Project 2: Graph Algorithms

Iskandar, Andrel, Yong Qiang, Wayne, Jin Han

Overview

Introduction
01

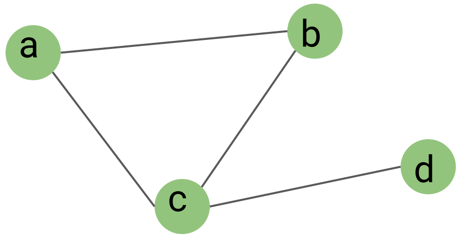
Analysis
02

Empirical
03

Data Processing
04

Conclusion
05

Graph Algorithms

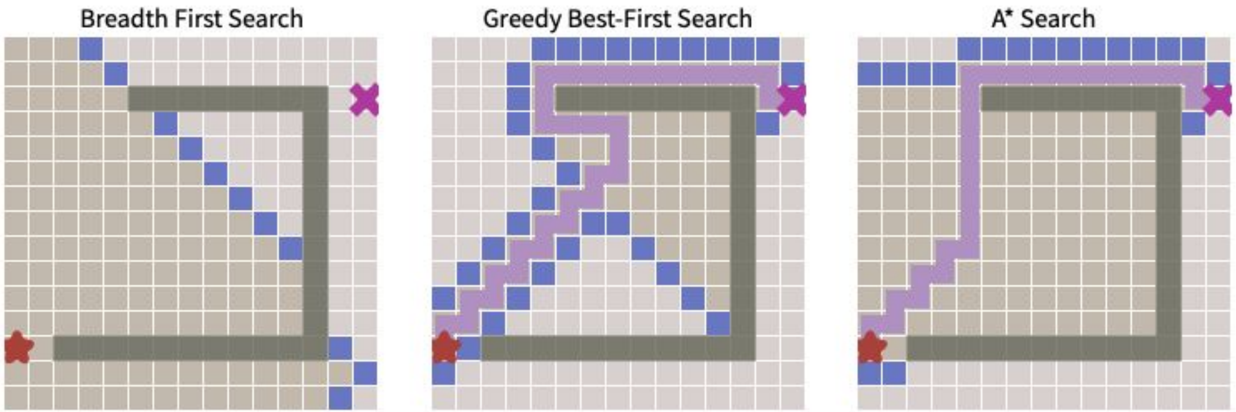


Undirected Graph

x	a	b	c	d
a	0	1	1	0
b	1	0	1	0
c	1	1	0	1
d	0	0	1	0

Adjacency matrix

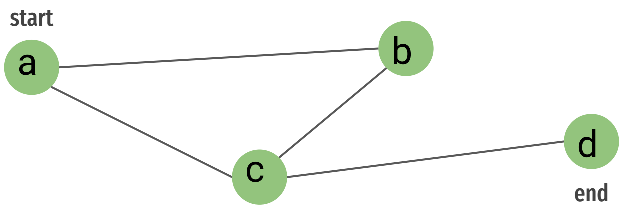
Used for data processing:
Conversion from edge list



A* Search

Pseudocode

- f(n) = g(n) cost + h(n) heuristic
- openList : Nodes
- closedList




```
openList = [ nodes ], closedList = []
g(start) = 0, h(start) = heuristic(start,end)
f(start) = g(start) + h(start)

loop openList when Not Null
        M = top node of openList
        If m == end
                End function
        Add M to closeList
        Remove M at openList

        Loop childnodes as n of M:
                If n exist in closeList:
                        Continue
                Cost = g(M) + distance(M,n)
                If n in closedList AND cost < g(n):
                        Remove n from openList for faster path
                If n in openList AND cost < g(n):
                        Remove n from closedList
                If n not in openList AND n not in closedList:
                        Add n to openList
                        g(n) = cost
                        h(n) = heuristic(n,end)
                        f(n) = g(n) + h(n)
```

Analysis of A* Search

- Combines BFS (efficiency) & Greedy (speed)
- Estimate shortest path to nearest hospital
- Time Complexity
  - Best Case: $O(E)$ --- target node beside start node
  - Worse Case: $O(|V|+|E|)$ --- no path found / node cannot be reached
  - Average Case: $O(1/e + (V+E)/V)$

