On the Graph Coloring Problem and Its Generalizations

ThanhVu H. Nguyen
Advisor: Dr. Thang N. Bui

Master Thesis in Computer Science
Penn State Harrisburg

November 8, 2006

Outline

1. Problems Definition
2. ABGC Algorithm
3. Results
4. Conclusion

Outline

1. Problems Definition
2. ABGC Algorithm
3. Results
4. Conclusion

The Classic Graph Coloring Problem (GCP)

The Classic Graph Coloring Problem (GCP)

Input

Undirected graph $G = (V, E)$

The Classic Graph Coloring Problem (GCP)

Input

Undirected graph $G = (V, E)$

Output

A minimum coloring of G

- That is, each vertex of G is assigned a color (an integer) such that adjacent vertices have different colors, and the number of colors is minimized.

An Example of GCP

An Example of GCP

A graph with **6** vertices and **7** edges.

An Example of GCP

Constraint: Assign colors to vertices so that adjacent vertices do not have the same color.

An Example of GCP

Constraint: Assign colors to vertices so that adjacent vertices do not have the same color.

An Example of GCP

Constraint: Assign colors to vertices so that adjacent vertices do not have the same color.

An Example of GCP

Constraint: Assign colors to vertices so that adjacent vertices do not have the same color.

An Example of GCP

Constraint: Assign colors to vertices so that adjacent vertices do not have the same color.

An Example of GCP

Objective: Minimize the number of colors used.

The Generalizations of Graph Coloring

The Generalizations of Graph Coloring

- Additional Constraints

The Generalizations of Graph Coloring

- Additional Constraints
- Similar objective: Minimize the number of colors used.

The Generalizations of Graph Coloring

- Additional Constraints
- Similar objective: Minimize the number of colors used.
- 3 generalizations are considered:

The Generalizations of Graph Coloring

- Additional Constraints
- Similar objective: Minimize the number of colors used.
- 3 generalizations are considered:
  - Bandwidth Coloring

The Generalizations of Graph Coloring

- Additional Constraints
- Similar objective: Minimize the number of colors used.
- 3 generalizations are considered:
  - Bandwidth Coloring
  - Multi Coloring

The Generalizations of Graph Coloring

- Additional Constraints
- Similar objective: Minimize the number of colors used.
- 3 generalizations are considered:
  - Bandwidth Coloring
  - Multi Coloring
  - Bandwidth Multi Coloring

The Bandwidth Coloring Problem (BCP)

The Bandwidth Coloring Problem (BCP)

Input

Undirected graph $G = (V, E)$ and positive edge weight function d.

The Bandwidth Coloring Problem (BCP)

Input

Undirected graph $G = (V, E)$ and positive edge weight function d.

Output

A minimum coloring such that the colors of adjacent vertices must differ by at least the weight of the edge connecting them.

The Bandwidth Coloring Problem (BCP)

Input

Undirected graph $G = (V, E)$ and positive edge weight function d.

Output

A minimum coloring such that the colors of adjacent vertices must differ by at least the weight of the edge connecting them.

$BCP = GCP$ if $d(u, v) = 1, \forall (u, v) \in E$.

An Example of BCP

An Example of BCP

An 8-coloring example.

An Example of BCP

Each edge has a **weight**.

An Example of BCP

$|3 - 8| \geq 2$

Constraint: The colors of two adjacent vertices must differ from each other by at least the weight of the shared edge connecting them.

An Example of BCP

Observation: Not all 8 colors are used in this 8-coloring.

The Multi Coloring Problem (MCP)

The Multi Coloring Problem (MCP)

Input

Undirected graph $G = (V, E)$ and positive vertex weight function w.

The Multi Coloring Problem (MCP)

Input

Undirected graph $G = (V, E)$ and positive vertex weight function w.

Output

Each vertex u is assigned a set of $w(u)$ distinct colors such that the color sets of any two adjacent vertices are disjoint, and the number of colors used is minimum.

The Multi Coloring Problem (MCP)

Input

Undirected graph $G = (V, E)$ and positive vertex weight function w.

Output

Each vertex u is assigned a set of $w(u)$ distinct colors such that the color sets of any two adjacent vertices are disjoint, and the number of colors used is minimum.

$MCP = GCP$ if $w(u) = 1, \forall u \in V$.

An Example of MCP

An Example of MCP

An 8-coloring example.

An Example of MCP

Each vertex has a **weight**.

An Example of MCP

$$|\{3,4,5,6,7\}| = 5$$

Constraints:

- A set of distinct colors is assigned to each vertex with the size of the set equals to the weight of that vertex.

An Example of MCP

$$\{3,4,5,6,7\} \cap \{1,2\} = \emptyset$$

Constraints:

- A set of distinct colors is assigned to each vertex with the size of the set equals to the weight of that vertex.
- The color sets of any two adjacent vertices are disjoint.

The Bandwidth Multi Coloring Problem (BMCP)

The Bandwidth Multi Coloring Problem (BMCP)

Input

Undirected graph $G = (V, E)$, positive edge weight function d and positive vertex weight function w.

The Bandwidth Multi Coloring Problem (BMCP)

Input

Undirected graph $G = (V, E)$, positive edge weight function d and positive vertex weight function w.

Output

Similar to the Multi Coloring problem with an additional constraint as in the Bandwidth Coloring problem.

- A minimum coloring of the graph such that if u and v are adjacent vertices, then each color in the color set of u must differ from each color in the color set of v by at least $d(u, v)$.

The Bandwidth Multi Coloring Problem (BMCP)

Input

Undirected graph $G = (V, E)$, positive edge weight function d and positive vertex weight function w.

Output

Similar to the Multi Coloring problem with an additional constraint as in the Bandwidth Coloring problem.

