Adjacency List Representation of Hypergraphs

Abstract. Hypergraphs are important data structures used to represent and model the concepts in various areas of Computer Science and Discrete Mathematics. As of now an adjacency matrix representation and a bipartite incidence representation have been given for its implementation. The present paper proposes two novel methods for hypergraph representation using adjacency list. A comparison has been made with the existing representations to show that the proposed approach is better in terms of time complexity. Various graph algorithms such as Breadth- first search, Depth-first search, Strongly connected components, Dijkstras shortest path algorithm are implemented and studied in detail using the proposed representation for hypergraphs.

Keywords: hyperedge, hypergraphs, adjacency list, incidence matrix

1 Introduction

A hypergraph H(V, E) is a generalization of a graph, where an edge can connect any number of vertices ($\geq$1). Here V is a finite set of elements, called nodes or vertices and E is a set of nonempty subsets of V called hyperedges [17]. If two or more edges have exactly the same properties and the same information flows through them then they can be combined together to form one hyperedge. Hypergraphs find applications to represent diagram components as well as spatial relationship between components [10]. An undirected hypergraph is same as an undirected graph but with the additional property of a hypergraph that an edge can connect more than two vertices. An example of such an undirected hypergraph is given in Fig. 1 and Fig. 2. A Directed Hypergraph (DH) is a hypergraph with directed hyperedges which doesnt obey the symmetric property i.e. aRb $\neq$ bRa. An example is given in Fig. 3 and Fig. 4.

Hypergraphs have been used in the satisfiability problem, AND/OR graphs and the analysis of passenger distribution in a transit system [5]. Directed hypergraphs find application in very crucial problems which were quite difficult to be perceived earlier. One such problem is the modeling of transit networks [13]. Directed hypergraphs also find a wide application as a modeling tool to represent concepts and structures in many application areas including formal languages, relational databases [4], production and manufacturing systems, public transportation systems [12] etc. Hypergraphs are closely related to other formalisms like context-free grammars [8], and deductive systems [14]. Hypergraphs are often used as data structure to fill in the gap between the given domain knowledge (architecture e.g. Floor layout, construction, machine building) and its internal representation in a computer programme [7]. Many intermediate tools have been built to address this problem. One such tool is diagram editor [11, 6] which not only has the capabilities of a drawing tool but can also understand edited diagrams to a certain extent.

In literature, only two representations have been found for hypergraphs. Two basic data structures for hypergraphs are bipartite incidence representation and incidence matrix representation [15]. Bipartite incidence representation is analogous to adjacency lists while incidence matrix representation is analogous to the adjacency matrix representation. Efforts have been taken to efficiently solve interesting problems by representing those using large hypergraphs with millions of edges and vertices. This work presents a novel representation for directed as

well as undirected hypergraphs so as to minimize the time required for traversal and hence for breadth first search (BFS), depth first search (DFS) and shortest path algorithms. The present manuscript is an attempt to represent hypergraphs to improve the time and space complexity for such problems.

The proposed concept introduces three lists. A vertical list consists of all the vertices in the hypergraph. Each vertex is connected to each of the hyperedges which emerge from it. Now each hyperedge is further connected to a horizontal list consisting of all the vertices to which it directs. The hyperedges are also arranged in a vertical list such that they can be accessed together when necessary. This decreases the search and insertion time in the hypergraph by manifolds. This is possible as proposed approaches avoid nested traversal and allow independent traversal of both vertices and edges.

The organization of this document is as follows. Section 2 explains the methods used for the graphical representation of hypergraph. Section 3 discusses the existing representations of hypergraph available. The proposed concepts have been explained in Section 4. Section 5 gives the comparison of the time complexities of the given representations. Section 6 gives the modified algorithms for traversal and shortest path using the proposed representation. The data used for experimentation is described in section 7 and the results are shown in Section 8. The work has been concluded in Section 9.

