Graph Compression Using Quadtrees

Dr. Amlan Chatterjee

Computer Science Department
California State University, Dominguez Hills

April 27, 2016

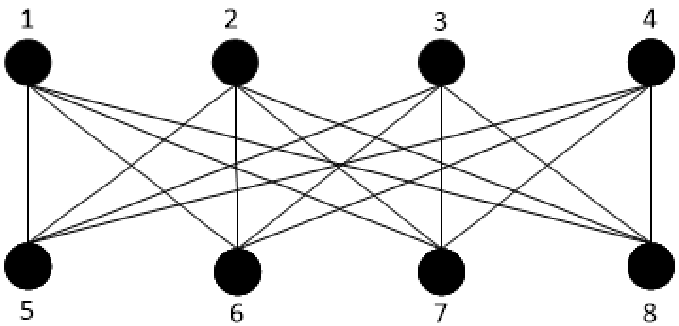
Outline

1. Storing graphs
2. Quadtree representation of graphs
3. Special graphs
4. Modifying graphs for efficient storage
5. Hybrid approach
6. Other representations

Outline

Storing Graphs

- Consider the following graph G = (V,E)

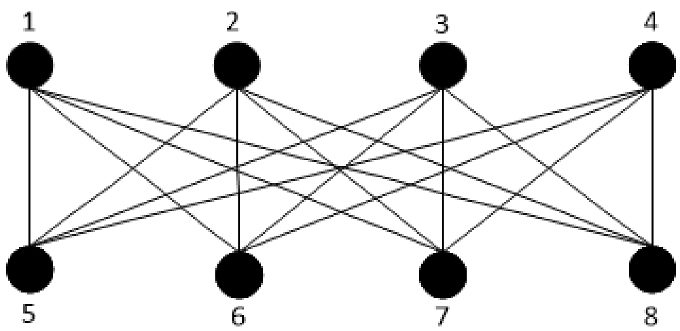


The adjacency matrix representation is given by:

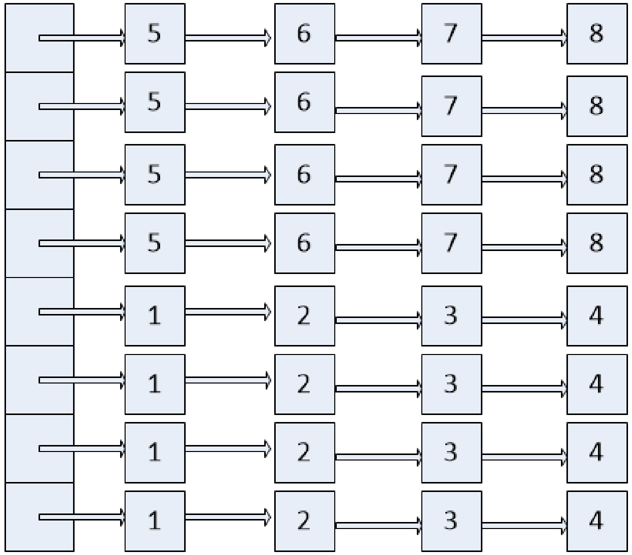
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0

Storing Graphs

- Consider the following graph G = (V,E)

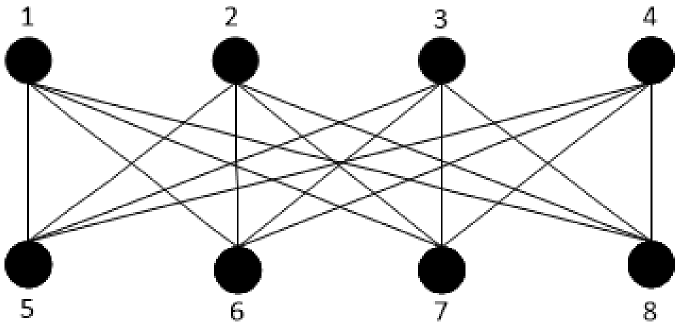


The adjacency list representation is given by:

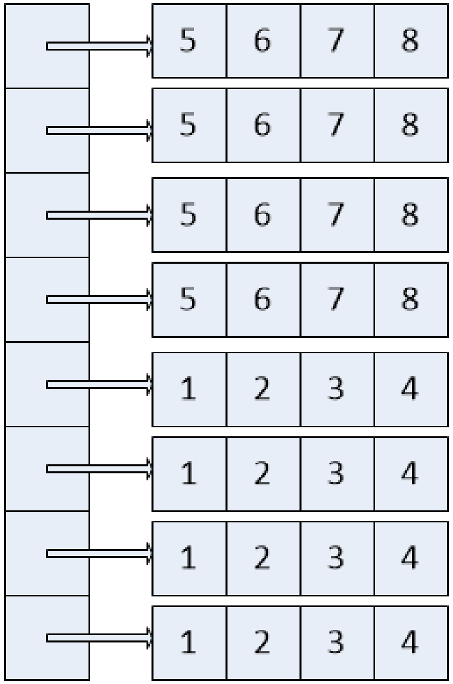


Storing Graphs

- Consider the following graph G = (V,E)



The adjacency array representation is given by:



Storing Graphs (contd.)

- The values in the adjacency matrix can be stored using boolean data type. For the sample graph G = (V, E), where |V| = 8, it would require 8x8 = 64 bytes.
- Since the value is either 0 or 1, using bits instead of boolean the size of the adjacency matrix can be reduced
- Size required to store adjacency matrix using bit array for the sample graph G:

  (nxn)/8 bytes = 8x8/8 bytes

  = 8 bytes

Storing Graphs (contd.)

