Algorithmic Strategies for Ramsey-Theoretic Games on Complete Graphs

Shivum Arora
Poolesville High School

Andy Wu
Poolesville High School

Mentor: Dr. William Gasarch
University of Maryland

August 20, 2025

Contents

Abstract

Ramsey theory is an area of mathematics focusing on the appearance of order and predictable patterns in a substructure broken down from a superstructure. K_n denotes a complete graph with n vertices, and for all 2-colorings of the edges of K_6 there is a mono K_3. In this paper, we explore Ramsey-style edge coloring games on complete graphs using different strategies. In these games, two players, one red and one blue, take turns coloring edges their color on a complete graph until one player creates a target monochromatic subgraph. An example is a game played on K_6 where both players aim to create a mono K_3 subgraph. This particular game is never a draw, and the winner of the game is whoever creates a mono K_3 of their color first. We first analyze hard-coded strategies without insight into the future, which involve winning immediately if possible, blocking the opponent, and making random moves. We also observed patterns in game configurations that guarantee a draw. We further analyzed algorithmic strategies, including greedy heuristics and search methods, such as Minimax and value iteration, as well as reinforcement learning approaches, such as Q-learning, and modeling the game as a Markov Decision Process. For $k = 3$, we found deterministic first player winning strategies, while for $k = 4$ we observed probabilistic advantages. Reinforcement learning consistently outperformed hardcoded strategies, showing the fact that new algorithmic pathways can be used to study Ramsey numbers and bounds rather than pure combinatorics.

1 Introduction

Ramsey's theorem states that for any two integers, s and t, there exists a number, $R(s,t)$, such that any complete graph with $R(s,t)$ vertices, where the edges of that graph are colored red or blue, will contain either a complete subgraph (clique) of r vertices with all red edges or a clique of s vertices with all blue edges. A mathematical way to express Ramsey's theorem is:

$$R(s,t) \leq R(s,t-1) + R(s-1,t) \quad \text{for } s,t > 2 \text{ [9]}$$

A notable example of Ramsey's theorem is $R(3,3) = 6$, which states that if you have a complete graph with 6 nodes, there will be either a red cyclic triangle or a blue cyclic triangle formed. By the theorem:

$$R(3,3) \leq R(3,2) + R(2,3) \Rightarrow 6 \leq 3 + 3$$

Ramsey Games Games based on Ramsey Theory, or Ramsey Games, are games involving players coloring edges on a graph. The players aim to form or avoid a target subgraph. In this paper, we examine Ramsey games played on complete graphs by two players, red and blue, taking turns coloring edges their color until one player forms a specified smaller complete graph of their color. Edges that have already been colored cannot be colored again, and the game ends if either player forms the target subgraph or there are no remaining uncolored edges.

Previous Research There's been extensive research to find Ramsey Numbers. Greenwood and Gleason established in 1955 that $R(4,4) = 18$ [6]. Ramsey Numbers of $R(5,5)$ and above are still unknown. $R(5,5)$ currently has a known lower bound of 43 [3] and a more recently found upper bound of 46 [1]. People have also attempted to use Neural Networks to find Ramsey-extremal graphs, which perform better than Random methods, although their objective differs from our objective of finding strategies for Ramsey games [5]. However, little research has been done to analyze patterns with various strategies, and the results that have been achieved by machine learning are still limited.

2 Notations and Definitions

Notations

1. $R(s,t)$: Denotes a Ramsey number with a red clique size s and a blue clique size t.