2 Graphical Representation of Undirected and Directed Hypergraphs

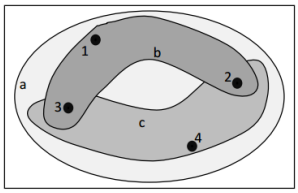

Fig. 1. An example undirected hypergraph (subset standard representation),with V = {1,2,3,4} and E = {a, b, c} = {{1,2,3,4}, {1,2,3}, {2,3,4}}.

There are two widely used methods for the graphical representation of a hypergraph. In the first method each pair of vertex in the same region has the same set of properties. These are assumed to be connected by a common edge termed as a hyperedge. This is also known as subset standard way of representation of hypergraph as shown in Fig. 1 [9]. This method can be used to represent undirected hypergraph only.

Another way of representing undirected and directed hypergraphs is shown in Fig. 2 and Fig. 3, which are known as edge standard way of representation of hypergraphs [1, 9]. Here boxes are used to represent the hyperedges and the ellipses represent the vertices. This method can be used to represent undirected as well as directed hypergraphs. A directed hyperedge takes the ordered pair form, E (T, H), where T is the tail and H is the head of the hyperedge [5]. Information may be considered to flow through the hyperedge from its tail to its head. Fig. 2 and Fig. 3 give representations for undirected and directed hypergraphs respectively. In Fig. 2, the vertices 1, 2, 3 and 4 are related by a common edge, termed hyperedge a.

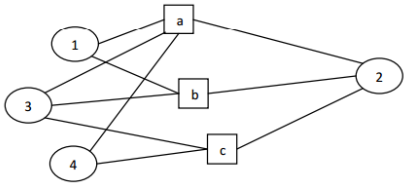

Fig. 2. Edge standard way of representation of undirected hypergraph (G)with V = {1,2,3,4} and E = {a,b,c} = {{1,2,3,4}, {1,2,3}, {2,3,4}}.

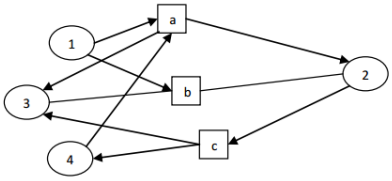

Fig. 3. Edge standard way of representation of a directed hypergraph (G').

3 Existing Approaches for Computer Representation of Hypergraphs

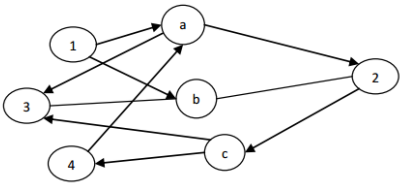

Fig. 4. Bipartite incidence representation of G'.

The **bipartite incidence structures of hypergraph** [15] for the graph G' shown in Fig. 3 is given in Fig. 4. It would assume all hyperedges to be the same as vertices. Thus the bipartite incidence structure has only vertices (represented by circles in the figure). Although, it is conceptually wrong to treat an edge as a vertex, but this representation may easily be converted into a simple digraph. The bipartite incidence representation in Fig. 4 can be transformed into an equivalent adjacency list representation as given in Table 1.

A hypergraph can also be represented in the form of an incidence matrix and termed as the **incidence matrix representation** [5] of hypergraph. For example, a directed hypergraph G' shown in Fig. 4 can be presented as an incidence matrix given in Table 2. The tail and the head of hyperedge E may be denoted by T(E) and H(E), respectively. In this form of representation, the head has been symbolized using a '-' followed by edge name and for the tail, an edge name has been used directly.

Table 1. Adjacency list representation of graph in Fig. 4.

vertex	*adjacentvertices*
1	a b
2	b c
3	b
4	a
a	2 3
b	2 3
c	4 3

Table 2. Incidence matrix representation of G'

	a	**b**	**c**
1	-1	-1	0
2	1	1/-1	-1
3	1	1/-1	1
4	-1	0	1

In Table 2, each column corresponds to a hyperedge and each row to a vertex. The cell corresponding to a vertex and a hyperedge is labeled 0 if the vertex is not connected to the hyperedge, 1 if the vertex is a tail of the hyperedge and -1 if the vertex is a head of the hyperedge. Representation of an undirected edge in this scheme requires either addition of an extra row/column to accommodate both 1 and -1 in a cell or a 3-dimentional matrix to include both. Thus, this representation fails to accommodate an undirected edge without any alterations.

