Combinatorial Game Representation and Analysis of Snort

Gio Carlo C. Borje
gborje@uci.edu
University of California, Irvine

ABSTRACT

The game of a Snort is a partisan, combinatorial game where solving which player has a winning strategy is a **PSPACE**-complete problem. We analyze the game of Snort by describing a state space prove that the game is unfair. However, rather than solving the game for general graphs, we demonstrate generalized winning strategies for a few common families of graphs such as star graphs, path graphs and cycle graphs.

1. INTRODUCTION

The game of Snort, invented by Simon Norton, is a two-player game with a planar map as a game board. The two players will be referred to as Red and Blue. The red and blue players have distinct game pieces denoted by their corresponding player's color. Players alternate turns placing pieces on the planar map with two restrictions:

1. Distinct game pieces cannot share a map position and
2. Distinct game pieces cannot be in adjacent map positions.

The game follows *normal play* where the last player to move wins.

1.1 Game Classification

The game can be classified as follows:

Determinate No elements of chance are presented.

Zero-sum No draws are possible.

Asymmetric Each player can have different strategies.

Perfect information No information of the game state is hidden from either player.

Sequential Players alternate turns.

Normal-play The last player to move wins the game.

Unfair The game is unfair because a winning strategy exists for a player on any given game state.

Solving the winner of a given game state is a **PSPACE**-complete problem as proven by Schaefer, so no polynomial time algorithms exists for optimal play unless **P** = **PSPACE** [2]. Hence, this paper will focus on solving special cases of maps. First, we must construct an appropriate game state representation.

2. GAME REPRESENTATION

We begin by using an equivalent, planar graph representation for the planar map. A planar graph is represented by a tuple, $(\mathbf{V}, \mathbf{E})$, where **V** is the set of vertices and the **E** is the set of edges. However, the planar graph is insufficient to represent the entire game because the game enforces both unary and binary constraints. Hence, we must reformulate the game as a discrete variable, finite domain constraint-satisfaction problem.

The constraint-satisfaction problem, **G**, is characterized by a three tuple: $(\mathbf{V}, \mathbf{D}, \mathbf{E})$ where we **V** is the set of vertices (or map regions), **D** is the set of domains from which vertices can obtain their colorings from and **E** is the set of unary and binary constraints.

A vertex is said to hold a *value* if it holds exactly one of the players' pieces. The value is the player's piece itself. Once a vertex obtains a value, it cannot hold another and it cannot be reassigned a value. For example, if the Red player places a piece on vertex 1, then the vertex 1 holds the value red. No further modifications can be made to the vertex 1.

The domain set is statically defined as the following set of *labels*. Labels are elements of the domain set which are sets themselves.

$$\mathbf{D} = \{\mathbf{A}, \mathbf{R}, \mathbf{B}, \mathbf{U}\}$$

Each label denotes a subset of the legal values that can be placed on a vertex:

$$\mathbf{A} = \{\text{Red}, \text{Blue}\}$$
$$\mathbf{R} = \{\text{Red}\}$$
$$\mathbf{B} = \{\text{Blue}\}$$
$$\mathbf{U} = \emptyset$$

A vertex is said to have a label if it can hold either of the values in the label's set. Conversely, we say that a label is

the domain of the vertex if its elements are all legal values for the vertex.

Initially, all vertices will have the label **A** such that either the Red or Blue player can place their piece on each vertex. When a player makes a move, a vertex holds a value and cannot obtain another. Hence, vertices which hold a piece have the label **U**.

The unary constraints of this graph are seen by the assignment of a label to a vertex. That is, when a non-**A** label is attached to a vertex, the label constricts the number of legal values of the vertex. For example, if the Red player selects vertex 1 on his turn, the vertex is labeled as **U** because no other player can select it. See figure 2.

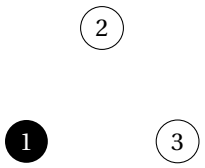

Figure 1: $3K_1$ **with Red move**

Further, the edges of this graph represent the binary constraints enforced by the game. Thus, the edges of the graph determine which sections of the graph to update when a move is made i.e. when a unary constraint is enforced on a vertex.

An edge, (i, j) where $i, j \in \mathbf{V}$, may only exist in the graph if exactly one of the following conditions hold: i and j are both labeled as **A** or exactly one of i and j is labeled as **A**.

