Non-Computability in Graphs

Oscar Levin

University of Northern Colorado

UW Colloquium
October 24, 2013

Computability Theory

- Interested in the nature of computable functions
- Alternatively: recursive functions, lambda calculus, Turing machines, algorithms, etc.
- The meat: how can we talk about non-computable functions?
- Connection to logic: the more non-computable a function is, the more quantifiers we need to define it.

Computability Theory

- Interested in the nature of computable functions
- Alternatively: recursive functions, lambda calculus, Turing machines, algorithms, etc.
- The meat: how can we talk about non-computable functions?
- Connection to logic: the more non-computable a function is, the more quantifiers we need to define it.

Computability Theory

- Interested in the nature of computable functions
- Alternatively: recursive functions, lambda calculus, Turing machines, algorithms, etc.
- The meat: how can we talk about non-computable functions?
- Connection to logic: the more non-computable a function is, the more quantifiers we need to define it.

Computability Theory

- Interested in the nature of computable functions
- Alternatively: recursive functions, lambda calculus, Turing machines, algorithms, etc.
- The meat: how can we talk about non-computable functions?
- Connection to logic: the more non-computable a function is, the more quantifiers we need to define it.

Computability and Graph Theory

Gain insight into computability theory using graphs as a canvas.
    Example: computable dimension.

Gain insight into graph theory using computability as a tool.
    Example: the Four Color Theorem.

Computability and Graph Theory

Gain insight into computability theory using graphs as a canvas.
Example: computable dimension.

Gain insight into graph theory using computability as a tool.
Example: the Four Color Theorem.

Computable Dimension

Given two computable copies $\mathcal{A}$, $\mathcal{B}$ of the same structure, there might or might not be a *computable* isomorphism between $\mathcal{A}$ and $\mathcal{B}$.

The number of copies of a structure up to computable isomorphism is the *computable dimension* of the structure.

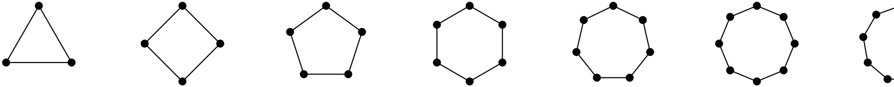
Computable Dimension

Given two computable copies $\mathcal{A}$, $\mathcal{B}$ of the same structure, there might or might not be a *computable* isomorphism between $\mathcal{A}$ and $\mathcal{B}$.

The number of copies of a structure up to computable isomorphism is the *computable dimension* of the structure.

1 or ω

A graph with computable dimension 1:

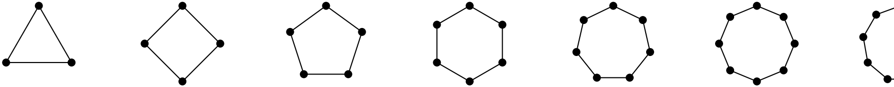


A graph with computable dimension ω:

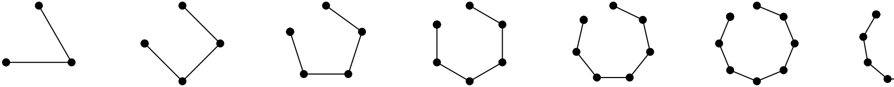
Question: Are there structures which have finite computable dimension greater than 1?

1 or ω

A graph with computable dimension 1:



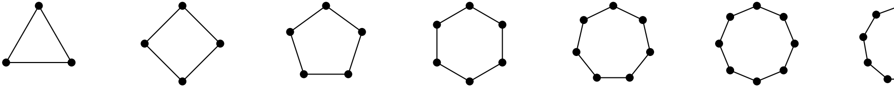
A graph with computable dimension ω:



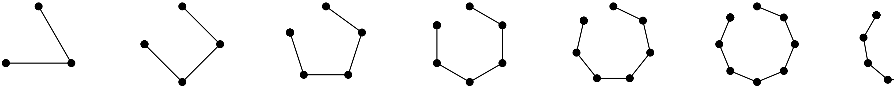
Question: Are there structures which have finite computable dimension greater than 1?

1 or ω

A graph with computable dimension 1:



A graph with computable dimension ω:



Question: Are there structures which have finite computable dimension greater than 1?

Computable chromatic number

Any planar graph has a 4-coloring

There are computable planar graphs with no computable k-coloring for any k.