4 Proposed Concept

4.1 Adjacency List Representation (Method 1)

In the first approach, incidence matrix representation of hypergraph is converted to adjacency list. Incidence matrix representation of hypergraph G given in Table 2 is converted to adjacency list representation as shown in Fig. 5.

The upper part of Fig. 5 is a conversion of incidence matrix into adjacency list, where the leftmost rectangles represent the hyperedges, i.e. a, b and c. Each of these edges is linked to a horizontal list of all the vertices connected to the hyperedge. Now each of these vertices has a vertical list of their own. Negative connotation of the letters (e.g. a, -b, -c) shows that these vertices have an outward edge towards these hyperedges, while the positive connotation shows that the vertices have an inward edge from these hyperedges. This representation is a loose annotation as the edges are not connected to each other. To avoid this, the second part of the diagram has been thought of. In this, the horizontal list has been replaced by pointers to vertex nodes which have a list of their own. The same negative and positive connotation hold true here as well.

This representation becomes too complex and cumbersome with the increasing size of incidence matrix and therefore run times of DFS, BFS, Strongly connected components, Dijkstras shortest path algorithm using this representation are found to be quite high.

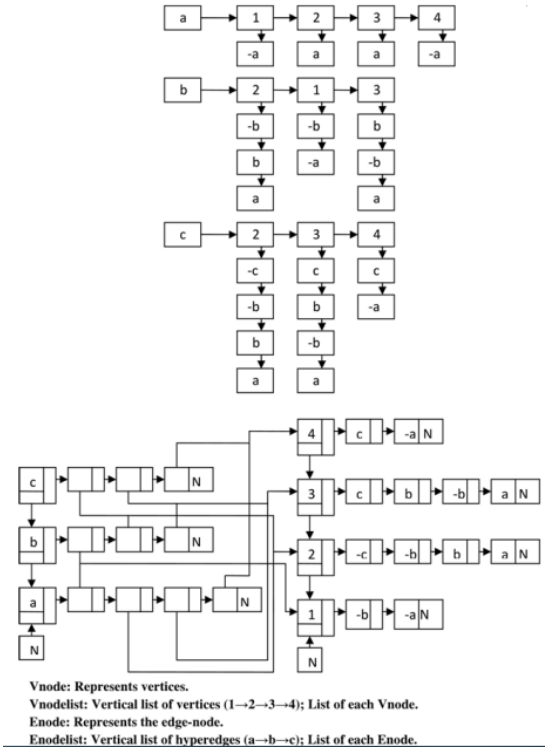

Fig. 5. Bipartite incidence representation of G'. The concept for the upper part of the image is taken from [5], while the lower part represents the real formation and representation in the memory.

4.2 Adjacency List Representation combined with RDF (Method 2)

The second method has been developed by using previous concepts of adjacency list representation of simple graph and adding the Resource Description Framework (RDF) graph representation model [12] to it. Using this second approach, hypergraph G can be represented as given in Table 3.

In Table 3, numbers (1, 2, 3 and 4) are used to represent vertices and alphabets (a, b and c) for hyperedges of a hypergraph. Each node is connected to all other connected nodes through a hyperedge e.g. Node 1 is connected to Node 2 and 3 via hyperedge a.

This relationship is represented by means of a horizontal list in Fig. 6 where a hyperedge (rectangle) leaves vertex (ellipse) 1 and enters vertices 2 and 3. The terminal interrupt boxes (rounded rectangles) shown for 2 and 3 are just pointers to the vertical list and hence the numbers are written and the arrow pointers are shown in red color. The hyperedges are also linked with each other in a list. The tiny boxes within the elliptical nodes are array of pointers to all the hyperedge nodes being pointed by the vertex.

