Sum and Product Game

Pan Lin

Supervised by Dr. Eric Babson

Fall 2023

Abstract

A Sum and Product Game is a logic puzzle first mentioned in a 1979 Gardner column. In this paper, we explore this game's properties and behaviors by modeling it as a pseudorandom bipartite graph and analyzing its structural properties. Moreover, we analyze the distribution of some specific substructures such as diamonds and fishes. Particularly, we discover the game's potential halting conditions, the strict upper bounds of the scatter plot of diamond patterns and the condition when diamonds become fishes. Overall, these works give some ideas for further research of our ultimate conjecture, that there exists an upper bound such that any Sum and Product Game either ends with a finite length lower than this bound or never halts.

Contents

1 Introduction

1.1 The Game Rule

A Sum and Product Game is a logic puzzle quoted from a 1979 Gardener column `www.math.uni-bielefeld.de/~sillke/PUZZLES/logic_sum_product`. In this game, Bob chooses two arbitrary integers greater than 2 and not greater than N, which are called the chosen answer numbers. Then Bob tells the sum of the two chosen numbers to Sara secretly and tells the product of the two chosen numbers to Peter secretly. Sara and Peter are trying to figure out what the two chosen are. The order does not matter. They can talk to each other but only with words "I know what the numbers are." or "I have no way to figure them out yet." honestly.

Example 1.1. For example, with $N = 10$, Bob picks 8 and 2. Then Bob tells Sara the sum 10 and tells Peter the product 16. Here is their conversation:
Peter: "I have no way to figure them out yet."
Sara: "I have no way to figure them out yet."
Peter: "I have no way to figure them out yet."
Sara: "I have no way to figure them out yet."
Peter: "I know what the numbers are."
Sara: "I know what the numbers are."

In this thesis, we are going to analyze this logic game and try to explore an open question:

Conjecture 1.1. *For arbitrary upper bound N for choosing the two numbers, is there some positive K such that the game never halts if and only if the length of the conversation is greater than K.*

1.2 How it works

Let's discuss what exactly Sara and Peter are communicating and what they are thinking about in such a restricted conversation. In the begining, Peter has the product 16, so he knows the answer must be one of the two pairs $(8, 2), (4, 4)$, and Sara's number can be 8 or 10. Then Peter cannot figure out which is the one they desire and has to tell Sara "I have no way to figure them out yet.".

For Sara, the answer must be among $(8, 2)$, $(7, 3)$, $(6, 4)$, $(5, 5)$, and Peter's number can be $16, 21, 24$ or 25. Since Peter did not figure out the answer at the beginning, Peter's numbers cannot be those that can be uniquely decomposed into the product of two integers greater than 2, that is 21 and 25, so $(7, 3)$ and $(5, 5)$ can be crossed out from Sara's list, but she still doesn't know which of $(8, 2)$ or $(6, 4)$ it is, so she has to tell Peter "I have no way to figure them out yet.".

Next, Peter knows that Sara's elimination is not done yet, and that Sara's list has at least two possible answers. If Sara's number is 10, then her possible answer list is $(8, 2), (6, 4)$, which we discussed above. If Sara's number is 8, then her possible answer list is $(6, 2), (4, 4)$, and $(5, 3)$ is crossed out. Since Peter still can't use the available information to get an answer from $(8, 2)$ and $(4, 4)$, he can only state that "I have no way to figure them out yet."...

As the length of conversation grows, the complexity of what they are thinking will be too complex to follow, so we better find a way to analyze the game globally.

2 Pseudorandom graph induced by a Sum and Product Game

Before discussing the game, let's make some conventions for convenience. Let's use A and B to denote the two numbers that Bob has chosen and suppose $A \geq B$ without loss of generality. Thus, an SPG (Sum and Product Game) can be uniquely determined by the three initial settings N, A, B.

In fact, even if A and B are not known in advance, once we know both Sara's and Peter's numbers, then we can uniquely determine A and B by solving the equation system $S = A + B$, $P = AB$, that is $A = \frac{S+\sqrt{S^2-4P}}{2}$, $B = \frac{S-\sqrt{S^2-4P}}{2}$.

Therefore, we can analyze this game by generating a bipartite graph instead of the complicated verbal analysis as in the previous section.

Definition 2.1. For $n \geq 2$, a graph induced by the Sum and Product Game with upper bound n is a bipartite graph $G_n := (S_n, P_n, \mathscr{E}_n)$ where

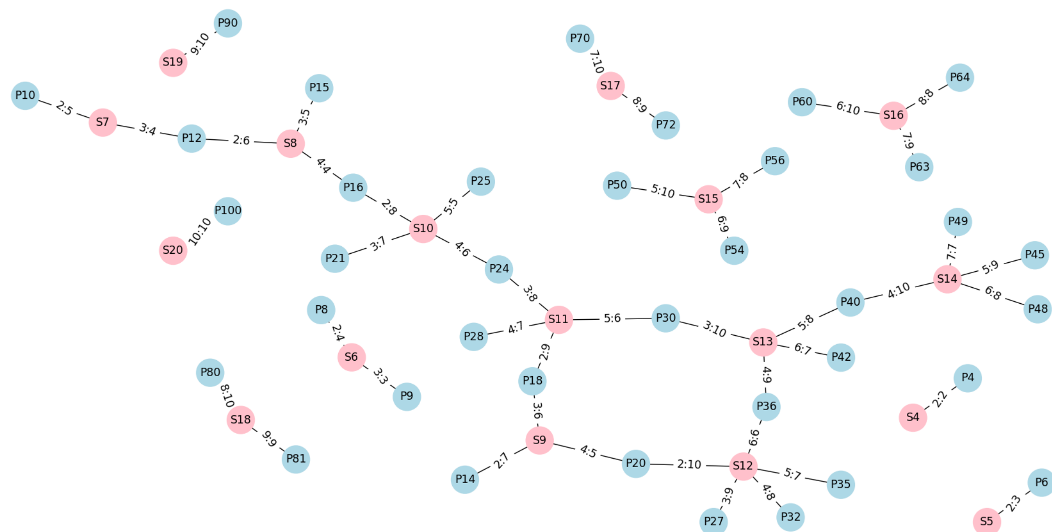
$$S_n := \{\mathbf{S}s : \exists a, b \in \{2..n\}.\ a + b = s\}$$

$$P_n := \{\mathbf{P}p : \exists a, b \in \{2..n\}.\ ab = p\}$$

$$\mathscr{E}_n := \{(\mathbf{S}s, \mathbf{P}p) : \exists a, b \in \{2..n\}.\ a + b = s \wedge ab = p\}$$

Let $\mathrm{E}(G)$ denote the set of edges and $\mathrm{S}(G)$ and $\mathrm{P}(G)$ the two sets of the vertices for a bipartite graph G. A bipartite graph $(S, P, \mathscr{E})$ can be simply regard as a graph $(S \cup P, \mathscr{E})$.

Example 2.1. If $N = 10$, then the graph G_{10} is like:



Whenever Peter states "I have no way to figure them out yet.", he is telling Sara that the information he currently have corresponds to multiple combinations of A and B. In the graph this is equivalent to saying that the possible product node is connected to more than one edge, so we can exclude all leaf product nodes from the graph. Otherwise, if Peter's number is from the leaves, he should state "I know what the numbers are." since the possible answer for him is unique, that is, the only edge connect to his product node. For Sara, in the current eliminated graph, if her sum node is adjacent to only one leaf product node, then she can determine the answer; otherwise, only Peter can get the answer. The analysis is similar for Sara.