2. K_n: Denotes a complete graph with n vertices.

Definitions

1. A graph $G(V,E)$ is a set of points where $V(G)$ and $E(G)$ are the sets of vertices and edges in G. [2] An example of the set V and the set E in a graph is:
$$V = \{1,2,3,4\}$$
$$E = \{\{1,2\},\{1,3\},\{1,4\}\}$$

2. A complete graph K_n is a graph where every vertex is connected to every other vertex by an edge. An example of the set V and the set E in a complete graph is:
$$V = \{1,2,3\}$$
$$E = \{\{1,2\},\{1,3\},\{2,3\}\}$$
K_n has:
$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2} = \binom{n}{2}$$
edges. [2]

3. A subgraph is a smaller graph contained within a larger graph

4. A clique is a complete subgraph where every pair of k vertices is connected by an edge

5. A win is when a player colors edges such that there exists a monochromatic clique of size k in their color. Players trying to win must find a way to win before their opponent can

6. A draw is when all edges are colored and neither player has formed a monochromatic clique of size k.

3 Research

Hardcoded Simulation Results

For our first attempt at optimal play strategies, we ran simulations for various configurations of n (number of nodes in the complete graph K_n and k (clique size to form), from $n \leq 4 \leq 20$ for both $k = 3$ and $k = 4$. We made algorithms without insight to the future, like these:

- R: Purely random coloring.
- Win + R: If a move immediately wins the game, play it (e.g., on Red's turn there are three vertices x, y, z with edges $\{x, z\}$ and $\{y, z\}$ both red, then Red colors the edge $\{x, y\}$). Otherwise, play random.
- Block + R: If the opponent has a move that immediately wins for them, block it (e.g., on Red's turn there are three vertices x, y, z with edges $\{x, z\}$ and $\{y, z\}$ both blue, then Red colors the edge $\{x, y\}$). Otherwise, play random.
- Win + Block + R: Try to make a winning move, then try to make a blocking move, then play random.

We ran 1000 trials for each configuration of n, k, and player strategies, and evaluated our results. Our hardcoded agent simulations, where blue plays first and red next, led us to the following conjectures:

1. For both $k = 3$ and $k = 4$, Win + R is generally more effective than Block + R for both players.
2. For $k = 3$, if the opponent uses Win + Block + R, Block + R is more effective than Win + R for both players.
3. The maximum n that guarantees a draw increases by either 1 or 2 for each increase in k. A draw is guaranteed if there does not exist a coloring that contains a monochromatic k-clique. Furthermore, the increase in n follows a pattern of either

$$\{1, 2, 1, 2, 1, 1, 2\}$$

or

$$\{1, 2, 1, 1, 2\}$$

These simulations show that smaller values of k in larger graphs are better with aggressive and greedy strategies for Blue. As k increases for a fixed n, the number of draws also increases rapidly. Additionally, Blue always has an advantage over Red when using the same strategy, regardless of n and k. This was the base model we could use to compare for more advanced strategies like Q-learning and Value Iteration. However, there were other hard-coded strategies we started to think about, like forming chains with connected edges, and simulating which move would be the best after 1-2 more moves.

Win rates between strategies for $k = 3$

We organized the results of our simulations into tables that compare how the four strategies perform against each other. We compiled tables for $4 \leq n \leq 20$. We summarize these results in a table with the mean win rate for Blue across $6 \leq n \leq 20$, still with 1000 trials for each game configuration and strategy combination. We exclude $n = 4$ and $n = 5$ because a large portion of games end in a draw, which skews the data.

Table 1: Mean Win % for Blue across $6 \leq n \leq 20$ for each strategy combination with $k = 3$

Red \ Blue	Win+Block+R	Win+R	Block+R	R
Win+Block+R	66.6%	15.3%	26.9%	0.5%
Win+R	95.5%	65.1%	21.3%	4.2%
Block+R	89.5%	87.4%	61.8%	15.7%
R	99.9%	98.7%	88.8%	54.2%

The complete data for each game configuration can be found at this **Google Sheet Link**. The results from Table 1 led us to conclude, for both players and games with $k = 3$:

- Win + Block + R performs the best, followed by Win + R, Block + R, and R
- Win + R performs better than Block + R against three out of four strategies: Win + R, Block + R, and R.
- Block + R only performs better than Win + R if the opponent uses Win + Block + R.

From our tables, we created another table displaying the differences between Blue's win percentage using Win + R and Block + R to see how big the difference is between the strategies.

