UCLouvain

LOUVAIN
School of Management

Louvain School of Management

A comparison of randomized optimal flow algorithms versus standard optimal flow algorithms

Auteur·e(s) : William Visée
Promoteur·rice(s) : Marco Saerens, Sylvain Courtain
Année académique 2020-2021
Travail de fin d'études (TFE) en vue d'obtenir le titre de
Master (60) en Sciences de Gestion
Horaire de jour / Horaire décalé

Contents

Abstract

The societies of today are ruled to produce optimized procedures to be competitive. There is always a moment of need to optimize flow problems inside or outside the company. It can be of several sorts like the transportation of goods from factories to shops or the scheduling of workers to tasks. A panel of algorithms has been developed to solve these problems and always gives the optimal solution. Then, new algorithms based on the bag-of-paths framework have been developed with a new solution property. Instead of having an optimal result, the output is more or less randomized depending on the degree of randomness that is wanted. It is interesting in many sectors like ecology to predict geographic animal behavior. This master thesis consists of the comparison of standard optimal flow algorithms and new algorithms developed in the bag-of-paths framework.

Acknowledgement

I would like to thanks my master thesis supervisors Professor Marco Saerens and assistant Sylvain Courtain for their guidance, encouragement, and availability through the development of this master thesis. Flow problems were a new subject for me and I enjoyed learning those new concepts and making this work.

1 Introduction

The companies of today are competitive and need to be efficient in all their actions to stay alive. A part of the problems they face are called flow problems [1, 16]. They can be of two types: maximum flow or minimum cost flow problems. The first one corresponds to the maximum flow possible inside a network with a maximum capacity of flow going through each edge. For example, it can tell the maximum truck, filled with goods, that can go from a factory to a client in a day. The second one corresponds to the cost-less flow inside a network considering that each unit of flow through an edge has a cost. For example, it can tell the best path, based on the edge costs, to take to send several trucks from multiple factories to multiple clients.

A lot of algorithms to solve flow problems have been created over the years. Each of them exploits different mathematical properties and is more or less efficient depending on the shape of the network. However, the output of these algorithms is always the same. [1]

New algorithms, based on the bag-of-paths framework [10, 20], have been recently developed with a new and interesting solution property. Instead of having an optimal solution, the output is more or less randomized depending on the degree of randomness that is wanted [4, 15]. It can be useful to predict randomized behaviors like in ecology where they try to predict geographic animal movements. They usually don't take an optimal path but they converge to it [21].

The randomness of these new algorithms is manipulated by defining a temperature variable. This variable balances the result between the optimal solution and pure random behavior. If the temperature is low, the result will tend to optimal solutions and if the temperature is high, it will lead to a more random solution [4, 15].

This master thesis compares randomized optimal flow algorithms versus optimal flow algorithms. The computation time of all implemented algorithms is compared. The comparison will be separated into two categories: the minimum cost flow problem and the maximum flow problem.

The remaining of the paper is as follows. Chapter 2 shows the applications of flow algorithms, while Chapter 3 explains all the concepts to understand the master thesis. Then, Chapter 4 defines the minimum cost flow problem and the maximum cost flow problem and presents standard optimal algorithms used in the comparison. The randomized optimal algorithms are introduced in Chapter 5. Chapter 6 explains the methodology and discusses the results of the comparison. Finally, Chapter 7 is the conclusion.

2 Application

Flow problems can arise in many applications. Few selected examples are explained in this chapter. The following sections contain respectively the transportation of goods, the scheduling of tasks, the improvement of internet quality of service, application in image processing, and finally, the prediction of animal behaviors.

2.1 Transportation of goods

Enterprises creating goods in many factories will have to transport them to geographically dispersed warehouses or directly to retailers. They want to ship them at the best cost to create an optimal profit. Minimum cost flow algorithms will find the solution to this problem [1, 16].

This problem can also be considered with a more complex supply chain. The raw/gross material being produced in multiple places has to be delivered to many transformation centers of product around the globe. This chain of production can be long and the use of an optimal solution can decrease the cost of shipping.

Tesla Motors is a good example of such a complicated supply chain. In 2017, it sells its products through an international network with over 350 suppliers and different facilities all over the world [12].

Randomized optimal cost flow algorithms can also be used to add some properties to the solution. The trucks will not have predictable behavior [4, 15] and it can reduce the risk of truck hijacking since the truck will take unpredictable paths.

2.2 Scheduling of tasks

Another well-known cost flow problem is the scheduling of tasks [1, 16]. Assigning people/machines to jobs is a common corporate problem. Each worker has a cost and will perform the task in a given time. Using a minimum cost flow solution will optimize the cost and the time to perform a set of tasks.

Imagine a company producing 3 different products with a set of 5 employees. Each of them has the knowledge to produce 2 types of product. Each good has a production cost and a selling price. A scheduling algorithm will assign each employee to the production of specific goods to optimize the income of the selling.

2.3 Improving the Internet quality of service

The Internet can be viewed as a big network with edge costs and capacities. The edge cost can be a distance, reliability, congestion, or an energy metric. On the other hand, the capacities can be bandwidth-distance or bandwidth-energy [24].

With the help of maximum and minimum cost flow algorithms, new routing algorithms have been developed to route aggregated video streams from cloud data centers in a Future-Internet network, with better energy efficiency and quality of service [24]. They give a solution to a multi-commodity flow problem. They are problems where there are multiple flow demands between different source and sink nodes [7].

2.4 Image processing and computer vision

Image processing and computer vision can be modeled in the form of energy minimization through Markov Random Fields and it is solved with maximum flow algorithms. Multiple applications exist like image segmentation, stereo, 3D reconstruction, shape-fitting, image synthesis, and photomontage [18, 29].

2.5 Prediction of animal movements

Animal movements like mass migrations are driven by spatial and temporal randomness because they follow suitable areas for activities such as feeding, resting, mating, raising newborns, and escaping predators. Unfortunately, migrations are especially vulnerable to habitat loss and fragmentation of the territory. This is caused by human infrastructure and it raises concerns for species conservation and human safety.

This concern was the source of the conception of a framework proposed to define the corridors of animal movement to try to limit the collision between them and humans. This is done with randomized optimal flow algorithms that try to map the zone of possible movement of these migrations [21].

3 Network theory

A quick reminder of network theory will be given to fully understand this master thesis. All the concepts explained in this part are used in the following chapters. First, a network definition is given with a mathematical representation. Then, residual networks are described and the transformation of networks into residual networks is explained. The following sections present network properties containing reduced cost, node excess, and the complementary slackness optimality conditions. This chapter terminates with an explanation of the labeling procedure.

3.1 Network definition

In this master thesis, the following notation for networks will be used. Let's denote the directed network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ containing the set of vertices $\mathcal{V} = \{1, 2, ..., n\}$ and the set of edges $\mathcal{E} = \{(i, j)\}$. The network is connected. It means there is a path between every pair of vertices. For each edge, we assign a flow, a capacity, and a cost. The flow is the number of units, noted x_{ij}, passing through the edge. The capacity is the maximum flow that can pass through the edge and it will be noted as u_{ij}. The cost is the price to pay to pass on the edge per unit of flow and it will be noted as c_{ij}. Finally, each node will be assigned a number, b_i, which represents the source/sink role. If $b_i > 0$, then the node is a source. If $b_i < 0$, then the node is a sink. When the node will be used as an intermediary, $b_i = 0$. Figure 3.1 is an example of such a network.

3.2 Structure representation

A network can be represented by using weighted adjacency matrix $\mathbf{A}$ and cost matrix $\mathbf{C}$ [5]. These matrices are of size $n \times n$ with n the number of nodes contained in the network. Each element a_{ij} in the adjacency matrix represent the local affinities between node i to node j (in this master thesis only binary affinities will be consider, so $a_{ij} \in [0, 1]$). Each element c_{ij} in the cost matrix is the cost to go from node i to node j. To illustrate this concept, let's take Figure 3.1. The corresponding adjacency matrix will be:

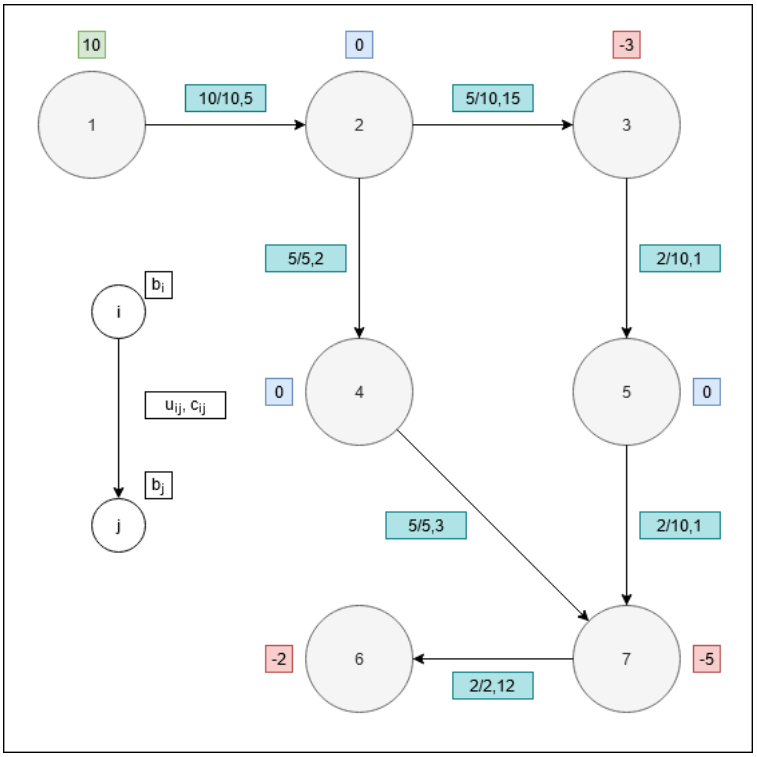

Fig. 3.1: This is an example of a network used in cost flow problems. b_i is represented in the small square next to the nodes. u_{ij} and c_{ij} are represented next to the edge in the form "Capacity, Cost". The optimal flow is mentioned next to the capacity.

