Applying Genetic Algorithms to Ramsey Theory

David White
Wesleyan University

November 2, 2009

Outline

1. Basic Graph Theory and graph coloring

Outline

Outline

1. Basic Graph Theory and graph coloring
2. Pigeonhole Principle
3. Definition and examples of Ramsey Numbers

Outline

1. Basic Graph Theory and graph coloring
2. Pigeonhole Principle
3. Definition and examples of Ramsey Numbers
4. Generalizing Ramsey numbers

Outline

1. Basic Graph Theory and graph coloring
2. Pigeonhole Principle
3. Definition and examples of Ramsey Numbers
4. Generalizing Ramsey numbers
5. Ramsey's Theorem

Outline

1. Basic Graph Theory and graph coloring
2. Pigeonhole Principle
3. Definition and examples of Ramsey Numbers
4. Generalizing Ramsey numbers
5. Ramsey's Theorem
6. Some bounds on Ramsey numbers

Outline

1. Basic Graph Theory and graph coloring
2. Pigeonhole Principle
3. Definition and examples of Ramsey Numbers
4. Generalizing Ramsey numbers
5. Ramsey's Theorem
6. Some bounds on Ramsey numbers
7. One further generalization

Outline

Outline

Outline

Outline

Outline

Basic Graph Theory

Definition

A ***graph*** *G is a pair (V, E) where V is a set of* ***vertices****, and E is a set of pairs of points (v_i, v_j) called* ***edges****.*

Basic Graph Theory

Definition

*A **graph** G is a pair (V, E) where V is a set of **vertices**, and E is a set of pairs of points (v_i, v_j) called **edges**.*

For us $|V|$ will always be finite.

Basic Graph Theory

Definition

*A **graph** G is a pair (V, E) where V is a set of **vertices**, and E is a set of pairs of points (v_i, v_j) called **edges**.*

For us $|V|$ will always be finite.

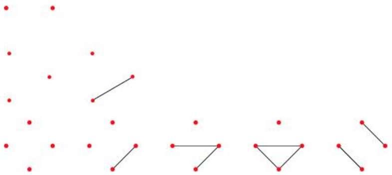


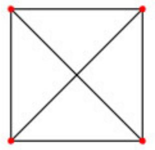
Figure: A graph

Complete Graphs

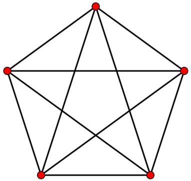
The complete graph on n vertices has n vertices and edges between all pairs of vertices.

Complete Graphs

The complete graph on n vertices has n vertices and edges between all pairs of vertices.



K_4



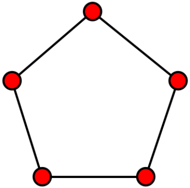
K_5

Cycle Graphs

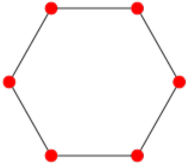
An n-cycle C_n has n vertices forming a regular n-gon and edges around the perimeter.

Cycle Graphs

An n-cycle C_n has n vertices forming a regular n-gon and edges around the perimeter.



C_5



C_6

Graph Colorings

Definition

*A **coloring** of a graph is an assignment of colors from a finite set* $\{c_1, \ldots, c_r\}$ *to the edges of the graph.*

Graph Colorings

Definition

*A **coloring** of a graph is an assignment of colors from a finite set* $\{c_1, \ldots, c_r\}$ *to the edges of the graph.*

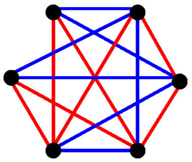


Figure: A 2-colored graph

Graph Colorings

Definition

*A **coloring** of a graph is an assignment of colors from a finite set $\{c_1, \ldots, c_r\}$ to the edges of the graph.*

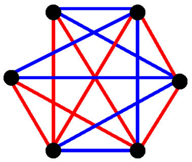


Figure: A 2-colored graph

We will be interested in colorings which avoid monochromatic subgraphs.

Graph Colorings

Definition

*A **coloring** of a graph is an assignment of colors from a finite set $\{c_1, \ldots, c_r\}$ to the edges of the graph.*

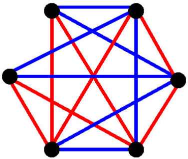


Figure: A 2-colored graph

We will be interested in colorings which avoid monochromatic subgraphs. This has no red triangle and no blue triangle,

Graph Colorings

Definition

*A **coloring** of a graph is an assignment of colors from a finite set* $\{c_1, \ldots, c_r\}$ *to the edges of the graph.*

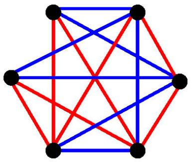


Figure: A 2-colored graph

We will be interested in colorings which avoid monochromatic subgraphs. This has no red triangle and no blue triangle, but last edge will force a monochromatic triangle.

Pigeonhole Principle

Proposition (Pigeonhole Principle)

1. *If you are placing $n+1$ pigeons into n holes, then some hole will end up containing at least two pigeons.*

Pigeonhole Principle

Proposition (Pigeonhole Principle)

1. *If you are placing $n+1$ pigeons into n holes, then some hole will end up containing at least two pigeons.*
2. *If you are placing $2n-1$ pigeons into 2 holes then some hole will end up containing at least n pigeons.*

Pigeonhole Principle

Proposition (Pigeonhole Principle)

1. *If you are placing $n + 1$ pigeons into n holes, then some hole will end up containing at least two pigeons.*
2. *If you are placing $2n - 1$ pigeons into 2 holes then some hole will end up containing at least n pigeons.*

If you have $2n - 1$ people at a party then at least n are of the same gender.

Pigeonhole Principle

Proposition (Pigeonhole Principle)

1. *If you are placing $n+1$ pigeons into n holes, then some hole will end up containing at least two pigeons.*
2. *If you are placing $2n-1$ pigeons into 2 holes then some hole will end up containing at least n pigeons.*

If you have $2n-1$ people at a party then at least n are of the same gender.

The notion of placing pigeons into 2 holes is exactly the same as 2-coloring the pigeons.

Ramsey Theory

Ramsey Theory generalizes the Pigeonhole Principle:

Ramsey Theory

Ramsey Theory generalizes the Pigeonhole Principle:

What is the minimum number of guests that must be invited so that at least n will know each other?

Ramsey Theory

Ramsey Theory generalizes the Pigeonhole Principle:

What is the minimum number of guests that must be invited so that at least n will know each other?

Definition

$R(n)$ is the smallest integer m such that in any 2-coloring of K_m there is a monochromatic K_n.

Ramsey Theory

Ramsey Theory generalizes the Pigeonhole Principle:

What is the minimum number of guests that must be invited so that at least n will know each other?

Definition

$R(n)$ is the smallest integer m such that in any 2-coloring of K_m there is a monochromatic K_n.

$R(1) = 1$ and $R(2) = 2$: an edge is a monochromatic edge.

Ramsey Theory

Ramsey Theory generalizes the Pigeonhole Principle:

What is the minimum number of guests that must be invited so that at least n will know each other?

Definition

$R(n)$ is the smallest integer m such that in any 2-coloring of K_m there is a monochromatic K_n.

$R(1) = 1$ and $R(2) = 2$: an edge is a monochromatic edge.

Generally: what is the smallest model guaranteed to contain the submodel I desire?

Theorem on Friends and Strangers

Theorem (Theorem on Friends and Strangers)

At any party with at least six people either three pairwise know each other or three are pairwise strangers.

Theorem on Friends and Strangers

Theorem (Theorem on Friends and Strangers)

At any party with at least six people either three pairwise know each other or three are pairwise strangers.
Equivalently: $R(3) = 6$

Theorem on Friends and Strangers

Theorem (Theorem on Friends and Strangers)

At any party with at least six people either three pairwise know each other or three are pairwise strangers.
Equivalently: $R(3) = 6$

Proof.

Here is a 2-coloring of K_5 with no monochromatic triangle.

Theorem on Friends and Strangers

Theorem (Theorem on Friends and Strangers)

At any party with at least six people either three pairwise know each other or three are pairwise strangers.
Equivalently: $R(3) = 6$

Proof.

Here is a 2-coloring of K_5 with no monochromatic triangle.

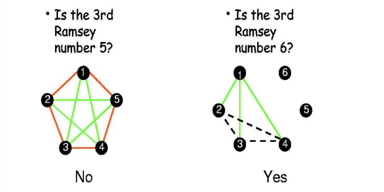


Figure: $R(3) \geq 6$

Theorem on Friends and Strangers

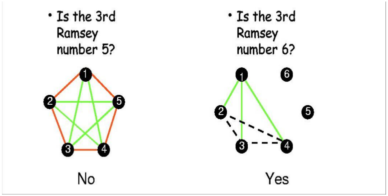


Figure: $R(3) \leq 6$

Proof.

A vertex has 5 edges touching it, so three of them are the same color, say green.

Theorem on Friends and Strangers

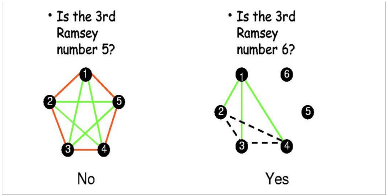


Figure: $R(3) \leq 6$

Proof.

A vertex has 5 edges touching it, so three of them are the same color, say green.