To demonstrate how Sara and Peter eliminate options step by step as they exchange information, we introduce the following definitions:

Definition 2.2. Given $G = (S, P, \mathscr{E})$, let the set of leaf vertices

$$\begin{aligned}
\mathrm{LS}(G) &:= \{s \in S : \deg(s) \leq 1\} \\
\mathrm{LP}(G) &:= \{p \in P : \deg(p) \leq 1\} \\
\mathrm{LV}(G) &:= \mathrm{LS}(G) \ \cup \ \mathrm{LP}(G)
\end{aligned}$$

Definition 2.3. Given $G = (S, P, \mathscr{E})$, let the set of leaf edges

$$\mathrm{LE}(G) := \{(u, v) \in \mathscr{E} : u \in \mathrm{LV}(G) \vee v \in \mathrm{LV}(G)\}$$

Definition 2.4. Given $G = (S, P, \mathscr{E})$, let the pruned graph $\mathrm{prun}(G) := (S', P', \mathscr{E}')$ where

$$\begin{aligned}
S' &:= S \backslash \mathrm{LS}(G) \\
P' &:= P \backslash \mathrm{LP}(G) \\
\mathscr{E}' &:= \mathscr{E} \cap (S' \times P')
\end{aligned}$$

Definition 2.5. A pruning process of a graph G is a descending sequence of graphs

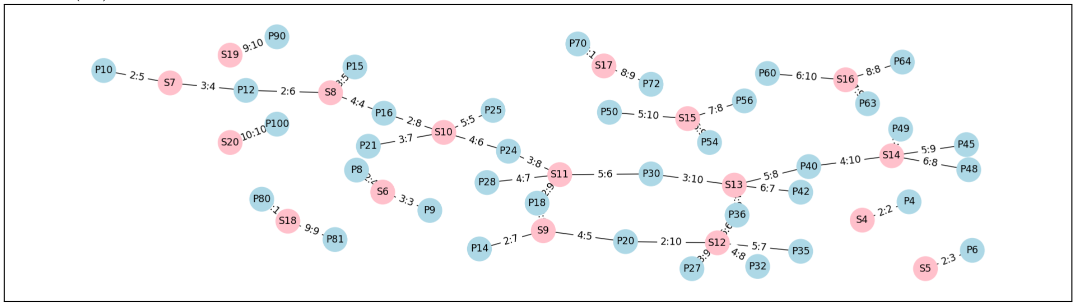
$$\mathrm{prun}^0(G) \supseteq \mathrm{prun}(G) \supseteq \mathrm{prun}^2(G) \supseteq \mathrm{prun}^3(G) \supseteq \ldots$$

where $\mathrm{prun}^0(G) := G$ and $\mathrm{prun}^{n+1}(G) := \mathrm{prun}(\mathrm{prun}^n(G))$.

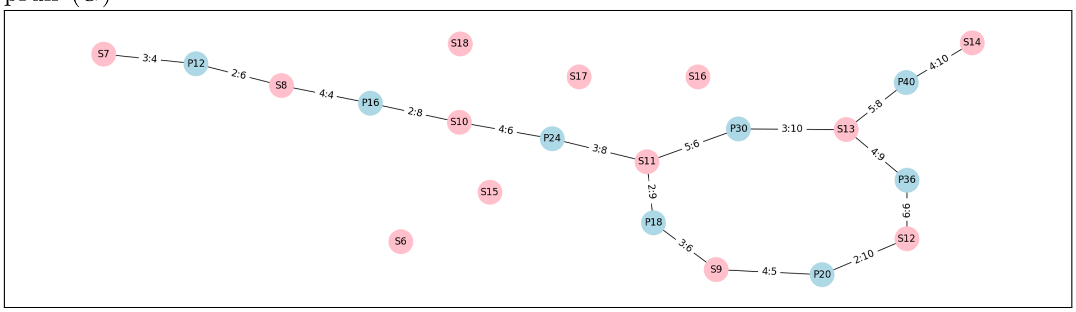
If n is even, note that $LS(prun^n(G)) = \varnothing$ so $prun^{n+1}(G)$ only removes product nodes compared to $prun^n(G)$; if n is odd, $LP(prun^n(G)) = \varnothing$ so $prun^{n+1}(G)$ only removes sum nodes compared to $prun^n(G)$.

Example 2.2. The pruning process of G_{10} is like:

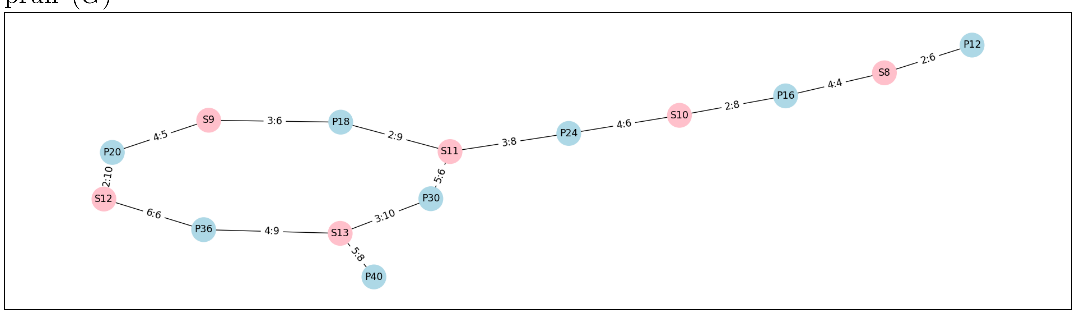
$\mathrm{prun}^0(G) =$



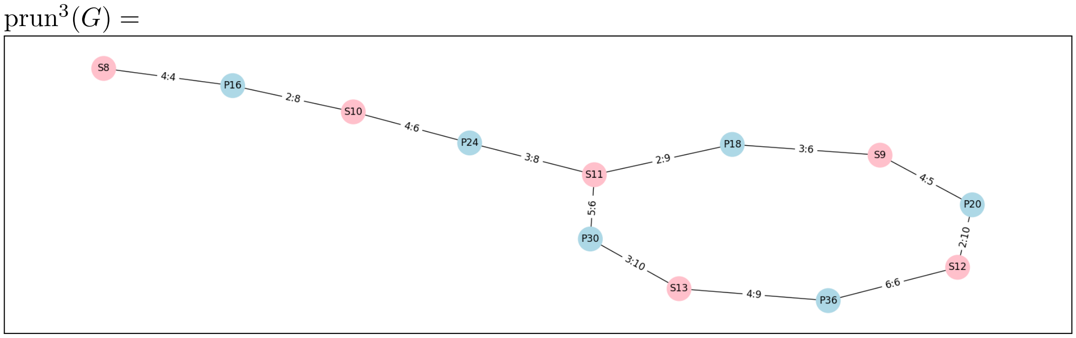
$\mathrm{prun}^1(G) =$



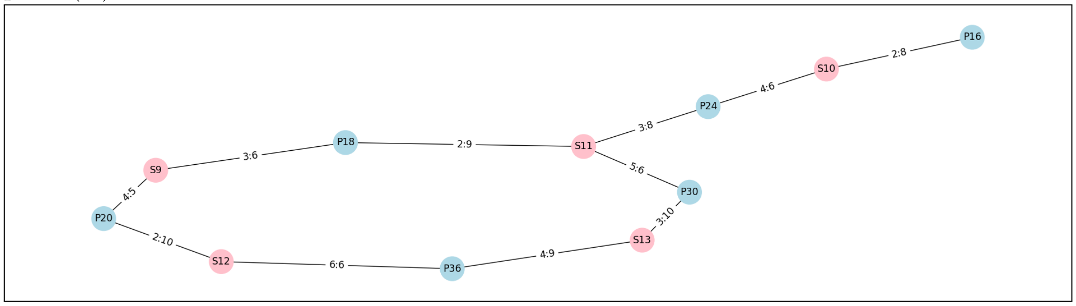
$\mathrm{prun}^2(G) =$



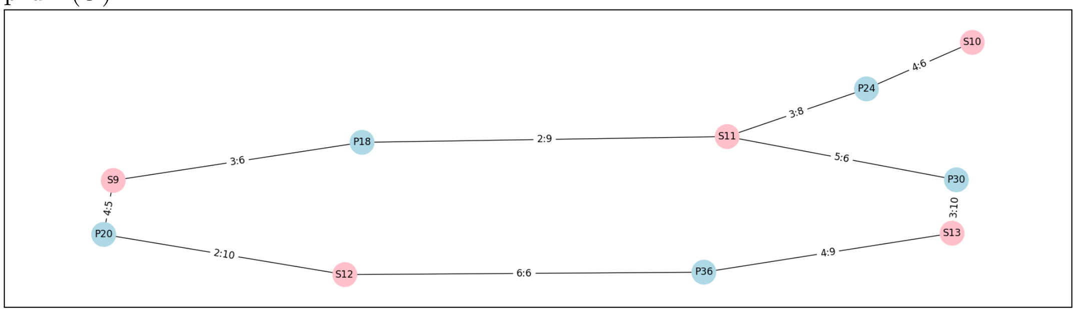
$\mathrm{prun}^3(G) =$



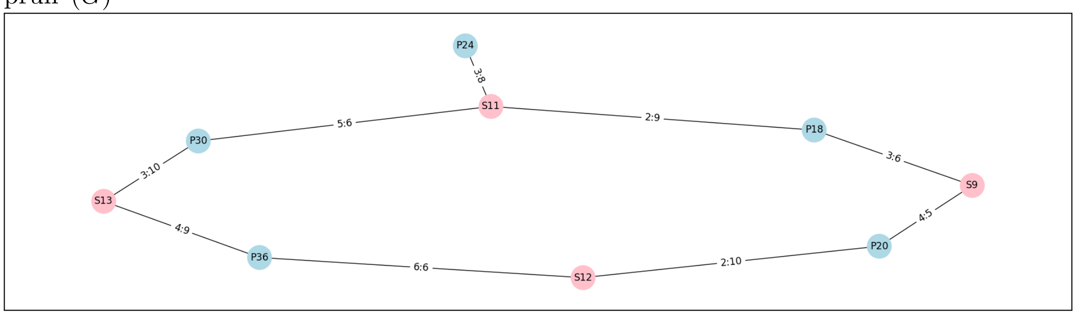
$\mathrm{prun}^4(G) =$



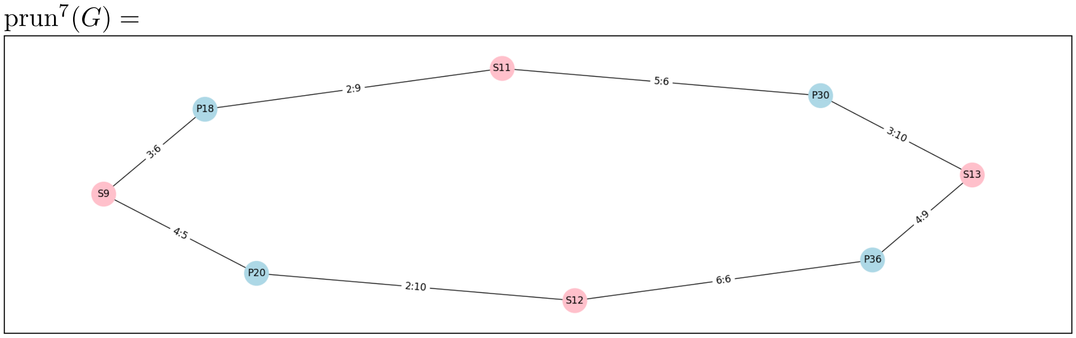
$\mathrm{prun}^5(G) =$



$\mathrm{prun}^6(G) =$



$\mathrm{prun}^7(G) =$



Note that $\mathrm{prun}^n(G) = \mathrm{prun}^7(G)$ for $n \geq 7$. We say $\mathrm{prun}^7(G)$ is a fixpoint of prun.

Definition 2.6. Given a graph $G = (V, E)$, $V' \subseteq V$ the induced subgraph is given by

$$G[V'] := (V', E \cap (V' \times V'))$$

Definition 2.7. Given a graph $G = (V, E)$, the 2-core of G is $K_2(G) := G[K_2(V)]$ where

$$K_2(V) := \bigcup \{V' \subseteq V : \forall v \in V'. \deg_{G[V']}(v) \geq 2\}$$

It is saying, the 2-core of G is the maximal subgraph of G with no leaves.

Definition 2.8. A filtering sequence of graph G is a sequence of set

$$\mathrm{LE}(\mathrm{prun}^0(G)), \mathrm{LE}(\mathrm{prun}^1(G)), \mathrm{LE}(\mathrm{prun}^2(G)), \ldots$$

Let $\mathrm{LE}_n(G) := \mathrm{LE}(\mathrm{prun}^n(G))$.

property 2.1. $\mathrm{LE}_n(G) = \mathrm{prun}^n(G) \backslash \mathrm{prun}^{n+1}(G)$ *for* $n \geq 0$

Intuitively, the filtering sequence is like cabbage leaves that are plucked off until nothing to prune and the 2-core of G remains.

property 2.2. $\mathrm{LE}_i(G) \cap \mathrm{LE}_j(G) = \varnothing$ *for* $i > j \geq 0$

property 2.3. $E(G) = \bigsqcup_{i=0}^{\infty} \mathrm{LE}_i(G) \sqcup K_2(E(G))$

Thus, we classify each edge with respect to their "survival time" in the pruning process.

Definition 2.9. Given a graph $G = (V, E)$, the lifetime is a function $\mathrm{life}_G : E \to \mathbb{N} \cup \{\infty\}$ with

$$\mathrm{life}_G(v) := \begin{cases} n, & \text{if } v \in \mathrm{LE}_n(G) \\ \infty, & \text{if } v \in K_2(E) \end{cases}$$

Consider completing a sentence as a turn, and let the number of turns it takes to start the game until someone says the first "I know..." be the length of the game.

Definition 2.10. Denote the length of a Sum and Product Game with initial setting N, A, B by $\text{len}(N, A, B)$.

Theorem 2.4. $\text{len}(N, A, B) = \text{life}_{G_N}((\mathbf{S}(A+B), \mathbf{P}AB)) + 1$

Example 2.3. From example 1.1, $\text{len}(10, 8, 2) = 5$.

In fact, there are four possible outcomes from the pruning process:

1. Peter and Sara can never determine A and B even after any rounds.

2. Peter is able to determine A and B, but Sara cannot, and the game ends.

3. Sara is able to determine A and B but Peter cannot, and the game ends.

4. Sara and Peter are both able to determine A and B.

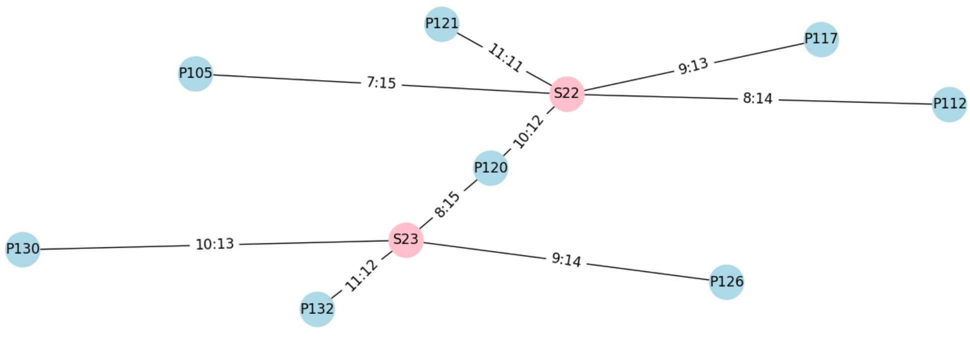
Theorem 2.5. *Given the initial setting N, A, B, let $s = A + B$ and $p = AB$, the results can be determined by following process:*

```
n := life_{G_n}(Ss, Pp);
if (Ss, Pp) ∈ K_2(𝓔_N) then
    return OutCome 1
else
    if 2 | n then
        if |{p̃ ∈ P_n : (Ss, Pp̃) ∈ LE_n(G_n)}| = 1 then
            return OutCome 4
        else
            return OutCome 2
        end
    else
        if |{s̃ ∈ S_n : (Ss̃, Pp) ∈ LE_n(G_n)}| = 1 then
            return OutCome 4
        else
            return OutCome 3
        end
    end
end
```

Example 2.4. Suppose $N, A, B = 10, 6, 5$, the game never ends as $(\mathbf{S}11, \mathbf{P}30) \in K_2(\mathscr{E}_{10})$, so $\text{len}(10, 6, 5) = \text{life}_{G_{10}}((\mathbf{S}11, \mathbf{P}30)) = \infty$

Example 2.5. Suppose $N, A, B = 10, 7, 7$, Peter can immediately get the answer, but Sara cannot determine the answer since she has three different possible options for A, B, that is $(7, 7), (9, 5), (8, 6)$.

Example 2.6. Suppose $N, A, B = 15, 12, 10$, Sara can get the answer in turn 2, but Peter cannot determine the answer since he has two different possible options for A, B, that is $(12, 10), (15, 8)$.



Example 2.7. Suppose $N, A, B = 10, 8, 2$, the game ends at turn 5.

Definition 2.11. For $\alpha \in \frac{1}{2}\mathbb{N}_{\geq 0}$, for $A \geq 2 + \alpha$ with $A - \alpha \in \mathbb{Z}$, let $E(A, \alpha) := (\mathbf{S}2A, \mathbf{P}(A^2 - \alpha^2))$

3 Pattern diagram and substructure

The graph generated by the Sum and Product Game grows chaotically as N increases. In order to study some features of parts of the graphs. We can abstract the some patterns out and discuss them individually.

Definition 3.1. A pattern diagram is a bipartite graph $(S, P, \mathscr{E})$.

Definition 3.2. Given a pattern diagram D, a D-indexed substructure bounded by n is an injective map $\sigma : D \hookrightarrow G_n$. Denote the set of D-indexed substructure bounded by n as G_n^D.

4 Diamond substructure

Definition 4.1. A diamond shape is a bipartite graph $\Diamond := (S_\Diamond, P_\Diamond, \mathscr{E}_\Diamond)$ where $S_\Diamond := \{\mathbf{S}s_1, \mathbf{S}s_2\}$, $P_\Diamond := \{\mathbf{P}p_1, \mathbf{P}p_2\}$, $\mathscr{E}_\Diamond := \{(\mathbf{S}s_1, \mathbf{P}p_1), (\mathbf{S}s_1, \mathbf{P}p_2), (\mathbf{S}s_2, \mathbf{P}p_1), (\mathbf{S}s_2, \mathbf{P}p_2)\}$

Definition 4.2. A diamond substructure is a $\Diamond$-indexed substructure .