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

and the corresponding cost matrix is:

$$\mathbf{C} = \begin{pmatrix} \infty & 5 & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 15 & 2 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 12 & \infty \end{pmatrix}$$

3.3 Residual network

The concept of residual network [1] is also needed to be able to understand the flow algorithms in Section 4. By picking up the Figure 3.1, we can transform it into such a network.

The creation of a residual network is always from a flow solution. On each edge of the network, the residual flow has to be computed. The formula is

$$r_{ij} = u_{ij} - x_{ij}$$

where u_{ij} is the capacity and x_{ij} the flow of edge (i, j).

Then we have to flow the residual flow in the opposite direction with negative cost. Figure 3.2 shows the residual network of the Figure 3.1[30]. The residual network of $\mathcal{G}$ will be noted $\mathcal{G}_x$.

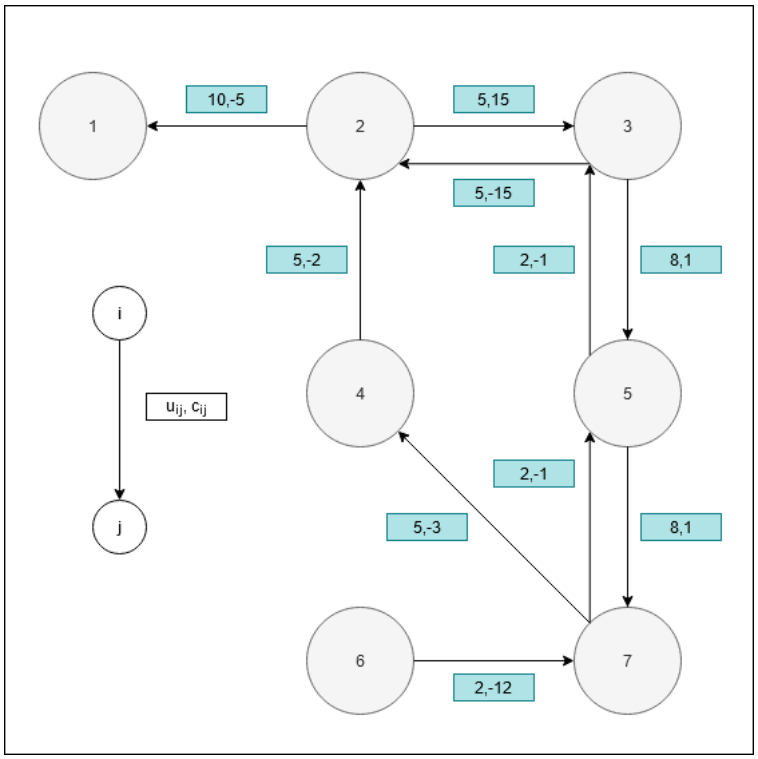

Fig. 3.2: This is the residual network of the Figure 3.1.

3.4 Reduced cost

Let's associate another quantity for each node of the network and call it the potential of node i [1]. The notation used in this document, to represent it, is π_i. It can be of

any value. In general, π_i is equal to the shortest path length from source to node i.

The reduced cost for an edge (i, j) is defined as

$$c_{ij}^{\pi} = c_{ij} + \pi_i - \pi_j \geq 0$$

This definition of this new cost for edges have an interesting property. Suppose W is a directed cycle, then :

$$\sum_{(i,j)\in W} c_{ij}^{\pi} = \sum_{(i,j)\in W} c_{ij}$$

This property tells us that if the network has a negative cycle, the cost of edges can be changed to delete the negative cycle without creating a new behavior in the network.

3.5 Node excess

The excess of node i is noted $e(i)$. It described the difference between the output flow and the input flow [1]. $\mathcal{I}$ is the set of edges ending in node i and $\mathcal{O}$ is the set of edges starting from node i. More formally it can be defined as:

$$e(i) = \sum_{(j,i)\in\mathcal{I}} x_{ji} - \sum_{(i,j)\in\mathcal{O}} x_{ij}$$

By example, the source node will always have a negative excess because it has only output flow, and the sink node a positive excess because it has only input flow. An active node is one with a positive excess and the source and sink node are never active by convention.

3.6 Complementary slackness optimality conditions

A feasible solution x^* is an optimal solution of the minimum cost flow problem if and only if for some set of node potentials π, the reduced costs and flow values satisfy the following complementary slackness optimality conditions for every arc $(i, j) \in \mathcal{E}$ [1] :

$$\text{If} \quad c_{ij}^{\pi} > 0, \quad \text{then} \quad x_{ij}^{*} = 0$$

$$\text{If} \quad 0 < x_{ij}^{*} < u_{ij}, \quad \text{then} \quad c_{ij}^{\pi} = 0$$

$$\text{If} \quad c_{ij}^{\pi} < 0, \quad \text{then} \quad x_{ij}^{*} = u_{ij}$$

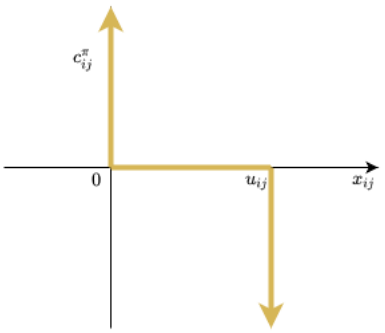

Fig. 3.3: Visualisation of the complementary slackness optimality conditions. If the point (x_{ij}, c_{ij}^{π}) lies on the thick yellow lines in the diagram, the arc verify the condition.

3.7 Labeling procedure

The labeling procedure [8, 9, 11] searches a path connecting indirectly two nodes defined. The path can be made of directed edges and reverse directed edges. First, the method will be explained. Then, an example will be shown to illustrate the procedure.

Let's take a network $\mathcal{G}$ with the will to launch the labeling procedure for nodes i and j. A node is chosen as the origin, let's say i, and is labeled with the vector $[0, \infty]$. The second node, j, is chosen as the terminal. The following rules are now applied:

1. Node i is labeled $[k^{+-}, e_i]$, node j is unlabeled and $(i, j) \in \mathcal{E}$ with the following constraints:

$$(a) \quad c_{ij}^{\pi} < 0, \quad x_{ij} < 0$$

$$(b) \quad c_{ij}^{\pi} \leq 0, \quad x_{ij} < u_{ij}$$

   Then node j is assigned to label $[i^{+}, e_j]$ where $e_j = min(e_i, -x_{ij})$ in case (a) and $e_j = min(e_i, u_{ij} - x_{ij})$ in case (b).

2. Node i is labeled $[k^{+-}, e_i]$, node j is unlabeled and $(j, i) \in \mathcal{E}$ with the following constraints:

$$(a) \quad c_{ji}^{\pi} \leq 0, \quad x_{ji} < 0$$

$$(b) \quad c_{ji}^{\pi} < 0, \quad x_{ji} < u_{ji}$$

Then node j is assigned to label $[i^-, e_j]$ where $e_j = min(e_i, x_{ji})$ in case (a) and $e_j = min(e_i, x_{ji} - u_{ji})$ in case (b).

This procedure terminates in two possible ways. The first one is called breakthrough and the terminal node is labeled. The second one is called non-breakthrough and the terminal node is not labeled. In a breakthrough, a path is found and can be obtained by backtracking the label from the terminal node. If the label is $[i^+, e_j]$ then (i, j) is a forward edge of the path and if the label is $[i^-, e_j]$ then (j, i) is a reverse edge of the path.

Figure 3.4 shows an example of a path find by this method with a defined network.

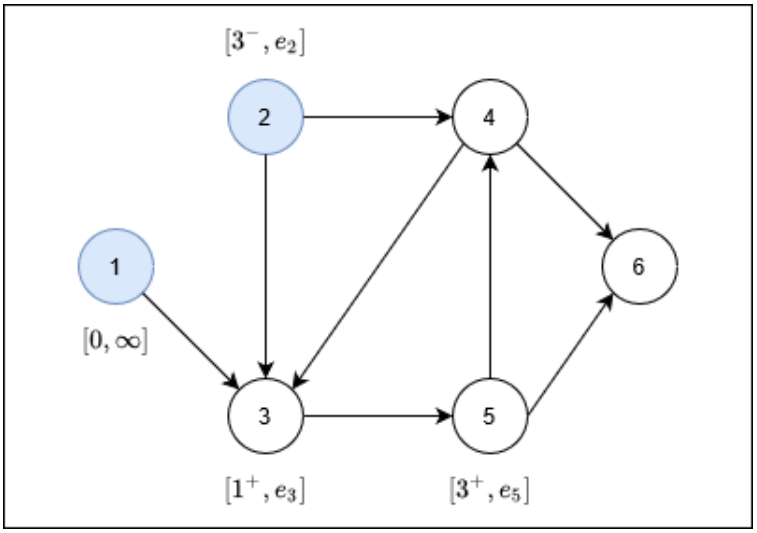

Fig. 3.4: The origin is node 1 and terminal is node 2. The origin is labeled. Node 3 is labeled by node 1. Node 5 is labeled by node 3. Node 2 is labeled by node 3. The procedure terminate because terminal node is labeled. The path is $[1, 3]$ and $[2, 3]$.

4 Standard optimal flow algorithms

This chapter contains the flow problems definition and briefly introduces standard optimal flow algorithms. In Section 1, minimum cost flow and maximum flow problems are described mathematically. Section 2 introduces linear programming and discusses the simplex method that is used to solve such problems. Section 3 presents the minimum cost flow algorithms that are used in the comparison. Finally, Section 4 presents the maximum flow algorithm used in the comparison.

4.1 Flow problems

As explained in several documents [1, 16], the minimum cost and maximum flow problems can be described as written in the following subsections. An example will also be shown to consolidate the understanding of the concepts.

4.1.1 Minimum cost flow problem

The network is a directed and connected network. At least one of the nodes is a supply node. At least one of the other nodes is a demand node. The network can have multiple sources and multiple destinations. All the remaining nodes are transshipment nodes. The flow through an arc is allowed only in the direction indicated by the arrowhead, where the maximum amount of flow is given by the capacity of that arc. If the flow can occur in both directions, this would be represented by a pair of arcs pointing in opposite directions. The network is supposed to have enough arcs with sufficient capacity to enable all the flow generated. The cost of the flow passing through each arc is proportional to the amount of that flow, where the cost per unit flow is known. The objective is to minimize the total cost of sending the available supply through the network to satisfy the demand.