Computable chromatic number

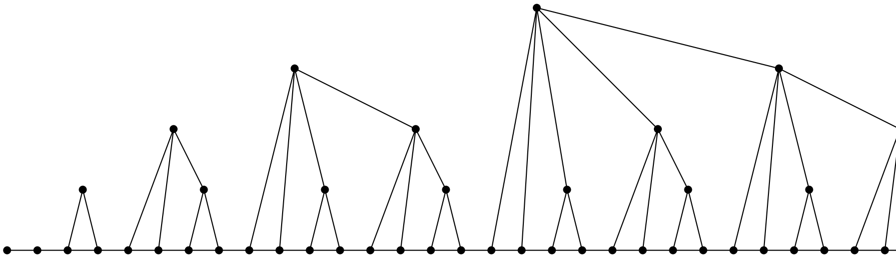
Any planar graph has a 4-coloring

There are computable planar graphs with no computable k-coloring for any k.

Computable chromatic number

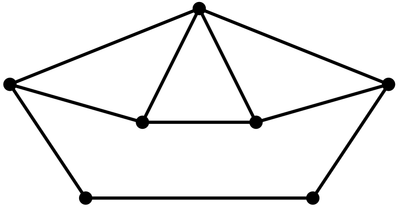
Any planar graph has a 4-coloring

There are computable planar graphs with no computable k-coloring for any k.



Dominating Sets in Graphs

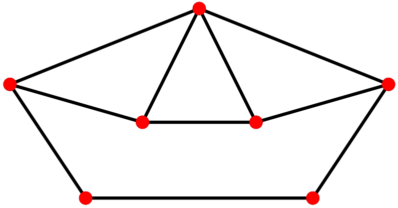
Given a graph, we look for sets of vertices close to everything.



A set is dominating if every vertex of G is in, or adjacent to a vertex in, the set.

Dominating Sets in Graphs

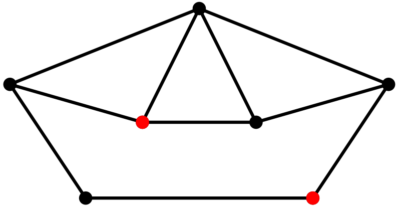
Given a graph, we look for sets of vertices close to everything.



A set is dominating if every vertex of G is in, or adjacent to a vertex in, the set.

Dominating Sets in Graphs

Given a graph, we look for sets of vertices close to everything.



A set is dominating if every vertex of G is in, or adjacent to a vertex in, the set.

Domatic Partitions

Definition

A domatic k-partition of a graph G is a partition of (all) the vertices of G into k (disjoint) dominating sets.

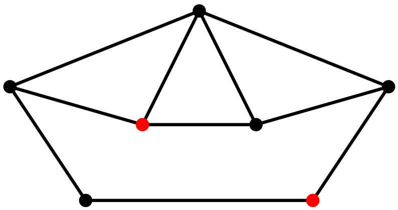
The domatic number $d(G)$ is the size of a largest domatic partition.

Domatic Partitions

Definition

A domatic k-partition of a graph G is a partition of (all) the vertices of G into k (disjoint) dominating sets.

The domatic number $d(G)$ is the size of a largest domatic partition.

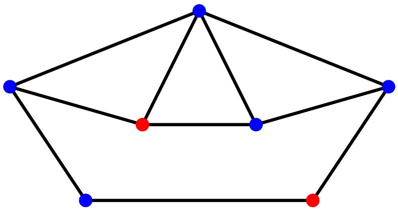


Domatic Partitions

Definition

A domatic k-partition of a graph G is a partition of (all) the vertices of G into k (disjoint) dominating sets.

The domatic number $d(G)$ is the size of a largest domatic partition.

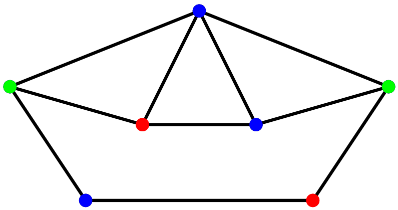


Domatic Partitions

Definition

A domatic k-partition of a graph G is a partition of (all) the vertices of G into k (disjoint) dominating sets.

The domatic number $d(G)$ is the size of a largest domatic partition.

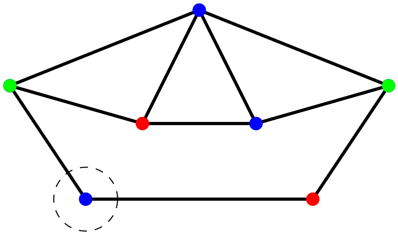


Domatic Partitions

Definition

A domatic k-partition of a graph G is a partition of (all) the vertices of G into k (disjoint) dominating sets.

The domatic number $d(G)$ is the size of a largest domatic partition.