2.1 Making Moves

A move is characterized by the following process:

1. A player selects a vertex that is labeled by their color or **A**.
2. The vertex obtains the label **U**.
3. All neighboring vertices are recursively updated to maintain *arc-consistency*.

Arc-consistency[1] is the state of the graph in which $(\mathbf{X}, \mathbf{Y}) \in \mathbf{E}$ is called consistent if $x \in \mathbf{X}$ and $y \in \mathbf{Y}$ are consistent. The value, $x \in \mathbf{X}$ is said to be consistent if and only if $x \in dom(\mathbf{X})$.

2.2 Terminal State Characterization

Definition 1. A game, **G**, is in a terminal state, $\mathbf{G_t}$, if and only if there are no vertices with the label **A** and there are no binary edges.

The terminal state has the property that a winner can be determined by the currently labeled vertices and the current player's turn. It is important to note that the set of gameover states is a subset of the set of terminal states. Subsequently, we will end our analysis of a game at a terminal state because a simple linear-time evaluation algorithm exists to evaluate the state.

The evaluation algorithm determines the *game value*, v of the state. The game value represents the payoff for a specified player. Without loss of generality, v will represent the payoff of the Red player and $\overline{v}$, the payoff for the Blue player, will simply be the negation of v according the rules of a zero-sum game.

The evaluation algorithm runs as follows:

1. Initialize an accumulator to 0.
2. Add the number of **R**-labeled vertices to the accumulator.
3. Subtract the number of **B**-labeled vertices from the accumulator.
4. If the current game state is Blue's turn, add $\frac{1}{2}$ to the accumulator.

According this evaluation algorithm, the following outcomes are possible relative to v.

- If $v > 0$, Red wins.
- If $v \leq 0$, Red loses.
- If $v = \frac{1}{2}$, Red wins by a single turn ahead.

Consider the following example in figure 2.2 where it is the Red player's Turn. We begin with an accumulator $a = 0$. There is one **R**-labeled vertex, so we add one: $a = 1$. There is one **B**-labeled vertex, so we subtract one: $a = 0$. The

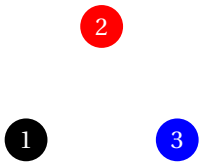

Figure 2: $3K_1$ **Terminal State on Blue's Turn**

game is balanced, hence the next player to move will lose i.e. this is a **P**-position. Since it is the Blue player's turn, we add $\frac{1}{2}$. Hence, $a = \frac{1}{2}$ and Red wins by a single turn ahead.

3. COMPUTABILITY AND COMPLEXITY BOUNDS ANALYSIS

Determining which player has a winning strategy is possible, though intractable, since the game is progressively bounded i.e. a valid terminal state will eventually be reached. Hence, the problem of determining which player has a winning strategy is computable.

We can determine the upper-bound on the number of possible game states using a constraint relaxation argument, because a game state can be completely represented

[1]Although arc-consistency maintenance is usually an expensive operation, $O(ed^3)$ through AC-3 [1], it can be shown that forward checking up to two vertices is sufficient due to the nature of the binary constraints i.e. $O(d^2)$.

by a discrete planar graph with finite values. That is, disregarding the conditions for which a vertex can obtain a label, a vertex has one of four labels exclusively. Hence, a planar graph with n vertices can have at most $o(4^n)$ potential states which both valid states and invalid states.

3.1 Unfairness

We can determine the fairness of the game by showing that a winning strategy exists for one of the players exclusively. For every game state, the game is progressively-bounded and there are no ties. It follows that a winning strategy exists for every game. Since a winning strategy exists for every game state, the game is unfair.

Instead of solving for the winning player for arbitrary graphs, however, we will solve for winning players of general graph families.

4. GENERAL GRAPH FAMILIES

For the following general families of graphs, we show empirical results which suggest a generalized winning strategy for each family of graphs. The results enables us to infer an optimal strategy for the winning player.

k	v of $\mathbf{K_{1,k}}$	v of $\mathbf{P_k}$	v of $\mathbf{C_k}$
1	0.5	0.5	0.5
2	1.5	1	1.5
3	2.5	2	2.5
4	3.5	1	0
5	4.5	1	1
6	5.5	0.5	0
7	6.5	1.5	1
8	7.5	1	0
9	8.5	2	1
10	9.5	1.5	0
11	10.5	1.5	0.5
12	11.5	1	0

Table 1: Empirical Game Values for Small Graphs

We begin with the families of graphs for which a generalized winning strategy is trivially known. When we outline a strategy, note that the Red player is the first player without loss of generality.