From a mathematical view, the problem can be represented as the following optimization problem with the following constraints:

$$Minimize \quad z(x) = \sum_{(i,j)\in\mathcal{E}} c_{ij}x_{ij}$$

subject to

$$\sum_{(i,j)\in\mathcal{O}} x_{ij} - \sum_{(j,i)\in\mathcal{I}} x_{ji} = b_i \quad \forall i \in \mathcal{V}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in \mathcal{E}$$

where, as explained in Chapter 3, x_{ij}, c_{ij} and u_{ij} are respectively the flow, the cost and the capacity of edge (i,j). b_i is the demand or the offer of the node i. $\mathcal{I}$ and $\mathcal{O}$ are the sets of edges coming to node i and going out of node i.

4.1.2 Maximum flow problem

As before, all flow through a directed and connected network originates at one node, called the source s, and terminates at one other node, called the sink t. All the remaining nodes are transshipment nodes. The flow passing through an arc is allowed only in the direction indicated by the arrowhead, where the maximum amount of flow is given by the capacity of that arc. At the source, all arcs point away from the node. At the sink, all arcs point into the node. The objective is to maximize the total amount of flow from the source to the sink. This amount is measured in either of two equivalent ways, namely, either the amount of flow leaving the source or the amount of flow entering the sink.

From a mathematical view, the problem can be represented as the following optimization problem with the following constraints:

$$Maximize \quad v$$

subject to

$$\sum_{(i,j)\in\mathcal{O}} x_{ij} - \sum_{(j,i)\in\mathcal{I}} x_{ji} = \begin{cases} v & \text{for} \quad i = s \\ 0 & \forall i \in \mathcal{V} - \{s,t\} \\ -v & \text{for} \quad i = t \end{cases}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in \mathcal{E}$$

where, as explained in Chapter 3, x_{ij} and u_{ij} are respectively the flow and the capacity of edge (i,j). $\mathcal{I}$ and $\mathcal{O}$ are the sets of edges coming to node i and going out of node i.

4.1.3 Example

Let's take a cost flow problem example from chapter 8 in "Introduction to Operations Research"[16]. Consider a company that has to deliver goods from their factories to their customers. Several paths exist to send the goods to their destination as mentioned in Figure 4.1.

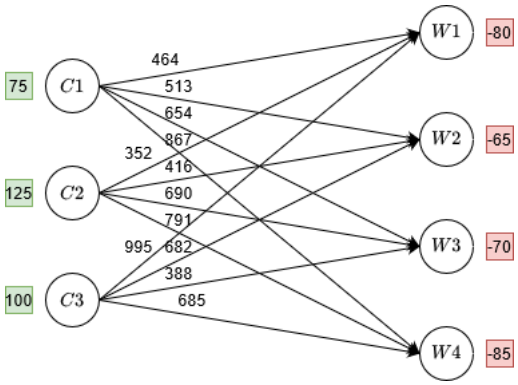

Fig. 4.1: $C1$, $C2$, and $C3$ are the factories. $W1$, $W2$, $W3$, and $W4$ are the customers. The number next to the factories is the number of products being created. The number next to the customers is the number of products to be delivered. Each factory is linked to the customers with a cost by unit. Let denote that the capacity u_{ij} of each edge is 1000 units [16].

In the case of the minimum cost flow problem, we will compute the best flow of the goods produced in the factories towards the client considering the costs, the capacity of the paths, the number of produced products, and the demand of the clients. This example is used in Subsection 4.2 where the solution is presented.

In the case of the maximum flow problem, one node s is created to link all the source nodes and one node t is created to link all the sink nodes as explained in the definition presented in Subsection 4.1.2. Here only the capacity will be considered and the computation will discover the maximum flow of products possibly send from the factories to the clients. In this case, as the capacity of each edge is 1000 units, the maximum flow possible will be 4000 units of flows from each factory. It means a flow of 12.000 units.

4.2 Linear programming

Linear programming [16, 19] uses a mathematical framework to describe the problem of concern. It can then solve minimum cost flow problems and maximum flow problems. The adjective linear means that all the mathematical functions in this model are required to be linear functions. The word programming does not refer here to computer programming; rather, it is essentially a synonym for planning.

Thus, linear programming involves the planning of activities to obtain an optimal result among all feasible alternatives. Several ways exist to solve linear programming problems. In this section, only the simplex method will be discussed.

The linear programming implementations used during the comparison of algorithms are GLOP and CVXOPT. The first one is a method developed by Google and it uses an optimized simplex algorithm. The method is not explained in this work but more information can be found on their website [14]. The second one is a method for convex optimization based on the Python programming language [2].

4.2.1 Description of the problem

The example from the definition of the cost flow problem in Section 4.1.3 will be used. Let create the variable

$$x_{ij}$$

to be the road from factory i to customer j. We have to minimize the cost, declared z:

$$\begin{aligned} z = 464x_{11} + 513x_{12} + 654x_{13} + 867x_{14} \\ +352x_{21} + 416x_{22} + 690x_{23} + 791x_{24} \\ +995x_{31} + 682x_{32} + 388x_{33} + 685x_{34} \end{aligned}$$

subject to the constraints of the demand/supply:

$$\begin{aligned} x_{11} + x_{12} + x_{13} + x_{14} = 75 \\ x_{21} + x_{22} + x_{23} + x_{24} = 125 \\ x_{31} + x_{32} + x_{33} + x_{34} = 100 \\ x_{11} + x_{21} + x_{31} = 80 \\ x_{12} + x_{22} + x_{32} = 65 \\ x_{13} + x_{23} + x_{33} = 70 \\ x_{14} + x_{24} + x_{34} = 85 \end{aligned}$$

subject to the constraints of the flow:

$$0 \leq x_{ij} \leq 1000 \quad \forall (i,j) \in \mathcal{E}$$

4.2.2 Simplex method

The simplex method is a method to solve the problem mentioned in Subsection 4.2.1. The method is described in chapter four in the book "Introduction to operations research" [16]. It isn't the most optimal one but it gives an understanding of how such problems can be solved using linear programming. First, the original form of the model will be transformed into the augmented form of the model. Then the algorithm will be executed.

The augmented form is obtained by converting the functional inequality constraints into equivalent equality constraints by introducing slack variables. In our example, all the constraints are equations. Our model is already in the augmented form.

Then, the variables must be split into 2 categories. On one side are the basic variables and on the other side are the nonbasic variables. The number of nonbasic variables is equal to the number of variables minus the number of constraints equations. In our example, this means 12-7=5. These five variables, chosen randomly, will be set to zero at the initialization phase. The system of equations can now be resolved. It will lead to a feasible solution.

If the augmentation of a nonbasic variables lead to a better solution, it means that the feasible solution isn't optimal. The nonbasic variable will be increased while adjusting other variable values to satisfy the system of equations. Stop to increase when the first basic variable drop to zero. A new feasible solution is now discovered. We repeat this process until none increase of the nonbasic variable lead to a better solution. In our example, the best solution is z, equal to 152.535 with the following x_{ij}:

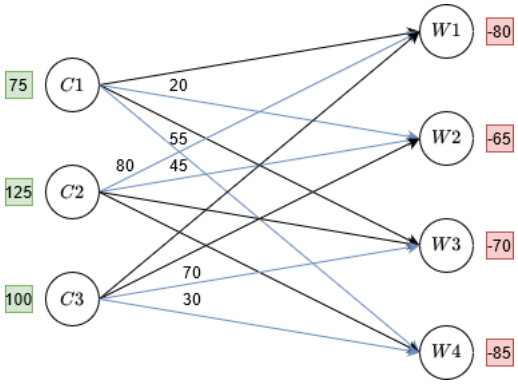

Fig. 4.2: Solution from Figure 4.1 where $x_{11} = 0; x_{12} = 20; x_{13} = 0; x_{14} = 55; x_{21} = 80; x_{22} = 45; x_{23} = 0; x_{24} = 0; x_{31} = 0; x_{32} = 0; x_{33} = 70; x_{34} = 30$

Several other subtleties exist but none are useful to the general comprehension of the method.

4.3 Minimum cost flow algorithms

These sections present the minimum cost flow algorithms used in the comparison. They all use different network properties. The pseudo-code is given for each algorithm and the main intuition of the execution is given. The time complexity is also presented. Some concepts used in this part can be found in Section 3.

4.3.1 Cycle-canceling algorithm

This algorithm is based on the negative cycle optimality condition. Let $\mathbf{X}$ be a feasible solution to a minimum cost flow problem. Then $\mathbf{X}$ is an optimal solution if and only if the residual network $\mathcal{G}_x$ contains no negative cost (directed) cycle. From a feasible solution $\mathbf{X}$, the algorithm will seek a negative cycle in the residual network $\mathcal{G}_x$. Then it will augment the units of flow along the cycle until the end of the negative cycle. It will repeat the process until no more negative cycle exists. Finally, it will transform the residual network into the initial network format [1].

Algorithm 1 is the pseudo-code taken from [30]. In this algorithm, the variable r_{ij} is the residual flow of (i, j) as explained in Subsection 3.3.

Algorithm 1: Cycle-canceling algorithm

```
Establish a feasible flow X in the network
while G_x contains a negative cycle do
    identify a negative cycle w
    σ = min(r_ij : (i, j) ∈ w)
    augment σ units of flow along the cycle w
    update G_x
end
```

The complexity of this algorithm is expressed with the following variables: M is the maximum capacity of an arc, H is the maximum absolute value of cost, $|\mathcal{E}|$ denotes the number of edges, and $|\mathcal{V}|$ denotes the number of vertices. It requires $O(|\mathcal{V}||\mathcal{E}|)$ to identify a negative cycle (using Bellman-Ford's algorithm by example) and $O(|\mathcal{E}|HM)$ to decrease any cycle. O($|\mathcal{V}||\mathcal{E}|^2HM$) is then the complexity of this algorithm. Some variants and optimizations of the algorithm can be found in [23].

4.3.2 Successive shortest path algorithm

In this algorithm, two vertices are added to the network. One source will be noted s and one sink which will be noted t. These vertices are connected to the network following this logic: if a node i in the network has $b_i > 0$, an edge with $u_{si} = b_i$ and

cost 0 is created between the source s and the node i. Oppositely, if a node j in the network has $b_j < 0$, an edge with $u_{tj} = -b_j$ and cost 0 is created between the sink t and the node j.

