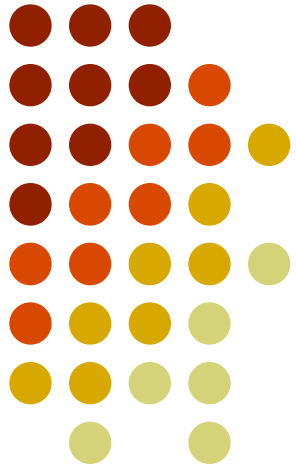
An Agent-Based Algorithm for Generalized Graph Colorings

Thang N. Bui and ThanhVu H. Nguyen
Computer Science Program
Penn State Harrisburg

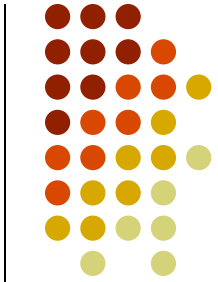


Outline

- Problems Definition
- ABGC Algorithm
- Results
- Conclusion

Outline

- **Problems Definition**
- ABGC Algorithm
- Results
- Conclusion



Problems Definition

- **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 total number of colors used is minimum.

- **The Generalizations**
  - Similar objective: Minimize the number of colors used.
  - Additional constraints
  - **3 generalizations are considered:**
    - Bandwidth Coloring
    - Multi Coloring
    - Bandwidth Multi Coloring

Bandwidth Coloring Problem (BCP)

- **Input**: Undirected graph *G=(V,E)* and edge weight function *d*
- **Output**: Similar to GCP with the additional constraint 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, for all (*u*,*v)* in *E*

Multi Coloring Problem (MCP)

- **Input**: Undirected Graph *G=(V,E)* and 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 total number of colors used is minimum
- MCP = GCP if *w(u)* = 1 for all *u* in *G*

Bandwidth Multi Coloring Problem (BMCP)

- **Input**: Undirected graph *G=(V,E)*, edge weight function *d* and vertex weight function *w*

- **Output**: Similar to Multi Coloring problem with an additional constraint as in the Bandwidth Coloring problem
  - 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 and *w(u)* = 1 for all *u*, *v* in *G*

Applications

- **Some applications for the Graph Coloring problem**
  - Scheduling (classrooms, jobs)
  - CPU register allocation
  - Air traffic flow control

- **Some applications for the Generalized Graph Coloring problems**

  - **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 of Generalized Graph Colorings

- NP-Hard
  - These generalizations are extensions to the Graph Coloring problem (GCP), a classic NP-Hard problem.

- Not much is known about the approximation complexity for the generalizations
  - For GCP, it is difficult to approximate

Previous heuristic approaches

- **For GCP**
  - Tabu search, local search, genetic and ant based algorithms
- **For the Generalizations**
  - Local search and constraint propagation
  - Squeaky wheel optimization (SWO)

Outline

- Problems Definition
- **ABGC Algorithm**
- Results
- Conclusion

An Agent Based Algorithm For Graph Coloring (ABGC)

- The algorithm features agents (or ants), 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 (to avoid local optima)

ABGC Algorithm Outline

Initialization

$k \leftarrow$ # of available colors

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.

$k \leftarrow$ # of colors 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

ABGC Algorithm Outline

Initialization

k ← # of available colors

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 ← # of colors 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

Initialization: Input Processing

- **Input 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.

Initialization: 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

Initialization: Initial Coloring

- Iterative Greedy algorithm returns a coloring with ***k*** colors.

- Attempt a new goal for the agents, ***k*** ← ***k*** − 1

The number of colors the agents can use to color the Graph with.

ABGC Algorithm Outline

Initialization

k ← # of available colors

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 ← # of colors 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 of the agents executes the following sequence of operations:
  - Place itself on a vertex with *maximum conflict*

Exploration: General Ideas

- Each of the agents executes the following sequence of operations:
  - Place itself on a vertex with *maximum conflict*

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

Exploration: General Ideas

- Each of the agents executes the following sequence of operations:
  - Place itself on a vertex with *maximum conflict*
  - Make a number of moves
  - Color the vertices

Exploration: How an Agent Moves

- Assume an agent is at vertex *u*. A move consists of two steps
  - **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 agent's Tabu list. (Each agent has its own fixed sized Tabu List)

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

- To color a vertex, an agent considers the following three sets:
  - AvailableColors: set of potential colors that the agent could use
  - ForbiddenColors: set of colors that cannot be used to color the current vertex as they will create conflict
  - EligibleColors = AvailableColors — ForbiddenColors

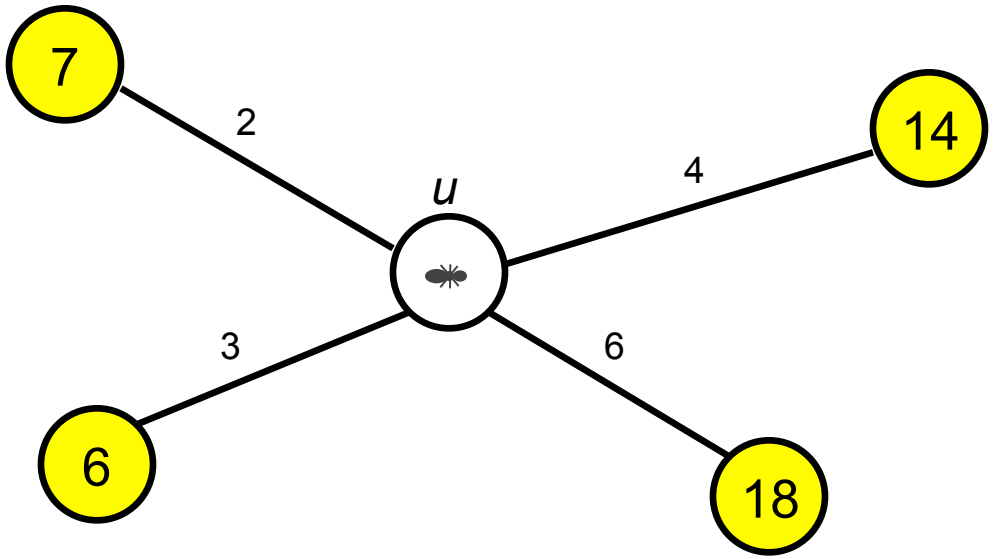
This is usually a union of intervals
e.g., {6, 7, 8, 9, 12,13, 14, 15}
= [6 … 9] U [12 … 15]

