# Fixed-Point Property on Triod-like Continua

Paul Gooderham-Belanger
(pjgooderhambelanger557@community.nipissingu.ca)

Nipissing University

May 28, 2015

# History

## Question (Bing 1969)

*Does each tree-like continuum have the fixed-point property?*

# History

## Question (Bing 1969)

*Does each tree-like continuum have the fixed-point property?*

Answered in the negative by D. P. Bellamy in 1979

# History

## Question (Bing 1969)

*Does each tree-like continuum have the fixed-point property?*

Answered in the negative by D. P. Bellamy in 1979

## Question (Hagopian 2007)

*Does every triod-like continuum have the fixed-point property?*

# History

## Question (Bing 1969)

*Does each tree-like continuum have the fixed-point property?*

Answered in the negative by D. P. Bellamy in 1979

## Question (Hagopian 2007)

*Does every triod-like continuum have the fixed-point property?*

## Question (Hagopian 2007)

*Do there exist two commuting maps of a triod onto itself that to not have a coincidence point?*

# History

## Question (Bing 1969)

*Does each tree-like continuum have the fixed-point property?*

Answered in the negative by D. P. Bellamy in 1979

## Question (Hagopian 2007)

*Does every triod-like continuum have the fixed-point property?*

## Question (Hagopian 2007)

*Do there exist two commuting maps of a triod onto itself that to not have a coincidence point?*

Confirming Question 3 would provide a counter-example for Question 2.

## Proposition

*Let $f$ and $g$ be two continuous maps of a triod onto itself. If $f$ and $g$ are disjoint and commuting, then there exists a triod-like continuum which does not have the fixed-point property.*

## Proposition

*Let $f$ and $g$ be two continuous maps of a triod onto itself. If $f$ and $g$ are disjoint and commuting, then there exists a triod-like continuum which does not have the fixed-point property.*

## Proof.

Let $f$ and $g$ be two continuous disjoint commuting maps of a triod $T$ onto itself.

## Proposition

*Let $f$ and $g$ be two continuous maps of a triod onto itself. If $f$ and $g$ are disjoint and commuting, then there exists a triod-like continuum which does not have the fixed-point property.*

## Proof.

Let $f$ and $g$ be two continuous disjoint commuting maps of a triod $T$ onto itself.

Let $S$ be the inverse limit of triods under the bonding maps $f_i = f$. Then $S = \varprojlim(T, f) = \{p = (p_1, p_2, \ldots) : p_i \in T \text{ and } f(p_{i+1}) = p_i \ \forall i\}$.

## Proposition

*Let $f$ and $g$ be two continuous maps of a triod onto itself. If $f$ and $g$ are disjoint and commuting, then there exists a triod-like continuum which does not have the fixed-point property.*

## Proof.

Let $f$ and $g$ be two continuous disjoint commuting maps of a triod $T$ onto itself.

Let $S$ be the inverse limit of triods under the bonding maps $f_i = f$. Then $S = \varprojlim(T, f) = \{p = (p_1, p_2, \ldots) : p_i \in T \text{ and } f(p_{i+1}) = p_i \ \forall i\}$.

Let $\widetilde{g} : S \to S$ be such that $\widetilde{g}(p) = (g(p_2), g(p_3), \ldots) \ \forall p \in S$.

## Proposition

*Let $f$ and $g$ be two continuous maps of a triod onto itself. If $f$ and $g$ are disjoint and commuting, then there exists a triod-like continuum which does not have the fixed-point property.*

## Proof.

Let $f$ and $g$ be two continuous disjoint commuting maps of a triod $T$ onto itself.

Let $S$ be the inverse limit of triods under the bonding maps $f_i = f$. Then $S = \varprojlim(T, f) = \{p = (p_1, p_2, \ldots) : p_i \in T \text{ and } f(p_{i+1}) = p_i \ \forall i\}$.

