

Technical Report: Final Project
EECE 2560: Fundamentals of Engineering
Algorithms
Smart City Resource Allocation

Matthew Garcia and Rostyslav Rozhok

Department of Electrical and Computer Engineering Northeastern University

garcia.matt@northeastern.edu

rozhok.r@northeastern.edu

December 1, 2024

Contents

1 Project Scope.....	2
2 Project Plan.....	2
2.1 Timeline.....	2
2.2 Milestones.....	3
3 Team Discussion Summary.....	3
3.1 Team Roles.....	3
3.2 Tools and Technologies.....	3
4 Methodology.....	3
4.1 Pseudocode and Main Functions.....	3
4.2 Time Complexity Analysis.....	5
4.3 DataCollection and Preprocessing.....	5
5 Results.....	5
6 Discussion.....	5
7 Conclusion.....	6
8 References.....	6
Appendix A.....	7
Appendix B.....	14

1 Project Scope

The aim of this project is to design and develop a smart city resource allocation system, making use of

greedy algorithms for optimization and graph algorithms to represent resource flow, in order to allocate resources to various sectors of a smart city.

The project's main objectives are to:

- Efficiently optimize resource allocation so usage is maximized and waste is minimized.
- Account for the dynamics presented within a city, such as a growing population and demand.
- Completely automate the process of allocating resources.

There is no exact definition of Smart City. Our project focuses on allocation of resources to the sectors that need them the most.

The key issues of any city:

- Energy management: optimization of use, conversion, transportation, and losses. One of the future practices will be the management of electric vehicles' batteries to take in the energy when there is surplus of energy and give off when there is shortage.
- Trash management. One of the solutions is diligent sorting of trash with further burning to get the energy, and reuse of remnants.
- Water purification. Chemicals in detergents, cleaning, fertilizers, pesticides, byproducts of production and heavy metals cause many illnesses. The policies and proper filters are of high priority for a smart city.
- Real estate development. New facilities and apartment complexes require infrastructure of energy, water ways, and waste connections and facilities which are usually not present in abundance.

While our project will focus on the quantifiable part of this, it is important to understand the issues behind the organization of a smart city.

2 Project Plan

2.1 Timeline

The overall timeline for this project will be divided into phases:

- Week 1 (October 7 - October 13): Define the scope of the project and how it will be implemented, allocate tasks to team members, create GitHub repository.
- Week 2 (October 14 - October 20): Do research on smart cities, create system design, start coding basic system functionalities.
- Week 3 (October 21 - October 27): Start technical report from research done prior week, start coding system design.
- Week 4 (October 28 - November 3): Work on algorithmic/system code, start coding frontend, continue the report.
- Week 5 (November 4 - November 10): Start user interface and integrate with system code, test code, continue the report.
- Week 6 and 7 (November 11 - November 17): Begin presentation, finalize testing and user interface, report, and presentation.
- Week 8 (November 18 - Dec 4): Present project, submit report and code.

2.2 Milestones

Key milestones include:

- Project Scope and Plan (October 10).
- GitHub Repository Setup (October 10).
- System Code Completion (November 10).
- User Interface Completion (November 15).
- Final System Testing and Report Draft (November 17).
- Final Presentation Draft and Report Draft Submission (November 28).
- Final Presentation and Report Submission (November 28).

3 Team Discussion Summary

3.1 Team Roles

- Matthew Garcia: System Development, GitHub Repository Management, Technical Documentation (Partially), System Testing.
- Rostyslav Rozhok: User Interface Development, System Development (Partially), Technical Documentation, System Testing.

3.2 Tools and Technologies

Team members need the following skills to complete the project:

- System Code: C++.
- User Interface: terminal console.
- Version Control: GitHub.
- Technical Report: Overleaf.

4 Methodology

4.1 Pseudocode and Main Functions

Allocator

1. **ResourceAllocator(const string& file)**
Initialisation of the allocator and call of loadResources to parse input.
2. **void loadResources(const string& file)**
Parsing of input files and populating resource-specific vectors with data.
3. **void allocateResources()**
Calling allocation methods for electricity, water, gas, waste.
4. **void allocateElectricity()**
Sorting of electricity resources and adjusting availability by peak demand.
5. **void allocateWater() and void allocateGas()**
Reduction of water and gas availability by a fixed amount for each district
6. **void allocateWasteManagement()**

Deduction of waste management availability with adjustments in the districts.