Exploration: How an Agent Colors a Vertex

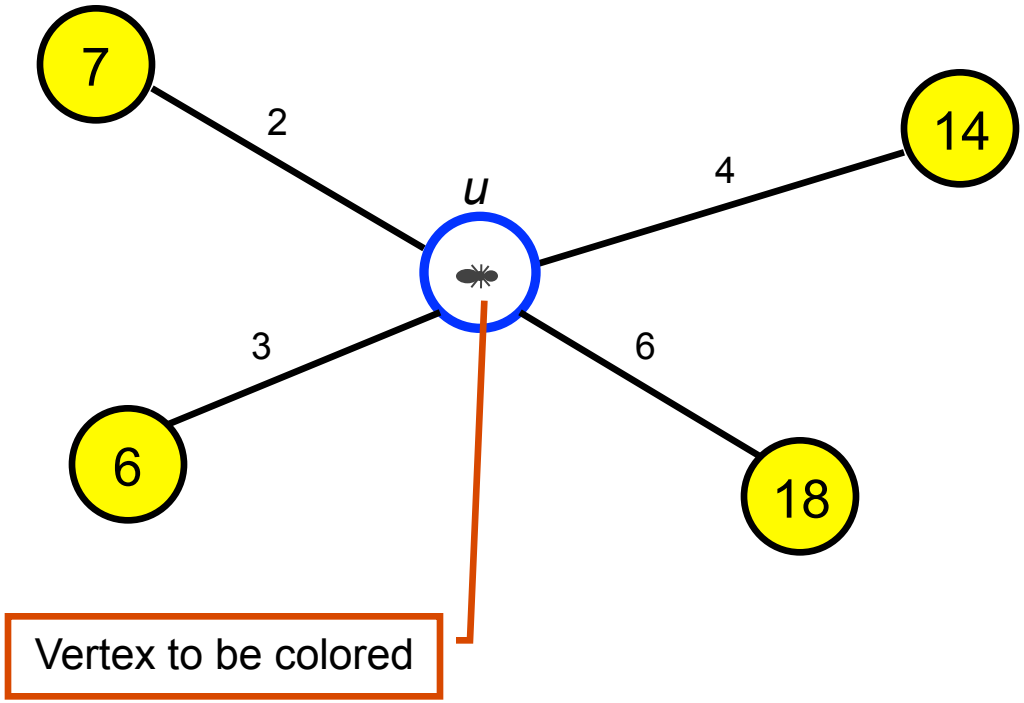
- If |EligibleColors| = 0
  - Choose the color that has the fewest conflicts with the adjacent vertices

- If |EligibleColors| > 1
  - Choose the color that is the median of the largest interval
    - **Idea**: Give neighbor vertices more room to meet their constraints