Then, the residual graph $\mathcal{G}_x$ will be computed. If $\mathcal{G}_x$ contains a path $\wp$ from s to t, the algorithm augments the current flow along $\wp$. It loops on these steps until $\mathcal{G}_x$ doesn't have any more path from s to t. This algorithm works unless a negative cost cycle is found in the residual network. Indeed the shortest path doesn't have a sense when a negative cycle exists.

We need to keep the edge costs non-negative on each iteration to prevent a negative cycle to appear. Node potentials are introduced to solve this problem as explained in Subsection 3.4.

The algorithm 2 is the pseudo-code taken from [30]. The original description of the algorithm can be found in [6].

Algorithm 2: Successive Shortest Path with potentials Algorithm

```
Transform network G by adding source and sink
Initial flow X is zero
Use Bellman-Ford's algorithm to establish potentials π
Reduce Cost(π)
while G_x contains a path from s to t do
    Find any shortest path ℘ from s to t
    Reduce Cost(π)
    Augment current flow X along ℘
    update G_x
end
```

Bellman-Ford's algorithm is called once to avoid negative costs on edges and takes $O(|\mathcal{V}||\mathcal{E}|)$ time. Dijkstra algorithm is called $O(|\mathcal{V}|B)$ times, where B is assigned to an upper bound on the largest supply of any node, and takes per call $O(|\mathcal{E}| \log |\mathcal{V}|)$. It can be conclude that the complexity is $O(|\mathcal{V}||\mathcal{E}|B \quad \log |\mathcal{V}|)$.

4.3.3 Primal-dual algorithm

The primal-dual algorithm is a variant of the successive shortest path algorithm. Instead of augmenting one shortest path at a time, it augments all shortest paths at once by using a maximum flow algorithm inside the so-called admissible network, which contains only those arcs in $\mathcal{G}_x$ with a zero reduced cost ($\mathcal{G}_x^0$) [1].

The algorithm 3 is the pseudo-code taken from [30].

Algorithm 3: Primal Dual algorithm

Transform network $\mathcal{G}$ by adding source and sink
Initial flow $\mathbf{X}$ is zero
Use Bellman-Ford's algorithm to establish potentials π
Reduce Cost(π)
while $\mathcal{G}_x$ *contains a path from* s *to* t **do**
    Calculate node potential π using Dijkstra's algorithm
    Reduce Cost(π)
    Establish a maximum flow from s to t in $\mathcal{G}_x^0$
    update flow $\mathbf{X}$
end

As a quick reminder, the reduce cost function is already explained in Subsection 3.4.

The iteration will not exceed $O(|\mathcal{V}|B)$ as mentioned in the successive shortest path algorithm. With M the maximum capacity of an arc and H the maximum absolute value of cost, if $S(|\mathcal{V}|, |\mathcal{E}|, H)$ and $Q(|\mathcal{V}|, |\mathcal{E}|, M)$ denote the solution times of shortest path and the maximum flow algorithms, the primal-dual algorithm has an overall complexity of $O(\{|\mathcal{V}|B\}.\{S(|\mathcal{V}|, |\mathcal{E}|, |\mathcal{V}|H) + Q(|\mathcal{V}|, |\mathcal{E}|, M)\})$.

4.3.4 Out-of-kilter algorithm

The name out-of-kilter algorithm reflects the fact that arcs in the network are either in-the-kilter or out-the-kilter. The complementary slackness optimality condition (mentioned in Section 3.6) determine if an arc is in or out. If the condition is accepted, the arc is in-the-kilter. If the condition is denied, the arc is out-the-kilter. The algorithm will loop through all out-of-kilter edges to insert them in-the-kilter. When the algorithm has transformed all the edges, the solution is then feasible and optimal as mentioned in the complementary slackness optimality condition. Inserting an out-of-kilter edge (i, j) into the kilter requires finding a path from node j to node i with the labeling procedure explained in Section 3.7. If a path is found, this path is augmented. If no paths are found, we compute a variable which is called σ. It is the minimum reduced cost between labeled nodes and not labeled nodes. If there are no such edges, (i, j) is added directly inside the kilter. If σ is computed, we add it to the π of unlabeled nodes [11].

Algorithm 4 is the pseudo-code created from the paper of Fulkerson Delbert R, page 21-23, [11].

With M the maximum capacity of an arc and H the maximum absolute value of cost, if $S(|\mathcal{V}|, |\mathcal{E}|, H)$ is the time required to solve the shortest path problem with non-

Algorithm 4: Out-of-kilter algorithm

```
π=0
establish a feasible flow X in the network
compute the kilter numbers of arcs
while the network contains an out-of-kilter arc do
    select an out-of-kilter arc (i, j) in G
    ℘ = find a path from j to i with the labeling procedure
    if ℘ is empty then
        compute σ
        if σ is infinite then
            force the edge to be in-the-kilter
        else
            add σ to π for all unlabeled nodes
        end
    else
        add flow to the path
        choose another out-of-kilter arc
    end
end
```

negative arc lengths, the out-of-kilter algorithm runs in $O(|\mathcal{E}|M * S(|\mathcal{V}|, |\mathcal{E}|, |\mathcal{V}|H))$ time with $|\mathcal{E}|M$ as the sum of maximum kilter numbers.

4.4 Maximum flow algorithms

These sections present the maximum flow algorithms used in the comparison. They all use different network properties. The pseudo-code is given for each algorithm and the main intuition of the execution is given. The time complexity is also presented. Several concepts used in this part can be found in Section 3.

4.4.1 Augmenting path algorithm

It is the simplest and most naive algorithm for solving the maximum flow problem. It creates the residual network of $\mathcal{G}$ and finds an augmenting path from the source to the sink. It updates $\mathcal{G}_x$. The process continues until no more augmenting path exists. Algorithm 5 is the pseudo-code taken from Ahuja book, page 180-183 [1]. In this algorithm, the variable r_{ij} is the residual flow of (i, j) as explained in Subsection 3.3.

How to identify an augmenting path or show that the network contains no such paths? In this work, the breadth-first search will be used (worst complexity case of $O(|\mathcal{V}| + |\mathcal{E}|)$). Other algorithms like the labeling algorithm can do the job in a

Algorithm 5: Augmenting Path Algorithm

$\mathcal{G}_x$ is the residual network of $\mathcal{G}$
while $\mathcal{G}_x$ *contains a directed path from* $source$ *to* $sink$ **do**
    Identify an augmenting path $\wp$ from node $source$ to node $sink$
    $\sigma := min(r_{ij} : (i,j) \in \wp)$
    Augment σ units of flow along $\wp$
    Update $\mathcal{G}_x$
end

worst-case computational complexity of $O(k|\mathcal{E}|M)$, k the number of augmentation and M the maximum capacity of an arc

4.4.2 FIFO preflow-push algorithm

The particularity of preflow-push is that it increases the flow on individual edges instead of an augmenting path, like the previous algorithm. This property improves the execution time in some contexts because the drawback of the augmenting path algorithm was the expensive operation of sending flow through a path.

During the computation, the network will have an infeasible solution caused by the active nodes (Section 3.5). Consequently, the basic operation in this algorithm is to select an active node and try to remove its excess by pushing flow to its neighbors. Where should the flow be sent to? The flow must arrive in the sink so the flow will be pushed towards nodes closer to the sink. The algorithm 6 is the pseudo-code taken from Ahuja book [1], page 223-232. The original description of this family of algorithms can be found in (Goldberg et al.) [13].

Algorithm 6: FIFO Preflow-Push Algorithm

$\mathbf{X} = 0$
compute the exact distance labels $d(i)$
$x_{ij} = u_{ij}$ for each arc $(source, j)$ (with node j neighbor of node $source$)
$d(source) = n$
while *The network contains an active node* **do**
    select an active node i
    if *The network contains an admissible arc* (i,j) **then**
        push $\sigma = min(e(i), r_{ij})$ units of flow from node i to node j
    else
        replace $d(i)$ by $min(d(j) + 1 : (i,j) \in A(i))$
    end
end

The way the current active node is selected will determine the complexity of the algorithm. In this master thesis, the FIFO implementation has been chosen. The complexity of the algorithm is $O(|\mathcal{V}|^3)$.

5 Randomized optimal flow algorithms

This chapter briefly introduces the randomized optimal flow algorithms used in the comparison. Section 1 presents the constrained bag-of-paths algorithms. It resolves the minimum cost flow problems with relaxation depending on the temperature. Section 2 presents the randomized shortest paths with capacity constraints. It resolves the maximum flow problems with relaxation depending on the temperature.

5.1 Constrained bag-of-paths algorithm

This algorithm is based on the bag-of-paths (BoP) framework [10, 20] which consists of a Gibbs-Boltzmann distribution on all feasible paths of a network. This probability distribution favors short paths over long ones, with a free parameter (the temperature T) controlling the entropic level of the distribution. When the temperature is low, it will favor the shortest path. The randomization is achieved by increasing the temperature. This distribution can be obtained by solving the following problem.

$$minimize_{\{\mathrm{P}(\wp)_{\wp\in\mathcal{P}}\}} \quad \mathrm{FE(P)} = \sum_{\wp\in\mathcal{P}} \mathrm{P}(\wp)\widetilde{c}(\wp) + T\sum_{\wp\in\mathcal{P}} \mathrm{P}(\wp)\log(\frac{\mathrm{P}(\wp)}{\mathrm{P}^{ref}(\wp)})$$

subject to

$$\sum_{\wp\in\mathcal{P}} \mathrm{P}(\wp) = 1$$

where $\mathbf{P}^{ref}(\wp)$ is the likelihood of the path $\wp$, that is, the product of the transition probabilities given by the transition matrix $\mathbf{P}^{ref}$made of $p_{ij}^{ref} = \frac{a_{ij}}{\sum_{k\in\mathrm{V}} a_{ik}}$. $\mathcal{P}$ is the set of paths. $\wp$ is a path inside the bag of paths and is made of a list of nodes. $\widetilde{c}(\wp)$ is the total cost of a path $\wp$. $\mathrm{P}(\wp)$ is the probability of picking path $\wp$ inside the bag-of-paths. FE(P) is called the free energy and corresponds to the expected cost.