Theorem 4.1. *Given $N, A, B, \alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathbb{N}$ with the following condtion:*

1. $4 \le B < A \le 2N$
2. $1 \le \alpha_1, \alpha_2, \beta_1, \beta_2 \le A - 2$
3. $\alpha_2 < \alpha_1$ *and* $\beta_2 < \beta_1$
4. A, α_1, α_2 *have the same parity*
5. B, β_1, β_2 *have the same parity*

, then $A^2 - \alpha_1^2 = B^2 - \beta_1^2$ and $A^2 - \alpha_2^2 = B^2 - \beta_2^2$ if and only if there exists a diamond substructure $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2) := (S, P, \mathscr{E})$ where $S := \{\mathbf{S}A, \mathbf{S}B\}$, $P := \{\mathbf{P}\frac{A^2-\alpha_1^2}{4}, \mathbf{P}\frac{A^2-\alpha_2^2}{4}\}$, $\mathscr{E} := \{E(\frac{A}{2}, \frac{\alpha_1}{2}), E(\frac{A}{2}, \frac{\alpha_2}{2}), E(\frac{B}{2}, \frac{\beta_1}{2}), E(\frac{B}{2}, \frac{\beta_2}{2})\}$

$$
\begin{array}{ccccc}
 & \frac{A}{2}\pm\frac{\alpha_1}{2} & \mathbf{S}A & \frac{A}{2}\pm\frac{\alpha_2}{2} & \\
\mathbf{P}(\frac{A^2}{4}-\frac{\alpha_1^2}{4}) = \mathbf{P}(\frac{B^2}{4}-\frac{\beta_1^2}{4}) & & & & \mathbf{P}(\frac{A^2}{4}-\frac{\alpha_2^2}{4}) = \mathbf{P}(\frac{B^2}{4}-\frac{\beta_2^2}{4}) \\
 & \frac{B}{2}\pm\frac{\beta_1}{2} & \mathbf{S}B & \frac{B}{2}\pm\frac{\beta_2}{2} &
\end{array}
$$

When we say "given a valid diamond $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$", we are actually saying "given $N, A, B, \alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathbb{N}$ satisfying the condition of theorem 4.1".

Example 4.1. $\Diamond(24, 16, 12; 21, 11, 3)$ is a diamond substructure.

Example 4.2. $\Diamond(17, 13, 11; 13, 7, 1)$ is a diamond substructure.

property 4.2. *Given a valid $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$, then A, α_1, α_2 have the same parity, and B, β_1, β_2 have the same parity, but A and B may have different parities.*

Proof. If A and α_1 have different parities, then $\frac{A}{2} + \frac{\alpha_1}{2} \notin \mathbb{N}$ which is contradictory to the assumption. Similarly, we can check other cases with $(A, \alpha_2), (B, \beta_1), (B, \beta_2)$. Example 4.1 gives the case where A, B have different parities. $\square$

Theorem 4.3. *Given a valid* $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$*, then* $A^2 - B^2 = \alpha_1^2 - \beta_1^2 = \alpha_2^2 - \beta_2^2$

Proof. Rearrange the equation $A^2 - \alpha_1^2 = B^2 - \beta_1^2$ to $A^2 - B^2 = \alpha_1^2 - \beta_1^2$, and rearrange the equation $A^2 - \alpha_2^2 = B^2 - \beta_2^2$ to $\alpha_1^2 - \beta_1^2 = A^2 - B^2 = \alpha_2^2 - \beta_2^2$. Then □

5 Boundary condition

Theorem 5.1. *Given* $N \geq 2$*. For* $A, B, C, D \in \{2..N\}$ *with* $A \geq B$ *and* $e \geq 0$*, if* $AB = CD$ *and* $C + D + e = A + B$*, then* $A + B \leq 2A + e - 2\sqrt{eA}$*.*

Proof. We know

$$\begin{aligned}
&(2A - (C + D))^2 - 4eA \\
&= 4A^2 - 4A(C + D) + (C + D)^2 - 4((A + B) - (C + D))A \\
&= 4A^2 - 4A(C + D) + (C + D)^2 - 4A^2 - 4AB + 4A(C + D) \\
&= (C + D)^2 - 4AB \\
&= (C + D)^2 - 4CD \\
&= (C - D)^2 \geq 0
\end{aligned}$$

By rearranging the inequality, it follows that $A + B \leq 2A + e - 2\sqrt{eA}$. □

6 Distribution of Diamonds

We note that if a game cannot be stopped, it means that the edge corresponding to the setting of that game is on a loop. Now, let us start our discussion with the simplest loop, the diamond substructure.

As shown below, this is a scatterplot of all the diamonds contained in G_N where the orange, yellow, green, and blue scatters are for cases $N = 100, 150, 200, 250$, respectively. The horizontal coordinates of each point in the graph indicate smaller sum nodes and the vertical coordinates indicate larger sum nodes, so there should be no points below the $y = x$ line. Note that a point can correspond to several different diamonds, since product nodes can be different.

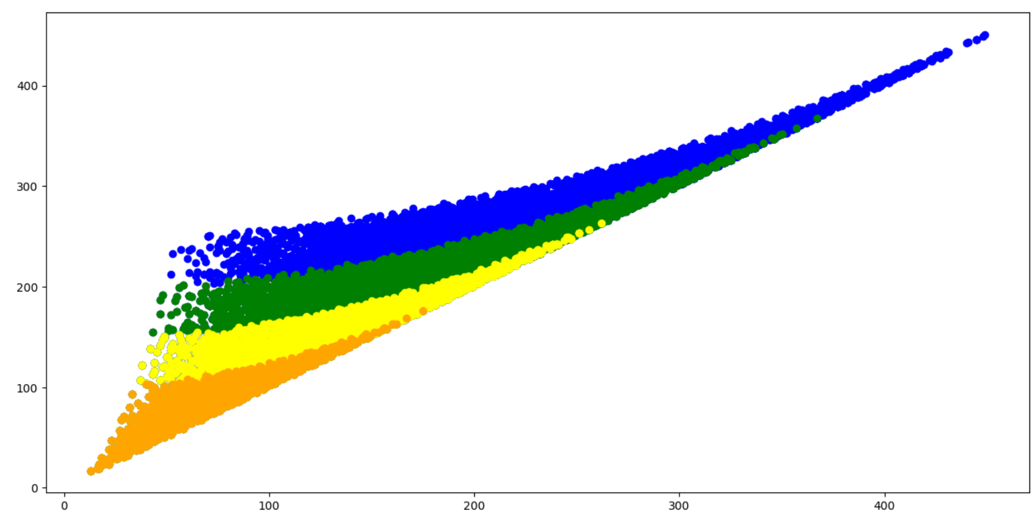
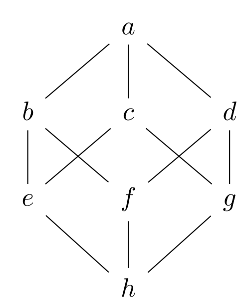


Figure 6.1: $x = A$, $y = B$

In addition to $y = x$, this scatter appears to be bounded by two different curves, where the curve on the left is independent of N, while the curves on the right shift upwards as N increases.



Theorem 6.1. *For $a, b, c, d, e, f, g, h \in \mathbb{N}_+$, if it satisfies all the following condition:*

1. $2 \leq abdf - cegh - acdg + befh < abdf + cegh + abce + dfgh \leq N$
2. $\left|\begin{smallmatrix} b & d \\ g & e \end{smallmatrix}\right| \cdot \left|\begin{smallmatrix} a & f \\ h & c \end{smallmatrix}\right| \neq 0$

then there exists a diamond substructure $\Diamond(\frac{1}{2}(abdf + cegh), \frac{1}{2}(abce + dfgh), \frac{1}{2}(acdg + befh); \frac{1}{2}(abdf - cegh), \frac{1}{2}(abce - dfgh), \frac{1}{2}(acdg - befh))$

Proof. We can arrange a, b, c, d, e, f, g as grid points. By property 4.3, we know $(A + B)(A - B) = (\alpha_1 + \beta_1)(\alpha_1 - \beta_1) = (\alpha_2 + \beta_2)(\alpha_2 - \beta_2)$. Let $A = \frac{1}{2}(abdf + cegh)$, $\alpha_1 = \frac{1}{2}(abce + dfgh)$, $\alpha_2 = \frac{1}{2}(acdg + befh)$, $B = \frac{1}{2}(abdf - cegh)$, $\beta_1 = \frac{1}{2}(abce - dfgh)$, $\beta_2 = \frac{1}{2}(acdg - befh)$. Since $cegh > 0$, we know $A \neq B$. To prove the product nodes are different, by theorem 4.1, it suffices to show that $\alpha_1 \neq \alpha_2$, i.e. $abce + dfgh \neq acdg + befh$, which can be rearranged as $(be - dg)(ac - fh) \neq 0$ $\square$