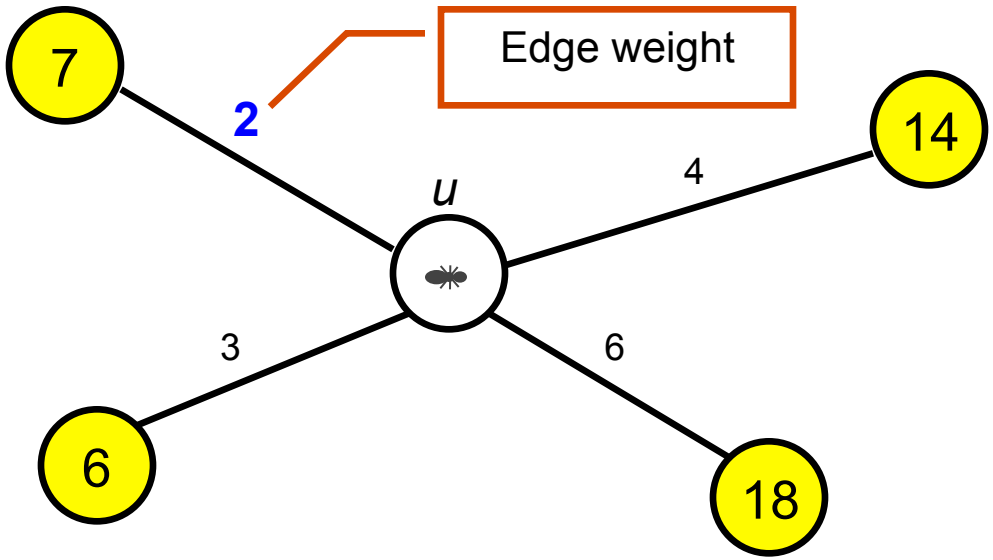
Example: How An Agent Colors



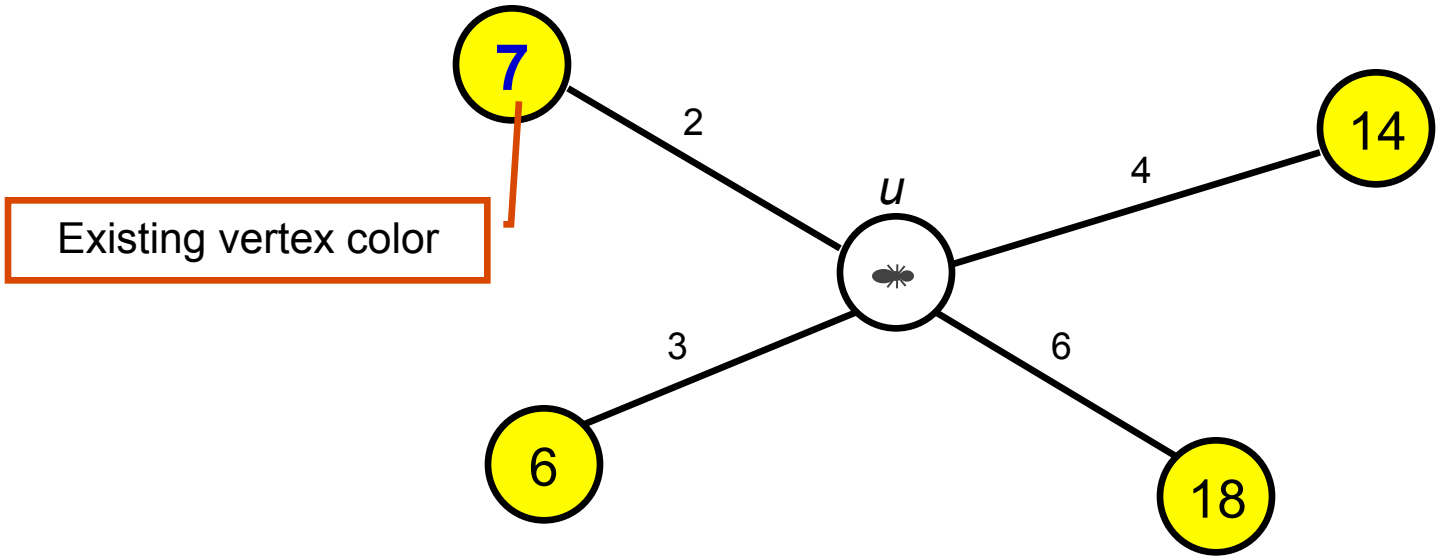
Example: How An Agent Colors



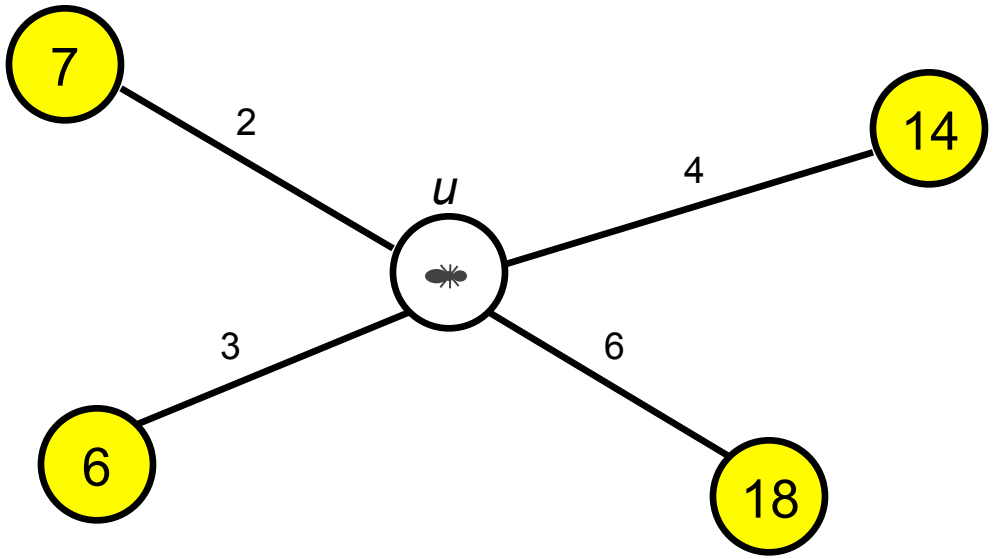
Example: How An Agent Colors



Example: How An Agent Colors

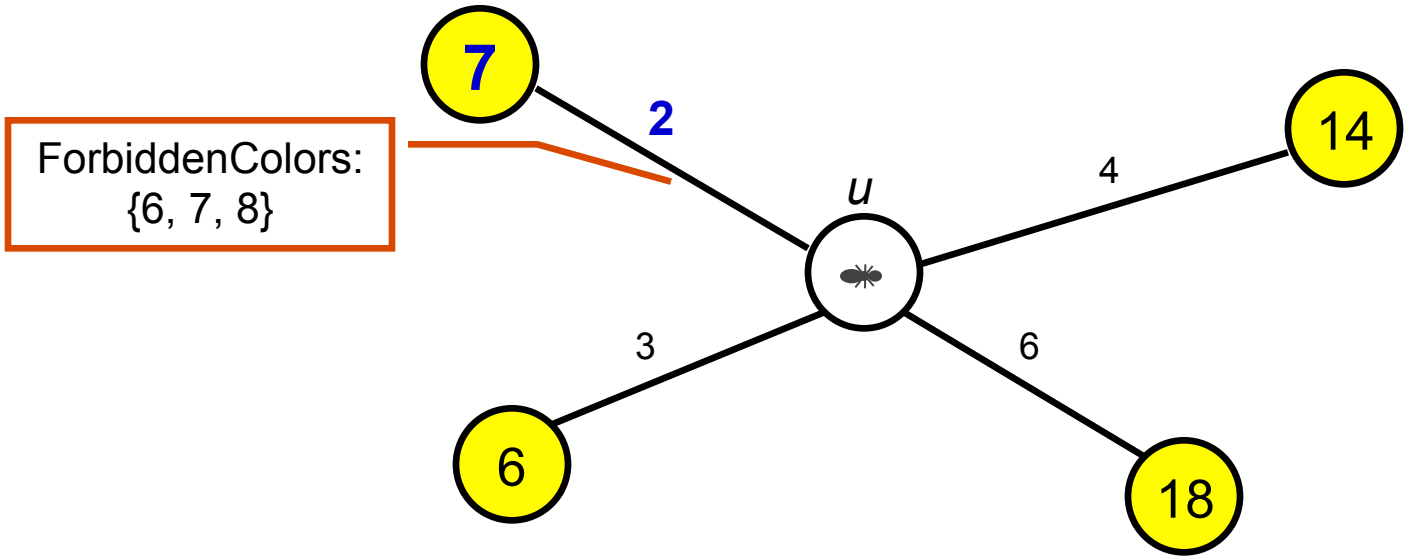


Example: How An Agent Colors

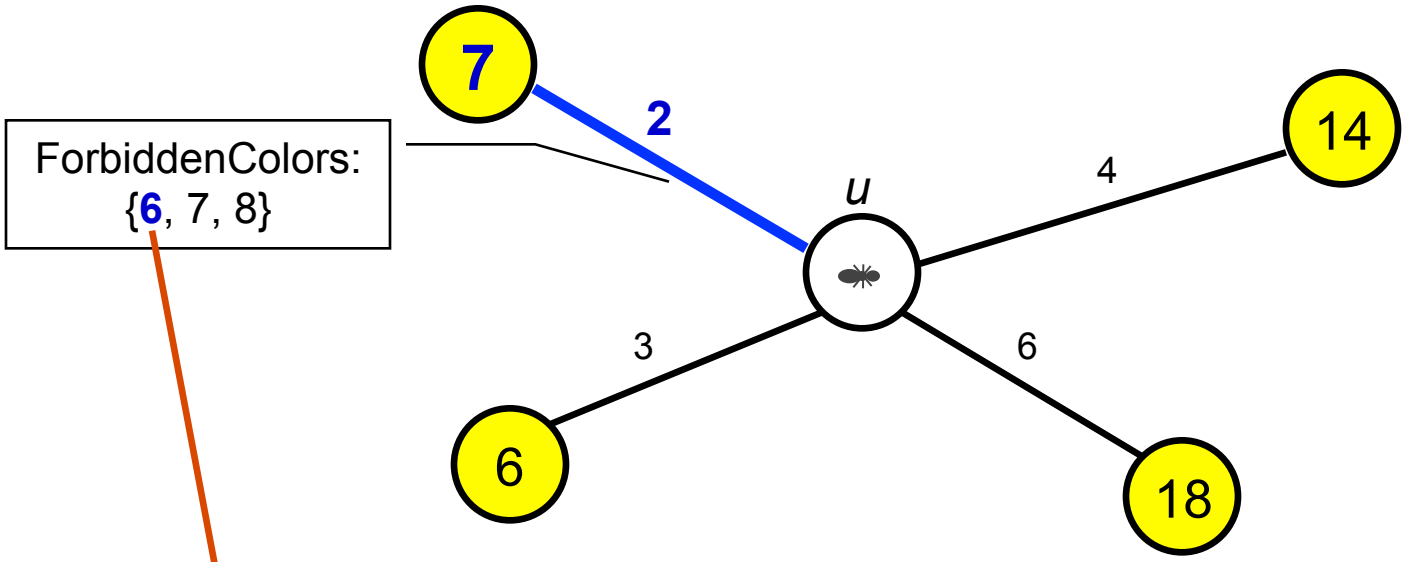


