EarSearch User Guide

Version 1.0

Derrick Stolee
University of Nebraska-Lincoln
s-dstolee1@math.unl.edu

April 5, 2011

Abstract

The Ear Search program implements isomorph-free generation of 2-connected graphs by ear augmentations. This document describes the interfaces used for customized searches, as well as describes three example searches: unique saturation, edge reconstruction, and extremal graphs with a fixed number of perfect matchings.

1 Introduction

The EarSearch library implements the generation algorithm of [7] to generate families of 2-connected graphs. It is based on the TreeSearch library [9]. The class `EarSearchManager` extends the class `SearchManager` and manages the search tree, using ear augmentations to generate children. It automates the canonical deletion selection in order to remove isomorphs.

2 Data Management

2.1 Graphs

Graphs are stored using the `sparsegraph` structure from the `nauty` library.

During the course of computation, these graphs are modified using edge and vertex deletions. To delete the ith vertex, set the `v` array to -1 in the ith position. To delete the edge between the i and j vertices, set the `e` array to -1 in two places: in the list of neighbors for i where j was listed and in the list of neighbors for j where i was listed. To place the vertices or edges back, place the previous values into those places.

2.2 Augmentations and Labels

The labels for each augmentation use two 32-bit integers. The first is the order of the augmented ear. The second is the index of the pair orbit which is used for the endpoints of the ear.

2.3 `EarNode`

Each level of the search tree is stored in a stack, where all data is stored in an `EarNode` object. All of the members of `EarNode` are public, in order to easily add data structures and flags that are necessary for each application. All pointers are initialized to 0 in the constructor and are checked to be non-zero before freeing up any memory in the destructor.

The core data necessary for `EarSearchManager` is stored in the following members:

- ear_length – the length of the augmented ear.
- ear – the byte-array description of the augmented ear.
- num_ears – the number of ears in the graph.
- ear_list – the list of ears in the graph (-1 terminated).
- graph – the graph at this node.
- max_verts – the maximum number of vertices in all supergraphs. Default to max_n from `EarSearchManager`.
- reconstructible – TRUE if detectably reconstructible
- numPairOrbits – the number of pair orbits for this graph.
- orbitList – the list of orbits, in a an array of arrays. Each array `orbitList[i]` contains pair-indices for pairs in orbit and is terminated by `-1`.
- canonicalLabels – the canonical labeling of the graph, stored as an integer array of values for each vertex
- solution_data – the data of a solution on this node.
- violatingPairs – A set of pair indices which cannot be endpoints of an ear.

3 Pruning

The interface `PruningAlgorithm` has an abstract method for pruning nodes of the search tree. The method `checkPrune` takes two `EarNode` objects: one for the parent and another for the child. Using this data, the method decides if no solution exists by augmenting beyond the child node. Since the pruning algorithm is called before the canonical deletion algorithm, this can also remove nodes which cannot possibly be canonical augmentations.

4 Canonical Deletion

The interface `EarDeletionAlgorithm` has an abstract method for finding a canonical ear deletion. The method `getCanonical` takes two `EarNode` objects for the parent and child and returns the array corresponding to the canonical ear. The `EarSearchManager` will determine if this canonical ear is in orbit with the augmented ear.

5 Solutions

The interface `SolutionChecker` is an abstract class which contains methods for finding solutions given a search node, storing the solution data, reporting on these solutions, and reporting application-specific statistics.

The method `isSolution` takes the parent, child, and depth and reports if there is a solution at the child node. It returns a non-null string if and only if there is a solution, and that string is a buffer containing the solution data. This buffer will be deallocated with `free()` by the `EarSearchManager`.

The method `writeStatisticsData()` returns a string of statistics (using the TreeSearch format) to be reported at the end of a job.

6 Example 0: 2-Connected Graphs

7 Example 1: Unique Saturation

7.1 Application-Specific Data

- adj_matrix_data – Data on the (directed) edges. For unique saturation, this gives -1 for edges, and for non-edges counts the number of copies of H given by adding that edge. Values are in $\{0, 1, 2\}$, since when 2 is listed, then there are too many copies of H.
- any_adj_zero – A boolean flag: are any of the cells in adj_matrix_data zero?
- any_adj_two – A boolean flag: are any of the cells in adj_matrix_data at least two?
- dom_vert – A boolean flag: is there a dominating vertex?
- copy_of_H – A boolean flag: is there a copy of H?

8 Example 2: Edge Reconstruction

8.1 Application-Specific Data

- child_data – the GraphData objects for immediate children, used for pairwise comparison.
- num_child_data – the number of GraphData objects currently filling the data.
- size_child_data – the number of pointers currently allocated.

9 Example 3: p-Extremal Graphs

This problem is investigated in [8]. See [1] and [3] for background on this problem.

9.1 Application-Specific Data

- extendable – A boolean flag: is the graph 1-extendable?
- numMatchings – The number of perfect matchings for this graph.
- barriers – The list of barriers of the graph, given as an array of Sets.
- num_barriers – the number of barriers in the graph.

9.2 Perfect Matching Algorithms

Algorithm 1 CountPM(G, P)

Algorithm 2 is1Extendable(H)

Algorithm 3 maximalSupergraphs(H)

References

[1] A. Dudek and J. Schmidt. On extremal graphs with a constant number of 1-factors. submitted. 2010.

[2] S. G. Hartke and A. J. Radcliffe. Mckays canonical graph labeling algorithm. In *Communicating Mathematics*, volume 479 of *Contemporary Mathematics*, 99111. 2009.

[3] S. G. Hartke, D. Stolee, D. B. West, and M. Yancey. On extremal graphs with a fixed number of perfect matchings. in preparation. 2011.

[4] B. D. McKay. Small graphs are reconstructible. *Australas. J. Combin.*, 15:123 126, 1997.

[5] B. D. McKay, Isomorph-free exhaustive generation *J. Algorithms*, 26(6):306-324. 1998.

[6] B. D. McKay, nauty user's guide (version 2.4) Dept. Computer Science, Astral. Nat. Univ., 2006.

[7] D. Stolee, Isomorph-free generation of 2-connected graphs with applications, in preparation, 2011.

[8] D. Stolee, Generating p-extremal graphs, in preparation, 2011.

[9] D. Stolee, TreeSearch user guide, available at http://www.github.com/derrickstolee/TreeSearch/ 2011.