Introduction to hypergraphs and odometers

Roscoe Casita
CIS 503: Thesis

May 14, 2017

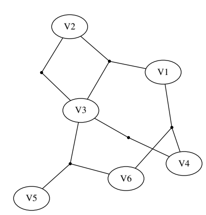
Abstract

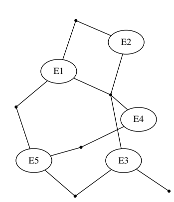
Hypergraph implementations have modeled real life systems with material gains in performance over graph representations of the same problems. Odometers serve as an alternative representation of hyperedges in a hypergraph from the traditional incident matrix. The traversal of hyperedges in a hypergraph using an odometer and incrementing function is shown to be similar to traversing a Hilbert curve through a N^{∞} dimensional Hilbert space.

1 Introduction

Hypergraphs are composed of a set of vertexes and hyperedges $HG = (v, he)$ commonly denoted in matrix format show below. Each column represents a nodes set of hyperedges. Each row represents a set of nodes in a hyperedge. The sample hypergraph presented as both nodes linked by hyperedges and hyperedges linked by nodes in the subsequent pictures. Displaying high dimensionality objects such as hypergraphs is a complex problem with active ongoing research in the area..

	V_1	V_2	V_3	V_4	V_5	V_6
E_1	1	1	1	0	0	0
E_2	0	1	1	0	0	0
E_3	0	0	1	0	1	1
E_4	0	0	1	1	0	0
E_5	1	0	0	1	0	1





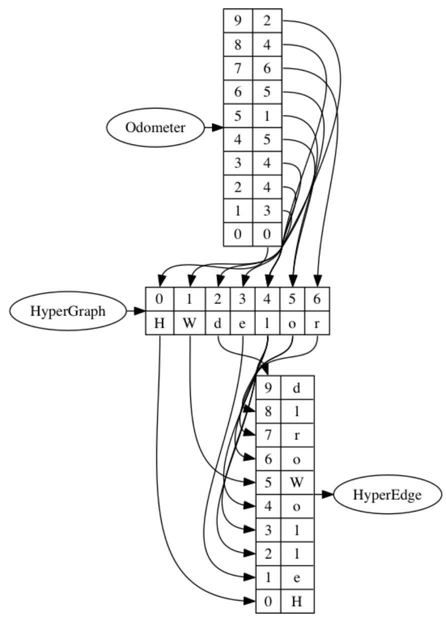
2 Hypergraph, odometer & hyperedges:

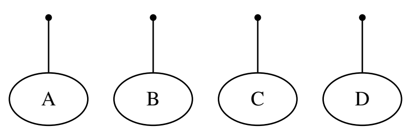
```
def makeHyperGraph(hyperedge):
    vector_things = [0 for x in range(len(hyperedge))]
    address_lookup = dict()
    for i in range(len(hyperedge)):
        node = hyperedge[i]
        vector_things[i] = node
        address_lookup[node] = i
    return (vector_things,address_lookup)

def getHyperEdge(hypernet,odometer):
    (vector_things,address_lookup) = hypernet
    hyperedge = [0 for x in range(len(odometer))]
    space_size = len(vector_things)
    for index in range(len(odometer)):
        node_index = odometer[index] % space_size
        hyperedge[index] = vector_things[node_index]
    return hyperedge

def getOdometer(hypergraph,hyperedge):
    (vector_things,address_lookup) = hypergraph
    odometer = [0 for x in range(len(hyperedge))]
    for index in range(len(hyperedge)):
        node = hyperedge[index]
        odometer[index] = address_lookup[node]
    return odometer
```

r0.5





3 Concepts

The presented hypergraph model supports conversion from an odometer to hyperedge or hyperedge to odometer in one function call. Traditionally all hyperedges in a hypergraph are explicitly declared and stored in memory or disk.

This model can explore super dense hypergraphs: 2^N, N^N even N^M hyperedges that standard matrix hypergraph models cannot represent. These hypergraphs share the common characteristic that they must be explored algorithmically. This image demonstrates a single odometer to hyperedge conversion via arbitrary hypergraph using the given definitions.

```
hypergraph =makeHyperGraph(sorted(set("Hello World!")))

odometer = getOdometer(hypergraph, "Hello World!")

hyperedge = getHyperEdge(hypergraph,odometer)

(vector_of_things,address_lookup) = hypergraph
```

4 Hypernets

The set of all hypernets represents the traversal of all hypergraphs over time. Thus the set of all hypernets represents all programs that can be interpreted. A difficult subject, as an instance of a hypernet is the running of the program over time, with a subset of all hypernets representing all possible paths. This is the equivalent of building a non-deterministic simulator in a deterministic environment. Exponential in nature the runtime grows beyond exponential into undecidable. Thus some instances of hypernets have been decided and actualized to produce some views into hypergraphs in this paper. Hypergraphs themselves are difficult to project in 2D and 3D.

