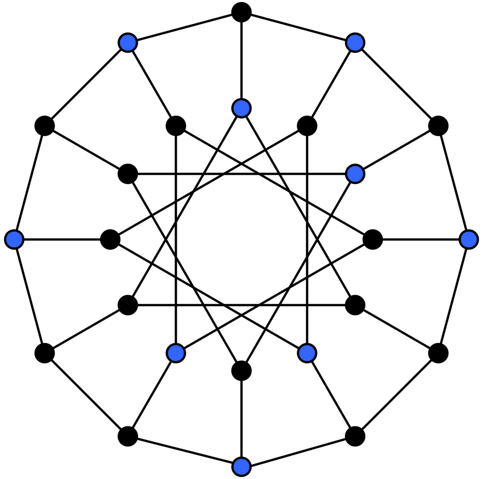
A new algorithm for the Maximum Independent Set Problem

Lochie Ashcroft - s5080439

Problem definition

- The largest subgraph in which no two nodes are adjacent
- The graph is undirected and unweighted
- Inverse of the maximum clique problem



By Life of Riley - Own work, GFDL, https://commons.wikimedia.org/w/index.php?curid=8321640

Literature

- Parallel algorithms
- Stochastic
- Greedy
- Solving for the maximum clique in the complement of the graph
- Simulated Annealing
- Genetic

High Level Algorithm Design - Greedy

- Degree - number of adjacent nodes
- Neighbourhood degree - sum of the neighbours degrees
- Deterministic

1. Sort nodes (degree / neighbourhood degree) [increasing]
2. Insert into the IS if independence holds
3. Repeat 1 & 2 until there are no nodes left

High Level Algorithm Design - LochieSolverV1

- Potential Set
    - Set of nodes that could be added to the IS

1. Loop X times
2. Choose a random node from Potential Set
3. Add to IS
4. Update Potential Set
5. Repeat 1 & 2 until Potential Set is empty

High Level Algorithm Design - LochieSolverV2

- Inspired by Fibonacci Heap
- Object and ObjectWrapper (container) - think nodes and Fibonacci Heap
- Creates multiple independent sets and combines them
- Collapses independent sets

High Level Algorithm Design - LochieSolverV2

```
1. Randomize queue of nodes
2. Perform Initial Pass
3. Perform Initial Merge
4. While X < LIMIT
    a. While queue is not empty
        i. Attempt to insert node in existing sets
        ii. If failed then create a new set
        iii. If time to merge then merge

    B. Collapse a random set
```

Initial Pass

- Attempt to fill the initial sets

Initial Merge

- Attempt to merge the initial sets

LSV2Object - ADT

- Insert(node)
- Merge(LSV2Object)
- Set Adjacency List(Adjacency List)

- Set of nodes
- Adjacency Map
- size

LSV2Object - Insert(node)

- Only insert if the node is not already present
- Don't insert if present in the Adjacency Map with a value > 0

- Update Adjacency Map
- Add node to nodes
- Update Adjacency Map using node’s Adjacent List

LSV2Object - Merge(LSV2Object B)

- Iterate over B’s nodes
- If node not in A’s nodes
    - Add node to A
    - If successful
    - Update A’s Adjacency Map using node’s Adjacency List
    - Update B’s Adjacent Map using node’s Adjacency List
    - Remove node from B’s nodes

LSV2ObjectWrapper - ADT

- constructor(Adjacent List, starting Size)
- add(LSV2Object)
- add()
- getSizeElement(index)
- merge(indexA, indexB)
- mergeAll()
- insert(index, node)
- attemptInsertAll()
- collapse(index)

- size
- best
- List of LSV2Objects
- Adjacency List

LSV2ObjectWrapper - MergeAll()

```
For i in range(size)
    For j in range(i, size)
        If getSizeElement(i) > getSizeElement(j)
            merge(i, j)
        Else
            merge(j, i)
```