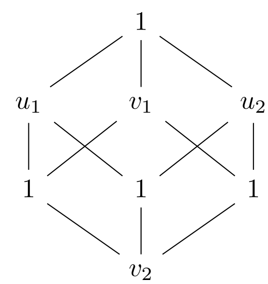
Corollary 6.2. *For $u_1, u_2, v_1, v_2 \in \mathbb{N}_+$, if it satisfies all the following condition:*

1. $2 \leq u_1(u_2 - v_2) - v_1(u_2 + v_2) < (u_2 + v_1)(u_1 + v_2) \leq N$

2. $u_1 \neq u_2$ *and* $v_1 \neq v_2$

then there exists a diamond substructure $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$ *where* $A = \frac{1}{2}(u_1u_2 + v_1v_2)$, $B = \frac{1}{2}(u_1u_2 - v_1v_2)$, $\alpha_1 = \frac{1}{2}(u_1v_1 + u_2v_2)$, $\alpha_2 = \frac{1}{2}(u_2v_1 + u_1v_2)$, $\beta_1 = \frac{1}{2}(u_1v_1 - u_2v_2)$, $\beta_2 = \frac{1}{2}(u_2v_1 - u_2v_1)$,

Consider the extreme case $u = u_1 = u_2$, $v = v_1 = v_2$. Then, a diamond substructure is given by $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$ where $A = \frac{1}{2}(u^2 + v^2)$, $B = \frac{1}{2}(u^2 - v^2)$, $\alpha_1 = uv$, $\alpha_2 = uv$, $\beta_1 = 0$, $\beta_2 = 0$. It follows that $u = \sqrt{A + B}$ and $v = \sqrt{A - B}$.



Theorem 6.3. *Given game upper bound N, let* $\sigma : G_N^{\Diamond} \to \mathbb{R}^2$, $\sigma(\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)) := (A + B, A - B)$, *then the scatter plot of the point set* $\sigma(G_N^{\Diamond})$ *is bounded above by the curve* $y = (-\sqrt{x} + 2\sqrt{N})^2$

Proof. We know $\frac{A}{2} + \frac{\alpha_1}{2}$ is the greatest answer number appearing in $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$ which is chosen from the range $\{2..N\}$; therefore, we have $\frac{A}{2} + \frac{\alpha_1}{2} \leq N$, implying $2N \geq A + \alpha_1 = \frac{1}{2}(u^2 + v^2) + uv = \frac{1}{2}(u + v)^2 \geq \frac{1}{2}(\sqrt{A + B} + \sqrt{A - B})^2$, implying $\sqrt{A - B} \leq -\sqrt{A + B} + 2\sqrt{N}$. $\square$

Theorem 6.4. *Given game upper bound N, let* $\sigma : G_N^{\Diamond} \to \mathbb{R}^2$, $\sigma(\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)) := (A + B, A - B)$, *then the scatter plot of the point set* $\sigma(G_N^{\Diamond})$ *is bounded above by* $y = (\sqrt{x} + \epsilon)^2$ *for some real* $\epsilon \geq 0$.

Proof. Similarly, we know $u - v$ must be not smaller than some constant $\epsilon \geq 0$; otherwise, it follows $B = 0$ out of range. Thus, $\epsilon \leq u - v = \sqrt{A + B} - \sqrt{A - B}$ implies $\sqrt{A - B} \leq \sqrt{A + B} - \epsilon$. $\square$

The following scatterplot Figure 6.2 transforms the coordinates of Figure 6.1, changing from $x = B$, $y = A$ to $x = A + B$, $y = A - B$. The curves $y = (-\sqrt{x} + 2\sqrt{N})^2$ and $y = (\sqrt{x} + \epsilon)^2$ are also shown.

In fact, within the bound $N \leq 500$, the diamond with the minimal $\sqrt{A+B} - \sqrt{A-B}$ is the one in Example 4.2, so we know $\epsilon \leq \sqrt{30} - 2$. In addition, Figure 6.3 below shows a scatterplot derived by squaring the coordinates of Figure 6.2, transforming $x = A+B, y = A-B$ into $x = \sqrt{A+B}$, $y = \sqrt{A-B}$. This transformation linearize the bounding curves to $y = -x + 2\sqrt{N}$ and $y = x + \epsilon$.

Conjecture 6.1. $\epsilon = \sqrt{30} - 2$

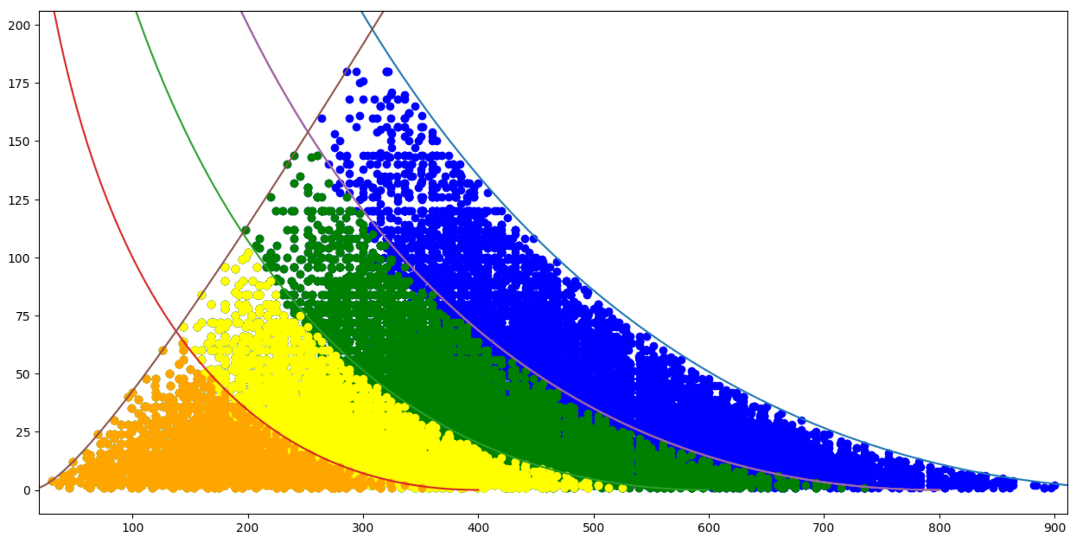


Figure 6.2: $x = A + B$, $y = A - B$

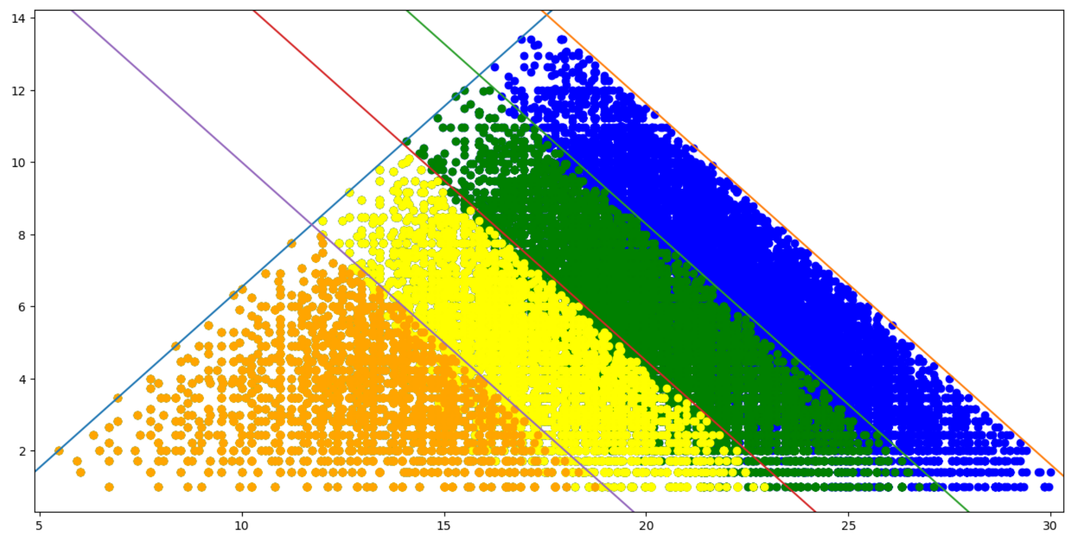


Figure 6.3: $x = \sqrt{A+B}$, $y = \sqrt{A-B}$

7 Fishes

Definition 7.1. A fish shape is a bipartite graph Fish $:= (S_{\text{Fish}}, P_{\text{Fish}}, \mathscr{E}_{\text{Fish}})$ where