Consider the three vertices those edges connect to.

Theorem on Friends and Strangers

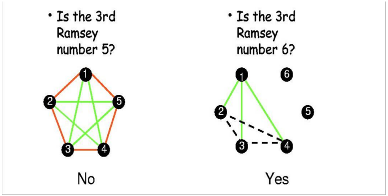


Figure: $R(3) \leq 6$

Proof.

A vertex has 5 edges touching it, so three of them are the same color, say green.
Consider the three vertices those edges connect to.
If any edge is green then we have a green triangle.

Theorem on Friends and Strangers

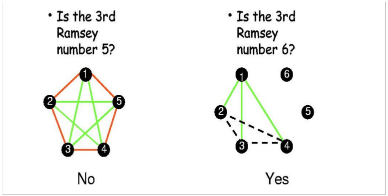


Figure: $R(3) \leq 6$

Proof.

A vertex has 5 edges touching it, so three of them are the same color, say green.
Consider the three vertices those edges connect to.
If any edge is green then we have a green triangle.
So all of these edges must be red, giving a red triangle. □

Known and Unknown Ramsey Numbers

David White
Wesleyan University

$R(3) = 6.$

Known and Unknown Ramsey Numbers

David White
Wesleyan University

$R(3) = 6$. $R(4) = 18$.

Known and Unknown Ramsey Numbers

$R(3) = 6$. $R(4) = 18$. To show $R(4) > 17$:

Known and Unknown Ramsey Numbers

$R(3) = 6$. $R(4) = 18$. To show $R(4) > 17$:

Find a 2-coloring of a K_{17} without mono K_4.

Known and Unknown Ramsey Numbers

$R(3) = 6$. $R(4) = 18$. To show $R(4) > 17$:

Find a 2-coloring of a K_{17} without mono K_4. Try coloring (i, j) red if $i - j$ is a square modulo 17 and blue otherwise.

Known and Unknown Ramsey Numbers

$R(3) = 6$. $R(4) = 18$. To show $R(4) > 17$:

Find a 2-coloring of a K_{17} without mono K_4. Try coloring (i, j) red if $i - j$ is a square modulo 17 and blue otherwise.

$K_4 \leq 18$: Any 2-coloring of K_{18} has a mono K_4.

Known and Unknown Ramsey Numbers

$R(3) = 6$. $R(4) = 18$. To show $R(4) > 17$:

Find a 2-coloring of a K_{17} without mono K_4. Try coloring (i, j) red if $i - j$ is a square modulo 17 and blue otherwise.

$K_4 \leq 18$: Any 2-coloring of K_{18} has a mono K_4.

$43 \leq R(5) \leq 49$ and $102 \leq R(6) \leq 165$ best bounds.

Known and Unknown Ramsey Numbers

$R(3) = 6$. $R(4) = 18$. To show $R(4) > 17$:

Find a 2-coloring of a K_{17} without mono K_4. Try coloring (i, j) red if $i - j$ is a square modulo 17 and blue otherwise.

$K_4 \leq 18$: Any 2-coloring of K_{18} has a mono K_4.

$43 \leq R(5) \leq 49$ and $102 \leq R(6) \leq 165$ best bounds.

To prove $R(5) \neq 43$ need to consider all 2-colorings of K_{43}.

Known and Unknown Ramsey Numbers

$R(3) = 6$. $R(4) = 18$. To show $R(4) > 17$:

Find a 2-coloring of a K_{17} without mono K_4. Try coloring (i, j) red if $i - j$ is a square modulo 17 and blue otherwise.

$K_4 \leq 18$: Any 2-coloring of K_{18} has a mono K_4.

$43 \leq R(5) \leq 49$ and $102 \leq R(6) \leq 165$ best bounds.

To prove $R(5) \neq 43$ need to consider all 2-colorings of K_{43}.

There are $\binom{43}{2}$ edges and each has two choices,

Known and Unknown Ramsey Numbers

$R(3) = 6$. $R(4) = 18$. To show $R(4) > 17$:

Find a 2-coloring of a K_{17} without mono K_4. Try coloring (i, j) red if $i - j$ is a square modulo 17 and blue otherwise.

$K_4 \leq 18$: Any 2-coloring of K_{18} has a mono K_4.

$43 \leq R(5) \leq 49$ and $102 \leq R(6) \leq 165$ best bounds.

To prove $R(5) \neq 43$ need to consider all 2-colorings of K_{43}.

There are $\binom{43}{2}$ edges and each has two choices, so number of colorings is $2^{\binom{43}{2}}$

Known and Unknown Ramsey Numbers

$R(3) = 6$. $R(4) = 18$. To show $R(4) > 17$:

Find a 2-coloring of a K_{17} without mono K_4. Try coloring (i, j) red if $i - j$ is a square modulo 17 and blue otherwise.

$K_4 \leq 18$: Any 2-coloring of K_{18} has a mono K_4.

$43 \leq R(5) \leq 49$ and $102 \leq R(6) \leq 165$ best bounds.

To prove $R(5) \neq 43$ need to consider all 2-colorings of K_{43}.

There are $\binom{43}{2}$ edges and each has two choices, so number of colorings is $2^{\binom{43}{2}} \approx 2^{1000}$.

Known and Unknown Ramsey Numbers

$R(3) = 6$. $R(4) = 18$. To show $R(4) > 17$:

Find a 2-coloring of a K_{17} without mono K_4. Try coloring (i, j) red if $i - j$ is a square modulo 17 and blue otherwise.

$K_4 \leq 18$: Any 2-coloring of K_{18} has a mono K_4.

$43 \leq R(5) \leq 49$ and $102 \leq R(6) \leq 165$ best bounds.

To prove $R(5) \neq 43$ need to consider all 2-colorings of K_{43}.

There are $\binom{43}{2}$ edges and each has two choices, so number of colorings is $2^{\binom{43}{2}} \approx 2^{1000}$. This is a HARD problem.

Lower Bound on $R(n)$

Proposition

$(n-1)^2 < R(n)$

Lower Bound on $R(n)$

David White
Wesleyan University

Proposition

$(n-1)^2 < R(n)$

Need to show there exists a coloring of $K_{(n-1)^2}$ without a monochromatic K_n. Have $4 < R(3)$ and $9 < R(4)$.

Lower Bound on $R(n)$

Proposition

$(n-1)^2 < R(n)$

Need to show there exists a coloring of $K_{(n-1)^2}$ without a monochromatic K_n. Have $4 < R(3)$ and $9 < R(4)$.

Partition $K_{(n-1)^2}$ into $n-1$ disjoint red K_{n-1}'s.

Lower Bound on $R(n)$

Proposition

$(n-1)^2 < R(n)$

Need to show there exists a coloring of $K_{(n-1)^2}$ without a monochromatic K_n. Have $4 < R(3)$ and $9 < R(4)$.

Partition $K_{(n-1)^2}$ into $n-1$ disjoint red K_{n-1}'s. Color all remaining edges blue.

Lower Bound on $R(n)$

Proposition

$(n-1)^2 < R(n)$

Need to show there exists a coloring of $K_{(n-1)^2}$ without a monochromatic K_n. Have $4 < R(3)$ and $9 < R(4)$.

Partition $K_{(n-1)^2}$ into $n-1$ disjoint red K_{n-1}'s. Color all remaining edges blue. Clearly no red K_n.

Lower Bound on $R(n)$

Proposition

$(n-1)^2 < R(n)$

Need to show there exists a coloring of $K_{(n-1)^2}$ without a monochromatic K_n. Have $4 < R(3)$ and $9 < R(4)$.

Partition $K_{(n-1)^2}$ into $n-1$ disjoint red K_{n-1}'s. Color all remaining edges blue. Clearly no red K_n. A blue K_n would have n vertices in $n-1$ groups

Lower Bound on $R(n)$

Proposition

$(n-1)^2 < R(n)$

Need to show there exists a coloring of $K_{(n-1)^2}$ without a monochromatic K_n. Have $4 < R(3)$ and $9 < R(4)$.

Partition $K_{(n-1)^2}$ into $n-1$ disjoint red K_{n-1}'s. Color all remaining edges blue. Clearly no red K_n. A blue K_n would have n vertices in $n-1$ groups so it needs 2 vertices in the same red group (Pigeonhole Principle),

Lower Bound on $R(n)$

Proposition

$(n-1)^2 < R(n)$

Need to show there exists a coloring of $K_{(n-1)^2}$ without a monochromatic K_n. Have $4 < R(3)$ and $9 < R(4)$.

Partition $K_{(n-1)^2}$ into $n-1$ disjoint red K_{n-1}'s. Color all remaining edges blue. Clearly no red K_n. A blue K_n would have n vertices in $n-1$ groups so it needs 2 vertices in the same red group (Pigeonhole Principle), so edge between them is red not blue!

Lower Bound on $R(n)$

Proposition

$(n-1)^2 < R(n)$

Need to show there exists a coloring of $K_{(n-1)^2}$ without a monochromatic K_n. Have $4 < R(3)$ and $9 < R(4)$.

Partition $K_{(n-1)^2}$ into $n-1$ disjoint red K_{n-1}'s. Color all remaining edges blue. Clearly no red K_n. A blue K_n would have n vertices in $n-1$ groups so it needs 2 vertices in the same red group (Pigeonhole Principle), so edge between them is red not blue!