The BoP formalism is extended by adding two independent equality constraints fixing starting and ending nodes distributions of paths. This new model assumes that the user knows only where paths on average start and where they on average end. σ_i^{in} is the probability of node i to be the first node of the path $\wp$. σ_j^{out} is the probability of node j to be the last node of the path $\wp$. These two new constraints can be added to the BoP formalism equations [15].

$$\sum_{j \in \mathcal{V}} \sum_{\wp_{ij} \in \mathcal{P}_{ij}} \mathrm{P}(\wp_{ij}) = \sigma_i^{in} \quad \forall i \in \mathcal{V}$$

$$\sum_{j \in \mathcal{V}} \sum_{\wp_{ij} \in \mathcal{P}_{ij}} \mathrm{P}(\wp_{ij}) = \sigma_j^{out} \quad \forall j \in \mathcal{V}$$

This new model is called margin-constrained BoP model and can be defined as a **randomized policy for the minimum cost flow problem** [27, 26] on a network because the starting and ending node distributions can be considered as supply and demand distributions for goods meant to be transported over the network. The randomization is achieved by increasing the temperature variable [15].

The algorithm considers two different types of paths: non-hitting paths where the ending node can appear as an intermediate node and hitting paths where the ending node only appears as the end of the path. The algorithm is described in [15] and will not be explained in this master thesis. The hitting algorithm is available on page 108 of [15]. The non-hitting algorithm is available on page 102 of [15]

Each algorithm will take as input the adjacency matrix $\mathbf{A}$, the cost matrix $\mathbf{C}$, the supply/demand nodes represented in vector σ and θ which is $\frac{1}{T}$. The output matrix that will be giving the solution is the matrix $\bar{\mathbf{N}}$. It contains the expected number of times an edge appears on the flows. It will represent the flow through the network.

5.1.1 Illustration

In this subsection, the hitting bag-of-paths has been chosen to illustrate the constrained bag-of-paths algorithm. Figure 5.1 represents the network chosen as an example. The hitting bag-of-path is computed on this network with variation of the attribute θ, which is $\frac{1}{T}$, to illustrate the flow inside each edge.

We can observe that the result converge to the optimal solution with $\theta = 0.01, 0.1, 1$. The result is optimal with $\theta = 2, 5, 10$.

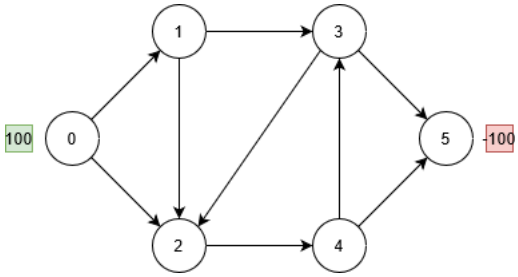

Fig. 5.1: A small directed network composed of six nodes. The cost of each edge is five. The supply is set to 100 and the demand to -100. The optimal solution to the minimum cost flow problem is a total cost of 1500.

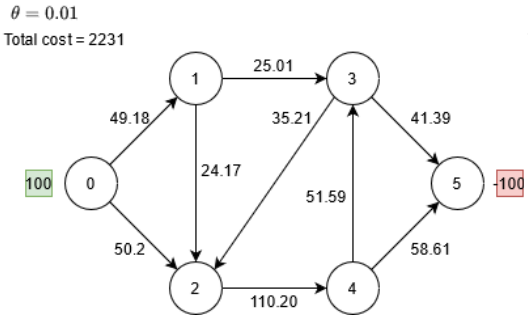

Fig. 5.2: Solution with θ set to 0.01

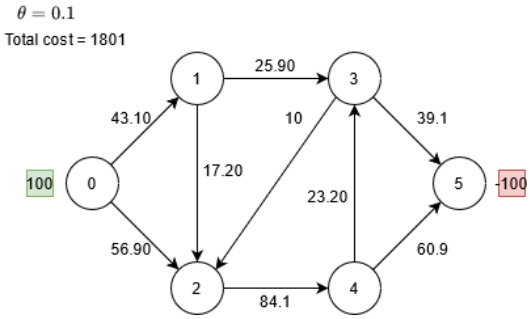

Fig. 5.3: Solution with θ set to 0.1

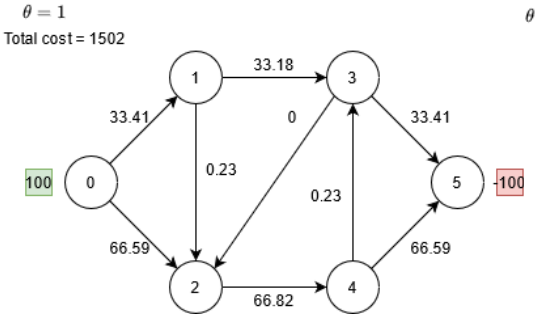

Fig. 5.4: Solution with θ set to 1

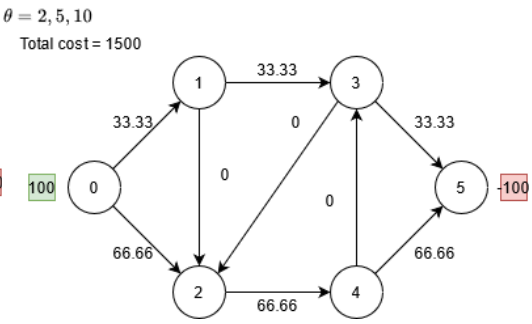

Fig. 5.5: Solution with θ set to 2, 5, 10

5.2 Randomized shortest paths with capacity constraints algorithm

This algorithm is based on the randomized shortest paths (RSP) framework [17, 22, 28] which consist of a Gibbs-Boltzmann distribution on all feasible paths of a network. The RSP framework is the same that the bag-of-paths framework except that it restricts the set of paths to hitting paths and only considers one source node and one target node. This distribution can be obtained by solving the following equations.

$$minimize_{\{\mathrm{P}(\wp)_{\wp\in\mathcal{P}_{st}}\}} \quad \phi(\mathrm{P}) = \sum_{\wp\in\mathcal{P}_{st}} \mathrm{P}(\wp)\widetilde{c}(\wp) + T \sum_{\wp\in\mathcal{P}_{st}} \mathrm{P}(\wp)\log(\frac{\mathrm{P}(\wp)}{\mathrm{P}^{ref}(\wp)})$$

subject to

$$\sum_{\wp\in\mathcal{P}_{st}} \mathrm{P}(\wp) = 1$$

As before, $\mathbf{P}^{ref}(\wp)$ is the likelihood of the path $\wp$, that is, the product of the transition probabilities given by the transition matrix $\mathbf{P}^{ref}$ made of $p_{ij}^{ref} = \frac{a_{ij}}{\sum_{k\in\mathrm{V}} a_{ik}}$. $\mathcal{P}_{st}$ is the set of paths starting from source s and ending in sink j. $\wp$ is a path inside the bag of paths and is made of a list of nodes. $\widetilde{c}(\wp)$ is the total cost of a path $\wp$. $\mathrm{P}(\wp)$ is the probability of picking path $\wp$ inside the bag-of-paths. $\phi(P)$ is called the free energy.

The algorithm used in this master thesis is obtained by adding capacity constrained to the previous equations [4]. Matrix $\mathbf{U}$ denote the capacity of the edges of the network. Therefore the flow is constrained by the following equations:

$$x_{ij} \leq \sigma_{ij} \quad \text{for edges} \quad (i,j) \in \mathbf{U}$$

The Lagrange function in case of capacity constraints is

$$\mathcal{L}(\mathrm{P},\lambda) = \sum_{\wp\in\mathcal{P}_{st}} \mathrm{P}(\wp)\widetilde{c}(\wp) + T \sum_{\wp\in\mathcal{P}_{st}} \mathrm{P}(\wp)\log(\frac{\mathrm{P}(\wp)}{\widetilde{\pi}(\wp)}) + \mu(\sum_{\wp\in\mathcal{P}_{st}} \mathrm{P}(\wp) - 1) + \sum_{(i,j)\in\mathbf{U}} \lambda_{ij}(x_{ij} - \sigma_{ij})$$

where μ is associated with the sum-to-one constraint and where a Lagrange parameter is associated with each constrained edge. λ_{ij} with $(i,j) \in \mathbf{U}$ are all non-negative in the case of inequality constraints and are stacked into the parameter vector λ.

The Lagrange function can be arranged by using the properties of the convexity of the Kullback-Leibler divergence [3].

$$\mathcal{L}(\mathrm{P},\lambda) = \sum_{\wp\in\mathcal{P}_{st}} \mathrm{P}(\wp)\widetilde{c}'(\wp) + T\sum_{\wp\in\mathcal{P}_{st}} \mathrm{P}(\wp)\log(\frac{\mathrm{P}(\wp)}{\widetilde{\pi}(\wp)}) + \mu(\sum_{\wp\in\mathcal{P}_{st}} \mathrm{P}(\wp) - 1) - \sum_{(i,j)\in\mathbf{U}} \lambda_{ij}\sigma_{ij}$$

where the costs c'_{ij} has been redefined into augmented costs integrating the additional "virtual" costs needed for satisfying the constraints. This cost can be computed following this equation:

$$c'_{ij} = \begin{cases} c_{ij} + \lambda_{ij} & \text{when edge } (i,j) \in \mathbf{U} \\ c_{ij} & \text{otherwise} \end{cases}$$

The RSP with capacity constraints can be solve by iterating through the following steps until convergence,

$$\begin{cases} \mathcal{L}(\lambda) = \mathcal{L}(\mathrm{P}^*(\lambda),\lambda) = minimize_{\{\mathrm{P}(\wp)_{\wp\in\mathcal{P}_{st}}\}}\mathcal{L}(\mathrm{P},\lambda) & \text{(compute the dual function)} \\ \lambda^* = \arg\max_\lambda \mathcal{L}(\mathrm{P},\lambda) & \text{(maximize the dual function)} \\ \lambda = \lambda^* & \text{(update } \lambda\text{)} \end{cases}$$

The solution of these equations can be defined as a **randomized policy for the maximum flow problem** [4]. In addition, the transition probabilities defining the optimal policy of the random walker can be easily derived. The pseudo-code of the algorithm can be found on page 20 of [4] and will not be explained in this master thesis.

