01 Apr 1991

Simulated Annealing on the Composite Graph Coloring Problem

Kevin Alons

Simulated Annealing on the Composite Graph Coloring Problem

Kevin Alons
Northwestern College
Orange City, Iowa 51041

The composite graph is a finite undirected graph G with no loops and no multiple edges and with positive integers associated with each vertex of G. The number of integers associated with each vertex is the chromaticity of the vertex. The chromatic degree of a vertex v is the sum of the chromaticity of v and the chromaticities of all vertices adjacent to v. The graph G is a v-composite graph if the chromaticities are not equal.

The v-composite graph coloring problem (CGCP) is the problem of assigning consecutive integers to each vertex of a graph in such a way that each vertex is assigned a number of integers equal to its chromaticity, no adjacent vertices have an integer in common and the number of integers used is a minimum. The minimum number of integers required to color a graph is referred to as the chromatic number of the graph.

The composite graph coloring problem is closely related to the non-composite graph coloring problem (GCP) and both are NP-complete. Exact algorithms for either of these problems exist only for very small graphs, so heuristic algorithms must be applied to solve practical problems.

For NP-complete problems, there is no known algorithm whose worst-case complexity is bounded by a polynomial in the size of the input. Local optimizing heuristic algorithms are often used to solve NP-complete problems approximately, i.e. to find "reasonable" solutions that are somewhat close to optimum. These algorithms explore a discrete space of reachable configurations, S, in a deterministic fashion. The set of solutions that can be reached in one change from a given configuration A is called the neighborhood of A. This is a generic description of local optimization:

1. Get an initial configuration S.
2. While there is an untested neighbor of S do the following:
   2.1 Let S' be an untested neighbor of S.
   2.2 If $cost(S') < cost(S)$, set $S = S'$.
3. Return S.

Starting from an initial configuration, a sequence of configurations is selected until a locally optimal solution is found. Typically, this local optimum is far from the globally best solution. Nonetheless, a better configuration cannot be reached from this local optimum as to escape this local region, an initial "bad" move must be accepted.

Simulated annealing (SA) is a randomizing heuristic algorithm recently applied to many NP-complete problems. While similar in many ways to its local optimization brethren, it differs in one important aspect. SA algorithms avoid local optimum through probabilistically accepting "uphill" or bad moves, in order to escape local optimum regions. This acceptance of uphill moves is controlled by a monotonically decreasing "temperature". As the temperature decreases, less uphill changes are allowed. This guides the algorithm to the global optimum region where it finds the best configuration. The following is the basic structure of simulated annealing in general:

1. Get an initial solution S.
2. Get an initial temperature $T > 0$.
3. While not yet "frozen" do the following:
   - 3.1 Perform the following loop L times:
     - 3.1.1 Pick a random neighbor S' of S.
     - 3.1.2 Let $\Delta = Cost(S') - S$.
     - 3.1.3 If $\Delta \leq 0$ (downhill move), Set $S = S'$.
     - 3.1.4 If $\Delta > 0$ (uphill move), Set $S = S'$ with probability $e^{-\Delta/T}$.
   - 3.2 Set $T = rT$ (reduce temperature).
4. Return S.

SA algorithms share four basic functions: accept, generate, update, and cost. The accept function decides whether to accept the proposed configuration, S'. The generate function generates this adjacent configuration, S'. The update function decreases the temperature by multiplying it by a cooling ratio r, $0< r <1$. The cost function generates the relative cost of a configuration, which is used in comparing different configurations.

Simulated annealing has been applied to the non-composite graph coloring problem. The results of these experiments prove that simulated annealing can perform as well as, if not better than most existing algorithms for the CGP. Since the two types of graph coloring are closely related, a method developed for the CGP was used for the CGCP.

Of the CGP simulated annealing methods developed, the penalty function method, was the most suitable for conversion to the CGCP. An initial solution using this algorithm is a partitioning of the vertices into color classes. This coloring (solution) is not required to be feasible. An adjacent solution is generated by randomly choosing a non-empty color class and a vertex within this color class. This vertex is then moved to a different randomly chosen color class which may be empty. This allows the number of color classes to increase, reducing the number of "bad edges" (adjacent vertices sharing colors).

The cost function used for penalty function annealing has two components, the first of which favors large color classes, the second which favors independent sets. It is:

$$cost\ (II) = -\sum_{i=1}^{k} |C_i|^2 + \sum_{i=1}^{k} 2\,|C_i| \cdot |E_i|$$

This cost function does not explicitly count the number k classes in Π; it minimizes this number as a side-effect of minimizing the cost of the configurations.

To convert this algorithm for use on the CGCP, modifications are needed. In generating a random neighboring configuration, the complexity in finding a valid vertex is increased. Instead of moving any random vertex, a vertex either starting or ending with the chosen color class must be found. This is due to the fact that vertices must have consecutive colors (integers). If a valid vertex in this color class does not exist, a new color class must be chosen. The constant in the second part of the equation used to penalize bad edges was increased to place more emphasis on independent sets. To ensure consistent convergence to feasible solutions, this additional penalizing of bad edges with composite graph coloring is required.

A functioning sequential algorithm was developed, which did converge to feasible, while not optimal, solutions. Developing the data structures to effectively implement simulated annealing required considerable time and effort. While improvements could be made to the

actual C program, it did incorporate dynamic lists and other techniques to reduce execution time and conserve memory. Run times of this algorithm on substantial graphs are quite large, which limited actual testing and timing of the algorithm. Instead, a parallel version of this algorithm was developed for a distributed memory parallel processing computer.

The parallel version was designed for and executed on a 16-node IPSC/2 hypercube. This application required the development of three separate programs: a host, a controller, and the node program. The host program acquires the requested number of nodes, and initializes and loads the cube. One node gets the controller program, the remaining nodes receive the worker program. This parallel application can be massively parallelized -- it can utilize any number of processors.

The controller program is essentially a copy of the sequential version with the inclusion of message passing within the outer loop. The worker programs contain only the inner loop of the sequential version and are coordinated by the controller. This coordination is not achieved through explicitly synchronized message passing, but is instead an inherent part of the algorithm design. Each node traces a different path through the solution space S by executing its inner loop a pre-specified number of times, then returning its best-solution-to-date to the controller. The controller takes the best *feasible* solution received and returns this solution and the decremented temperature to all nodes. This continues until the controller terminates the entire algorithm in the same manner as the sequential version.

This application appears to offer increased consistency in convergence to optimal feasible solutions with reduced running times. Since more paths from each common origin in the solution space are investigated, it stands to reason (and limited observance) that an optimal path will be located more efficiently in parallel. These predictions have yet to be proven, as conclusive data was not obtained.

A major goal of this program was to learn about and design a reliable, efficient parallel algorithm. While the sequential version took considerable work, the parallel version was the end result. Once the sequential algorithm was reliable, conversion to parallel required only an efficient system of message passing. This was especially critical as distributed memory parallel machines have high communication overhead. The goal was met as the program was reliable and promised increased efficiency.

Both the sequential and parallel versions of this algorithm prove that simulated annealing are applicable to the CGCP. Since the research done was primarily a study on parallelization and parallel algorithm design, little time was allocated for confirming the actual results of simulated annealing. More work is needed optimizing the many different parameters used (required by any simulated annealing application) before any realistic comparison can be done to existing CGCP algorithms.

Works Cited

Johnson, D. S., C. R. Aragon, L. A. McGeoch, and C. Schevon. 1990a. Optimization by Simulated Annealing: An Experimental Evaluation, Part II (Graph Coloring and Number Partitioning).