Constructive methods like this can give polynomial lower bounds of any fixed degree, but nothing reaching c^n.

Upper Bound on $R(n)$

Proposition

$R(n) \leq 4^n$

Upper Bound on $R(n)$

Proposition

$R(n) \leq 4^n$

Need to show that in ANY 2-coloring of K_{4^n} there is a monochromatic K_n.

Upper Bound on $R(n)$

Proposition

$R(n) \leq 4^n$

Need to show that in ANY 2-coloring of K_{4^n} there is a monochromatic K_n.

This requires a proof, not an example.

Upper Bound on $R(n)$

Proposition

$R(n) \leq 4^n$

Need to show that in ANY 2-coloring of K_{4^n} there is a monochromatic K_n.

This requires a proof, not an example. There are MUCH better bounds and they use sophisticated mathematics.

Upper Bound on $R(n)$

Proposition

$R(n) \leq 4^n$

Need to show that in ANY 2-coloring of K_{4^n} there is a monochromatic K_n.

This requires a proof, not an example. There are MUCH better bounds and they use sophisticated mathematics.

There are automated theorem provers, but Ramsey Theory proofs need tricks not logic.

Upper Bound on $R(n)$

Proposition

$R(n) \leq 4^n$

Need to show that in ANY 2-coloring of K_{4^n} there is a monochromatic K_n.

This requires a proof, not an example. There are MUCH better bounds and they use sophisticated mathematics.

There are automated theorem provers, but Ramsey Theory proofs need tricks not logic. Computer science will likely not give better upper bounds than mathematics.

Upper Bound on $R(n)$

Proposition

$R(n) \leq 4^n$

Need to show that in ANY 2-coloring of K_{4^n} there is a monochromatic K_n.

This requires a proof, not an example. There are MUCH better bounds and they use sophisticated mathematics.

There are automated theorem provers, but Ramsey Theory proofs need tricks not logic. Computer science will likely not give better upper bounds than mathematics.

We will focus on constructing lower bound examples.

Generalizing Ramsey numbers

Definition (Off-Diagonal Ramsey Numbers)

$R(s,t)$ = *minimal* m *such that for any 2-coloring of the edges of* K_m

Generalizing Ramsey numbers

Definition (Off-Diagonal Ramsey Numbers)

$R(s,t)$ = *minimal* m *such that for any 2-coloring of the edges of* K_m *there is a red* K_s *or a blue* K_t*.*

Generalizing Ramsey numbers

Definition (Off-Diagonal Ramsey Numbers)

$R(s,t)$ = *minimal* m *such that for any 2-coloring of the edges of* K_m *there is a red* K_s *or a blue* K_t. $R(n) = R(n,n)$

Generalizing Ramsey numbers

Definition (Off-Diagonal Ramsey Numbers)

$R(s,t)$ = *minimal* m *such that for any 2-coloring of the edges of* K_m *there is a red* K_s *or a blue* K_t. $R(n) = R(n,n)$

$R(2,s) = s$ for all $s \geq 2$:

Generalizing Ramsey numbers

Definition (Off-Diagonal Ramsey Numbers)

$R(s,t)$ = minimal m such that for any 2-coloring of the edges of K_m there is a red K_s or a blue K_t. $R(n) = R(n,n)$

$R(2,s) = s$ for all $s \geq 2$: either blue K_s or red edge

Generalizing Ramsey numbers

Definition (Off-Diagonal Ramsey Numbers)

$R(s,t)$ = minimal m such that for any 2-coloring of the edges of K_m there is a red K_s or a blue K_t. $R(n) = R(n,n)$

$R(2,s) = s$ for all $s \geq 2$: either blue K_s or red edge

$R(3,4) \leq 9$: any K_9 has red K_3 or blue K_4.

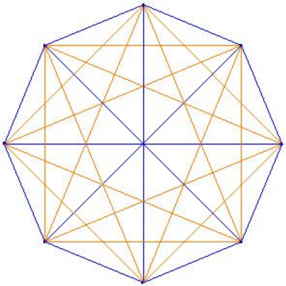
Generalizing Ramsey numbers

Definition (Off-Diagonal Ramsey Numbers)

$R(s,t)$ = minimal m such that for any 2-coloring of the edges of K_m there is a red K_s or a blue K_t. $R(n) = R(n,n)$

$R(2,s) = s$ for all $s \geq 2$: either blue K_s or red edge

$R(3,4) \leq 9$: any K_9 has red K_3 or blue K_4. $R(3,4) > 8$:



A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$,

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$.

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$.

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, so Ramsey numbers exist

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$. Any vertex x in any 2-coloring of K_n has degree $n - 1 = n_1 + n_2 - 1$.

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$. Any vertex x in any 2-coloring of K_n has degree $n-1 = n_1 + n_2 - 1$. There are either n_1 red or n_2 blue edges coming out of x (pigeonhole principle), switching red and blue if necessary.

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$. Any vertex x in any 2-coloring of K_n has degree $n-1 = n_1 + n_2 - 1$. There are either n_1 red or n_2 blue edges coming out of x (pigeonhole principle), switching red and blue if necessary. Assume the first case.

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$. Any vertex x in any 2-coloring of K_n has degree $n-1 = n_1 + n_2 - 1$. There are either n_1 red or n_2 blue edges coming out of x (pigeonhole principle), switching red and blue if necessary. Assume the first case.

Red neighbors form a K_{n_1}, so this graph either has blue K_t or red K_{s-1}.

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$. Any vertex x in any 2-coloring of K_n has degree $n-1 = n_1 + n_2 - 1$. There are either n_1 red or n_2 blue edges coming out of x (pigeonhole principle), switching red and blue if necessary. Assume the first case.

Red neighbors form a K_{n_1}, so this graph either has blue K_t or red K_{s-1}. With x this makes a red K_s.

A Useful Proposition

Proposition

$R(s,t) \le R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$. Any vertex x in any 2-coloring of K_n has degree $n-1 = n_1 + n_2 - 1$. There are either n_1 red or n_2 blue edges coming out of x (pigeonhole principle), switching red and blue if necessary. Assume the first case.

Red neighbors form a K_{n_1}, so this graph either has blue K_t or red K_{s-1}. With x this makes a red K_s. 2nd case similar.

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$. Any vertex x in any 2-coloring of K_n has degree $n-1 = n_1 + n_2 - 1$. There are either n_1 red or n_2 blue edges coming out of x (pigeonhole principle), switching red and blue if necessary. Assume the first case.

Red neighbors form a K_{n_1}, so this graph either has blue K_t or red K_{s-1}. With x this makes a red K_s. 2nd case similar.

$R(4,4) \leq R(3,4) + R(4,3) = 9 + 9 = 18$. Sharp

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$. Any vertex x in any 2-coloring of K_n has degree $n-1 = n_1 + n_2 - 1$. There are either n_1 red or n_2 blue edges coming out of x (pigeonhole principle), switching red and blue if necessary. Assume the first case.

Red neighbors form a K_{n_1}, so this graph either has blue K_t or red K_{s-1}. With x this makes a red K_s. 2nd case similar.

$R(4,4) \leq R(3,4) + R(4,3) = 9 + 9 = 18$. Sharp
$R(3,5) \leq R(2,5) + R(3,4) = 5 + 9 = 14$. Sharp

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$. Any vertex x in any 2-coloring of K_n has degree $n-1 = n_1 + n_2 - 1$. There are either n_1 red or n_2 blue edges coming out of x (pigeonhole principle), switching red and blue if necessary. Assume the first case.

Red neighbors form a K_{n_1}, so this graph either has blue K_t or red K_{s-1}. With x this makes a red K_s. 2nd case similar.

$R(4,4) \leq R(3,4) + R(4,3) = 9 + 9 = 18$. Sharp
$R(3,5) \leq R(2,5) + R(3,4) = 5 + 9 = 14$. Sharp
$R(3,4) \leq R(3,3) + R(2,4) = 10$.

A Useful Proposition

Proposition

$R(s,t) \leq R(s-1,t) + R(s,t-1)$, *so Ramsey numbers exist*

$R(s,t) = R(t,s)$. $n_1 = R(s-1,t)$, $n_2 = R(s,t-1)$, $n = n_1 + n_2$. Any vertex x in any 2-coloring of K_n has degree $n-1 = n_1 + n_2 - 1$. There are either n_1 red or n_2 blue edges coming out of x (pigeonhole principle), switching red and blue if necessary. Assume the first case.

Red neighbors form a K_{n_1}, so this graph either has blue K_t or red K_{s-1}. With x this makes a red K_s. 2nd case similar.

$R(4,4) \leq R(3,4) + R(4,3) = 9 + 9 = 18$. Sharp
$R(3,5) \leq R(2,5) + R(3,4) = 5 + 9 = 14$. Sharp
$R(3,4) \leq R(3,3) + R(2,4) = 10$. NOT sharp!

