A Branch & Cut Algorithm for the Partitioned Graph Coloring Problem

Santiago Palladino, Isabel Méndez-Díaz, Paula Zabala
{spalladino,imendez,pzabala}@dc.uba.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

ALIO-INFORMS
Joint International Meeting

June 2010

Index

Partitioned Colouring Problem

Definition

Given a graph $G = (V, E, P)$, being P a partitioning of the set of nodes $\{V_1, \ldots, V_p\}$, G is *partition coloured* if exactly one node v_i per partition V_i is coloured and no two adjacents nodes have the same colour.

As with the standard graph colouring problem, we seek to minimize the number of colours required to partition-colour the graph.

Motivation

Introduced as a means to solve the Routing and Wavelength Assignment (RWA) problem in Wavelength Division Multiplexed (WDM) optical networks.

Used to handle wavelength assignment conflicts between lightpaths sharing common fiber links, minimizing the number of wavelengths used, when all connection requirements are already known.

Other techniques to solve the RWA problem itself are out of the scope of this work.

Related work

Li, Simha; 2000	Greedy one step and two step heuristics
Noronha, Ribeiro; 2006	Tabu search heuristic
Frota et al; 2009	Branch and cut based on asymmetric representatives formulation

Related work

Li, Simha; 2000	Greedy one step and two step heuristics
Noronha, Ribeiro; 2006	Tabu search heuristic
Frota et al; 2009	Branch and cut based on asymmetric representatives formulation

We will propose an alternative integer programming formulation of the problem based on Mendez-Diaz and Zabala's model for standard coloring.

Model

Every variable x_{ij} is true if vertex i is colored with color j.
Variables w_j are true if color j is used in coloring the graph.

OBJECTIVE: Minimize sum of colors used

$$\min \sum_{j \in C} w_j$$

Model

Every variable x_{ij} is true if vertex i is colored with color j.
Variables w_j are true if color j is used in coloring the graph.

OBJECTIVE: Minimize sum of colors used

$$\min \sum_{j \in C} w_j$$

SUBJECT TO: Neighbours shall not have the same colour, and variable w_j must be true if the color is to be used

$$x_{ij} + x_{kj} \leq w_j \qquad \forall j, \forall (i,k) \in E$$

Model

Every variable x_{ij} is true if vertex i is colored with color j.
Variables w_j are true if color j is used in coloring the graph.

OBJECTIVE: Minimize sum of colors used

$$\min \sum_{j \in C} w_j$$

SUBJECT TO: Neighbours shall not have the same colour, and variable w_j must be true if the color is to be used

$$x_{ij} + x_{kj} \leq w_j \qquad \forall j, \forall (i,k) \in E$$

SUBJECT TO: Every vertex has exactly one colour assigned

$$\sum_{j \in C} x_{ij} = 1 \qquad \forall i \in V$$

Model

Every variable x_{ij} is true if vertex i is colored with color j.
Variables w_j are true if color j is used in coloring the graph.

OBJECTIVE: Minimize sum of colors used

$$\min \sum_{j \in C} w_j$$

SUBJECT TO: Neighbours shall not have the same colour, and variable w_j must be true if the color is to be used

$$x_{ij} + x_{kj} \leq w_j \qquad \forall j, \forall (i,k) \in E$$

SUBJECT TO: Every partition has exactly one colour assigned

$$\sum_{x_i \in p} \sum_{j \in C} x_{ij} = 1 \qquad \forall i \in V, p \in P$$

Improving LP

We also take symmetry breaking constraints from the original model and additional restrictions for eliminating fractional solutions.

SUBJECT TO: Color j cannot be used unless color $j-1$ was used

$$w_j \leq w_{j-1} \qquad \forall j \neq 0 \in C$$

SUBJECT TO: The label of a color used for a partition cannot exceed χ

$$\sum_{j \in C} w_j \geq \sum_{x_i \in p} \sum_{j \in C} x_{ij} \qquad \forall p \in P$$

Reworking adjacency constraints

Adjacency constraints can be replaced by either of the following, which improve the relaxation's resolution.

SUBJECT TO: A node i_0 and all of its neighbours in a partition p_0 cannot share the same the color

$$\sum_{i \in p_0 \cap N(i_0)} x_{ij_0} + x_{i_0 j_0} \leq w_{j_0} \qquad \forall i_0 \in V, j_0 \in C, p_0 \in P$$

SUBJECT TO: A color j_0 may be used on a node i_0 or on at most r of its neighbours, being r the number of different partitions in $N(i_0)$

$$\sum_{i \in N(i_0)} x_{ij_0} + r x_{i_0 j_0} \leq r w_{j_0} \qquad \forall j_0 \in C, i_0 \in V$$

Extended clique inequalities

Let $K \subseteq V$ an *extended clique* if for every pair $v, w \in K$, either v and w are adjacent or belong to the same partition.

We define the extended clique inequality as

$$\sum_{i \in K} x_{ij_0} \leq w_{j_0} \qquad \forall j_0 \in C$$

We use a greedy heuristic to find maximal cliques in the graph which violate this inequality.

Block color inequalities

Given the symmetry breaking constraints, if a partition is not be coloured with color j then it cannot be coloured using any colour with a higher label.

This allows us to define the block color constraints as:

$$\sum_{i \in p_0} \sum_{j \geq j_0} x_{ij} \leq w_{j_0} \qquad \forall p_0 \in P, j_0 \in C$$

All these constraints are simply handled by brute force.

Component independent set inequalities

Let $I \subseteq V$ be a *component independent set* if for every pair $v, w \in I$, v and w are not adjacent and belong to different partitions.