Table 3. Adjacency list representation of G'

Node	Edge	Connected Node	
1	a	2	3
2	b	3	2
2	c	3	4
3	b	2	3
4	a	2	3

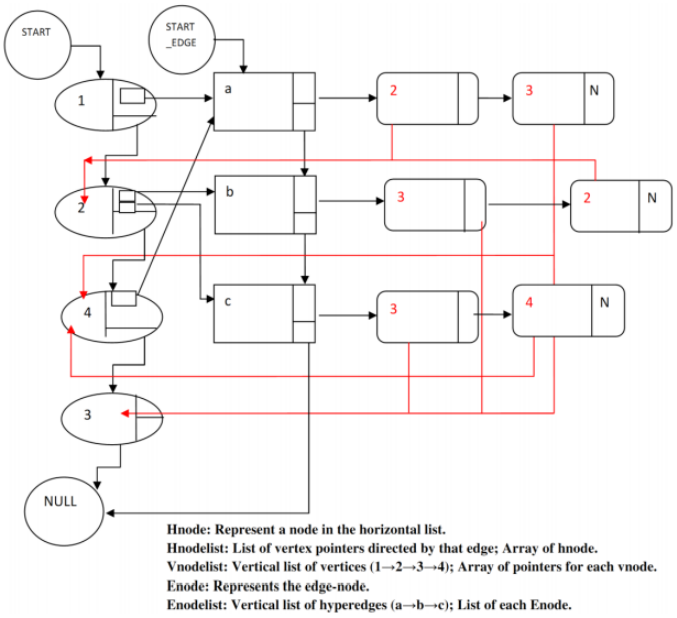

Fig. 6. Proposed Concept (Method 2)

5 Comparison of Time Complexity of Proposed Methods

5.1 Traversal and Insertion Complexities

Consider a graph with n vertices and n_e hyperedges. A comparison in terms of time complexity using Big-O notation has been performed for both the methods proposed in section 4. Table 4 gives the information about the complexities related to traversal and similarly, Table 5 lists the details of insertion complexity.

5.2 Best and Worst Case

The minimum value of ne is 0 and its maximum value is $2^n - 1$. On the basis of these values the best and the worst case complexities have been calculated which are as listed in Table 6. Method 2 proves itself to be a good choice for all the cases of traversal as well as insertion except one i.e. the best case for traversal where Method 1 performs better.

Table 4. Traversal Complexity

	Method 1	**Method 2**
	$O(n)$ to search the vertices in the vnode list. $O(2n_e)$ to search for enodes of those vertices in the list for each vnode. $O(n_e)$ to search for enodes in the enode list. $O(n)$ to search for a enode in the list for each enode.	$O(n)$ to search the vertices in the vnode list. $O(n_e)$ to search for enodes of those vertices in the array of pointers. $O(n)$ to search for hnodes in the hnode list for each enode.
Overall:	$O(n+2n_e+n_e+n)$=$O(2n+3n_e)$	$O(n+n_e+n)$=$O(2n+n_e)$

Table 5. Insertion Complexity

	Method 1	**Method 2**
	$O(n)$ to search the vertices in the vnode list. $O(2n_e)$ to search for enodes of those vertices in the list for each vnode. $O(n_e)$ to search for enodes in the enode list. $O(n)$ to search for a enode in the list for each enode. $O(n_e)$ to insert its positive and negative values in list for each vnode.	$O(n)$ to search the vertices in the vnode list. $O(n_e)$ to search for enodes of those vertices in the array of pointers. $O(n)$ to search for hnodes in the hnode list for each enode.
Overall:	$O(2n+3n_e+n_e)$=$O(2n+4n_e)$	$O(n+n_e+n)$=$O(2n+n_e)$

Table 6. Traversal and insertion times of the proposed representations in terms of the nodes and the number of hyperedges

	Method 1	**Method 2**
Traversal	$O(2n+3n_e)$	$O(2n+n_e)$
Traversal(Best Case)	$2n+3n_e=2n$(for $n_e=0$)	$2n+n_e=2n$ (for $n_e=0$)
Traversal(Worst Case)	$2n+3(2^n-1)$	$2n+2^n-1$
Insertion	$O(2n+4n_e)$	$O(2n+n_e)$
Insertion (Best Case)	$2n+4n_e$=$2n$(for $n_e=0$)	$2n+2n_e=2n$(for $n_e=0$)
Insertion (Worst Case)	$2n+4(2^n-1)$	$2n+2(2^n-1)$