Table 2: Blue (Win + R) win% – (Block + R) win% across values of n, against different Red strategies for $k = 3$

	Red Strategy			
n	R	Win + R	Block + R	Win + Block + R
4	13.3	12.6	0.0	0.0
5	7.8	-3.2	6.4	-8.5
6	5.4	9.0	-12.3	-17.6
7	8.3	25.3	5.7	-10.8
8	12.0	33.1	12.6	-9.6
9	11.9	39.0	19.5	-10.9
10	10.5	42.5	25.4	-9.0
11	13.3	48.5	28.3	-10.9
12	11.6	46.9	31.4	-14.5
13	10.5	50.3	33.9	-10.5
14	9.9	51.2	29.8	-10.3
15	8.7	51.5	33.0	-12.1
16	10.2	50.7	33.2	-10.2
17	11.4	51.1	33.7	-11.8
18	9.4	51.3	35.1	-12.9
19	7.5	55.0	36.2	-10.6
20	7.7	51.4	38.8	-11.4

Table 2 displays the differences between Blue's win percentage using Win + R and Block + R for $4 \leq n \leq 20$ against all four of Red's strategies. A positive value indicates that Win + R performed better by that amount, while a negative value indicates Block + R performed better by that amount. Our simulations led us to the following conclusions:

- Win + R generally outperforms Block + R
- Win + R outperforms Block + R the most when Red uses Win + R, followed by Block + R and R.
- Win + R increasingly outperforms Block + R as n increases in the interval $4 \leq n \leq 20$ if Red uses either Win + R or Block + R, increasing from 12.57 to 51.40 and 0 to 38.80, respectively.
- Block + R always outperforms Win + R if red uses Win + Block + R, except for $n = 4$ where every game ends in a draw.

We used the same comparisons for Red, which had similar patterns.

Table 3: Red (Win + R) win% – (Block + R) win% across values of n, against different Blue strategies for $k = 3$

	Blue Strategy			
n	R	Win + R	Block + R	Win + Block + R
4	-7.0	-13.6	0.0	0.0
5	-7.2	-28.1	-1.5	0.4
6	4.2	-9.2	12.1	0.8
7	8.8	4.4	24.0	-0.9
8	13.9	12.3	32.8	-3.8
9	18.5	14.9	34.4	-4.9
10	15.4	20.8	37.9	-5.1
11	14.3	20.8	41.0	-6.9
12	15.1	25.5	41.0	-6.1
13	11.8	27.5	43.9	-8.1
14	11.8	24.0	45.4	-8.1
15	12.4	30.2	48.7	-6.4
16	11.3	31.8	49.3	-6.4
17	7.6	32.1	49.5	-7.9
18	9.4	33.2	49.4	-10.4
19	9.3	32.7	51.5	-6.4
20	7.8	34.5	47.1	-8.6

Table 3 displays the differences between Red's win percentage using Win + R and Block + R for $4 \leq n \leq 20$ against all four of Blue's strategies. Our simulations led us to similar conclusions for red:

- Win + R generally outperforms Block + R for $n \geq 6$ if Blue is using R, Win + R, or Block + R.
- For boards where $n < 6$, Block + R is more effective than Win + R
- Win + R outperforms Block + R the most when Blue uses Block + R, followed by Win + R and R.
- Win + R increasingly outperforms Block + R as n increases in the interval $4 \leq n \leq 20$ if Red uses either Win + R or Block + R, increasing from -13.64 to 34.50 and 0 to 47.10, respectively.
- Block + R always achieves a higher win rate than Win + R for $n \geq 7$ if Blue uses Win + Block + R.

Evidently, there are a few differences between the conclusions we reached for Blue and Red:

- For Blue, Win + R outperforms Block + R the most when the opponent uses Win + R, while it's the other way around for Red.
- On small boards with $n < 6$, Block + R performs better than Win + R for Red while Win + R still achieves better results for Blue

These differences may be due to Blue going first, thereby assuming an offensive position, while Red goes second, assuming a defensive position.

