AN EXPERIMENTAL ANALYSIS OF A COMPACT GRAPH REPRESENTATION

Endrias Kahssay

WHY GRAPH COMPRESSION?

Graphs are getting larger (billons of edges) and take a large amount of memory to store.

A compression scheme can result in them fitting on RAM / cache.

Large performance penalty for an access to disk for graphs that don’t fit.

THE ORDER OF VERTICES MATTERS

Most graph algorithms access neighbors of a vertex u after accessing u.

We want a representation that puts edges of neighbors of vertices to be close to each other spatially.

In practice, this means making the labels of related vertices as close as possible.

PAPER CONTRIBUTION

This paper empirically evaluates a representation that has both compression and good order.

Compression of edges using consecutive edge difference.

Order based on separator trees; a structure that attempts to encode a hierarchical relationship among nodes.

SEPARATORS

An edge-separator is a set of edges (vertices) whose removal partitions the graph into two almost equal sizes regions.

The min separator for a graph is one that minimizes the set of edges (vertices) to be removed.

We say that a graph has a good separator if its significantly better than a random graph.

SEPERATORS IN REAL GRAPHS

Real world graphs have good separators because they are based on communities!

Include citation graphs, phone call graphs, friendship relationships, etc.

Useful for partioning graphs for parallel processing, clustering, etc.

COMPRESSED REPRESENTATION

Rep: an encoding where neighbors for each vertex are stored in a difference-encoded adjacency list.

I.e if a vertex has a sorted neighbor v1, v2, v3 , .., we store v1 – v, v2 – v1 contagiously in bits.

Difference stored using a logarithmic code which uses O(logd) bits to encode a diff of size d.

REPRESENTATION

Adjacency list concatenated to form an adjacency table.

An indexing structure is used to access the adjacency list for vertices, which is also compressed.

Not theoretically optimal, but works well in practice.

REPRESENTATION

1) Generate edge separator tree for the graph.

2) Label in the vertices in-order across the leaves.

3) Use an adjacency table to represent labeled graph.

In previous work, the authors describe an $O(n)$ bit encoding with $O(1)$ access time for graphs satisfying $O(n^c)$ for $c < 1$ separator bound.

IMPLMENTATION: SEPARATOR TREES

Collapses edges until a single vertex remains.

Many heuristics to collapse edges, paper used $w(E_{AB})/ s(A)s(B)$.

$w(E_{AB})$ is the number of edges between A and B, and S(B) is the number of vertices in the multi vertex B.

INDEXING STRUCTURE

Used to map a vertex to its start location.

Tradeoff between space and look speed.

Uses semi-direct-16, which stores the start location for 16 vertices in 5 32-bit words.

Based on storing offsets from some set of vertices encoded, and if an offset doesn't fit, a pointer is stored to it instead.

CODES AND DECODING

Originally used gamma code, which represents an integer d using a unary code for [logd] followed by binary code for d – $2^{[\log d]}$.

Decoding gamma code is expensive (even though O(1)).

Developed snip, nibble, and byte code, optimized for machine word access based on continue bits and chunking.

DYNAMIC VERSION

In dynamic version, the edge list of each vertex is dynamically managed, so can’t be stored contagiously.

Initially array with one memory block for each vertex.

If additional memory is needed, vertex gets additional blocks from a pool.

Blocks connected in a linked list.

DYNAMIC VERSION

Hashing technique to reduce size of pointers to 8 bits.

A separate pool for each 1024 vertices for locality.

Caching to avoid repetitively encoding and decoding neighbors.

EXPERIMENTS

Test graphs include 3d mesh graphs, street connectivity graphs, webpage connectivity from Google, programming contest, and circuits.

Graph	Vtxs	Edges	Max Degree	Source
auto	448695	6629222	37	3D mesh [35]
feocean	143437	819186	6	3D mesh [35]
m14b	214765	3358036	40	3D mesh [35]
ibm17	185495	4471432	150	circuit [1]
ibm18	210613	4443720	173	circuit [1]
CA	1971281	5533214	12	street map [34]
PA	1090920	3083796	9	street map [34]
googleI	916428	5105039	6326	web links [10]
googleO	916428	5105039	456	web links [10]
lucent	112969	363278	423	routers [25]
scan	228298	640336	1937	routers [25]

MACHINE SETUP

Two machines, each with 32-bit processors.

A .7GHz Pentium III processor with .1Ghz bus and 1GB of ram (cache line 32 bytes).

2.4Ghz, Pentium 4, with 4 processors, .8GHz bus, and 1GB of ram (cache line 128 bytes). Supports quad vectorization, and hardware prefetching.

BENCHMARK

Benchmark on depth-first-search, and also varying edge insertion order.

DFS visits in a non trivial order.

Use a character array to mark visited vertices.

INSERTING EDGES

Consider three ways of inserting edges:

Linear: Insert all out edges for first vertex, then second, and so on.

Transpose: insert all in edge for first vertex, etc.

Random: random order of edges.

