Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 31 No. 4s (2024)

Innovations in Transport Optimization: Topological Graphical Methodologies and Computational Strategies

Dr. E. Kungumaraj[1*], C.Ruby Sharmila[2], Deepak Umrao Sarwe[3], N. Preethi[4], Mrs.T.Jayapriya[5], Dr. Sanjay Sharma [6], Dr Mithilesh Deo Pandey[7]

[1]Associate Professor of Mathematics, Department of Science and Humanities, Nehru Institute of Engineering and Technology, Coimbatore, (corresponding Author) Email ID: kungum99522@gmail.com

[2]Assistant Professor, Department of Mathematics, Tamilnadu College of Engineering, Coimbatore
Email ID: sharmi.ruby2011@gmail.com

[3]Assistant professor, Department of mathematics, University of Mumbai , Kalina, Santacruz East 400098.
Email ID: deepak.sarve@mathematics.mu.ac.in

[4]Assistant Professor, Department of Science and Humanities, Sri Krishna College of Engineering and Technology, Coimbatore. Email ID:preethin@skcet.ac.in

[5]Assistant Professor, Department of Mathematics, M.Kumarasamy College of Engineering, Karur, Tamil Nadu.
Email ID: jayapriyathangavel@gmail.com

[6]Assistant Professor , Department of Mathematics, NIET, NIMS University Rajasthan, Jaipur.
Email id sanjay.sharma1@nimsuniversity.org

[7]Associate Professor, Department of Applied Mathematics , Bhilai Institute of Technology Durg.
Email ID: mithileshpandey1978@gmail.com

Article History:

Received: *18-04-2024*

Revised: *02-06-2024*

Accepted: *20-06-2024*

Abstract:

This article presents a novel approach for solving the transportation problem(TP) by leveraging Topologized Bipartite graphs along with computational techniques. The methodology involves three key steps: the digitization of the transportation problem, the digitization of topological spaces, and the digitization of graphs. The study initially reconditions the TP into a graphical format and then transforms it into a topologized graphical illustration. Using the recommended method, it effectively determines the best cost for shipping goods from supply sockets to destination sockets. This pioneering method underscores the importance of digitalizing the concepts and relationships among topological spaces, transportation issues, and graph theory. Moreover, it paves the way for exploring various solutions to transportation challenges.

Keywords: Topological Graphical methodologies; Computational Strategies.

1. Introduction

In recent times, Mathematics has seen significant applications in two major domains: Topology and Graph Theory. In 2005, Antonie Vella pioneered the integration of topological properties into graph theory. Building on this foundation, Vimala and Kalpana advanced the model, known as the Topologized Bipartite Graph (TBG), applying it to challenges such as Matching and Coloring. One of the most common uses of measurable analysis in addressing commercial problems is optimizing the circulation of goods, frequently referred to as TP. The primary goal is to decrease delivery costs while ensuring that the demands of respective destination are met, and delivery locations operate within their capacity. To tackle a transportation problem effectively, key decision parameters like supply, demand,

and unit transportation costs must be defined with precision. Chanas et al. [2] introduced an optimal solution for transportation problems (TP) in 1996. Kadhim et al. [3] developed the Modified Kruskal's Algorithm in 2015. Kaufmann [4] presented a graphical approach for TP in 2015. Koopsman [5] proposed the utilization of TP in 1949. Mesbahuddin Ahmed et al. [6] offered a novel methodology to solving TP. Santhi et al. [7, 8] and Shungani Poonam et al. [9] presented solutions for TP in a fuzzy environment in 2016. While many researchers have proposed various methods to find optimal solutions for TP, there has been limited exploration of topological perspectives in this area. Therefore, this article introduces a novel topological approach to achieving optimal solutions for transportation problems, building on the innovative approaches of the Topologized Graph by Antoine Vella [1] in 2005 and Vimala et al. [10, 11] in 2017. The Python program presented in this article implements the topologized graphical method, leveraging graph theory and digitalization techniques. This approach offers a versatile solution applicable to a wide range of scenarios, including transportation and network flow challenges. By converting the problem into a topologized graphical (TG) representation, users can efficiently discover optimal solutions. This tool proves invaluable in fields like logistics, supply chain management, and network design. This article comprises the following sections: Introduction, Preliminaries, Proposed Algorithm, Numerical Illustration with Computation Techniques, Conclusion and References.