Comparison of Win + R and Block + R for $k = 4$

We obtained simulation results for $6 \leq n \leq 20$ and compiled them into tables. We excluded $6 \leq k \leq 9$ because too many draws would skew the data. The results are summarized in this table with mean win rates for blue:

Table 4: Mean Win % for Blue across $10 \leq n \leq 20$ for each strategy combination with $k = 4$

Red \ Blue	Win+Block+R	Win+R	Block+R	R
Win+Block+R	54.4%	19.8%	7.2%	0.7%
Win+R	84.1%	52.0%	30.7%	8.7%
Block+R	93.8%	73.6%	50.4%	10.0%
R	99.4%	73.4%	91.9%	52.2%

Table 4 led us to conclude, for games with $k = 4$:

- Win + Block + R performs the best, followed by Win + R, Block + R, and R
- Win + R performs better than Block + R against all strategies

We compared the Win + R and Block + R strategies in the same way for $k = 4$, finding some patterns that differed from $k = 3$.

Table 5: Blue (Win + R) win% – (Block + R) win% across values of n, against different Red strategies for $k = 4$

	Red Strategy			
n	R	Win + R	Block + R	Win + Block + R
6	0.6	1.7	-0.4	0.0
7	8.4	0.3	1.5	1.7
8	16.3	6.8	4.1	4.1
9	10.5	-5.4	12.7	4.6
10	0.9	-7.9	23.1	5.9
11	-2.8	-2.7	22.1	3.4
12	-0.5	7.6	21.6	6.7
13	1.1	12.5	19.8	9.5
14	1.7	20.2	17.9	11.6
15	2.8	20.9	19.4	10.5
16	1.3	21.1	24.0	13.4
17	-1.0	24.2	26.2	14.8
18	1.4	26.5	24.7	15.1
19	2.3	26.7	27.7	13.8
20	4.3	32.0	27.1	17.6

The results from Table 5 led us to conclude:

- Win + R generally outperforms Block + R regardless of Red's strategy
- For $n < 14$, Win + R outperforms Block + R the most when Red uses Block + R, with Block + R sometimes achieving a higher win rate than Win + R from $9 \leq n \leq 12$ if Red uses R or Win + R.
- For $n \geq 14$, Win + R outperforms Block + R most if Red uses Win + R or Block + R, followed by Win + Block + R and R.

Once again, we ran the same comparisons for Red.

Table 6: Red (Win + R) win% – (Block + R) win% across values of n, against different Red strategies for $k = 4$

	Blue Strategy			
n	R	Win + R	Block + R	Win + Block + R
6	2.0	1.0	0.0	0.0
7	7.0	5.7	0.4	0.6
8	5.8	-2.0	4.7	1.4
9	6.1	-8.1	8.4	3.0
10	2.7	-11.2	17.4	1.6
11	-4.0	-4.1	22.5	4.9
12	-2.4	7.2	21.6	5.4
13	-1.3	13.2	16.5	7.1
14	0.9	17.3	21.2	9.5
15	0.4	20.9	22.4	10.4
16	3.2	24.8	21.9	9.7
17	1.8	25.8	23.8	12.7
18	2.8	27.6	29.4	10.4
19	3.0	29.8	28.8	12.1
20	3.5	27.6	32.5	12.8

The conclusions we reached for Red mirror the conclusions for Blue. Both have a cluster where Block + R outperforms Win + R in the same area, with overlap at $11 \leq n \leq 12$ if Blue uses R and $9 \leq n \leq 11$ for Win + R. From our simulations for $k = 3$ and $k = 4$, we determined that Win + R is more effective than Block + R for these board configurations. Block + R only performs better for $k = 3$ and the opponent uses Win + Block + R.

Maximum n where a draw is guaranteed for a given k

To determine the maximum n that guarantees a draw for a given k, we tested values of $n \geq k$, increasing n by 1 until a draw is possible. A draw is guaranteed if neither player can win. That is, there aren't enough vertices and edges for a player to win, no matter how they play. To identify whether a draw is possible, we simulated a game with heuristics to create a best-case scenario for Blue.