PERFORMANCE FOR DFS, STATIC

Graph	Array			Our Structure									
	Rand	Sep		Byte		Nibble		Snip		Gamma		DiffByte	
	T_1	T/T_1	Space	T/T_1	Space	T/T_1	Space	T/T_1	Space	T/T_1	Space	T/T_1	Space
auto	0.268s	0.313	34.17	0.294	10.25	0.585	7.42	0.776	6.99	1.063	7.18	0.399	12.33
feocean	0.048s	0.312	37.60	0.312	12.79	0.604	10.86	0.791	11.12	1.0	11.97	0.374	13.28
m14b	0.103s	0.388	34.05	0.349	10.01	0.728	7.10	0.970	6.55	1.320	6.68	0.504	11.97
ibm17	0.095s	0.536	33.33	0.536	10.19	1.115	7.72	1.400	7.58	1.968	7.70	0.747	12.85
ibm18	0.113s	0.398	33.52	0.442	10.24	0.867	7.53	1.070	7.18	1.469	7.17	0.548	12.16
CA	0.920s	0.126	43.40	0.146	14.77	0.243	10.65	0.293	10.55	0.333	11.25	0.167	14.81
PA	0.487s	0.137	43.32	0.156	14.76	0.258	10.65	0.310	10.60	0.355	11.28	0.178	14.80
lucent	0.030s	0.266	41.95	0.3	14.53	0.5	11.05	0.566	10.79	0.700	11.48	0.333	14.96
scan	0.067s	0.208	43.41	0.253	15.46	0.402	11.84	0.477	11.61	0.552	12.14	0.298	16.46
googleI	0.367s	0.226	37.74	0.258	11.93	0.405	8.39	0.452	7.37	0.539	7.19	0.302	13.39
googleO	0.363s	0.250	37.74	0.278	12.59	0.460	9.72	0.556	9.43	0.702	9.63	0.327	13.28
Avg		0.287	38.202	0.302	12.501	0.561	9.357	0.696	9.07	0.909	9.424	0.380	13.662

Table 2: Performance of our **static** algorithms compared to performance of an adjacency array representation. Space is in bits per edge; time is for a DFS, normalized to the first column, which is given in seconds.

separator-based representation with byte codes is a factor of
3.3 faster than adjacency arrays with random ordering but about 5% slower for the separator ordering.
6.6 more compact than adjacency list.

	Linked List							Our Structure					
	Random Vtx Order			Sep Vtx Order				Space Opt			Time Opt		
Graph	Rand T_1	Trans T/T_1	Lin T/T_1	Rand T/T_1	Trans T/T_1	Lin T/T_1	Space	Block Size	Time T/T_1	Space	Block Size	Time T/T_1	Space
auto	1.160s	0.512	0.260	0.862	0.196	0.093	68.33	16	0.148	9.35	20	0.087	13.31
feocean	0.136s	0.617	0.389	0.801	0.176	0.147	75.21	8	0.227	12.97	10	0.117	14.71
m14b	0.565s	0.442	0.215	0.884	0.184	0.090	68.09	16	0.143	8.92	20	0.086	13.53
ibm17	0.735s	0.571	0.152	0.904	0.357	0.091	66.66	12	0.205	10.53	20	0.118	14.52
ibm18	0.730s	0.524	0.179	0.890	0.276	0.080	67.03	10	0.190	10.13	20	0.108	14.97
CA	1.240s	0.770	0.705	0.616	0.107	0.101	86.80	3	0.170	10.62	5	0.108	15.65
PA	0.660s	0.780	0.701	0.625	0.112	0.109	86.64	3	0.180	10.69	5	0.115	15.64
lucent	0.063s	0.634	0.492	0.730	0.190	0.142	83.90	3	0.285	13.67	6	0.174	20.49
scan	0.117s	0.735	0.555	0.700	0.188	0.128	86.82	3	0.290	15.23	8	0.170	28.19
googleI	0.975s	0.615	0.376	0.774	0.164	0.096	75.49	4	0.211	12.04	16	0.125	28.78
googleO	0.960s	0.651	0.398	0.786	0.162	0.108	75.49	5	0.231	13.54	16	0.123	26.61
Avg		0.623	0.402	0.779	0.192	0.108	76.405		0.207	11.608		0.121	18.763

Table 4: The performance of our **dynamic** algorithms compared to linked lists. For each graph we give the space- and time-optimal block size. Space is in bits per edge; time is for a DFS, normalized to the first column, which is given in seconds.

Label ordering has up to a 7x effect Insertion order has up to a factor of 11x effect.

ALGORITHMIC BENCHMARKS

PageRank (summing neighbors), and Bipartite matching (based on DFS).

Representation	Time (sec)		Space
	PIII	P4	(b/e)
Dyn-B4	30.40	11.05	17.54
Dyn-N4	32.96	12.48	13.28
Dyn-B8	26.55	9.23	19.04
Dyn-N8	30.29	11.25	15.65
Gamma	38.56	15.60	9.63
Snip	34.19	13.38	9.43
Nibble	26.38	10.94	9.72
Byte	21.09	8.04	12.59
ArrayOrdr	21.12	6.38	37.74
ArrayRand	33.83	27.59	37.74
ListOrdr	30.96	6.12	75.49
ListRand	44.56	28.33	75.49

CONCLUSION

Simple and fast separator heuristic works well in practice, and real world graphs have small separators.

Decoding cost is small even for simple graph algorithms like DFS.

Order of vertices makes a big difference in terms of a performance (up to 11x).

Separator works well for other graph representations.

STRENGTHS AND WEAKNESS

Paper explains background information well.

Rigorous experiment section, and admits limitations in comparisons.

Would have been interesting to test other traversal patterns like BFS.

QUESTIONS

What would these results look like if they were tested on a modern machine/ with NUMA?

How well would separators work for laying the graph in a NUMA memory setting?