The algorithm will take as input the adjacency matrix $\mathbf{A}$, the capacity matrix $\mathbf{U}$, the supply and demand nodes, θ which is $\frac{1}{T}$ and α which is a gradient ascent step. The output matrix that will be giving the solution is the matrix $\bar{\mathbf{N}}$. It contains the expected number of times an edge appears on the path. It will represent the flow through the network. From this matrix, the maximum flow possible inside the network can be computed.

5.2.1 Illustration

In this subsection, the randomized shortest path with capacity constraints is illustrated. Figure 5.6 represent the network chosen as example. The randomized

shortest path with capacity constraints is computed on this network with variation of the attribute θ, which is $\frac{1}{T}$, to illustrate the flow inside each edge. α is set to $\frac{1}{\theta}$.

We can observe that the result converge to the optimal solution with $\theta = 0.01, 0.1, 1$. The result is optimal with $\theta = 2, 5, 10$.

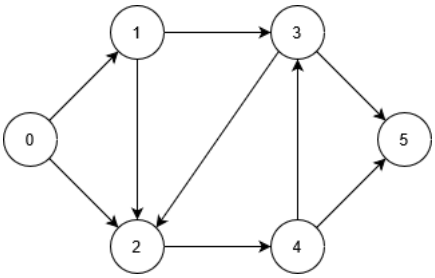

Fig. 5.6: A small directed network composed of six nodes. The source is node 0 and the sink is node 5. The capacity of each edge is 20. The optimal solution to the maximum flow problem is a maximum flow of 40.

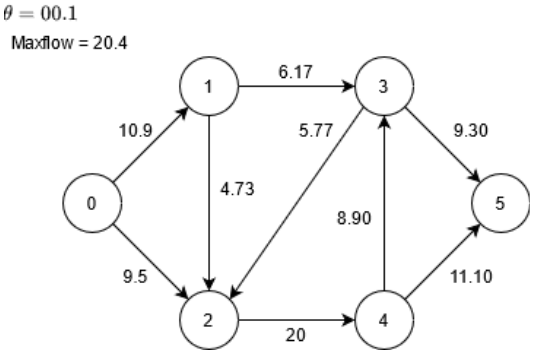

Fig. 5.7: Solution with θ set to 0.01

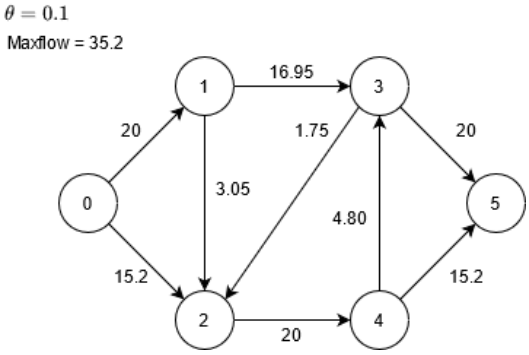

Fig. 5.8: Solution with θ set to 0.1

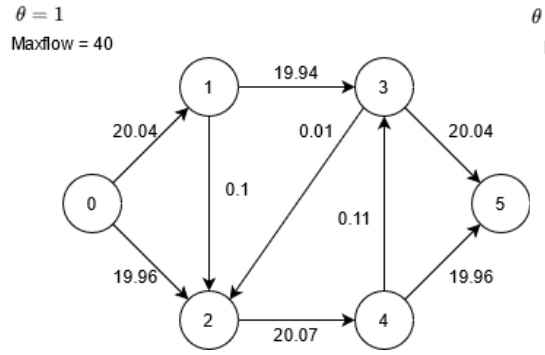

Fig. 5.9: Solution with θ set to 1

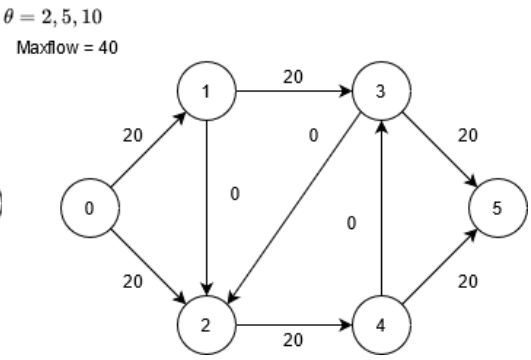

Fig. 5.10: Solution with θ set to 2, 5 or 10

6 Comparison

This chapter contains a comparison of the previously described algorithms. Section 1 contains the methodology of the comparison. Section 2 discusses the comparison of the minimum cost flow algorithms. Section 3 discusses the comparison of the maximum flow algorithms. Section 4 is the conclusion of the Chapter.

6.1 Methodology

The methodology is divided into four subsections to avoid any confusion. The implementation in Python is first explained. Then, the list of algorithms used is reminded. The datasets for the computation are shown and finally, the method to make the comparison is explained.

6.1.1 Codes

Python3 has been chosen to implement the algorithms for overhead reasons and because it is one of the most popular languages nowadays. This code, defined on Section 4 and 5, can be found on the following Github repository :

```
https://github.com/Wvisee/Optimal-and-randomized-flow-algorithms
```

The code is divided into the four following folders:

1. Maximum flow: containing all the maximum flow algorithms in separate files.

2. Minimum cost flow: containing all the minimum cost flow algorithms in separate files.

3. API: containing all the generic functions used by the algorithms (by example converting a network to a residual network, a residual network to a network, augmenting flow toward a path, etc.).

4. Root: containing all the files necessary to compares the algorithms and to output graphs from this section.

6.1.2 Algorithms

Here are the algorithms used for the comparison. They are separated into 4 bags. The algorithms can be either a maximum flow algorithm or a minimum cost flow algorithm. In addition, the algorithm is either in the family of optimal algorithms or in the family of the randomized one. For overhead reasons, a smallest identifier has been attributed to each of these algorithms and can be found next to their name inside brackets. The identifier will be used to legend the graphs further in this chapter.

	Maximum flow	Minimum cost flow
Randomized	randomized shortest paths with capacity constraints (capacity)	constrained hitting bag-of-paths (hitting), constrained non-hitting bag-of-paths (nonhitting)
Optimal	augmenting path (generic), FIFO preflow-push (preflow), GLOP (glop), CVXOPT (cvx)	cycle-canceling (cycle), successive shortest path (successive), primal dual (primal), out-of-kilter (kilter), GLOP (glop), CVXOPT (cvx)

6.1.3 Datasets

The datasets used in the comparison are listed in the following table. It contains the number of nodes and the number of edges of each network.

Id	Nodes	Edges
1	100	162
2	500	830
3	1000	1639
4	2500	4159
5	5000	8309
6	10000	16687

The networks have been generated with the networkx API [25]. The function $gn\_graph(nodes\_number, seed = x)$ was used. It output a graph with the number of nodes notified. The graph is constructed node by node. First, an initial source node is created. Then, the next node is linked to a previously created node. The node chosen to link the new node is chosen randomly between all the nodes already inside the network. Finally, all the nodes with no child are linked to a sink node.

Each graph has one source and one sink. For minimum cost flow problems, the demand and the supply are fixed to 100 units. The cost of the edge is 5 for all edges. The capacity is fixed to 20 for all edges.

6.1.4 Method of comparison

In a first step, the generation of the 6 networks described in Subsection 6.1.3 is made. Then, each algorithm is executed with each dataset with each attribute that needs to be tested. The optimal algorithms do not have attributes to be tested. The attributes to be tested for the randomized algorithms is θ for the constrained bag-of-paths algorithms and θ, α for the randomized shortest paths with capacity constraints algorithm.

For each algorithm, the current time is stored in a variable before the execution. The algorithms are executed 100, 10, or one time depending on their speed for better precision (the number of times each algorithm is executed can be found inside the Github repository). When the algorithm finishes, the execution time is computed by subtracting the time after the execution from the time stored before the execution. This number is divided by the number of executions. The execution time is then:

$$time = \frac{(\text{time after executions} - \text{time before executions})}{\text{number of executions}}$$

All the times are stored inside lists and then used to create plots.

The computation was made on a personal computer made of a processor Intel(R) Core(TM) i5-9300H 2.40GHz and 8GB of ram. The OS was Windows 10.

6.2 Minimum cost flow algorithms comparison

In this section, the computation time comparison of the minimum cost flow algorithms giving optimal solution is shown and discussed. Then, the computation time of the hitting and non-hitting constrained bag-of-paths will be analyzed by doing variations of θ.

6.2.1 Time comparison with optimal solution

You can see the comparison of minimum cost flow algorithms in Figure 6.1. A logarithm scale on the y-axis has been added to better visualize the behavior of

each algorithm. Each of them has been tested with all the networks except the CVX linear programming algorithm with networks bigger than 1000 nodes. It is due to the number of constraints that give an unacceptable matrix rank. The constrained bag-of-paths algorithms were set up to $\theta = 10$.

In this figure, we can see 3 groups of algorithms. The fastest one is composed of glop and kilter. The slowest one is composed of cvx and successive. The one in-between is composed of primal, cycle, hitting, and nonhitting.

In these datasets, kilter and glop crush the other algorithms. They compute the solution of the biggest network in about one second. Kilter is fast because it uses a breadth-first search approach which is efficient in these types of networks. However, glop is not fair in this competition as it uses a C++ library that is optimized compared to the Python language.

Hitting is faster than nonhitting. It is understandable as nonhitting will try to compute wider paths. Nevertheless, they run in an acceptable time compared to the global run of algorithms. In addition, they are faster than linear programming (cvx) as shown in [15] and verified in this work.

Successive is not competitive in these datasets. This can be explained by the fact that the algorithm relies on naive mathematical properties. Primal is faster because it pushes all the flow directly instead of multiple times like Successive. Finally, Cycle is the fourth fastest algorithm of the list and it is because the number of cycle inside the networks are low. It takes advantage of the datasets configuration as searching a cycle is an expensive operation.