- Out of n nodes, Blue selects the first k of those nodes. Blue can only color edges involving selected nodes unless none are available.
- Red can only color edges that involve at least one node not selected by Blue unless none are available
- If Red colors in an edge where both nodes are selected by Blue, the game is a guaranteed draw. Otherwise, if all edges involving only selected nodes are colored, the game is not a guaranteed draw.

For instance, if $k = 4$ and $n = 5$, blue selects the first k nodes $\{0, 1, 2, 3\}$ and can only color edges where both nodes are selected, meaning any edges involving node 4 are avoided. On the other hand, red can only color edges that include node 4.

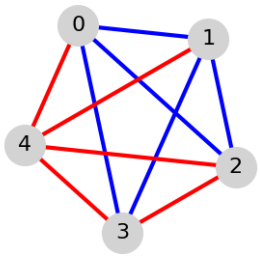

Figure 1: Simulated game with best-case scenario for blue with $k = 4$ and $n = 5$

During the simulation shown in Figure 1, Blue only colors edges with nodes 0, 1, 2, and 3. Red includes node 4 in every colored edge. However, K_4 has 6 edges, meaning Blue requires 6 moves to win. Red only has 4 edges that don't interfere with Blue, and is eventually forced to color edges $\{1, 4\}$ and $\{2, 4\}$. Thus, Blue can't create a monochromatic K_4, and the game is always a draw.

During the simulation shown in Figure 2, the same heuristics are used. Because there's an extra node, Red has more than six edges that don't

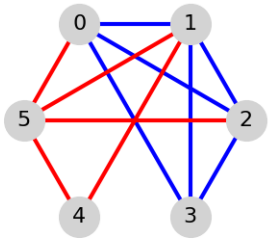

Figure 2: Simulated game with best-case scenario for blue with $k = 4$ and $n = 6$

interfere with Blue, and Blue can create a monochromatic K_4. Thus, this game is not a guaranteed draw.

Because only one simulation was needed for each n, k pair, we were able to simulate games with much higher values of n and k.

We analyzed how the maximum n to guarantee a draw, denoted $n(k)$, grows with k. To analyze the progression of $n(k)$, we also consider the differences $\Delta n = n(k) - n(k-1)$. We observed that Δn follows a repeating pattern of either $1, 2, 1, 2, 1, 1, 2$ or just $1, 2, 1, 1, 2$.

k	$n(k)$	Δn
2	2	–
3	3	1
4	5	2
5	6	1
6	8	2
7	9	1
8	10	1
9	12	2

(a) Pattern $1, 2, 1, 2, 1, 1, 2$

k	$n(k)$	Δn
23	32	–
24	33	1
25	35	2
26	36	1
27	37	1
28	39	2

(b) Pattern $1, 2, 1, 1, 2$

Table 7: Excerpts of $n(k)$ and Δn illustrating two distinct difference patterns.

These patterns persisted until $k = 105$, with the full table available in **the linked spreadsheet**. Whether this continues passed $k = 105$ and what causes these patterns are still unknown.

Using Minimax to find optimal strategies

After using Minimax algorithms to find optimal strategies, we found a four-move winning strategy for blue for games where $k = 3$ and $n > 4$. We used a tree depth of 2.

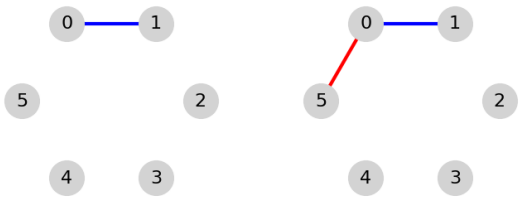

Figure 3: Blue's first move can be anything. Red's first move connects to Blue's edge, but it can also be disconnected.

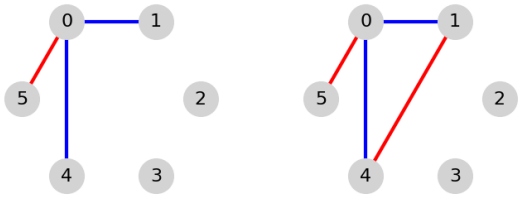

Figure 4: If the first two edges are connected, blue connects the vertex contained in both edges (0) with another vertex (2, 3, 4, or 5). If they're separate, either vertex of blue's first edge (0 or 1) works. Red is then forced to block.