Flow graph

1. **ResourceFlowGraph(int n)**
Constructs a flow graph with specified node count and adjacency list.
2. **void addEdge**
Addition of edge and reverse edge to support capacity and flow modeling.

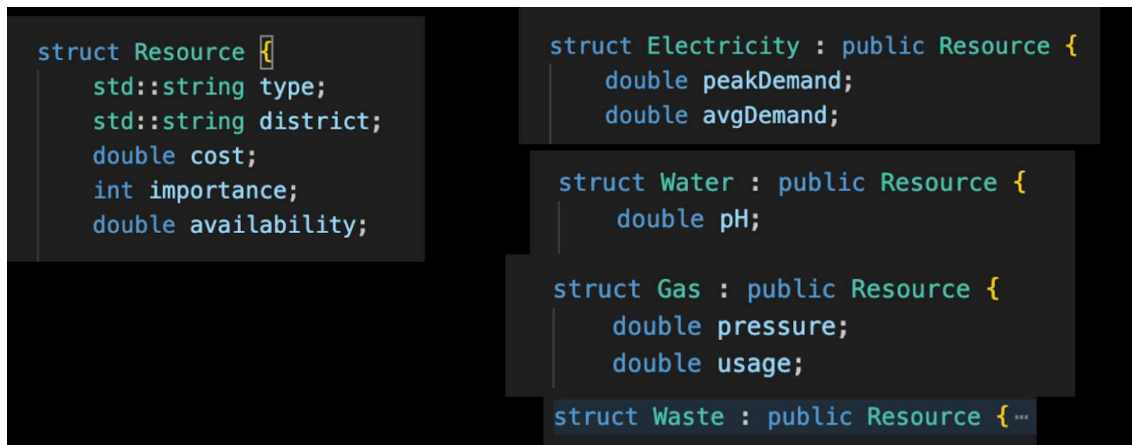
Sorting Algorithm

```
for (i=0 to resourceList.size - 1)
    for (j=0 to resourceList.size - i - 1)

        resourceA = resourceList[j]
        resourceB = resourceList[j + 1]
        // Importance comparison (higher-better)
        if resourceA.importance < resourceB.importance
            swap(resourceList[j], resourceList[j + 1])

        else if
            //importance the same? Compare cost (lower-better)
            resourceA.importance == resourceB.importance AND resourceA.cost >
            resourceB.cost
                swap(resourceList[j], resourceList[j + 1])
```

Above is the idea of the sorting algorithm to sort resources based on their importance and cost. The idea behind is that we don't want to draw a resource from a far located district, because it is expensive. We also want to optimize for resources of first necessity.



```
struct Resource {
    std::string type;
    std::string district;
    double cost;
    int importance;
    double availability;
};

struct Electricity : public Resource {
    double peakDemand;
    double avgDemand;
};

struct Water : public Resource {
    double pH;
};

struct Gas : public Resource {
    double pressure;
    double usage;
};

struct Waste : public Resource { ...
```

Picture 1: Base class and derived classes in ResourceAllocator.h

```

class ResourceAllocator {
private:
    map<string, vector<Resource*>> resources;

public:
    explicit ResourceAllocator(const string& file);
    ~ResourceAllocator();

    void addNewResourceType(Resource* resource);
    void allocateResources();
    void displayAllocation();

private:
    void loadResources(const string& file);
    void allocateResourceType(const string& type);
};

```

Picture 2: ResourceAllocator class and its methods

4.2 Time Complexity Analysis

We decided not to use Bubble Sort and used the built-in IntroSort algorithm, which combines quicksort, heap sort and insertion sort we learned in class. The time complexity of our project, dictated by a sorting algorithm, is $O(n \log n)$, where n is a resource object. The resource loading and parsing, resource allocation and display have $O(n)$ time complexity.

The Graph time complexity of our unfinished code is $O(n+e)$, where n is the number of nodes and e the number of edges.

More characteristics (e.g peak and average demand of electricity) can be added to each resource without change of time complexity, unless more complex algorithms have to be introduced for new characteristics.