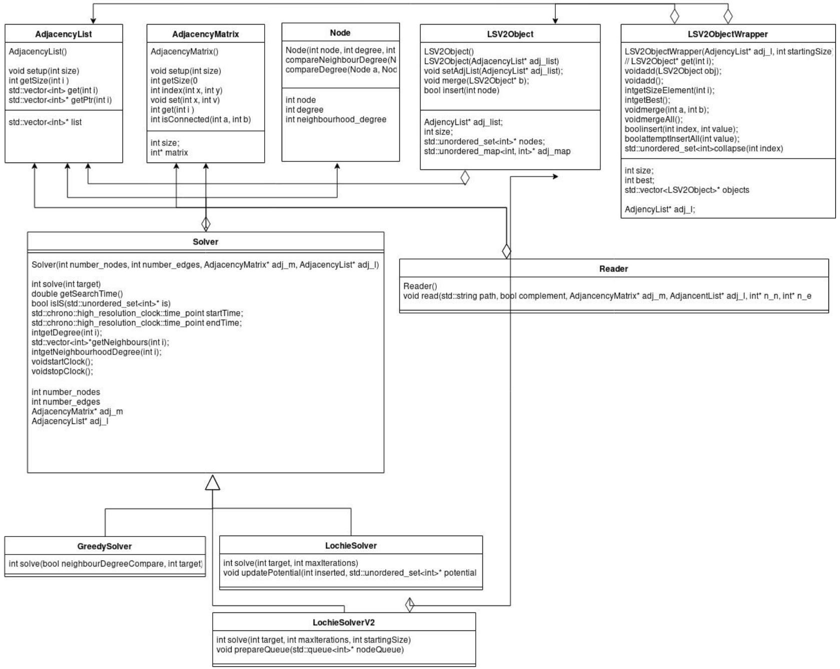
LSV2ObjectWrapper - attemptInsertAll(node)

```
For i in range(size)
    If insert(i, node)
        Update best size if necessary
        Return true
Return false
```

Implementation Details - General

- Object-Oriented approach
- Solvers accept a target value for early termination
- Focus on speed
- Adjacency List and Matrix
- Pointers, dynamic memory allocation
- STL
  - std::unordered_set<T>
  - std::unordered_map<T>
  - std::vector<T>
  - std::queue<T>
  - std::priority_queue<T>

Implementation Details - Program structure



AdjacencyList

```
AdjacencyList()

void setup(int size)
int getSize(int i )
std::vector<int> get(int i)
std::vector<int>* getPtr(int i)

std::vector<int>* list
```

AdjacencyMatrix

```
AdjacencyMatrix()

void setup(int size)
int getSize(0
int index(int x, int y)
void set(int x, int v)
int get(int i )
int isConnected(int a, int b)

int size;
int* matrix
```

Node

```
Node(int node, int degree, int
compareNeighbourDegree(N
compareDegree(Node a, Noc

int node
int degree
int neighbourhood_degree
```

LSV2Object

```
LSV2Object()
LSV2Object(AdjacencyList* adj_list)
void setAdjList(AdjacencyList* adj_list);
void merge(LSV2Object* b);
bool insert(int node)

AdjencyList* adj_list;
int size;
std::unordered_set<int>* nodes;
std::unordered_map<int, int>* adj_map
```

LSV2ObjectWrapper

```
LSV2ObjectWrapper(AdjencyList* adj_l, int startingSize
// LSV2Object* get(int i);
voidadd(LSV2Object obj);
voidadd();
intgetSizeElement(int i);
intgetBest();
voidmerge(int a, int b);
voidmergeAll();
boolinsert(int index, int value);
boolattemptInsertAll(int value);
std::unordered_set<int>collapse(int index)

int size;
int best;
std::vector<LSV2Object>* objects

AdjencyList* adj_l;
```