5

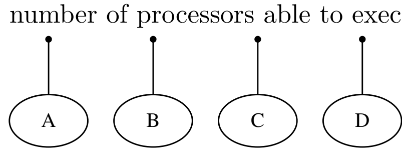
Analysis of the simple problem of selecting N *random* things from a set of $Variables$ with $Values$.

6 Background

An vector of objects is a simple array that is addressable by number. Usually included in all fundamental programming languages, this allows a number to be used to get and set the value at an address in memory.

A hypergraph is an simply a vector of type T nodes. The inverse lookup

and a vector of type T node values to numbers. Access time for arrays is $O(c)$. In both get functions the access time is $O(n)$ where n is the magnitude of the edge/odometer. As all objects are vector arrays, these loops can all potentially be executed in parallel changing the runtime to $O(n/p)$ where p is the number of processors able to execute the address lookup.



An odometer, hypergraph, and advancement function can represent all of the different programming and mathematical structures as both discrete points and functions to compute them. Collecting hyperedges from an odometer and advancement function until the function returns False is equivalent to exploring a *space* in its entirety. Here the dimensions of the enumerated hyperedges are restricted to one expressing the enumeration of a vector.

```
def OdometerAsList(hypergraph,odometer):
    if len(odometer) == 1:
        if odometer[0] + 1 < len(hypergraph):
            odometer[0] += 1
            return True
    return False
```

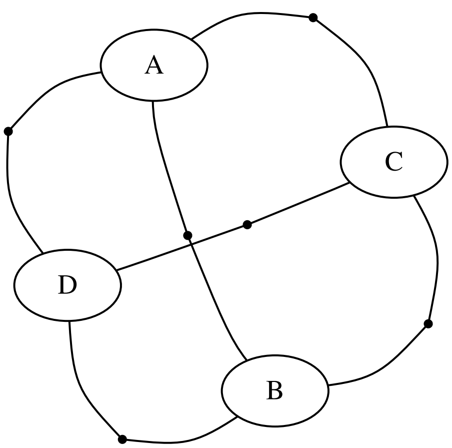
```
def EnumerateOdometer(hypergraph,odometer,func):
    returnValue = [ getHyperEdge(hypergraph,odometer) ]
    while func(hypergraph,odometer):
        returnValue.append( getHyperEdge(hypergraph,odometer))
    return returnValue

hypergraph = makeHyperGraph(sorted("ABCD"))
odometer = [0]
func = OdometerAsList
hyperedges_as_list = EnumerateOdometer(hypergraph,odometer,func)
```

Sorting a list is now equivalent to finding the correct odometer encoding expressing the enumeration of the hypergraph as a sorted list. A function which advances the odometer from the current object to the next largest object which takes N enumerations is equal in representation as an odometer of length N where the next index contains the index of the node in the hypergraph which is next in the sort order. Interpretation depends upon the meta-context of the program using the hypergraph.

7 Graph Representation

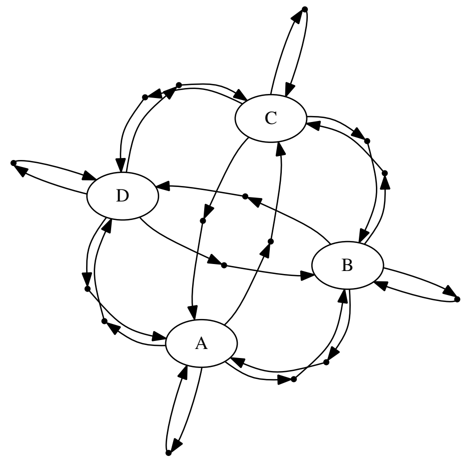
r0.5



A fully connected undirected graph where every node is connected to every node is represented correctly by assigning the numbers that come in the second position of the odometer to values equal to the first position plus one.