5.3 Run-time Comparison

The run-time comparison of discussed hypergraph representations with the proposed methods is given in Table 7. The worst-case time of traversing from one vertex to another via any hyperedge is less for the adjacency list representation rather than the adjacency matrix representation (using lists, the Method 1) and other existing approaches. This analysis clearly shows that the proposed Method 2 has an edge over proposed Method 1. Therefore, Method 2 is further explored in the remaining sections of this paper. Algorithms like DFS, BFS, Strongly connected components, Dijkstras shortest path if implemented using Method 2, will take lesser time.

Table 7. Comparison of run-time of various representations for traversal and insertion.

Representations	**Traversal**		**Insertion**	
	Best Case	**Worst Case**	**Best Case**	**Worst Case**
Incidence Matrix	1	$n(2^n - 1)$	1	1
Bipartite Incidence	n	$n+(2^n - 1)+n(2^n - 1)$	1	1
Method 1	2n	$2n + 3(2^n - 1)$	2n	$2n+4(2^n - 1)$
Method 2	2n	$2n + 2^n - 1$	2n	$2n+2(2^n - 1)$

6 Modified Algorithms for Hypergraphs

Algorithms given in [3] have been studied and modified for hypergraphs using Method 2. These modified algorithms for BFS, DFS, strongly connected components and Dijkstras shortest path algorithm are given in this section. Spanning tree problem remains an NP-complete problem [2, 16], therefore not discussed here.

BFS: This procedure takes a graph H = (V, E), represented using adjacency lists, as input. The variables color[u] and π[u] store the color and the predecessor of each vertex u∈V. The distance from source s to vertex u computed by the algorithm is stored in d[u]. A first-in, first-out queue Q is used to manage the set of gray vertices.

DFS: Input to this procedure is an undirected or directed graph H = (V, E). The global variable time is used for time stamping. Each vertex v has two timestamps: first timestamp (d[v]) records when v is first discovered, and second timestamp (f[v]) records when the search finishes examining v's adjacency list. Also, d[u] ¡f [u]. Vertex is WHITE before d[u], GRAY between time d[u] and time f[u], and BLACK thereafter.

Strongly Connected Components: This algorithm computes the strongly connected components of a directed graph H = (V, E) using two DFS, one on H and one on transpose of H i.e. H^T. H^T is defined to be a graph H^T = (V, E^T), where E^T = (u, v) : (v, u) ∈ E i.e. E^T consists of the edges of H with their directions reversed.

```
BFS(H,s)
1. for each vertex u ε X[H]-s
2.   do color[u] ← WHITE
3.     d[u] ← ∞
4.     π[u] ← NIL
5. color[s] ← GRAY
6. d[s] ← 0
7. π[s] ← NIL
8. Q← ø
9. ENQUEUE(Q, s)
10. while Q ≠ ø
11.   do u ← DEQUEUE(Q)
12.     for each v ε Adj[u → e (hyperedge)] //change only needed here
13.       do if color[v] = WHITE
14.         then color[v] ← GRAY
15.           d[v] ← d[u] + 1
16.           π[v] ← u
17.           ENQUEUE(Q, v)
18.     color[u] ← BLACK
```

```
DFS(H)
1. for each vertex u ∈ V [H]
2.   do color[u] ← WHITE
3.     π[u] ← NIL
4. time ← 0
5. for each vertex u ∈ V [H]
6.   do if color[u] = WHITE
7.    then DFS-VISIT(u)
```

```
DFS-VISIT(u)
1. color[u] ← GRAY //White vertex u has just been discovered.
2. time ← time +1
3. d[u] ← time
4. for each v ∈ Adj[u → e (hyperedge)] //Explore edge(u, v) –change only
needed here–
5. do if color[v] = WHITE
6.   then π[v] ← u
7.     DFS-VISIT(v)
8. color[u] ← BLACK //Blacken u; it is finished.
9. f[u]← time ← time +1
```