- AvailableColors = { 1, … , 26 }

Example: How An Agent Colors



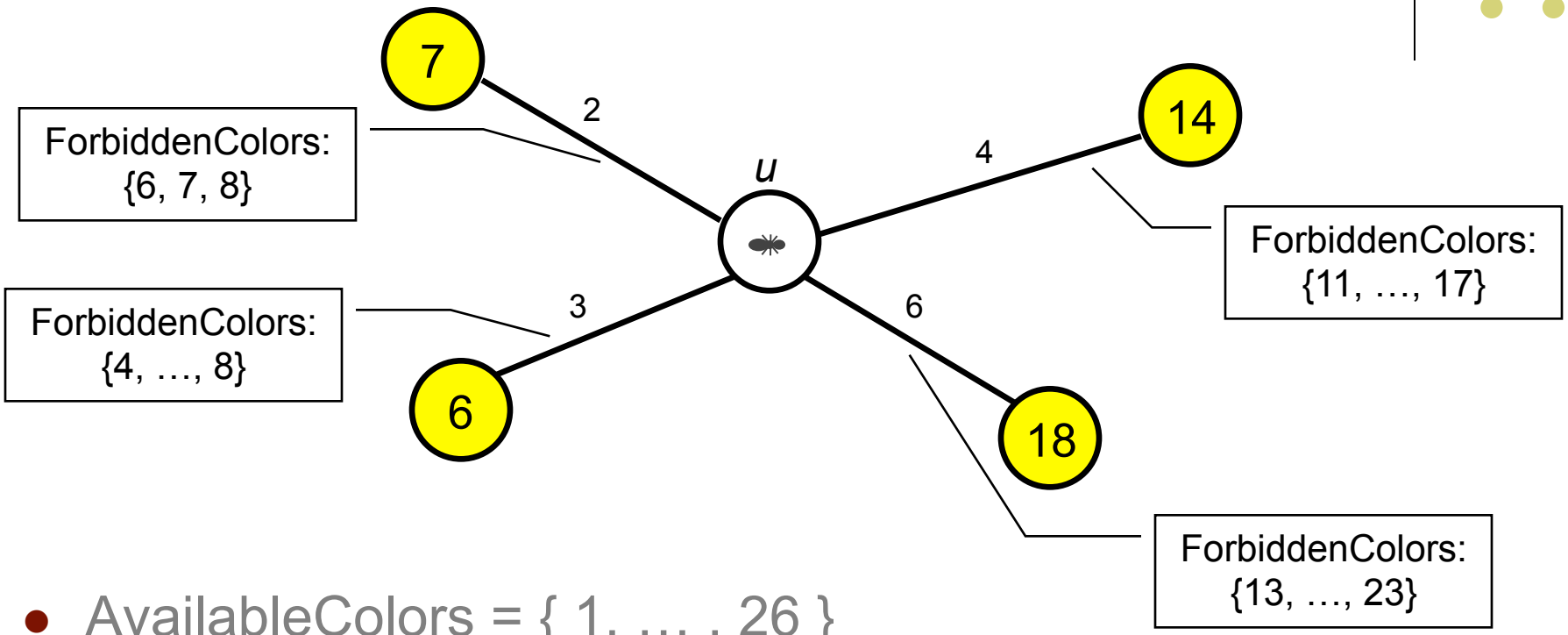
Example: How An Agent Colors



ForbiddenColors:
{6, 7, 8}

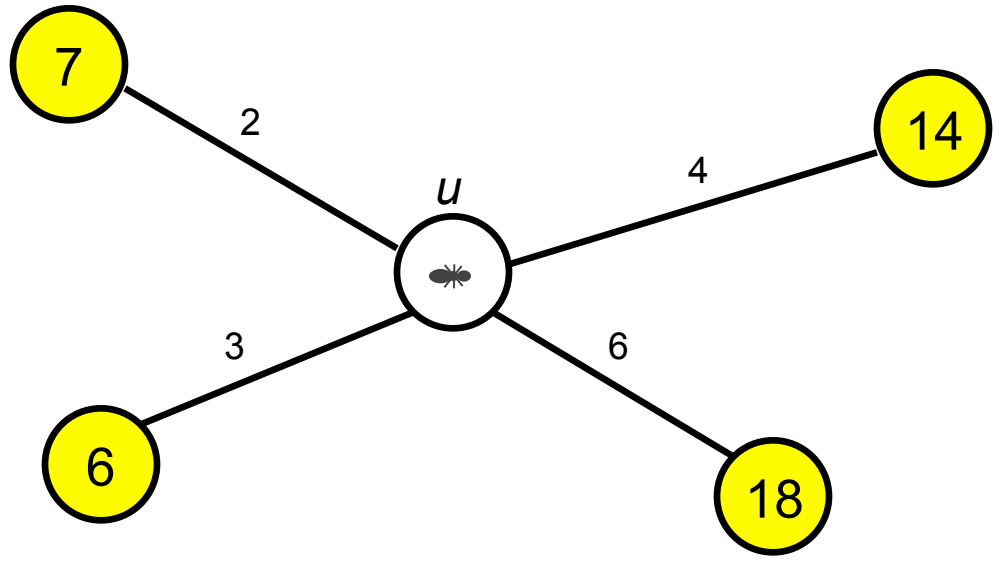
6 is a forbidden color for *u* because
| 7 – 6 | = 1, less than 2,
the edge weight

Example: How An Agent Colors