Solver

```
Solver(int number_nodes, int number_edges, AdjacencyMatrix* adj_m, AdjacencyList* adj_l)

int solve(int target)
double getSearchTime()
bool isIS(std::unordered_set<int>* is)
std::chrono::high_resolution_clock::time_point startTime;
std::chrono::high_resolution_clock::time_point endTime;
intgetDegree(int i);
std::vector<int>*getNeighbours(int i);
intgetNeighbourhoodDegree(int i);
voidstartClock();
voidstopClock();

int number_nodes
int number_edges
AdjacencyMatrix* adj_m
AdjacencyList* adj_l
```

Reader

```
Reader()
void read(std::string path, bool complement, AdjancencyMatrix* adj_m, AdjancentList* adj_l, int* n_n, int* n_e
```

GreedySolver

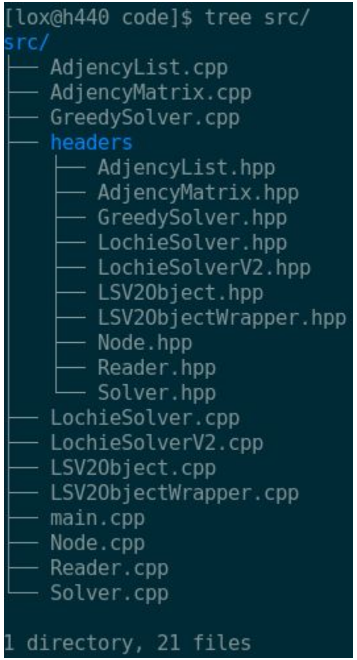
```
int solve(bool neighbourDegreeCompare, int target)
```

LochieSolver

```
int solve(int target, int maxIterations)
void updatePotential(int inserted, std::unordered_set<int>* potential
```

LochieSolverV2

```
int solve(int target, int maxIterations, int startingSize)
void prepareQueue(std::queue<int>* nodeQueue)
```



```
[lox@h440 code]$ tree src/
src/
├── AdjencyList.cpp
├── AdjencyMatrix.cpp
├── GreedySolver.cpp
├── headers
│   ├── AdjencyList.hpp
│   ├── AdjencyMatrix.hpp
│   ├── GreedySolver.hpp
│   ├── LochieSolver.hpp
│   ├── LochieSolverV2.hpp
│   ├── LSV2Object.hpp
│   ├── LSV2ObjectWrapper.hpp
│   ├── Node.hpp
│   ├── Reader.hpp
│   └── Solver.hpp
├── LochieSolver.cpp
├── LochieSolverV2.cpp
├── LSV2Object.cpp
├── LSV2ObjectWrapper.cpp
├── main.cpp
├── Node.cpp
├── Reader.cpp
└── Solver.cpp

1 directory, 21 files
```

Implementation Details - System

gcc version: 8.3.1 20190223 (Red Hat 8.3.1-2) (GCC)