4.3 DataCollection and Preprocessing

Input: A csv file is used as a data input. A graph configuration should be provided.

Output: Resource allocation statistics as console output, and a file for graph representation of resource flow.

A user is also able to use the console to add resources without the need to change the csv file.

5 Results

Resources were allocated successfully and users can enter in new districts and have resources be reallocated respectively. The outputs can be seen in Appendix B

6 Discussion

Further development may include properly functioning graph representation. We spent time trying to develop graph algorithms, but didn't succeed with the logic. Since there was no graphical representation it didn't make much sense to make a GUI, as the code works primarily with text data and a csv file is a very convenient way to store the data of the format we used. We expanded the console interaction menu, which we initially didn't plan to have, to allow easier use, small changes and more friendly functionality testing.

Another possible development is expansion of resources' characteristics, which is both manageable and important.

7 Conclusion

Designing and building Smart Cities is one of the most crucial objectives of humanity in the 21st century. Developments in this area can greatly enhance the living conditions, efficiency and security. Well designed interconnected systems can also allow for much work to be distributed to robots and automated machines.

Obtaining data and having knowledge of various needs of inhabitants and infrastructure upkeep necessities lead to more optimal design of control systems.

8 References

M. Zaman, M. Al Islam, A. Tantawy, C. J. Fung and S. Abdelwahed, "Adaptive Control for Smart Water Distribution Systems," *2021 IEEE International Smart Cities Conference (ISC2)*, Manchester, United Kingdom, 2021, pp. 1-6, doi: 10.1109/ISC253183.2021.9562812.

R. Chen, R. Wang, W. Zhao, H. Sun, X. Wu and J. Xie, "Cooperative benefit allocation for domestic electricity of smart communities based on coalitional game theory," *2020 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*, Chongqing, China, 2020, pp. 185-189, doi: 10.1109/ICICAS51530.2020.00045.

Bash, Cullen, et al. "IT for Sustainable Smart Cities: A Framework for Resource Management and a Call for Action." *The Bridge*, National Academy of Engineering, 17 Mar. 2023, <https://www.nae.edu/290991/IT-for-Sustainable-Smart-Cities-A-Framework-for-Resource-Management-and-a-Call-for-Action>.

Appendix A

main.cpp

```
#include <iostream>
#include "ResourceAllocator.h"
#include "ResourceFlowGraph.h"
using namespace std;

void displayMenu() {
    cout << "\n=== Resource Allocation Menu ===\n";
    cout << "1. Display data for a specific district\n";
    cout << "2. Add a new district\n";
    cout << "3. Display the resource flow graph\n";
    cout << "4. Exit\n";
    cout << "Enter your choice: ";
}

void handleDisplayDistrict(ResourceAllocator& allocator) {
    string district;
    cout << "Enter the district name: ";
    cin >> district;
    cout << "\nResources in " << district << ":\n";
    allocator.displayAllocation(); // Display all and filter district manually if needed
}

void handleAddDistrict(ResourceAllocator& allocator) {
    string type, district;
    double cost, availability, additional1 = 0, additional2 = 0;
    int importance;
    cout << "Enter resource type (Electricity, Water, Gas, Waste): ";
    cin >> type;
    cout << "Enter district name: ";
    cin >> district;
    cout << "Enter cost: ";
    cin >> cost;
    cout << "Enter importance: ";
    cin >> importance;
    cout << "Enter availability: ";
    cin >> availability;
    if (type == "Electricity" || type == "Gas") {
        cout << "Enter peak demand/pressure: ";
        cin >> additional1;
        cout << "Enter average demand/usage: ";
        cin >> additional2;
    } else if (type == "Water") {
        cout << "Enter pH level: ";
        cin >> additional1;
    }
    Resource* resource = nullptr;
    if (type == "Electricity") {
        resource = new Electricity(type, district, cost, importance, availability, additional1, additional2);
    } else if (type == "Water") {
        resource = new Water(type, district, cost, importance, availability, additional1);
    } else if (type == "Gas") {
        resource = new Gas(type, district, cost, importance, availability, additional1, additional2);
    } else if (type == "Waste") {
```

```

        resource = new Waste(type, district, cost, importance, availability);
    }
    if (resource) {
        allocator.addNewResourceType(resource);
        cout << "Resource added successfully.\n";
    } else {
        cout << "Invalid resource type.\n";
    }
}
void handleDisplayGraph(ResourceFlowGraph& graph) {
    graph.displayGraph();
}
int main() {
    ResourceAllocator allocator("C:/Users/gmatt/CLionProjects/AlgosProject/resources_data.csv");
    ResourceFlowGraph graph(5);
    int choice;
    do {
        displayMenu();
        cin >> choice;
        switch (choice) {
            case 1:
                handleDisplayDistrict(allocator);
                break;
            case 2:
                handleAddDistrict(allocator);
                break;
            case 3:
                handleDisplayGraph(graph);
                break;
            case 4:
                cout << "Exiting the program.\n";
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 4);
    return 0;
}

