Minimum Degree and Fill-in Heuristics

Eli Fox-Epstein

June 30, 2016

This abstract corresponds to the submission for the 2016 PACE Challenge for Track A. This is a serial, heuristic implementation. Source code is available at: `https://github.com/elitheeli/2016-pace-challenge`.

This entry is provided as a baseline, implementing a simple strategy: the Minimum Degree Heuristic [1] and Minimum Fill-In Heuristic. In the time given, this implementation first quickly finds a tree decomposition with the Minimum Degree Heuristic and then runs the Minimum Fill-In Heuristic repeatedly, maintaining the best decomposition.

This implementation is written in C++11. Tree decompositions are represented by two vectors: one is a vector of bags (each of which is a vector of vertex IDs) and one stores parent-child relationships.

The graph is represented using an adjacency matrix (compactly represented as a `std::vector<bool>`), as well as a adjacency list for each vertex (using a `std::vector`) to support efficient neighborhood iteration and a vector tracking vertex degrees. At any given time, the adjacency lists contain a *superset* of the neighbors of a vertex: on iteration through a neighborhood, each potential neighbor is verified with the matrix and discarded if it is not a true neighbor.

The implementation is templated by a function of the graph and a vertex that returns an floating point number. The algorithm consists of a number of iterations, each of which selects the vertex minimizing the function, adds edges to turn its neighborhood into a clique, and deletes the vertex from the graph. The order of vertex selection is a perfect elimination ordering in the graph consisting of all edges that were ever in the graph.

This template is instantiated for functions that return the degree of the vertex (for the Minimum Degree Heuristic) and the number of edges missing in its neighborhood (for the Minimum Fill-In Heuristic). Internally, the algorithm maintains a priority queue on the vertices, keyed by the templated function plus a small amount of random noise. The noise ensures a random choice among the vertices minimizing the function. Three additional vectors are maintained: one to store the order in which vertices are selected, one mapping a vertex to whether or not it has been selected, and one tracking the function value of a vertex when last enqueued. Initially, all vertices are enqueued. When processing a vertex v, if v has already been seen, it is simply discarded. If its key is less than its current function value, it is re-enqueued with the correct key. Finally, if its key is correct, it is added to the ordering, marked as visited, and its neighbors are connected into a clique and then disconnected from v. Some neighbors may be re-enqueued if their function values decreased.

References

[1] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3):255–269, 1957.