Empirical Study

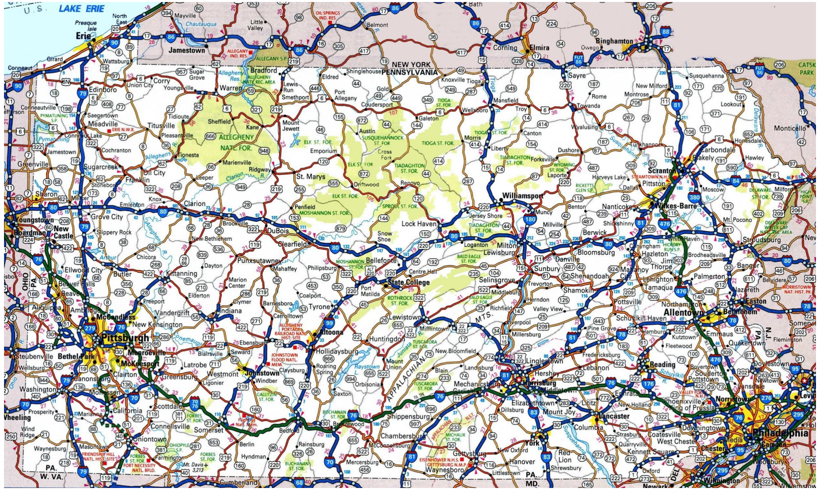
- Dire scenarios and disasters taken into consideration
- Performance and Speed of algorithms
- Theoretic Analysis and Evaluation:
    - Breadth First Search
    - Depth First Search
    - Greedy Search

Analysis Overview of Algorithms

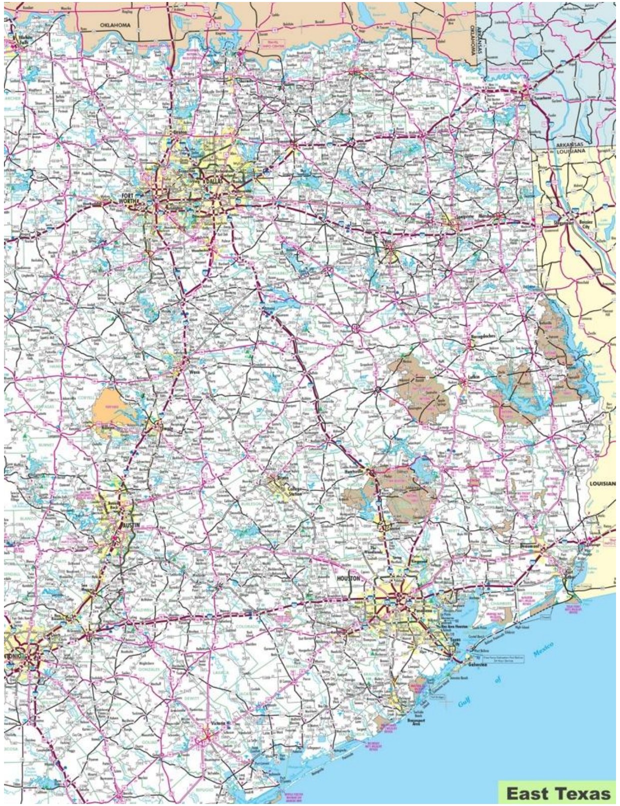
Algorithm	Random Graph Generated	Road Network Graph in Pennsylvania	Road Network Graph in Texas
👍 A* Search	Fast, Optimal	Fast, Optimal	Fast, Optimal
Breadth First Search	Slow, Optimal	Slow, Optimal	Slow, Optimal
Last First Search	Average speed, Not-optimal	Average speed, Not-optimal	Average speed, Not-optimal
Greedy Search	Fast, Less-Optimal	Fast, Less-Optimal	Fast, Less-Optimal

Real road network

- Huge data
- Process in to graph / adjacency matrix



Pennsylvania Road Map
- 1,088,092 Nodes + 1,541,898 Edges



Texas Road Map
- 1,379,917 Nodes + 1,921,660 Edges

Which to use?

CPU (instruction)	GPU (data processing)
● Serial processing ● Random branching instruction stream	● Parallel processing ● Data stream ● Vast amount of data

GPU Processing of road map

- Data stream - eliminate nested loops
- Convert codes to Cuda for GPU to read and run
- Dual GPU for faster processing

GPU Data Processing

Pseudocode

```
Roadmap = ReadRoadMap("roadmap.txt")

CudaArray = Cuda.Transfer(Roadmap)

Use GPU 0:
        CudaArray.SortAscending
        totalNodes = CudaArray.CountUniqueNodes

Use GPU 1:
        startNode, endNode = CudaArray.HorizontalSplit

Use GPU 0,1: (Matrix)
        currentRow = 0
        matrixRow = Cuda.ZeroValRows(totalNodes)
        Loop totalNodes:
                if(current != startNode):
                        AppendTextFile(matrixRow)
                        Current = startNode
                        matrixRow = Cuda.ZeroValRows(totalNodes)
                matrixRow[endNode] = 1
        Else:
                AppendTextFile(matrixRow)

Use GPU 0,1: Adjacency List
        adjList = [], currentNode = 0
        Loop i in totalNodes:
                if(currentNode == U):
                        U = startNode[i]
                        V = endNode[i]
                Else:
                        AppendTextFile(adjList)
                        adjList = []
                adjList[U].append(V)
```