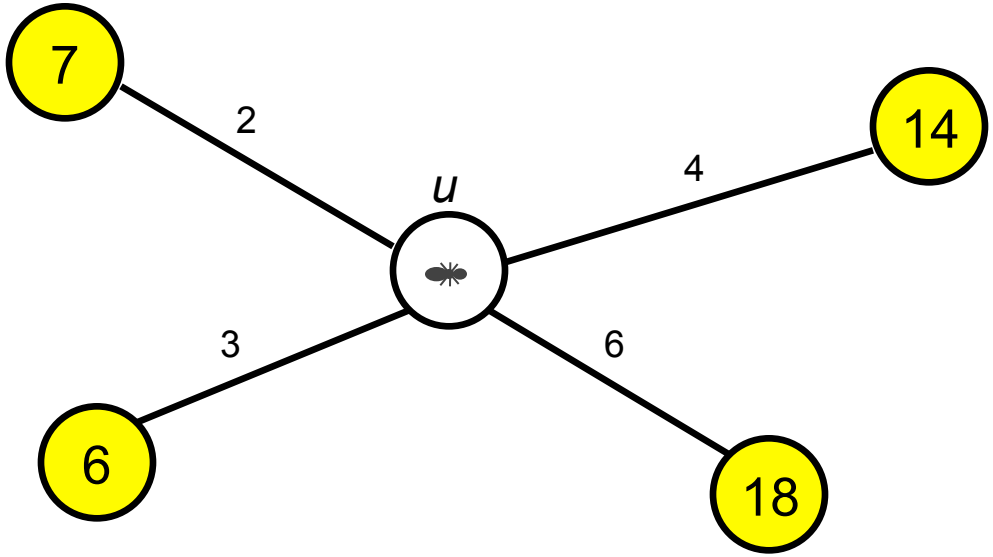
- AvailableColors = { 1, … , 26 }
- ForbiddenColors = { 4, …, 8, 11, …, 23 }

Example: How An Agent Colors



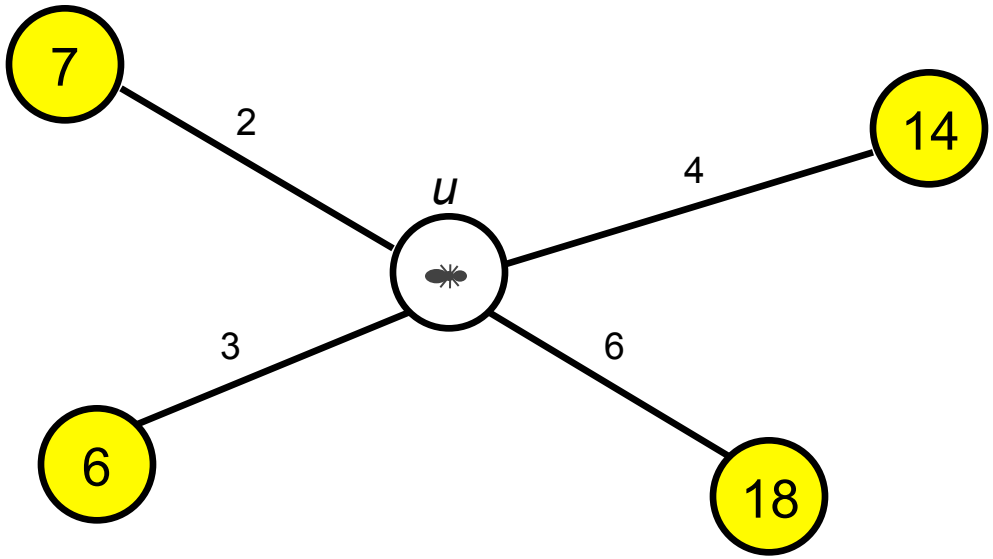
- AvailableColors = { 1, … , 26 }
- ForbiddenColors = { 4, …, 8, 11, …, 23 }
- EligibleColors = AvailableColors — ForbiddenColors
  = [1 … 3] U [9 … 10] U [24 … 26]