Kernel: 5.0.16-200.fc29.x86_64g

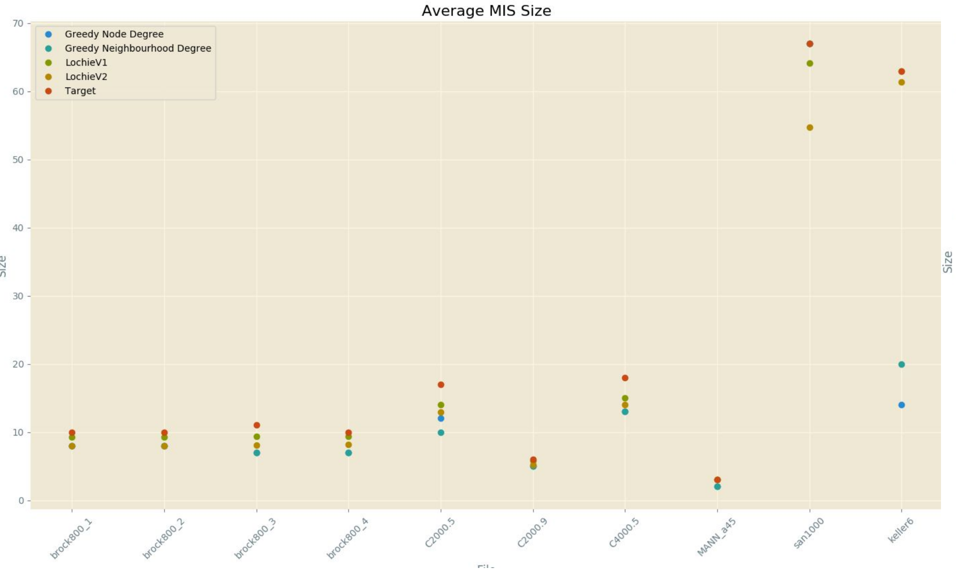
CPU: i7 6700k 4~4.2Ghz, 8MB cache

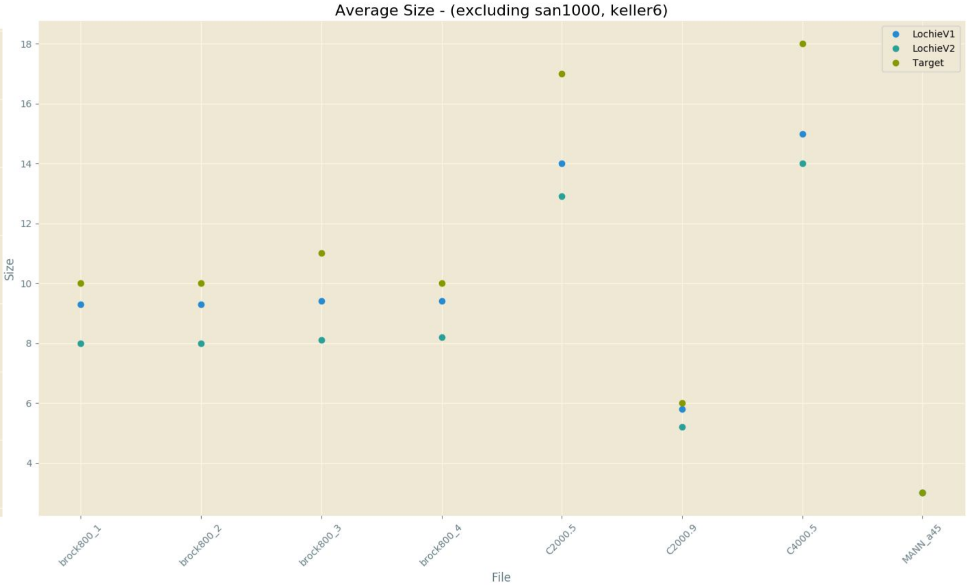
RAM: 16GB 2400Mhz

Compiled with optimizations

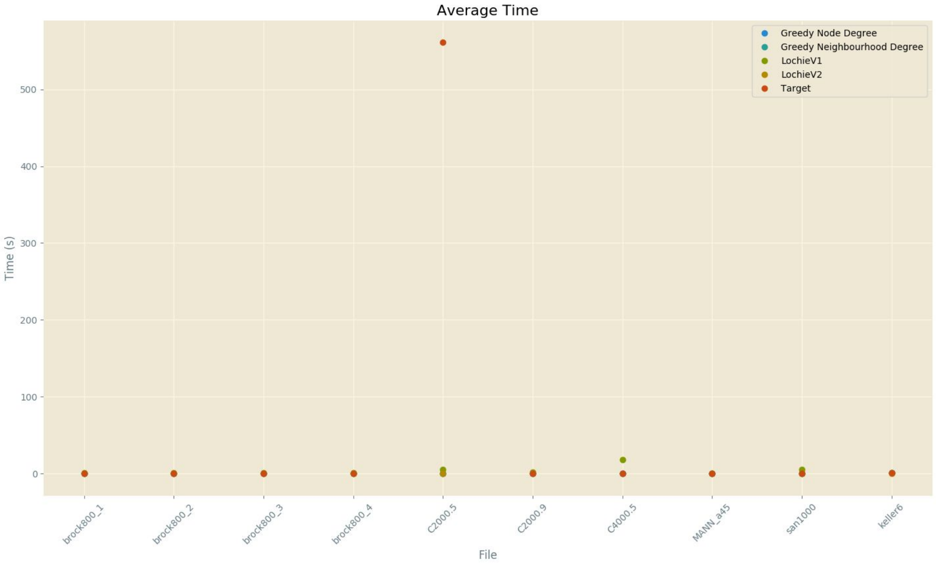
- g++ -02 ....
- Up to 6x performance was observed