STRONGLY-CONNECTED-COMPONENTS (H)
1. call DFS (H) to compute finishing times f[u] for each vertex u
2. compute H^T
3. call DFS (H^T), but in the main loop of DFS, consider the vertices in order of decreasing f[u] (as computed in line 1)
4. output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

Dijkstras Shortest Path Algorithm: It takes a weighed, directed graph H = (V, E) as input. For each edge (u, v) ∈ E, it's weight w(u, v) 0. The algorithm maintains a set of vertices, S, whose final shortest-path weights from the source s has already been determined. Q is a min-priority queue of vertices, keyed by their d values.

DIJKSTRA(H, w, s)
1. INITIALIZE-SINGLE-SOURCE(H, s)
2. S← ø
3. Q ← V[H]
4. while Q ≠ ø
5. do u ← EXTRACT-MIN(Q)
6. S ← S u
7. for each vertex v ∈ Adj[u → e (hyperedge)] //change only needed here
8. do RELAX(u, v, w)

7 Data Set Used for Experimentation

The algorithm developed for traversing hypergraphs has been tested on two large hypergraphs. The first hypergraph has been formed using the database of Indian pin codes from the website http://www.indiapost.com. This was arranged in the form of a table with states, districts, areas and their respective pin codes as columns. For such data, districts and pin codes are considered as vertices and states and areas as hyperedges. This set of data finally consists of 151,202 vertices and 19,955 hyperedges.

Another data set was prepared by imitating a mail-server artificially. The table had three columns sender, recipient and the date on which a mail is sent from the sender to the recipient. Random email ids were generated from a fixed set of domains. These email ids were filled in sender and recipient columns of the server. The third column of dates is again populated randomly. This data could now represent as that of a mail server. Finally, this data set consisted of 269,756 vertices and 3,051 hyperedges.

8 Results

To compare the performances of modified algorithms with proposed Method 2, experiments have been carried out on both the datasets (discussed in the pre-

vious section) for large hypergraphs. System with common computing facilities consisting of an Intel Core 2 Quad processor, 4GB RAM and 500GB of hard disk is used for all the tests. Results shown in Table 8 are obtained by compiling and running the algorithms using different methods on the two datasets. This table does not contain entries for the execution of the incidence matrix method on dataset 1 because of its large dimensions i.e. 151,202 X 19,955 X 2 with restricted computing power available.

Table 8. Results for running times of various algorithms using incidence matrix method and proposed Method 2 on the two datasets. A zero (0) entry represents a very little fraction of a second and not exactly 0 seconds. All results are in seconds

Algorithms	**Dataset 1 (Proposed Method 2)**	**Dataset 2 (Incidence Matrix Method)**	**Dataset3 (Proposed Method 2)**
Entry time	168.4	2.82	942.88
Insertion Time	0	-	0
DFS	0.03	1778.48	2.77
Strong Connected Components	463.03	0	81.43
BFS	0	5.76	0
Shortest path	0	767.78	0.01

Nodes or hyperedges corresponding to an application are inserted one by one to build the entire structure on computer. It is very clear from the results that the time to build the entire data structure with the entries (i.e. entry time) through the proposed Method 2 is much more than that through the incidence matrix method. Insertion of any node or hyperedge by the proposed Method 2 takes a very small fraction of a second even if node(s) corresponding to the new hyperedge does not exist in the data structure. Moreover insertions may be done dynamically at run time. Incidence matrix method also takes a minute fraction of a second to insert an existing node or hyperedge, but being statically created matrix for building the structure it wont allow the insertion of a new node or hyperedge afterwards.

The large difference between the run times of DFS algorithm for the incidence matrix method and proposed Method 2 can be attributed to the fact that the proposed method explores the complete depth of a node initially, hence is much faster. Once DFS is calculated, strongly connected components are determined by taking the transpose of the entire hypergraph, which can easily be calculated using a matrix. So, in this case the incidence matrix method takes much lesser time as compared to the proposed Method 2. Again with the calculations of BFS and Dijkstras algorithm, the proposed Method 2supersedes the incidence matrix method by manifolds due to its efficient structure.