- A minimum coloring of the graph such that if u and v are adjacent vertices, then each color in the color set of u must differ from each color in the color set of v by at least $d(u, v)$.

$BMCP = GCP$ if
$d(u, v) = 1, \forall (u, v) \in E$ and $w(u) = 1, \forall u \in V$.

An Example of BMCP

An Example of BMCP

13-coloring example.

An Example of BMCP

$$|\{6,7,8,9,10\}| = 5$$

Constraints:

- A set of of distinct colors is assigned to each vertex with the size of the set equals the weight of the vertex.

An Example of BMCP

Constraints:

- A set of of distinct colors is assigned to each vertex with the size of the set equals the weight of the vertex.

An Example of BMCP

Each color in $\{9, 11, 13\}$ differs from every other colors in the set by at least 2.

Constraints:

- A set of of distinct colors is assigned to each vertex with the size of the set equals the weight of the vertex.

An Example of BMCP

Each color in $\{1, 2\}$ differs from each color in $\{6, 7, 8, 9, 10\}$ by at least 4.

Constraints:

- A set of of distinct colors is assigned to each vertex with the size of the set equals the weight of the vertex.
- Any two colors of any two adjacent vertices must differ by the weight of the edge connecting them.

Some Applications for the GCP

Some Applications for the GCP

- Scheduling (classrooms, jobs)
- CPU register allocation
- Frequency Assignments in Telecommunication

An Application for GCP: Course Scheduling

An Application for GCP: Course Scheduling

Chem

Math

Art

Music

Bio

Econ

An Application for GCP: Course Scheduling

Cannot be offered at the same time

- Math – Bio
- Math – Chem
- Bio – Music
- Bio – Econ
- Music – Chem
- Music – Econ
- Chem – Art
- Art – Econ

An Application for GCP: Course Scheduling

Cannot be offered at the same time

- Math – Bio
- Math – Chem
- Bio – Music
- Bio – Econ
- Music – Chem
- Music – Econ
- Chem – Art
- Art – Econ

An Application for GCP: Course Scheduling

Cannot be offered at the same time

- Math – Bio
- Math – Chem
- Bio – Music
- Bio – Econ
- Music – Chem
- Music – Econ
- Chem – Art
- Art – Econ

An Application for GCP: Course Scheduling

Cannot be offered at the same time

Math – Bio
Math – Chem
Bio – Music
Bio – Econ
Music – Chem
Music – Econ
Chem – Art
Art – Econ

An Application for GCP: Course Scheduling

Cannot be offered at the same time

- Math – Bio
- Math – Chem
- Bio – Music
- Bio – Econ
- Music – Chem
- Music – Econ
- Chem – Art
- Art – Econ

An Application for GCP: Course Scheduling

Goal : minimizing the number of colors (periods) used

An Application for GCP: Course Scheduling

Some Applications for the Generalizations

Some Applications for the Generalizations

- **Cell phone network**: different cells require frequencies that must be at some distance apart in order to minimize interferences.
  This can be modeled by the Bandwidth Coloring problem.

Some Applications for the Generalizations

- **Cell phone network**: different cells require frequencies that must be at some distance apart in order to minimize interferences.
  This can be modeled by the Bandwidth Coloring problem.
- If multiple frequencies are assigned to a cell then this problem can be modeled by the Bandwidth Multi Coloring problem.

Complexity

Complexity

Worst-Case Complexity

- GCP is a classic $\mathcal{NP}$-hard problem.

Complexity

Worst-Case Complexity

- GCP is a classic $\mathcal{NP}$-hard problem.
- The generalizations are $\mathcal{NP}$-hard (extensions to GCP).

Complexity

Worst-Case Complexity

- GCP is a classic $\mathcal{NP}$-hard problem.
- The generalizations are $\mathcal{NP}$-hard (extensions to GCP).

Complexity

Worst-Case Complexity

- GCP is a classic $\mathcal{NP}$-hard problem.
- The generalizations are $\mathcal{NP}$-hard (extensions to GCP).

Approximation Complexity

- For GCP, even finding approximation solution is difficult.

Complexity

Worst-Case Complexity

- GCP is a classic $\mathcal{NP}$-hard problem.
- The generalizations are $\mathcal{NP}$-hard (extensions to GCP).

Approximation Complexity

- For GCP, even finding approximation solution is difficult.
  - Cannot be approximated within $\alpha = |V|^{1/7-\epsilon}$ in polynomial time, for any $\epsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$.

Complexity

Worst-Case Complexity

- GCP is a classic $\mathcal{NP}$-hard problem.
- The generalizations are $\mathcal{NP}$-hard (extensions to GCP).

Approximation Complexity

- For GCP, even finding approximation solution is difficult.
  - Cannot be approximated within $\alpha = |V|^{1/7-\epsilon}$ in polynomial time, for any $\epsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$.
- Not much is known about the approximation complexity for the generalizations.

