ACADEMIC PRESS

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

J. Parallel Distrib. Comput. 63 (2003) 786–800

Journal of Parallel and Distributed Computing

http://www.elsevier.com/locate/jpdc

Neural network-based heuristic algorithms for hypergraph coloring problems with applications

Dmitri Kaznachey,[a,*] Arun Jagota,[b] and Sajal Das[c]

[a] *Portfolio Analytics, Fannie Mae, Washington, DC 20016, USA*
[b] *Department of Computer Science, UCSC, Santa Cruz, CA 95064, USA*
[c] *Department of Computer Science, and Engineering, University of Texas at Arlington, Arlington, TX 76-19-0015, USA*

Received 5 May 1999; revised 2 May 2000

Abstract

The graph coloring problem is a classic one in combinatorial optimization with a diverse set of significant applications in science and engineering. In this paper, we study several versions of this problem generalized to hypergraphs and develop solutions based on the neural network approach. We experimentally evaluate the proposed algorithms, as well as some conventional ones, on certain types of random hypergraphs. We also evaluate our algorithms on specialized hypergraphs arising in implementations of parallel data structures. The neural network algorithms turn out to be competitive with the conventional ones we study. Finally, we construct a family of hypergraphs that is hard for a greedy strong coloring algorithm, whereas our neural network solutions perform quite well.

Keywords: Hopfield network; Hypergraph coloring; Combinatorial optimization

1. Introduction

The graph coloring problem is to color the vertices of an undirected graph with the fewest number of colors such that no two adjacent vertices are assigned the same color. This problem is a classic one in combinatorial optimization with a diverse set of significant applications in science and industry. In this paper we study several versions of this problem generalized onto hypergraphs.

A *hypergraph*, like a graph, is comprised of a vertex-set $V = \{v_1, \ldots, v_n\}$ and an edge-set $E = \{e_1, \ldots, e_m\}$. A hypergraph differs in that a (hyper) edge is an arbitrary subset of V, not just a 2-element subset. In this paper we often deal with a restricted class of hypergraphs called d-cardinality hypergraphs in which hyperedges have cardinality at most d.

Hypergraph problems often retain their theoretical and practical applications in a generalized setting. Applications of hypergraph optimization problems include course scheduling (hypergraph vertex cover); data access in parallel memory systems (hypergraph coloring); job scheduling in multiprocessor systems (hypergraph matching); and others (see [15]).

The graph coloring problem may be generalized to hypergraphs in more than one way. One still wishes to assign colors to vertices in such a way that all edges are "properly" colored, but now the definition of proper is not unique. Hypergraph coloring remains a computationally challenging problem, and inherits the applications of graph coloring in a generalized setting.

1.1. Hypergraph strong coloring

The *Hypergraph Strong Coloring Problem* (*HSC*) [1] is defined as follows. Given a hypergraph $H = (V, E)$, color the vertices V with the minimum number of colors in such a way that no two vertices in the same hyperedge $e \in E$ have the same color.

This problem arises naturally in applications involving conflict-free access of data in a parallel memory system (see [3,4], for example) consisting of a number of processors and memory modules. The success of parallel computation depends on the ability of the system to fetch data in parallel.

*Corresponding author.
E-mail address: dkaznachey@yahoo.com (D. Kaznachey).

0743-7315/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/S0743-7315(03)00064-9

Define a *template* as a set of data elements that need to be processed in parallel. Hence the data elements from a template should be stored in different memory modules to allow concurrent access. For a given set of templates the problem is to assign elements to the minimum number of memory modules in such a way that for each template all its elements are stored in different memory modules so that these elements can be accessed in one memory cycle. This is an HSC problem where data elements are vertices, templates are edges, and memory modules are colors.

We now observe that the hypergraph strong coloring problem is reducible to the graph coloring problem.

Lemma 1.1. *From a given hypergraph* $H=(V_h,E_h)$, *construct a graph* $G=(V_g,E_g)$ *as follows. Let* $V_g=V_h$ *and initially* $E_g=\emptyset$. *For each edge* $e=\{v_{i_1},\ldots,v_{i_k}\}\in E_h$, *add* $\binom{k}{2}$ *edges to* E_g, *one for each* 2*-element subset* $\{v_{i_r},v_{i_s}\}$ *of* e. *Then we have the following.*

C *is a proper coloring of* V_h *in* H (*no two vertices in the same hyperedge* $e\in E$ *have the same color*) *if and only if* C *is a proper coloring of* V_g *in* G (*no end points of an edge have the same color*).

In some applications, the number of available colors may be fixed in advance (for example, fixed number of memory modules in parallel data structures). This motivates the following variant of the strong coloring problem in which one relaxes the aim of coloring all the vertices. For a hypergraph $H=(V,E)$, let C denote the subhypergraph of H induced by the vertex-set $S\subseteq V$. The vertex-set of $H[S]$ is S. The hyperedge-set of $H[S]$ is the set of those hyperedges of H that are subsets of S. (There may be another definition of $H[S]$ in which $e\in E$ is in $H[S]$ only if $e\subseteq S$.) The *Maximum Induced Subhypergraph Strong k-coloring Problem* (*HkC*) is to color the largest (having the largest number of vertices) induced subgraph H' of H with k colors so that no two vertices in the same hyperedge in H' have the same color.

Both HSC and HkC are NP-hard, since their graph coloring versions are.

1.2. Edge-uniform coloring

The strong coloring constraint often requires a large number of colors to satisfy it. If one has only a few colors available, then the problem needs to be suitably relaxed. Relaxing the constraint that all vertices be colored yields to the HkC problem above. In this section we study a different kind of relaxation—one in which the constraint that "every vertex in edge e be colored differently" will be relaxed to "the k colors be spread evenly amongst the vertices in e".

More formally, the *Hypergraph Edge-Uniform k-coloring Problem* (*HeukC*) is to color the vertices in a hypergraph $H=(V,E)$ with k colors so that, for each edge e, the k colors are distributed as evenly as possible over the vertices of e. Define $s_{vc}=1$ if a vertex v is colored by color c and 0 otherwise. Then the goal may be formulated so as to minimize the following cost function:

$$C=\sum_{e\in E}\max_{c=1}^{k}\sum_{v\in e}s_{vc} \tag{1.1}$$

In parallel memory access systems HeukC represents a conflict minimization problem. In this case the number of memory modules is fixed and some of the data elements from a template may be stored in the same memory module. The number of memory cycles needed to access a template equals to the maximum number of data elements stored in the same memory module. The problem is to assign data elements to memory modules so that the number of memory cycles is minimized for each template.

2. Benchmark hypergraphs

Below we describe various types of hypergraphs on which we will evaluate our algorithms for hypergraph coloring problems. We will often use the notation *d-cardinality hypergraphs* to denote hypergraphs whose hyperedges have cardinality at most d. Throughout this paper n will denote the number of vertices and m the number of edges.

2.1. Random hypergraphs

We will let $H_{n,d,p}$ denote an n-vertex, d-cardinality hypergraph generated randomly as follows: each subset of the vertex-set V of cardinality at most d is, independently of others, made an edge of $H_{n,d,p}$ with probability p.

2.2. Data structure hypergraphs

This section describes hypergraphs that arise in certain parallel data structure applications. In such applications there is some data structure (e.g. a tree T) on a set of vertices V with certain substructures called templates (e.g. subset of V that induces a subtree T' of T) all of whose nodes have to be assigned different memory modules [4]. We can model the conflict-free access problem by constructing a hypergraph H whose vertex-set equals V and whose edges are the substructures (templates) and then solving an appropriate version of the hypergraph coloring problem with the colors representing the memory modules. For this