Let $\widetilde{g} : S \to S$ be such that $\widetilde{g}(p) = (g(p_2), g(p_3), \ldots) \ \forall p \in S$.

Note that $\widetilde{g}(p) \in S$ since $f(g(p_{i+1})) = g(f(p_{i+1})) = g(p_i)$.

## Proposition

*Let $f$ and $g$ be two continuous maps of a triod onto itself. If $f$ and $g$ are disjoint and commuting, then there exists a triod-like continuum which does not have the fixed-point property.*

## Proof.

Let $f$ and $g$ be two continuous disjoint commuting maps of a triod $T$ onto itself.

Let $S$ be the inverse limit of triods under the bonding maps $f_i = f$. Then $S = \varprojlim(T, f) = \{p = (p_1, p_2, \ldots) : p_i \in T \text{ and } f(p_{i+1}) = p_i \ \forall i\}$.

Let $\widetilde{g} : S \to S$ be such that $\widetilde{g}(p) = (g(p_2), g(p_3), \ldots) \ \forall p \in S$.

Note that $\widetilde{g}(p) \in S$ since $f(g(p_{i+1})) = g(f(p_{i+1})) = g(p_i)$.

Now suppose that $\widetilde{g}$ has a fixed point. Then for some $p \in S$ we have $\widetilde{g}(p) = p \Rightarrow (g(p_2), g(p_3), \ldots) = (p_1, p_2, \ldots) \Rightarrow g(p_2) = p_1$.

## Proposition

*Let $f$ and $g$ be two continuous maps of a triod onto itself. If $f$ and $g$ are disjoint and commuting, then there exists a triod-like continuum which does not have the fixed-point property.*

## Proof.

Let $f$ and $g$ be two continuous disjoint commuting maps of a triod $T$ onto itself.

Let $S$ be the inverse limit of triods under the bonding maps $f_i = f$. Then $S = \varprojlim(T, f) = \{p = (p_1, p_2, \ldots) : p_i \in T \text{ and } f(p_{i+1}) = p_i \ \forall i\}$.

Let $\widetilde{g} : S \to S$ be such that $\widetilde{g}(p) = (g(p_2), g(p_3), \ldots) \ \forall p \in S$.

Note that $\widetilde{g}(p) \in S$ since $f(g(p_{i+1})) = g(f(p_{i+1})) = g(p_i)$.

Now suppose that $\widetilde{g}$ has a fixed point. Then for some $p \in S$ we have $\widetilde{g}(p) = p \Rightarrow (g(p_2), g(p_3), \ldots) = (p_1, p_2, \ldots) \Rightarrow g(p_2) = p_1$.

But $f$ and $g$ are disjoint and we have $f(p_2) = p_1 = g(p_2)$. This is a contradiction. Thus $\widetilde{g}$ does not contain a fixed point.

# Data Structure

- $T = \{(i, d) : i \in \{0, 1, 2\}, d \in [0, 1]\}$

# Data Structure

- $T = \{(i, d) : i \in \{0, 1, 2\}, d \in [0, 1]\}$
- A triod is represented as a series of connected vertices

# Data Structure

- $T = \{(i, d) : i \in \{0, 1, 2\}, d \in [0, 1]\}$
- A triod is represented as a series of connected vertices
- Given $N, M \in \mathbb{N}$, a mapping $f : T_N \to T_M$ maps a triod with $3N + 1$ vertices onto a triod with $3M + 1$ vertices.

# Data Structure

- $T = \{(i, d) : i \in \{0, 1, 2\}, d \in [0, 1]\}$
- A triod is represented as a series of connected vertices
- Given $N, M \in \mathbb{N}$, a mapping $f : T_N \to T_M$ maps a triod with $3N + 1$ vertices onto a triod with $3M + 1$ vertices.
- Points on triod are interpolated between vertices

# Data Structure

- $T = \{(i, d) : i \in \{0, 1, 2\}, d \in [0, 1]\}$
- A triod is represented as a series of connected vertices
- Given $N, M \in \mathbb{N}$, a mapping $f : T_N \to T_M$ maps a triod with $3N + 1$ vertices onto a triod with $3M + 1$ vertices.
- Points on triod are interpolated between vertices
- Interpolation enforces continuity of resultant maps.