Previous Heuristic Approaches

Previous Heuristic Approaches

For GCP

- Tabu search

Previous Heuristic Approaches

For GCP

- Tabu search
- Local search

Previous Heuristic Approaches

For GCP

- Tabu search
- Local search
- Genetic algorithms

Previous Heuristic Approaches

For GCP

- Tabu search
- Local search
- Genetic algorithms
- Ant-based algorithms

Previous Heuristic Approaches

For GCP

- Tabu search
- Local search
- Genetic algorithms
- Ant-based algorithms

Previous Heuristic Approaches

For GCP

- Tabu search
- Local search
- Genetic algorithms
- Ant-based algorithms

For the Generalizations

Previous Heuristic Approaches

For GCP

- Tabu search
- Local search
- Genetic algorithms
- Ant-based algorithms

For the Generalizations

- Local search and constraint propagation

Previous Heuristic Approaches

For GCP

- Tabu search
- Local search
- Genetic algorithms
- Ant-based algorithms

For the Generalizations

- Local search and constraint propagation
- Squeaky wheel optimization (SWO)

Outline

1. Problems Definition
2. ABGC Algorithm
3. Results
4. Conclusion

Characteristics of Agent-based Algorithm

Characteristics of Agent-based Algorithm

Agent-based algorithm:

- is an Evolutionary Computing (EC) approach,

Characteristics of Agent-based Algorithm

Agent-based algorithm:

- is an Evolutionary Computing (EC) approach,
- uses agents to explore the search space to find solution,

Characteristics of Agent-based Algorithm

Agent-based algorithm:

- is an Evolutionary Computing (EC) approach,
- uses agents to explore the search space to find solution,
- improves the quality of the solution incrementally,

Characteristics of Agent-based Algorithm

Agent-based algorithm:

- is an Evolutionary Computing (EC) approach,
- uses agents to explore the search space to find solution,
- improves the quality of the solution incrementally,
- employs local optimization to fine-tune the solutions,

Characteristics of Agent-based Algorithm

Agent-based algorithm:

- is an Evolutionary Computing (EC) approach,
- uses agents to explore the search space to find solution,
- improves the quality of the solution incrementally,
- employs local optimization to fine-tune the solutions,
- uses perturbation to escape from local optima.

An Agent-based Algorithm for Graph Coloring

An Agent-based Algorithm for Graph Coloring

ABGC

- The algorithm features agents, exploring and coloring the graph.

An Agent-based Algorithm for Graph Coloring

ABGC

- The algorithm features agents, exploring and coloring the graph.
- Each agent works on a local portion of the graph. Individual agents **do not** build complete solutions.

An Agent-based Algorithm for Graph Coloring

ABGC

- The algorithm features agents, exploring and coloring the graph.
- Each agent works on a local portion of the graph. Individual agents **do not** build complete solutions.
- Additional techniques to help the agents:

An Agent-based Algorithm for Graph Coloring

ABGC

- The algorithm features agents, exploring and coloring the graph.
- Each agent works on a local portion of the graph. Individual agents **do not** build complete solutions.
- Additional techniques to help the agents:
  - Tabu list, greedy-based local optimization, perturbations, etc.

Algorithm Outline

Algorithm Outline

Initialization

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Until some criteria are met

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Until some criteria are met

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.

Until some criteria are met

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.

Until some criteria are met

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.
- $k \longleftarrow$ the highest color value in the bestColoring – 1

Until some criteria are met

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.
- $k \longleftarrow$ the highest color value in the bestColoring – 1

Jolt operation

- If k has not changed for a while, *perturb* the current coloring.

Until some criteria are met

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.
- $k \longleftarrow$ the highest color value in the bestColoring – 1

Jolt operation

- If k has not changed for a while, *perturb* the current coloring.

Until some criteria are met

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.
- $k \longleftarrow$ the highest color value in the bestColoring – 1

Jolt operation

- If k has not changed for a while, *perturb* the current coloring.

Until some criteria are met

Return the *best* coloring found

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.
- $k \longleftarrow$ the highest color value in the bestColoring – 1

Jolt operation

- If k has not changed for a while, *perturb* the current coloring.

Until some criteria are met

Return the *best* coloring found

Input Processing

Input Processing

Pre-processing

- If the input is an instance of BCP (or GCP), no processing is needed.

Input Processing

Pre-processing

- If the input is an instance of BCP (or GCP), no processing is needed.
- If the input is an instance of MCP or BMCP, it is transformed into an instance of BCP.

Example: Transforming BMCP to BCP

Example: Transforming BMCP to BCP

A **BMCP** instance

Example: Transforming BMCP to BCP

Each vertex a in the original graph

Example: Transforming BMCP to BCP

Each vertex a in the original graph is transformed into a **clique** C_a of size $w(a)$,

Example: Transforming BMCP to BCP

Each vertex a in the original graph is transformed into a **clique** C_a of size $w(a)$, and all edges in C_a have weights equal to the self-loop weight, $d(a, a)$.

Example: Transforming BMCP to BCP

Each vertex a in the original graph is transformed into a **clique** C_a of size $w(a)$, and all edges in C_a have weights equal to the self-loop weight, $d(a, a)$.

Example: Transforming BMCP to BCP

Each vertex a in the original graph is transformed into a **clique** C_a of size $w(a)$, and all edges in C_a have weights equal to the self-loop weight, $d(a, a)$.

Example: Transforming BMCP to BCP

Each vertex a in the original graph is transformed into a **clique** C_a of size $w(a)$, and all edges in C_a have weights equal to the self-loop weight, $d(a, a)$.

Example: Transforming BMCP to BCP

If a, b are adjacent in the original graph,

Example: Transforming BMCP to BCP

If a, b are adjacent in the original graph, then each vertex in C_a is connected to each vertex in C_b

Example: Transforming BMCP to BCP

If a, b are adjacent in the original graph, then each vertex in C_a is connected to each vertex in C_b with edge weight $d(a, b)$.

Example: Transforming BMCP to BCP

If a, b are adjacent in the original graph, then each vertex in C_a is connected to each vertex in C_b with edge weight $d(a, b)$.

Example: Transforming BMCP to BCP

The completed transformation.

Initial Coloring

Initial Coloring

Iterative Greedy algorithm

- **Objective**: Find an initial valid coloring quickly.

Initial Coloring

Iterative Greedy algorithm

- **Objective**: Find an initial valid coloring quickly.