- For the sample graph G = (V, E), where |V| = n and |E| = m, using boolean data type and assuming 64-bit pointer (i.e., 8 byte pointer), the space required for the adjacency list is:

  2m*64 + 2mlog(n) + nlogn

  (size of pointers) (node numbers) (size of each list)

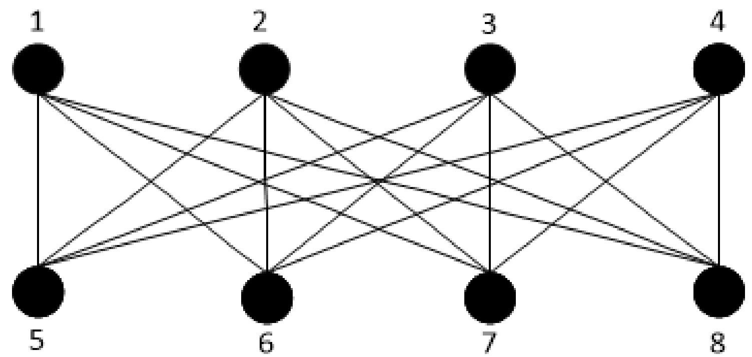
- Similarly, the space required for the adjacency array is:

  n*64 + 2mlog(n) + nlogn

  (size of pointers) (node numbers) (size of each list)

Storing Graphs (contd.)

Considering the previous example graph
G = (V,E)



For the above graph, n = 8, m = 16.

The adjacency matrix size is: 64 bits

The adjacency list size is: 2168 bits

The adjacency array size is: 632 bits

Other techniques to store graphs

- Other than using adjacency matrix, adjacency list and adjacency array, the following are some other common techniques to store graphs

- Unordered edge sequences:
  – The data is represented as pair values, each indicating the pair of vertices where an edge exists
- Incidence matrix
  – Two edges are said to be incident if they share a vertex; incidence matrix contains data with respect to edges

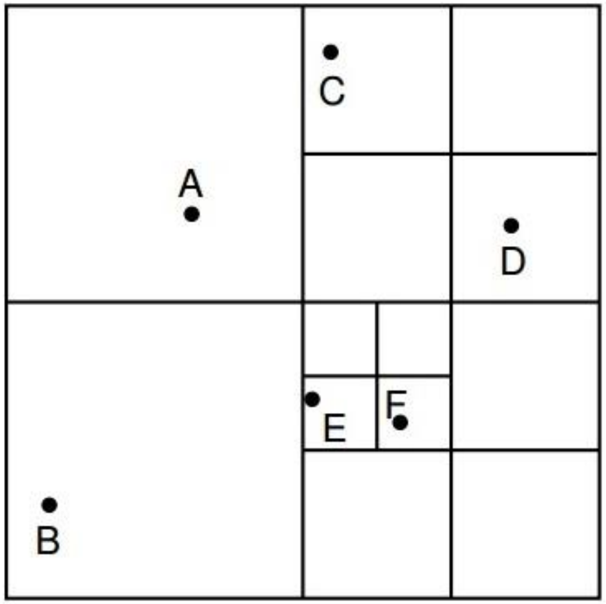
Outline

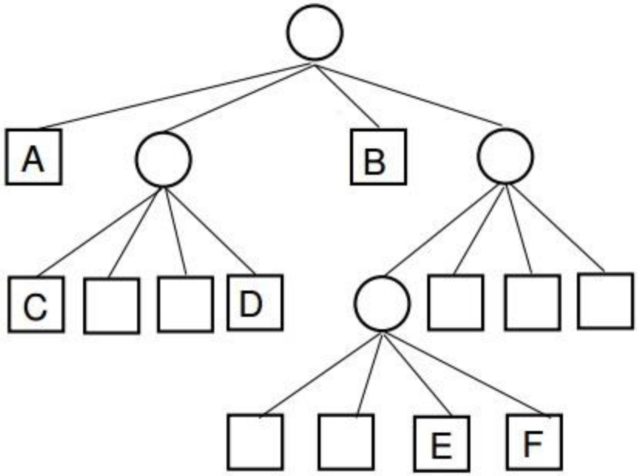
Quadtree

- Quadtree is a data structure which is used to normally represent images using partitioning of the two dimensional space by recursively subdividing into four quadrants or regions; each internal node of the quadtree has exactly four children

- Quadtrees can also be used to store graphs efficiently

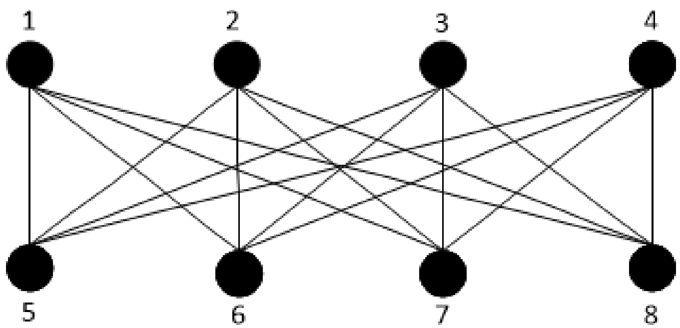
Sample Data Points in a 2-D space & Quadtree





Quadtree representation of a graph

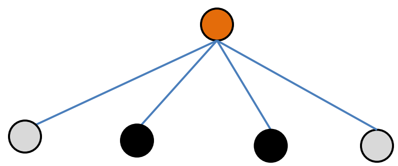
Graph G = (V,E)



The adjacency matrix: size 64bits

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0

Quadtree representation:



Size: 10 bits (5 elements)

Quadtree representation of a graph (contd.)