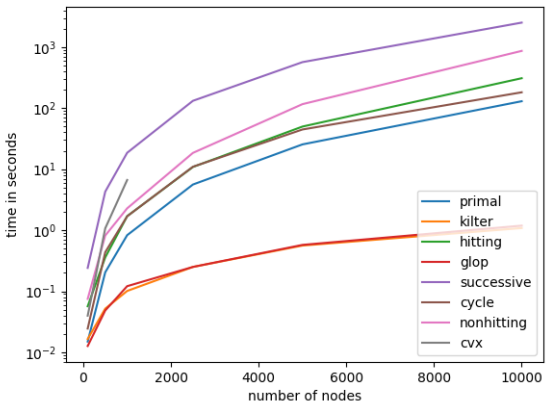

Fig. 6.1: Computation time of minimum cost flow algorithms on networks composed of different number of nodes. The algorithms compared are the cycle-canceling (cycle), successive shortest path (successive), primal-dual (primal), out-of-kilter (kilter), hitting bag-of-path (hitting), non-hitting bag-of-path (nonhitting) and the linear programming algorithms (cvx and glop).

6.2.2 Time comparison by doing variation of θ

In this subsection, we wanted to see if the variation of θ with the constrained bag-of-paths algorithms was leading to different execution times. The execution time of hitting can be found in Figure 6.2 and nonhitting in Figure 6.3.

We can see that the variation of θ does not have an impact on the performance of the algorithms.

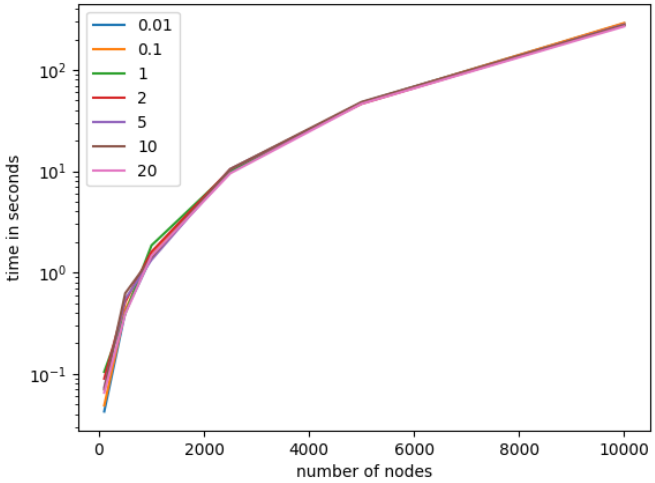

Fig. 6.2: Computation time of the hitting constrained bag-of-paths algorithm on networks composed of different number of nodes. The values of θ used are 0.01, 0.1, 1, 2, 5, 10 and 20.

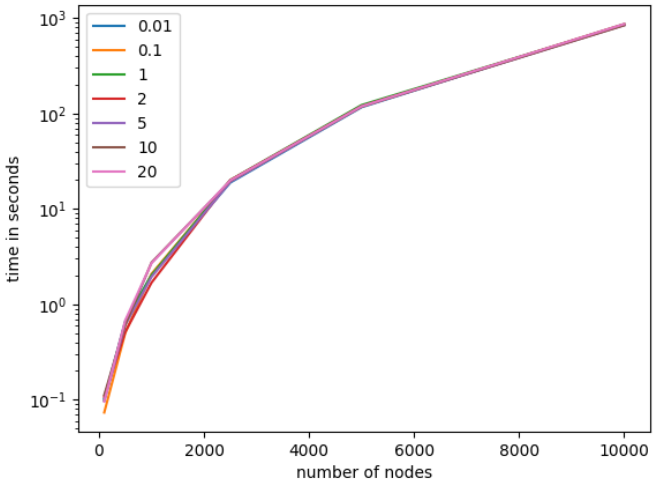

Fig. 6.3: Computation time of the non-hitting constrained bag-of-paths algorithm on networks composed of different number of nodes. The values of θ used are 0.01, 0.1, 1, 2, 5, 10 and 20.

6.3 Maximum flow algorithms comparison

In this section, the computation time comparison of the maximum flow algorithms giving optimal solution is shown and discussed. Then, the computation time of the randomized shortest path with capacity constraints will be analyzed by doing variations of θ and α.

6.3.1 Time comparison with optimal solution

You can see the comparison of the maximum flow algorithms in Figure 6.4. A logarithm scale on the y-axis has been added to better visualize the behavior of each algorithm. As before, each of them has been tested with all the networks except the CVX linear programming algorithm with networks bigger than 1000 nodes. It is due to the number of constraints that give an unacceptable matrix rank. The randomized shortest path with capacity constraints was set up to $\theta = 10$ and $\alpha = \frac{1}{\theta}$.

The glop and generic algorithms are the fastest. As before, glop is not really in the competition as it uses a C++ API that is time optimized compare to Python. However, we can see that generic is faster than glop. How is this possible? After a closer look at the code, we can tell they execute in the same amount of time. Glop has to transform the network into a matrix representing the constrained of the problems and the equations to optimize. This operation takes several milliseconds that makes the difference. On the other hand, generic does not need to transform the network to compute the solution. In addition, generic uses breadth-first search which takes advantage of the shape of the datasets. Indeed, there is a high probability that the shortest path from source to sink is a small path so doing a breadth-first search is the best searching in this context.

Preflow and CVX do not perform well on these datasets. Preflow push tries to push labels to nodes that are not on the best path to arrive at the sink. On the other hand, CVX reachs its limit with the number of constraints that it can manage.

Capacity is the less competitive algorithm of this work. It is due to the naive gradient approach used in the algorithm. Nevertheless, it is understandable because this algorithm is not meant to be used to compute optimal solutions.

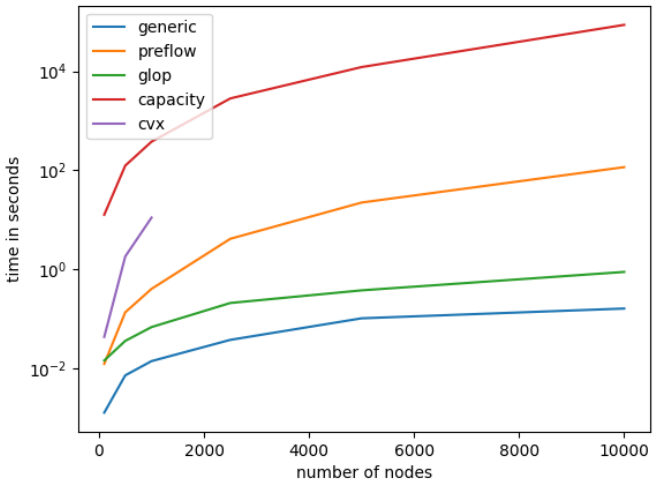

Fig. 6.4: Computation time of maximum flow algorithms on networks composed of different number of nodes. The algorithms compared are the augmenting path (generic), FIFO preflow-push (preflow), randomized shortest paths with capacity constraints (capacity) and the linear programming algorithms (cvx and glop).

6.3.2 Time comparison by doing variation of θ and α

In this subsection, we wanted to see if the variation of θ and α with the randomized shortest path with capacity constraints algorithms was leading to different execution times. Figure 6.5, 6.6, 6.7 represent the computation time with 3 different α values each of them testing different θ values.

We can see that the variation of θ does not have an impact on the performance of the algorithm. On the other hand, if we compare the average computation time for each alpha value, we obtain an interesting result as shown in Figure 6.8. We can observe that when alpha is low, it will take more time to compute. When alpha is high, it will be faster. However, these time differences are really small and it doesn't improve a lot the speed of the algorithm.

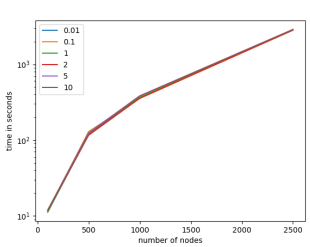

Fig. 6.5: Time with α set to $\frac{1}{\theta}$.

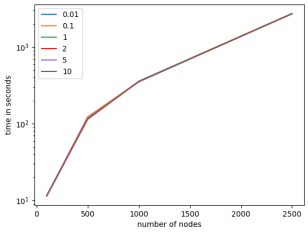

Fig. 6.6: Time with α set to 0.0000001.

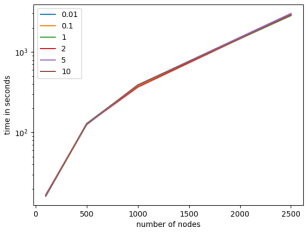

Fig. 6.7: Time with α set to 10000.

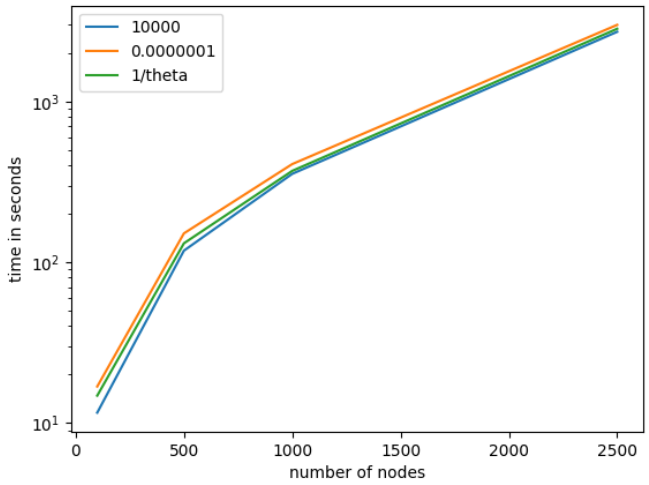

Fig. 6.8: Time obtained with different α values.

6.4 Conclusion of the comparison

As shown in this chapter, algorithms performing breadth-first search like out-of-kilter and augmenting path are competitive in these datasets because the networks have a small shortest path between the source and the sink.

Glop crush all the other algorithms due to his C++ API optimization. It is not fair but it shows that these problems can be computed fast with an optimized procedure.

Hitting and non-hitting constraints bag-of-paths are performing well on average. The variation of θ does not change their computation time. They aren't the fastest but these algorithms are not meant to compute optimal solutions.

Randomized shortest path with capacity constraints is not competitive. It is due to a naive gradient procedure that could certainly be optimized. The variation of θ does not change the computation time. However, increasing α can increase the computation time of the algorithm but it stays still not competitive. However, it is not meant to compute optimal solutions.