Main Question

Question

Given a computable graph G with domatic number n, what is the size of the largest computable domatic partition of G?
In other words, what is $d^c(G)$, the computable domatic number?

If $d(G) = 2$ then $d^c(G) = 2$.

Suppose G has a domatic 2-partition (so no isolated vertices).
There is an algorithm which produces a domatic 2-partition.

Vertices: $\{v_0, v_1, v_2, \ldots\}$

Put $v_0 \in A$.

Put $v_n \in B$ iff there is an adjacent vertex $v_k \in A$ (with $k < n$)

A is a dominating set: if $v_n \notin A$ then ...

B is a dominating set: if $v_n \notin B$ then ...

If $d(G) = 2$ then $d^c(G) = 2$.

Suppose G has a domatic 2-partition (so no isolated vertices). There is an algorithm which produces a domatic 2-partition.

Vertices: $\{v_0, v_1, v_2, \ldots\}$

Put $v_0 \in A$.

Put $v_n \in B$ iff there is an adjacent vertex $v_k \in A$ (with $k < n$)

A is a dominating set: if $v_n \notin A$ then ...

B is a dominating set: if $v_n \notin B$ then ...

If $d(G) = 2$ then $d^c(G) = 2$.

Suppose G has a domatic 2-partition (so no isolated vertices).
There is an algorithm which produces a domatic 2-partition.

Vertices: $\{v_0, v_1, v_2, \ldots\}$

Put $v_0 \in A$.

Put $v_n \in B$ iff there is an adjacent vertex $v_k \in A$ (with $k < n$)

A is a dominating set: if $v_n \notin A$ then ...

B is a dominating set: if $v_n \notin B$ then ...

If $d(G) = 2$ then $d^c(G) = 2$.

Suppose G has a domatic 2-partition (so no isolated vertices).
There is an algorithm which produces a domatic 2-partition.

Vertices: $\{v_0, v_1, v_2, \ldots\}$

Put $v_0 \in A$.

Put $v_n \in B$ iff there is an adjacent vertex $v_k \in A$ (with $k < n$)

A is a dominating set: if $v_n \notin A$ then $\ldots$

B is a dominating set: if $v_n \notin B$ then $\ldots$

If $d(G) = 2$ then $d^c(G) = 2$.

Suppose G has a domatic 2-partition (so no isolated vertices).
There is an algorithm which produces a domatic 2-partition.

Vertices: $\{v_0, v_1, v_2, \ldots\}$

Put $v_0 \in A$.

Put $v_n \in B$ iff there is an adjacent vertex $v_k \in A$ (with $k < n$)

A is a dominating set: if $v_n \notin A$ then ...

B is a dominating set: if $v_n \notin B$ then ...

If $d(G) = 2$ then $d^c(G) = 2$.

Suppose G has a domatic 2-partition (so no isolated vertices). There is an algorithm which produces a domatic 2-partition.

Vertices: $\{v_0, v_1, v_2, \ldots\}$

Put $v_0 \in A$.

Put $v_n \in B$ iff there is an adjacent vertex $v_k \in A$ (with $k < n$)

A is a dominating set: if $v_n \notin A$ then ...

B is a dominating set: if $v_n \notin B$ then ...

If $d(G) = 2$ then $d^c(G) = 2$.

Suppose G has a domatic 2-partition (so no isolated vertices).
There is an algorithm which produces a domatic 2-partition.

Vertices: $\{v_0, v_1, v_2, \ldots\}$

Put $v_0 \in A$.

Put $v_n \in B$ iff there is an adjacent vertex $v_k \in A$ (with $k < n$)

A is a dominating set: if $v_n \notin A$ then ...

B is a dominating set: if $v_n \notin B$ then ...

What if $d(G) = 3$?

Proposition

There is a computable graph with domatic number 3 but computable domatic number 2.

To prove this, we diagonalize against all computable functions.

What if $d(G) = 3$?

Proposition

There is a computable graph with domatic number 3 but computable domatic number 2.

To prove this, we diagonalize against all computable functions.

Some More Computability Theory

There is an effective list of all (partial) computable functions:

$$\varphi_0, \varphi_1, \varphi_2, \cdots$$

These can be simulated by a universal computable function

We can run these programs "simultaneously" to see if any look like they compute a domatic 3-partition.

Meanwhile, we build a computable graph with a 3-partition

When some φ_e tries to compute a 3-partition, we thwart it.

Some More Computability Theory

There is an effective list of all (partial) computable functions:

$$\varphi_0, \varphi_1, \varphi_2, \dots$$

These can be simulated by a universal computable function

We can run these programs "simultaneously" to see if any look like they compute a domatic 3-partition.