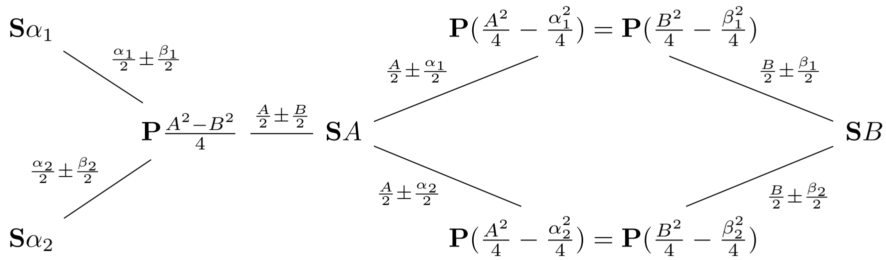
$S_{\text{Fish}} := \{\mathbf{S}s_1, \mathbf{S}s_2, \mathbf{S}s_3, \mathbf{S}s_4\}$, $P_{\text{Fish}} := \{\mathbf{P}p_1, \mathbf{P}p_2, \mathbf{P}p_3\}$,

$\mathscr{E}_{\text{Fish}} := \{(\mathbf{S}s_1, \mathbf{P}p_1), (\mathbf{S}s_1, \mathbf{P}p_2), (\mathbf{S}s_2, \mathbf{P}p_1), (\mathbf{S}s_2, \mathbf{P}p_2), (\mathbf{S}s_1, \mathbf{P}p_3), (\mathbf{S}s_3, \mathbf{P}p_3), (\mathbf{S}s_4, \mathbf{P}p_3)\}$

Theorem 7.1. *Given game upper bound N and a valid diamond $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$, if it satisfies all the following condition:*

1. $\alpha_1, \alpha_2 \in \{4..2N\} \backslash \{B\}$
2. *Each of the pairs $(A, B), (\alpha_1, \beta_1), (\alpha_2, \beta_2)$ has the same parity*
3. $A - B, \alpha_1 - \beta_1, \alpha_2 - \beta_2 \geq 4$

then there exists a substructure like the diagram shown below:



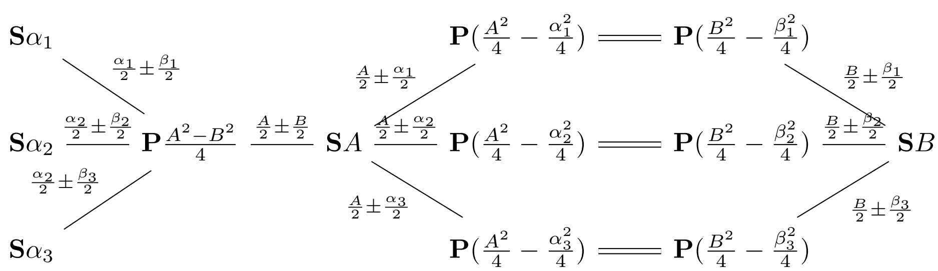
Proof. By given conditions, we know $\mathbf{S}\alpha_1, \mathbf{S}\alpha_2 \in S_N$. As A, B have the same parity, it is the case $A+B$ and $A-B$ are even, implying $\frac{A^2-B^2}{4}$ are integers. As $A+B \geq A-B \geq 4$, we know $\frac{A^2-B^2}{4} \geq 4$, so $\mathbf{P}\frac{A^2-B^2}{4} \in P_N$. As $\frac{A}{2} - \frac{B}{2}, \frac{\alpha_1}{2} - \frac{\beta_1}{2}, \frac{\alpha_2}{2} - \frac{\beta_2}{2} \geq 2$, by theorem 4.3, we have $\frac{A^2-B^2}{4} = \frac{\alpha_1^2-\beta_1^2}{4} = \frac{\alpha_2^2-\beta_2^2}{4}$, implying $E(\frac{A}{2}, \frac{B}{2}), E(\frac{\alpha_1}{2}, \frac{\beta_1}{2}), E(\frac{\alpha_2}{2}, \frac{\beta_2}{2}) \in \mathscr{E}_N$. $\square$

The following theorems show two basic ways that two diamonds can stick together to form some interesting fish structures.

Theorem 7.2. *Given game upper bound N and valid diamonds $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$, $\Diamond(A, \alpha_2, \alpha_3; B, \beta_2, \beta_3)$, if it satisfies all the following condition:*

1. $\alpha_1, \alpha_2, \alpha_3 \in \{4..2N\} \backslash \{B\}$
2. *Each of the pairs* $(A, B), (\alpha_1, \beta_1), (\alpha_2, \beta_2), (\alpha_3, \beta_3)$ *has the same parity*
3. $A - B, \alpha_1 - \beta_1, \alpha_2 - \beta_2, \alpha_3 - \beta_3 \geq 4$

then there exists a substructure like the diagram shown below:



Proof. Combining the theorem 7.1 and theorem 4.3 with $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$ and $\Diamond(A, \alpha_2, \alpha_3; B, \beta_2, \beta_3)$, we have $\frac{A^2-B^2}{4} = \frac{\alpha_1^2-\beta_1^2}{4} = \frac{\alpha_2^2-\beta_2^2}{4} = \frac{\alpha_3^2-\beta_3^2}{4}$, so $E(\frac{A}{2}, \frac{B}{2}), E(\frac{\alpha_1}{2}, \frac{\beta_1}{2}), E(\frac{\alpha_2}{2}, \frac{\beta_2}{2}), E(\frac{\alpha_3}{2}, \frac{\beta_3}{2}) \in \mathscr{E}_N$. $\square$

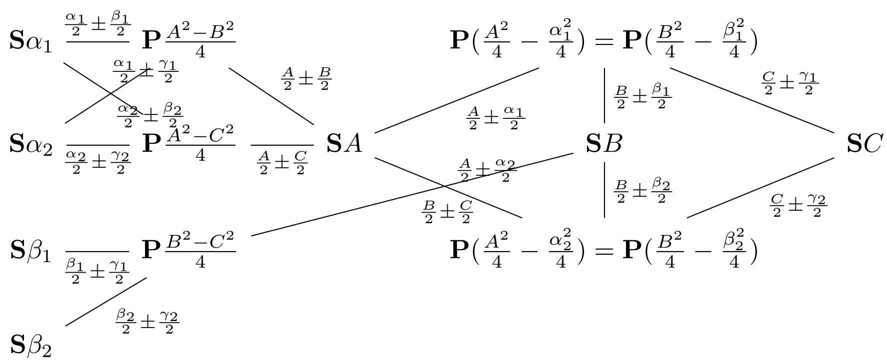
Theorem 7.3. *Given game upper bound N and valid diamonds $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$, $\Diamond(B, \alpha_1, \alpha_2; C, \gamma_1, \gamma_2)$, if it satisfies all the following condition:*

1. $\alpha_1, \alpha_2 \in \{4..2N\} \backslash \{B, C\}$

2. *Each of the triples* $(A, B, C), (\alpha_1, \beta_1, \gamma_1), (\alpha_2, \beta_2, \gamma_2)$ *has the same parity*

3. $A - B, B - C, \alpha_1 - \beta_1, \alpha_2 - \beta_2, \beta_1 - \gamma_1, \beta_2 - \gamma_2 \geq 4$

then there exists a substructure like the diagram shown below:



Proof. Note that $A - C = (A - B) + (B - C) \geq 8 > 4$. Similarly, we have $\alpha_1 - \gamma_1, \alpha_2 - \gamma_2 > 4$ Apply theorem 7.1 to $\Diamond(A, \alpha_1, \alpha_2; B, \beta_1, \beta_2)$, $\Diamond(B, \alpha_1, \alpha_2; C, \beta_1, \beta_2)$ and $\Diamond(A, \alpha_1, \alpha_2; C, \beta_1, \beta_2)$, then it gives the desired substructure. $\square$