How it works

Initial Coloring

Iterative Greedy algorithm

- **Objective**: Find an initial valid coloring quickly.

How it works

- Find 21 colorings of the graph and keep the best coloring found.

Initial Coloring

Iterative Greedy algorithm

- **Objective**: Find an initial valid coloring quickly.

How it works

- Find 21 colorings of the graph and keep the best coloring found.
- Do one greedy coloring

Initial Coloring

Iterative Greedy algorithm

- **Objective**: Find an initial valid coloring quickly.

How it works

- Find 21 colorings of the graph and keep the best coloring found.
- Do one greedy coloring
  - Vertices are considered in decreasing order of degree.

Initial Coloring

Iterative Greedy algorithm

- **Objective**: Find an initial valid coloring quickly.

How it works

- Find 21 colorings of the graph and keep the best coloring found.
- Do one greedy coloring
  - Vertices are considered in decreasing order of degree.
  - When a vertex is considered, it is colored with the smallest feasible color.

Initial Coloring

Iterative Greedy algorithm

- **Objective**: Find an initial valid coloring quickly.

How it works

- Find 21 colorings of the graph and keep the best coloring found.
- Do one greedy coloring
  - Vertices are considered in decreasing order of degree.
  - When a vertex is considered, it is colored with the smallest feasible color.
- Do twenty (20) random colorings:

Initial Coloring

Iterative Greedy algorithm

- **Objective**: Find an initial valid coloring quickly.

How it works

- Find 21 colorings of the graph and keep the best coloring found.
- Do one greedy coloring
  - Vertices are considered in decreasing order of degree.
  - When a vertex is considered, it is colored with the smallest feasible color.
- Do twenty (20) random colorings:
  - Same as above but vertices are considered in a random order.

Initial Coloring

Iterative Greedy algorithm

- **Objective**: Find an initial valid coloring quickly.

How it works

- Find 21 colorings of the graph and keep the best coloring found.
- Do one greedy coloring
  - Vertices are considered in decreasing order of degree.
  - When a vertex is considered, it is colored with the smallest feasible color.
- Do twenty (20) random colorings:
  - Same as above but vertices are considered in a random order.
- Return the best of these 21 colorings.

Initial Coloring

Initial Coloring

- Iterative Greedy algorithm returns a coloring with k as the highest color value.

Initial Coloring

- Iterative Greedy algorithm returns a coloring with k as the highest color value.
- Attempt a new goal for the agents, $k \longleftarrow k - 1$

Initial Coloring

- Iterative Greedy algorithm returns a coloring with k as the highest color value.
- Attempt a new goal for the agents, $k \longleftarrow k - 1$
  - k is the # of colors the agents can use to color the graph with.

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.
- $k \longleftarrow$ the highest color value in the bestColoring – 1

Jolt operation

- If k has not changed for a while, *perturb* the current coloring.

Until some criteria are met

Return the *best* coloring found

Exploration: General Ideas

Each agent executes the following operations:

Exploration: General Ideas

Each agent executes the following operations:

- Place itself on a vertex with *maximum* conflict.

Exploration: General Ideas

Each agent executes the following operations:

- Place itself on a vertex with *maximum* **conflict**.

Definition

The **conflict** at a vertex v denotes the number of vertices adjacent to v having inconsistent color set with the coloring of v.

Exploration: General Ideas

Each agent executes the following operations:

- Place itself on a vertex with *maximum* conflict.
- Make a number of moves.

Exploration: General Ideas

Each agent executes the following operations:

- Place itself on a vertex with *maximum* conflict.
- Make a number of moves.
- Color the vertices.

Exploration: How an Agent Moves

Exploration: How an Agent Moves

A move consists of two steps
(assume an agent is at vertex u)

Exploration: How an Agent Moves

A move consists of two steps
(assume an agent is at vertex u)

- Step 1: Randomly select a vertex v among the vertices adjacent to u. Go to v.

Exploration: How an Agent Moves

A move consists of two steps
(assume an agent is at vertex u)

- Step 1: Randomly select a vertex v among the vertices adjacent to u. Go to v.
- Step 2: Select the vertex with the highest conflict, say w, among the vertices adjacent to v. Go to w.

Exploration: How an Agent Moves

A move consists of two steps
(assume an agent is at vertex u)

- Step 1: Randomly select a vertex v among the vertices adjacent to u. Go to v.
- Step 2: Select the vertex with the highest conflict, say w, among the vertices adjacent to v. Go to w.
  - The agent colors w, then adds it to the **Tabu list** of the agent. (Each agent has its own fixed sized Tabu List)

Exploration: How an Agent Colors a Vertex

Exploration: How an Agent Colors a Vertex

- **Objective**: each agent colors or re-colors a vertex so that the conflict at that vertex is zero, if possible.

Exploration: How an Agent Colors a Vertex

- **Objective**: each agent colors or re-colors a vertex so that the conflict at that vertex is zero, if possible.
- Color decision is based on local information, no global knowledge.

