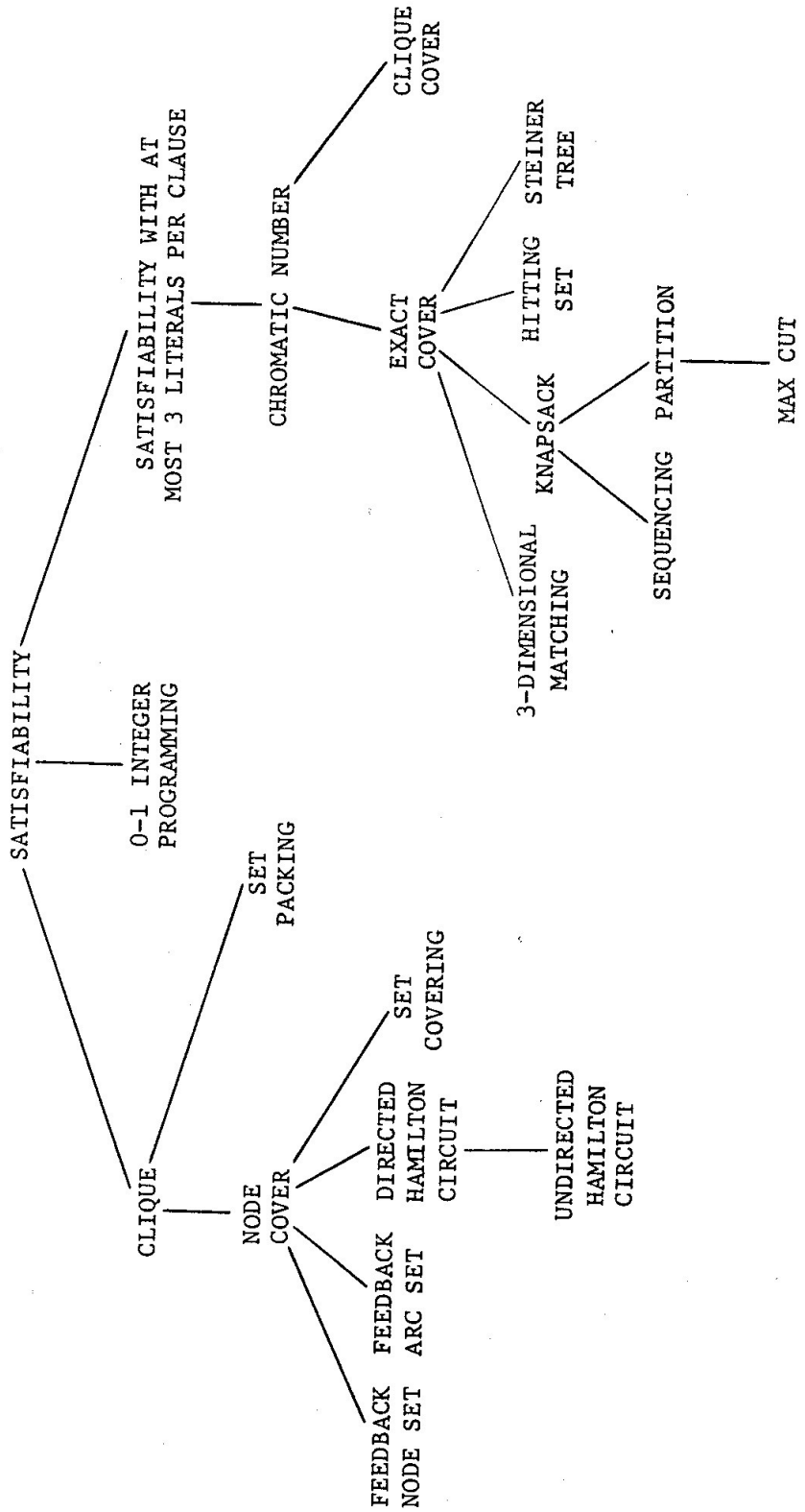


FIGURE 1 - Complete Problems