Федеральное агентство связи

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М. А. БОНЧ-БРУЕВИЧА» (СПбГУТ)

Факультет информационных технологий и программной инженерии Кафедра: Программная инженерия. Разработка программного обеспечения и приложений искусственного интеллекта в киберфизических системах

Курсовой проект

по дисциплине **«Математические модели в сетях связи»**

Тема: Расчет пропускной способности линий связи

Выполнил:
Студент группы ИКПИ-42
Терещенко М. А.

Подпись _____

Принял:
к.т.н., доцент кафедры СС и ПД
Гребенщикова А. А.
Подпись _____

Санкт-Петербург 2025

Оглавление

1. Цель работы

Определить необходимые пропускные способности линий связи в сети с заданной структурой, а также подобрать их оптимальные значения, обеспечивающие достижение требуемых показателей, в том числе соблюдение целевого уровня задержки.

2. Исходные данные

Параметры сети:

- Количество узлов в сети связи – **27**
- Интенсивность удельной абонентской нагрузки – **y_0 = 0,1 Эрл**
- Тип кодека – **G.711**
- Скорость потока для кодека G.711 – **a_0 = 85,6 кбит/с**
- Длина пакета – **L = 200 байт**
- Начальное требование к величине задержки – **T_0 = 100 мс = 0,1 с**
- Доля вызовов, обслуженных с гарантированным качеством – **q = 98 %** (p_0 = 0,02)

Таблица распределения абонентов по узлам сети:

Узел	Кол-во абонентов	Доля k_i
1	5207	0,036
2	9877	0,069
3	4981	0,035
4	7521	0,052
5	8779	0,061
6	7058	0,049
7	1003	0,007
8	5007	0,035
9	7912	0,055
10	1133	0,008

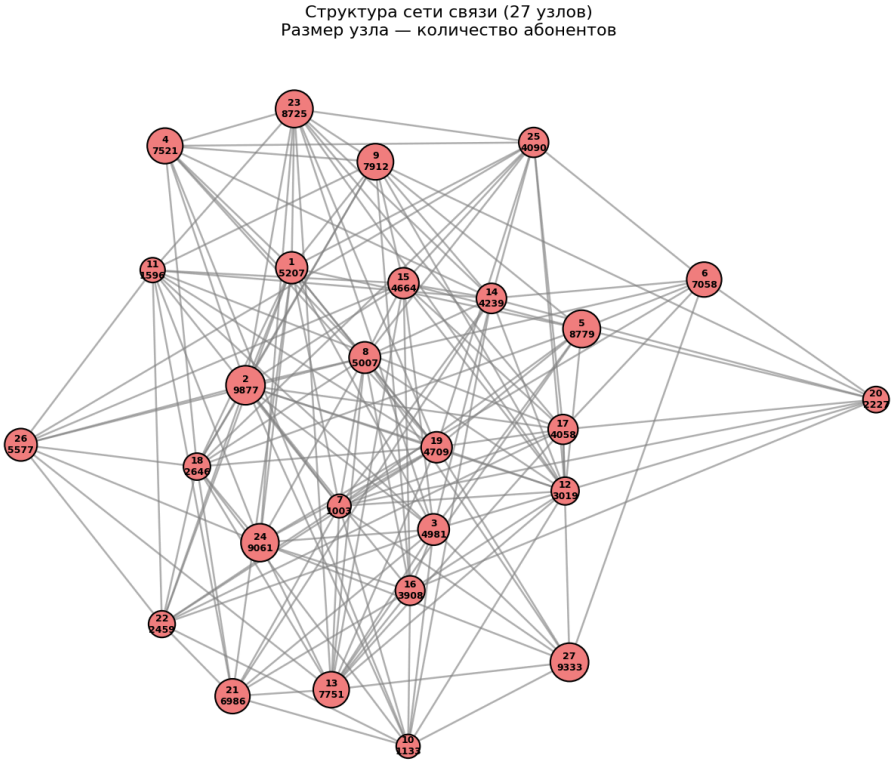
Узел	Кол-во абонентов	Доля k_i
11	1596	0,011
12	3019	0,021
13	7751	0,054
14	4239	0,030
15	4664	0,032
16	3908	0,027
17	4058	0,028
18	2646	0,018
19	4709	0,033
20	2227	0,016
21	6986	0,049
22	2459	0,017
23	8725	0,061
24	9061	0,063
25	4090	0,028
26	5577	0,039
27	9333	0,065

Матрица расстояний – индивидуальная, сгенерирована в excel файле для варианта 4222.