Example: How An Agent Colors



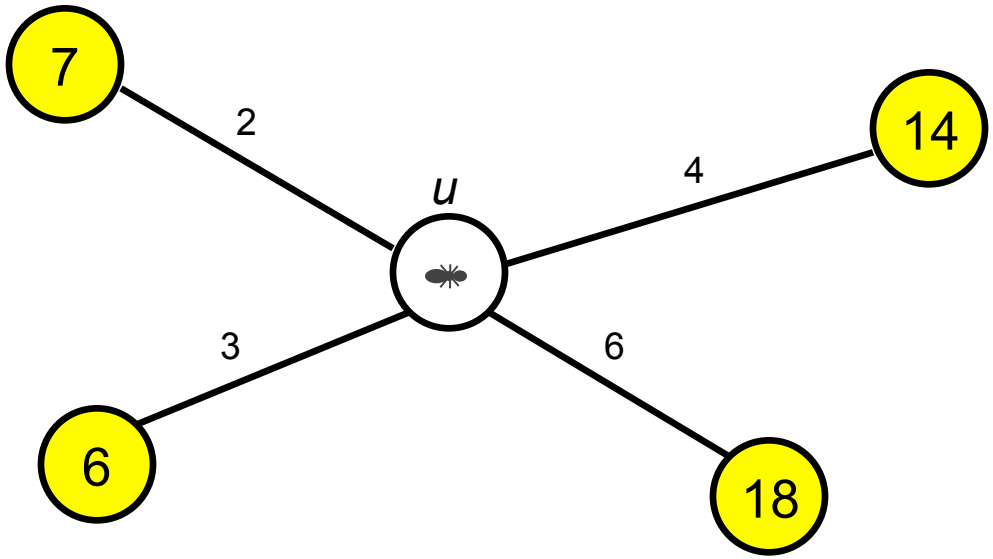
- AvailableColors = { 1, … , 26 }
- ForbiddenColors = { 4, …, 8, 11, …, 23 }
- EligibleColors = AvailableColors — ForbiddenColors
  = [1 … 3] U [9 … 10] U [24 … 26]
  - Largest Interval Size: 3

Example: How An Agent Colors



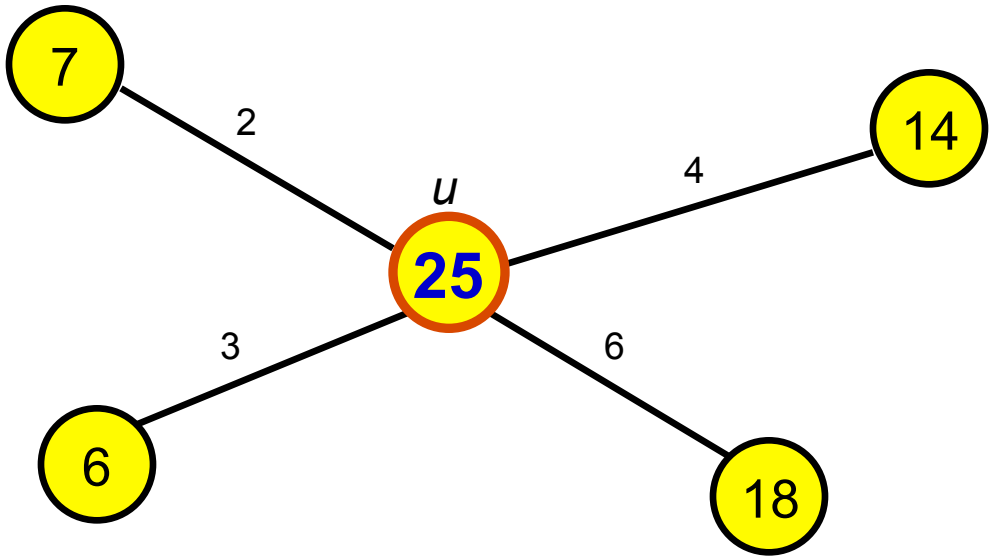
- AvailableColors = { 1, … , 26 }
- ForbiddenColors = { 4, …, 8, 11, …, 23 }
- EligibleColors = AvailableColors — ForbiddenColors
  = [1 … 3] U [9 … 10] U [24 … 26]
  - Select randomly one of the largest size intervals

Example: How An Agent Colors



- AvailableColors = { 1, … , 26 }
- ForbiddenColors = { 4, …, 8, 11, …, 23 }
- EligibleColors = AvailableColors — ForbiddenColors
  = [1 … 3] U [9 … 10] U [24 25 26]
  - Pick the median of the selected interval as the color for *u*

Example: How An Agent Colors



- AvailableColors = { 1, … , 26 }
- ForbiddenColors = { 4, …, 8, 11, …, 23 }
- EligibleColors = AvailableColors — ForbiddenColors
  = [1 … 3] U [9 … 10] U [24 25 26]
  - Pick the median of the selected interval as the color for *u*

ABGC Algorithm Outline

Initialization

k ← # of available colors

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 ← # of colors 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

- **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*** ← # of colors in the current best coloring – 1

ABGC Algorithm Outline

Initialization

k ← # of available colors

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 ← # of colors 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

Jolt Operation

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

- **How it works**
  - Randomly recolor neighbors of the top 10% conflicted vertices

- Perturbation is done whenever a period of time has passed without any improvement.