- Given a quadtree, the entire graph information can be stored in the form of an array using bits
- The quadrants are converted and stored according to the row major order of the adjacency matrix
- The contents of the bit array are stored as follows

  0: all 0's in quadrant

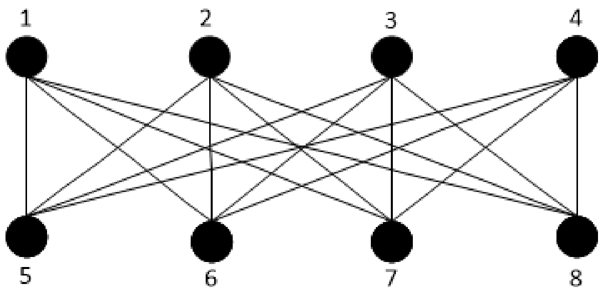
  1: all 1's in quadrant

  2: 0’s in diagonal, and rest 1’s

  3: the quadrant needs to be expanded further
- Since there are only 4 types of values, using 2 bits for each is enough

Quadtree representation of a graph (contd.)

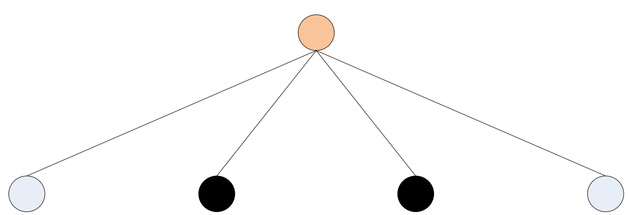
Graph G = (V,E)



The adjacency matrix: size 64bits

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0

Quadtree representation:



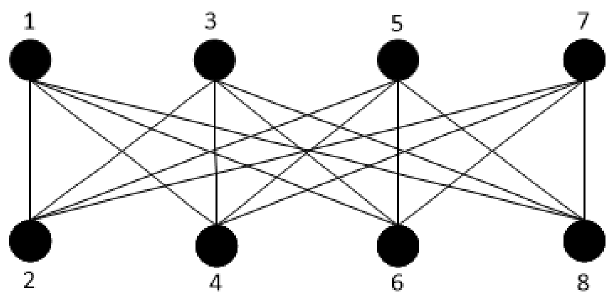
Byte representation of the Quadtree:

Q = {3, 0, 1, 1, 0}

Representing Graphs

- Consider the following graph

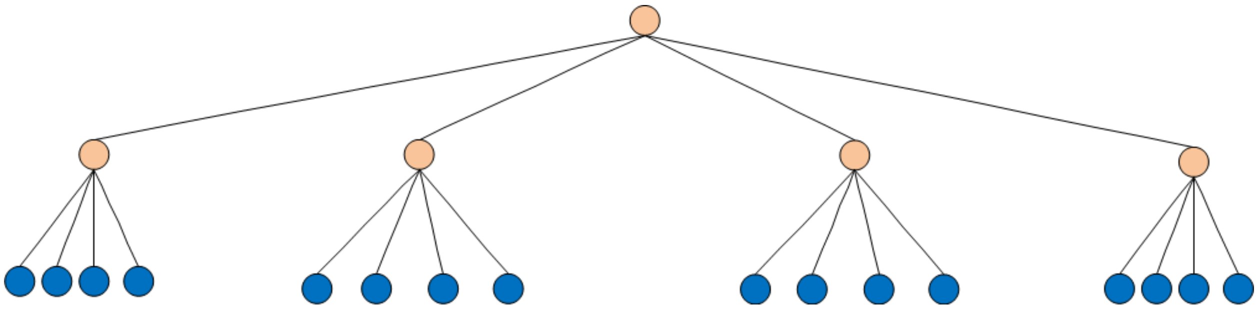
Graph G = (V,E)



The adjacency matrix:

0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0

Quadtree:

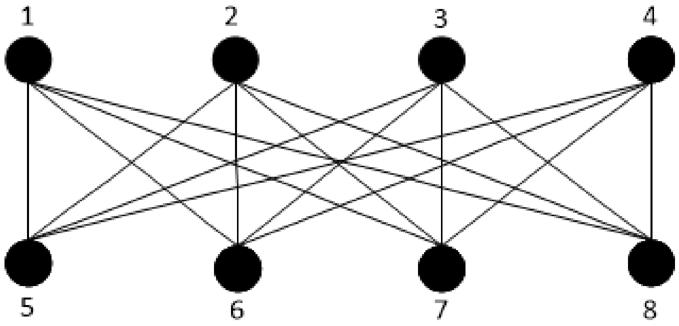


Size: 21 elements: 42 bits

Representing Graphs (contd.)

- The previous example graph using different numbering

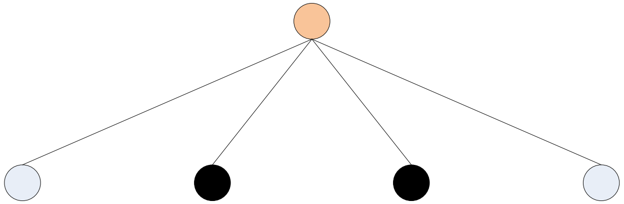
Graph G = (V,E)



The adjacency matrix:

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0

Quadtree:



Size: 5 elements: 10 bits

Numbering matters

- The numbering of the nodes of the graph G = (V, E) matters when represented using quadtrees
- The adjacency matrix representation varies according to the node numbering
- In quadtrees, quadrants with uniform values don't expand further, while others do increasing the overall space required

Problem statement

- The size required for representing the graphs is directly proportional to the number of quadrants that are non-uniform
- Since the adjacency matrix varies with the numbering of the nodes, some combinations might be better than others
- Therefore, the problem at hand can be stated as: Given a graph G = (V,E), does there exist a numbering Y: v -> v', such that the number of quadrants that have to be expanded is the smallest