Generalizing Ramsey numbers

Theorem (Ramsey's Theorem)

Given integers $n_1, \ldots, n_r$ *there is a number* $m = R(n_1, \ldots, n_r)$ *such that for any* r*-coloring of the edges of* K_m

Generalizing Ramsey numbers

Theorem (Ramsey's Theorem)

Given integers $n_1, \ldots, n_r$ *there is a number* $m = R(n_1, \ldots, n_r)$ *such that for any* r*-coloring of the edges of* K_m *for some* i *there is a* K_{n_i} *monochromatic in color* i*.*

Generalizing Ramsey numbers

Theorem (Ramsey's Theorem)

Given integers $n_1, \ldots, n_r$ *there is a number* $m = R(n_1, \ldots, n_r)$ *such that for any* r*-coloring of the edges of* K_m *for some* i *there is a* K_{n_i} *monochromatic in color* i*.*

Induction proof:

$R(n_1, \ldots, n_r) \leq R(n_1, \ldots, n_{r-2}, R(n_{r-1}, n_r))$

Generalizing Ramsey numbers

Theorem (Ramsey's Theorem)

Given integers $n_1, \ldots, n_r$ there is a number $m = R(n_1, \ldots, n_r)$ such that for any r-coloring of the edges of K_m for some i there is a K_{n_i} monochromatic in color i.

Induction proof:
$R(n_1, \ldots, n_r) \leq R(n_1, \ldots, n_{r-2}, R(n_{r-1}, n_r))$

$R(3, 3, 3) = 17.$

Generalizing Ramsey numbers

Theorem (Ramsey's Theorem)

Given integers $n_1, \ldots, n_r$ *there is a number* $m = R(n_1, \ldots, n_r)$ *such that for any* r*-coloring of the edges of* K_m *for some* i *there is a* K_{n_i} *monochromatic in color* i*.*

Induction proof:
$R(n_1, \ldots, n_r) \leq R(n_1, \ldots, n_{r-2}, R(n_{r-1}, n_r))$

$R(3,3,3) = 17$. Only non-trivial $R(n_1, \ldots, n_r)$ known.

Generalizing Ramsey numbers

Theorem (Ramsey's Theorem)

Given integers $n_1, \ldots, n_r$ there is a number $m = R(n_1, \ldots, n_r)$ such that for any r-coloring of the edges of K_m for some i there is a K_{n_i} monochromatic in color i.

Induction proof:
$R(n_1, \ldots, n_r) \leq R(n_1, \ldots, n_{r-2}, R(n_{r-1}, n_r))$

$R(3,3,3) = 17$. Only non-trivial $R(n_1, \ldots, n_r)$ known.

$R(s,t,2) = R(s,t)$ because need to avoid green edge.

Generalizing Ramsey numbers

Theorem (Ramsey's Theorem)

Given integers $n_1, \ldots, n_r$ there is a number $m = R(n_1, \ldots, n_r)$ such that for any r-coloring of the edges of K_m for some i there is a K_{n_i} monochromatic in color i.

Induction proof:
$R(n_1, \ldots, n_r) \leq R(n_1, \ldots, n_{r-2}, R(n_{r-1}, n_r))$

$R(3,3,3) = 17$. Only non-trivial $R(n_1, \ldots, n_r)$ known.

$R(s,t,2) = R(s,t)$ because need to avoid green edge.

$30 \leq R(3,3,4) \leq 31$ is next closest to being finished

Generalizing Ramsey numbers

Theorem (Ramsey's Theorem)

Given integers $n_1, \ldots, n_r$ there is a number $m = R(n_1, \ldots, n_r)$ such that for any r-coloring of the edges of K_m for some i there is a K_{n_i} monochromatic in color i.

Induction proof:
$R(n_1, \ldots, n_r) \leq R(n_1, \ldots, n_{r-2}, R(n_{r-1}, n_r))$

$R(3,3,3) = 17$. Only non-trivial $R(n_1, \ldots, n_r)$ known.

$R(s,t,2) = R(s,t)$ because need to avoid green edge.

$30 \leq R(3,3,4) \leq 31$ is next closest to being finished

$55 \leq R(3,4,4) \leq 79$

One further generalization

Definition

$R(G_1, \ldots, G_r)$ is the smallest m such that for any r-coloring of the edges of K_m

One further generalization

Definition

$R(G_1, \ldots, G_r)$ is the smallest m such that for any r-coloring of the edges of K_m for some i there is a monochromatic G_i in color i.

One further generalization

Definition

$R(G_1, \ldots, G_r)$ is the smallest m such that for any r-coloring of the edges of K_m for some i there is a monochromatic G_i in color i.

$R(K_3, K_3) = R(3) = 6$

One further generalization

Definition

$R(G_1, \ldots, G_r)$ is the smallest m such that for any r-coloring of the edges of K_m for some i there is a monochromatic G_i in color i.

$R(K_3, K_3) = R(3) = 6$ and $R(K_3, K_4) = R(3, 4) = 9$

One further generalization

Definition

$R(G_1, \ldots, G_r)$ is the smallest m such that for any r-coloring of the edges of K_m for some i there is a monochromatic G_i in color i.

$R(K_3, K_3) = R(3) = 6$ and $R(K_3, K_4) = R(3, 4) = 9$
$R(C_4, C_4, C_4) = 11$,

One further generalization

Definition

$R(G_1, \ldots, G_r)$ is the smallest m such that for any r-coloring of the edges of K_m for some i there is a monochromatic G_i in color i.

$R(K_3, K_3) = R(3) = 6$ and $R(K_3, K_4) = R(3, 4) = 9$
$R(C_4, C_4, C_4) = 11$, $R(C_4, C_4, K_3) = 12$,

One further generalization

Definition

$R(G_1, \ldots, G_r)$ is the smallest m such that for any r-coloring of the edges of K_m for some i there is a monochromatic G_i in color i.

$R(K_3, K_3) = R(3) = 6$ and $R(K_3, K_4) = R(3, 4) = 9$
$R(C_4, C_4, C_4) = 11$, $R(C_4, C_4, K_3) = 12$, $R(C_5, C_5, C_5) = 17$

One further generalization

Definition

$R(G_1, \ldots, G_r)$ is the smallest m such that for any r-coloring of the edges of K_m for some i there is a monochromatic G_i in color i.

$R(K_3, K_3) = R(3) = 6$ and $R(K_3, K_4) = R(3, 4) = 9$
$R(C_4, C_4, C_4) = 11$, $R(C_4, C_4, K_3) = 12$, $R(C_5, C_5, C_5) = 17$

Our K_8 coloring had a yellow C_4.

One further generalization

Definition

$R(G_1, \dots, G_r)$ is the smallest m such that for any r-coloring of the edges of K_m for some i there is a monochromatic G_i in color i.

$R(K_3, K_3) = R(3) = 6$ and $R(K_3, K_4) = R(3, 4) = 9$
$R(C_4, C_4, C_4) = 11$, $R(C_4, C_4, K_3) = 12$, $R(C_5, C_5, C_5) = 17$

Our K_8 coloring had a yellow C_4.

$R(G, H) \geq (\chi(G) - 1)(c(H) - 1) + 1$ for $\chi =$ chromatic number, $c =$ size of largest connected component

One further generalization

Definition

$R(G_1, \ldots, G_r)$ is the smallest m such that for any r-coloring of the edges of K_m for some i there is a monochromatic G_i in color i.

$R(K_3, K_3) = R(3) = 6$ and $R(K_3, K_4) = R(3, 4) = 9$
$R(C_4, C_4, C_4) = 11$, $R(C_4, C_4, K_3) = 12$, $R(C_5, C_5, C_5) = 17$

Our K_8 coloring had a yellow C_4.

$R(G, H) \geq (\chi(G) - 1)(c(H) - 1) + 1$ for $\chi =$ chromatic number, $c =$ size of largest connected component

$R(T_m, K_n) = (m - 1)(n - 1) + 1$ for any tree T_m

Examples

Early K_5 coloring shows $R(C_4, C_4) > 5$.

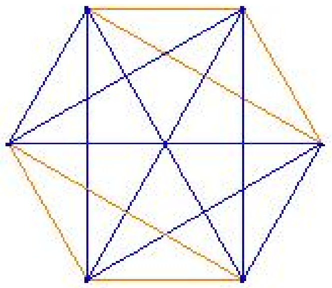
Examples

David White Wesleyan University

Early K_5 coloring shows $R(C_4, C_4) > 5$. $R(K_3, C_4) > 6$:

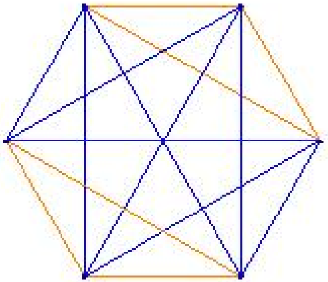
Examples

Early K_5 coloring shows $R(C_4, C_4) > 5$. $R(K_3, C_4) > 6$:



Examples

Early K_5 coloring shows $R(C_4, C_4) > 5$. $R(K_3, C_4) > 6$:



$$\begin{bmatrix} & y & y & b & b & b \\ y & & y & b & b & b \\ y & y & & b & b & b \\ b & b & b & & y & y \\ b & b & b & y & & y \\ b & b & b & y & y & \end{bmatrix}$$

EC Encoding

The **adjacency matrix** is symmetric,

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

Shardul Rao's thesis (1997) attempted to use EC to construct lower bounds.