Exploration: How an Agent Colors a Vertex

Exploration: How an Agent Colors a Vertex

To color a vertex, an agent considers the three sets

1. *AvailableColors*: set of potential colors that the agent could use

Exploration: How an Agent Colors a Vertex

To color a vertex, an agent considers the three sets

1. *AvailableColors*: set of potential colors that the agent could use
2. *ForbiddenColors*: set of colors that *cannot* be used to color the current vertex as they will create conflict

Exploration: How an Agent Colors a Vertex

To color a vertex, an agent considers the three sets

1. *AvailableColors*: set of potential colors that the agent could use
2. *ForbiddenColors*: set of colors that *cannot* be used to color the current vertex as they will create conflict
3. $EligibleColors = AvailableColors - ForbiddenColors$

Exploration: How an Agent Colors a Vertex

To color a vertex, an agent considers the three sets

1. *AvailableColors*: set of potential colors that the agent could use
2. *ForbiddenColors*: set of colors that *cannot* be used to color the current vertex as they will create conflict
3. *EligibleColors* = *AvailableColors* – *ForbiddenColors*
   - This is usually a union of intervals, e.g., $\{6, 7, 8, 9, 12, 13, 14, 15\} = [6 \ldots 9] \cup [12 \ldots 15]$

Exploration: How an Agent Colors a Vertex

Exploration: How an Agent Colors a Vertex

If EligibleColors $= \emptyset$

Exploration: How an Agent Colors a Vertex

If EligibleColors $= \emptyset$

Choose the color that has the **fewest** conflicts with the adjacent vertices.

Exploration: How an Agent Colors a Vertex

If EligibleColors $= \emptyset$

Choose the color that has the **fewest** conflicts with the adjacent vertices.

If EligibleColors $\neq \emptyset$

Exploration: How an Agent Colors a Vertex

If EligibleColors $= \emptyset$

Choose the color that has the **fewest** conflicts with the adjacent vertices.

If EligibleColors $\neq \emptyset$

Choose the color that is the *median* of the *largest* interval.

Exploration: How an Agent Colors a Vertex

If EligibleColors $= \emptyset$

Choose the color that has the **fewest** conflicts with the adjacent vertices.

If EligibleColors $\neq \emptyset$

Choose the color that is the *median* of the *largest* interval.

- **Idea**: Give neighbor vertices more room to meet their constraints.

Example: How an Agent Colors

Example: How an Agent Colors

Example: How an Agent Colors

Example: How an Agent Colors

Example: How an Agent Colors

Example: How an Agent Colors

- $AvailableColors = \{1, \ldots, 27\}$

Example: How an Agent Colors

Example: How an Agent Colors

Example: How an Agent Colors

- $AvailableColors = \{1, \ldots, 27\}$
- $ForbiddenColors = \{4, \ldots, 8, 11, \ldots, 23, 27\}$

Example: How an Agent Colors

- $AvailableColors = \{1, \ldots, 27\}$
- $ForbiddenColors = \{4, \ldots, 8, 11, \ldots, 23, 27\}$
- $EligibleColors = AvailableColors - ForbiddenColors$
  $= [1 \ldots 3] \cup [9\ 10] \cup [24\ 25\ 26]$

Example: How an Agent Colors

- $AvailableColors = \{1, \ldots, 27\}$
- $ForbiddenColors = \{4, \ldots, 8, 11, \ldots, 23, 27\}$
- $EligibleColors = AvailableColors - ForbiddenColors$
  $= [1 \ldots 3] \cup [9\ 10] \cup [24\ 25\ 26]$
  - Largest Interval Size: 3

Example: How an Agent Colors

- $AvailableColors = \{1, \ldots, 27\}$
- $ForbiddenColors = \{4, \ldots, 8, 11, \ldots, 23, 27\}$
- $EligibleColors = AvailableColors - ForbiddenColors$
  $= [1 \ldots 3] \cup [9\ 10] \cup [24\ 25\ 26]$
  - Select randomly one of the largest size intervals

Example: How an Agent Colors

- $AvailableColors = \{1, \ldots, 27\}$
- $ForbiddenColors = \{4, \ldots, 8, 11, \ldots, 23, 27\}$
- $EligibleColors = AvailableColors - ForbiddenColors$
  $= [1 \ldots 3] \cup [9\ 10] \cup [24\ 25\ 26]$
  - Pick the median of the selected interval as the color for u

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.
- $k \longleftarrow$ the highest color value in the bestColoring – 1

Jolt operation

- If k has not changed for a while, *perturb* the current coloring.

Until some criteria are met

Return the *best* coloring found

Exploitation: Local Optimization

Exploitation: Local Optimization

Local Optimization

- **Input**: a (valid) coloring for graph G
- **Output**: a different coloring for graph G

Exploitation: Local Optimization

Local Optimization

- **Input**: a (valid) coloring for graph G
- **Output**: a different coloring for graph G

- **How it works**

Exploitation: Local Optimization

Local Optimization

- **Input**: a (valid) coloring for graph G
- **Output**: a different coloring for graph G

- **How it works**
  - Similar to the Iterative Greedy algorithm.

Exploitation: Local Optimization

Local Optimization

- **Input**: a (valid) coloring for graph G
- **Output**: a different coloring for graph G

- **How it works**
  - Similar to the Iterative Greedy algorithm.
  - Sort the vertices to be re-colored in *decreasing* order of the input coloring.

Exploitation: Local Optimization

Local Optimization

- **Input**: a (valid) coloring for graph G
- **Output**: a different coloring for graph G

- **How it works**
  - Similar to the Iterative Greedy algorithm.
  - Sort the vertices to be re-colored in *decreasing* order of the input coloring.
  - Erase all colors from the graph then re-color the vertices using the sorted order.

Exploitation: Local Optimization

Local Optimization

- **Input**: a (valid) coloring for graph G
- **Output**: a different coloring for graph G

- **How it works**
  - Similar to the Iterative Greedy algorithm.
  - Sort the vertices to be re-colored in *decreasing* order of the input coloring.
  - Erase all colors from the graph then re-color the vertices using the sorted order.
- Replace the current coloring with the one returned from Local Opt (if it is better).

Exploitation: Local Optimization

Local Optimization

- **Input**: a (valid) coloring for graph G
- **Output**: a different coloring for graph G

- **How it works**
  - Similar to the Iterative Greedy algorithm.
  - Sort the vertices to be re-colored in *decreasing* order of the input coloring.
  - Erase all colors from the graph then re-color the vertices using the sorted order.
- Replace the current coloring with the one returned from Local Opt (if it is better).
- Attempt a new goal (fewer colors) for the agents.