2. Preliminaries

Numerous methods have been developed to achieve the smallest transportation schedules. It is essential to recall the basic definitions of transportation problem, bipartite graph, and topological spaces, in addition to introducing the concept of the topologized graph.

Definition: Topologized graph [1]

A topologized graph (TG) is comprising a topology(τ) of G also satisfies the following two conditions

(i) Every one-point set is open set or the closed set in the collection of subsets of G.

(ii) For all g ∈ G such that $|\partial(g)| \leq 2$, $\partial(g)$ is symbolized for the boundary of a vertex g.

At this point the topology is well-defined on the graph, meanwhile the space G is the union of vertices and edges.

3. **Computational Techniques of Topologized Graph Method**

Transportation problems are normally resolved by means of methods such as the MODI method, stepping stone method, and zero-point method, among others, which are all numerical approaches. In this study, we familiarize a novel technique for solving TP from a topological perspective consuming computational techniques. Initially, we translate the standard transportation table into a graphical depiction, then transform it into a TG representation. This innovative approach marks a significant advancement in finding solutions to transportation problems. Our proposed method is simpler than existing methods and yields an optimal solution with lower costs compared to traditional approaches.

3.1 **Recommended Algorithm**

The projected algorithm comprises the succeeding steps:

Step 1: Build a balanced transportation table ($\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$) for the given TP.

Step 2: If it is an Unbalanced TP, modify into a well-adjusted TP by adding row or column with suitable supply or demand quantity with zero cost.

Step 3: Represent the TP in a diagram and consider available & requirement places as vertices s_i & d_j and connecting lines are denoted for unit arrangement cost c_{ij} from i^{th} available place to j^{th} requirement place.

Step 4: Recognize the foremost two minimum arrangement cost of each row & column then modify the graphical depiction with the selected nodes.

Step 5: Create a topological space consisting of vertices $S_i, D_j, (i, j \in N, N\ is\ the\ natural\ numbers)$ and connecting lines e_{ij} of the improved graph. If the improved graph obliges the two necessary conditions of a TG compute from the python coding, continue to the succeeding step; or else, reschedule the graph to meet these necessary conditions.

Step 6: Categorize the vertex in the TG which consumes the minimum transportation cost and distribute $x_{ij} = \text{minimum}\ \{a_i, b_j\}$

- Sub condition (i): If $\text{minimum}\{a_i, b_j\} = a_i$ then distribute $x_{ij} = a_i$ and skip the i^{th} available place and decrease a_i at requirement place D_j, then proceed to the succeeding step.
- Sub condition (ii): If $\text{minimum}\{a_i, b_j\} = b_j$ then distribute $x_{ij} = b_j$ and skip the j^{th} requirement place and decrease b_j at supply point S_i, then progress to the succeeding step.
- Sub condition (iii): If $\min\{a_i, b_j\} = a_i = b_j$ then distribute $x_{ij} = a_i = b_j$ and skip both available place and requirement place further proceed to the succeeding step.

Step 7: Compute new available and requirement for outstanding quantities at S_i & D_j and recall the procedure in Step 6 until all available and requirement are satisfied.

Step 8: Suppose some numbers at a_i or b_j are not distributed at any available and requirement places identify that specific vertex, include a connecting line and distribute hence to fulfil the residual available and requirement numbers.

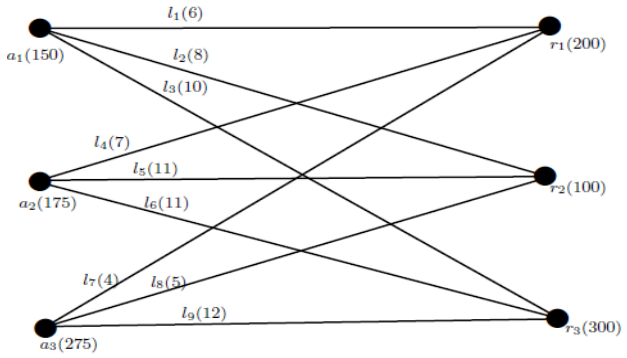
Step 9: If all available and requirement places have two or three distributions, this indicates the best solution for the taken TP. If not, fine-tune the TG and recall Steps 5 through 9.