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

Shardul Rao's thesis (1997) attempted to use EC to construct lower bounds. His encoding uses the above encoding as well as a permutation to represent the order in which edges are colored.

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

Shardul Rao's thesis (1997) attempted to use EC to construct lower bounds. His encoding uses the above encoding as well as a permutation to represent the order in which edges are colored. Contains more information, but is it useful?

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

Shardul Rao's thesis (1997) attempted to use EC to construct lower bounds. His encoding uses the above encoding as well as a permutation to represent the order in which edges are colored. Contains more information, but is it useful? Coloring order doesn't matter.

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

Shardul Rao's thesis (1997) attempted to use EC to construct lower bounds. His encoding uses the above encoding as well as a permutation to represent the order in which edges are colored. Contains more information, but is it useful? Coloring order doesn't matter.
For this permutation encoding we need an order-based GA.

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

Shardul Rao's thesis (1997) attempted to use EC to construct lower bounds. His encoding uses the above encoding as well as a permutation to represent the order in which edges are colored. Contains more information, but is it useful? Coloring order doesn't matter.
For this permutation encoding we need an order-based GA. Rao experimented with three crossovers:

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

Shardul Rao's thesis (1997) attempted to use EC to construct lower bounds. His encoding uses the above encoding as well as a permutation to represent the order in which edges are colored. Contains more information, but is it useful? Coloring order doesn't matter.
For this permutation encoding we need an order-based GA.
Rao experimented with three crossovers:

- Partially matched cross-over

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

Shardul Rao's thesis (1997) attempted to use EC to construct lower bounds. His encoding uses the above encoding as well as a permutation to represent the order in which edges are colored. Contains more information, but is it useful? Coloring order doesn't matter.
For this permutation encoding we need an order-based GA. Rao experimented with three crossovers:

- Partially matched cross-over
- Order cross-over

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

Shardul Rao's thesis (1997) attempted to use EC to construct lower bounds. His encoding uses the above encoding as well as a permutation to represent the order in which edges are colored. Contains more information, but is it useful? Coloring order doesn't matter.
For this permutation encoding we need an order-based GA. Rao experimented with three crossovers:

- Partially matched cross-over
- Order cross-over
- Cycle cross-over

EC Encoding

The **adjacency matrix** is symmetric, so we only need to store lower triangle and can use a single dimensional array.

Shardul Rao's thesis (1997) attempted to use EC to construct lower bounds. His encoding uses the above encoding as well as a permutation to represent the order in which edges are colored. Contains more information, but is it useful? Coloring order doesn't matter.
For this permutation encoding we need an order-based GA. Rao experimented with three crossovers:

- Partially matched cross-over
- Order cross-over
- Cycle cross-over

A lookup table is used to get values of i and j from given edge $e = (i, j)$.

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$.

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1.

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$.

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1.

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$.

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child,

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents.

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Example: $p_1 = (2\ 3\ 5\ 6\ 4\ 1\ 7\ 8)$ and $p_2(1\ 4\ 2\ 3\ 6\ 5\ 8\ 7)$

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Example: $p_1 = (2\ 3\ 5\ 6\ 4\ 1\ 7\ 8)$ and $p_2(1\ 4\ 2\ 3\ 6\ 5\ 8\ 7)$
$c = (2\ x\ x\ x\ x\ 1\ x\ x)$

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Example: $p_1 = (2\ 3\ 5\ 6\ 4\ 1\ 7\ 8)$ and $p_2(1\ 4\ 2\ 3\ 6\ 5\ 8\ 7)$
$c = (2\ x\ x\ x\ x\ 1\ x\ x) = (2\ x\ 5\ x\ x\ 1\ x\ x)$

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Example: $p_1 = (2\ 3\ 5\ 6\ 4\ 1\ 7\ 8)$ and $p_2(1\ 4\ 2\ 3\ 6\ 5\ 8\ 7)$
$c = (2\ x\ x\ x\ x\ 1\ x\ x) = (2\ x\ 5\ x\ x\ 1\ x\ x)$ and
$p_1[3] = 5 \Rightarrow p_2[3] = 2 = p_1[1]$, giving a cycle:

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Example: $p_1 = (2\ 3\ 5\ 6\ 4\ 1\ 7\ 8)$ and $p_2(1\ 4\ 2\ 3\ 6\ 5\ 8\ 7)$
$c = (2\ x\ x\ x\ x\ 1\ x\ x) = (2\ x\ 5\ x\ x\ 1\ x\ x)$ and
$p_1[3] = 5 \Rightarrow p_2[3] = 2 = p_1[1]$, giving a cycle:
$c = (2\ 4\ 5\ 3\ 6\ 1\ 7\ 8)$

Example: $p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ and $p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Example: $p_1 = (2\ 3\ 5\ 6\ 4\ 1\ 7\ 8)$ and $p_2(1\ 4\ 2\ 3\ 6\ 5\ 8\ 7)$
$c = (2\ x\ x\ x\ x\ 1\ x\ x) = (2\ x\ 5\ x\ x\ 1\ x\ x)$ and
$p_1[3] = 5 \Rightarrow p_2[3] = 2 = p_1[1]$, giving a cycle:
$c = (2\ 4\ 5\ 3\ 6\ 1\ 7\ 8)$

Example: $p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ and $p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$
$c = (1\ x\ x\ 4\ x\ x\ x\ x\ x)$

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Example: $p_1 = (2\ 3\ 5\ 6\ 4\ 1\ 7\ 8)$ and $p_2(1\ 4\ 2\ 3\ 6\ 5\ 8\ 7)$
$c = (2\ x\ x\ x\ x\ 1\ x\ x) = (2\ x\ 5\ x\ x\ 1\ x\ x)$ and
$p_1[3] = 5 \Rightarrow p_2[3] = 2 = p_1[1]$, giving a cycle:
$c = (2\ 4\ 5\ 3\ 6\ 1\ 7\ 8)$

Example: $p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ and $p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$
$c = (1\ x\ x\ 4\ x\ x\ x\ x\ x) = (1\ x\ 3\ 4\ x\ x\ x\ 8\ x)$

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Example: $p_1 = (2\ 3\ 5\ 6\ 4\ 1\ 7\ 8)$ and $p_2(1\ 4\ 2\ 3\ 6\ 5\ 8\ 7)$
$c = (2\ x\ x\ x\ x\ 1\ x\ x) = (2\ x\ 5\ x\ x\ 1\ x\ x)$ and
$p_1[3] = 5 \Rightarrow p_2[3] = 2 = p_1[1]$, giving a cycle:
$c = (2\ 4\ 5\ 3\ 6\ 1\ 7\ 8)$

Example: $p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ and $p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$
$c = (1\ x\ x\ 4\ x\ x\ x\ x\ x) = (1\ x\ 3\ 4\ x\ x\ x\ 8\ x)$
$= (1\ 2\ 3\ 4\ x\ x\ x\ 8\ x)$

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Example: $p_1 = (2\ 3\ 5\ 6\ 4\ 1\ 7\ 8)$ and $p_2(1\ 4\ 2\ 3\ 6\ 5\ 8\ 7)$
$c = (2\ x\ x\ x\ x\ 1\ x\ x) = (2\ x\ 5\ x\ x\ 1\ x\ x)$ and
$p_1[3] = 5 \Rightarrow p_2[3] = 2 = p_1[1]$, giving a cycle:
$c = (2\ 4\ 5\ 3\ 6\ 1\ 7\ 8)$

Example: $p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ and $p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$
$c = (1\ x\ x\ 4\ x\ x\ x\ x\ x) = (1\ x\ 3\ 4\ x\ x\ x\ 8\ x)$
$= (1\ 2\ 3\ 4\ x\ x\ x\ 8\ x)$ and $p_1[2] = 2 \Rightarrow p_2[2] = 1 = p_1[1]$,
giving a cycle:

Cycle Cross-over

Insists $c[i] = p_1[i]$ or $c[i] = p_2[i]$ for all i.

Put $c[1] = p_1[1]$. Find where $p_2[1]$ appears in p_1. Call this place i and put $c[i] = p_1[i]$. Find where $p_2[i]$ appears in p_1. Call this place j and put $c[j] = p_1[j]$. Continue till $p_2[x]$ appears in the child, i.e. when you've reached a cycle between the parents. Put $c[a] = p_2[a]$ for all remaining a.

Example: $p_1 = (2\ 3\ 5\ 6\ 4\ 1\ 7\ 8)$ and $p_2(1\ 4\ 2\ 3\ 6\ 5\ 8\ 7)$
$c = (2\ x\ x\ x\ x\ 1\ x\ x) = (2\ x\ 5\ x\ x\ 1\ x\ x)$ and
$p_1[3] = 5 \Rightarrow p_2[3] = 2 = p_1[1]$, giving a cycle:
$c = (2\ 4\ 5\ 3\ 6\ 1\ 7\ 8)$

Example: $p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ and $p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$
$c = (1\ x\ x\ 4\ x\ x\ x\ x\ x) = (1\ x\ 3\ 4\ x\ x\ x\ 8\ x)$
$= (1\ 2\ 3\ 4\ x\ x\ x\ 8\ x)$ and $p_1[2] = 2 \Rightarrow p_2[2] = 1 = p_1[1]$,
giving a cycle: $c = (1\ 2\ 3\ 4\ 7\ 6\ 9\ 8\ 5)$

Selection and Mutation

Tournament Selection:

Selection and Mutation

David White
Wesleyan University

Tournament Selection: run two tournaments and record winners and losers.