7 Conclusion

The companies of today are competitive and need to be efficient in all their actions to stay alive. They have to optimize all their resource to create the best good or service at the best cost. A part of the problems they face are called flow problems [1, 16]. They can be of two types: maximum flow or minimum cost flow problems. The first one corresponds to the maximum flow possible inside a network with a maximum capacity of flow going through each edge. The second one corresponds to the cost-less flow inside a network considering that each unit of flow through an edge has a cost.

Sometimes randomized optimality is necessary to fill other needs [4, 15, 21, 22]. It is a maximum flow or minimum cost flow solution with an error rate that tends to the optimal solution. The error rate is chosen by a variable named the temperature.

In this master thesis, the computation time of all implemented algorithms has been compared based on different network sizes. The algorithms were separated into the maximum flow and minimum cost flow algorithm. Then, θ of the randomized algorithms has been manipulated to see if the computation time was changing based on the variation. Finally, the attribute α from the randomized shortest paths with capacity constraints algorithm has also been tuned to see if the computation time was changing based on the variation.

First, it has been shown that the fastest algorithms were the ones taking advantage of the network shapes like out-of-kilter and augmenting path algorithm. Indeed, breadth-first search is the key to speed up the process with the datasets used because the shortest path between the source and the sink is small. In addition, the Google linear programming solver was also really competitive but it is not fair as it uses C++ API which is optimized instead of Python. It still shows that these problems can be solved efficiently.

Then, we have shown that the constrained bag-of-paths was performing in an acceptable computation time. The hitting algorithm was faster than the non-hitting algorithm because the non-hitting algorithm computes wider paths. The θ variation was not performing any speed improvement on these algorithms.

Finally, the randomized shortest paths with capacity constraints algorithm is not competitive. It is due to the naive gradient approach. However, it is not meant to compute optimal solutions. The θ variation was not performing any speed improvement. The variation of α can improve the computation time but the algorithm stays not competitive.

7.1 Limitations

First, the hitting bag-of-paths and the non-hitting bag-of-paths algorithms do not take into account the capacity of networks. For consistent reasons, the standard minimum cost flow algorithms have also been forced not to use the capacity in their computation. An improvement could be to implement the concept of capacity inside the hitting bag-of-path and the non-hitting bag-of-path. This could lead to more realistic results.

Finally, Python isn't the most time competitive programming language. More optimized language could be used to decrease the computation time of the different algorithms. This could allow to test wider networks and to have a more realistic view of the performance that could be obtained on such flow problems.

7.2 Further work

First, the number of algorithms implemented in this work is small next to the number of algorithms that have been developed to solve such problems. Implementing more algorithms could be useful to compare the performance of the randomized algorithms.

Then, the datasets used in this work are static and always of the same shape per number of nodes. Improving the diversity and the size of networks could lead to more realistic results.

Finally, he randomized shortest paths with capacity constraints algorithm relies on a naive gradient method that could be optimized. This new algorithm would certainly be more competitive.

Bibliography

[1]Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice hall, 1993.

[2]Martin Andersen, Joachim Dahl, and Vandenberghe Lieven. *CVXOPT*. URL: `https://cvxopt.org/`. (accessed: 06.08.2021).

[3]Dimitri P Bertsekas. "Nonlinear programming". In: *Journal of the Operational Research Society* 48.3 (1997), pp. 334–334.

[4]Sylvain Courtain, Pierre Leleux, Ilkka Kivimäki, Guillaume Guex, and Marco Saerens. "Randomized shortest paths with net flows and capacity constraints". In: *Information Sciences* 556 (2021), pp. 341–360.

[5]Alan Dolan and Joan M Aldous. *Networks and algorithms: an introductory approach*. John Wiley & Sons Incorporated, 1993.

[6]Michael Engquist. "A successive shortest path algorithm for the assignment problem". In: *INFOR: Information Systems and Operational Research* 20.4 (1982), pp. 370–384.

[7]Shimon Even, Alon Itai, and Adi Shamir. "On the complexity of time table and multi-commodity flow problems". In: *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*. IEEE. 1975, pp. 184–193.

[8]Lester Randolph Ford and Delbert R Fulkerson. "A simple algorithm for finding maximal network flows and an application to the Hitchcock problem". In: *Canadian journal of Mathematics* 9 (1957), pp. 210–218.

[9]Lester Randolph Ford and Delbert R Fulkerson. "Constructing maximal dynamic flows from static flows". In: *Operations research* 6.3 (1958), pp. 419–433.

[10]Kevin Françoisse, Ilkka Kivimäki, Amin Mantrach, Fabrice Rossi, and Marco Saerens. "A bag-of-paths framework for network data analysis". In: *Neural Networks* 90 (2017), pp. 90–111.

[11]Delbert R Fulkerson. "An out-of-kilter method for minimal-cost flow problems". In: *Journal of the Society for Industrial and Applied Mathematics* 9.1 (1961), pp. 18–27.

[12]Pietro Gianesello, Dmitry Ivanov, and Daria Battini. "Closed-loop supply chain simulation with disruption considerations: A case-study on Tesla". In: *International Journal of Inventory Research* 4.4 (2017), pp. 257–280.

[13]Andrew V Goldberg and Robert E Tarjan. "A new approach to the maximum-flow problem". In: *Journal of the ACM (JACM)* 35.4 (1988), pp. 921–940.

[14] Google. *The Glop Linear Solver*. URL: https://developers.google.com/optimization/lp/glop. (accessed: 31.07.2021).

[15] Guillaume Guex, Ilkka Kivimäki, and Marco Saerens. "Randomized optimal transport on a graph: framework and new distance measures". In: *Network Science* 7.1 (2019), pp. 88–122.

[16] Frederick S Hillier. *Introduction to operations research*. Tata McGraw-Hill Education, 2012.

[17] Ilkka Kivimäki, Masashi Shimbo, and Marco Saerens. "Developments in the theory of randomized shortest paths with a comparison of graph node distances". In: *Physica A: Statistical Mechanics and its Applications* 393 (2014), pp. 600–616.

[18] Vladimir Kolmogorov, Yuri Boykov, and Carsten Rother. "Applications of parametric maxflow in computer vision". In: *2007 IEEE 11th International Conference on Computer Vision*. IEEE. 2007, pp. 1–8.

[19] Bernhard Korte and Jens Vygen. *Combinatorial optimization: theory and algorithms*. Springer Science & Business Media, 2009.

[20] Amin Mantrach, Nicolas Van Zeebroeck, Pascal Francq, et al. "Semi-supervised classification and betweenness computation on large, sparse, directed graphs". In: *Pattern recognition* 44.6 (2011), pp. 1212–1224.

[21] Manuela Panzacchi, Bram Van Moorter, Olav Strand, et al. "Predicting the continuum between corridors and barriers to animal movements using Step Selection Functions and Randomized Shortest Paths". In: *Journal of Animal Ecology* 85.1 (2016), pp. 32–42.

[22] Marco Saerens, Youssef Achbany, Francois Fouss, and Luh Yen. "Randomized shortest-path problems: Two related models". In: *Neural Computation* 21.8 (2009), pp. 2363–2404.

[23] Palaniswamy T Sokkalingam, Ravindra K Ahuja, and James B Orlin. "New polynomial-time cycle-canceling algorithms for minimum-cost flows". In: *Networks: An International Journal* 36.1 (2000), pp. 53–63.

[24] Ted H Szymanski. "Max-flow min-cost routing in a future-Internet with improved QoS guarantees". In: *IEEE transactions on communications* 61.4 (2013), pp. 1485–1497.

[25] Networkx team. *networkx*. URL: https://networkx.org/. (accessed: 24.05.2021).

[26] Cédric Villani. *Optimal transport: old and new*. Vol. 338. Springer, 2009.

[27] Cédric Villani. *Topics in optimal transportation*. 58. American Mathematical Soc., 2003.

[28] Luh Yen, Marco Saerens, Amin Mantrach, and Masashi Shimbo. "A family of dissimilarity measures between nodes generalizing both the shortest-path and the commute-time distances". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 785–793.

[29] Jing Yuan, Egil Bae, and Xue-Cheng Tai. "A study on continuous max-flow and min-cut approaches". In: *2010 ieee computer society conference on computer vision and pattern recognition*. IEEE. 2010, pp. 2217–2224.

[30] Zealint. *Minimum cost flow algorithms*. URL: https://www.topcoder.com/thrive/articles/Minimum%5C%20Cost%5C%20Flow%5C%20Part%5C%20Two:%5C%20Algorithms. (accessed: 02.05.2021).

Abstract :

The societies of today are ruled to produce optimized procedures to be competitive. There is always a moment of need to optimize flow problems inside or outside the company. It can be of several sorts like the transportation of goods from factories to shops or the scheduling of workers to tasks. A panel of algorithms has been developed to solve these problems and always gives the optimal solution. Then, new algorithms based on the bag-of-paths framework have been developed with a new solution property. Instead of having an optimal result, the output is more or less randomized depending on the degree of randomness that is wanted. It is interesting in many sectors like ecology to predict geographic animal behavior. This master thesis consists of the comparison of standard optimal flow algorithms and new algorithms developed in the bag-of-paths framework.

Résumé :

Les sociétés d'aujourd'hui sont tenues de produire des procédures optimisées pour être compétitives. Il y a toujours un moment où il est nécessaire d'optimiser les problèmes de flux à l'intérieur ou à l'extérieur de l'entreprise. Il peut s'agir de plusieurs types de problèmes comme le transport de marchandises des usines aux magasins ou l'ordonnancement des travailleurs aux tâches. Un panel d'algorithmes a été développé pour résoudre ces problèmes et donne toujours la solution optimale. Ensuite, de nouveaux algorithmes basés sur la structure logiciel du bag-of-paths ont été développés avec une nouvelle propriété de solution. Au lieu d'avoir un résultat optimal, la sortie est plus ou moins aléatoire en fonction du degré d'aléa souhaité. Il est intéressant dans de nombreux secteurs comme dans l'écologie pour prédire le comportement géographique des animaux. Cette thèse de master consiste en la comparaison d'algorithmes standards de flux optimaux et de nouveaux algorithmes développés dans la structure logiciel du bag-of-paths.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Louvain School of Management
Place des Doyens, 1 bte L2.01.01, 1348 Louvain-la-Neuve
Boulevard Emile Devreux 6, 6000 Charleroi, Belgique
Chaussée de Binche 151, 7000 Mons, Belgique
www.uclouvain.be/lsm