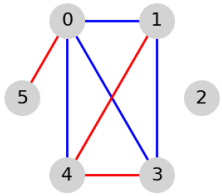

Figure 6: Blue colors the other edge and wins the game

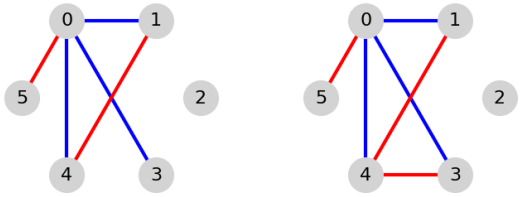

Figure 5: Both of blue's edges share a vertex (0). Blue connects it with an unused vertex (2 or 3), creating a fork with two ways to win. Red can only block one of these.

Since red cannot win if blue plays perfectly, red's strategy relies on blue making mistakes during one of these moves. Therefore, there is a way for blue to win every time if they start first, where $k = 3$ and $n > 4$. We were unable to use Minimax for $k > 3$ because of limitations with computational ability.

Modeling as an MDP + Reinforcement Learning Frameworks

After using Minimax to find the four-move winning strategy, we wanted to take it further and find a more general strategy to all board configurations.

To formalize our Ramsey game, we modeled it as a finite Markov Decision Process (MDP). [7] This gave us a structured way to apply reinforcement learning algorithms and analyze convergence properties. We decided that there were two effective ways to create reinforcement learning models: Value Iteration and Q Learning. Value iteration is a dynamic programming algorithm that solves the MDP exactly for small graphs by using the known transition structure, while Q Learning is a model-free reinforcement learning algorithm that approximates the optimal policy when the state space becomes too large for exact methods. For Value Iteration, we used the Bellman optimality update equation for the state value function:

$$V_{k+1}(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, V_k(s') \right]$$

Here, s represents the current board state, a is a possible edge placement, $R(s,a)$ is the immediate reward value, and $P(s' \mid s,a)$ is the transition probability to the next state. This model computes the optimal strategy in a small board configuration repeatedly until it converges.

For Q Learning, we used the Bellman optimality equation, but for action values:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

Here,

- $Q^*(s,a)$ is the optimal Q-value, eg, the maximum expected long-term reward from taking action a in state s and then playing optimally.
- $R(s,a)$ is the immediate reward obtained from taking action a in state s.
- γ is the discount factor ($0 \leq \gamma < 1$), which makes future rewards less significant than immediate ones.
- $P(s' \mid s,a)$ is the probability of transitioning to state s' given action a in state s.
- $\max_{a'} Q^*(s',a')$ represents the best possible future Q-value obtainable from the next state s'.

Intuitively, this equation states that the value of taking an action is the immediate reward plus the best possible discounted value expected in the future.

State Space. Each state is defined by the coloring of the edges of K_n. We represented the board state as an array of edges mapped to player moves, like

$$[0, -1, 1, 0, 0, 1, -1, 0, 0, 0]$$

, where 1 indicates that the edge is colored blue, -1 if it's red, and 0 if it's not colored. Since each edge can be uncolored, red, or blue, the total number of states is $3^{\binom{n}{2}}$. For example, when $n = 5$, there are $\binom{5}{2} = 10$ edges, and at most $3^{10} = 59049$ possible states.

Actions. At any state, the acting player chooses an uncolored edge and colors it with their color. Thus, while the action space decreases linearly, the number of possible board configurations shrinks exponentially as moves are made

Transitions. The game is deterministic. Once an action is taken, the next state is determined by the new edge coloring.