8 Appendix

Here is the code to plot the graph induced by a Sum and Product Game, stat some data of diamond substructures and sketch the scatterplot of diamond substructures as in the figures.

```
import math
import re

import numpy as np
import networkx as nx
from itertools import combinations_with_replacement, combinations, chain
import matplotlib.pyplot as plt
```

```python
from typing import *
from time import time
from sklearn.linear_model import LogisticRegression
from scipy.optimize import curve_fit


def timing(f):
    def timing_f(*args, **kwargs):
        start_time = time()
        result = f(*args, **kwargs)
        print("--- %s seconds ---" % (time() - start_time))
        return result

    return timing_f


class SPG:
    def __init__(self, graph: nx.Graph, edge_labels: dict):
        self.graph: nx.Graph = graph
        self.edge_labels: dict = edge_labels
        self.colors = ['pink' if node.startswith('S') else 'lightblue' for node in self.graph]

    @staticmethod
    def by_max(maximum: int, highlights_cond=None):
        graph = nx.Graph()
        edge_labels = {}
        for i, j in combinations_with_replacement(range(2, maximum + 1), 2):
            edge = (f'S{i + j}', f'P{i * j}')
            edge_labels[edge] = f'{i}:{j}'
            graph.add_edge(*edge, _=(i, j))
        G = SPG(graph, edge_labels)
        if highlights_cond is not None:
            for index, node in enumerate(G.graph):
                if highlights_cond(node):
```

```
                    G.colors[index] = 'blue' if re.match(r'P(\d*)', node) is not None else 'red'
        return G

    def copy(self) -> 'SPG':
        return SPG(self.graph.copy(), self.edge_labels.copy())

    def plot(self, num: int = 1, figsize: Tuple[int, int] = (6, 6), options: Dict = None):
        if options is None:
            options = {}
        current_options = {
            'node_color': self.colors,
            'node_size': 600,
            'font_size': 10,
            'width': .8,
            'with_labels': True,
        }
        current_options.update(options)
        plt.figure(num, figsize)
        pos = nx.nx_agraph.graphviz_layout(self.graph)
        nx.draw(self.graph, pos, **current_options)
        nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=self.edge_labels)

    def leaves(self) -> List:
        return [i for i in self.graph if self.graph.degree(i) <= 1]

    def rot(self) -> List:
        self.graph.remove_nodes_from(res := self.leaves())
        self.colors = ['pink' if node.startswith('S') else 'lightblue' for node in self.graph]
        self.edge_labels = dict([(key, self.edge_labels[key]) for key in self.edge_labels.keys() if
                                 key[0] not in res and key[1] not in res])
        print(self.edge_labels)
```

```
        return res

    def succ(self) -> 'SPG':
        ret = self.graph.copy()
        ret.remove_nodes_from(self.leaves())
        return SPG(ret, self.edge_labels)

    def game_life(self) -> int:
        count = 0
        g = self.copy()
        while g.rot():
            count += 1
        return count


def SPG_stats(maximum: int, modes=()) -> List:
    G = SPG.by_max(maximum)
    initial_G = G.copy()
    last_leaves = []
    dropped_nodes = iter(())
    life_count = 0
    while True:
        G.graph.remove_nodes_from(last_leaves)
        if leaves := G.leaves():
            last_leaves = leaves
        else:
            res = []
            if 'game_life' in modes:
                res.append(life_count)
            if 'last_leaves' in modes:
                res.append(last_leaves)
            if 'chains' in modes:
                chains = initial_G.copy()
                chains.graph.remove_nodes_from(G.graph)
                res.append(chains)
```

```
            if 'initial_graph' in modes:
                res.append(initial_G)
            if 'terminal_graph' in modes:
                res.append(G)
            return res
        life_count += 1


def longest_chain(maximum: int) -> SPG:
    last_leaves, chains = SPG_stats(maximum, ('last_leaves', 'chains'))
    nodes = nx.node_connected_component(chains.graph, last_leaves[0])
    chains.graph.remove_nodes_from([n for n in chains.graph if n not in nodes])
    return chains


def n_step_chains(maximum: int, step_upper: int, step_lower: int = 1) -> SPG:
    G = SPG.by_max(maximum)
    subgraph_nodes = set()
    for s_node in G.graph:
        if (s_re := re.match(r'S(\d*)', s_node)) is not None and (s_node_val := int(
    s_re.group(1))):
            p_nodes = G.graph.neighbors(s_node)
            p_exist = False
            for p_node_0, p_node_1 in combinations(p_nodes, 2):
                diff = abs(int(re.match(r'P(\d*)', p_node_0).group(1)) - int(re.
    match(r'P(\d*)', p_node_1).group(1)))
                print(diff)
                if step_upper >= diff >= step_lower:
                    subgraph_nodes.add(p_node_0)
                    subgraph_nodes.add(p_node_1)
                    p_exist = True
            if p_exist:
                subgraph_nodes.add(s_node)

    return SPG(G.graph.subgraph(subgraph_nodes), {})
```

```python


def strict_n_step_chains(maximum: int, step_upper: int, step_lower: int = 1) -> SPG:
    G = SPG.by_max(maximum)
    subgraph_nodes = set()
    for s_node in G.graph:
        if (s_re := re.match(r'S(\d*)', s_node)) is not None and (s_node_val := int(
    s_re.group(1))):
            p_nodes = G.graph.neighbors(s_node)

            p_node_vals = [int(re.match(r'P(\d*)', p).group(1)) for p in p_nodes]
            if (diff := abs(max(p_node_vals) - min(p_node_vals))) <= step_upper and
    diff >= step_lower:
                subgraph_nodes.add(s_node)
                subgraph_nodes.update(G.graph.neighbors(s_node))

    return SPG(G.graph.subgraph(subgraph_nodes), {})


def deg_n_prod_nodes(maximum: int, deg: int, sum_lower_bound: int):
    G = SPG.by_max(maximum)
    counter = 0
    subgraph_edges = set()
    for node in G.graph:
        if G.graph.degree(node) == deg and re.match(r'P(\d*)', node) is not None:
            for neighbor in G.graph.neighbors(node):
                if neighbor < sum_lower_bound:
                    break
            else:
                counter += 1
                for edge in G.graph.edges(node):
                    subgraph_edges.add(edge)
    return counter, SPG(G.graph.edge_subgraph(subgraph_edges), {})


```

```
def deg_n_highlight(maximum: int, deg: int):
    G = SPG.by_max(maximum)
    for index, node in enumerate(G.graph):
        if G.graph.degree(node) == deg:
            G.colors[index] = 'blue' if re.match(r'P(\d*)', node) is not None else 'red'
    return G


def embed_graph(maximum: int, maximum_sub: int):
    G = SPG.by_max(maximum)
    G_sub = SPG.by_max(maximum_sub)
    for index, node in enumerate(G.graph):
        if node not in G_sub.graph:
            G.colors[index] = 'blue' if re.match(r'P(\d*)', node) is not None else 'red'
    return G


def unlooped_lifetime(node: str):
    maximum = 4
    while True:
        g = SPG.by_max(maximum)
        while res := g.rot():
            if node in res:
                return maximum
        maximum += 1


def substructrue_diamond(maximum: int, sum_lower_bound: int):
    G = SPG.by_max(maximum)
    diamond_nodes = set()
    diamond_leading_nodes = set()
    diamonds = []
    for index, node in enumerate(G.graph):
```

```
        if node not in diamond_leading_nodes and (sum_matched := re.match(r'S(\d*)', node)):
            diamond_leading_nodes.add(node)
            if int(sum_matched[1]) >= sum_lower_bound:
                for neighbor1, neighbor2 in combinations(G.graph.neighbors(node), 2):
                    for nn in G.graph.neighbors(neighbor1):
                        if nn not in diamond_leading_nodes and nn in G.graph.neighbors(neighbor2) \
                                and int(re.match(r'S(\d*)', nn)[1]) >= sum_lower_bound:
                            diamond_nodes.update([node, nn, neighbor1, neighbor2])
                            diamonds.append([node, nn, neighbor1, neighbor2])
    return diamonds, diamond_nodes


def substructrue_chains(maximum: int):
    G = SPG.by_max(maximum)
    chain_xs = []
    while leaves := G.rot():
        chain_xs += leaves
    return chain_xs

class Diamond:
    def __init__(self, A, a1, a2, B, b1, b2):
        self.A = A
        self.a1 = a1
        self.a2 = a2
        self.B = B
        self.b1 = b1
        self.b2 = b2
        self.P1 = (A**2 - a1**2) // 4  # = (B**2 - b1**2) // 4
        self.P2 = (A**2 - a2**2) // 4  # = (B**2 - b2**2) // 4

    @staticmethod
```

```python
    def from_SSPP(S1, S2, P1, P2):
        return Diamond(A=S1,
                       a1=math.sqrt(S1 ** 2 - 4 * P1),
                       a2=math.sqrt(S1 ** 2 - 4 * P2),
                       B=S2,
                       b1=math.sqrt(S2 ** 2 - 4 * P1),
                       b2=math.sqrt(S2 ** 2 - 4 * P2),
                       )

    @staticmethod
    def from_4nodes(dia):
        return Diamond.from_SSPP(*[int(node[1:]) for node in dia])

    def __repr__(self):
        return f"(A={self.A}, a1={self.a1}, a2={self.a2}, B={self.B}, b1={self.b1}, b2={self.b2}, P1={self.P1}, P2={self.P2})"

def induced_sum_diamond_diagram(maximum: int):
    graph = nx.Graph()
    for s_edge in [[int(p[1:]) for p in ps[0:2]] for ps in substructrue_diamond(maximum, 2)[0]]:
        graph.add_edge(*s_edge)
    return graph


def diamond_upper_curve(a_plus_b: int):
    ...

# {s = x + y, d = x - y}, Tsd represents this kind of replacement
def diamond_sum_nodes_Tsd(maximum: int):
    xs_s = [[int(p[1:]) for p in ps[0:2]] for ps in substructrue_diamond(maximum, 0)[0]]
    xs_Tsd_s = [[p[0] + p[1], p[1] - p[0]] for p in xs_s]
    return xs_Tsd_s