```

ResourceAllocator.cpp

```

#include "ResourceAllocator.h"
#include <fstream>
#include <sstream>
ResourceAllocator::ResourceAllocator(const string& file) {
    loadResources(file);
}
ResourceAllocator::~ResourceAllocator() {
    for (auto& [type, resourceList] : resources) {
        for (auto* res : resourceList) {
            delete res;
        }
    }
}

```



```

}
void ResourceAllocator::addNewResourceType(Resource* resource) {
    resources[resource->type].push_back(resource); // Corrected from push_back on map to accessing
vector by key
}
void ResourceAllocator::loadResources(const std::string& filePath) {
    std::ifstream file(filePath);
    if (!file.is_open()) {
        throw std::runtime_error("Could not open file: " + filePath);
    }
    std::string line;
    while (std::getline(file, line)) {
        std::istringstream ss(line);
        std::string type, district;
        double cost, availability, additional1, additional2;
        int importance;
        ss >> type >> district >> cost >> importance >> availability >> additional1 >> additional2;
        Resource* resource = nullptr;
        if (type == "Electricity") {
            resource = new Electricity(type, district, cost, importance, availability, additional1, additional2);
        } else if (type == "Water") {
            resource = new Water(type, district, cost, importance, availability, additional1);
        } else if (type == "Gas") {
            resource = new Gas(type, district, cost, importance, availability, additional1, additional2);
        } else if (type == "Waste") {
            resource = new Waste(type, district, cost, importance, availability);
        }
        if (resource) {
            resources[resource->type].push_back(resource); // Correctly using push_back for the vector
inside the map
        }
    }
    file.close();
}
void ResourceAllocator::allocateResources() {
    for (auto& [type, resourceList] : resources) {
        allocateResourceType(type);
    }
}
void ResourceAllocator::allocateResourceType(const string& type) {
    auto& resourceList = resources[type];
    sort(resourceList.begin(), resourceList.end(),
        [](Resource* a, Resource* b) {
            return a->importance > b->importance ||
                (a->importance == b->importance && a->cost < b->cost);
        });
    double totalAvailability = 0;
    for (const auto* resource : resourceList) {
        totalAvailability += resource->availability;
    }
    for (auto* resource : resourceList) {
        double allocation = std::min(resource->availability, totalAvailability);
        resource->availability -= allocation;
        totalAvailability -= allocation;
    }
}

```

```

    }
}
void ResourceAllocator::displayAllocation() {
    for (const auto& [type, resourceList] : resources) {
        cout << "\n" << type << " Allocation:\n";
        for (const auto* res : resourceList) {
            res->display();
        }
    }
}
}

```

ResourceAllocator.h

```

#ifndef RESOURCE_ALLOCATOR_H
#define RESOURCE_ALLOCATOR_H
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <algorithm>
using namespace std;
struct Resource {
    std::string type;
    std::string district;
    double cost;
    int importance;
    double availability;
    Resource(const std::string& t, const std::string& d, double c, int i, double a)
        : type(t), district(d), cost(c), importance(i), availability(a) {}
    // Display method for Resource
    virtual void display() const {
        std::cout << "Resource Type: " << type
            << ", District: " << district
            << ", Cost: " << cost
            << ", Importance: " << importance
            << ", Availability: " << availability
            << std::endl;
    }
};
struct Electricity : public Resource {
    double peakDemand;
    double avgDemand;
    Electricity(const std::string& t, const std::string& d, double c, int i, double a, double p, double avg)
        : Resource(t, d, c, i, a), peakDemand(p), avgDemand(avg) {}
    // Override display method for Electricity
    void display() const override {
        std::cout << "Electricity: Type: " << type
            << ", District: " << district
            << ", Cost: " << cost
            << ", Importance: " << importance
            << ", Availability: " << availability
            << ", Peak Demand: " << peakDemand
            << ", Average Demand: " << avgDemand
            << std::endl;
    }
};