3.2 **Numerical Example**

	A	B	C	Supply
1	6	8	10	150
2	7	11	11	175
3	4	5	12	275
Demand	200	100	300	

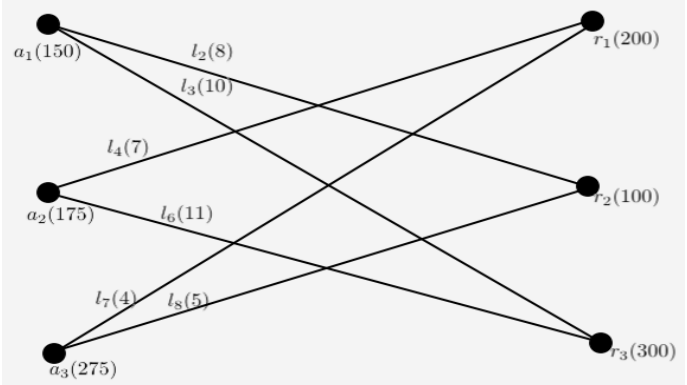
A transport corporation is scheduling to dispense its preserved automobiles to towns A, B, and C. The company's executives have arranged shipping tables that outline the costs associated with transporting from the warehouses (available places) to these cities (requirement places).

The graphical illustration of the TP which follows



The improved demonstration of the above TP using TBG.

```
# Topologized Graph:
from pulp import *
import networkx as nx
from networkx.algorithms import bipartite
# Sources and Destinations: List
Available = ['a1','a2','a3']
Requirement = ['r1','r2','r3']
# In[1]:
G = nx.Graph()
# In[2]:
G.add_nodes_from(['a1','a2','a3'], bipartite=0)
G.add_nodes_from(['r1','r2','r3'],bipartite=1)
# In[3]:
G.add_edges_from([('a1',"r2"),('a1',"r3"), ('a2', "r1"),('a2', "r3"),('a3', "r1"),('a3', "r2")])
# In[4]:
bipartite.is_bipartite(G)
# In[5]:
nx.draw_networkx(G, pos = nx.drawing.layout.bipartite_layout(G, ['a1','a2','a3']), width = 2)
```



In this context $a_1, a_2, a_3, r_1, r_2\ and\ r_3$ represents the vertices of the graph corresponding to the available and demand points. The connecting lines $l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9$ illustrate the connection between the supply and the demand points. We now need to verify if the graph meets the criteria of a topologized graph.

```
 Topology Coding
from typing import Set, FrozenSet
def generate_topology(subsets: Set[Set[str]]) -> Set[Set[FrozenSet[str]]]:
    all_topologies = set()
    for subset in subsets:
        if subset == set():  # Skip the empty set
            continue
        topology = generate_subset_topology(subset, subsets)
        all_topologies.add(frozenset(topology))
    return all_topologies
def generate_subset_topology(subset: Set[str], subsets: Set[Set[str]]) -> Set[FrozenSet[str]]:
    # Generate topology by iteratively combining existing subsets
    topology = set(subset)
    added_subsets = set(subset)
    while True:
        new_subsets = set()
        for A1 in added_subsets:
            for A2 in subsets:
                if A2 == set():  # Skip the empty set
                    continue
                new_subset = frozenset(A1).union(A2)
                if new_subset not in topology:
                    new_subsets.add(new_subset)
        if not new_subsets:
            break
        topology.update(new_subsets)
        added_subsets = new_subsets
    return topology
def is_true_topology(topology: Set[FrozenSet[str]], X: Set[str]) -> bool:
    # Check if the generated topology is a true topology for X
    if not {frozenset(), frozenset(X)} in topology:
        return False
    for subset1 in topology:
        for subset2 in topology:
            if not subset1.intersection(subset2) in topology:
                return False
    return True
# Provide your true topology for X
```

```
X = {{a1}, {a2}, {a3}, {r1}, {r2}, {r3}}
true_topology_subsets = [{φ}, X, {a1}, {a3}, {a1, a3}, {a1, l1, r2, l2, r3, l6, a3, l5, r1, l3, a2},
{r1, l3, a2, l4, r3}, {a1,r1, l3, a2, l4, r3}, {a1, a3, r1, l3, a2, l4, r3}, {a3, r1, l3, a2, l4, r3}]
topologies = generate_topology(true_topology_subsets)
for topology in topologies:
   print("Topology:", topology)
   print("Is True Topology:", is_true_topology(topology, X))
   print()
```