Reward Function. We used a sparse reward scheme:

$$R(s,a) = \begin{cases} +1 & \text{if acting player completes a } k \text{ length clique,} \\ -1 & \text{if the opponent wins,} \\ 0 & \text{otherwise.} \end{cases}$$

Results with Machine Learning

After creating Q-Learning Models with Epsilon-Greedy algorithms [8] and Value Iteration models [10], we wanted to test them against each other and against our hardcoded strategies. We trained the Q Learning model with 5000 training/evaluation epochs and then tested each model against each of the others. For value iteration, our configurations were that gamma=0.95, and the depth limit=4. For Q-Learning, epsilon=1.0, epsilon min=0.05, and epsilon decay=0.995. We did 200 games for each configuration with Q Learning and Value Iteration, and 1000 for the 4x4 Hardcoded Heuristics.

Table 8: Win % for Blue ($n = 6$, $k = 3$) across strategies against Red strategies

Red \ Blue	Win+Block+R	Win+R	Block+R	R	Value Iteration	Q-Learning
Win+Block+R	87.9%	17.1%	38.2%	2.2%	100.0%	100.0%
Win+R	98.7%	73.5%	62.8%	17.6%	100.0%	85.0%
Block+R	99.1%	64.7%	76.7%	22.4%	100.0%	100.0%
R	99.6%	94.8%	90.5%	61.7%	100.0%	96.5%
Value Iteration	100.0%	100.0%	100.0%	93.5%	100.0%	100.0%
Q-Learning	100.0%	99.5%	97.0%	53.0%	100.0%	100.0%

Table 9: Win % for Blue ($n = 6$, $k = 4$) across strategies against Red strategies

Red \ Blue	Win+Block+R	Win+R	Block+R	R	Value Iteration	Q-Learning
Win+Block+R	74.4%	29.7%	36.0%	9.9%	96.0%	35.5%
Win+R	93.4%	66.4%	72.1%	28.1%	98.0%	37.0%
Block+R	96.1%	65.1%	69.4%	37.0%	96.0%	77.0%
R	98.7%	90.9%	86.9%	64.1%	99.0%	66.5%
Value Iteration	100.0%	100.0%	100.0%	96.5%	100.0%	100.0%
Q-Learning	97.5%	96.5%	78.5%	70.0%	100.0%	100.0%

Table 10: Win % for Blue ($n = 7$, $k = 3$) across strategies against Red strategies

Red \ Blue	Win+Block+R	Win+R	Block+R	R	Value Iteration	Q-Learning
Win+Block+R	75.8%	16.1%	29.8%	1.4%	100.0%	93.5%
Win+R	98.0%	71.5%	46.7%	9.2%	100.0%	47.0%
Block+R	97.4%	72.4%	70.4%	19.5%	100.0%	100.0%
R	99.6%	96.9%	87.6%	57.8%	100.0%	82.5%
Value Iteration	100.0%	100.0%	100.0%	99.0%	100.0%	100.0%
Q-Learning	100.0%	99.5%	99.0%	56.0%	100.0%	100.0%

Note: The Q-Learning agent is the same agent for both blue and red

Even though in these simulations value iteration performed the best out of any other strategy, it took much more time than the rest of the configurations to run, since the number of paths it had to explore grew exponentially. This left Q-learning as the only viable strategy in the end, since it performed well and took less time than Value-Iteration. The time complexity for Q-Learning also did not skyrocket as n or k increased. However, the time for Q-Learning still did increase significantly, though it grew slower than value iteration.

4 Analysis

Theoretical Connections to Ramsey Theory

The results we obtained share many similarities and connections with traditional Ramsey theory. Ramsey's theorem guarantees monochromatic cliques in large complete graphs, and our simulations and RL models are a way to see how quickly such cliques can be forced through a competitive game style. For example, since a monochromatic triangle must appear in any two player coloring of K_6, the ability of the first player to guarantee a win with fewer vertices demonstrates the fact that algorithmic strategies are a great way to shift traditional Ramsey theory from combinatorial proofs to explicit and algorithmic methods.

Our methods could be used to potentially uncover new forceful algorithms guaranteeing wins for much higher K_n and k clique sizes, which may offer useful ways to estimate lower bounds for higher Ramsey numbers, like $R(5,5)$, $R(6,6)$, etc.