Exploitation: Local Optimization

Local Optimization

- **Input**: a (valid) coloring for graph G
- **Output**: a different coloring for graph G

- **How it works**
  - Similar to the Iterative Greedy algorithm.
  - Sort the vertices to be re-colored in *decreasing* order of the input coloring.
  - Erase all colors from the graph then re-color the vertices using the sorted order.
- Replace the current coloring with the one returned from Local Opt (if it is better).
- Attempt a new goal (fewer colors) for the agents.
  - $k \longleftarrow$ the highest color value in the current best coloring $-$ 1

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.
- $k \longleftarrow$ the highest color value in the bestColoring – 1

Jolt operation

- If k has not changed for a while, *perturb* the current coloring.

Until some criteria are met

Return the *best* coloring found

Perturbation: Jolt Operation

Perturbation: Jolt Operation

Jolt Operation

- **Objective**: Attempt to escape local optima by perturbing the current coloring.

Perturbation: Jolt Operation

Jolt Operation

- **Objective**: Attempt to escape local optima by perturbing the current coloring.
- **How it works**

Perturbation: Jolt Operation

Jolt Operation

- **Objective**: Attempt to escape local optima by perturbing the current coloring.
- **How it works**
  - Randomly recolor neighbors of the top β % conflicted vertices.

Perturbation: Jolt Operation

Jolt Operation

- **Objective**: Attempt to escape local optima by perturbing the current coloring.
- **How it works**
  - Randomly recolor neighbors of the top β % conflicted vertices.

Perturbation: Jolt Operation

Jolt Operation

- **Objective**: Attempt to escape local optima by perturbing the current coloring.
- **How it works**
  - Randomly recolor neighbors of the top β % conflicted vertices.
- Perturbation is done whenever a period of time has passed without any improvement.

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.
- $k \longleftarrow$ the highest color value in the bestColoring – 1

Jolt operation

- If k has not changed for a while, *perturb* the current coloring.

Until some criteria are met

Return the *best* coloring found

Stopping Criteria

ABGC terminates when

Stopping Criteria

ABGC terminates when

- A number of cycles has passed.

Stopping Criteria

ABGC terminates when

- A number of cycles has passed.

OR

Stopping Criteria

ABGC terminates when

- A number of cycles has passed.

OR

- A number of cycles has passed without any improvement.

Algorithm Outline

Initialization

$k \longleftarrow$ # of available colors based on the Initialization.

Repeat

Exploration

- Each agent moves around a portion of the graph and colors some of the visited vertices using at most k colors.

Exploitation

- A local *optimization* technique is used to improve the coloring done in the exploration phase.
- Keep the *best* coloring found so far.
- $k \longleftarrow$ the highest color value in the bestColoring – 1

Jolt operation

- If k has not changed for a while, *perturb* the current coloring.

Until some criteria are met

Return the *best* coloring found

Outline

1. Problems Definition
2. ABGC Algorithm
3. Results
4. Conclusion

Testing Details

Testing Details

Implementation Details

- Implemented in C++
- Test machine: 3GHz Pentium 4, 2GB of RAM, Linux operating system

Testing Details

Implementation Details

- Implemented in C++
- Test machine: 3GHz Pentium 4, 2GB of RAM, Linux operating system

Benchmark Instances

Testing Details

Implementation Details

- Implemented in C++
- Test machine: 3GHz Pentium 4, 2GB of RAM, Linux operating system

Benchmark Instances

- For the GCP, tested against 119 (DIMACS) instances of various graph types.

Testing Details

Implementation Details

- Implemented in C++
- Test machine: 3GHz Pentium 4, 2GB of RAM, Linux operating system

Benchmark Instances

- For the GCP, tested against 119 (DIMACS) instances of various graph types.
- For the Generalizations, tested against 99 instances created from 33 (DIMACS) graphs for 3 different coloring problems.

Testing Details

Implementation Details

- Implemented in C++
- Test machine: 3GHz Pentium 4, 2GB of RAM, Linux operating system

Benchmark Instances

- For the GCP, tested against 119 (DIMACS) instances of various graph types.
- For the Generalizations, tested against 99 instances created from 33 (DIMACS) graphs for 3 different coloring problems.
- The algorithm is run 100 times for each instance.

Results for GCP

Results for GCP

- Out of **63** instances with best-known bound, ABGC found **57**.

Results for GCP

- Out of **63** instances with best-known bound, ABGC found **57**.
- The results of the other **6** instances are within **1** of the best-known bound.

Results for GCP

- Out of **63** instances with best-known bound, ABGC found **57**.
- The results of the other **6** instances are within **1** of the best-known bound.
- No other algorithms in the literature were tested on such a comprehensive set of graphs (119) comparing to ABGC.

Results for the Generalizations

Results for the Generalizations

Compared against the following algorithms

Results for the Generalizations

Compared against the following algorithms

- Squeaky Wheel Optimization

Results for the Generalizations

Compared against the following algorithms

- **Squeaky Wheel Optimization**
  - **SWO** (all generalizations, Lim et al, 2003)

Results for the Generalizations

Compared against the following algorithms

- **Squeaky Wheel Optimization**
  - **SWO** (all generalizations, Lim et al, 2003)
  - SWO + Tabu Search (**SWO/TS**) (all generalizations, Lim et al, 2005)

Results for the Generalizations

Compared against the following algorithms

- **Squeaky Wheel Optimization**
  - **SWO** (all generalizations, Lim et al, 2003)
  - SWO + Tabu Search (**SWO/TS**) (all generalizations, Lim et al, 2005)
    - Consider only results from SWO/TS since it outperforms SWO.