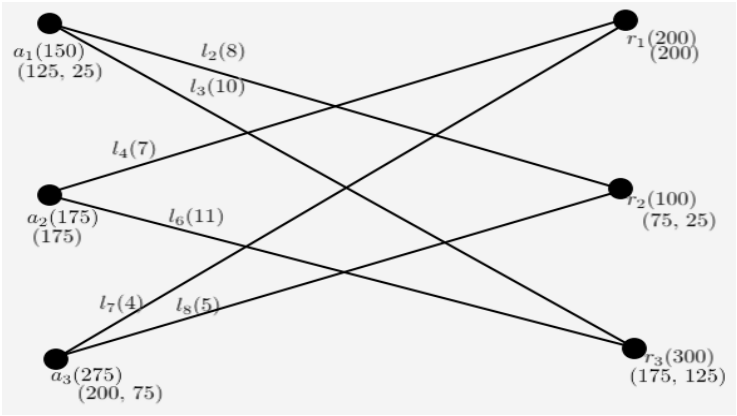
Let X = $\{a_1, a_2, a_3, r_1, r_2, r_3\}$ be a topological space with the topology

$$\tau = \{\{\varphi\}, X, \{a_1\}, \{a_3\}, \{a_1, a_3\}, \{a_1, l_1, r_2, l_2, r_3, l_6, a_3, l_5, r_1, l_3, a_2\}, \{r_1, l_3, a_2,$$

$l_4, r_3\}, \{a_1, r_1, l_3, a_2, l_4, r_3\}, \{a_1, a_3, r_1, l_3, a_2, l_4, r_3\}, \{a_3, r_1, l_3, a_2, l_4, r_3\}\}$.

Subsequently all one-point sets are one closed or the other open in X and each vertex has a limit of 2, the graph encounters the criteria of a topologized graph.