```

```

    }
};
struct Water : public Resource {
    double pH;
    Water(const std::string& t, const std::string& d, double c, int i, double a, double p)
        : Resource(t, d, c, i, a), pH(p) {}
    // Override display method for Water
    void display() const override {
        std::cout << "Water: Type: " << type
            << ", District: " << district
            << ", Cost: " << cost
            << ", Importance: " << importance
            << ", Availability: " << availability
            << ", pH: " << pH
            << std::endl;
    }
};
struct Gas : public Resource {
    double pressure;
    double usage;
    Gas(const std::string& t, const std::string& d, double c, int i, double a, double p, double u)
        : Resource(t, d, c, i, a), pressure(p), usage(u) {}
    // Override display method for Gas
    void display() const override {
        std::cout << "Gas: Type: " << type
            << ", District: " << district
            << ", Cost: " << cost
            << ", Importance: " << importance
            << ", Availability: " << availability
            << ", Pressure: " << pressure
            << ", Usage: " << usage
            << std::endl;
    }
};
struct Waste : public Resource {
    Waste(const std::string& t, const std::string& d, double c, int i, double a)
        : Resource(t, d, c, i, a) {}
    // Override display method for Waste
    void display() const override {
        std::cout << "Waste: Type: " << type
            << ", District: " << district
            << ", Cost: " << cost
            << ", Importance: " << importance
            << ", Availability: " << availability
            << std::endl;
    }
};
class ResourceAllocator {
private:
    map<string, vector<Resource*>> resources;
public:
    explicit ResourceAllocator(const string& file);
    ~ResourceAllocator();
    void addNewResourceType(Resource* resource);

```

```

    void allocateResources();
    void displayAllocation();
private:
    void loadResources(const string& file);
    void allocateResourceType(const string& type);
};
#endif // RESOURCE_ALLOCATOR_H

```

ResourceFlowGraph.cpp

```

#include "ResourceFlowGraph.h"
ResourceFlowGraph::ResourceFlowGraph(int n) : adjList(n) {}
void ResourceFlowGraph::addEdge(int from, int to, double capacity) {
    adjList[from].push_back({to, capacity});
    adjList[to].push_back({from, 0}); // Reverse edge
}
void ResourceFlowGraph::displayGraph() const {
    cout << "Resource Flow Graph:\n";
    for (size_t i = 0; i < adjList.size(); ++i) {
        cout << "Node " << i + 1 << ":\n";
        for (const auto& edge : adjList[i]) {
            cout << " -> Node " << edge.to + 1
                << " receives " << edge.capacity << " units\n";
        }
    }
}

```

ResourceFlowGraph.h

```

#ifndef RESOURCE_FLOW_GRAPH_H
#define RESOURCE_FLOW_GRAPH_H
#include <vector>
#include <iostream>
using namespace std;
struct Edge {
    int to;
    double capacity;
};
class ResourceFlowGraph {
private:
    vector<vector<Edge>> adjList;
public:
    explicit ResourceFlowGraph(int n);
    void addEdge(int from, int to, double capacity);
    void displayGraph() const;
};
#endif // RESOURCE_FLOW_GRAPH_H

```

resources_data.csv

Type,District,Cost,Importance,Availability,PeakDemand_MW,AverageDemand_MW,WaterQuality(pH),
 GasPressure_kPa,GasUsage_MillionCubicMeters,WaterUsage_MillionLiters
 Electricity,1,,0.4,20,30,20,,,,,