When the restriction is removed the representation changes to be equivalent to representing a fully connected digraph where every there is a directional edge from every node to every node.

```
def OdometerAsFullUndirectedGraph(hypergraph,odometer):
if len(odometer) == 2:
if odometer[1] + 1 < len(hypergraph):
odometer[1] += 1
return True
else:
if odometer[0] + 2 < len(hypergraph):
odometer[0] += 1
odometer[1] = odometer[0] +1
return True
return False
```




```
def OdometerAsFullDirectedGraph(hypergraph,odometer):
if len(odometer) == 2:
if odometer[1] + 1 < len(hypergraph):
odometer[1] += 1
return True
else:
if odometer[0] + 1 < len(hypergraph):
odometer[0] += 1
odometer[1] = 0
return True
return False
```

Notice that the restriction lifting now gives allows the edge $\{A \rightarrow A\}$ which has mathematical relevance but may make no sense in the context of the graph being interpreted. Thus selecting the correct mathematical representation as an expressive function that restricts the domain properly is critical in ensuring the enumeration represents the correct model. The function which advances odometers both defines what the next discrete point in the space will be and also sets the bounds of the mathematical space.

8 Forest of Trees

9 Uniform selection from multiple variables

Sampling N items from a set which is larger than is feasible to compute is a complex problem as the quality of sample points is a significant factor in the quality of the final data set. Each variable has a domain of values that can be mapped to a hypergraph. A vector of numbers is computed that contains the size of the domain of each variable. Each index in the odometer represents the index in the vector of hypergraphs. Each value in the odometer represents the node to select from the hypergraph. Thus there is only one value selected for each variable for a given odometer.

Odometers can map a space of size $\Pi_{i=1}^{V} D_i$ unto a single dimensional number line-path. This path can then be sliced into the number of samples. The odometer is advanced by the distance between sample points on the number line. This transformation is linear-polynomial to take N samples from a mapping whose size exceeds the size of the universe.

```python
def getNextSampleOdometer(odometer, odometer_state, domain_sizes, step_size):
    control = 0
    domain_size = mul_list(domain_sizes)
    step_size = step_size % domain_size

    while control < len(odometer):
        size = domain_sizes[control]
        step = step_size % size
        step_size = step_size // size
        cur_num = odometer[control]
        dir_num = odometer_state[control]

        if step_size %2 == 1:
            cur_num = (size - 1) - cur_num
            if dir_num == 1:
                dir_num = -1
            else:
                dir_num = 1

        cur_num = cur_num + dir_num * step

        if cur_num < 0:
            cur_num +=1
            dir_num = 1
            cur_num = (size -1) - cur_num
            step_size +=1
        if cur_num >= size:
            dir_num =-1
            cur_num = (size -1) - (cur_num - size)
            step_size +=1

        odometer[control] = cur_num
        odometer_state[control] = dir_num

        if step_size == 0:
            return
        control+=1
    return
```

References

[1] G. Ausiello, G. F. Italiano, and U. Nanni. Optimal traversal of directed hypergraphs. Technical report, University of Berkeley, 1992.

[2] C. Berge. *Hypergraphs*. North-Holland Mathematical Library, 1989.

[3] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdag, R. Heaphy, and L. A. Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. Technical Report 1-4244-0910-1/07, Ohio State University and Sandia National Laboratories, Columbus, OH 43210, USA, 2007.

[4] J. Clausen. Branch and bound algorithms - principles and examples. Master's thesis, University of Copenhagen, 1999.

[5] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurk. Parallel hypergraph partitioning for scientific computing. 2006.

[6] A. Ducournau and A. Bretto. Random walks in directed hypergraphs and application to semi-supervised image segmentation. *Computer Vision and Image Understanding*, pages 91–102, 2014.

[7] P. P. Fuchs. Permutation odometers. 2016.

[8] H. Haverkort. An inventory of three-dimensional hilbert space-filling curves. Master's thesis, Eindhoven University of Technology, 2011.

[9] J. K. Lawder. Calculation of mappings between one and n-dimensional values using the hilbert space-filling curve. Technical Report JL1/00, Birkbeck College University of London, Malet Street London WC1E 7HX United Kingdom, August 2000.

[10] M. F. Mokbel, W. G. Aref, and I. Kamel. Performance of multi-dimensional space-filling curves. Master's thesis, Purdue University, 2002.

[11] B. Molnár. Applications of hypergraphs in informatics: A survey and opportunities for research. *Annales Univ. Sci. Budapest.*, July 2014.

[12] L. R. Nielsen, K. A. Anderson, and D. Pretolani. Finding the k shortest hyperpaths: algorithms and applications. Technical report, University of Aarhus, 2004.

[13] C. of the ACM. How to partition a billion-node graph. 2014.

[14] A. Rita and T. Murali. Pathway analysis with signaling hypergraphs. *Publications of the ACM*, September 2014.

[15] S. E. Schaeffer. Survey: Graph clustering. (I):27–64, May 2007.

[16] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. 2012.

[17] D. Zhou, J. Huang, and B. Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. Technical report, School of Computer Science University of Waterloo, 2006.