Begin the distribution from a_3, which has the lowest unit transportation cost to r_1 and r_2. Follow the projected process and distribute all quantities accordingly. The final distribution schedule is as follows:




```
 import numpy as np
def get_balanced(available, requirement, costs, penalties = None):
   total_ available = sum(available)
   total_demand = sum(demand)
      if total_ available < total_demand:
      if penalties is None:
         raise Exception(' available less than demand, penalties required')
      new_ available = available + [total_demand - total_ available]
      new_costs = costs + [penalties]
      return new_ available, demand, new_costs
   if total_ available > total_demand:
      new_demand = demand + [total_ available - total_demand]
```

```
        new_costs = costs + [[0 for _ in demand]]
        return available, new_demand, new_costs
    return available, demand, costs
def vogel(available, demand,costs):
    supply_copy = supply.copy()
    demand_copy = demand.copy()
    m=len(available)
    n=len(demand)
    i = 0
    j = 0
    bfs = []
    while len(bfs) < len(supply) + len(demand) - 1:
        s = available _copy[i]
        d = demand_copy[j]
        v = min(s, d)
        available _copy[i] -= v
        demand_copy[j] -= v
        bfs.append(((i, j), v))
        if available _copy[i] == 0 and i < len(available) - 1:
            i += 1
        elif demand_copy[j] == 0 and j < len(demand) - 1:
            j += 1
    cost=0
    bfs_arr = [[0 for i in range(n)] for j in range(m)]
    for item in bfs:
        bfs_arr[item[0][0]][item[0][1]]=item[1]
    print('\n The initial bfs is:\n',bfs_arr)
    for item in bfs:
        cost=cost+costs[item[0][0]][item[0][1]]*item[1]
    print('total bfs cost is: ',cost)
    return bfs
In [6]:
def get_us_and_vs(bfs, costs):
    us = [None] * len(costs)
    vs = [None] * len(costs[0])
    us[0] = 0
    bfs_copy = bfs.copy()
    while len(bfs_copy) > 0:
        for index, bv in enumerate(bfs_copy):
            i, j = bv[0]
            if us[i] is None and vs[j] is None: continue
            cost = costs[i][j]
```

```
            if us[i] is None:
                us[i] = cost - vs[j]
            else:
                vs[j] = cost - us[i]
            bfs_copy.pop(index)
            break
    return us, vs
In [7]:
def get_ws(bfs, costs, us, vs):
    ws = []
    for i, row in enumerate(costs):
        for j, cost in enumerate(row):
            non_basic = all([p[0] != i or p[1] != j for p, v in bfs])
            if non_basic:
                ws.append(((i, j), us[i] + vs[j] - cost))
    return ws
In [8]:
def can_be_improved(ws):
    for p, v in ws:
        if v > 0: return True
    return False
In [9]:
def get_entering_variable_position(ws):
    ws_copy = ws.copy()
    ws_copy.sort(key=lambda w: w[1])
    return ws_copy[-1][0]
In [10]:
def get_possible_next_nodes(loop, not_visited):
    last_node = loop[-1]
    nodes_in_row = [n for n in not_visited if n[0] == last_node[0]]
    nodes_in_column = [n for n in not_visited if n[1] == last_node[1]]
    if len(loop) < 2:
        return nodes_in_row + nodes_in_column
    else:
        prev_node = loop[-2]
        row_move = prev_node[0] == last_node[0]
        if row_move: return nodes_in_column
        return nodes_in_row
In [11]:
def get_loop(bv_positions, ev_position):
    def inner(loop):
        if len(loop) > 3:
            can_be_closed = len(get_possible_next_nodes(loop, [ev_position])) == 1
```

```
            if can_be_closed: return loop
        not_visited = list(set(bv_positions) - set(loop))
        possible_next_nodes = get_possible_next_nodes(loop, not_visited)
        for next_node in possible_next_nodes:
            new_loop = inner(loop + [next_node])
            if new_loop: return new_loop
    return inner([ev_position])
In [12]:
def loop_pivoting(bfs, loop):
    even_cells = loop[0::2]
    odd_cells = loop[1::2]
    get_bv = lambda pos: next(v for p, v in bfs if p == pos)
    leaving_position = sorted(odd_cells, key=get_bv)[0]
    leaving_value = get_bv(leaving_position)
    new_bfs = []
    for p, v in [bv for bv in bfs if bv[0] != leaving_position] + [(loop[0], 0)]:
        if p in even_cells:
            v += leaving_value
        elif p in odd_cells:
            v -= leaving_value
        new_bfs.append((p, v))
    return new_bfs
In [13]:
def transportation_method(supply, demand, costs, penalties = None):
    balanced_supply, balanced_demand, balanced_costs = get_balanced(
        supply, demand, costs
    )
    def inner(bfs):
        us, vs = get_us_and_vs(bfs, balanced_costs)
        ws = get_ws(bfs, balanced_costs, us, vs)
        if can_be_improved(ws):
            ev_position = get_entering_variable_position(ws)
            loop = get_loop([p for p, v in bfs], ev_position)
            return inner(loop_pivoting(bfs, loop))
        return bfs
    basic_variables = inner(vogel(balanced_supply, balanced_demand,costs))
    ans = np.zeros((len(costs), len(costs[0])))
    for (i, j), v in basic_variables:
        ans[i][j] = int(v)
    return ans
In [16]:
def get_total_cost(costs, ans):
```

```
    total_cost = 0
    for i, row in enumerate(costs):
        for j, cost in enumerate(row):
            total_cost += cost * ans[i][j]
    return total_cost
In [20]:
m=int(input("Enter m:"))
n=int(input("Enter n:"))
print("Enter all costs in one line")
entries = list(map(int, input().split()))
costs = np.array(entries).reshape(m, n)
print('\n The costs are:\n',costs)
print('\n')
supply = list(map(int,input("Enter the supply values : ").strip().split()))[:m]
print('\n')
demand = list(map(int,input("Enter the demand values : ").strip().split()))[:n]
print('\n')
ans = transportation_method(supply, demand, costs)
print('\n')
print('\n The optimal solution is:\n',ans)
print('total optimal cost: ', get_total_cost(costs, ans))
```