Although time required to build a hypergraph with the proposed method is very large, but once a hypergraph is generated, most of the operations executed do not take much time.

9 Conclusions

The paper discusses simple representations of hypergraphs i.e. edge-standard way in the format of adjacency list. The study carried out shows that incidence matrices are not useful for perfect representation of directed hypergraph and hence two of its simple and efficient representations have been presented. Further, a novel method, adjacency list representation combined with RDF has been explored. Comparison of this novel method with incidence matrix is done and the proposed approach is found to be much better than the incidence matrix method. Similarity of this approach with simple digraphs has also been exposed. Also, the basic codes of DFS, BFS, Strongly-connected hypergraphs and Dijkstras algorithms have been studied, modified and used. Slight modification in some of these algorithms has resulted in a perfect reutilization of the existing codes.

References

1. Andersson, P.: Hyperedge replacement grammars
2. Ausiello, G., Giaccio, R., Italiano, G.F., Nanni, U.: Optimal traversal of directed hypergraphs (1992)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2001)
4. Fagin, R.: Degrees of acyclicity for hypergraphs and relational database schemes. J. ACM 30(3), 514–550 (Jul 1983), http://doi.acm.org/10.1145/2402.322390
5. Gallo, G., Longo, G., Nguyen, S., Pallottino, S.: Directed hypergraphs and applications (1992)
6. Grabska, E., Grzesiak-Kope, K., Lembas, J., achwa, A., lusarczyk, G.y.: Hypergraphs in diagrammatic design. In: Wojciechowski, K., Smolka, B., Palus, H., Kozera, R., Skarbek, W., Noakes, L. (eds.) Computer Vision and Graphics, Computational Imaging and Vision, vol. 32, pp. 111–117. Springer Netherlands (2006), http://dx.doi.org/10.1007/1-4020-4179-9_17
7. Grabska, E., lusarczyk, G., Glogaza, M.: Design description hypergraph language. In: Kurzynski, M., Puchala, E., Wozniak, M., Zolnierek, A. (eds.) Computer Recognition Systems 2, Advances in Soft Computing, vol. 45, pp. 763–770. Springer Berlin / Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-75175-5_94
8. Huang, L.: Dynamic programming algorithms in semiring and hypergraph frameworks. Qualification Exam Report pp. 1–19 (2006)
9. Mäkinen, E.: How to draw a hypergraph. International Journal of Computer Mathematics 34(3), 177–185 (1990), http://dx.doi.org/10.1080/00207169008803875
10. Minas, M.: Hypergraphs as a uniform diagram representation model. In: UNIVERSITY OF PADERBORN. pp. 24–31. Springer (1998)
11. Minas, M.: Concepts and realization of a diagram editor generator based on hypergraph transformation. Sci. Comput. Program. 44(2), 157–180 (Aug 2002), http://dx.doi.org/10.1016/S0167-6423(02)00037-0
12. Morales, A.A.M., Serodio, M.E.V.: A directed hypergraph model for rdf. In: Proc. of Knowledge Web PhD Symposium. Citeseer (2007)
13. Nguyen, S., Pallottino, S., Gendreau, M.: Implicit enumeration of hyperpaths in a nested logit model for transit networks (1994)
14. Shieber, S.M., Schabes, Y., Pereira, F.C.N.: Principles and implementation of deductive parsing. JOURNAL OF LOGIC PROGRAMMING (1995)
15. Skiena, S.S.: The algorithm design manual (2008)
16. Warme, D.M.: Spanning trees in hypergraphs with applications to Steiner trees. Ph.D. thesis, Citeseer (1998)
17. Wikipedia: Hypergraph — wikipedia, the free encyclopedia (2013), \url{http://en.wikipedia.org/w/index.php?title=Hypergraph&oldid=535060188}, [Online; accessed 20-February-2013]