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers.

Selection and Mutation

David White
Wesleyan University

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers. Each generation only replaces the worst pair

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers. Each generation only replaces the worst pair (steady state).

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers. Each generation only replaces the worst pair (steady state).

Pros: slow convergence,

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers. Each generation only replaces the worst pair (steady state).

Pros: slow convergence, cheap to evaluate new fitness.

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers. Each generation only replaces the worst pair (steady state).

Pros: slow convergence, cheap to evaluate new fitness.
Cons: very little exploration,

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers. Each generation only replaces the worst pair (steady state).

Pros: slow convergence, cheap to evaluate new fitness.
Cons: very little exploration, VERY slow convergence,

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers. Each generation only replaces the worst pair (steady state).

Pros: slow convergence, cheap to evaluate new fitness.
Cons: very little exploration, VERY slow convergence, new parameter of tournament size

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers. Each generation only replaces the worst pair (steady state).

Pros: slow convergence, cheap to evaluate new fitness.
Cons: very little exploration, VERY slow convergence, new parameter of tournament size

Mutation is not explicitly described in the thesis.

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers. Each generation only replaces the worst pair (steady state).

Pros: slow convergence, cheap to evaluate new fitness.
Cons: very little exploration, VERY slow convergence, new parameter of tournament size

Mutation is not explicitly described in the thesis. We can assume it makes a small random change, say swapping an edge color.

Selection and Mutation

Tournament Selection: run two tournaments and record winners and losers. Tournament is filled randomly and all individuals are compared to get best and worst.

Children of winners replace the losers. Each generation only replaces the worst pair (steady state).

Pros: slow convergence, cheap to evaluate new fitness.
Cons: very little exploration, VERY slow convergence, new parameter of tournament size

Mutation is not explicitly described in the thesis. We can assume it makes a small random change, say swapping an edge color. Changing the order doesn't matter.

Selection Code

Selection Code

best_fitness = -9999
worst_fitness = 9999

Selection Code

best_fitness = -9999
worst_fitness = 9999

for $i = 0$ to tournament_size-1 do
  get a random j (no repeats allowed)

Selection Code

best_fitness = -9999
worst_fitness = 9999

for $i = 0$ to tournament_size-1 do
  get a random j (no repeats allowed)
  if fitness[j] > best_fitness then
    best_fitness = fitness[j]
    winner = j endif

Selection Code


```
best_fitness = -9999
worst_fitness = 9999

for i = 0 to tournament_size-1 do
   get a random j (no repeats allowed)
   if fitness[j] > best_fitness then
      best_fitness = fitness[j]
      winner = j endif
   if fitness[j] < worst_fitness then
      worst_fitness = fitness[j]
      loser = j endif
end
```

Selection Code


```
best_fitness = -9999
worst_fitness = 9999

for i = 0 to tournament_size-1 do
   get a random j (no repeats allowed)
   if fitness[j] > best_fitness then
     best_fitness = fitness[j]
     winner = j endif
   if fitness[j] < worst_fitness then
     worst_fitness = fitness[j]
     loser = j endif
end
```

Note the random filling of the tournament.

Selection Code


```
best_fitness = -9999
worst_fitness = 9999

for i = 0 to tournament_size-1 do
   get a random j (no repeats allowed)
   if fitness[j] > best_fitness then
     best_fitness = fitness[j]
     winner = j endif
   if fitness[j] < worst_fitness then
     worst_fitness = fitness[j]
     loser = j endif
end
```

Note the random filling of the tournament. Might be better to bias this towards getting some of the highest and some of the lowest fitness individuals.

Fitness Function

We wish to find coloring which has no monochromatic G_i in color i,

Fitness Function

We wish to find coloring which has no monochromatic G_i in color i, so we should lower the fitness by 1 each time there is a mono G_i in color i.

Fitness Function

We wish to find coloring which has no monochromatic G_i in color i, so we should lower the fitness by 1 each time there is a mono G_i in color i.

We should also lower the fitness by 1 if more colors are used than some fixed user parameter no_of_colors.

Fitness Function

We wish to find coloring which has no monochromatic G_i in color i, so we should lower the fitness by 1 each time there is a mono G_i in color i.

We should also lower the fitness by 1 if more colors are used than some fixed user parameter no_of_colors. This is all Rao's fitness function does.

Fitness Function

We wish to find coloring which has no monochromatic G_i in color i, so we should lower the fitness by 1 each time there is a mono G_i in color i.

We should also lower the fitness by 1 if more colors are used than some fixed user parameter no_of_colors. This is all Rao's fitness function does.

1. hero is a function which gets the individual with best fitness when called.

Fitness Function

We wish to find coloring which has no monochromatic G_i in color i, so we should lower the fitness by 1 each time there is a mono G_i in color i.

We should also lower the fitness by 1 if more colors are used than some fixed user parameter no_of_colors. This is all Rao's fitness function does.

1. hero is a function which gets the individual with best fitness when called.
2. The individual we're working with is p[who], an array.

Fitness Function

We wish to find coloring which has no monochromatic G_i in color i, so we should lower the fitness by 1 each time there is a mono G_i in color i.

We should also lower the fitness by 1 if more colors are used than some fixed user parameter no_of_colors. This is all Rao's fitness function does.

1. hero is a function which gets the individual with best fitness when called.
2. The individual we're working with is p[who], an array.

You could improve on Rao by making a smarter fitness function.

Fitness Function

We wish to find coloring which has no monochromatic G_i in color i, so we should lower the fitness by 1 each time there is a mono G_i in color i.

We should also lower the fitness by 1 if more colors are used than some fixed user parameter no_of_colors. This is all Rao's fitness function does.

1. hero is a function which gets the individual with best fitness when called.
2. The individual we're working with is p[who], an array.

You could improve on Rao by making a smarter fitness function. His does not take into account how badly a graph fails,

Fitness Function

We wish to find coloring which has no monochromatic G_i in color i, so we should lower the fitness by 1 each time there is a mono G_i in color i.

We should also lower the fitness by 1 if more colors are used than some fixed user parameter no_of_colors. This is all Rao's fitness function does.

1. hero is a function which gets the individual with best fitness when called.
2. The individual we're working with is p[who], an array.

You could improve on Rao by making a smarter fitness function. His does not take into account how badly a graph fails, and it only has the -1 rather than something more sophisticated.

Checking for mono G_a

How to check for a monochromatic triangle in color a:

Checking for mono G_a

How to check for a monochromatic triangle in color a:

for $k = 0$ to $N - 1$ do

Checking for mono G_a

How to check for a monochromatic triangle in color a:

for $k = 0$ to $N - 1$ do
  if i, j, k are distinct then

Checking for mono G_a

David White
Wesleyan University

How to check for a monochromatic triangle in color a:

for $k = 0$ to $N - 1$ do
  if i, j, k are distinct then
    if $a = \text{color}(i, j) = \text{color}(i, k) = \text{color}(j, k)$ then

Checking for mono G_a

How to check for a monochromatic triangle in color a:

for $k = 0$ to $N - 1$ do
  if i, j, k are distinct then
    if $a = \text{color}(i, j) = \text{color}(i, k) = \text{color}(j, k)$ then
      return false

Checking for mono G_a

How to check for a monochromatic triangle in color a:

for $k = 0$ to $N - 1$ do

  if i, j, k are distinct then

    if $a = \text{color}(i, j) = \text{color}(i, k) = \text{color}(j, k)$ then

      return false

    endif

  endif

Checking for mono G_a

How to check for a monochromatic triangle in color a:

for $k = 0$ to $N - 1$ do
  if i, j, k are distinct then
    if $a = \text{color}(i, j) = \text{color}(i, k) = \text{color}(j, k)$ then
      return false
    endif
  endif
end
return true

Checking for mono G_a

How to check for a monochromatic triangle in color a:

for $k = 0$ to $N - 1$ do
  if i, j, k are distinct then
    if $a = \text{color}(i, j) = \text{color}(i, k) = \text{color}(j, k)$ then
      return false
    endif
  endif
end
return true

Checking for other G_a is similar and easy.

