

עובדת גמר בטיחות

רשותה

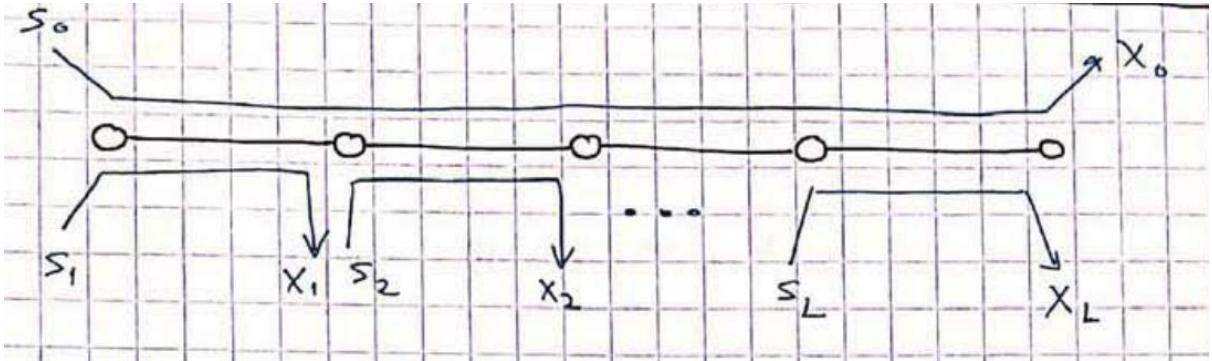
אוניברסיטת בן-גוריון בנגב | הפקולטה למדעי ההנדסה
בית הספר להנדסת חשמל ומחשבים | Ben-Gurion University of the Negev

אביב בן אריה, דולב קיזר, נועם וקנין, מיכאל ליב
אוניברסיטת בן גוריון בנגב | סמסטר חורף 2024

שלב א' פרויקט גמר רשות

שאלה 1 אנליזית :

נתונה רשת תקשורת עם L לינקים ו $L + 1$ משתמשים הנ"ל כאשר קיבול כל לינק הינו $C_l = 1$ JA :



א. הקצאת הקצבים עבור הקритריון תביא למקסימום את סכום התועלת. מההרצאה, עבור קритריון

fairness – אוגיניות מוחדרת פונקציית התועלת הינה $U_r(X_r) = \left(\frac{X_r^{1-\alpha}}{1-\alpha}\right)$, ומקסימום סכום התועלת הינה $\sum_r U_r(X_r) = \arg\max_{X_r \in S} \sum_r U_r(X_r)$ עבור אילוצי קצב חיובי למשתמש ($S \in S$) וכך אילוצי קיבול על לינק ($C_l \leq \sum_{r=0}^L X_r$) וכן כי $C_i = 1$ לכל i שכן נקבע $1 = C_i = \sum_{r=0}^L X_r$. לכן, התקבלה פונקציית מטרה קעורה עם אילוצים אפיניים כלומר אופטימיזציה קעורה לה פונקציית לא-גרנזיין מהצורה הבאה :

$$L(x, \lambda) = \sum_{r=0}^L U_r(X_r) - \sum_{i=1}^L \lambda_i * h_i = \sum_{r=0}^L \left(\frac{X_r^{1-\alpha}}{1-\alpha} \right) - \sum_{i=1}^L \lambda_i (X_0 + X_i - 1) \quad (1)$$

כעת על מנת למצוא את X_i^α נציג בחלוקת למקרים (X_i לכל i שונה מ-0 וכן :

$$X_0^\alpha = \frac{1}{\sum_{i=1}^L \lambda_i} \quad X_0 = \frac{\partial L}{\partial X_0} = (1 - \alpha) X_0^{\frac{1-\alpha}{\alpha}-1} - \sum_{i=1}^L \lambda_i = 0 \quad \text{ונקבל}$$

$$X_i^\alpha = \frac{1}{\lambda_i} \quad X_i = \frac{\partial L}{\partial X_i} = \frac{1}{X_i^\alpha} - \lambda_i = 0 \quad \text{ונקבל}$$

כעת לפי תנאי 3 ב-KKT נגידר רפויון משלים :

$0 \geq \lambda_i \geq 0, \forall i = 1, 2, \dots, L; X_i > 0, \forall i = 1, 2, \dots, L$; $\lambda_i(X_0 + X_i - 1) = 0$.

למינוס אינסוף בו אין אופטימום לפונקציית התועלת שכן $0 < X_i^\alpha$ ולכן מהרפיון מתקיים

$\lambda_1 = \dots = \lambda_L = 1, \dots, L$, $X_1 = \dots = X_L = (X_0 + X_i - 1) = 0, \forall i = 1, 2, \dots, L$ וכן $X_0 = \dots = X_L$ ואמ

ניצב בנסיבות שקיבלנו יתקיים $X_0 = \frac{1}{\alpha \sqrt{L \cdot \lambda_1}}$ וכאשר נציב בתוצרך הרפיון את X_0 נקבל

$$\lambda_1 = \left(\frac{(1+\alpha\sqrt{L})}{\alpha \sqrt{L}} \right)^\alpha \quad \text{ולאחר העברת אגפים והוצאה של } \lambda_1 \text{ לקבלת}$$

$$\forall i = 1, 2, \dots, L; X_i = \sqrt[L]{L} \cdot \frac{1}{1+\sqrt[L]{L}}, \quad X_0 = \frac{1}{1+\sqrt[L]{L}}$$

נציב במשוואות X_0 ונקבל :

шибיאו למקסימום סכום התועלת.

ב. עבור קритריון *proportional fairness* כפי שלמדנו בכיתה, נציב ערך $1 = \alpha$ מה שיביא את X_0

$$L : 1 = \frac{1}{1+L}, \quad X_0 = \frac{1}{1+L}, \quad X_i = L \cdot \frac{1}{1+L}, \quad \forall i = 1, 2, \dots, L$$

ג. בקריטריון מקסימום מינימום נשאייף את $\infty \rightarrow \alpha$ לכן $1 \rightarrow \sqrt[L]{L}$ ונקבל :

$$X_0 = \frac{1}{1+L^0} = \frac{1}{2}, X_i = \frac{1}{2} \quad \forall i = 1, 2, \dots, L$$

בקритריון מינימום השהייה נציג $2 = \alpha$ ונקבל :

$$X_0 = \frac{1}{1+\sqrt{L}}, X_i = \frac{\sqrt{L}}{1+\sqrt{L}} \quad \forall i = 1, 2, \dots, L, U_{r(X_r)} = -\frac{1}{X_r}$$

- ד. לפי הגדרת עקרון מקס'imum מינימום , עבורリンク בו יש שתי משתמשים כמו באיר המצורף, כאשר הקיבול זהה בכלリンク ושווה ל 1, נקבל קצב שווה בראשת בין שני המשתמשים, כלומר $\frac{1}{2} = X_r$ מה שמתכתב עם התוצאות מסעיף ג'.

המשך שלב א' : כתיבת הסימולטור

2.+ .1

בסעיף זה הتابعנו ליצור רשט הכללת N צמתים פרושים אקראיית בתוך איזור גיאוגרפי דו מימדי.

לאורך כל העבודה נציג את רשת התקשרות כגרף. עבדנו עם ספריית **Networkx** שונחה מאד לעובדה עם גרפים. הצמתים פרושים באיזור ברדיוס M ומוחברים עם מרחקם קטן מז.

נציג את הפונקציה שכתבנו:

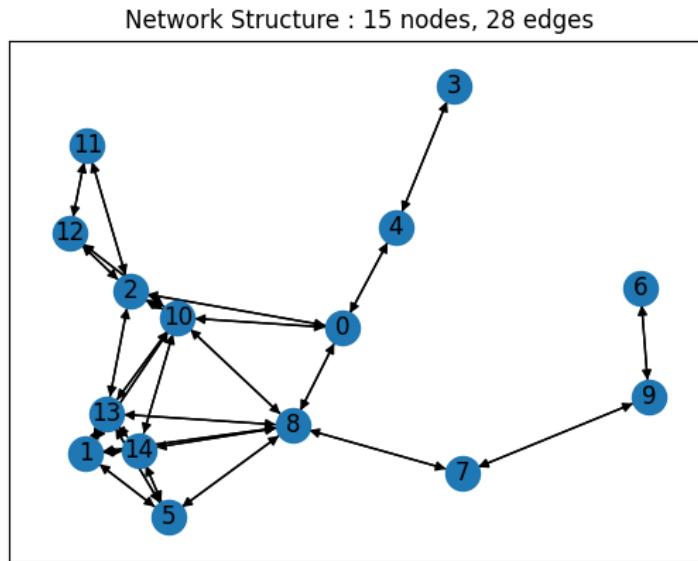
```
def create_random_graph(num_nodes, M, r, seed=123) -> object:
    """
    :param num_nodes: number of nodes in the graph
    :param M: Circular Communication area in radium M
    :param r: nodes within distance r form an edge
    :param seed: seed for random operations
    :return: adjacency matrix and node positions
    """
    random.seed(seed)
    np.random.seed(seed)
    G = nx.Graph()

    # Creates random position for each node inside the circular area with radius M
    positions = {}
    for node in range(num_nodes):
        angle = 2 * math.pi * np.random.random()
        radius = M * math.sqrt(np.random.random()) # Creates a random radius to be inside the circle
        x = radius * math.cos(angle)
        y = radius * math.sin(angle)
        positions[node] = (x, y)
    G.add_nodes_from(positions)
    # ----- #

    # Creates an edge between nodes only if distance is less than r
    for node1 in G.nodes():
        for node2 in G.nodes():
            if node1 != node2 and np.linalg.norm(np.array(positions[node1]) - np.array(positions[node2])) <= r:
                G.add_edge(node1, node2)

    A = nx.to_numpy_array(G)
    A = np.clip(A + A.T, a_min=0, a_max=1)
    pos = np.stack(list(positions.values()), axis=0)
    return A, pos
```

דוגמה לרשת עם 15 צמתים, רדיו רשות 1000 מטר ורדיו חיבור 600 מטר:



3. בסעיף זה הたבקחנו להעניק לכל צומת ברשת הספק שידור. באופן כללי כל הרשת שלנו שמורה כ Class Shmackbl' את מיקומי הצמתים, מטריצת שכנות (Adjacency) מהפו' הקודמת, יחד עם רשימה של flows :

```
class Network:  
    def __init__(self, flows, adjacency_matrix, node_positions, consider_interference=False, seed=37, **kwargs):
```

להספק השידור כתבנו פונקציה שמאפשרת לבחור אם יהיה אחידים או רנדומליים:

```
def __node_transmission_power(self, random_transmission_power=False):  
    """  
    :param random_transmission_power: chose whether Pi is equal for all nodes  
    :return: array of transmission power for all nodes  
    """  
    N = self.num_nodes  
    if random_transmission_power:  
        node_transmission_power = np.random.uniform(low=0.5, high=1, size=N) * np.ones(N)  
    else:  
        node_transmission_power = np.ones(N)  
  
    return node_transmission_power
```

.4

כל מושדר רשיי לשדר מתוך K ערוצים אורתוגונליים: בהגדרת הסימולטור ניתן לבחור K ואז כל מושדר בוחר ערוץ בצורה רנדומלית. בשאלות מאוחרות, אם יהיה צורך לשנות ערוצי שידור כתוצאה ממסלולים ברשות, נעשה שימושים רלוונטיים. מימושו בצורה הבאה:

```
def __transmission_channels(self):
    """
    :return: a list in the size of N nodes. every item is the selected channel for streaming
    """
    np.random.seed(self.seed)

    self.link_to_channel = dict()
    if self.K == 1:
        channel_list = np.zeros(self.num_nodes, dtype=int)
    else:
        channel_list = np.random.randint(low=0, high=self.K - 1, size=self.num_nodes)
    # correlate link with streaming channel, what channel the link uses to stream data
    for u in range(self.num_nodes):
        for v in range(u + 1, self.num_nodes):
            self.link_to_channel[(u, v)] = channel_list[u]
            self.link_to_channel[(v, u)] = channel_list[v]

    return channel_list
```

נקבל רשימה באורך מספר הצמתים ברשות וערך מסהר רנדומלי לערז שידור מתוך K אפשריים.

5.

כל צומת ברשות מושדרת ברוחב פס B_i לפי פונקציה שקבענו. נתנו לכל צומת ערך בין רוחב פס מינימלי-מקסימלי:

```
def __node_bandwidth(self):
    """
    :return: random transmission bandwidth for each node
    """

    # Inserts a random bandwidth for each user
    node_bandwidth = np.random.randint(low=self.kwargs.get("min_capacity", 100),
                                        high=self.kwargs.get("max_capacity", 500),
                                        size=self.num_nodes)

    return node_bandwidth
```

6.

בסעיף זה קבענו את הגברי העורצים $|h|_{i,j}^I$ עבור כלリンク (j,i) ואת הגברי הփרעה את הגברוי העורץ קבענו לפי מודל path loss ומודל כפלי של Rayleigh Fading:

```
def __channel_gains(self):
    """
    :return: |h|_ij for all links (i,j)
    """

    # Channel gains are calculated with path loss, rayleigh fading parameters

    L = self.num_edges // 2
    channel_gains = np.ones(L)
    channel_small_scale_fading = np.random.rayleigh(scale=self.kwargs.get('rayleigh_scale', 1), size=L)
    paths_loss = np.zeros(L)

    for j, edge in enumerate(self.id_to_edge):
        source = edge[0]
        destination = edge[1]
        distance = np.linalg.norm(self.graph_pos[source] - self.graph_pos[destination]) * 1e0  # Meters
        c = 3e8  # speed of light
        Free_Space_Path_Loss = distance + (4 * np.pi / c)
        path_loss = 1 / np.sqrt(Free_Space_Path_Loss)  # H = (1 / sqrt(PL(d)))
        paths_loss[j] = path_loss

    channel_gains = channel_gains * (1 / channel_small_scale_fading) * paths_loss
    return np.abs(channel_gains)
```

את מפת הփרעה בין לינקים חישבנו לפי מרחק בין לינקים + מודל כפלי של הփרעה לפי זווית בין לינקים. כאמור כאשר שני משתמשים משדרים אותן אותות בינם, אוניות צד של האלקטרומגנטי יכולות להפריע לשידור בין לינקים אחרים. קבענו הփרעה מקסימלית כאשר זווית בין לינקים היא $0\backslash 180$ מעלות והփרעה מינימלית כאשר זווית בין לינקים היא 90 מעלות, נציג את הפונ' לחישוב הփרעה על סמך זווית:

```

def interference_factor_based_on_angle(self, edge1, edge2):
    """
    :param edge1: (s1,d1)
    :param edge2: (s2,d2)
    :return: factor based on angle.
    angle between links calculated with dot product
    dot(v,u) = |v||u|cos(theta)
    """

    s1, d1, s2, d2 = edge1[0], edge1[1], edge2[0], edge2[1]
    vec_edge1 = np.array(self.graph_pos[d1] - self.graph_pos[s1])
    vec_edge2 = np.array(self.graph_pos[d2] - self.graph_pos[s2])

    # Calculate the dot product
    dot_product = np.dot(vec_edge1, vec_edge2)

    # Calculate magnitudes
    magnitude_edge1 = np.linalg.norm(vec_edge1)
    magnitude_edge2 = np.linalg.norm(vec_edge2)
    # Calculate the angle in radians
    angle_rad = np.arccos(np.clip(cos_theta, -1.0, a_max: 1.0))

    # Convert the angle to degrees
    angle_deg = np.degrees(angle_rad)

    # as close to the same line = More interference
    if angle_deg > 90:
        angle_deg = np.abs(angle_deg - 180)

    """
    now convert angle into interference_factor.
    low angles cause high interference, close to 90 degrees small interference
    no interference ==> factor = 0.
    high interference ==> factor = 1.0
    """

    # Normalize the angle to the range [0, 1]
    normalized_angle = angle_deg / 90.0

    # Invert the normalized angle to get interference factor (optional)
    interference_factor = 1.0 - normalized_angle

    # Clip the interference factor to ensure it's within [0, 1]
    interference_factor = max(0.0, min(1.0, interference_factor))

    return interference_factor

```

לאחר שכתבנו פונקציה זו, מימושו את הגברי ההפרעה בצורה הבאה:

```
def __channel_interference_gains(self, consider=False):
    """
    :return: an interference gain map, for every link
    calculates the interference gain with all other links
    """
    L = self.num_edges // 2
    channel_interference_gains = np.zeros((L, L))
    if not consider:
        return channel_interference_gains # In some questions, we don't consider interference return zeros
    else:
        for i in range(L):
            for j in range(i + 1, L):
                edge1 = self.id_to_edge[i]
                edge1_pos = self.link_pos[i]
                edge2 = self.id_to_edge[j]
                edge2_pos = self.link_pos[j]
                link_distance = np.linalg.norm(edge1_pos - edge2_pos)
                angle_interference = self.interference_factor_based_on_angle(edge1, edge2)
                channel_interference_gains[i, j] = angle_interference * (1 / np.sqrt(link_distance))
                channel_interference_gains[j, i] = angle_interference * (1 / np.sqrt(link_distance))

    return np.abs(channel_interference_gains) # ** 2
```

בסוף של דבר, יש לנו מפת הגברי הפרעה, השורה ה- i מסמלת את כל הגברי ההפרעה שלリンク –
ו- מכל שאר הלינקים.

7.