Meanwhile, we build a computable graph with a 3-partition

When some φ_e tries to compute a 3-partition, we thwart it.

Some More Computability Theory

There is an effective list of all (partial) computable functions:

$$\varphi_0, \varphi_1, \varphi_2, \cdots$$

These can be simulated by a universal computable function

We can run these programs "simultaneously" to see if any look like they compute a domatic 3-partition.

Meanwhile, we build a computable graph with a 3-partition

When some φ_e tries to compute a 3-partition, we thwart it.

Some More Computability Theory

There is an effective list of all (partial) computable functions:

$$\varphi_0, \varphi_1, \varphi_2, \dots$$

These can be simulated by a universal computable function

We can run these programs "simultaneously" to see if any look like they compute a domatic 3-partition.

Meanwhile, we build a computable graph with a 3-partition

When some φ_e tries to compute a 3-partition, we thwart it.

Some More Computability Theory

There is an effective list of all (partial) computable functions:

$$\varphi_0, \varphi_1, \varphi_2, \dots$$

These can be simulated by a universal computable function

We can run these programs "simultaneously" to see if any look like they compute a domatic 3-partition.

Meanwhile, we build a computable graph with a 3-partition

When some φ_e tries to compute a 3-partition, we thwart it.

The Construction

G will start with copies of K_4, one for each φ_e.

Build G in stages. At each stage, build a new K_4 and check whether φ_e has halted on its copy of K_4.

If φ_e looks like it computes a 3-partition on its K_4, spring the trap!

The Construction

G will start with copies of K_4, one for each φ_e.

Build G in stages. At each stage, build a new K_4 and check whether φ_e has halted on its copy of K_4.

If φ_e looks like it computes a 3-partition on its K_4, spring the trap!

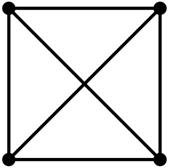
The Construction

G will start with copies of K_4, one for each φ_e.

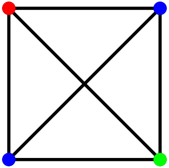
Build G in stages. At each stage, build a new K_4 and check whether φ_e has halted on its copy of K_4.

If φ_e looks like it computes a 3-partition on its K_4, spring the trap!

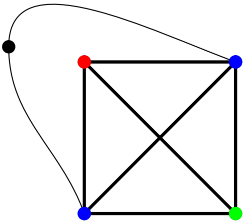
The Trap



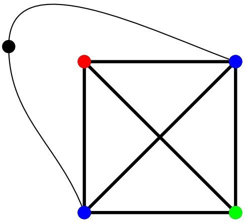
The Trap



The Trap

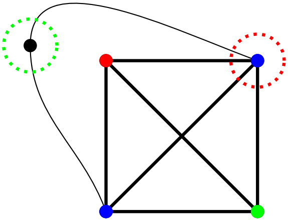


The Trap



The sprung trap still has a 3-partition, but not the one φ_e claims.

The Trap



The sprung trap still has a 3-partition, but not the one φ_e claims.

~~Worse~~ Better than that...

Proposition

For any n, there is a computable graph with domatic number n but computable domatic number 2.

Use $K_{3(n-2)+1}$ as the trap to diagonalize against all possible computable domatic 3-partitions.

~~Worse~~ Better than that. . .

Proposition

For any n, there is a computable graph with domatic number n but computable domatic number 2.

Use $K_{3(n-2)+1}$ as the trap to diagonalize against all possible computable domatic 3-partitions.

Stupid φ_e

Why does φ_e partition its trap so soon?

Just because G is computable, doesn't mean we can compute the degree of a given vertex!

But what if we could?

Stupid φ_e

Why does φ_e partition its trap so soon?

Just because G is computable, doesn't mean we can compute the degree of a given vertex!

But what if we could?

Stupid φ_e

Why does φ_e partition its trap so soon?

Just because G is computable, doesn't mean we can compute the degree of a given vertex!

But what if we could?

Highly computable graphs

Definition

A graph is highly computable if it is computable and degree function is computable.

Does this extra information help φ_e compute a domatic partition?

Proposition

There is a highly computable graph with domatic number 3 but computable domatic number 2.

Highly computable graphs

Definition

A graph is highly computable if it is computable and degree function is computable.

Does this extra information help φ_e compute a domatic partition?

Proposition

There is a highly computable graph with domatic number 3 but computable domatic number 2.

The idea: remotely sprung traps

Wait for φ_e to partition some fixed vertices. Then act.

Our action cannot change the degree of any vertex in the graph.

φ_e might never partition its vertices, but we don't know that at any finite stage.