ABGC Algorithm Outline

Initialization

k ← # of available colors

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 ← # of colors 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**
  - A number of cycles has passed

  Or …

  - A number of cycles has passed without any improvement

ABGC Algorithm Outline

Initialization

k ← # of available colors

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 ← # of colors in the *bestColoring* — 1

Jolt operation

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

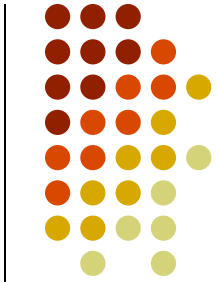
until some criteria are met

return the best coloring found

Outline

- Problems Definition
- ABGC Algorithm
- **Results**
- Conclusion

Testing Details



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

- **Benchmark Instances**
  - 99 instances were created from 33 (DIMACS) graphs for 3 different coloring problems.
  - The algorithm is run 100 times for each instance.

Comparison

- Results are compared against the following algorithms
  - Squeaky Wheel Optimization
    - Lim et al's **SWO** (all generalizations, 2003)
    - Lim et al's **SWO + Tabu Search (SWO/TS)** (all generalizations, 2005)

    *Consider only results from **SWO/TS** since it outperforms **SWO***

  - Local Search & Constraint Propagation
    - Prestwich's **FCNS** (Bandwidth Coloring only, 2002)
    - Prestwich's **SATURN** (Bandwidth Multi Coloring only, 2002)

    There are no results for the Multi Coloring Problem

Result: Bandwidth Coloring Problem

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom20a	20	22	20	0.03
geom20b	13	14	13	0.04
geom30	28	29	28	0.07
geom30a	27	32	27	0.09
geom40a	37	38	37	0.18
geom40b	33	34	33	0.25
geom50a	50	52	50	0.39
geom50b	35	38	36	0.39
geom60	33	34	33	0.39
geom60a	50	53	50	0.65
geom60b	43	46	43	0.83
geom70a	62	63	62	0.84
geom70b	48	54	51	1.02

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom80	41	42	41	0.70
geom80a	63	66	63	1.26
geom80b	61	65	63	1.60
geom90a	64	69	66	1.85
geom90b	72	77	74	2.57
geom100	50	51	50	1.23
geom100a	70	76	72	2.75
geom100b	73	88	76	3.59
geom110	50	53	50	1.48
geom110a	74	82	75	3.35
geom110b	79	88	84	4.22
geom120	60	62	59	1.67
geom120a	84	92	86	3.93
geom120b	87	98	90	5.90

Result: Bandwidth Coloring Problem

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom20a	20	22	20	0.03
geom20b	13	14	13	0.04
geom30	28	29	28	0.07
geom30a	27	32	27	0.09
geom40a	37	38	37	0.18
geom40b	33	34	33	0.25
geom50a	50	52	50	0.39
geom50b	35	38	36	0.39
geom60	33	34	33	0.39
geom60a	50	53	50	0.65
geom60b	43	46	43	0.83
geom70a	62	63	62	0.84
geom70b	48	54	51	1.02

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom80	41	42	41	0.70
geom80a	63	66	63	1.26
geom80b	61	65	63	1.60
geom90a	64	69	66	1.85
geom90b	72	77	74	2.57
geom100	50	51	50	1.23
geom100a	70	76	72	2.75
geom100b	73	88	76	3.59
geom110	50	53	50	1.48
geom110a	74	82	75	3.35
geom110b	79	88	84	4.22
geom120	60	62	59	1.67
geom120a	84	92	86	3.93
geom120b	87	98	90	5.90

Result: Bandwidth Coloring Problem

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom20a	20	22	20	0.03
geom20b	13	14	13	0.04
geom30	28	29	28	0.07
geom30a	27	32	27	0.09
geom40a	37	38	37	0.18
geom40b	33	34	33	0.25
geom50a	50	52	50	0.39
geom50b	35	38	36	0.39
geom60	33	34	33	0.39
geom60a	50	53	50	0.65
geom60b	43	46	43	0.83
geom70a	62	63	62	0.84
geom70b	48	54	51	1.02

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom80	41	42	41	0.70
geom80a	63	66	63	1.26
geom80b	61	65	63	1.60
geom90a	64	69	66	1.85
geom90b	72	77	74	2.57
geom100	50	51	50	1.23
geom100a	70	76	72	2.75
geom100b	73	88	76	3.59
geom110	50	53	50	1.48
geom110a	74	82	75	3.35
geom110b	79	88	84	4.22
geom120	60	62	59	1.67
geom120a	84	92	86	3.93
geom120b	87	98	90	5.90

Result: Bandwidth Coloring Problem

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom20a	20	22	20	0.03
geom20b	13	14	13	0.04
geom30	28	29	28	0.07
geom30a	27	32	27	0.09
geom40a	37	38	37	0.18
geom40b	33	34	33	0.25
geom50a	50	52	50	0.39
geom50b	35	38	36	0.39
geom60	33	34	33	0.39
geom60a	50	53	50	0.65
geom60b	43	46	43	0.83
geom70a	62	63	62	0.84
geom70b	48	54	51	1.02

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom80	41	42	41	0.70
geom80a	63	66	63	1.26
geom80b	61	65	63	1.60
geom90a	64	69	66	1.85
geom90b	72	77	74	2.57
geom100	50	51	50	1.23
geom100a	70	76	72	2.75
geom100b	73	88	76	3.59
geom110	50	53	50	1.48
geom110a	74	82	75	3.35
geom110b	79	88	84	4.22
geom120	60	62	59	1.67
geom120a	84	92	86	3.93
geom120b	87	98	90	5.90

Result: Bandwidth Coloring Problem

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom20a	20	22	20	0.03
geom20b	13	14	13	0.04
geom30	28	29	28	0.07
geom30a	27	32	27	0.09
geom40a	37	38	37	0.18
geom40b	33	34	33	0.25
geom50a	50	52	50	0.39
geom50b	35	38	36	0.39
geom60	33	34	33	0.39
geom60a	50	53	50	0.65
geom60b	43	46	43	0.83
geom70a	62	63	62	0.84
geom70b	48	54	51	1.02