How Algorithmic Strategies can be used to find lower bounds By analyzing the strategies we identified, we can find optimal forcing strategies in which the two agents prevent each other from winning and attempt to force a draw. If a draw is achievable for a given configuration, it shows that a clique cannot be formed. Consequently, this shows that the corresponding value of n does not serve as a valid lower bound for $R(k,k)$.

Extensions of hardcoded and RL models

Instead of using $R(s,t)$ where $s = t$, we could try making the two players have 2 different goals and $s \neq t$. This would let us find strategies for many more configurations, allowing us to potentially uncover lower bounds for other Ramsey numbers. Additionally, instead of using the same Q-Learning agent for both red and blue, an extension of this is to train two seperate agents. This way, both agents will develop their own strategy akin to their needs.

Instead of using a sparse reward scheme, where the rewards happen infrequently, making it hard for the agent to know if it was making a good or bad move, we could give it more informative reward schemes. For example, we could use these more elaborate reward schemes:

1. Partial progress toward a clique:
    - $+0.1$ for adding an edge that extends a potential k-clique in the agent's color.
    - -0.1 for adding an edge that helps the opponent form a k-clique.
2. Fork creations:
    - $+0.3$ if a move creates multiple ways to win (eg, two potential k-cliques that need only one more edge each).
3. Blocking opponent threats:
    - $+0.2$ if a move prevents the opponent from completing a k-clique in their next turn.

We could also implement Deep Q-Learning or Deep Reinforcement Learning to handle larger graphs where the state space is too large for regular Q-Learning [4]. In DQN, instead of storing Q-values in a table for every state-action pair, we use a neural network to approximate the Q-function. In deep reinforcement learning, we use a deep neural network or a convolutional neural network that takes in the game frames, feeding into a policy network.

Limitations

Due to limited computing resources, we were unable to run more than 1000 trials for each game configuration. We were also unable to run simulations for large graphs with $n > 20$ or $k > 4$. We also used Python, which limits the speed of our programs compared to other faster languages. We were also unable to run simulations in a reasonable amount of time with Value Iteration and Q-Learning for $n > 7$ and $k \geq 4$ due to our limited computing resources.

5 Conclusion

In this paper, we researched Ramsey edge coloring games on complete graphs using both hardcoded strategies and machine learning frameworks. Our experiments demonstrated that greedy algorithms can provide strong baselines, but reinforcement learning, MDP, and minimax algorithms can discover deeper and more effective strategies that outperform simple models. We found deterministic winning strategies for small (n, k) configurations and consistent advantages for Blue when starting first, aligning with traditional Ramsey theory.

6 Acknowledgements

We would like to greatly thank our mentor, Dr. William Gasarch, for his support, discussions, guidance, and feedback on our designs and progress. We would also like to thank our colleagues for helping us and sharing ideas.

References

[1] Vigleik Angeltveit and Brendan D. McKay. $r(5, 5) \leq 46$, 2024.

[2] Colin Barton. Ramsey theory and its applications. `https://www.whitman.edu/documents/Academics/Mathematics/2016/Barton.pdf`, 2016. Senior thesis, Whitman College.

[3] G. Exoo. A lower bound for $r(5, 5)$. *Journal of Graph Theory*, 13(1):1–5, 1989.

[4] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning, 2020.

[5] Amur Ghose, Amit Levi, and Yingxue Zhang. Graph neural networks for ramsey graphs, 2023. Accessed: 2025-08-25.

[6] R. E. Greenwood and A. M. Gleason. Combinatorial relations and chromatic graphs. *Canadian Journal of Mathematics*, 7:1–7, 1955.

[7] Emily Hawboldt. A machine learning approach to constructing ramsey graphs: Leads to the trahtenbrot-zykov problem. `https://ir.library.louisville.edu/cgi/viewcontent.cgi?article=5453&context=etd`, August 2023. Master's thesis, University of Louisville, Electronic Theses and Dissertations, ThinkIR: The University of Louisville's Institutional Repository.

[8] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019.

[9] F. P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, s2-30(1):264–286, 1930.

[10] Aviv Tamar, Sergey Levine, and Pieter Abbeel. Value iteration networks. *CoRR*, abs/1602.02867, 2016.