Implementation

initialize population and set fitness[i] =fv(i) for all i
trial $= 0$

Implementation

initialize population and set fitness[i] =fv(i) for all i

trial = 0

while true

    trial++

Implementation

initialize population and set fitness[i] =fv(i) for all i
trial $= 0$
while true
  trial++
  if trial $\geq$ LOOPS then break endif

Implementation

```
initialize population and set fitness[i] =fv(i) for all i
trial = 0
while true
   trial++
   if trial ≥ LOOPS then break endif
   tournament(t_size, p_1, c_1)
   tournament(t_size, p_2, c_2)
```

Implementation

initialize population and set fitness$[i]$ =fv(i) for all i
trial = 0
while true
    trial++
    if trial $\geq$ LOOPS then break endif
    tournament(t_size, p_1, c_1)
    tournament(t_size, p_2, c_2)
    make_children(p_1, p_2, c_1, c_2)

Implementation

initialize population and set fitness[i] =fv(i) for all i
trial $= 0$
while true
    trial++
    if trial $\geq$ LOOPS then break endif
    tournament(t_size, p_1, c_1)
    tournament(t_size, p_2, c_2)
    make_children(p_1, p_2, c_1, c_2)
    if MUT_RATE $\geq$ 0.0 then mutate(c_1), mutate(c_2) endif

Implementation

initialize population and set fitness$[i]$ =fv(i) for all i
trial = 0
while true
  trial++
  if trial $\geq$ LOOPS then break endif
  tournament(t_size, p_1, c_1)
  tournament(t_size, p_2, c_2)
  make_children(p_1, p_2, c_1, c_2)
  if MUT_RATE $\geq$ 0.0 then mutate(c_1), mutate(c_2) endif
  fitness$[c_1]$ =fv(c_1), fitness$[c_2]$ =fv(c_2)

Implementation


```
initialize population and set fitness[i] =fv(i) for all i
trial = 0
while true
   trial++
   if trial ≥ LOOPS then break endif
   tournament(t_size, p_1, c_1)
   tournament(t_size, p_2, c_2)
   make_children(p_1, p_2, c_1, c_2)
   if MUT_RATE ≥ 0.0 then mutate(c_1), mutate(c_2) endif
   fitness[c_1] =fv(c_1), fitness[c_2] =fv(c_2)
   if hero≥ MAX_HERO then break endif
end
```

Implementation

initialize population and set fitness[i] =fv(i) for all i
trial = 0
while true
  trial++
  if trial $\geq$ LOOPS then break endif
  tournament(t_size, p_1, c_1)
  tournament(t_size, p_2, c_2)
  make_children(p_1, p_2, c_1, c_2)
  if MUT_RATE $\geq$ 0.0 then mutate(c_1), mutate(c_2) endif
  fitness[c_1] =fv(c_1), fitness[c_2] =fv(c_2)
  if hero$\geq$ MAX_HERO then break endif
end

The mutation code must be a typo because he later claims mutation rate did matter in experiments.

Implementation

initialize population and set fitness[i] =fv(i) for all i
trial = 0
while true
    trial++
    if trial $\geq$ LOOPS then break endif
    tournament(t_size, p_1, c_1)
    tournament(t_size, p_2, c_2)
    make_children(p_1, p_2, c_1, c_2)
    if MUT_RATE $\geq$ 0.0 then mutate(c_1), mutate(c_2) endif
    fitness[c_1] =fv(c_1), fitness[c_2] =fv(c_2)
    if hero$\geq$ MAX_HERO then break endif
end

The mutation code must be a typo because he later claims mutation rate did matter in experiments. But he does not show us the data or statistics so we can't be sure.

Statistics

There are five parameters that affect performance:

Statistics

David White
Wesleyan University

There are five parameters that affect performance:

- Population size

Statistics

There are five parameters that affect performance:

- Population size
- Mutation rate (unless there was no typo)

Statistics

There are five parameters that affect performance:

- Population size
- Mutation rate (unless there was no typo)
- Tournament size

Statistics

There are five parameters that affect performance:

- Population size
- Mutation rate (unless there was no typo)
- Tournament size
- Max number of loops allowed

Statistics

There are five parameters that affect performance:

- Population size
- Mutation rate (unless there was no typo)
- Tournament size
- Max number of loops allowed
- Seed, i.e. initial population

Statistics

There are five parameters that affect performance:

- Population size
- Mutation rate (unless there was no typo)
- Tournament size
- Max number of loops allowed
- Seed, i.e. initial population

Rao investigated these via 25 runs for each type of crossover and each seed value,

Statistics

There are five parameters that affect performance:

- Population size
- Mutation rate (unless there was no typo)
- Tournament size
- Max number of loops allowed
- Seed, i.e. initial population

Rao investigated these via 25 runs for each type of crossover and each seed value, but he shows no statistics whatsoever.

Statistics

There are five parameters that affect performance:

- Population size
- Mutation rate (unless there was no typo)
- Tournament size
- Max number of loops allowed
- Seed, i.e. initial population

Rao investigated these via 25 runs for each type of crossover and each seed value, but he shows no statistics whatsoever.

He couldn't see the effect of tournament size, population size, and mutation rate because the majority of solutions were found in the initial population,

Statistics

There are five parameters that affect performance:

- Population size
- Mutation rate (unless there was no typo)
- Tournament size
- Max number of loops allowed
- Seed, i.e. initial population

Rao investigated these via 25 runs for each type of crossover and each seed value, but he shows no statistics whatsoever.

He couldn't see the effect of tournament size, population size, and mutation rate because the majority of solutions were found in the initial population, BEFORE evolution!

Statistics

There are five parameters that affect performance:

- Population size
- Mutation rate (unless there was no typo)
- Tournament size
- Max number of loops allowed
- Seed, i.e. initial population

Rao investigated these via 25 runs for each type of crossover and each seed value, but he shows no statistics whatsoever.

He couldn't see the effect of tournament size, population size, and mutation rate because the majority of solutions were found in the initial population, BEFORE evolution! So this "EA" was really just brute force!

Results

Mutation rate on all good runs was 0.001,

Results

Mutation rate on all good runs was 0.001, the lowest possible.

Results

Mutation rate on all good runs was 0.001, the lowest possible. So again, algorithms is mostly exploitation.

Results

Mutation rate on all good runs was 0.001, the lowest possible. So again, algorithms is mostly exploitation.

Rao matched known bounds for $R(C_4, C_4, C_4)$, $R(C_4, C_4, K_3)$, $R(C_4, K_3, K_3)$, and $R(C_5, C_5, C_5)$.

Results

Mutation rate on all good runs was 0.001, the lowest possible. So again, algorithms is mostly exploitation.

Rao matched known bounds for $R(C_4, C_4, C_4)$, $R(C_4, C_4, K_3)$, $R(C_4, K_3, K_3)$, and $R(C_5, C_5, C_5)$. He used special seed values for these based on prior results.

Results

Mutation rate on all good runs was 0.001, the lowest possible. So again, algorithms is mostly exploitation.

Rao matched known bounds for $R(C_4, C_4, C_4)$, $R(C_4, C_4, K_3)$, $R(C_4, K_3, K_3)$, and $R(C_5, C_5, C_5)$. He used special seed values for these based on prior results.

He found new bounds on never before investigated numbers: $R(C_4, C_4, K_3, K_3) \geq 25$ and $R(C_5, C_4, K_3) \geq 13$.

Results

Mutation rate on all good runs was 0.001, the lowest possible. So again, algorithms is mostly exploitation.

Rao matched known bounds for $R(C_4, C_4, C_4)$, $R(C_4, C_4, K_3)$, $R(C_4, K_3, K_3)$, and $R(C_5, C_5, C_5)$. He used special seed values for these based on prior results.

He found new bounds on never before investigated numbers: $R(C_4, C_4, K_3, K_3) \geq 25$ and $R(C_5, C_4, K_3) \geq 13$. How much does this matter?

Results

Mutation rate on all good runs was 0.001, the lowest possible. So again, algorithms is mostly exploitation.

Rao matched known bounds for $R(C_4, C_4, C_4)$, $R(C_4, C_4, K_3)$, $R(C_4, K_3, K_3)$, and $R(C_5, C_5, C_5)$. He used special seed values for these based on prior results.

He found new bounds on never before investigated numbers: $R(C_4, C_4, K_3, K_3) \geq 25$ and $R(C_5, C_4, K_3) \geq 13$. How much does this matter?

He found numerous different colorings to prove these,

Results

Mutation rate on all good runs was 0.001, the lowest possible. So again, algorithms is mostly exploitation.

Rao matched known bounds for $R(C_4, C_4, C_4)$, $R(C_4, C_4, K_3)$, $R(C_4, K_3, K_3)$, and $R(C_5, C_5, C_5)$. He used special seed values for these based on prior results.

He found new bounds on never before investigated numbers: $R(C_4, C_4, K_3, K_3) \geq 25$ and $R(C_5, C_4, K_3) \geq 13$. How much does this matter?

He found numerous different colorings to prove these, but one coloring suffices for a proof.

How to improve on this

Because there is so little exploration, the seed matters a TON.

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly.

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

To really test an EA on Ramsey Theory you need to ask a harder question.

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

To really test an EA on Ramsey Theory you need to ask a harder question. Other subgraphs than C_n's and K_n's?

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

To really test an EA on Ramsey Theory you need to ask a harder question. Other subgraphs than C_n's and K_n's? Classical $R(s,t)$ and $R(n)$ numbers?

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

To really test an EA on Ramsey Theory you need to ask a harder question. Other subgraphs than C_n's and K_n's? Classical $R(s,t)$ and $R(n)$ numbers?
To make Rao's algorithm better, look into:

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

To really test an EA on Ramsey Theory you need to ask a harder question. Other subgraphs than C_n's and K_n's? Classical $R(s,t)$ and $R(n)$ numbers?
To make Rao's algorithm better, look into:

- Better way to fill the tournament

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

To really test an EA on Ramsey Theory you need to ask a harder question. Other subgraphs than C_n's and K_n's? Classical $R(s,t)$ and $R(n)$ numbers?
To make Rao's algorithm better, look into:

- Better way to fill the tournament
- Better selection method in general to encourage more exploration. Make it less steady-state.