```

```
def estimate_epsilon(maximum: int):
    # to find out the constant epsilon such that the line "sqrt(D) = sqrt(S) + epsilon" bounds the scatter
    xs_s = [[int(p[1:]) for p in ps[0:2]] for ps in substructrue_diamond(maximum, 0)[0]]
    epsilon = maximum
    result_p = ()
    for dia in substructrue_diamond(maximum, 0)[0]:
        # print(epsilon)
        s0 = int(dia[0][1:])
        s1 = int(dia[1][1:])
        if (next_epsilon := math.sqrt(int(s0 + s1) - math.sqrt(s0 + s1))) < epsilon:
            epsilon = next_epsilon
            result_p = dia
    return epsilon, Diamond.from_4nodes(result_p)


def diamond_sum_nodes_AB(maximum: int):
    return [[int(p[1:]) for p in ps[0:2]] for ps in substructrue_diamond(maximum, 0)[0]]


def diamond_sum_nodes_sqTsd(maximum: int):
    xs_s = [[int(p[1:]) for p in ps[0:2]] for ps in substructrue_diamond(maximum, 0)[0]]
    xs_sqTsd_s = [[math.sqrt(p[0] + p[1]), math.sqrt(p[1] - p[0])] for p in xs_s]
    return xs_sqTsd_s


def diamond_scatter_AB(maximum: int, color: str = 'red'):
    xas_AB_s = np.array(diamond_sum_nodes_AB(maximum))
    ps_AB_s = xas_AB_s.transpose()
    plt.scatter(ps_AB_s[0], ps_AB_s[1], c=color)

```

```

def diamond_scatter_Tsd(maximum: int, color: str = 'red'):
    x = np.linspace(0, 4*maximum, 100)
    y = np.array([(-math.sqrt(xi) + 2*math.sqrt(maximum)) ** 2 for xi in x])
    xas_Tsd_s = np.array(diamond_sum_nodes_Tsd(maximum))
    ps_Tsd_s = xas_Tsd_s.transpose()
    plt.scatter(ps_Tsd_s[0], ps_Tsd_s[1], c=color)
    plt.plot(x,y)


def diamond_scatter_sqTsd(maximum: int, color: str = 'red'):
    xas_sqTsd_s = np.array(diamond_sum_nodes_sqTsd(maximum))
    ps_sqTsd_s = xas_sqTsd_s.transpose()
    plt.scatter(ps_sqTsd_s[0], ps_sqTsd_s[1], c=color)


def diamond_sqS_vs_sqD_scatter(maximum: int, scatter_color: str = 'red'):
    sqS = np.linspace(0, 2 * math.sqrt(maximum), 100)
    sqD = np.array([-sqS_i + 2 * math.sqrt(maximum) for sqS_i in sqS])
    diamond_scatter_sqTsd(maximum, color=scatter_color)
    plt.plot(sqS, sqD)


def diamond_scatter_sd(maximum: int, color: str = 'red'):
    xas_Tsd_s = np.array(diamond_sum_nodes_Tsd(maximum))
    ps_Tsd_s = xas_Tsd_s.transpose()
    plt.scatter(ps_Tsd_s[0], ps_Tsd_s[0] * ps_Tsd_s[1] / maximum, c=color)


def diamond_scatter_Tsd_parity(maximum: int, AmB_bound: int):
    xs_Tsd_s = [p for p in diamond_sum_nodes_Tsd(maximum) if p[1] <= AmB_bound]
    ps_Tsd_s_same = np.array([p for p in xs_Tsd_s if p[0] % 2 == 0]).transpose()
    ps_Tsd_s_diff = np.array([p for p in xs_Tsd_s if p[0] % 2 == 1]).transpose()
    plt.scatter(ps_Tsd_s_same[0], ps_Tsd_s_same[1], c='red')
    plt.scatter(ps_Tsd_s_diff[0], ps_Tsd_s_diff[1], c='blue')
```

```python
def diamond_scatter_Tsd_parity_sqrt(maximum: int, AmB_bound: int):
    xs_Tsd_s = [p for p in diamond_sum_nodes_Tsd(maximum) if p[1] <= AmB_bound]
    ps_Tsd_s_same = np.array([[math.sqrt(p[0]), math.sqrt(p[1])] for p in xs_Tsd_s if p[0] % 2 == 0]).transpose()
    ps_Tsd_s_diff = np.array([[math.sqrt(p[0]), math.sqrt(p[1])] for p in xs_Tsd_s if p[0] % 2 == 1]).transpose()
    plt.scatter(ps_Tsd_s_same[0], ps_Tsd_s_same[1], c='red')
    plt.scatter(ps_Tsd_s_diff[0], ps_Tsd_s_diff[1], c='blue')


def density_of_tails(maximum: int, AmB_bound: int):
    all_diamonds = [p for p in substructrue_diamond(maximum, 0)[0] if int(p[0][1:]) - int(p[1][1:]) <= AmB_bound]
    diamond_fishes = [p for p in substructrue_diamond(maximum, 0)[0] if (int(p[0][1:]) - int(p[1][1:])) % 2 == 0]
    return len(diamond_fishes) / len(all_diamonds)
    # xs = [p for p in diamond_sum_nodes_Tsd(maximum) if p[1] <= AmB_bound]
    # xs_same = [p for p in xs if p[0] % 2 == 0]
    # return len(xs_same)/len(xs)


def diamond_minimize_B(maximum: int):
    return max(*diamond_sum_nodes_Tsd(maximum), key=lambda p: p[1])


def eight_factors(ApB: int, AmB: int):
    ...


def life_time_of_sum_node(maximum: int):
    G = SPG.by_max(maximum)
    life_sums = [0 for _ in range(2 * maximum + 1)]
    current_turn = 0
```

```python
    while leaves := G.rot():
        for leaf in leaves:
            if sum_matched := re.match(r'S(\d*)', leaf):
                print(leaf)
                life_sums[int(sum_matched[1])] = current_turn
        current_turn += 1
    for loop_node in G.graph:
        if sum_matched := re.match(r'S(\d*)', loop_node):
            print(loop_node, "!")
            life_sums[int(sum_matched[1])] = -1
    return life_sums


def minimized_N_to_make_sum_node_immortal(maximum_N: int):
    minN_of_sums = [0 for _ in range(2 * maximum_N + 1)]
    visited_sums = set()
    for maximum in range(maximum_N):
        G = SPG.by_max(maximum)
        while G.rot():
            pass
        for loop_node in G.graph:
            if loop_node not in visited_sums \
                    and (sum_matched := re.match(r'S(\d*)', loop_node)):
                visited_sums.add(loop_node)
                minN_of_sums[int(sum_matched[1])] = maximum - ...
    return minN_of_sums


@timing
def main():
    # ==================================
    N = 200
    Epsilon = math.sqrt(30) - 2

    diamond_scatter_AB(250, color='blue')
```

```python
    diamond_scatter_AB(200, color='green')
    diamond_scatter_AB(150, color='yellow')
    diamond_scatter_AB(100, color='orange')

    # ===================================

    # ===================================
    # N = 200
    # Epsilon = math.sqrt(30) - 2
    # x = np.linspace(0, 4*N, 100)
    # y = np.array([(-math.sqrt(xi) + 2*math.sqrt(N)) ** 2 for xi in x])
    # y_left = np.array([(math.sqrt(xi) - Epsilon) ** 2 for xi in x])

    # diamond_scatter_Tsd(500, color='violet')
    # diamond_scatter_Tsd(250, color='blue')
    # diamond_scatter_Tsd(200, color='green')
    # diamond_scatter_Tsd(150, color='yellow')
    # diamond_scatter_Tsd(100, color='orange')
    # plt.plot(x, y)
    # plt.plot(x, y_left)
    # ===================================
    # print(estimate_epsilon(200))      # 3.4641016151377544
    # print(estimate_epsilon(300))     # 3.464101615137535
    #  N=1000 -> 4.9520474982524485
    # ===================================
    # N = 200
    # Epsilon = 3.464101615137535  # min(sqrt(A+B) - sqrt(A-B))
    # sqS = np.linspace(0, 2 * math.sqrt(N), 100)
    # sqD_left = sqS - Epsilon
    # plt.plot(sqS, sqD_left)
    # diamond_sqS_vs_sqD_scatter(250, scatter_color='blue')
    # diamond_sqS_vs_sqD_scatter(200, scatter_color='green')
    # diamond_sqS_vs_sqD_scatter(150, scatter_color='yellow')
    # diamond_sqS_vs_sqD_scatter(100, scatter_color='orange')

```

```
main()
plt.show()
```