# Generating Mappings

# Generating Mappings

- Every mapping starts with the image of the branch point

# Generating Mappings

- Every mapping starts with the image of the branch point
- Let $N$ be the number of vertices per leg of the domain triod
- The for each leg $i$, take $f(i, \frac{d+1}{N})$ such that it is adjacent to $f(i, \frac{d}{N})$.

# Generating Mappings

- Every mapping starts with the image of the branch point
- Let $N$ be the number of vertices per leg of the domain triod
- The for each leg $i$, take $f(i, \frac{d+1}{N})$ such that it is adjacent to $f(i, \frac{d}{N})$.
- Recursively permute over every combination

# The search space grows very quickly!

# The search space grows very quickly!

| M \N | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 88 | 1648 | 34 240 | 694 528 | 14 182 912 |
| 2 | X | 4759 | 313 281 | 3 572 647 | 96 722 809 |
| 3 | X | X | 211 972 | 6 295 732 | 186 132 664 |
| 4 | X | X | X | 8 307 175 | 252 055 519 |
| 5 | X | X | X | X | 302 795 176 |

# The search space grows very quickly!

| M \N | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 88 | 1648 | 34 240 | 694 528 | 14 182 912 |
| 2 | X | 4759 | 313 281 | 3 572 647 | 96 722 809 |
| 3 | X | X | 211 972 | 6 295 732 | 186 132 664 |
| 4 | X | X | X | 8 307 175 | 252 055 519 |
| 5 | X | X | X | X | 302 795 176 |

Remember that we need to check each of these pair-wise with each other. Thus for N=4, M=2, we need to check $\binom{3572647}{2} = 6.38 \times 10^{12}$ pairs!

# High Performance Computing to the Rescue

# High Performance Computing to the Rescue

- Can utilize hundreds of CPU Cores at once to speed up the process.

# High Performance Computing to the Rescue

- Can utilize hundreds of CPU Cores at once to speed up the process.
- Program was implemented on the SHARCNET Compute cluster in Python over MPI

# High Performance Computing to the Rescue

- Can utilize hundreds of CPU Cores at once to speed up the process.
- Program was implemented on the SHARCNET Compute cluster in Python over MPI
- Program uses a master/slave layout

# High Performance Computing to the Rescue

- Can utilize hundreds of CPU Cores at once to speed up the process.
- Program was implemented on the SHARCNET Compute cluster in Python over MPI
- Program uses a master/slave layout
- Master program generates pairs of empty mappings specifying only the image of the base point

# High Performance Computing to the Rescue

- Can utilize hundreds of CPU Cores at once to speed up the process.
- Program was implemented on the SHARCNET Compute cluster in Python over MPI
- Program uses a master/slave layout
- Master program generates pairs of empty mappings specifying only the image of the base point
- Master program distributes pairs to worker programs

# High Performance Computing to the Rescue

- Can utilize hundreds of CPU Cores at once to speed up the process.
- Program was implemented on the SHARCNET Compute cluster in Python over MPI
- Program uses a master/slave layout
- Master program generates pairs of empty mappings specifying only the image of the base point
- Master program distributes pairs to worker programs
- Worker programs generate all completions and check for commutativity and disjointness.