בסיוף זה קבענו את קיובלי הלינקים לפי נוסחת **Shannon** :

$$C_{i,j} = B_i \log_2(1 + SINR)$$

כמובן שערוצי ההפרעה נלקחים רק מלינקים פעילים, لكن הפונקציה שכתבנו מקבלת ארגומנט paths כדי לדעת באיזה לינקים נעשים שימוש בשידור בראשת.
בנוסף אנחנו סופרים בכמה מסלולים נעשה שימוש בכלリンク כדי לחלוק את הקיבולות במידת הצורך.
בסיוף זה עוד לא ברור איך מתחילק הקיבול لكن פלט הפונקציה הוא ללא חילוק, בסעיפים מאוחרים בעבודה(כמו 6), ניתנת הוראה איך לבצע את חילוק הקיבול لكن שם נחלק מהשקבילנו פה:

```

def calc_link_capacity(self, paths):
    """
    :param paths: paths given in the network for each flow
    :return: a list with capacity for each link that takes part in some path.
    for links that are not involved with any path I give capacity 0.
    might consider to change
    """

    link_capacities = np.zeros(self.num_edges // 2)

    # Count number of appearances for every link
    # We need to know how many transmissions appear on one link in order to divide capacity
    link_counter = Counter()
    for path in paths:
        # Iterate over consecutive nodes in the path and update the counter
        for i in range(len(path) - 1):
            link = (path[i], path[i + 1])
            link_counter[link] += 1
    # -----
    active_links = list(link_counter.keys())
    active_links_indices = np.array([self.eids[link] for link in active_links])

    for path in paths:
        path_capacities = []
        for i in range(len(path) - 1):
            s, d = path[i], path[i + 1]
            link_interference = (self.channel_interference_gains[:, self.eids[s, d]])[active_links_indices] # all gains
            # link_interference = np.sum(link_interference_gains * self.node_transmission_power) # I_l
            link_interference = np.sum(link_interference)
            transmission_power = self.node_transmission_power[s] * self.channel_gains[self.eids[s, d]] # P_l*h^2 tr
            sinr = transmission_power / (link_interference + 1) # SINR_l assuming noise with unit variance
            link_bandwidth = self.node_bandwidth[s] # Bandwidth of the link
            cap = link_bandwidth * np.log2(1 + sinr)

            # Shannon
            link_capacity = np.minimum(link_bandwidth,
                                        np.maximum(1, np.floor(link_bandwidth * np.log2(1 + sinr)))))

            # If multiple links transmit on same link, divide the capacity

            # link_capacity = link_capacity // link_counter[(s, d)]
            link_capacities[self.eids[s, d]] = link_capacity

        return link_capacities

```

8.

הסימולטור מקבל רשימה של זרמי DATA **Flows** המוגדרים לפי מקור,יעד,כמות חבילות להעבירה.
נראה דוגמה:

```

flows = [{"destination": 3, "source": 0, "packets": 230}, {"destination": 7, "source": 2, "packets": 460},
          {"destination": 8, "source": 1, "packets": 670}]

```

שאלה 2:

בשאלה זו התקשנו להשתמש בתשתיית שלנו לימוש האלגוריתם המבוזר הפרימרי שלמדנו בהרצאה. נזכיר בצעד האלגוריתם:

$$\frac{5/11/19}{\dot{x}_r = \underbrace{k_r(x_r)}_{\text{תוקן }} \left[U_r(x_r) - \sum_{l \in \text{links}} f_l \left(\sum_{s \in \text{links}} x_s \right) \right] \quad \text{for all } r \in S}$$

מחד שולח --> בקר צפיפות מזריך קצב

בקצבים גובאים מתחמונת העלאת קצב
מקערות שלחתומת

המחיר בlienק המחר מתחזקן מיידית דרך הקצבים המשודכנים

אינו נקי מנקזות

- * כי jedem יוט זר אוניברסלי header תלוי בסכום מחירים
- * נמי לא קיימת איסת גאנט דואתי גאנט מוניטר או פיקט header
- * הייז יתיר נזק נזק עזם נזק עזם נזק

כאשר f_l הינה הנגזרת של פונקציה ה- B_l *Barrier* - B_l

במקרה של Alpha-fairness מקבל $U_r(x_r) = \frac{x_r^{1-\alpha}}{1-\alpha} U'_r(x_r) = \frac{1}{x_r^\alpha}$.. את הפונקציה f_l נקבע

להיות: $e^{c:y} = f_l$ כאשר y הינו סכום הקצבים העוברים על link l . c הוא קבוע משתנה.
באלגוריתם המבוזר, בכל איטרציה נבחר משתמש אקראית לבצע עדכו. פלט האלגוריתם הוא סט
קצבים שפותר את בעיית **NUM**. נציג את הפונקציות שכתבנו:

```
def alpha_fairness_utility(rate, alpha=0.5):
    """
    :param rate: allocated rate for some flow
    :param alpha: chosen alpha
    :return: utility of flow with alpha fairness
    """
    utility = (rate ** (1 - alpha)) / (1 - alpha)
    return utility

def Barrier_for_link_l_derivative(y_l, c_l, penalty_coeff):
    """
    :param penalty_coeff: coeff to multiply exponent
    :param y_l: sum rates on link l
    :param c_l: links capacity
    :return: B'(y_l)
    """
    if y_l - c_l <= 0:
        return 0
    else:
        derivative = c_l / (y_l ** 2)

    derivative = np.e ** (penalty_coeff * y_l)
    return derivative

def update_step_gradient_ascent(rate, alpha, learning_rate, path_penalties):
    """
    :param alpha: alpha for alpha fairness
    :param rate: rate for flow in previous iteration
    :param learning_rate: Kr in gradient ascent
    :param path_penalties: np array of penalty derivatives for each link in path
    :return: updated rate for next iteration
    """
    new_rate = rate + learning_rate * (alpha_fairness_utility_derivative(rate, alpha) - np.sum(path_penalties))
    return new_rate
```

```

def primal_distributed_NUM_algorithm(network, link_capacities, paths, penalty_coeff, alpha=0.5, learning_rate=0.1,
                                     max_iters=100):
    """
    :param link_capacities: capacity of every link
    :param penalty_coeff: coefficient to multiply exponent in Barrier calc
    :param network: Our Simulator with information about nodes, edges, interference
    :param alpha: alpha for alpha fairness utility calculation
    :param paths: given paths in the network. we choose randomly
    :param learning_rate: learning rate Kr for gradient ascent
    :param max_iters: number of iterations for the algorithm
    :return: allocated rate for every flow after the algorithm
    """
    # link_capacities = network.calc_link_capacity(paths)
    # link_capacities = np.ones(len(paths))

    # initialize rates for every flow
    # rates = np.random.randint(low=np.min(link_capacities), high=np.max(link_capacities), size=len(paths))
    all_iterations_rates = np.zeros((max_iters, len(paths)))
    # initialize rates for every flow
    rates = np.ones(len(paths))
    all_iterations_rates[0] = rates

    # start Primal Algorithm
    for iteration in range(1, max_iters):

        # Counter that keeps the sum of rates of all active links, all links on same path get flow's rate
        links_rates_counter = Counter()
        for flow_idx, path in enumerate(paths):
            for j in range(len(path) - 1):
                link = (path[j], path[j + 1])
                links_rates_counter[network.eids[link]] += rates[flow_idx]

        # randomly select a flow to update its rate
        selected_flow_index = np.random.choice(range(len(paths)))
        path = paths[selected_flow_index]
        rate = rates[selected_flow_index]
        # calc derivative of penalty func for every link in the path given links capacity and rates sum
        # keep an array for penalties on the path
        path_penalties = np.zeros(len(path) - 1)
        for i in range(len(path) - 1):
            link = (path[i], path[i+1])
            link_index = network.eids[link]
            c_l = link_capacities[link_index]
            y_l = links_rates_counter[link_index] # Sum of all rates on the link
            path_penalties[i] = Barrier_for_link_l_derivative(y_l, c_l, penalty_coeff=penalty_coeff)

        # Gradient ascent step
        rate = update_step_gradient_ascent(rate=rate, alpha=alpha, learning_rate=learning_rate,
                                           path_penalties=path_penalties)
        # update rates vector
        rates[selected_flow_index] = rate
        all_iterations_rates[iteration] = rates

    return rates, all_iterations_rates

```

שאלה 3:

נחזיר על שאלה 2, הפעם עם האלגוריתם הדואלי. נזכיר בצד האלגוריתם של מדנו בכתיבה:

$$\begin{aligned}
 & \text{רעיון עט גממודים (גטן) גדקינו בדעת:} \\
 & \text{(A1) } X_r = (U'_r)^{-1} [q_r] \quad \text{הנכון } q_r = \sum_{l \in r} \lambda_l \quad \text{sum of Lagrange multipliers over links in route } r \\
 & \text{(A2) } \dot{\lambda}_e = h_e (y_e - c_e)^+ \\
 & \quad \downarrow \quad \downarrow \\
 & \quad \frac{\lambda_e(t+1) - \lambda_e(t)}{\Delta} \quad \text{תבונת שטח}
 \end{aligned}$$

$$(y_e - c_e)^+ = \begin{cases} y_e - c_e & \text{if } \lambda_e > 0 \\ \max(y_e - c_e, 0) & \text{if } \lambda_e = 0 \end{cases}$$

גממודים (גטן) נתקן עט דגם פארקיי זילטיגר.

האלגוריתם עובד בצורה מבוזרת כך שבכל פעם נבחר משתמש באקראי, הוא יעדכן את הקצב שלו לפי סכום כופלי לגראנץ במסלול שלו. לאחר מכן נעדכן את כופלי לגראנץ בכל לינק במסלול לפי

Gradient descent בעזרת סכום הקצבים על אותו לינק פחות קיבול הלינק.

פעם נוספת הפלט יהיה סט קצבים לכל המשתמשים.

נצרף את הפונקציה שכתבנו בעמוד הבא:

```

def Dual_distributed_NUM_algorithm(network, link_capacities, paths, alpha=0.5, learning_rate=0.1, max_iters=100):
    """
    :param link_capacities: capacity of every link
    :param network: Our Simulator with information about nodes, edges, interference
    :param alpha: alpha for alpha fairness utility calculation
    :param paths: given paths in the network. we choose randomly
    :param learning_rate: learning rate Kr for gradient ascent
    :param max_iters: number of iterations for the algorithm
    :return: allocated rate for every flow after the algorithm
    """

    # link_capacities = network.calc_link_capacity(paths)
    # link_capacities = np.ones(len(paths))

    # initialize rates for every flow
    # rates = np.random.randint(low=np.min(link_capacities), high=np.max(link_capacities), size=len(paths))
    # for plotting later
    all_iterations_rates = np.zeros((max_iters, len(paths)))

    # initialize rates for every flow
    rates = np.ones(len(paths))
    all_iterations_rates[0] = rates

    # initialize rates for every flow
    rates = np.ones(len(paths))
    all_iterations_rates[0] = rates

    # start Primal Algorithm
    for iteration in range(1, max_iters):

        # Counter that keeps the sum of rates of all active links, all links on same path get flow's rate
        links_rates_counter = Counter()
        for flow_idx, path in enumerate(paths):
            for j in range(len(path) - 1):
                link = (path[j], path[j + 1])
                links_rates_counter[network.eids[link]] += rates[flow_idx]

        # randomly select a flow to update its rate
        selected_flow_index = np.random.choice(range(len(paths)))
        path = paths[selected_flow_index]
        rate = rates[selected_flow_index]
        rates[selected_flow_index] = new_rate
        all_iterations_rates[iteration] = rates

        # Counter that keeps the sum of rates of all active links, all links on same path get flow's rate
        # after rate update
        links_rates_counter = Counter()
        for flow_idx, path in enumerate(paths):
            for j in range(len(path) - 1):
                link = (path[j], path[j + 1])
                links_rates_counter[network.eids[link]] += rates[flow_idx]

        # Update every lagrange multiplier in chosen path according to new rate
        # Step A2
        for i in range(len(path) - 1):
            link = (path[i], path[i+1])
            link_index = network.eids[link]
            c_l = link_capacities[link_index]
            y_l = links_rates_counter[link_index] # Sum of all rates on the link
            lambda_l = lagrange_multipliers[link_index]
            new_lagrange_multiplier = gradient_descent(lambda_l, learning_rate, y_l, c_l)
            lagrange_multipliers[link_index] = new_lagrange_multiplier

    return rates, all_iterations_rates

```

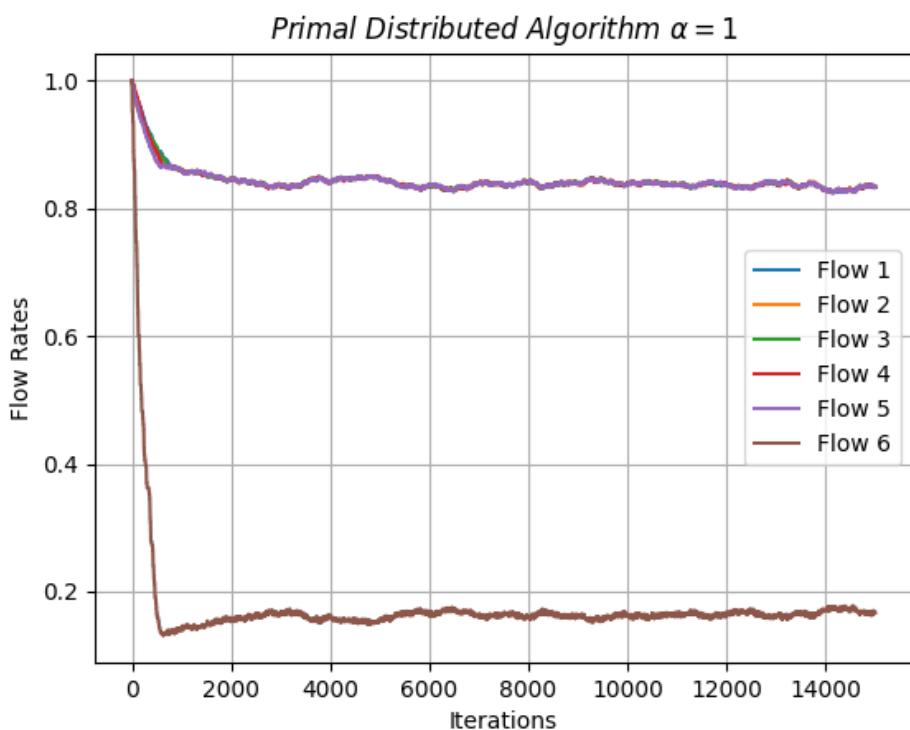
שאלה 4:

כעת נבחן את האלגוריתמים שכתבנו בשאלות 2,3 על הרשות מהשאלה התיאורטית שפתרנו עבור

$$L=5 . \text{ נזכיר שקיבלו } X_0 = \frac{1}{1+\sqrt[5]{L}} , X_i = \frac{\sqrt[5]{L}}{1+\sqrt[5]{L}} \text{ א } i \neq 0 \text{ ו } c_l = 1 \text{ א } c_l = 1 .$$

אלגוריתם פרימלי:

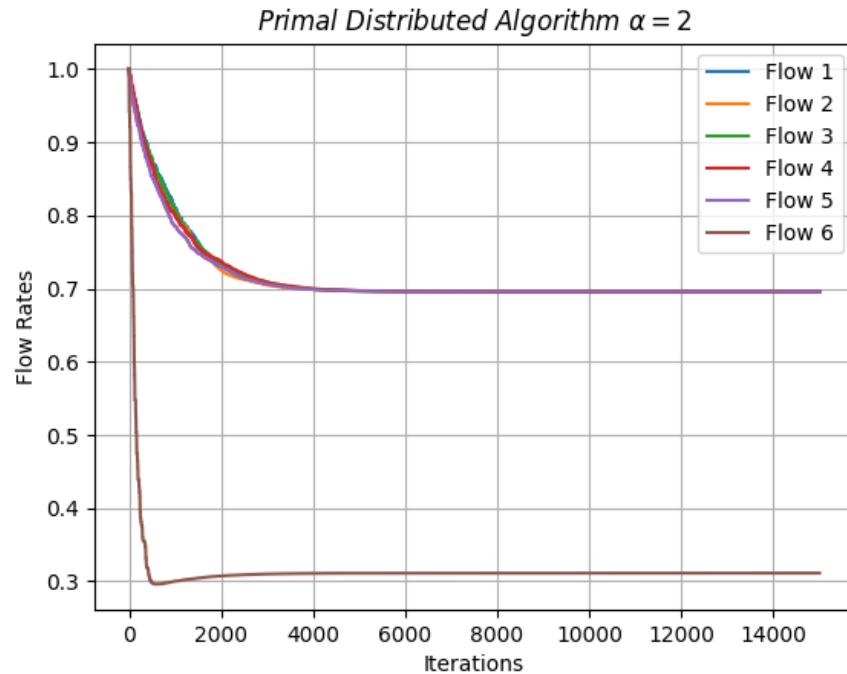
א. ראשית ניקח $\alpha = 1$ ונצפה לקבל $0 \neq i$. $X_0 = \frac{1}{1+\sqrt[5]{5}} = 0.1667$, $X_i = \frac{\sqrt[5]{5}}{1+\sqrt[5]{5}} = 0.833$ ניקח את הקבוע בנגזרת פ' המחיר להיות 0.62 ונרי' ל 15000 איטרציות עם גודל צעד 0.001 קיבל:



ניתן לראות שכאן יש הוכנסות לערכים שקיבלו בתיאוריה.

rates						
0	0.8337622012745045	0.8344194582432348	0.833056950216249	0.8333694040919326	0.8343933704008133	0.16671786127332835

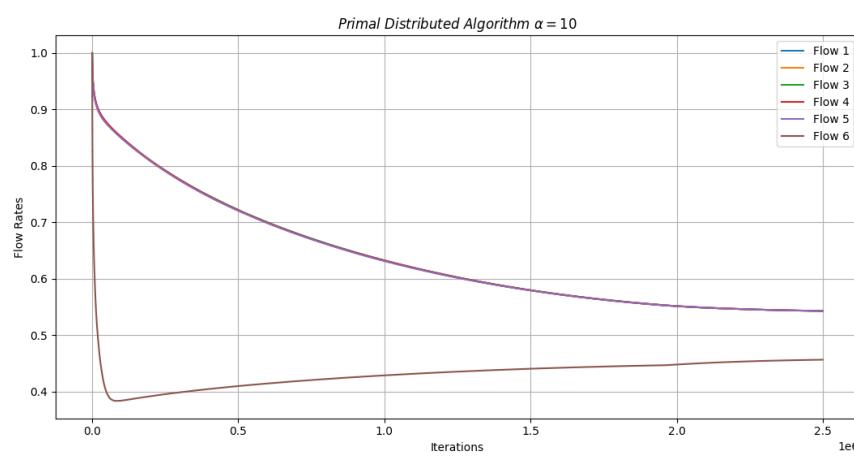
ב'. כעט ניקח $\alpha = 2$. ונצפה לקבל $0 \neq i$.
 $X_0 = \frac{1}{1+\sqrt[2]{5}} = 0.309$, $X_i = \frac{\sqrt[2]{5}}{1+\sqrt[2]{5}} = 0.691$ א
 ניקח את הקבוע בנגזרת פ' המחריר להיות 0.72. נרץ עם גודל עד 0.001 ל-15000 צעדים נקבל:



נראה שכאן קיילנו תוצאות שדומות לתיאוריה

rates					
0	1	2	3	4	5
0.6958936558004383	0.695893655807069	0.6958936567029882	0.6958936560807111	0.6958936560640955	0.31121310105813366

ג'. כעט ניקח $\alpha = 10$. ונצפה לקבל $0 \neq i$ א
 ניקח את מקדם האקספוננט להיות 6 ונרוץ 2500000 צעדים עם גודל עד 0.00000001 נקבל:

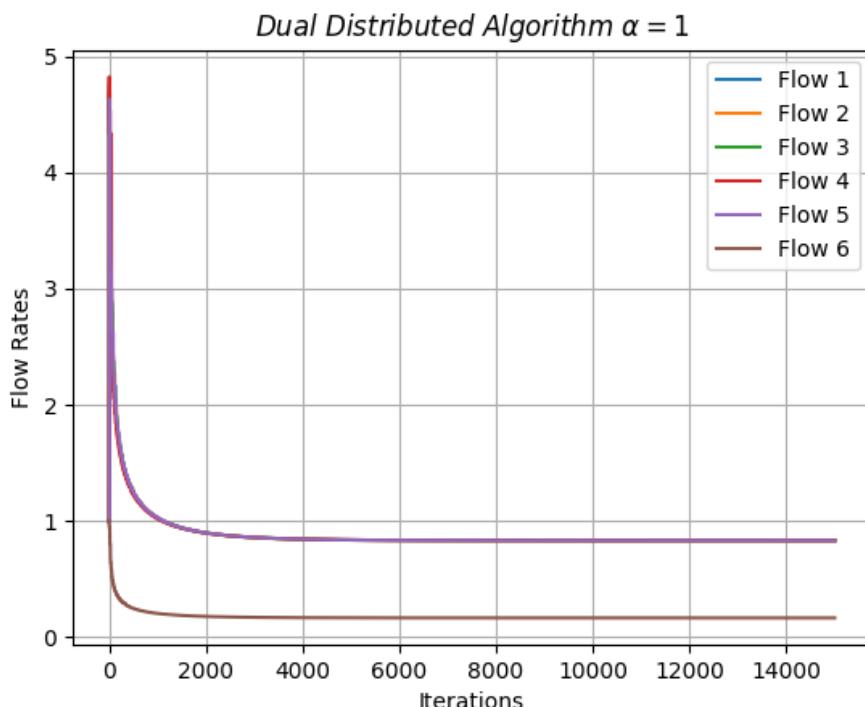


פעם נוספת נראה שאנו מתקנים לערך טוב כמו בתיאוריה.

	0	1	2	3	4	5
0	0.5431649227998282	0.543164405079901	0.5431646274411206	0.5431647072089841	0.543164514687049	0.4568354083028439

אלגוריתם פרימלי

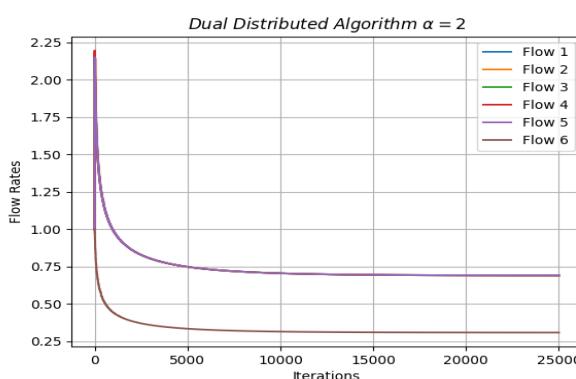
. $X_0 = \frac{1}{1+\sqrt[4]{5}} = 0.1667$, $X_i = \frac{\sqrt[4]{5}}{1+\sqrt[4]{5}} = 0.833$ וצפה לקבל 0 $\neq i$ $\forall i$ נרוץ 15000 איטרציות עם גודל צעד 0.001 ונקבל:



פעם נוספת קיבלנו תוצאות טובות שדומות לתיאוריה.

	0	1	2	3	4	5
0	0.8333343788160198	0.8333344400288023	0.8333344305368762	0.8333342142680576	0.8333344652158925	0.16666688098084206

ב'. כעת ניקח $\alpha = 2$. ונצפה לקביל 0 $\forall i \neq 0$ גודל צעד 0.001: נרץ 25000 איטרציות עם גודל צעד 0.001:

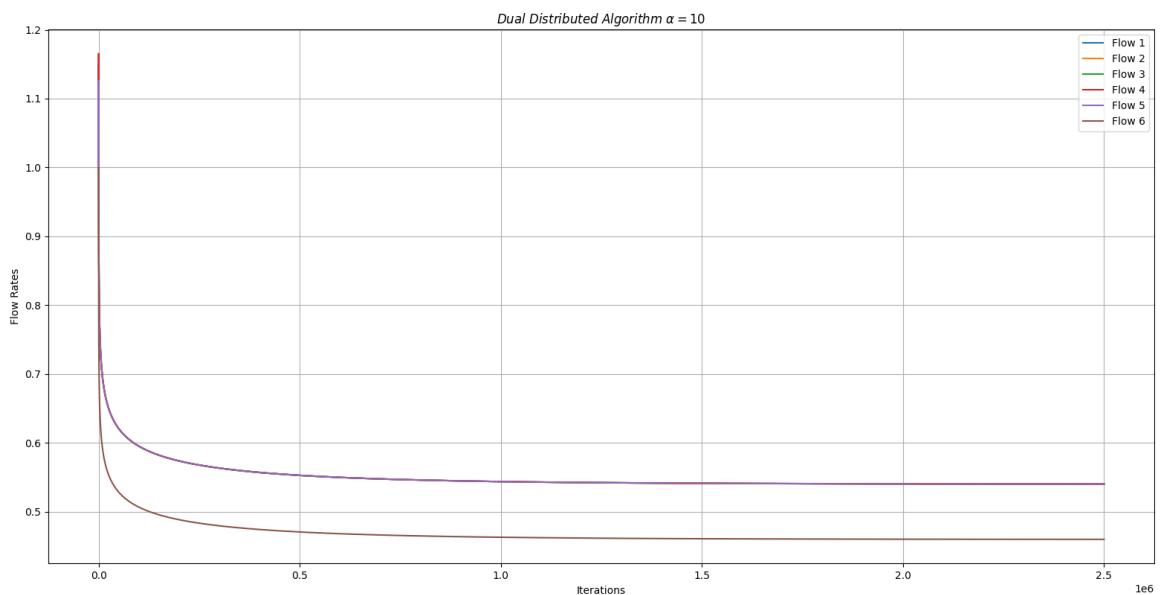


	0	1	2	3	4	5
0	0.6913683778140841	0.6913729611955005	0.6913706996340419	0.6913633038051314	0.691373216292881	0.3091896284061416

פעם נוספת תוצאות טובות וקרובות לתיאוריה.

$$X_0 = \frac{1}{1 + \sqrt[10]{5}} = 0.459 , X_i = \frac{\sqrt[10]{5}}{1 + \sqrt[10]{5}} = 0.54 \text{ } \forall i \neq 0$$

ג'. כעט ניקח $\alpha = 10$. ונרצה לקבל 0.01 ונקבל:
נróż 2500000 עם גודל צעד 0.01 ונקבל:

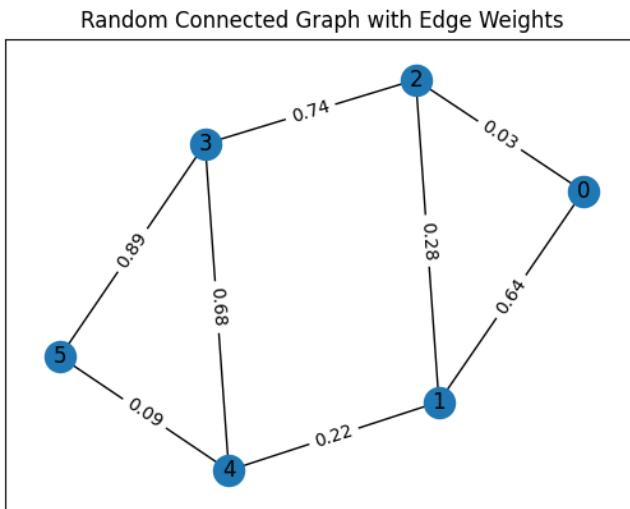


פעם נוספת תוצאות טובות וקרובות לתיאוריה.

	0	1	2	3	4	5
0	0.5402959733265796	0.5402959934671951	0.5402959853162855	0.5402959421260548	0.5402959935380347	0.4599755386880039

שאלה 5:

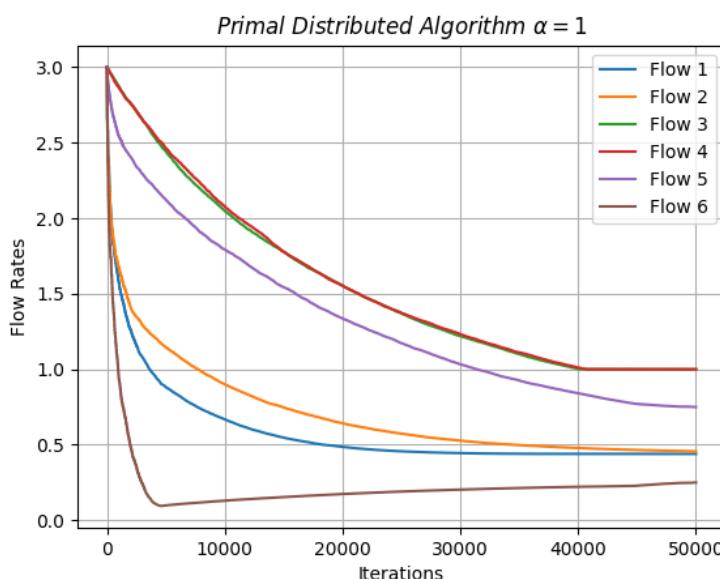
בשאלה זו התקשנו לחזור על שאלות 2, 3 כאשר מסלולים נקבעים לפי אלגוריתם דיקסטרה. לקחנו השראה מ שאלה 4 וקבענו את הרשות להיות בעלת 6 משתמשים עם לינקים בעלי, קיבול יחידה. להלן הרשות יחד עם המשקלות האקראיים בין 0 ל-1:



הרצינו את האלגוריתם לקבלת מסלולים:

```
> paths = {list: 6} [[0, 2, 1], [1, 2], [2, 3], [3, 4], [4, 5], [0, 2, 1, 4, 5]]
```

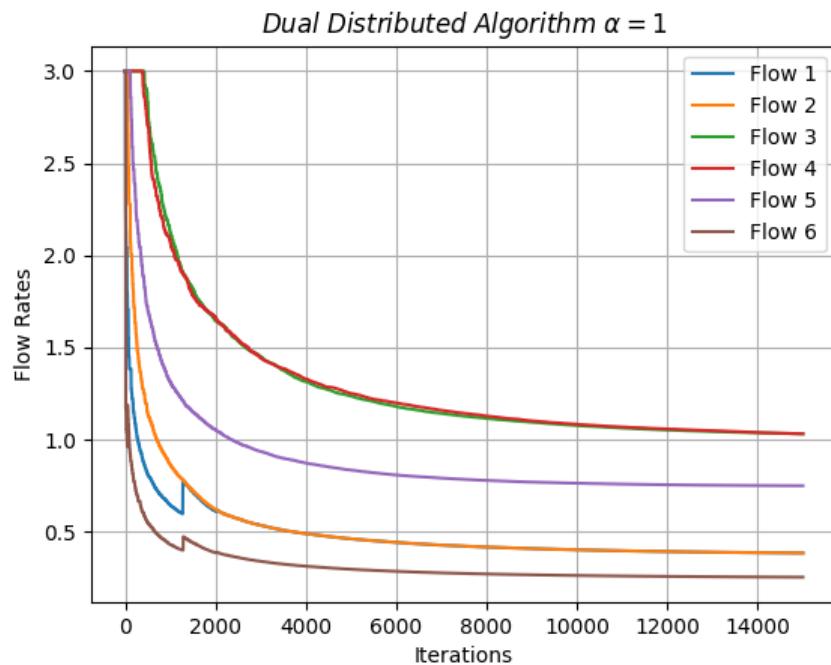
ראשית נריץ האלגוריתם הפרימלי:
בחרנו $\alpha = 1$. הרצינו 50,000 איטרציות עם גודל צעד 0.0001. קיבלנו:



0	0.43978000760516583	0.45682673272066504	0.999921724971162	1.0000634294365596	0.7499366665946114	0.2501364126435196
---	---------------------	---------------------	-------------------	--------------------	--------------------	--------------------

ציפינו לקבל **proportional fairness** ואכן קיבלנו. למסלולים 3,4 קצבים הći גבוהים כי ניתן לראות שמסלולים אחרים לא משתמשים בלינקים הללו لكن זה הגיוני.

כעת הרצנו את האלגוריתם הדואלי. בחרנו $\alpha = 1$. הרצנו 15,000 איטרציות עם גודל צעד 0.001. קיבלנו:



	0	1	2	3	4	5
0	0.3857395741066342	0.3857317417935255	1.0321650709927281	1.0322337203070115	0.7507447550698383	0.2548307207778364

פעם נוספת קיבלנו מה ציפינו, הקצבים מוחולקים באופן אחיד פחת או יותר כך שמשתמשים שעושים שימוש בלינקים שהכי פחות מסלולים עוברים בהם מקבלים קצב גבוה יותר. ניתן לראות שיש שינוי קטן במספרים הסופיים שקיבלנו משנה האלגוריתם. השינוי נראה נובע מטייב הקוד שכתבנו או מתנאי ההתחלה של האלגוריתם. הקוד נציג את הקוד שלנו למציאת מסלולים לפי דיקוסטרא:

```

def Dijkstra(draw_graph=False, seed=42):
    G = nx.Graph()
    G.add_nodes_from([0, 1, 2, 3, 4, 5])
    G.add_edges_from([(0, 1), (0, 2), (1, 2), (1, 4), (2, 3), (3, 4), (3, 5), (4, 5)])
    A = np.array(nx.to_numpy_array(G))
    A = np.clip(A + A.T, a_min=0, a_max=1)
    pos = nx.spring_layout(G)
    pos = np.stack(list(pos.values()), axis=0)

    random.seed(seed)
    for edge in G.edges():
        G.edges[edge]['weight'] = round(random.uniform(a: 0, b: 1), 2)

    # Calculate the shortest paths from the source to all reachable nodes
    paths = [nx.shortest_path(G, source=0, target=1, method='dijkstra', weight='weight'),
             nx.shortest_path(G, source=1, target=2, method='dijkstra', weight='weight'),
             nx.shortest_path(G, source=2, target=3, method='dijkstra', weight='weight'),
             nx.shortest_path(G, source=3, target=4, method='dijkstra', weight='weight'),
             nx.shortest_path(G, source=4, target=5, method='dijkstra', weight='weight'),
             nx.shortest_path(G, source=0, target=5, method='dijkstra', weight='weight')]

    # Draw Graph
    if draw_graph:
        nx.draw_networkx(G, pos)
        labels = nx.get_edge_attributes(G, name: 'weight')
        nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
        plt.title("Random Connected Graph with Edge Weights")
        plt.show()

    return paths, A, pos, G

```

שאלה 6:

בשאלה זו התקשנו לכתוב פונקציה שמחזירה קצבים לכל **flow** ברשות בקרה הבא:
 קיבול כלリンク מחלוקת ע"י **TDMA** בין **flows** שעוברים דרכו לפי יחס החבילות שהם צריכים להעביר. כאשר לכל משתמש יש את קיבולו הילינקם במסלול שלו, קצב **flow** נקבע לפי הילינק על הקיבול המינימלי, נקרא **bottleneck**. לסיום משווים בין צוואר הבקוק לבין כמות החבילות שהמשתמש רוצה להעביר ולקחים את המינימלי. כמובן שאם כמות החבילות קטנה מקיבול הילינק נשתמש רק בכמה שצריך לא יותר. את המסלולים בכל **flow** לוקח אנחנו מחשבים ע"י דיקוסטרא בשכבות הרשת. ברשות שבנו שמננו משקלות יחידה לכלリンク כדי לשמר על אחידות.
 להלן הפונקציה שכתבנו:

```

def TDMA(M, r, num_nodes, number_of_flows, min_demand=100, max_demand=1000, min_capacity=100, max_capacity=500,
         seed=10):
    """
    :param seed: seed for random operations
    :param M: radius M for the network
    :param r: nodes within radius r are connected
    :param num_nodes: number of nodes in the network
    :param number_of_flows: number of flows to be allocated
    :param min_demand: min packets for flow
    :param max_demand: max packets
    :param min_capacity: min link capacity
    :param max_capacity: max capacity
    :return: rate for every flow
    """

    # Generate random network topology with assignments instructions
    A, pos = create_random_graph(num_nodes=num_nodes, M=M, r=r, seed=seed)

    # Generate random flows in the network
    flows = get_random_flows(A=A, num_nodes=num_nodes, num_flows=number_of_flows, min_flow_demand=min_demand,
    # Create a network
    network = Network(flows=flows, adjacency_matrix=A, node_positions=pos, consider_interference=True,
                       min_capacity=min_capacity, max_capacity=max_capacity, seed=seed)

    print(network.flows)
    # show graph
    network.show_graph()

    # Get paths with Dijkstra
    paths = []
    # init weights
    for u, v, attr in network.graph.edges(data=True):
        attr['weight'] = 1

    # get paths
    for flow in flows:
        p = nx.shortest_path(G=network.graph, source=flow["source"], target=flow["destination"], method="dijkstra")
        # if p not in paths:
        paths.append(p)

    # Calculate link capacities
    link_capacities = network.calc_link_capacity(paths=paths) * network.kwargs.get("min_capacity", 200)
    # dict that keeps for each link which flows transmit on it
    link_dict = {network.eids[link]: [] for link in network.id_to_edge}
    for path_idx, path in enumerate(paths):
        for i in range(len(path) - 1):
            link = (path[i], path[i+1])
            link_dict[network.eids[link]].append(path_idx)

    # Start looking for flow rates

    flows_bottleneck = np.zeros(len(flows))
    for flow_idx, route in enumerate(paths):
        path_capacities = []
        for j in range(len(route) - 1):
            l = (route[j], route[j + 1])
            l_index = network.eids[l]

            # get all flows indices that transmit on the link
            flows_on_link = link_dict[l_index]

            link_packets = np.sum(np.array([network.packets[flow_id] for flow_id in flows_on_link]))

            # capacity is divided by the amount of data the flow needs compared to other flows that need same link

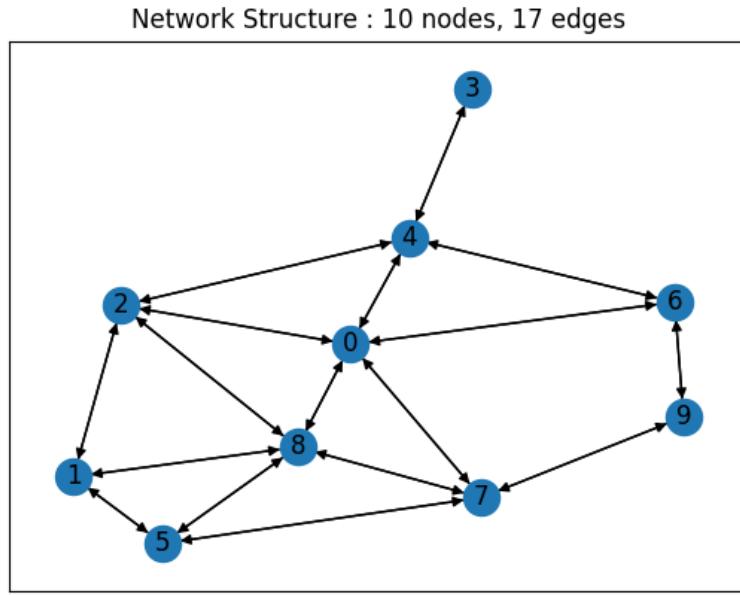
            link_capacity = link_capacities[l_index] * (network.packets[flow_idx] / link_packets)
            path_capacities.append(int(link_capacity))

        # flow rate is the paths bottleneck
        bottleneck = np.min(path_capacities)
        flows_bottleneck[flow_idx] = bottleneck

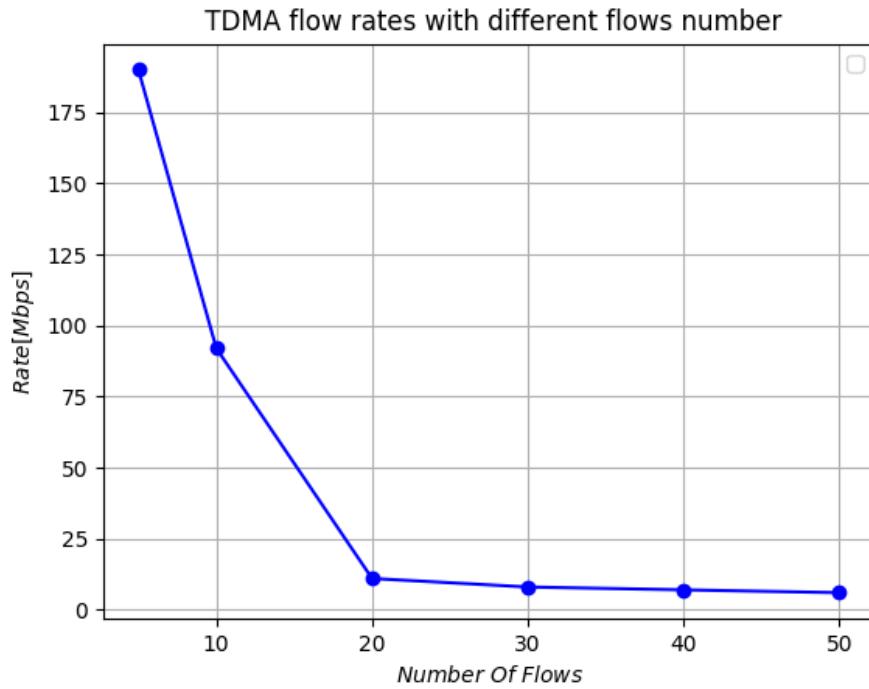
    # At the end. the rate of a flow is the minimum between its demand and bottleneck
    flows_rate = np.array([min(flow["packets"], flows_bottleneck[idx]) for idx, flow in enumerate(network.flows)])
    print(flows_rate)
    return flows_rate

```

במהשך השאלה נרץ על הרשת הבאה:

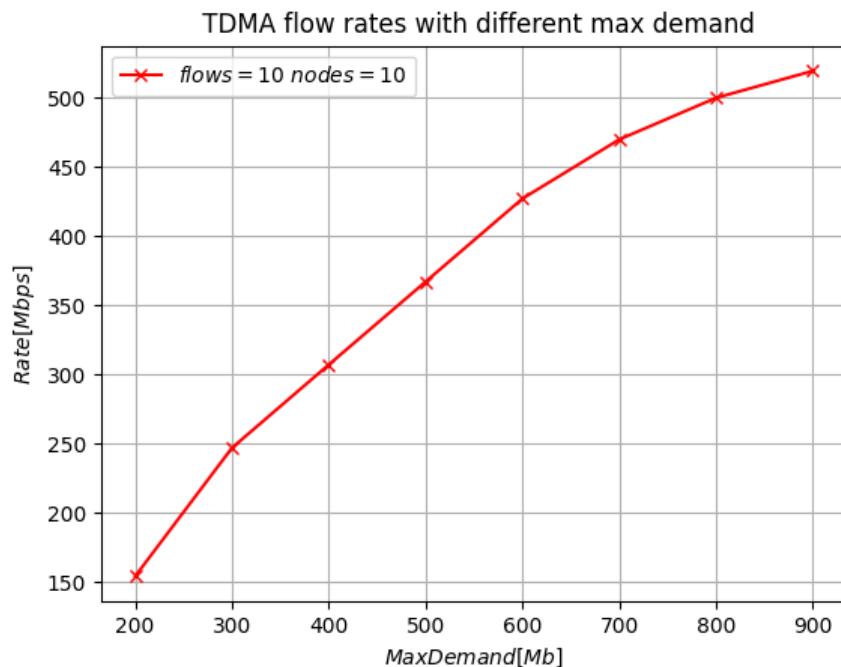


ראשית, נרץ על הרשת את האלגוריתם עבור מספר משתמשים שונה, כלומר **flows** ברשת. נצפה שכל שיש יותר **flows** כך קיובי הליינקים יתחלקו ליותר משתמשים וכל משתמש יקבל קצב שידור נמוך יותר. בחנו בכל פעם את משתמש שמקבל קצב מינימלי ועקבנו אחר השינוי. להלן התוצאות:



כמו שציפינו. ככל שמספר המשתמשים גדול יותר כך הקצב קטן יותר.

שנית, ניסינו לראות מה קורה כאשר הדרישת המקסימלית עולה, לעומת מספר הפקtotות המקסימלי גdal. כדי להבחן בשינוי ללחנו מצב שבו קיבולי הلينקים לא יהיו תמיד קטנים מהדרישה. נזפה שככל שהדרישה העולה הקצב יעלה במעט. עצור הקצב נקבע לפי המינימום בין הדרישת לקיבול של הلينק המינימי במסלול. כאשר כיבולי הلينק קטנים מהדרישה, ככל שנעלה אותה כך גם יעלה הקצב. בוחנו את הקצבים כאשר כל איטרציה הגדלנו את הדרישת המקסימלית. מה שקיבلونו:



כפי. במצב שקיבול הلينקים קטן מהדרישה, ככל שהדרישה תעלה לינארית ככה הקצב

שאלה 7:

עבור שאלה זו K כלומר כל מדרן באחד מתוך K ערכאים אורטוגונליים. בנוסף ערכוי הפעולה גודלים מ-0. בדומה לשאלה 6 המסלולים ברשת ניתנים ע"י דיקטורה שכבת הרשת. קיבול כל לינק מחולק ע"י **TDMA** בין **flows** שעוברים דרכו לפי יחס הדטה שציריך להעבירה. הקצב העובר כל **flow** נקבע עפ"י הלינק בעל קצב השירות המינימלי במסלול שלו, ה- **bottleneck**. ניתן לבחור ערכאים שונים לשידור לאורך המסלול.

מציע את האלגוריתם הבא לבחירת ערכוי שידור:
נזכיר שהסימולטור שלנו מאותחל כך שכל לינק מאופיין ע"י ערכוי שידור בו מדרן צומת המקור של הלינק בגרף המכון. ערכוי השידור מאותחלים בצורה אקראית. לאחר קבלת המסלולים מדיקטורה שכבת הרשת, נעבור על כל מסלול של **flow** ברשת ונבחן את כל ערכוי השידור של משתמשים שעוסקים באותו לינק. אם מספר כל המשתמשים שעוסקים לשדר על אותו לינק קטן ממספר הערכאים האופציוני, נdag שכלם ישדרו בעריך אחר וכך תימנע הפרעה וקיבול הלינק לא יצטרך להיות מחולק שכן ניתן לשדר בו - זמינות בערכאים אורטוגונליים לא הת庵כות של גלי הנושא. השינוי בימוש באידי ביטוי כך:

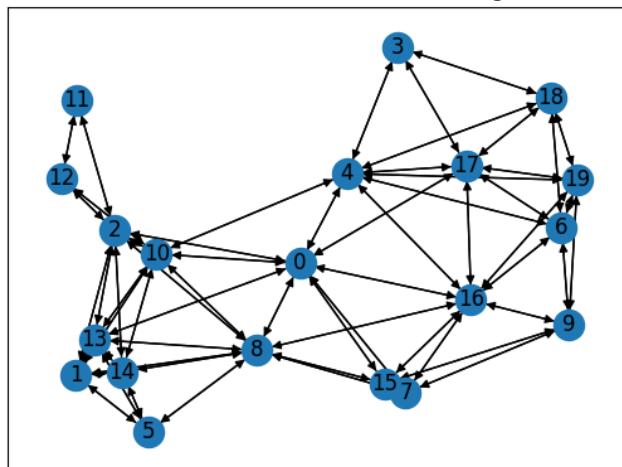
```
# capacity is divided by the amount of data the flow needs compared to other flows that need same link
# If there are more channels, each flow uses different channel

if len(flows_on_link) <= network.K:
    link_capacity = link_capacities[l_index]
else:
    link_capacity = link_capacities[l_index] * (network.packets[flow_idx] / link_packets)

path_capacities.append(int(link_capacity))
```

אם מספר ה **flows** שעוסקים שימוש באותו לינק קטן ממספר הערכאים האפשריים, תבחר ערכץ אחר ועל תחלק את קיבול הלינק, אם לא אפשר חלק לפי גודל הדטה שרוצה להעבירה. מבחינת התוצאות נצפה שככל שיש יותר ערכאים אפשריים, כך יהיה פחות צורך לחלק את קיבולי הלינקים, צוואר הבקבוק יהיה גבוה יותר ונקבל קצבים גבוהים יותר. כמובן שעובדה זו תבוא לידי ביטוי יותר כאשר מספר ה **flows** ברשת יותר גבוהה ואז יהיו יותר משתמשים שציריכים את אותם לינקים. כאשר מספר ה **flows** נמוך, יתכן שלא יהיה שימוש בהם במסלולים שנקל לארה. יהיה צורך לשדר בערכאים שונים ונשאריר אותם בערכאים המקוריים.
את ההשוואה נעשה על הרשת הבאה:

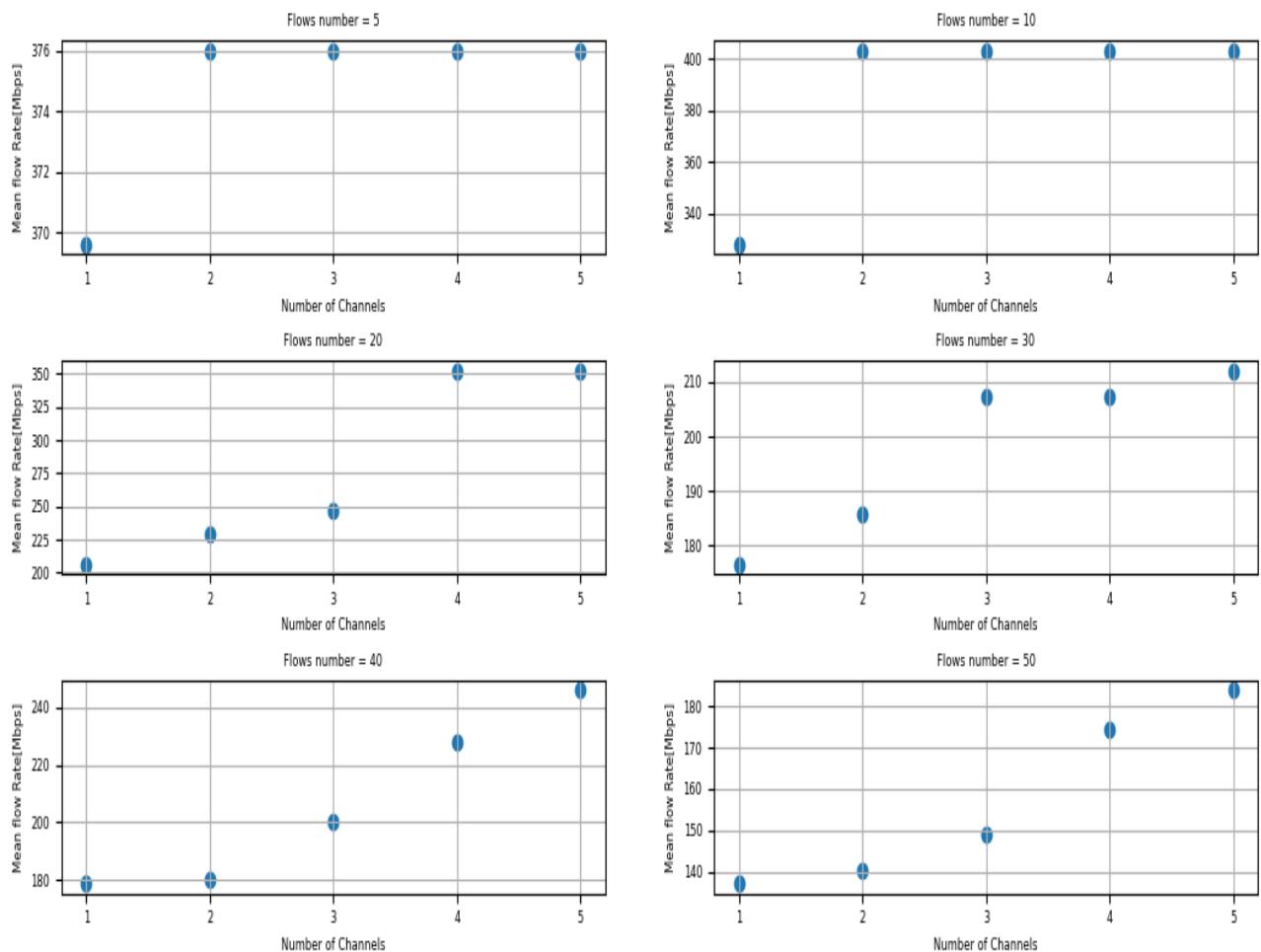
Network Structure : 20 nodes, 61 edges



לקחנו רשות יחסית סבוכה כדי שנוכל להבחן בשיפור של אפשרויות שונות למספר ערוצי שידור. ההשוואה נעשה بصورة הבא: עבור הרשות הנתונה, עבור מספר **flows** שונים, כלומר מספר משתמשים שרצים להעביר מידע ברשת במסלול מסוים, נשווה את הקצב הממוצע שהם מקבלים עבור אפשרות $\leq K$ ערוצים שונים. למשל, עבור מספר **flows** נתון ברשות נשווה איך הקצב הממוצע משתנה כאשר כל משתמש יכול לבחיר מספר ערוצים שונה.