Results for the Generalizations

Compared against the following algorithms

- **Squeaky Wheel Optimization**
  - **SWO** (all generalizations, Lim et al, 2003)
  - SWO + Tabu Search (**SWO/TS**) (all generalizations, Lim et al, 2005)
    - Consider only results from SWO/TS since it outperforms SWO.
- **Local Search & Constraint Propagation**

Results for the Generalizations

Compared against the following algorithms

- **Squeaky Wheel Optimization**
  - **SWO** (all generalizations, Lim et al, 2003)
  - SWO + Tabu Search (**SWO/TS**) (all generalizations, Lim et al, 2005)
    - Consider only results from SWO/TS since it outperforms SWO.
- **Local Search & Constraint Propagation**
  - **FCNS** (Bandwidth Coloring only, Prestwich, 2002)

Results for the Generalizations

Compared against the following algorithms

- **Squeaky Wheel Optimization**
  - **SWO** (all generalizations, Lim et al, 2003)
  - SWO + Tabu Search (**SWO/TS**) (all generalizations, Lim et al, 2005)
    - Consider only results from SWO/TS since it outperforms SWO.
- **Local Search & Constraint Propagation**
  - **FCNS** (Bandwidth Coloring only, Prestwich, 2002)
  - **SATURN** (Bandwidth Multi Coloring only, Prestwich, 2002)

Results for the Generalizations

Compared against the following algorithms

- **Squeaky Wheel Optimization**
  - **SWO** (all generalizations, Lim et al, 2003)
  - SWO + Tabu Search (**SWO/TS**) (all generalizations, Lim et al, 2005)
    - Consider only results from SWO/TS since it outperforms SWO.
- **Local Search & Constraint Propagation**
  - **FCNS** (Bandwidth Coloring only, Prestwich, 2002)
  - **SATURN** (Bandwidth Multi Coloring only, Prestwich, 2002)
  - There are no results for the Multi Coloring Problem.

Result: Bandwidth Coloring Problem

Result: Bandwidth Coloring Problem

Instance
geom20a
geom20b
geom30
geom30a
geom40a
geom40b
geom50a
geom50b
geom60
geom60a
geom60b
geom70a
geom70b

Result: Bandwidth Coloring Problem

Instance	FCNS
geom20a	20
geom20b	13
geom30	28
geom30a	27
geom40a	37
geom40b	33
geom50a	50
geom50b	35
geom60	33
geom60a	50
geom60b	43
geom70a	62
geom70b	48

Result: Bandwidth Coloring Problem

Instance	FCNS	SWO/TS
geom20a	20	22
geom20b	13	14
geom30	28	29
geom30a	27	32
geom40a	37	38
geom40b	33	34
geom50a	50	52
geom50b	35	38
geom60	33	34
geom60a	50	53
geom60b	43	46
geom70a	62	63
geom70b	48	54

Result: Bandwidth Coloring Problem

Instance	FCNS	SWO/ TS	ABGC
geom20a	20	22	20
geom20b	13	14	13
geom30	28	29	28
geom30a	27	32	27
geom40a	37	38	37
geom40b	33	34	33
geom50a	50	52	50
geom50b	35	38	36
geom60	33	34	33
geom60a	50	53	50
geom60b	43	46	43
geom70a	62	63	62
geom70b	48	54	51

Result: Bandwidth Coloring Problem

Instance	FCNS	SWO/ TS	ABGC	Instance	FCNS	SWO/ TS	ABGC
geom20a	20	22	20	geom80	41	42	41
geom20b	13	14	13	geom80a	63	66	64
geom30	28	29	28	geom80b	61	65	64
geom30a	27	32	27	geom90a	64	69	65
geom40a	37	38	37	geom90b	72	77	74
geom40b	33	34	33	geom100	50	51	50
geom50a	50	52	50	geom100a	70	76	71
geom50b	35	38	36	geom100b	73	83	79
geom60	33	34	33	geom110	50	53	50
geom60a	50	53	50	geom110a	74	82	75
geom60b	43	46	43	geom110b	79	88	83
geom70a	62	63	62	geom120	60	62	59
geom70b	48	54	51	geom120a	84	92	86
				geom120b	87	98	91

Result: Bandwidth Coloring Problem

Instance	FCNS	SWO/ TS	ABGC	Instance	FCNS	SWO/ TS	ABGC
geom20a	20	22	20	geom80	41	42	41
geom20b	13	14	13	geom80a	63	66	64
geom30	28	29	28	geom80b	61	65	64
geom30a	27	32	27	geom90a	64	69	65
geom40a	37	38	37	geom90b	72	77	74
geom40b	33	34	33	geom100	50	51	50
geom50a	50	52	50	geom100a	70	76	71
geom50b	35	38	36	geom100b	73	83	79
geom60	33	34	33	geom110	50	53	50
geom60a	50	53	50	geom110a	74	82	75
geom60b	43	46	43	geom110b	79	88	83
geom70a	62	63	62	geom120	60	62	59
geom70b	48	54	51	geom120a	84	92	86
				geom120b	87	98	91

Result: Bandwidth Coloring Problem

Instance	FCNS	SWO/ TS	ABGC	Instance	FCNS	SWO/ TS	ABGC
geom20a	20	22	20	geom80	41	42	41
geom20b	13	14	13	geom80a	63	66	64
geom30	28	29	28	geom80b	61	65	64
geom30a	27	32	27	geom90a	64	69	65
geom40a	37	38	37	geom90b	72	77	74
geom40b	33	34	33	geom100	50	51	50
geom50a	50	52	50	geom100a	70	76	71
geom50b	35	38	**36**	geom100b	73	83	79
geom60	33	34	33	geom110	50	53	50
geom60a	50	53	50	geom110a	74	82	75
geom60b	43	46	43	geom110b	79	88	83
geom70a	62	63	62	geom120	60	62	59
geom70b	48	54	51	geom120a	84	92	86
				geom120b	87	98	91