This allows us to reuse the independent set inequality from the standard coloring problem.

$$\sum_{i \in I} x_{ij_0} \leq \alpha(I) w_{j_0} \qquad \forall j_0 \in C$$

Which can be strengthened considering symmetry breaking.

$$\sum_{i \in I} x_{ij_0} + \sum_{j=n-\alpha(I)+1}^{n} \sum_{i \in V} x_{ij} \leq \alpha(I) w_{j_0} + w_{n-\alpha(I)+1} \qquad \forall j_0 \leq n - \alpha(G)$$

Hole and path inequalities

Component independent set inequalities can be specialized as component hole and component path inequalities, using $\lfloor |H|/2 \rfloor$ and $\lceil |P|/2 \rceil$ respectively as their cardinals.

We use a greedy heuristic constructing a path from each node for each colour. The criteria for choosing a node is based on its corresponding x_{ij} value and whether it has been previously visited or not.

G' independent set inequalities

Given a partitioned graph $G = (V, E, P)$, we define the graph $G' = (V', E')$, with $V' = P$ and E' such that $p_1, p_2 \in V'$ are adjacent iif every node in partition p_1 in G is adjacent to every node in partition p_2 in G. More formally:

$$E' = \{(p_1, p_2) : p_1, p_2 \in V' \wedge \forall v \in p_1 \forall w \in p_2 : (v, w) \in E\}$$

This grants another way to reuse independent set inequalities from the standard coloring problem; let I' be an independent set in G', then:

$$\sum p \in I' \sum_{i \in p} x_{ij_0} \leq \alpha(I') w_{j_0} \qquad \forall j_0 \in C$$

Branch and Cut

Using the model previously defined along with the valid inequalities implemented as cutting planes, we implemented a branch and cut algorithm to tackle this problem, adding initial and primal heuristics and specific branching strategies.

Cutting planes

We implemented the inequalities in the previous section as cutting planes, tested initially in a Cut and Branch algorithm, and then added to a Branch and Cut structure.

Cuts are applied aggressively on the root of the branching tree and every few nodes on the rest of the tree until a certain depth, in order to keep relaxations easy to solve.

Cutting planes

The cuts with the best performance were Extended Clique and Block Color, and are therefore the most aggressively applied.

Component independent set and G' independent set cuts are applied only if a minimum number of the previous cuts is not found.

DSatur

DSatur is an exact method for finding an optimal coloring for a graph by implicitly enumerating all possible colourings.

It is a sequential algorithm in which nodes are chosen based on the *degree of saturation*: the number of different colours used for its neighbours in the current solution.

Harder nodes (higher degree of saturation) are chosen first.

Extending DSatur

Classic DSatur is used for standard colouring. We propose two extensions for partitioned colouring:

- *Easiest node:* We first select the easiest node (lowest degree of saturation) to be coloured in each partition, and then we apply standard criteria to pick the hardest one from that set.
- *Hardest partition:* We first pick the hardest partition to be coloured according to its degree of saturation, size and uncoloured nodes; and then we pick the easiest node (lowest degree) from that partition.

Heuristics

Although DSatur is an exact algorithm, it quickly generates good enough solutions, therefore being a good heuristic by bounding the number of iterations or time.

DSatur proved to be an excellent initial heuristic, sometimes actually arriving to the optimal solution by itself.

We also adapted it to serve as a primal heuristic by fixing all nodes with a high enough value in the relaxation and executing DSatur on the remaining ones.

Branching priorities

We implemented a static branching strategy for x_{ij} variables prioritizing on:

- The higher the number of partitions adjacent to node i
- The smaller the size of the partition where node i is
- The higher the colour label j

DSatur branching

A second branching strategy implemened was picking the nodes with the highest degree of saturation in the current solution.

The previous strategy was used as a tie-breaker for nodes with equal degree, picking a single node i_0.

The chosen x_{ij} variable for branching was the one with the highest value among all $x_{i_0 j}$.

Pruning

When the number of x_{ij} variables fixed to 1 during the branching process is high enough, we stop the branch and cut algorithm in that node, pruning the subtree.

The exact solution corresponding to that node is found using DSatur as an exact algorithm, enumerating all possible colourings of the remaining partitions.

Implementation

The previous algorithm was implemented in Java 1.6 on top of Cplex 12.1 and executed on a 2.80 GHz core with 4 Gb RAM.

We implemented both a Cut and Branch and a Branch and Cut algorithm to test the effectiveness of the strategies devised.

Note that work is still in progress and the implementation, much less its parametrization, is not final.

Test Suites

We built two test suites with random generated graphs:

- Fixed density of 50%, nodes from 60 to 90
- Fixed node count of 80, density from 20% to 80%

Partition size for both suites was fixed to 2.

Each set was run for 30 minutes and the solution gap and node count in the branch tree is reported for both Branch and Cut and Cut and Branch.

Fixed node count

Graph	Branch & Cut		Cut & Branch	
density	nodes	gap	nodes	gap
0.2	485	0.00	2536	0.00
0.4	1362	0.30	31209	0.48
0.6	377	0.41	7329	0.50
0.8	225	0.46	5135	0.51

Fixed density

Graph	Branch & Cut		Cut & Branch	
nodes	nodes	gap	nodes	gap
60	3039	0.00	121779	0.22
70	1856	0.29	54437	0.38
80	1077	0.31	25909	0.40
90	244	0.45	5817	0.51
100	337	0.54	13267	0.58

Future work

- Theoretical analysis of the polytope
- Implement more families of cutting planes
- Optimize primal and separation heuristics
- Build a larger and more comprehensive test suite
- Determine optimal parametrizations for different graphs
- Check performance of full branch and cut on larger instances

Questions...?