כמו שתכננו קודם, ברשות סבוכה שכזו נצפה שייהי שינוי ככל שמספר **flows** עולה, ככל שהוא יותר גבוה כך הסיכוי לשדר בלינקים משותפים עולה ולכן האפשרות לשדר בערוצים אורתוגונליים תשפיע. להלן התוצאות:

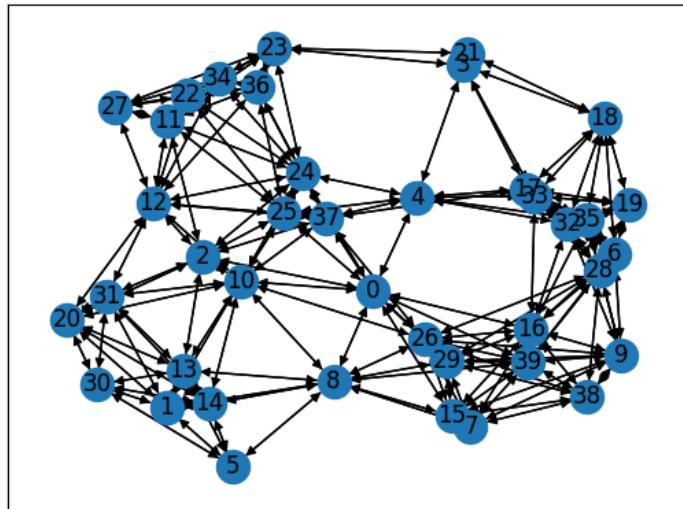
Mean Flow Rate with different K



בדוגמה לעיל יש לכל משתמש אפשרות לבחור באחד מתוך $1 \setminus 2 \setminus 3 \setminus 4 \setminus 5$ ערוצים. כאשר מספר **flows** נמוך ניתן לראות שהגדלת מספר הערוצים לא מושפעה כל כך כי אין הרבה שידורים על LINKים משותפים, בעוד לבןיגוד לכך כאשר מספר **flows** גבוה ניתן לראות שככל שמנגדלים את האפשרות ליותר ערוצים, הקצב הממוצע עולה בכל פעם. בנוסף לכך ניתן לראות שהקצב הממוצע יורד ככל שמספר **flows** עולה בהתאם למה שראינו בשאלה 6.

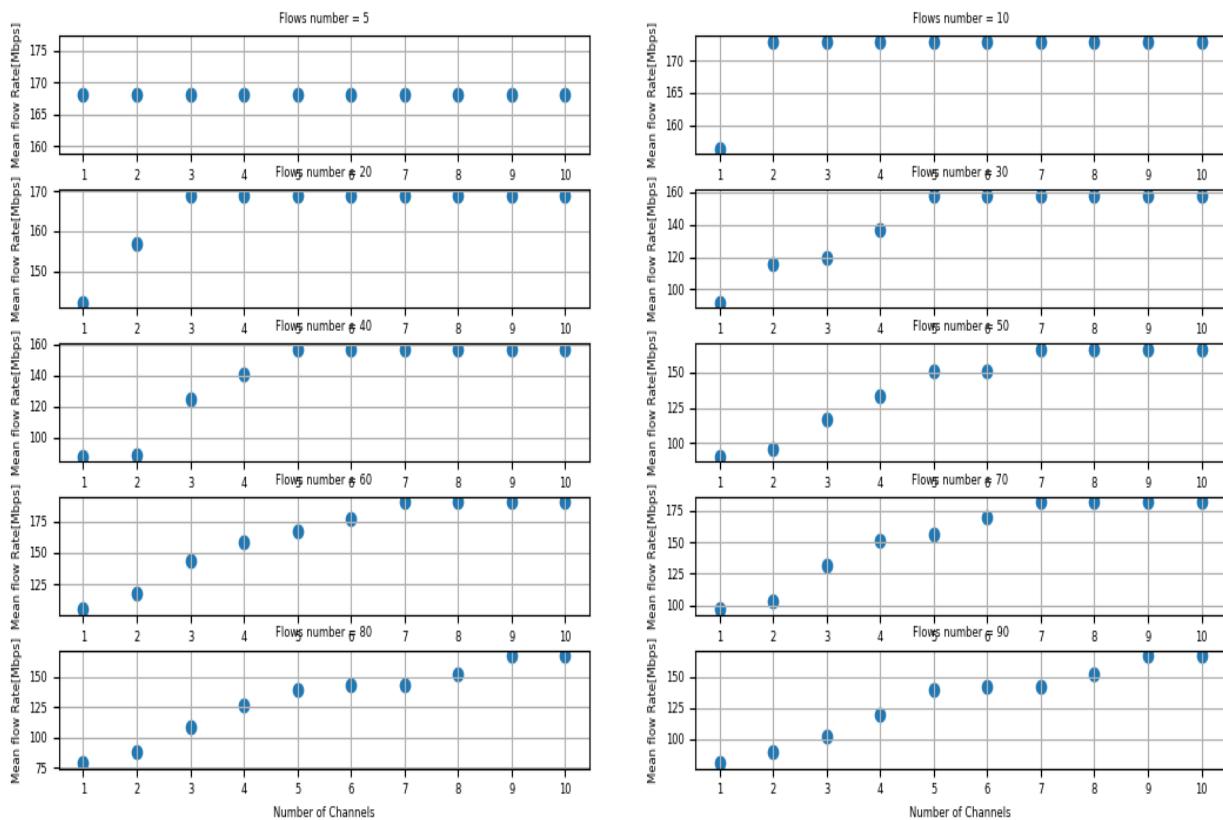
כדי לראות תוצאות עבור מספר K גבוהה יותר לקחנו רשות יותר סבוכה. את הרשות הבאה:

Network Structure : 40 nodes, 183 edges



icut אפרנו עד 10 ערוצים אופציונליים, עד 90 קישורים ל-**flows**. נצפה לתוכאות זהות לרשות הקודמת כך שcut כאשר מספר **flows** מאד גבוה, נדרש מספר ערוצים רב כדי לשפר את הקצב המוצע להלן התוצאות: פעם נוספת ניתן לראות את השיפור ככל שניתן לבחור יותר ערוצים.

Mean Flow Rate with different K



פרק שני - מאמר נלמד

Multi- Flow Transmission in Wireless Interference Networks: A Convergent Graph Learning Approach

by Raz Paul, Kobi Cohen, Gil Kedar

1. תקציר:

במאמר שלפנינו אנו חשים לב עית תקשורת מרובת משתמשים ברשות הפעלה אלחוטיות, כלומר אוטות מידע העוברים במסלולים שונים יכולים להפריע אחד לשני כתוצאה משידור בילינקים מסויימים לאורך המסלול, ולגרום לקיבולי לינקים קטנים יותר מאשר הקיבולים האופטימליים. יש צורך באלגוריתם ניתוב שمبיא את ההפרעות ברשות בחשבון ומספק לכל משתמש מסלול אופטימלי. המאמר מציג אלגוריתם שפותר בעיה זו עם כלים חדשניים של מידת מכונה, למידה عمוקה, למידה חייזקית עמוקה.

מטרת האלגוריתם המוצג במאמר היא: בהינתן N דרישות של שלשות מהצורה - (משתמש מקור, משתמש יעד, כמות מידע להעביר מהמקור ליעד) לספק מסלולים אופטימליים שיביאו את הרשות למוקסימום יעילות או במילאים אחרות: (Network Utility Maximization NUM). מציאת מסלולים אלה היא בעיה חשובה קשה מקורה מרחב חישוב מאד גדול, האלגוריתם המוצע מתמודד עם בעיה זו ע"י מבנה ייחודי חדשני. שם האלגוריתם:

DIAMOND(Dual-stage Interference-Aware Multi-flow Optimization of Network Data-signals).

האלגוריתם כולל שני חלקים, חלק ראשון ריכוזי וחלק שני מבוזר. יישום הכלול מעבר היברידי בין נקודת ריכוזית לבין עדכון מבוזר בניסיון להציג לתוצאות אופטימליות הוא מאפיין של טכנולוגיות 5G ומעבר לכך יכול להתאים לשימוש בזמן אמת ברשות עדכניות.

בחלק הראשון של האלגוריתם, השלב הריכוזי מחשב את המסלולים השונים לכל סוף ברשות בעזרת ארכיטקטורה המשלבת התנסות של סוכן למידה חייזקית עמוקה במרחב המסלולים האפשריים (RL Agent) אשר מקבל החלטות לפי אסטרטגייה הנקבעת לפי מודל למידה עמוקה של גרפים (GNN- Graph Neural Network) שמטרתו לנצל את מבנה הרשת ש-לרבות ממודלת כגרף, לקבלת החלטות טובות.

בחלק השני של האלגוריתם, השלב המבוזר עוזר לנו לשפר את ההחלטה שקיבלנו בשלב הראשון, בעזרת עדכוניים של המשתמשים השונים על מסלול חדש שבחרו. שאר המשתמשים מקבלים את העדכון בזמן אמת ומקבלים החלטה על מסלול חדש שיגרום פחות הפרעה לאחרים בהתאם לכך. במאמר מוצגת הוכחה מתמטית שהמסלולים שמתקבלים כפלט האלגוריתם מתכנסים לאלה האופטימליים ככל שהזמן חולף. בנוסף מוצגות סימולציות שמראות בטופולוגיות רשת שונות את עליונות האלגוריתם על פני מתחריו באופקטים שונים.

2. הקדמה:

א. תיאור כללי של הבעיה:

הבעיה העיקרית שאיתה כותב המאמר מנסה להתמודד היא ש-יחד עם ההתקפות המהירה של טכנולוגיות ש망רתון לספק את הדרישה הולכת וגוברת לשירותי תקשורת אלחוטית בעולם המודרני, כמו טכנולוגיות 5G והלאה, המיעוט ברוחב פס פוני (spectrum scarcity) עדין מהוות בעיה מהותית למשך דרישת זו. לכן נחוצים אלגוריתמים שעושים הרבה מרבבי במשאים הנטוניים ומספקים יכולת טובה לניטוב המידע בשרותות תקשורת אלחוטיות.

ב. תוצאות עיקריות:

התוצאות המרכזיות במאמר הן שהאלגוריתם המוצע בו מספק מסלולים אופטימליים לשידור מידע בסביבת רשת המתחשבת בשידור בו בזמן ייחד עם הפרעות הנוצרות משידור במסלולים מקבילים. ניתן הוכח מתמטית ששימוש באלגוריתם הנ"ל מביא את הרשות לנקודת אופטימלית ככל שעובר הזמן.

האלגוריתם הפופולרי ביותר כיום לניטוב מידע בראשת הוא: OSPF(Open Shortest Path First). המשתמש בקביעת מסלולים בין מקור לעד ע"י המסלול הקצר ביותר. הוא פופולרי בגלל הפשטות שלו והעובדת שעוזב טוב בשרותות שנחשבות ללא עומסות והפרעה בהן היא לא גבוהה, אך החיסרון שלו הוא עשוי לגרום לצפיפות (congestion) בראשת ולהביא לביצועים רעים בשרותות עומסות ובשלות הפרעה גבוהה.

בשנים האחרונות נעשו שיפורים משמעותיים ע"י פיתוח אלגוריתמים הנעזרים בלמידת מכונה לניטוב מידע בראשת בניסיון להתמודד עם אי-ודאות במצבים אקרים בראשת. במרבית האלגוריתמים הללו הייתה התמקדות בשידור בקפיצה אחת (one hop), במאמר שלנו ההתמקדות היא ב-hop multi transmission במהלך הלמידה, כך שתאפשר בחירה יעילה של מסלולים לשידור.

בנוסף, במקרים רבים אין התייחסות להפרעה הנובעת משידור במסלול מסוים על מסלולים מקבילים. לכן גם אם שיטות כאלה יעבדו בראשת קטנה ופחות "סבוכה", בראשות עומסות הביצועים יהיו הרבה פחות טובים. האלגוריתם הנוכחי במאמר מתגבר על בעיה זו בכך שלומד אסטרטגיית שידור בהתאם למצב הנוכחי של הרשת, וכך מביא למינימום את הרפרעה הנובעת מבחירה מסלול על אלו האחרים תוך כדי עדכון דינامي ובחירה מסלולים אלטרנטיביים בצורה מושגת.

3. מודל המערכת ותיאור הבעיה:

בסעיף זה נציג נमدل את רשת התקשרות, את הבעיה בצורה מתמטית כמתואר במאמר: רשת תקשורת אלחוטית ממודלת כגרף $(V, E) = G$ כאשר V מסמל את קבוצת הצלמים, כלומר E מסמל את קבוצת הקשיות, כלומר לינק תקשורת בין משתמשים. לינק בין משתמש i למשתמש n ממודל כlienk ביןיהם $E \in (n, i)$ בgraf מכוון. לינקים גורמים להפרעות כתוצאה משידור באיזור גיאוגרפי סמוך. נסמן ב- F סט של $awsflows$ N ברשות. כאשר כל איבר ב- F קבוע לפי צומת מקור וצומתיעד $V \in A_n, d_n$. נסמן ב- $\{\varphi_1^n, \varphi_2^n, \dots, \varphi_{k_n}^n\} = A_n$ את המסלולים האפשריים ל- n . $A_n \in \sigma$ מסמל את המסלול הנבחר. $(\sigma_1, \sigma_2, \dots, \sigma_N) = \sigma$ מסמל את וקטור המסלולים לכל $awsflow$ ברשות - $(\sigma_N, \dots, \sigma_{n+1}, \sigma_n, \dots, \sigma_1) = \sigma$ וקטור כל המסלולים למעט זה של n . $awsflow$. נסמן ב- (σ, u) פונקציית תועלת חסומה כלשהי למ- $awsflow$ כך ש: $n \leq u \leq (\sigma, u)$. נשים לב שהתועלות של כל $awsflow$ מושפעת מבחרית המסלול שלו וגם מבחרית המסלולים האחרים שגורמים לו הפרעה. נסמן ב- N את סט כל ה- $awsflows$ גורמים מספיק להפרעה ב- $awsflow$, (נראה לו שכנים מפריעים). כל שאר המסלולים אינם גורמים מספיק הפרעה כדי להיות בסט זה. לעומת זאת התועלות של כל $awsflow$ תליה במסלול הנבחר בשילוב יחד עם המסלולים שנבחרו לאו $awsflow$ המפריעים לו. במילים אחרות: $(\sigma_i, \dots, \sigma_n) = (\sigma, u)$.

המטרה של האלגוריתם המוצע במאמר הוא למצוא את וקטור המסלולים σ הפתרן של בעית המקסימום תועלת – (NUM): *Network Utility Maximization*.
 $\{(\sigma, u) \in \sigma \in A_n\}_{n=1}^N = arg \max \sum_{n=1}^N u_n$. תועלת של n $awsflow$ היא פונקציה של הקצב שמקצת לשידור. לדוגמה, ב-*Proportional Fairness* שראינו בהרצאה- $(\sigma, u) = \log(R_n)$. קצב של כל $awsflow$ נקבע עפ"י הקיבולת של הלינק היכי איטי במסלול, נקרא גם ה-*Bottleneck*. קיבולת של כל לינק במסלול מחושבת עפ"י הנוסחה של שאנון: $C_l = B_l \cdot \log_2(1 + SINR_l)$. כאשר B_l הוא רוחב הפס של לינק l ו- $SINR_l$ (Signal To Interference plus Noise Ratio) מאופיין ע"י המשוואה $SINR_l = \frac{P_l}{I_l + \sigma^2}$. כאשר P_l הוא הספק השידור בלינק l ו- σ^2 רעש גaus. (σ, u) הוא הסכום של כל ההפרעה שנובעת משידור בלינקים במסלולים אחרים ששיכים לסט $awsflows$ השכנים N .

הנחה שנעשתה במודל המוצע במאמר היא שהספק השידור בכל הלינקים זהה. אין שוני שיכול לנבוע מסוג המשדר. בנוסף בחישוב מפת ההפרעות בין לינקים, נלקח בחשבון רק מרחק גיאוגרפי ביניהם ולא אלמנטים אחרים שיכולים להיות חשובים כמו הרזותם ביןיהם שתגרום להפרעה מכיוון האנטנות. הנחה נוספת היא שהשידור נעשה בערוץ אחד, כלומר בבעית האופטימלית אין אפשרות לבחור מתווך K ערוצים אורטוגונליים לשידור סיגナル.