Result: Bandwidth Coloring Problem

Instance	FCNS	SWO/ TS	ABGC	Instance	FCNS	SWO/ TS	ABGC
geom20a	20	22	20	geom80	41	42	41
geom20b	13	14	13	geom80a	63	66	64
geom30	28	29	28	geom80b	61	65	64
geom30a	27	32	27	geom90a	64	69	65
geom40a	37	38	37	geom90b	72	77	74
geom40b	33	34	33	geom100	50	51	50
geom50a	50	52	50	geom100a	70	76	71
geom50b	35	38	36	geom100b	73	83	79
geom60	33	34	33	geom110	50	53	50
geom60a	50	53	50	geom110a	74	82	75
geom60b	43	46	43	geom110b	79	88	83
geom70a	62	63	62	geom120	60	62	59
geom70b	48	54	51	geom120a	84	92	86
				geom120b	87	98	91

Result: Bandwidth Coloring Problem

Instance	FCNS	SWO/ TS	ABGC	Instance	FCNS	SWO/ TS	ABGC
geom20a	20	22	20	geom80	41	42	41
geom20b	13	14	13	geom80a	63	66	64
geom30	28	29	28	geom80b	61	65	64
geom30a	27	32	27	geom90a	64	69	65
geom40a	37	38	37	geom90b	72	77	74
geom40b	33	34	33	geom100	50	51	50
geom50a	50	52	50	geom100a	70	76	71
geom50b	35	38	36	geom100b	73	83	79
geom60	33	34	33	geom110	50	53	50
geom60a	50	53	50	geom110a	74	82	75
geom60b	43	46	43	geom110b	79	88	83
geom70a	62	63	62	geom120	60	62	59
geom70b	48	54	51	geom120a	84	92	86
				geom120b	87	98	91

Result: Multi Coloring Problem

Result: Multi Coloring Problem

- ABGC tied with SWO based algorithms in all instances.

Result: Multi Coloring Problem

- ABGC tied with SWO based algorithms in all instances.
- There are no results from the Local Search & Constraint Propagation based algorithms for the Multi Coloring problem.

Result: Bandwidth Multi Coloring Problem

Result: Bandwidth Multi Coloring Problem

Instance	SATURN	SWO/ TS	ABGC	Instance	SATURN	SWO/ TS	ABGC
geom20	159	149	149	geom80	–	383	382
geom20a	175	169	169	geom80a	–	379	367
geom30	168	160	160	geom80b	152	141	139
geom30a	235	209	210	geom90	–	332	332
geom30b	79	77	77	geom90a	–	377	378
geom40	189	167	167	geom90b	–	157	150
geom40a	260	213	214	geom100	–	404	405
geom40b	80	74	74	geom100a	–	459	440
geom50	257	224	224	geom100b	–	170	164
geom50a	395	318	317	geom110	–	383	378
geom50b	89	87	85	geom110a	–	494	487
geom60	279	258	258	geom110b	–	206	208
geom60a	–	358	357	geom120	–	402	398
geom60b	128	116	117	geom120a	–	556	548
geom70	310	273	267	geom120b	–	199	198
geom70a	–	469	470				
geom70b	133	121	121				

Result: Bandwidth Multi Coloring Problem

Instance	SATURN	SWO/ TS	ABGC	Instance	SATURN	SWO/ TS	ABGC
geom20	159	149	149	geom80	–	383	382
geom20a	175	169	169	geom80a	–	379	367
geom30	168	160	160	geom80b	152	141	139
geom30a	235	209	210	geom90	–	332	332
geom30b	79	77	77	geom90a	–	377	378
geom40	189	167	167	geom90b	–	157	150
geom40a	260	213	214	geom100	–	404	405
geom40b	80	74	74	geom100a	–	459	440
geom50	257	224	224	geom100b	–	170	164
geom50a	395	318	317	geom110	–	383	378
geom50b	89	87	85	geom110a	–	494	487
geom60	279	258	258	geom110b	–	206	208
geom60a	–	358	357	geom120	–	402	398
geom60b	128	116	117	geom120a	–	556	548
geom70	310	273	267	geom120b	–	199	198
geom70a	–	469	470				
geom70b	133	121	121				

– indicates unavailable result

Outline

1. Problems Definition
2. ABGC Algorithm
3. Results
4. Conclusion

Conclusion

Conclusion

ABGC

- Agent-based, hybrids with other techniques (Tabu List, Greedy Local optimization, etc)

Conclusion

ABGC

- Agent-based, hybrids with other techniques (Tabu List, Greedy Local optimization, etc)
- Produced competitive results

Conclusion

ABGC

- Agent-based, hybrids with other techniques (Tabu List, Greedy Local optimization, etc)
- Produced competitive results
- **Generality**: applicable to the classical Graph Coloring problem as well as its (three) generalizations.

Conclusion

ABGC

- Agent-based, hybrids with other techniques (Tabu List, Greedy Local optimization, etc)
- Produced competitive results
- **Generality**: applicable to the classical Graph Coloring problem as well as its (three) generalizations.

Conclusion

ABGC

- Agent-based, hybrids with other techniques (Tabu List, Greedy Local optimization, etc)
- Produced competitive results
- **Generality**: applicable to the classical Graph Coloring problem as well as its (three) generalizations.

Future Work

- Use pheromone
- Explore parallel implementation