We must be able to force φ_e's partition to be wrong, by modifying the graph arbitrarily far away from φ_e's vertices.

The idea: remotely sprung traps

Wait for φ_e to partition some fixed vertices. Then act.

Our action cannot change the degree of any vertex in the graph.

φ_e might never partition its vertices, but we don't know that at any finite stage.

We must be able to force φ_e's partition to be wrong, by modifying the graph arbitrarily far away from φ_e's vertices.

The idea: remotely sprung traps

Wait for φ_e to partition some fixed vertices. Then act.

Our action cannot change the degree of any vertex in the graph.

φ_e might never partition its vertices, but we don't know that at any finite stage.

We must be able to force φ_e's partition to be wrong, by modifying the graph arbitrarily far away from φ_e's vertices.

The idea: remotely sprung traps

Wait for φ_e to partition some fixed vertices. Then act.

Our action cannot change the degree of any vertex in the graph.

φ_e might never partition its vertices, but we don't know that at any finite stage.

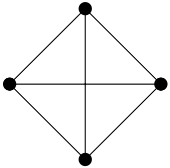
We must be able to force φ_e's partition to be wrong, by modifying the graph arbitrarily far away from φ_e's vertices.

Highly intricate trap

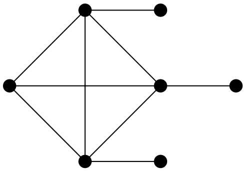
A path:

Every third vertex must be colored the same.

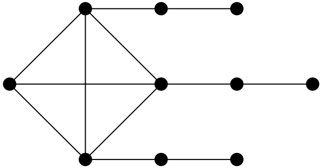
Springing the trap



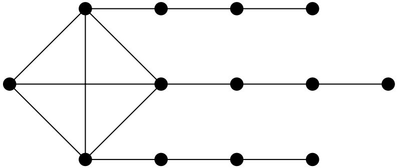
Springing the trap



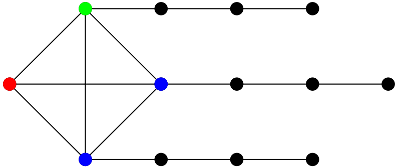
Springing the trap



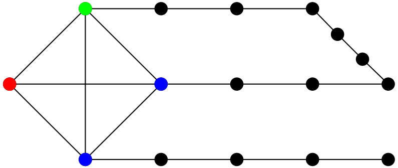
Springing the trap



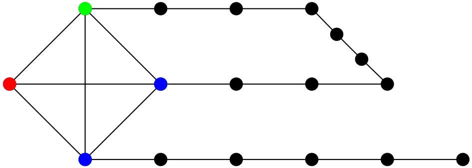
Springing the trap



Springing the trap



Springing the trap



If $d(G) = 4$ then...

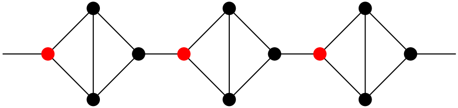
Proposition

There is a highly computable graph with domatic number 4 but computable domatic number 3.

If $d(G) = 4$ then...

Proposition

There is a highly computable graph with domatic number 4 but computable domatic number 3.



Can we do better?

Is it easier to find smaller domatic partitions in highly computable graphs?

Conjecture

Any highly computable graph with domatic number n has computable domatic number at least $f(n)$.

Maybe $f(n) = n - 1$. Or $f(n) = (n + 1)/2$

Can we do better?

Is it easier to find smaller domatic partitions in highly computable graphs?

Conjecture

Any highly computable graph with domatic number n has computable domatic number at least $f(n)$.

Maybe $f(n) = n - 1$. Or $f(n) = (n+1)/2$

Can we do better?

Is it easier to find smaller domatic partitions in highly computable graphs?

Conjecture

Any highly computable graph with domatic number n has computable domatic number at least $f(n)$.

Maybe $f(n) = n - 1$. Or $f(n) = (n + 1)/2$

The End

Thanks for listening

Partial results towards and away from the conjecture

Proposition

Let $k \geq 3$. For any non-computable c.e. set A, there is an A-computable graph $G = (V, E)$ such that G has a domatic k-partition, but no computable domatic 3-partition.

Proposition

There is a highly computable graph with domatic number n, but no computable splittable domatic $(n-1)$-partition.

Partial results towards and away from the conjecture

Proposition

Let $k \geq 3$. For any non-computable c.e. set A, there is an A-computable graph $G = (V, E)$ such that G has a domatic k-partition, but no computable domatic 3-partition.

Proposition

There is a highly computable graph with domatic number n, but no computable splittable domatic $(n-1)$-partition.