4.1 Star Graphs

Star graphs of $k+1$ vertices are denoted as $\mathbf{K_{1,k}}$. Trivially, selecting the center vertex is always the optimal strategy for the current player. Hence, all star graphs are **N**-positions.

As an example of the trivial strategy, consider the claw graph: Selecting the center vertex will isolate the surrounding vertices and restrict their values from the opposing player's color.

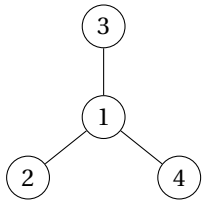

Figure 3: Claw, N-Position

4.2 Path Graphs

Path graphs with k vertices are denoted as $\mathbf{P_k}$. Path graphs contain $k-1$ edges that create binary links between vertices similar to a linked list.

Special points on the graph are defined as follows:

Definition 2. For all path graphs, $\mathbf{P_k}$ where $k \geq 3$, the leftmost vertex is said to be the head of the graph.

Definition 3. For all path graphs, $\mathbf{P_k}$ where $k \geq 3$, the rightmost vertex is said to be the tail of the graph.

THEOREM 1. *All path graphs,* $\mathbf{P_k}$ *with k vertices are* ***N****-positions.*

PROOF. To prove that theorem 1 holds, we must consider two cases:

- Every even-length path graph is an **N**-position.
- Every odd-length path graph is an **N**-position.

Case 1. The optimal strategy for path graphs of even-length is as follows:

1. Red player selects the vertex neighboring the head or tail. Without loss of generality, we assume the head.
2. The path graph disconnects into two subgraphs, $\mathbf{P_1}, \mathbf{P_{k-2}}$.
3. Blue player selects the vertex neighboring the leftmost **R**-labeled vertex, u, on the larger subgraph such that the u obtains label **U**.
4. Red player applies the same strategy as in step (3) such that the neither player will never have an game value advantage of greater than $\frac{1}{2}$. In the case of $\mathbf{P_4}$ with a **R**-labeled head, the game value is $\frac{1}{2}$. In the case of $\mathbf{P_6}$ with a **R**-labeled head, the game value is $-\frac{1}{2}$. All other even-length path graphs alternate this property.
5. By the end of the game, regardless of which player has the advantage in the larger subgraph, the Red player already had a game value advantage of 1. Hence, the Red player always wins.

Case 2. The optimal strategy for path graphs of odd-length is as follows:

1. Red player selects the center vertex.

2. Blue player selects the vertex which shares a mutual neighbor from rightmost **B**-labeled vertex or the head if no such vertex exists yet.

3. Red player selects the vertex which shares a mutual neighbor from the leftmost **R**-labeled vertex or the tail if no such vertex exists yet.

4. Steps (2) and (3) repeat until the graph is completely disconnected in which case both players will have the same number of colored vertices; however, Blue had one turn ahead in selecting vertices on a symmetric game as seen in step (2).

5. Hence, Blue will always lose i.e. Blue will be the next to move in the absolute gameover state.

Because the set of even-length path graphs union the set of odd-length path graphs encapsulates the entire set of path graphs, all path graphs, $\mathbf{P_k}$ where $k \geq 1$, are **N**-positions. □

Odd-length path graphs also yield the property that we call the *odd-rule*.

Definition 4. For all path graphs, $\mathbf{P_k}$ where k is odd, the odd-rule states that $\mathbf{P_k}$ will end on the opposing player's turn such that the current player will move afterwards if the game state were to continue.

Remark 1. Assuming that the leftmost vertex is blue and the rightmost vertex is red without loss of generality, the odd-rule holds.

4.3 Cycle Graphs

A cycle graph, $\mathbf{C_k}$, has k vertices with edges $(i, i+1 \pmod k)$ for $i \in [0, k]$. For cycle graphs with $k > 3$, the winning player alternates.

THEOREM 2. *For a cycle graph,* $\mathbf{C_k}$ *where* $k > 3$*, if* k *is even,* $\mathbf{C_k}$ *is a* ***P****-position.*

Beware that $\mathbf{C_4}$ is a special case that decomposes into $\mathbf{K_3}$ for Blue player. As an illustration of the strategy, see figure 4.3. Red player selects any vertex without loss of generality. Blue player has only one option: choose the vertex

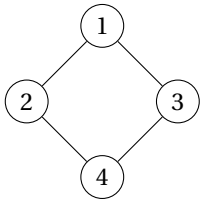

Figure 4: C_4, P-Position

across from it. Once the vertex has been selected, all of Red player's options become void. Hence, Blue player always wins.