4. פתרונות לבעיה:

בחלק זה נציג את שלוש השלבים המרכזיים באלגוריתם המוצע במאמר:

שלב א' - מצטום מרחב החיפוש DIAMOND:

זיכרון שמרתת האלגוריתם היא מציאות וקטור מסלולים s לכל flow ברשות שיפור את בעית NUM . יחד עם זאת מספר המסלולים האפשריים הוא ענק וגם עליה מעריכית עם מספר ה flow . לכן יש צורך למצער את מרחב החיפוש. למשל נקבע מרחב החיפוש בכניסה לאלגוריתם ירכיב מ K_n מסלולים אפשריים לכל flow . במחקרנים אחרים ניסו למצער את מרחב החיפוש ע"י מציאת מסלולים קצרים ביותר אך זה לא רלוונטי ברטחות הפרעה שכן מסלולים קצרים לאו דווקא מבאים למינימום הפרעה. יש צורך באלגוריתם יותר מתחכם למצטום מרחב החיפוש. במאמר מוצע אלגוריתם זהה. לכל flow מרכיבים את האלגוריתם כך שהפלט הוא רשימה של K_n מסלולים אפשריים לבחור ביניהם. האלג' עושה שימוש בדיקסטרה איטרטיבי בצורה מושכלת ע"י שינוי המשקولات בכל איטרציה, מתחילה מזורה איחודית ובכל איטרציה מוסיפים את הערך ההופכי של הלינק ה"קצר" ביותר. האלג' נראה כך:

Algorithm 1 Construction of \mathcal{A}_n for flow n

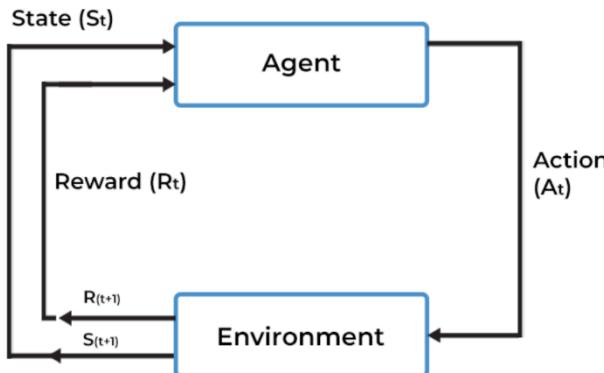
```
1: Input:  $G = (\mathcal{V}, \mathcal{E})$ ,  $s_n, d_n \in \mathcal{V}$ 
2: Initialize:  $W(\varepsilon) \leftarrow 1 \forall \varepsilon \in \mathcal{E}$ ,  $P \leftarrow []$ 
3: for  $i = 1, 2, \dots, K_n$  do
4:    $\varphi \leftarrow \text{shortest-path}(s_n, d_n, G, \{W(\varepsilon)\}_{\varepsilon \in \mathcal{E}})$ 
5:    $P \leftarrow P \cup \{\varphi\}$ 
6:    $W(\varepsilon) \leftarrow W(\varepsilon) + \min_{\ell \in \varphi} d^{-1}(\ell, \varepsilon), \forall \varepsilon \in \mathcal{E}$ 
7: end for
8: Return:  $P$ 
9:  $\mathcal{A}_n \leftarrow P$ 
```

שלב ב' - השלב הריכוזי (GRRL- Graph neural network Routing agent via Reinforcement Learning)

שלב זה פועל בצורה ריכוזית ומשלב טכניקות של למידה חיזוקית (RL) עם ארכיטקטורה של רשתות נירוניים عمוקות (DNN), בפרט רשתות גרפיים (GNN). נסביר על הנושאים (עוד בנספחים). העולם

של למידה חיזוקית מרכיב מסוכן (Agent) שעושה פעולות בסביבה מסוימת (environment) , מקבל עליהן פרס וכך צריך ללמוד איזה פעולות יביאו לו פרס עתידי הכי גבוה. במאמר שלנו, Action מודמה למסלול הנבחר לכל הFLOW ברשות כלומר וקטור המסלולים σ , Flow לכל Flow מרכיב מהתועלות שלו באлокציה הנוכחית פחותה ההפרעה שהוא גורם למסלולים

REINFORCEMENT LEARNING MODEL

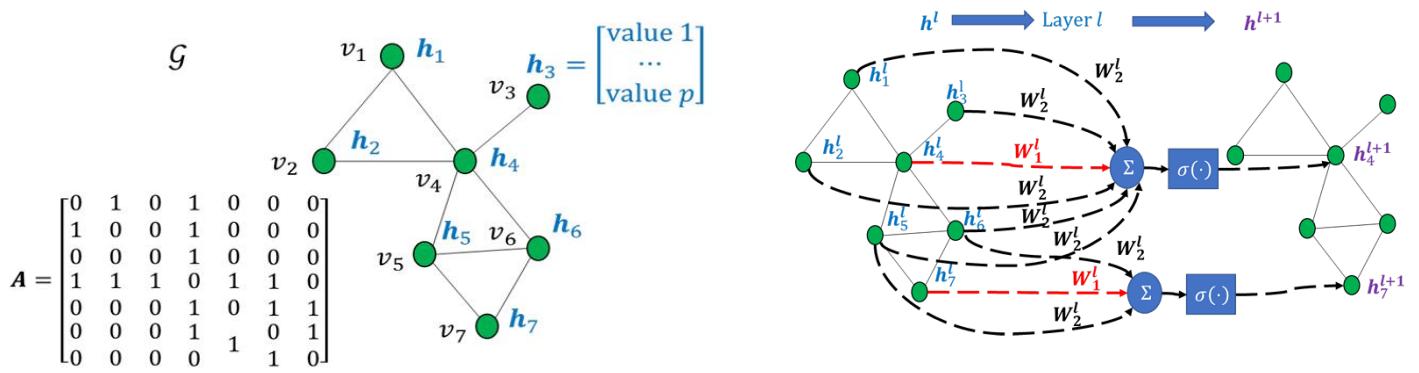


אחרים. لكن המטרה של המודל שלנו היא למדוד איזה מסלולים (Actions) יגרמו לו למקסם את התועלת ברשות יחיד עם שיקול ההפרעה (Rewards).

ברשותות תקשורת קשה למודל מצבים (States) لكن כותב המאמר משתמש באלאgorיתם REINFORCE שבו הסוכן לוקח פעולות לאורך trajectory, מקבל פרסים ומעדכן את משקולות המודל בהתאם(הרחבת בנספחים).

כדי לקבל את וקטור המסלולים σ נעשה שימוש במודל GNN-Graph Neural Network שמתՐתו לאפשר לקבל ריצה עיליה על מרחב האפשרויות למסלולים האפשריים לכל Flow.

מודלים של רשתות גרפים פרצו בשנים האחרונות כארQUITקטורת למידה عمוקה שעזרה להתמודד עם דאטה המופיע בזורה גרפית כדוגמת תקשורת, מערכות בגוף האדם, רשת הכבישים, תשתיות החשמל וכו'. בדומה לכך שרשתות מסוג RNN, Recurrent CNN מסוגלות למדוד מידע ספציפי בזורה יعلاה כמו תמונות ושפה, כך מצאו שארQUITקטורות GNN למודד בזורה יعلاה מידע המיצג כגרף. לכל צומת בgraf יש וקטור פיצ'רים או ובעזרת תהליכי הלמידה אנחנו יכולים לקבל וקטור ה עם ערכים אופטימליים למשימה שהגדכנו לעצמנו. מבון שיש המונן ארQUITקטורות שונות שמנגדירות איר נראית שכבה אבל באופן כללי כל וקטור פיצ'רים מוכפל באיזושהי מטריצה נלמדת והמכפלה עוברת דרך פונקציית אקטיבציה:



במאמר שלפנינו, הכותב מתייחס לכל-link $E \in \mathcal{E}(n, u) = l$ כאיל צומת ב NN . וקטור הפיצרים h_{in} לכל צומת מורכב משולשה ערכים (הפרעה על הלינק, קיבולת הלינק, האם היה בו שימוש בקביעה الأخيرة של המסלול).

$$h^{(t)}(\ell) = \gamma \left(h^{(t-1)}(\ell)W_1^t + \sum_{e \in E_{in}(\ell)} h^{(t-1)}(e)W_2^{(t)} \right) + \sum_{e \in E_{out}(\ell)} h^{(t-1)}(e)W_2^{(t)}, \quad (5)$$

לבסוף מתקיים לכל-link ברשות ייצוג $h(l) \in \mathbb{R}^d$ וייצוג לgraf כמוצע של הלינקים: $h(G) \in \mathbb{R}^d$.

לאחר מכן כל מסלול אפשרי i له *flow* מקבל *embedding* $h(i) \in \mathbb{R}^d$ דרך מעבר בעוד רשות יחיד עם הייצוג שקיבלו כל הלינקים שמרכיבים אותו. ייצוג זה מסומן ב- π_i .

בחירת המסלול הנבחר φ מתאפשר ע"י פעולה *softmax* לכל הייצוגים של של המסלולים:

$$p_i^n = \frac{\exp(u_i)}{\sum_{j=1}^{K_n} \exp(u_j)}$$

לסיכום. שלב ה-GRRL נותן את המסלול לכל *flow* כתוצאה מכך סוכן הלמידה החזוקית ל选取 צעד (*Action*) ומתקבל פרט. ככה למספר איטרציות שאנו קובעים מראש. בסופה של דבר הפלט של שלב זה הוא וקטור מסלולים σ :

Algorithm 2 The operation of the DRL agent

```

1: Inputs:  $G = (\mathcal{V}, \mathcal{E}), \{s_n, d_n\}_{n=1}^N, \{K_n\}_{n=1}^N$ 
2: Output:  $\{\varphi_n\}_{n=1}^N$ 
3:  $\mathcal{G} \leftarrow \text{env.reset}()$ 
4: for  $n = 1, 2, \dots, N$  do
5:    $\mathcal{A}_n \leftarrow \text{Alg.1}(s_n, d_n, K_n)$ 
6:    $\pi_n \leftarrow \text{GNN}(\mathcal{A}_n, \mathcal{G})$ 
7:    $\varphi_n \sim \text{Softmax}(\pi_n)$ 
8:    $\mathcal{G} \leftarrow \text{env.allocate}(\varphi_n)$ 
9: end for
10: Return:  $\{\varphi_n\}_{n=1}^N$ 
```

שלב ג' השלב המבוזר(*NB3R- Noisy Best-Response for Route Refinement*):

לאחר שנבחרו מסלולים לכל *flow* בשלב ב'. בשלב ג' נועד לשפר את החלטה ולהגיע למינימום גלובלי ע"י עדכון מבוזר של משתמשים בראשת. יש הנחה שכלי *flow* מעדכן את האסטרטגיה שלו בזמןים שונים ואין התנגדויות בין הזמןנים, כאשר *flow* מסוים מגיע לעדכן את הבחירה שלו למסלול מסוים הוא מניח שמצוב בראשת נשאר כפי שהוא עד שיקבל החלטה. מגדירים תועלת משותפת:

$$\mathcal{U}_n(\sigma) = u_n(\sigma) + \sum_{m \in \mathcal{N}(n)} u_m(\sigma).$$

התועלת המשותפת של n *flow* מורכבת מהתועלת אישית יחד סכום התועלות של השכנים המפריעים. השלב המבוזר עובד בצורה הבאה:

בכל שלב עדכון בוחרים סט של *flows*. כל *flow* בסט לדוגמה l מבקש מהשכנים שלו לחשב את התועלת האישית שלהם לכל אחת מהאפשרויות למסלול של *flow*. (Nazcor שכל מסלול שיבחר ישפי על התועלת האישית של השכנים שבחישובה כולל הפרעה): $(\sigma_{-n}, \varphi_l) \in K_n$ אפשרויות שונות

למסלול. לאחר שיקבל את המידע מהשכנים. יחשב את פונקציית התועלת המשותפת שהרaterno קודם ל K_n האפשרויות השונות. לבסוף יעשה פעולה Softmax על כל האפשרויות השונות:

. יבחר מסלול חדש (לא בהכרח) ויעדכן את כל שכניו על השינוי. ככה נמשיך עד להתכנסות:

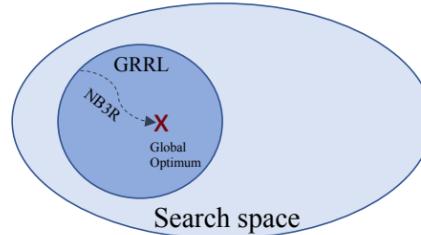
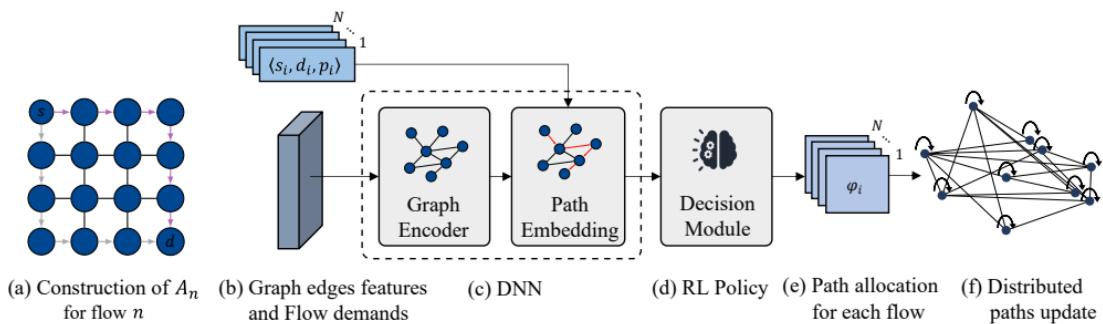
Algorithm 3 NB3R Algorithm

```

1: Inputs:  $G = (\mathcal{V}, \mathcal{E})$ ,  $\{s_n, d_n, \mathcal{A}_n\}_{n=1}^N$ , path allocation
    $\{\varphi_n\}_{n=1}^N$  output by Alg. 2
2: Output: Path  $\varphi_n^*$  for each flow  $n$  (distributedly)
3: Initialize: Each flow (say  $n$ ) transmits its data signal
   through path  $\varphi_n$ 
4: repeat (at each updating time)
5:    $\mathcal{F}_{opt} \leftarrow$  updated set of optimizing flows
6:   for  $n \in \mathcal{F}_{opt}$  do:
7:     Query  $K_n$  utility values:  $u_m(\varphi_1^n, \sigma_{-n}),$ 
      ...,  $u_m(\varphi_{K_n}^n, \sigma_{-n})$  from all neighbors
       $m \in \mathcal{N}(n)$ 
8:     Compute  $U_n(\varphi_1^n, \sigma_{-n}), \dots, U_n(\varphi_{K_n}^n, \sigma_{-n})$  (9)
9:     Draw  $\varphi_n$  from distribution (10)
10:    Inform all neighbors  $m \in \mathcal{N}(n)$  of the up-
      dated  $\varphi_n$ 
   end for
11: until all utilities converge
12: Return:  $\{\varphi_n^*\}_{n=1}^N$ 
```

לסיכום, נציג בעזרה תמונות הלקוחות מהמאמר את השלבים השונים באlgorigם ואת ההתכנסות
שלו לבחירה אופטימלית:

4



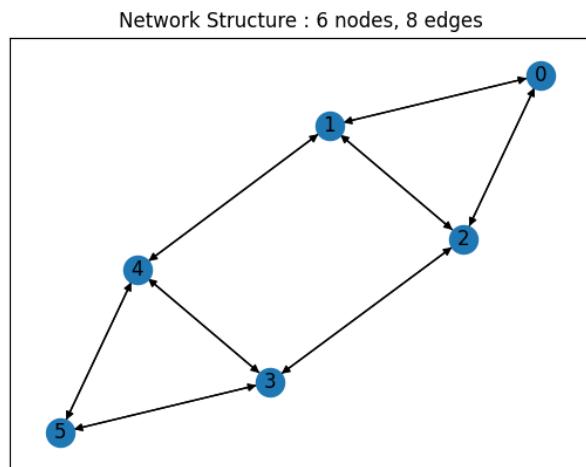
במאמר מוצגת הוכחה מתמטית שעשויה שימוש בכלים מתורת המשחקים המראה שהאלגוריתם מגיע למסלול שפותרת את בעיית הNUM בצורה מיטבית ככל שעובר הזמן. אך לא ניתן לפירוט בשלב זה.

5. תוצאות סימולציה:

בUPI זה נשווה תוצאות שקיבלנו בשלב הסימולטור לבין תוצאות שקיבלנו עם כלים מהמאמר. נזכיר שפלט האלגוריתם המתואר במאמר הוא וקטור מסלולים לכל s ו f ו ברשות. لكن ההשוואה تعשה בתוצאות שקיבלנו עבור תרגילים שעסקו בפתרון בעיות בהינתן מסלולים.

השוואה עם שאלה 5:

בשאלה 5 בשלב הסימולטור התקשנו להריץ את האלגוריתם הפרימלי והדואלי על רשת נתונה ולקבל סט קצבים אשר יביא את הרשת לנקודת עבודה טובה. בשאלה היא, המסלולים נקבעו עפ"י Dijkstra בשכבת הרשת.Cut המסלולים יקבעו עפ"י אלג' DIAMOND המוצע במאמר. ראשית נזכיר שאנו עובדים עם רשת מהצורה:



כasher לכל link קיבול יחידה. נזכיר שטת ה- Flows שאנו עובדים איתם הם מהצורה:

```
flows = [{"source": 0, "destination": 1, "packets": np.random.randint( low: 100, high: 300)}, {"source": 1, "destination": 2, "packets": np.random.randint( low: 100, high: 300)}, {"source": 2, "destination": 3, "packets": np.random.randint( low: 100, high: 300)}, {"source": 3, "destination": 4, "packets": np.random.randint( low: 100, high: 300)}, {"source": 4, "destination": 5, "packets": np.random.randint( low: 100, high: 300)}, {"source": 0, "destination": 5, "packets": np.random.randint( low: 100, high: 300)}]
```

עבור אלגוריתם דייקסטרה עם משקלות רנדומליות, קיבלנו סט מסלולים:

```
> paths = {list: 6} [[0, 2, 1], [1, 2], [2, 3], [3, 4], [4, 5], [0, 2, 1, 4, 5]]
```

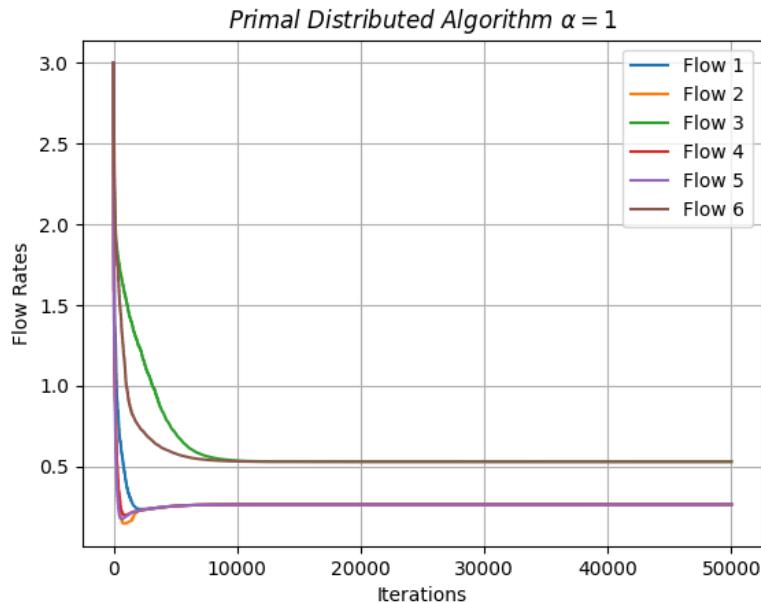
cut עפ"י DIAMOND קיבלנו סט מסלולים:

```
> diamond_paths = {list: 6} [[0, 2, 3, 4, 1], [1, 4, 5, 3, 2], [2, 3], [3, 2, 1, 4], [4, 1, 2, 3, 5], [0, 1, 4, 5]]
```

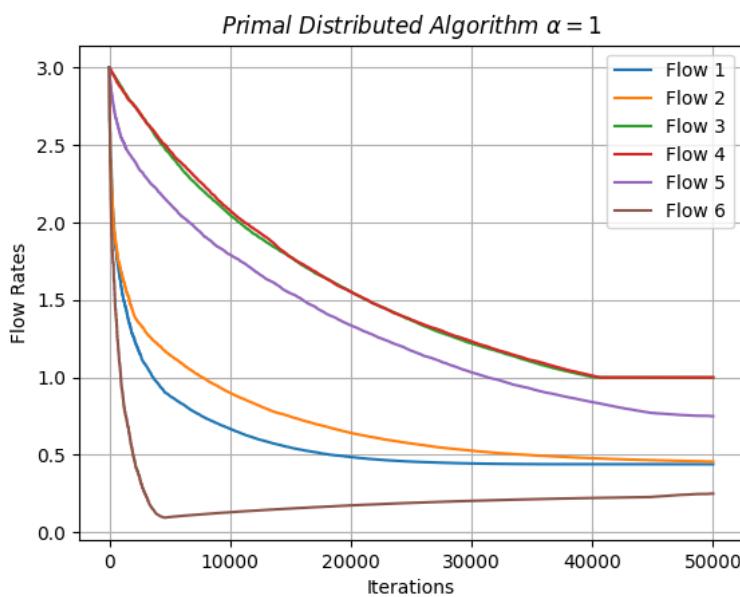
ונסה להבין את ההיגיון בשינוי – נזכיר שדייקסטרה לא לוקח בכלל הפרעות כתוצאה משידור בלינקים סמוכים לעומת האלג' המוצע במאמר. لكن הגיוני מכך שהמסלולים יהיו הרבה יותר ארוכים כדי למנוע הפרעה. אם מתסכלים על המסלולים ואירוגרף, ניתן לראות שהרבה פעמים משתמש ש्रוצה לשדר למשתמש סמור ישמש בדרך עקיפה ולא ישירה כדי "להתחמק" משידור בלינקים סמוכים. שימוש בדייקסטרה מביא למסלול קצר אך עלול לגרום להפרעה רבה ואיבוד כבilities בראשת סבוכה יותר.