Outline

Special graphs

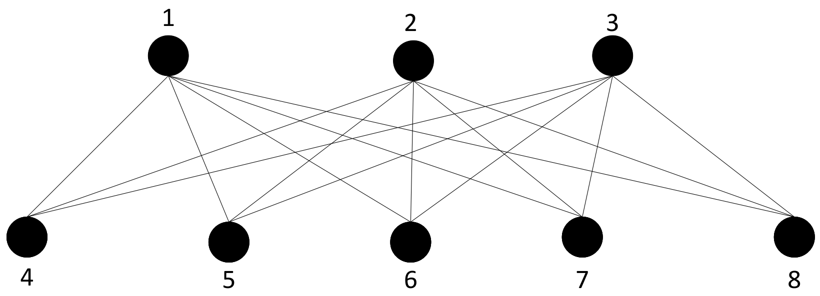
Following are some of the special graphs that we consider for representing using quadtrees:

a. Complete bipartite graph
b. Complete k-partite graph
c. Block graphs
d. Chordal graphs

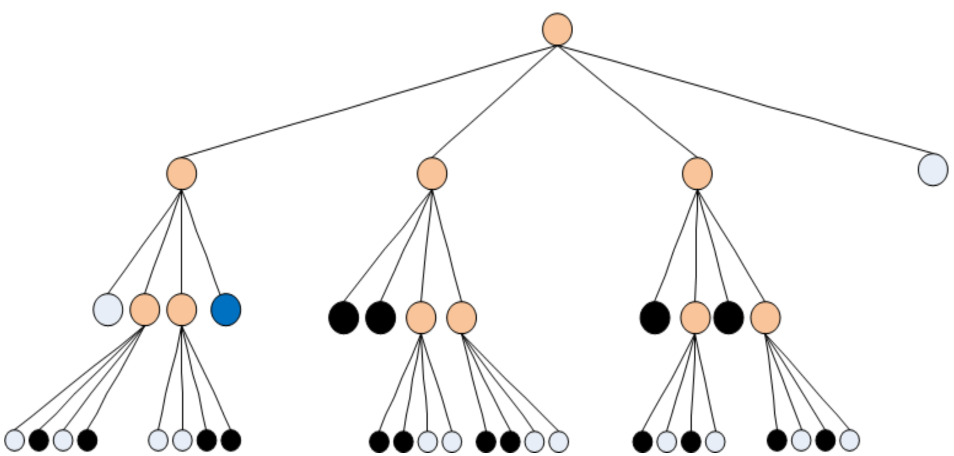
Special graphs (contd.)

a. Complete bipartite graph

Graph G = (V,E)



Quadtree:



The adjacency matrix:

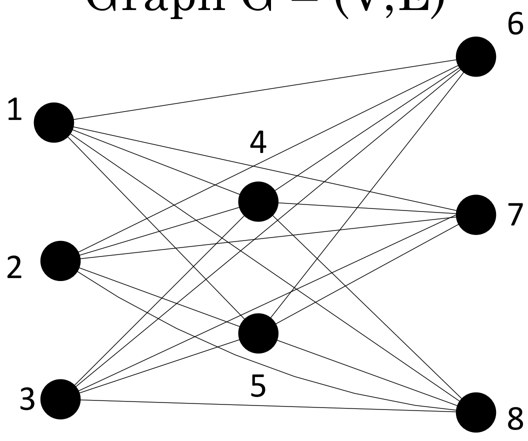
0	0	0	1	1	1	1	1
0	0	0	1	1	1	1	1
0	0	0	1	1	1	1	1
1	1	1	0	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	0	0	0	0	0

Size:
Adjacency matrix: 64 bits
Quadtree: 82 bits

Special graphs (contd.)

b. Complete k-partite graph

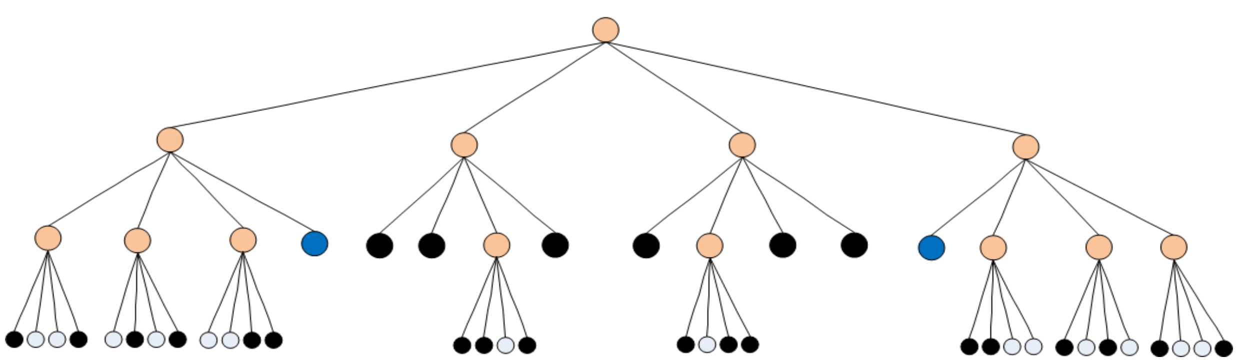
Graph G = (V,E)



The adjacency matrix:

0	0	0	1	1	1	1	1
0	0	0	1	1	1	1	1
0	0	0	1	1	1	1	1
1	1	1	0	0	1	1	1
1	1	1	0	0	1	1	1
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0

Quadtree:

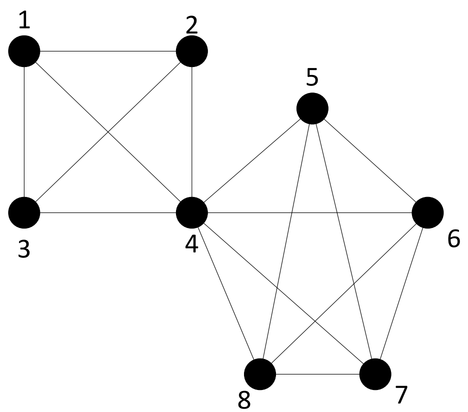


Size:
Adjacency matrix: 64 bits
Quadtree: 106 bits

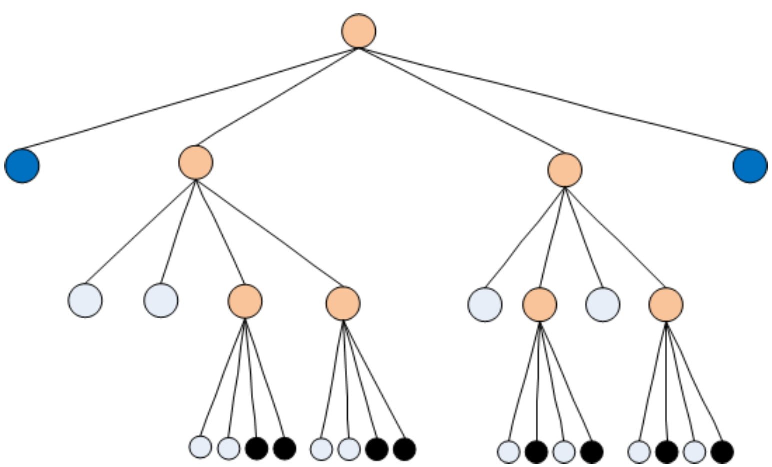
Special graphs (contd.)

c. Block graphs

Graph G = (V,E)



Quadtree:



The adjacency matrix:

0	1	1	1	0	0	0	0
1	0	1	1	0	0	0	0
1	1	0	1	0	0	0	0
1	1	1	0	1	1	1	1
0	0	0	1	0	1	1	1
0	0	0	1	1	0	1	1
0	0	0	1	1	1	0	1
0	0	0	1	1	1	1	0

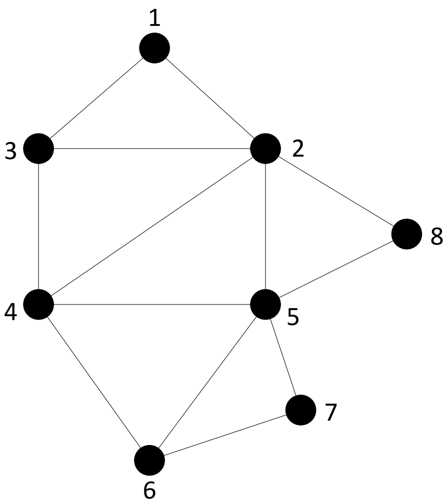
Size:
Adjacency matrix: 64 bits
Quadtree: 58 bits

Special graphs (contd.)

d. Chordal graphs

- Definition: An undirected graph G = (V, E) is chordal (triangulated, rigid circuit) if every cycle of length greater than three has a chord: namely an edge connecting two non-consecutive vertices on the cycle.

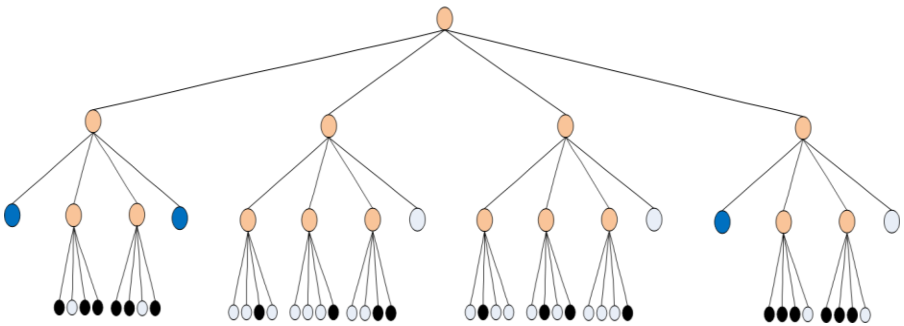
Given graph G



The adjacency matrix:

0	1	1	0	0	0	0	0
1	0	1	1	1	0	0	1
1	1	0	1	0	0	0	0
0	1	1	0	1	1	0	0
0	1	0	1	0	1	1	1
0	0	0	1	1	0	1	0
0	0	0	0	1	1	0	0
0	1	0	0	1	0	0	0

Quadtree representation



Size:
Adjacency matrix: 64 bits
Quadtree: 61 elements; 122 bits

Special graphs (contd.)

Chordal graphs (contd.)

In a graph $G = (V, E)$, a vertex v is called simplicial if and only if the subgraph of G induced by the vertex set $\{v\} \cup N(v)$ is a complete graph, where $N(v)$ is the set of neighboring vertices of v.

A graph G on n vertices is said to have a perfect elimination ordering if and only if there is an ordering $\{v_1, \ldots v_n\}$ of G's vertices, such that each v_i is simplicial in the subgraph induced by the vertices $\{v_1, \ldots v_i\}$.

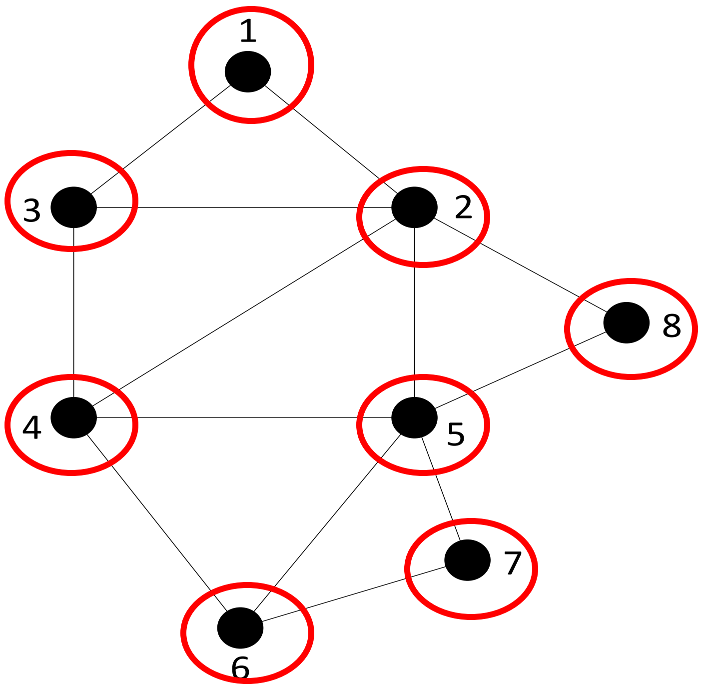
The chordal graphs may also be characterized as the graphs that have perfect elimination orderings.