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom80	41	42	41	0.70
geom80a	63	66	63	1.26
geom80b	61	65	63	1.60
geom90a	64	69	66	1.85
geom90b	72	77	74	2.57
geom100	50	51	50	1.23
geom100a	70	76	72	2.75
geom100b	73	88	76	3.59
geom110	50	53	50	1.48
geom110a	74	82	75	3.35
geom110b	79	88	84	4.22
geom120	60	62	59	1.67
geom120a	84	92	86	3.93
geom120b	87	98	90	5.90

Result: Bandwidth Coloring Problem

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom20a	20	22	20	0.03
geom20b	13	14	13	0.04
geom30	28	29	28	0.07
geom30a	27	32	27	0.09
geom40a	37	38	37	0.18
geom40b	33	34	33	0.25
geom50a	50	52	50	0.39
geom50b	35	38	36	0.39
geom60	33	34	33	0.39
geom60a	50	53	50	0.65
geom60b	43	46	43	0.83
geom70a	62	63	62	0.84
geom70b	48	54	51	1.02

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom80	41	42	41	0.70
geom80a	63	66	63	1.26
geom80b	61	65	63	1.60
geom90a	64	69	66	1.85
geom90b	72	77	74	2.57
geom100	50	51	50	1.23
geom100a	70	76	72	2.75
geom100b	73	88	76	3.59
geom110	50	53	50	1.48
geom110a	74	82	75	3.35
geom110b	79	88	84	4.22
geom120	60	62	59	1.67
geom120a	84	92	86	3.93
geom120b	87	98	90	5.90

Result: Bandwidth Coloring Problem

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom20a	20	22	20	0.03
geom20b	13	14	13	0.04
geom30	28	29	28	0.07
geom30a	27	32	27	0.09
geom40a	37	38	37	0.18
geom40b	33	34	33	0.25
geom50a	50	52	50	0.39
geom50b	35	38	36	0.39
geom60	33	34	33	0.39
geom60a	50	53	50	0.65
geom60b	43	46	43	0.83
geom70a	62	63	62	0.84
geom70b	48	54	51	1.02

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom80	41	42	41	0.70
geom80a	63	66	63	1.26
geom80b	61	65	63	1.60
geom90a	64	69	66	1.85
geom90b	72	77	74	2.57
geom100	50	51	50	1.23
geom100a	70	76	72	2.75
geom100b	73	88	76	3.59
geom110	50	53	50	1.48
geom110a	74	82	75	3.35
geom110b	79	88	84	4.22
geom120	60	62	59	1.67
geom120a	84	92	86	3.93
geom120b	87	98	90	5.90

Result: Bandwidth Coloring Problem

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom20a	20	22	20	0.03
geom20b	13	14	13	0.04
geom30	28	29	28	0.07
geom30a	27	32	27	0.09
geom40a	37	38	37	0.18
geom40b	33	34	33	0.25
geom50a	50	52	50	0.39
geom50b	35	38	36	0.39
geom60	33	34	33	0.39
geom60a	50	53	50	0.65
geom60b	43	46	43	0.83
geom70a	62	63	62	0.84
geom70b	48	54	51	1.02

Graph	FCNS	SWO/ TS	ABGC	
			Result	Time (sec)
geom80	41	42	41	0.70
geom80a	63	66	63	1.26
geom80b	61	65	63	1.60
geom90a	64	69	66	1.85
geom90b	72	77	74	2.57
geom100	50	51	50	1.23
geom100a	70	76	72	2.75
geom100b	73	88	76	3.59
geom110	50	53	50	1.48
geom110a	74	82	75	3.35
geom110b	79	88	84	4.22
geom120	60	62	59	1.67
geom120a	84	92	86	3.93
geom120b	87	98	90	5.90

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.

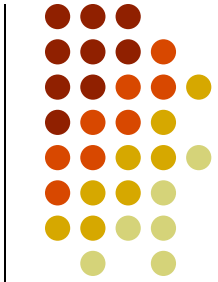
Result: Bandwidth Multi Coloring Problem

Graph	SATURN	SWO/ TS	ABGC	
			Result	Time (sec)
geom20	159	149	149	6.85
geom20a	175	169	169	11.27
geom30	168	160	160	9.49
geom30a	235	209	210	25.39
geom30b	79	77	77	1.24
geom40	189	167	167	24.57
geom40a	260	213	214	66.72
geom40b	80	74	74	3.04
geom50	257	224	224	57.48
geom50a	395	318	317	379.48
geom50b	89	87	85	4.54
geom60	279	258	258	64.39
geom60a	—	358	357	203.23
geom60b	128	116	117	10.64
geom70	310	273	267	110.96
geom70a	—	469	470	276.63
geom70b	133	121	121	12.46

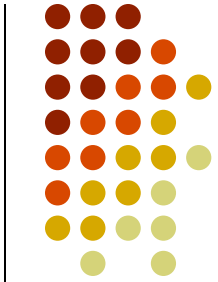
Graph	SATURN	SWO/ TS	ABGC	
			Result	Time (sec)
geom80	—	383	382	157.88
geom80a	—	379	367	239.61
geom80b	—	141	139	18.01
geom90	—	332	332	180.91
geom90a	—	377	378	387.54
geom90b	—	157	150	22.50
geom100	—	404	405	292.10
geom100a	—	459	440	548.34
geom100b	—	170	164	27.85
geom110	—	383	378	405.16
geom110a	—	494	487	1069.85
geom110b	—	206	208	43.86
geom120	—	402	398	790.21
geom120a	—	556	548	1660.62
geom120b	—	199	198	41.25

Outline

- Problems Definition
- ABGC Algorithm
- Results
- **Conclusion**



Conclusion



- **ABGC**
  - Agent based, hybrids with many other techniques (Tabu List, Greedy Local optimization, etc)
  - Produced competitive results
  - **Generality**: applicable to all three generalizations (as well as the classic coloring problem)
- **Future Work**
  - Use pheromone
  - Explore parallel implementation