The strategy for all even-sized cycle graphs where $k \geq 6$ decomposes a cycle-graph into two equal-length path-graphs, $2\mathbf{P}_{\frac{k-2}{2}}$. The strategy is as follows.

1. Red player selects a vertex without loss of generality.

2. Blue player selects the vertex directly across from it.

3. Two path graphs of same size are generated.

4. Since it is Red player's turn, we have shown that the first to move in the path graph always wins. Hence, Red player wins the first path graph.

5. Since Red player wins the first path graph, it follows that Blue player begins the second path graph. Hence, Blue player wins the second path graph.

6. By symmetry, any game value, v, obtained by the Red player on the first graph is obtained by the Blue player as $\overline{v}$. Consequently, the total game value is $v + \overline{v} = 0$. Consequently, the Blue player wins.

Note that after step (1), the odd-rule applies which simplifies the proof; however, the roles of the Red and Blue player are reversed and thus, the Red player will always lose.

THEOREM 3. *For a cycle graph,* $\mathbf{C_k}$ *where* $k > 3$*, if* k *is odd,* $\mathbf{C_k}$ *is an* ***N****-position.*

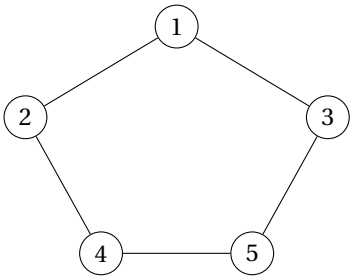

Figure 5: C_5, N-Position

The idea behind the strategy is that any vertex selected reduces the game to an even-sized path graph, $\mathbf{P_{k-1}}$ with a head and tail of the same label as the current player. On the following move, the path graph is broken into two smaller graphs which are even and odd in length. The odd graph has a special property which enforces a turn advantage. The strategy is as follows:

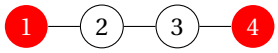

Figure 6: P_4 with Red Head and Tail, P-Position

1. Red player selects a vertex without loss of generality.

2. The graph unfolds into a path graph with $k-1$ vertices, $\mathbf{P_{k-1}}$ i.e. even-sized (see figure 4.3).

3. Blue player selects one of two center vertices in the path graph without loss of generality.

4. The graph unfolds into a two path graphs of size $\lceil \frac{k-2}{2} \rceil$ and $\lfloor \frac{k-2}{2} \rfloor$ each with a game value of zero. The two path graphs have the property that one is odd-length and the other is even-length. By the odd-rule, the Red player can choose the odd-length such that it is the Red player's turn to start the even-length game. Further, because all even-length graphs are **N**-positions, the Red player wins the even-length game. Because all subgames have been solved, the Red player wins.

5. GAME VARIATION BY PLAYERS

It is due to the number of players that the complexity of the game grows exponentially. Recall that that number of labels is also the number of subsets of all possible values that can be obtained by a vertex. We begin by noting that there the **A** and **U** labels are always required. However, for n players, there are $n+2$ possible values. Hence, the size of the state space becomes $o((n+2)^{|V|})$ where $|V|$ is the number of vertices of a specified map.

Trivially, the star graph family will always be an **N**-position regardless of the number of players.

6. CONCLUSION

We have shown that for non-cycle, graphs, the game of Snort is almost always biased towards the first player. On our simulation, we randomly-generated graphs through an Erdős-Rényi algorithm of vertex range $(5, 13)$ and edge probability of 0.2. Through 100 games using a minimax solver, it was seen that almost all games had winning strategies by the first player. Due to the immense state space size, graphs larger than 13 often took longer than 30 minutes to solve. Subsequently, a practical heuristic, if state-space search is intractable, is to select vertices of high-degree.

We have also shown a proof of the tractability of cycle graphs shows that proofs of tractability can be created for composite graphs i.e. graphs composed of several, simpler graphs. Hence, there are several open questions available regarding the tractability of other special cases of composite graphs using simple graph reductions.

7. REFERENCES

[1] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[2] Thomas J. Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185 – 225, 1978.