כעת נריץ את האלג' הפרימלי והדוואלי לקבלת סט הקצבים לכל המשתמשים. בדומה לשאלה בחלק הסימולטור, נריץ עם $\alpha = 1$ וنبיא את הרשות לנו' עבודה טובה.

ראשית אלג' פרימלי עם מסלולי DIAMOND:

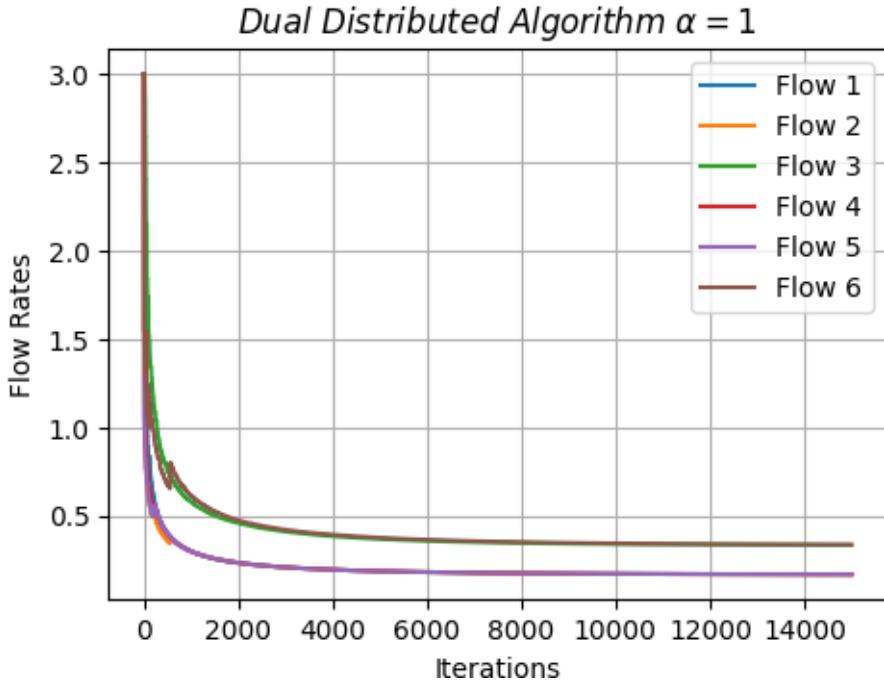


נזכיר שהאלג' הפרימלי נותן קצבים גבוהים יותר למשתמשים שעושים בלינקים פחות עמוסים, ככלומר פחות משתמשים "צריכים" את הלינק הזה. אכן ניתן לראות שימוש שמשתמשים 3,6 עושים שימוש בלינקים פחות "פופולריים" ולכן מקבלים קצבים גבוהים יותר. כאשר הרצינו עם מסלולים מד"יקסטרה קיבלו:



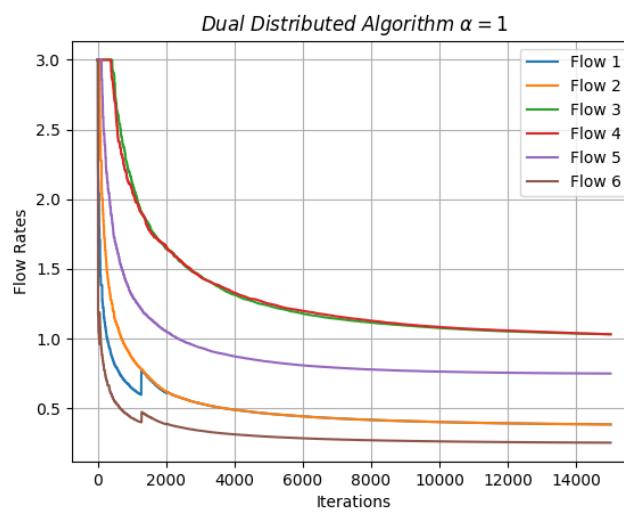
ניתן לראות התוכנות לקצבים שונים לחישוטן שנובעים מהשוני במסלולים. ניתן לראות שימוש משתמשים מקבלים כעט קצבים נמוכים יותר והרשות יותר מאוזנת.

cut נרץ את האלג' הדואלי ונזכה לקבל תוצאות זהות לאלה של הפרימלי עם נתוני רשות, מסלולים
זהים:



פעם נוספת ניתן לראות שflow 6 קיבל את אותו קצב לעומת כל השאר שגם מקבלים קצב זהה
נמוך יותר כמו באלג' הפרימלי.

בתוצאות עם מסלולים מדיקסטרא קיבלנו:

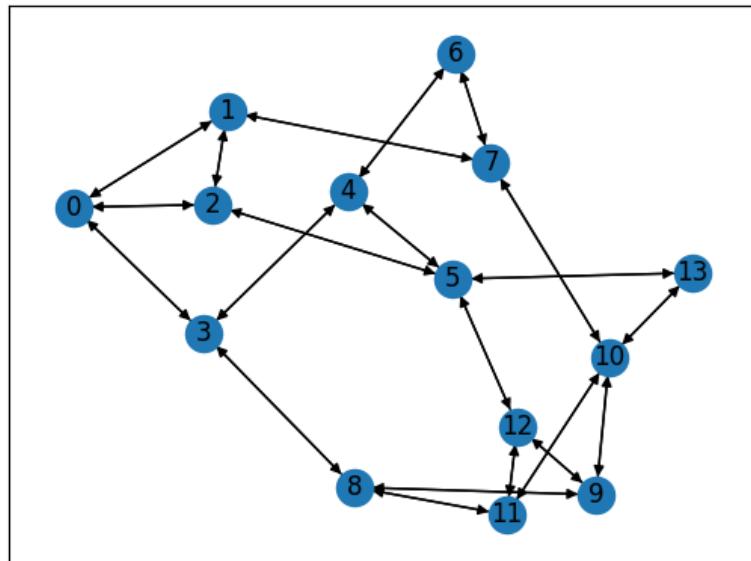


התוצאות זהות לאלה שראינו באלג' הפרימלי لكن גם ההסברים על השוני בין תוצאות הסימולטור
שלמו לבין ריצה עם מסלולים הנbowים מאלג' DIAMOND .

השוויה עם שאלה 6:

ב שאלה 6 בשלב הסימולטור התבוננו למצוא סט קצבים לכל flow ברשת. סט הקצבים נקבע עפ"י bottleneck של כל מסלול כאשר קיומי לינקים התחלקו לפי TDMA לפי יחס הדטה של כל flow. המסלולים נקבעו לפי דיקסטרה בשכבות הרשת. כעת נשווה ע"י כך שנכנייס לאלגוריתם את המסלולים שאנו מתקבלים מאלגוריתם DIAMOND שמתוחשב בהפרעות בין לינקים. את ההשוואה נבצע על הרשת הבאה:

Network Structure : 14 nodes, 21 edges



נעשה 2 השוואות. בכל אחת נדרש מספר flows שונה ברשת, נקבל מסלולים מדיקסטרה\DIAMOND ונחשב את סט הקצבים לפי האלגוריתם TDMA שכתבנו. ראשית ניקח דרישת 5 Flows ברשת(הדרישה רנדומלית):

```
flows = [{"destination": 3, "packets": 280, "source": 1}, {"destination": 11, "packets": 400, "source": 4}, {"destination": 0, "packets": 140, "source": 7}, {"destination": 5, "packets": 750, "source": 1}, {"destination": 0, "packets": 760, "source": 5}]
```

ע"י דיקסטרה עם משקלות ייחידה קיבל את המסלולים:

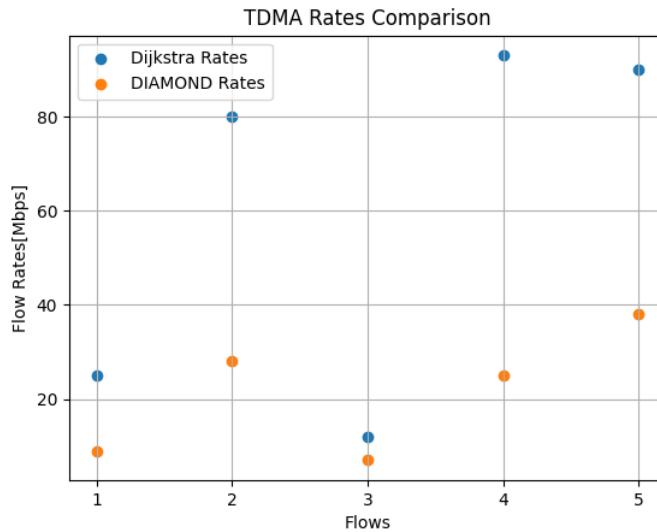
```
> paths = {[list: 5] [[1, 0, 3], [4, 3, 8, 11], [7, 1, 0], [1, 2, 5], [5, 2, 0]]]
```

כעת, ע"י הרצת DIAMOND קיבלנו את המסלולים הבאים:

```
diamond_paths = {[list: 5] [[1, 0, 3], [4, 5, 13, 10, 11], [7, 6, 4, 3, 0], [1, 0, 3, 4, 5], [5, 4, 3, 0]]}
```

פעם נוספת נוכיח מבחן אחד שאלגוריתם מסלולים יותר "ארוכים" כדי לקבל דרך עם פחות הפרעות כדי להשיג את המטרה שהוא שואף להגעה אליה.

כאשר נריץ את האלגוריתם נזכה לקבל קצבים יותר גבוהים באlg' דיקסטרה. ניתן לראות שעבור מסלולים ארוכים יותר ישנים שימושים נוספים לינקים لكن יש יותר שידורים בלינקים משותפים. החישוב בפוי יביא לחלוקת קיבולי הLINK ליותר משתמשים. נציג CUT את הקצבים שקיבלו עבור חמשת ה-flows ברשות:



בהתאם למה שתיארנו ניתן לראות שמסלולים קצבים נמוכים יותר עבורalg' DIAMOND כי נעשים בו יותר שידורים בלינקים משותפים.

CUT נעשה השוואה עבור אותה רשת עם דרישת של 20 flows במקום 5:

```
flows = [{"source": 1, "destination": 3, "packets": 280}, {"source": 4, "destination": 11, "packets": 400}, {"source": 7, "destination": 0, "packets": 140}, {"source": 1, "destination": 5, "packets": 750}, {"source": 5, "destination": 0, "packets": 760}, {"source": 11, "destination": 7, "packets": 570}, {"source": 9, "destination": 0, "packets": 790}, {"source": 12, "destination": 13, "packets": 260}, {"source": 8, "destination": 0, "packets": 720}, {"source": 11, "destination": 12, "packets": 350}, {"source": 13, "destination": 4, "packets": 350}, {"source": 7, "destination": 1, "packets": 520}, {"source": 1, "destination": 0, "packets": 770}, {"source": 1, "destination": 11, "packets": 300}, {"source": 7, "destination": 0, "packets": 650}, {"source": 0, "destination": 13, "packets": 810}, {"source": 5, "destination": 3, "packets": 300}, {"source": 7, "destination": 4, "packets": 730}, {"source": 2, "destination": 8, "packets": 480}, {"source": 9, "destination": 13, "packets": 830}]
```

פעם נוספת נמצא מסולים עם דיקוסטרא בשכבה הרשות(ניתן להשוו את המרחקים הקצרים עם תרשימים הרשות):

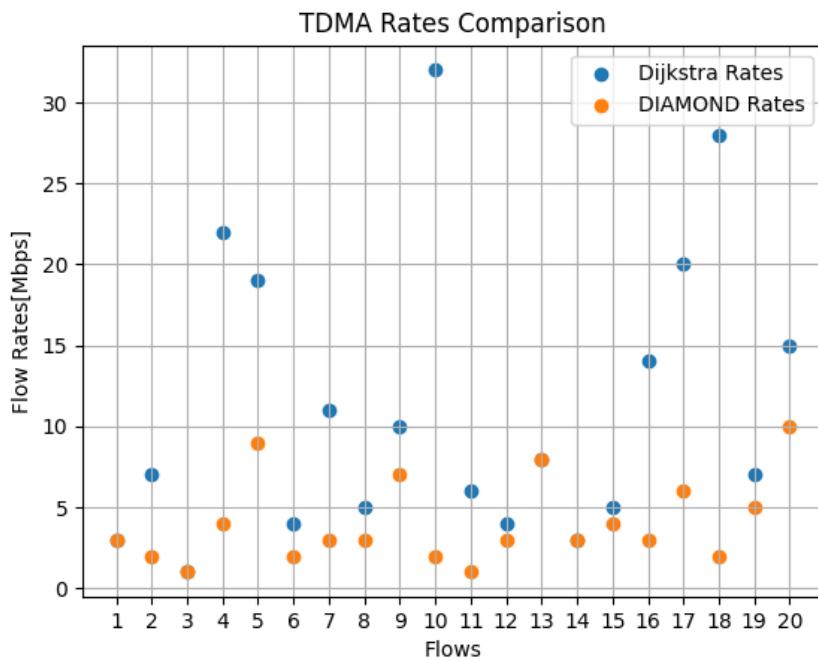
```
> 2 00 = {list: 3} [1, 0, 3]
> 1 01 = {list: 4} [4, 3, 8, 11]
> 1 02 = {list: 3} [7, 1, 0]
> 1 03 = {list: 3} [1, 2, 5]
> 1 04 = {list: 3} [5, 2, 0]
> 1 05 = {list: 3} [11, 10, 7]
> 1 06 = {list: 4} [9, 8, 3, 0]
> 1 07 = {list: 3} [12, 5, 13]
> 1 08 = {list: 3} [8, 3, 0]
> 1 09 = {list: 2} [11, 12]
> 1 10 = {list: 3} [13, 5, 4]
> 1 11 = {list: 2} [7, 1]
> 1 12 = {list: 2} [1, 0]
> 1 13 = {list: 4} [1, 7, 10, 11]
> 1 14 = {list: 3} [7, 1, 0]
> 1 15 = {list: 4} [0, 2, 5, 13]
> 1 16 = {list: 3} [5, 4, 3]
> 1 17 = {list: 3} [7, 6, 4]
> 1 18 = {list: 4} [2, 0, 3, 8]
> 1 19 = {list: 3} [9, 10, 13]
```

נמצא מסולים לפי DIAMOND :

```
> 2 00 = {list: 4} [1, 2, 0, 3]
> 1 01 = {list: 5} [4, 5, 13, 10, 11]
> 1 02 = {list: 4} [7, 1, 2, 0]
> 1 03 = {list: 5} [1, 7, 6, 4, 5]
> 1 04 = {list: 3} [5, 2, 0]
> 1 05 = {list: 5} [11, 12, 9, 10, 7]
> 1 06 = {list: 6} [9, 10, 13, 5, 2, 0]
> 1 07 = {list: 4} [12, 11, 10, 13]
> 1 08 = {list: 3} [8, 3, 0]
> 1 09 = {list: 5} [11, 10, 13, 5, 12]
> 1 10 = {list: 5} [13, 10, 7, 6, 4]
> 1 11 = {list: 2} [7, 1]
> 1 12 = {list: 6} [1, 2, 5, 4, 3, 0]
> 1 13 = {list: 5} [1, 0, 3, 8, 11]
> 1 14 = {list: 4} [7, 1, 2, 0]
> 1 15 = {list: 4} [0, 2, 5, 13]
> 1 16 = {list: 5} [5, 12, 9, 8, 3]
> 1 17 = {list: 5} [7, 10, 13, 5, 4]
> 1 18 = {list: 4} [2, 0, 3, 8]
> 1 19 = {list: 5} [9, 12, 11, 10, 13]
```

כעת, מכיוון שיש הרבה יותר משתמשים שורצים לשדר, יש הרבה יותר הפרעות שיש להתחשב בהם. ניתן לשים לב שהמסלולים ארוכים יותר כדי למנוע הפרעות בlienks סמוכים.