# High Performance Computing to the Rescue

- Can utilize hundreds of CPU Cores at once to speed up the process.
- Program was implemented on the SHARCNET Compute cluster in Python over MPI
- Program uses a master/slave layout
- Master program generates pairs of empty mappings specifying only the image of the base point
- Master program distributes pairs to worker programs
- Worker programs generate all completions and check for commutativity and disjointness.
- Worker programs report winning pairs back to master program

# High Performance Computing to the Rescue

- Can utilize hundreds of CPU Cores at once to speed up the process.
- Program was implemented on the SHARCNET Compute cluster in Python over MPI
- Program uses a master/slave layout
- Master program generates pairs of empty mappings specifying only the image of the base point
- Master program distributes pairs to worker programs
- Worker programs generate all completions and check for commutativity and disjointness.
- Worker programs report winning pairs back to master program
- Master program records winning mappings for further investigation

# Checking Disjointness

# Checking Disjointness

$\forall i \in \{0, 1, 2\}$:
$\forall$ vertex $\frac{d}{N}$:

# Checking Disjointness

$\forall i \in \{0, 1, 2\}$:
$\forall$ vertex $\frac{d}{N}$:
If $f(i, \frac{d}{N}) = g(i, \frac{d}{N})$ then we have co-incidence at the vertex.

# Checking Disjointness

$\forall i \in \{0, 1, 2\}$:
$\forall$ vertex $\frac{d}{N}$:
If $f(i, \frac{d}{N}) = g(i, \frac{d}{N})$ then we have co-incidence at the vertex.
If $f(i, \frac{d}{N}) = g(i, \frac{d+1}{N})$ and $f(i, \frac{d+1}{N}) = g(i, \frac{d}{N})$ then we have co-incidence between vertices.

# Checking Disjointness

$\forall i \in \{0, 1, 2\}$:
$\forall$ vertex $\frac{d}{N}$:
If $f(i, \frac{d}{N}) = g(i, \frac{d}{N})$ then we have co-incidence at the vertex.
If $f(i, \frac{d}{N}) = g(i, \frac{d+1}{N})$ and $f(i, \frac{d+1}{N}) = g(i, \frac{d}{N})$ then we have co-incidence between vertices.
Otherwise the maps are disjoint.

# Checking Commutativity

# Checking Commutativity

- Define distance using a railway metric

# Checking Commutativity

- Define distance using a railway metric

Choose a small $\epsilon > 0$. Choose a large $D \in \mathbb{N}$

# Checking Commutativity

- Define distance using a railway metric

Choose a small $\epsilon > 0$. Choose a large $D \in \mathbb{N}$
Let $d = \max_{v \in \{0,1,2,\ldots,D\}} \{|(f \circ g)(i, \frac{v}{D}) - (g \circ f)(i, \frac{v}{D})|\}$.

# Checking Commutativity

- Define distance using a railway metric

Choose a small $\epsilon > 0$. Choose a large $D \in \mathbb{N}$
Let $d = \max_{v \in \{0,1,2,\ldots,D\}} \{|(f \circ g)(i, \frac{v}{D}) - (g \circ f)(i, \frac{v}{D})|\}$.
If $d < \epsilon$ then the maps commute.

# Reducing the Search Space

# Reducing the Search Space

- Enforcing Disjointness

# Reducing the Search Space

- Enforcing Disjointness
- Surjectivity

# Enforcing Disjointness

# Enforcing Disjointness

- After generating the first map, generate the second map based on it
- skip completions which would cause a co-incidence condition

# Surjectivity

# Surjectivity

- We can enforce that each map is surjective

# Surjectivity

- We can enforce that each map is surjective
- The vast majority of maps are not surjective

# Surjectivity

- We can enforce that each map is surjective
- The vast majority of maps are not surjective
- **eg:** With N=5, M=3 we have 186 132 664 maps.

# Surjectivity

- We can enforce that each map is surjective
- The vast majority of maps are not surjective
- **eg:** With N=5, M=3 we have 186 132 664 maps.
- Only 86 094 are surjective.

# Results so far

# Results so far

- Ran up to N=4, M=2 without surjective generation

# Results so far

- Ran up to N=4, M=2 without surjective generation
- No Maps found yet

# Results so far

- Ran up to N=4, M=2 without surjective generation
- No Maps found yet

Thank you.

`https://github.com/Nomadluap/TriodEnumeration`