Results

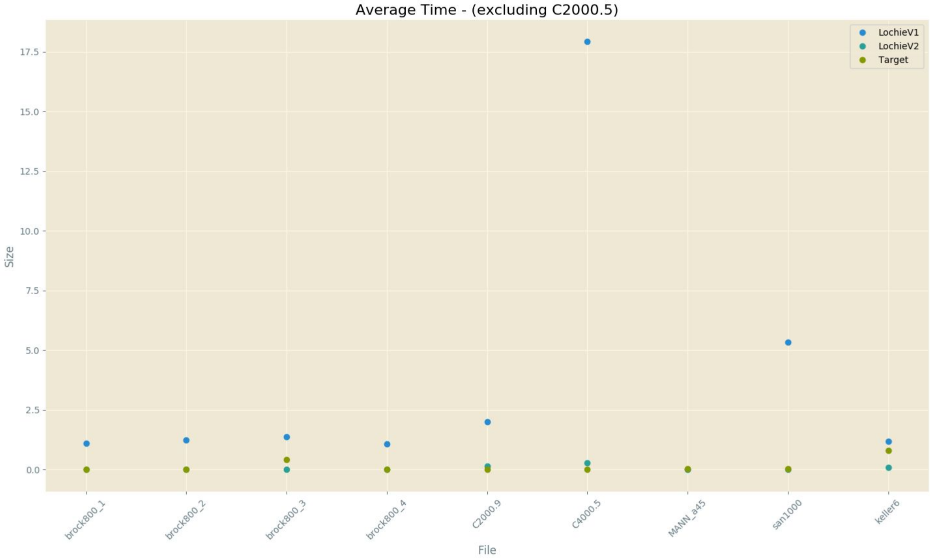

Average MIS Size
Greedy Node Degree
Greedy Neighbourhood Degree
LochieV1
LochieV2
Target
Size
0
10
20
30
40
50
60
70
brock800_1
brock800_2
brock800_3
brock800_4
C2000.5
C2000.9
C4000.5
MANN_a45
san1000
keller6
File


Average Size - (excluding san1000, keller6)
LochieV1
LochieV2
Target
Size
4
6
8
10
12
14
16
18
brock800_1
brock800_2
brock800_3
brock800_4
C2000.5
C2000.9
C4000.5
MANN_a45
File