Electricity,2,,0.3,70,50,35,,,,
 Electricity,3,,0.5,200,400,220,,,,
 Electricity,4,,0.3,300,210,130,,,,
 Electricity,5,,0.6,190,350,200,,,,
 Water,1,,0.5,30,,,7.4,,,10
 Water,2,,0.5,40,,,8,,,80
 Water,3,,0.4,500,,,7.8,,,56
 Water,4,,0.3,200,,,6.8,,,89
 Water,5,,0.5,15,,,6.56,,,210
 Gas,1,,0.3,100000,,,4,60000,
 Gas,2,,0.6,300000,,,3,200000,
 Gas,3,,0.5,100000,,,2,300000,
 Gas,4,,0.5,200000,,,5,450000,
 Gas,5,,0.4,1500000,,,11,1100000,
 Waste management,1,,0.3,,,,,,,,
 Waste management,2,,0.5,,,,,,,,
 Waste management,3,,0.3,,,,,,,,
 Waste management,4,,0.6,,,,,,,,
 Waste management,5,,0.5,,,,,,,,

Appendix B

Csv file

resources_data.csv

Type	District	Cost	Importance	Availability	PeakDemand_MW	AverageDemand_MW	WaterQuality(pH)	GasPressure_kPa	GasUsage_MillionCubicMeters	WaterUsage_MillionLiters
Electricity	1		0.4	20	30	20				
Electricity	2		0.3	70	50	35				
Electricity	3		0.5	200	400	220				
Electricity	4		0.3	300	210	130				
Electricity	5		0.6	190	350	200				
Water	1		0.5	30			7.4			10
Water	2		0.5	40			8			80
Water	3		0.4	500			7.8			56
Water	4		0.3	200			6.8			89
Water	5		0.5	15			6.56			210
Gas	1		0.3	100000				4	60000	
Gas	2		0.6	300000				3	200000	
Gas	3		0.5	100000				2	300000	
Gas	4		0.5	200000				5	450000	
Gas	5		0.4	1500000				11	1100000	
Waste management	1		0.3							
Waste management	2		0.5							
Waste management	3		0.3							
Waste management	4		0.6							
Waste management	5		0.5							

```
Type,District,Cost,Importance,Availability,Peak
Demand_MW,AverageDemand_MW,WaterQuality(pH),Gas
Pressure_kPa,GasUsage_MiliionCubicMeters,WaterU
sage_MillionLiters
Electricity,1,,0.4,20,30,20,,,,
Electricity,2,,0.3,70,50,35,,,,
Electricity,3,,0.5,200,400,220,,,,
Electricity,4,,0.3,300,210,130,,,,
Electricity,5,,0.6,190,350,200,,,,
Water,1,,0.5,30,,,7.4,,,10
Water,2,,0.5,40,,,8,,,80
Water,3,,0.4,500,,,7.8,,,56
Water,4,,0.3,200,,,6.8,,,89
Water,5,,0.5,15,,,6.56,,,210
Gas,1,,0.3,100000,,,4,60000,
Gas,2,,0.6,300000,,,3,200000,
Gas,3,,0.5,100000,,,2,300000,
Gas,4,,0.5,200000,,,5,450000,
Gas,5,,0.4,1500000,,,11,1100000,
Waste management,1,,0.3,,,,,
Waste management,2,,0.5,,,,,
Waste management,3,,0.3,,,,,
Waste management,4,,0.6,,,,,
Waste management,5,,0.5,,,,,
```

Output:

```

Allocating Resources
Allocations in districtAllocations:
District: 1, Resource: Electricity, Amount: 0
District: 1, Resource: Gas, Amount: 0
District: 1, Resource: Waste, Amount: 1
District: 1, Resource: Water, Amount: 1
District: 2, Resource: Electricity, Amount: 0
District: 2, Resource: Gas, Amount: 1
District: 2, Resource: Waste, Amount: 1
District: 2, Resource: Water, Amount: 1
District: 3, Resource: Electricity, Amount: 0
District: 3, Resource: Gas, Amount: 1
District: 3, Resource: Waste, Amount: 1
District: 3, Resource: Water, Amount: 1
District: 4, Resource: Electricity, Amount: 0
District: 4, Resource: Gas, Amount: 1
District: 4, Resource: Waste, Amount: 1
District: 4, Resource: Water, Amount: 1
District: 5, Resource: Electricity, Amount: 0
District: 5, Resource: Gas, Amount: 1
District: 5, Resource: Waste, Amount: 1
District: 5, Resource: Water, Amount: 1

```