Data Process

CudaProcessing.py   adjList.txt

Project 2 > Graph > adjList.txt

```
[1, 6309, 6353]
[0]
[3, 4, 7]
[2, 309]
[2, 273, 274, 388]
[6, 8, 9]
[5, 305, 309, 310, 1060307]
[2, 8, 16, 3998]
[5, 7]
[5, 12, 10464]
[11, 22, 77]
[10, 44]
[9, 13, 14]
[12]
[12, 15, 20]
[14, 310, 312]
```

```python
def ConvertToAdjList(arr1):
    #Device 1 is Asus Strix RTX 3090
    #cp.cuda.Device(1).use()
    cpSortedArr = arr1[arr1[:,0].argsort()]
    start,end = cp.hsplit(cpSortedArr, 2)
    node = cp.unique(cpSortedArr).size

    #Device 0 is MSI Gaming X Trio RTX 3090
    #cp.cuda.Device(0).use()
    adjList = [[] for k in range(node)]

    #Line 52 ~ 58 are executed in parallel
    #Line 78 wait for both of my GPU to finish calculating for continuing to the next line
    #cp.cuda.Device({0,1}).synchronize()

    #Using both GPU to process at the same time
    #cp.cuda.Device({0,1}).use()
    for i in range(node):
        u = int(start[i].get())
        v = int(end[i].get())
        adjList[u].append(v)

    np.savetxt("adjList.txt", adjList, delimiter=" ", fmt="%s")
```

```
λ python CudaProcessing.py
Edge List to Adj List Completed 14.256047500000001
```

Data Process

CudaProcessing.py | matrixList.txt ×

Project 2 > Graph > matrixList.txt

```
[0 1 0 ... 0 0 0]
[1 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 1 ... 0 0 0]
[0 0 1 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 1 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
```

```
λ python CudaProcessing.py
Edge List to Matrix List Completed 16.4808332
```

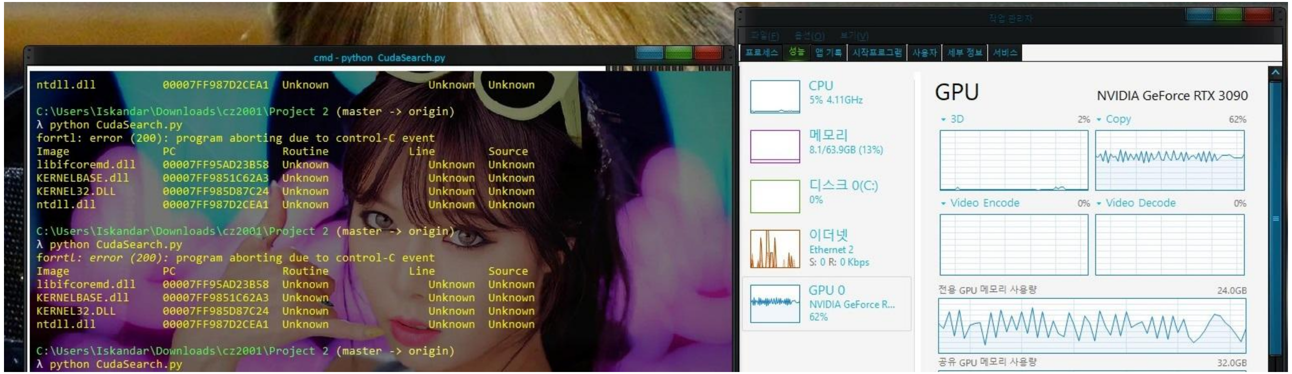
```python
def CompressedAdjMatrix(arr1):
    #Device 1 is Asus Strix RTX 3090
    #cp.cuda.Device(1).use()
    size = cp.unique(arr1).size
    cpSortedArr = arr1[arr1[:,0].argsort()]

    #Device 0 is MSI Gaming X Trio RTX 3090
    #cp.cuda.Device(0).use()
    #Initialize a [0,0,0,...,0] with a size of n in parallel
    arr = cp.zeros(size, dtype="int32")
    cur = 0

    #Don't uncomment this line, I'm using 2 GPU(s) on SLI to process
    #cp.cuda.Device({0,1}).use()
    with open("matrixList.txt", 'w') as f:
        for row,col in cpSortedArr:
            if(cur != row):
                f.write(cp.array_str(arr) + "\n")
                cur = row
                arr = cp.zeros(size, dtype="int32")
            arr[col.get()] = 1
        else:
            f.write(cp.array_str(arr) + "\n")
            f.close()
```

Analysis of GPU Processing

- Utilize CUDA Cores
- Parallel Processing
- Converts Edge List to Adjacency Matrix and List
- Time Complexity
  - C = Cuda Cores
  - Adjacency Matrix: $O([V^2/C] + \log C)$
  - Adjacency List: $O([V+E/C] + \log C)$



Conclusion

- A* Search (fast & optimal)
- Analysis of GPU and CPU processing
- GPU vs CPU for huge data processing