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

To really test an EA on Ramsey Theory you need to ask a harder question. Other subgraphs than C_n's and K_n's? Classical $R(s,t)$ and $R(n)$ numbers?
To make Rao's algorithm better, look into:

- Better way to fill the tournament
- Better selection method in general to encourage more exploration. Make it less steady-state.
- Actually using mutation

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

To really test an EA on Ramsey Theory you need to ask a harder question. Other subgraphs than C_n's and K_n's? Classical $R(s,t)$ and $R(n)$ numbers?
To make Rao's algorithm better, look into:

- Better way to fill the tournament
- Better selection method in general to encourage more exploration. Make it less steady-state.
- Actually using mutation
- Make the fitness penalty smarter than just -1

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

To really test an EA on Ramsey Theory you need to ask a harder question. Other subgraphs than C_n's and K_n's? Classical $R(s,t)$ and $R(n)$ numbers?
To make Rao's algorithm better, look into:

- Better way to fill the tournament
- Better selection method in general to encourage more exploration. Make it less steady-state.
- Actually using mutation
- Make the fitness penalty smarter than just -1
- Smarter search space with fewer obvious bad colorings

How to improve on this

Because there is so little exploration, the seed matters a TON. This is why the best solutions found were usually found so quickly. The evolution here does almost nothing.

To really test an EA on Ramsey Theory you need to ask a harder question. Other subgraphs than C_n's and K_n's? Classical $R(s,t)$ and $R(n)$ numbers?
To make Rao's algorithm better, look into:

- Better way to fill the tournament
- Better selection method in general to encourage more exploration. Make it less steady-state.
- Actually using mutation
- Make the fitness penalty smarter than just -1
- Smarter search space with fewer obvious bad colorings
- Ask about Ramsey numbers of directed graphs or hypergraphs.

Is there any hope?

Will EAs be able to make progress on finding lower bounds for Ramsey numbers?

Is there any hope?

Will EAs be able to make progress on finding lower bounds for Ramsey numbers? Definitely not if they are basically doing brute force!

Is there any hope?

Will EAs be able to make progress on finding lower bounds for Ramsey numbers? Definitely not if they are basically doing brute force!

Currently, the best lower bounds mathematics has come up with are **non-constructive**.

Is there any hope?

Will EAs be able to make progress on finding lower bounds for Ramsey numbers? Definitely not if they are basically doing brute force!

Currently, the best lower bounds mathematics has come up with are **non-constructive**. They are proven to exist using probability theory.

Is there any hope?

David White
Wesleyan University

Will EAs be able to make progress on finding lower bounds for Ramsey numbers? Definitely not if they are basically doing brute force!

Currently, the best lower bounds mathematics has come up with are **non-constructive**. They are proven to exist using probability theory. But just because humans can't find clever colorings doesn't mean computers can't.

Is there any hope?

Will EAs be able to make progress on finding lower bounds for Ramsey numbers? Definitely not if they are basically doing brute force!

Currently, the best lower bounds mathematics has come up with are **non-constructive**. They are proven to exist using probability theory. But just because humans can't find clever colorings doesn't mean computers can't.

If nothing else this is a good hard problem like TSP to test a new EA implementation on.

Is there any hope?

Will EAs be able to make progress on finding lower bounds for Ramsey numbers? Definitely not if they are basically doing brute force!

Currently, the best lower bounds mathematics has come up with are **non-constructive**. They are proven to exist using probability theory. But just because humans can't find clever colorings doesn't mean computers can't.

If nothing else this is a good hard problem like TSP to test a new EA implementation on. It's a great "Humies" problem because humans have basically given up on finding optimal colorings.

Is there any hope?

Will EAs be able to make progress on finding lower bounds for Ramsey numbers? Definitely not if they are basically doing brute force!

Currently, the best lower bounds mathematics has come up with are **non-constructive**. They are proven to exist using probability theory. But just because humans can't find clever colorings doesn't mean computers can't.

If nothing else this is a good hard problem like TSP to test a new EA implementation on. It's a great "Humies" problem because humans have basically given up on finding optimal colorings. Just waiting for a break-through.

Is there any hope?

Will EAs be able to make progress on finding lower bounds for Ramsey numbers? Definitely not if they are basically doing brute force!

Currently, the best lower bounds mathematics has come up with are **non-constructive**. They are proven to exist using probability theory. But just because humans can't find clever colorings doesn't mean computers can't.

If nothing else this is a good hard problem like TSP to test a new EA implementation on. It's a great "Humies" problem because humans have basically given up on finding optimal colorings. Just waiting for a break-through.

No one has attacked Ramsey theory using EAs in even a remotely clever way.

Is there any hope?

Will EAs be able to make progress on finding lower bounds for Ramsey numbers? Definitely not if they are basically doing brute force!

Currently, the best lower bounds mathematics has come up with are **non-constructive**. They are proven to exist using probability theory. But just because humans can't find clever colorings doesn't mean computers can't.

If nothing else this is a good hard problem like TSP to test a new EA implementation on. It's a great "Humies" problem because humans have basically given up on finding optimal colorings. Just waiting for a break-through.

No one has attacked Ramsey theory using EAs in even a remotely clever way. Plenty of room for improvement.

Is there any hope?

Will EAs be able to make progress on finding lower bounds for Ramsey numbers? Definitely not if they are basically doing brute force!

Currently, the best lower bounds mathematics has come up with are **non-constructive**. They are proven to exist using probability theory. But just because humans can't find clever colorings doesn't mean computers can't.

If nothing else this is a good hard problem like TSP to test a new EA implementation on. It's a great "Humies" problem because humans have basically given up on finding optimal colorings. Just waiting for a break-through.

No one has attacked Ramsey theory using EAs in even a remotely clever way. Plenty of room for improvement. We finish with some ideas for how to do this.

Other attacks

Geoffrey Exoo (1998) used simulated annealing.

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search.

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter,

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter, so there is room for improvement.

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter, so there is room for improvement. But he improved bounds on $R(5, t)$ for $9 \leq t \leq 15$

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter, so there is room for improvement. But he improved bounds on $R(5, t)$ for $9 \leq t \leq 15$

Try the "evolving non-determinism" algorithm from sorting networks.

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter, so there is room for improvement. But he improved bounds on $R(5,t)$ for $9 \leq t \leq 15$

Try the "evolving non-determinism" algorithm from sorting networks. This found solutions lacking symmetry which the best humans missed.

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter, so there is room for improvement. But he improved bounds on $R(5, t)$ for $9 \leq t \leq 15$

Try the "evolving non-determinism" algorithm from sorting networks. This found solutions lacking symmetry which the best humans missed.

Be greedy but smart:

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter, so there is room for improvement. But he improved bounds on $R(5,t)$ for $9 \leq t \leq 15$

Try the "evolving non-determinism" algorithm from sorting networks. This found solutions lacking symmetry which the best humans missed.

Be greedy but smart: fill in as many edges in color 1 as possible first without monochromatic G_1.

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter, so there is room for improvement. But he improved bounds on $R(5,t)$ for $9 \leq t \leq 15$

Try the "evolving non-determinism" algorithm from sorting networks. This found solutions lacking symmetry which the best humans missed.

Be greedy but smart: fill in as many edges in color 1 as possible first without monochromatic G_1. While doing so, try to space out your edges to break up uncolored K_n's

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter, so there is room for improvement. But he improved bounds on $R(5,t)$ for $9 \leq t \leq 15$

Try the "evolving non-determinism" algorithm from sorting networks. This found solutions lacking symmetry which the best humans missed.

Be greedy but smart: fill in as many edges in color 1 as possible first without monochromatic G_1. While doing so, try to space out your edges to break up uncolored K_n's

Use a hierarchical GA:

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter, so there is room for improvement. But he improved bounds on $R(5,t)$ for $9 \leq t \leq 15$

Try the "evolving non-determinism" algorithm from sorting networks. This found solutions lacking symmetry which the best humans missed.

Be greedy but smart: fill in as many edges in color 1 as possible first without monochromatic G_1. While doing so, try to space out your edges to break up uncolored K_n's

Use a hierarchical GA: solve smaller instances of the problem and then combine solutions.

Other attacks

Geoffrey Exoo (1998) used simulated annealing. Went from random search to Tabu search. The annealing was very fast, and he didn't play with that parameter, so there is room for improvement. But he improved bounds on $R(5,t)$ for $9 \leq t \leq 15$

Try the "evolving non-determinism" algorithm from sorting networks. This found solutions lacking symmetry which the best humans missed.

Be greedy but smart: fill in as many edges in color 1 as possible first without monochromatic G_1. While doing so, try to space out your edges to break up uncolored K_n's

Use a hierarchical GA: solve smaller instances of the problem and then combine solutions. If $R(G_1, G_2) = n$ avoid an uncolored K_n when you have only 2 colors left.

References

Images from:

1. New Mexico Supercomputing Challenge: http://www.challenge.nm.org/archive/06-07/finalreports/07/
2. Wolfram: www.mathworld.wolfram.com
3. Mathematica Player 7 program: 'GraphsAndTheirComplements'
4. Professor Tibor Szabó: http://www.ti.inf.ethz.ch/ew/teaching/tspz-01.html