נريץ את TDMA לקבالت סט הקצבים:

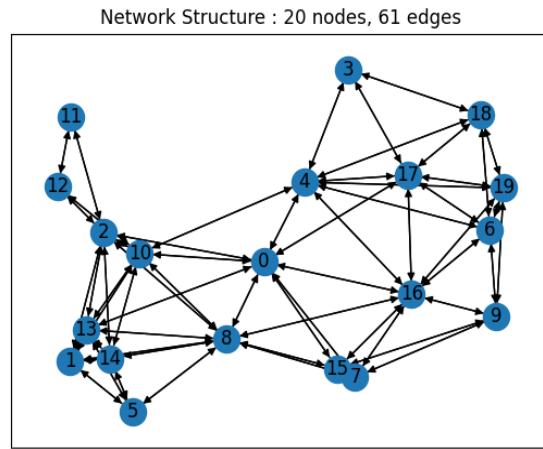


פעם נוספת נקבע שהאלג' ש商量א לאפשרות שידורים על אותו link יביא לסת קצבים גובה יותר.

השוואה עם שאלה 7:

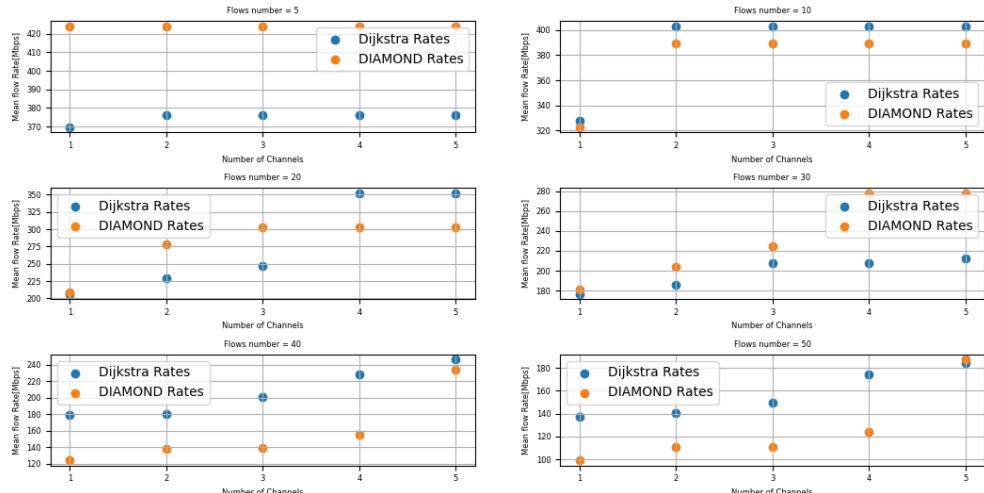
זכור שבשלב הסימולטור בשאלה 7 חישבנו קצבים לflows ברשת בדומה לשאלה 6 אך ההבדל היה שבעת בכל link, משתמשים יכולים לשדר באחד מתוך K ערוצים אורתוגונליים שונים. מימשנו את בחירת הערוצים ע"י מעבר על כל המשתמשים שעושים שימוש באותו link במסלול שלהם ודאגו שישדרו בערוצים שונים כך שהשידור יוכל להתבצע בו-זמןית ולא נctrarr לחלק את קיבול link. בהשוואה עם אלג' DIAMOND קיבל את המסלולים ברשת ע"י האלג' הנ"ל במקום דיקסטרא בשככת הרשת. כמו שראינו בהשוואה קודמות נصفה שהמסלולים יהיו לעתים יותר ארוכים אך כוללם פלוט הפרעה ברשת, שזו מטרת כותב המאמר.

נעsha את אותם חישובים כמו בשאלה 7 בשלב הסימולטור רק שהמסלולים יהיו שונים. ראשית על הרשת:



להלן התוצאות שקיבלו:

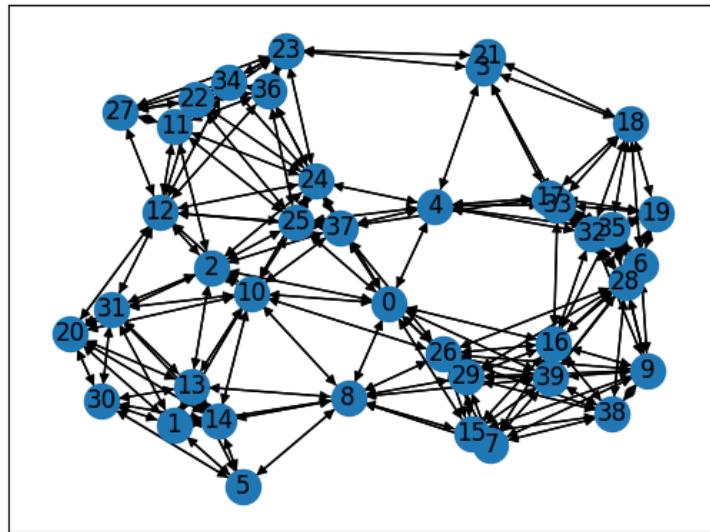
Mean Flow Rate with different K Dijkstra VS DIAMOND



ראשית נשים לב שכל שמספר ערוצי השידור האופציונליים גובה יותר כRK המוצע גבוה יותר. העניין בא לידי ביטוי בכך כאשר מספר הזרועות גדול יותר ולכן יותר משתמשים שעושים שימוש בlianekim משותפים, כמו כן גם הקצב המוצע יורד ככל שמספר הזרועות גדולים יותר. מבחינת השוואת התוצאות של מסלולים מדijkstra אל מול DIAMOND ניתן לראות שכאר מסוף הזרועות גובה יותר הקצב המוצע הגבוה יותר נוטה לטובות מסווגים שהתקבלו מדijkstra. האלגוריתם שם בעדיפות ראשונה פחות הפרעות בראשת لكن לא בהכרח מסתכל על קיבול הלינקיהם שעושים בו שימוש, אך יתכן ויקח מסלולים שבהם bottleneck יהיה נמוך יותר וכך הקצב המוצע. ניתן לראות ששאר העומס ביןוני DIAMOND יהיה דווקא עדיף בהרבה מהמקרים. קשה לראות חוקיות ברורה כי הבחירה של האלג'ריטהム לעמום לכך לפעמים צואර הבקבוק יצא נמוך יותר ולפעמים גובה יותר.

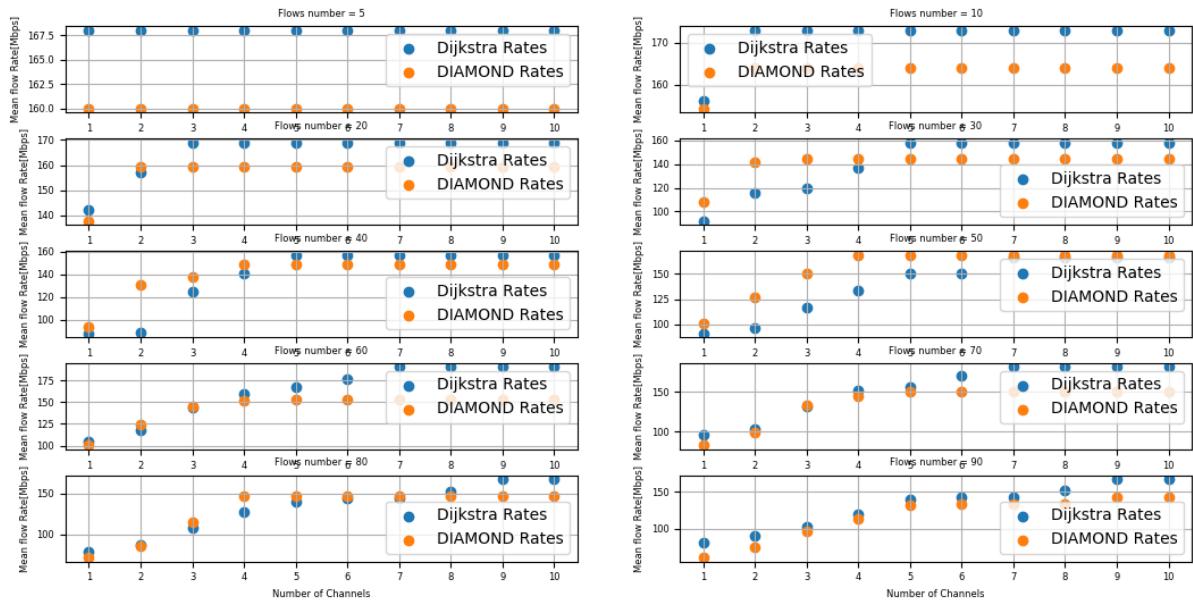
השוואה עם רשת 2:

Network Structure : 40 nodes, 183 edges



נקודות:

Mean Flow Rate with different K



ברשת יותר סבוכה זו קיבל תוצאות מאד דומות בין האלגוריתמים. האפשרויות לflows שונים גדולים יותר וכן הרבה פעמים דרך קצירה ודרך עם פחות הפרעות לאו דווקא יבואו אחד על חשבון השני. עם נוספת ניתן שכך מספר העריצים האופציונליים גבוה יותר הקצב הממוצע יהיה גבוה יותר בהתאם בשני האלגוריתמים.

מבחןיה תיאורית היא מוצפים לקבל תוצאות יותר חד-משמעיות ו纠וני בין האלג'. יתכן שבנויות הסימולטור שלנו נבדلت מעט מהסבירה שעליה אנחנו מרכיבים לקבלת מסלולים מאלג' DIAMOND ומשם נובעת סטיית תקן. יתכן שהסבירה לכך שאנו לא רואים חוקיות ברורה היא באלג' DIAMOND כותב המאמר לא לפקח בחשבון אפשרות לשידור ב-**B** עrozים אורטוגונליים, בעית האופטימיזציה לא לוחחת זאת בחשבון لكن פلت המסלולים בהתאם.

6. מסקנות וביקורת על המאמר:

אחד מהנקודות החזקות של המאמר היא שהאלגוריתם המוצע במאמר כולל שלב מבוזר ושלב ריכוזי. מאפיין של טכנולוגיות G5 ומעברו, העובדות עם נקודות חישוב ריכוזיות. מאפיין זה מאפשר למש את אלגוריתם הניתוב במערכות "אמיטיות" יחד עם טכנולוגיה מתאימה ולהביא אותה לידי שימוש. בנוסף מובאת במאמר גישה חדשנית המשלבת כליל למידה عمוקה שנמצאים בשנים האחרונות בתפתחות בלתי נגמרה. חוקרים רבים יכולים לחתה השראה ממאמר זהה ולפתח אותו בעקבות ארכיטקטורות מתקדמות שככל שנה יוצאות יכולות לשפר את יכולות האלגוריתם. בנוסף האלגוריתם זוכה לביצועים מעולים לעומת מתחריו כאשר מספר הflows בראש עולה, בעיקר בגלל שמתחשב בהפרעות לעומת אלגוריתמים כמו OSPF . הרבה פעמים יש רשות מרובות הפרעה כאשר התשתית לא טובה, לדוגמה אתרים צבאיים. במקומות כאלה יש צורך בפרטוקול יותר מתוחכם אשר מודע לביעות ומתחשב בהפרעות שונות במהלך השידור.

נקודת פחד חזקה באלגוריתם שפותח במאמר היא שהאלג' מקבל בכל נקודת זמן N דרישות לflows ומחריר N מסלולים. אין התייחסות לכך שם באמרflows מסוימים כמו המידע הנדרש להעביר קטנה יותר, הם יסימו לפני אחרים ויכנסו לרשף דרישות חדשות. האלגוריתם כרגע לא מטפל במקרים כאלה, חוקרים שניסו להמשיך את הכותב עלולים להסתכל על נקודת זו ולעבוד עליה.

בנוסף, האלגוריתם מניח שככל דרישת מתקיימת, כולם אם נתנו מסלול לשאflow מסוים בעל דרישת למיניהם, המידע יגיע לעד, אין התייחסות למודל הסתברותי של לינקים בדרך שייאלצו לקחת מסלול חלופי בראשת. כולם יש דרכים שאומנם יפריעו פחות לאחרים אבל ההסתברות של נפילת לינקים יותר גדולה لكن אולי עדיף מסלול אחר. יתכן שחוקרים שימושו את המאמר ינסו לבנות מודל הסתברותי שוכל להביא מיד נסוף בשיקול בבחירה הדרכים.

נקודת נוספת שאין אליה התייחסות באלגוריתם היא تعدוף חבילות מסוימות על אחרות. כולם מトン priority לכל flow . כרגע לכל ה-flows אותה חשיבות לכך האלגוריתם יטפל במידע שיש להעביר לפि הסדר. בראשת של היום יש חבילות בעלות חשיבות יותר גבוהה, יותר רגשות latency . لكن חוקרים עתידיים עשויים לתת לקלט האלג', בנוסף ל- מוקור, יעד, כמות DATA אולג' גם מספר تعدוף או אפילו דומה כך שהמידע שMOVIBER על המסלול הנבחר לו יעבור מהר לאט יותר מאשר האחרים. נקודת נוספת היא ערזוי השידור. במאמר יש הנחה שככל משתמשים מסדרים על אותו ערז תקשורת. בפועל יכולים להיות כמה ערזים אורטוגונליים כך שבעל לינק יוכל לבחור לשדר על ערז אחר. כאשר שני משתמשים מסדרים על אותו לינק בערזים שונים, ניתן להניח שאין הפרעה ביניהם כי גלי הנושא אינם "מתנגשים" וכן קיבול הלינק לא יצטרך להתחלק. חוקרים עתידיים יכולים לנסות לקחת את נקודת זו ולבחוין איך בבחירה המסלולים תשנה אם יש חופש בחירה של ערזים אורטוגונליים לשידור.

7. נספחין:

בחלק זה נזכיר כמה כלים מתמטיים, בעיקר מועלם ה- RL וה- GNN אשר יעזרו להבין יותר טוב את הנכתב במאמר בשלבים העשויים שימוש באלגוריתמים מועלמות הללו.

אלגוריתם : REINFORCE

The REINFORCE algorithm

Input:

A differentiable policy parameterization $\pi(a|s, \theta)$

Initialized policy parameters $\theta \in \mathbb{R}$

Begin:

1: **Repeat** forever:

2: Generate an episode $S_0, A_0, R_0, \dots, S_{T-1}, A_{T-1}, R_{T-1}$, following $\pi(\cdot | \cdot, \theta)$

3: **For each** step of the episode $t = 0, \dots, T - 1$:

4: $G_t \leftarrow$ return from step t

5: $\theta \leftarrow \theta + \alpha \gamma^t G \nabla_\theta \ln \pi(A_t | S_t, \theta)$

REINFORCE with Baseline

- Because of our use of sampling, the REINFORCE algorithm might fluctuate and be slow to converge (i.e., high variance)
- One way to mitigate this problem is by including a baseline – a value that will reduce the size of the update
 - The most important thing: it must **not** vary with a
 - Otherwise, the baseline will be correlated with what we're trying to optimize
- $$\theta_{t+1} = \theta_t + \alpha((G_t - b(S_t)) \frac{\nabla_\theta \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)})$$
- One logical choice for $b(S_t)$: the state-value function $\hat{v}(S_t, w)$
 - w needs to be calculated by method other than policy gradients
 - Can also be updated by sampling
 - Rewards actions that outperform the “average” for the state

דוגמאות לשכבות בארכיטקטורות GNN מקובלות:

Graph Convolutional networks:

$$\mathbf{h}_i^{l+1} = \sigma \left(\mathbf{W}_1^l \mathbf{h}_i^l + \frac{1}{d_i} \sum_{j \in \mathcal{N}(i)} \mathbf{A}_{i,j} \mathbf{W}_2^l \mathbf{h}_j^l \right) = \sigma \left(\mathbf{W}_1^l \mathbf{h}_i^l + \mathbf{W}_2^l \left(\frac{1}{d_i} \sum_{j \in \mathcal{N}(i)} \mathbf{A}_{i,j} \mathbf{h}_j^l \right) \right).$$

Graph Sage:

$$\mathbf{h}_i^{l+1} = \sigma \left(\mathbf{W}_1^l \mathbf{h}_i^l + \text{Aggregate}^l \left(\{\mathbf{h}_j^l, j \in \mathcal{N}(i)\} \right) \right),$$

Graph attention networks:

$$\begin{aligned} \mathbf{h}_i^{l+1} &= \mathbf{h}_i^l + \text{ReLU} \left(\mathbf{W}_1^l \mathbf{h}_i^l + \sum_{j \in \mathcal{N}(i)} \mathbf{e}_{i,j}^l \odot \mathbf{W}_2^l \mathbf{h}_j^l \right). \\ \mathbf{e}_{i,j}^l &= \frac{\sigma(\hat{\mathbf{e}}_{i,j}^l)}{\sum_{j' \in \mathcal{N}(i)} \sigma(\hat{\mathbf{e}}_{i,j'}^l) + \epsilon}, \\ \hat{\mathbf{e}}_{i,j}^l &= \hat{\mathbf{e}}_{i,j}^{l-1} + \text{ReLU} \left(\mathbf{V}_1^l \mathbf{h}_i^{l-1} + \mathbf{V}_2^l \mathbf{h}_j^{l-1} + \mathbf{V}_3^l \hat{\mathbf{e}}_{i,j}^{l-1} \right), \end{aligned}$$

(ויקיפדיה) : Nash Eqalibrium

The Nash equilibrium is decision-making theorem within game theory that states a player can achieve the desired outcome by not deviating from their initial strategy. In the Nash equilibrium, each player's strategy is optimal when considering the decisions of other players

הגדרת רשותות שהובאו בשלב הסימולציות במאמר:

NSFNET: The NSFNET (National Science Foundation Network) was a program and network infrastructure that played a crucial role in the early development of the internet. It was funded by the National Science Foundation (NSF) in the United States and operated from the mid-1980s until the mid-1990s. NSFNET served as a backbone network, connecting various regional networks and academic institutions across the country.

GEANT2: GÉANT2 refers to the second phase of the GÉANT (Gigabit European Academic Network) project. GÉANT2 was a pan-European research and education network that aimed to provide high-capacity and advanced data communications infrastructure for the European research and education community.

8. סימוכין:

נוסחאות GNN ל��וחות משקפים בקורס 361.2.2260 "מבוא למידה عمودה" מעת ניר שלזינגר.
נוסחאות RL ל��וחות משקפים בקורס 371.2.5910 "למידה חזקית عمוקה" מעת גלעד צץ.