Output:

```
Enter m:3
Enter n:3
Enter all costs in one line
6 8 10 7 11 11 4 5 12
 The costs are:
 [[ 6.  8. 10.]
 [ 7. 11. 11.]
 [ 4.  5. 12.]]
Enter the supply values : 150 175 275
Enter the demand values : 200 100 300
 The initial bfs is:
 [[150.0, 0, 0], [50.0, 100.0, 25.0], [0, 0, 275.0]]
total bfs cost is:  5925.0
 The optimal solution is:
 [[ 25.   0. 125.]
 [  0.   0. 175.]
 [175. 100.   0.]]
total optimal cost:  4550.0
```

	A	B	C	Supply
1	6	8(25)	10(125)	150
2	7	11	11(175)	175
3	4(200)	5(75)	12	275
Demand	200	100	300	

The table below shows the optimal distribution based on the earlier described procedure:

The total transportation cost $= 8 * 25 + 10 * 125 + 11 * 175 + 4 * 200 + 5 * 75 = 4550.$

4 Conclusion

The research article has introduced a novel and promising approach to solving transportation problems through the application of the Topologized Graphical Method and computational techniques. This study has highlighted the significance of digitalization in optimizing logistics, supply chain management, and network design. The utilization of topological spaces and graph theory has provided a fresh perspective on addressing transportation problems, opening up new avenues for optimization. By transforming these problems into topologized graphical representations, the research has demonstrated the efficiency and versatility of this approach in finding optimal solutions. The Python coding established as portion of this research offers a practical tool for implementing the topologized graphical method, making it accessible and applicable to a wide range of scenarios. Overall, this research contributes valuable insights and a practical solution to address transportation problems using digitalization and mathematical techniques in real-world problem-solving. It is expected that this research will stimulate further exploration and application of topological methods in optimization and computational fields.

References :

[1] Antoine Vella. A, Fundamendally topological perspective on graph theory. Ph.D., Thesis, Waterloo, Ontario, Canada; 2005.

[2] Chanas. S, Kuchta. D, A concept of the optimal solution of the transportation problem with fuzzy cost coefficients, Fuzzy Sets and Systems, ISSN 0165-0114, Volume 82, Issue 3, 1996, Pages 299-305, 1996.

[3] Kadhim B.S. Aljanabi and Anwar Nsaif Jasim, An Approach for solving Transportation Problem Using Modified Kruskal's Algorithm, International Journal of Science and Research, Vol.4, Issue 7, July 2015.

[4] Kaufmann. A., Introduction a la Theorie des sonsensembles flous, Masson Paris, Vol. 1, 41-189, 1973.

[5] Koopsman. C. Tjalling., Utilization of the Transportation System, Econometrica, vol.17, pp. 136-146, July 1949.

[6] Mollah Mesbahuddin Ahmed et.al., A New Approach to Solve Transportation Problems, Open Journal of Optimization, Vol 5, pp 2230, 2016.

[7] Priyadharsini. S, Kungumaraj.E and Santhi. R, An Evaluation of Triangular Neutrosophic PERT Analysis for Real-life Project Time and Cost Estimation, Vol.63, Nuetrosophic Sets and Systems, pp 62-81, 2024.

[8] Santhi. R and Kungumaraj. E, Topological solution of a transportation problem using Topologized graph, IAETSD Journal for Advanced Research in Applied Sciences, Volume VI, Issue VI, pp 30-38, June 2019.

[9] Shugani Poonam, Abbas S. H. and Gupta V.K., Fuzzy Transportation Problem of Triangular Numbers with *αcut* and Ranking Technique, IOSR Journal of Engineering, Vol.2(5),pp 1162-1164 May 2012.

[10] Vimala.S and Kalpana.S, Matching and Coloring in Topologized Bipartite Graph, International Journal of Innovative research in Science, Engineering and Technology Vol.6, Issue 4, pp 7079-7086, Apr 2017.

[11] Vimala. S and Kalpana. S, Topologied Bipartite Graph, Asian Research Journal of Mathematics,ISSN 2456-477X, Vol.4(1), pp 1-10, May 2017.