Results

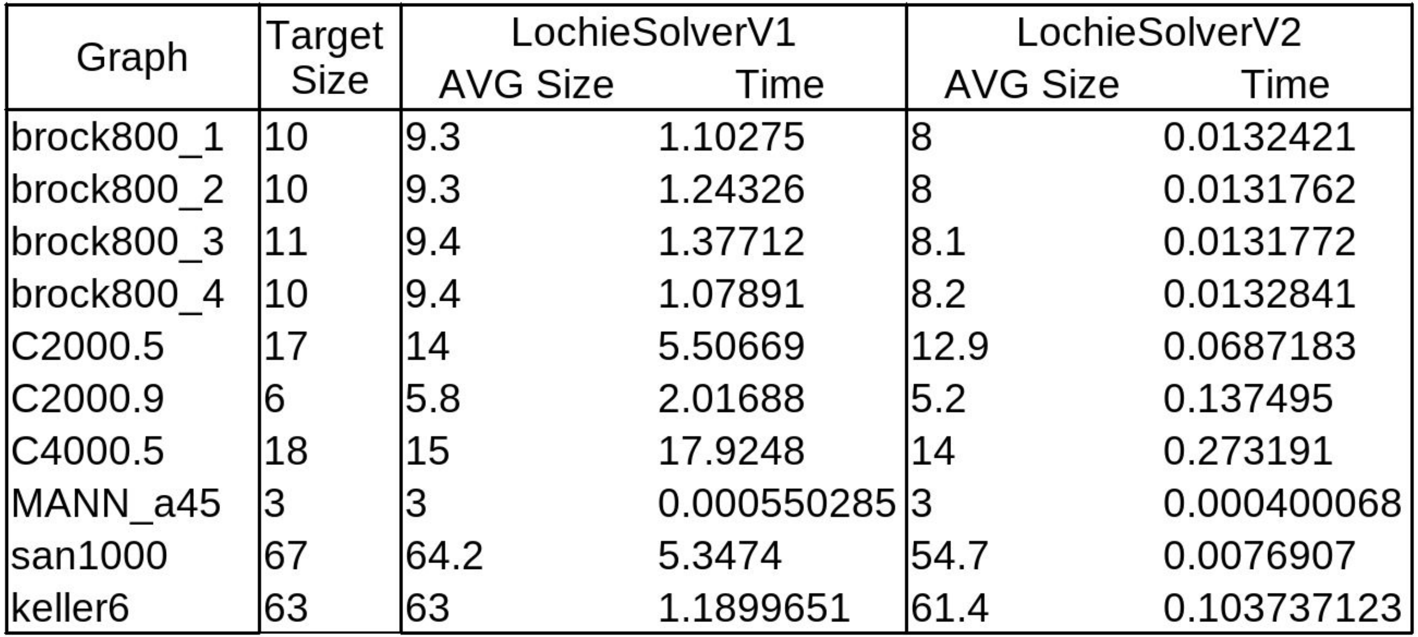


Average Time
Greedy Node Degree
Greedy Neighbourhood Degree
LochieV1
LochieV2
Target
Time (s)
0
100
200
300
400
500
brock800_1
brock800_2
brock800_3
brock800_4
C2000.5
C2000.9
C4000.5
MANN_a45
san1000
keller6
File



Average Time - (excluding C2000.5)
LochieV1
LochieV2
Target
Size
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
brock800_1
brock800_2
brock800_3
brock800_4
C2000.9
C4000.5
MANN_a45
san1000
keller6
File

Results



Graph	Target Size	LochieSolverV1		LochieSolverV2	
		AVG Size	Time	AVG Size	Time
brock800_1	10	9.3	1.10275	8	0.0132421
brock800_2	10	9.3	1.24326	8	0.0131762
brock800_3	11	9.4	1.37712	8.1	0.0131772
brock800_4	10	9.4	1.07891	8.2	0.0132841
C2000.5	17	14	5.50669	12.9	0.0687183
C2000.9	6	5.8	2.01688	5.2	0.137495
C4000.5	18	15	17.9248	14	0.273191
MANN_a45	3	3	0.000550285	3	0.000400068
san1000	67	64.2	5.3474	54.7	0.0076907
keller6	63	63	1.1899651	61.4	0.10373712

Improvements

- std::unordered_set for Adjacency lists
- Randomization in the insertion and merging in LochieSolverV2
- Reliable collapsing - change queue to circular buffer
- Statistical analysis on the times/mis size
- Compliments - to see change in density
- More testing
  - Larger sample size
  - Testing of parameters