Special graphs (contd.)

Chordal graphs (contd.)

Perfect Elimination Ordering (PEO): Using the sample graph G=(V,E), a PEO is shown below

Given graph G



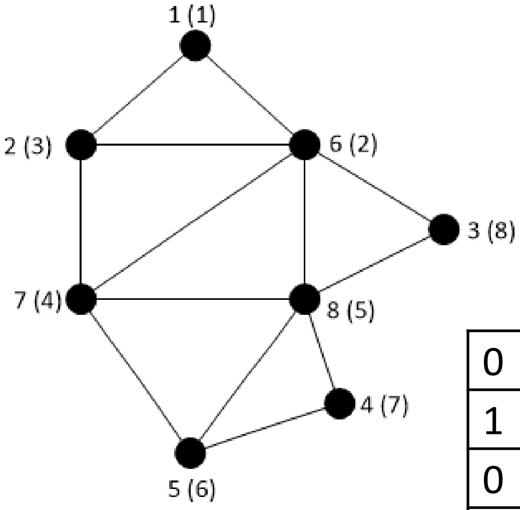
PEO: 1 ,3 , 8, 7, 6, 2 , 4, 5

Special graphs (contd.)

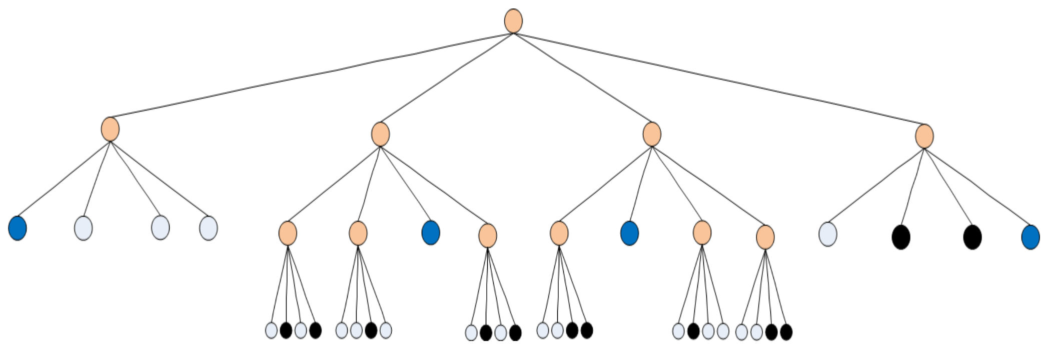
Chordal graphs (contd.)

Renumbering the nodes according to the PEO; the old numbers are shown in parentheses

Given graph G



Quadtree representation



The adjacency matrix:

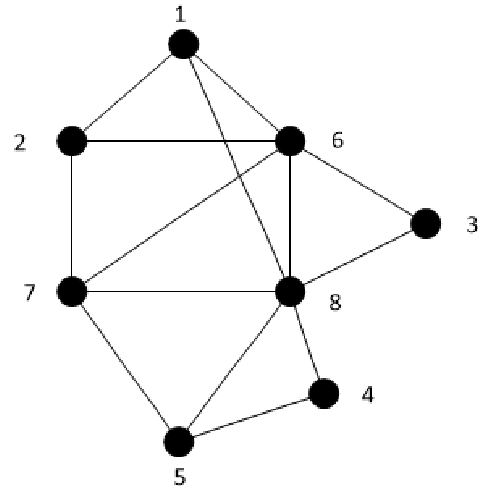
0	1	0	0	0	1	0	0
1	0	0	0	0	1	1	0
0	0	0	0	0	1	0	1
0	0	0	0	1	0	0	1
0	0	0	1	0	0	1	1
1	1	1	0	0	0	1	1
0	1	0	0	1	1	0	1
0	0	1	1	1	1	1	0

Size: 45 elements: 90 bits
So, the size decreases by 32 bits by renumbering according to PEO

Outline

Modifying graphs

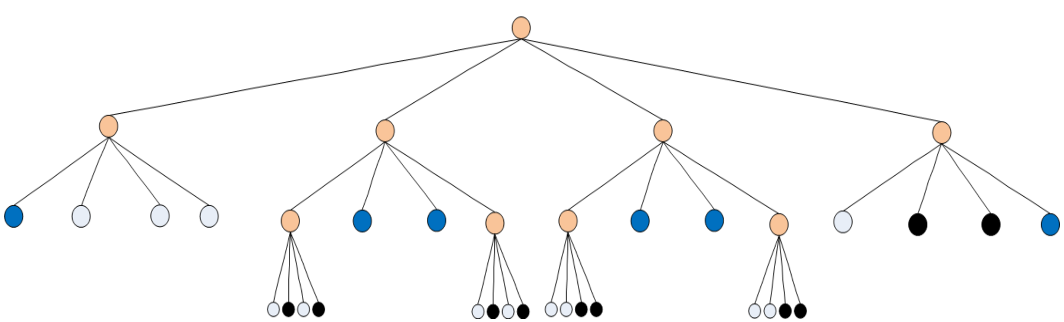
Consider the following graph



The adjacency matrix:

0	1	0	0	0	1	0	1
1	0	0	0	0	1	1	0
0	0	0	0	0	1	0	1
0	0	0	0	1	0	0	1
0	0	0	1	0	0	1	1
1	1	1	0	0	0	1	1
0	1	0	0	1	1	0	1
1	0	1	1	1	1	1	0

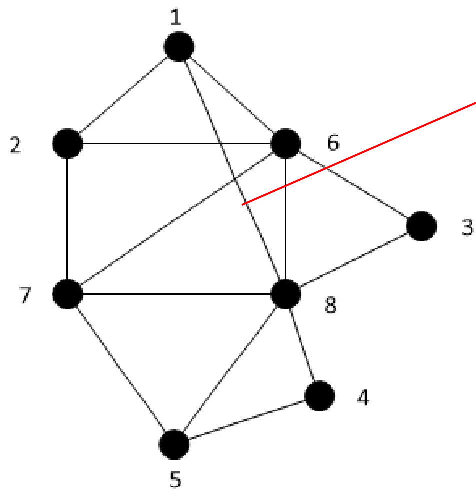
Quadtree representation



Size: 37 elements: 74 bits

Modifying graphs

Consider the following graph

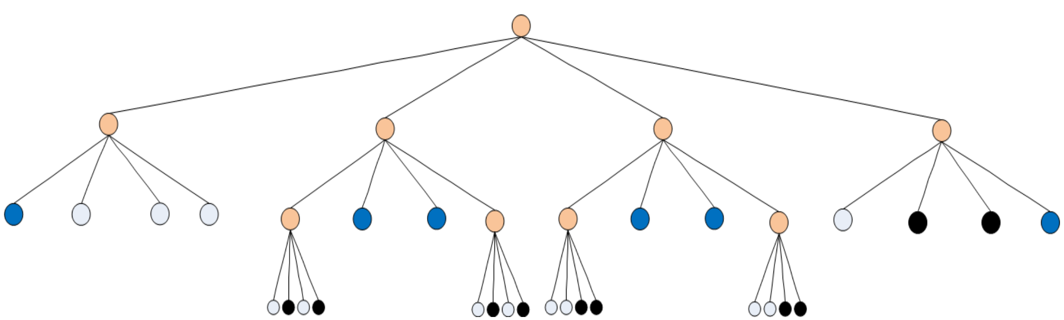


This edge has been added to the PEO numbered chordal graph

The adjacency matrix:

0	1	0	0	0	1	0	1
1	0	0	0	0	1	1	0
0	0	0	0	0	1	0	1
0	0	0	0	1	0	0	1
0	0	0	1	0	0	1	1
1	1	1	0	0	0	1	1
0	1	0	0	1	1	0	1
1	0	1	1	1	1	1	0

Quadtree representation



Size: 37 elements: 74 bits

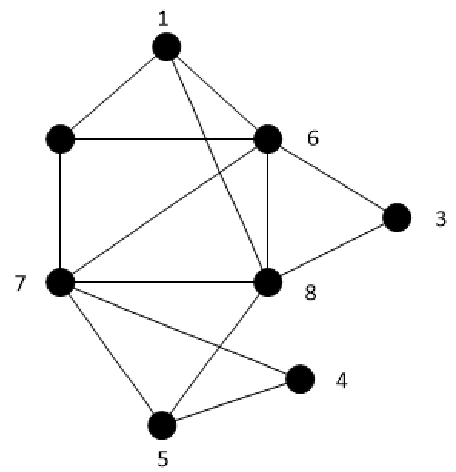
Additional edge info: 6 bits

Total space required to store the PEO numbered chordal graph: 80 bits

Space reduced by: 10 bits

Modifying graphs (Contd.)

Consider the following graph

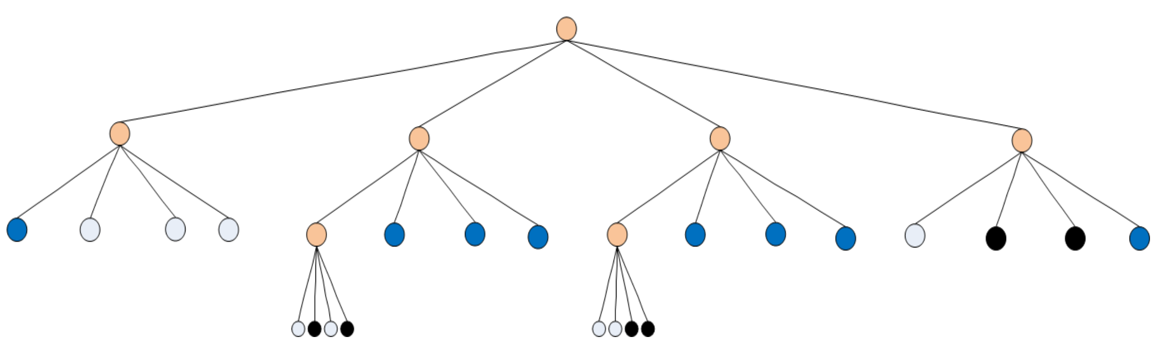


The adjacency matrix:

0	1	0	0	0	1	0	1
1	0	0	0	0	1	1	0
0	0	0	0	0	1	0	1
0	0	0	0	1	0	1	0
0	0	0	1	0	0	1	1
1	1	1	0	0	0	1	1
0	1	0	1	1	1	0	1
1	0	1	0	1	1	1	0

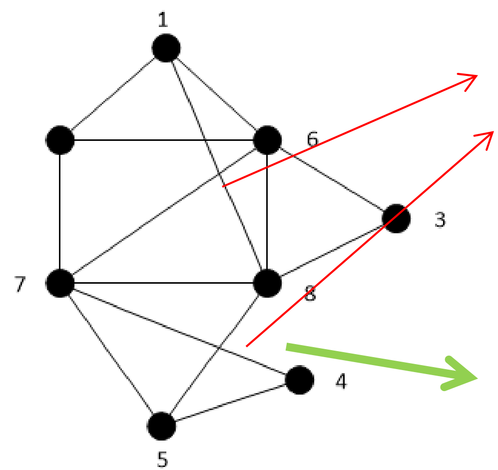
Size: 29 elements: 58 bits

Quadtree representation



Modifying graphs (Contd.)

Consider the following graph



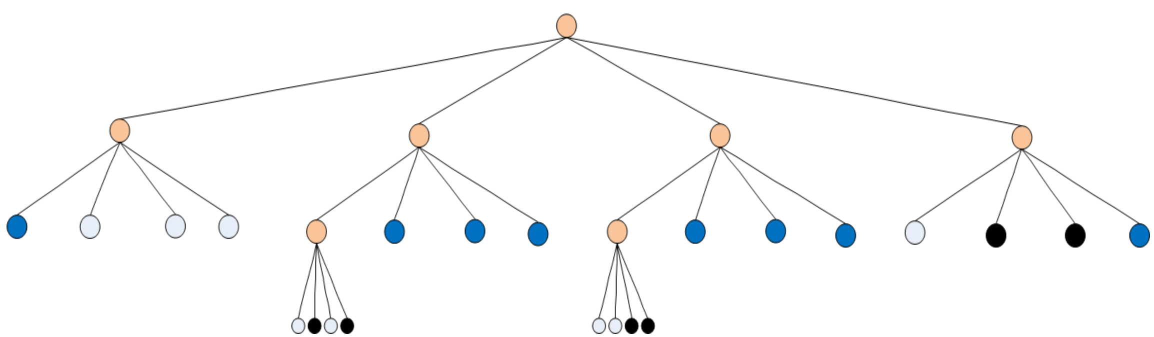
These edges have been added to the PEO numbered chordal graph

Edge (4,8) has been removed

The adjacency matrix:

0	1	0	0	0	1	0	1
1	0	0	0	0	1	1	0
0	0	0	0	0	1	0	1
0	0	0	0	1	0	1	0
0	0	0	1	0	0	1	1
1	1	1	0	0	0	1	1
0	1	0	1	1	1	0	1
1	0	1	0	1	1	1	0

Quadtree representation



Size: 29 elements: 58 bits

Edge information for 3 edges need to be stored (2 removed, 1 added)
Extra space required: 18 bits (6 bits per edge)

Total space: 76 bits

Space Reduced by : 14 bits

Modifying graphs (Contd.)

- Further modifications can be made to the chordal graph to reduce space required
- In addition to adding (1,8), (4,7) and removing (4,8), the following needs to change
  - Add (2,5)
  - Remove (2,6)
- These changes reduce the quadtree size to 42 bits; additional 30 bits are required to store the modified edge information
- The total size for this case is 72 bits, which is significantly reduced from the original size of 122 bits for the chordal graph

Outline

Hybrid approach

- For many cases, the quadtree representation for graphs of size 8 nodes require more than 64 bits, which is inefficient compared to the adjacency matrix representation
- However, for larger graphs, the quadtree approach is efficient compared to other data structures
- Even for larger graphs, when the quadrant reduces to 8x8 bits, the quadtree would require more space for further reductions
- Therefore, a hybrid approach, where the recursive division of the quadrants stop whenever the quadrant size reaches 8x8 is a better technique

Hybrid approach (Contd.)

- In the byte representation of the quadtree, an additional bit for each node is required to indicate whether the quadrant is further expanded or represented using adjacency matrix

- Although this would need additional bits, overall the space required decreases.

Outline

1-bit coding

- Real-world graphs are usually sparse, and most of the quadrants consist of just 0's in them.
- Hence, instead of using 2-bits to represent each element, the method can be modified to use 1-bit for each element; in this case a quadrant with all 0's is represented by a 0, else it is broken into smaller quadrants which is denoted by a 1.

Comparison of sizes: 1024 node graph

Technique	Size (bits)	% Compared to Adjacency matrix
Adjacency Matrix	1048576	100
2-bit	261370	24.93
1-bit	130873	12.48

What about using 3 bit coding?

Code	Meaning
000	All 0's
001	All 1's
010	All 1's except diagonal
011	All 0's except one 1
100	All 0's except two 1's
101	All 0's except three 1's
110	Raw Data
111	Divide Further

Comparison of sizes: 1024 node graph

Technique	Size (bits)	% Compared to Adjacency matrix
Adjacency Matrix	1048576	100
2-bit	261370	24.93
1-bit	130873	12.48
3-bit	300227	28.63
3-bit hybrid	125538	11.97
3-bit hybrid + folding	62889	6

Topological Information helps?

- The patterns used for the 3-bit compression are fixed for all graphs
- Using the topology for specific graphs or domains, relevant patterns can be chosen
- This provides an additional 30% compression

Conclusion

- Comparing with the adjacency matrix representation, all the techniques achieve more than 70% compression.
- The techniques with 1-bit and 3-bits outperform the 2-bit one
- Since real-world graphs are sparse, the 3-bit technique does not reach it's potential with many of the patterns reporting low counts of occurrences.
- In other domains, where the graphs are denser, the 3-bit compression schemes should be able to take advantage of the common patterns and perform better.

- Questions..??

References

[1] L.S. Chandran, L. Ibarra, F. Ruskey, J. Sawada, Generating and characterizing the perfect elimination orderings of a chordal graph, Theoretical Computer Science, 307 (2003), pp. 303–317

[2] H. Samet, Using quadtree to represent spatial data, NATO ASI Series, Vol. F18, pp.229 – 247, 1985

[3] M. Nelson, S. Radhakrishnan, A. Chatterjee, and C. N. Sekharan. On compressing massive streaming graphs with Quadtrees, 2015 IEEE International Conference on Big Data, pages 2409–2417, Oct 2015.