Матрица расстояний

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0,0	7,1	79,5	93,5	96,3		91,4	46,4	55,9									2,5	24,1		43,8	13,2	44,2	13,5	57,9		
2	7,1	0,0	72,5	12,0			48,2	22,7	29,1	14,3		91,8			72,3	68,6	34,0	70,6	14,1			24,6	75,9			89,4	
3	79,5	72,5	0,0		99,4					36,3	10,6	86,4	97,9	83,3	86,4	42,9					78,8	53,2	12,2	91,6			82,7
4	93,5	12,0		0,0				96,3	29,2	21,2				13,4				61,7	22,2				47,4		28,9		
5	96,3		99,4		0,0	85,0	48,0		22,4			84,9	10,3		52,6			49,7	78,5	13,9							
6					85,0	0,0		76,9						86,3			67,3			25,4				37,1	86,1		32,4
7	91,4	48,2			48,0		0,0	71,6		5,0	77,6	0,4	39,1	41,9			86,8		97,3	5,5	96,6	83,8					93,6
8	46,4	22,7		96,3		76,9	71,6	0,0				82,3	16,5	89,0	20,1	74,4		86,9	4,3					5,8	34,1	51,5	37,9
9	55,9	29,1		29,2	22,4				0,0		41,6	24,2		83,1	17,4	81,9		44,9		9,0			64,6				
10		14,3	36,3	21,2			5,0			0,0		97,4		12,6		76,3		92,5			34,0	49,4					91,6
11			10,6				77,6		41,6		0,0			75,4	85,7		12,8		7,2		3,9	5,2	68,9	63,0		56,9	
12		91,8	86,4		84,9		0,4	82,3	24,2	97,4		0,0	20,9	66,6	2,5	79,7	68,2		96,5	43,3			31,1		58,4		
13			97,9		10,3		39,1	16,5				20,9	0,0		32,2	70,7	19,1	58,1	84,1		39,4		4,9	53,1		26,3	11,0
14			83,3	13,4		86,3	41,9	89,0	83,1	12,6	75,4	66,6		0,0	50,6					40,0	65,7		43,7		95,2		
15		72,3	86,4		52,6			20,1	17,4		85,7	2,5	32,2	50,6	0,0	65,3	95,7								21,1	14,5	
16		68,6	42,9					74,4	81,9	76,3		79,7	70,7		65,3	0,0				10,4	31,1			79,1			
17		34,0				67,3	86,8				12,8	68,2	19,1		95,7		0,0		91,7	67,2		93,7	15,8		57,9		74,1
18	2,5	70,6		61,7	49,7			86,9	44,9	92,5			58,1					0,0	87,2		62,6	79,5		24,1	26,8	94,6	
19	24,1	14,1		22,2	78,5		97,3	4,3			7,2	96,5	84,1				91,7	87,2	0,0			17,5	44,9	45,7	19,1		99,5
20					13,9	25,4	5,5		9,0			43,3		40,0		10,4	67,2			0,0							
21	43,8		78,8				96,6			34,0	3,9		39,4	65,7		31,1		62,6			0,0	33,8					
22	13,2	24,6	53,2				83,8			49,4	5,2						93,7	79,5	17,5		33,8	0,0				76,2	
23	44,2	75,9	12,2	47,4					64,6		68,9	31,1	4,9	43,7			15,8		44,9				0,0	35,2	14,1		
24	13,5		91,6			37,1		5,8			63,0		53,1			79,1		24,1	45,7				35,2	0,0		47,8	60,6
25	57,9			28,9		86,1		34,1				58,4		95,2	21,1		57,9	26,8	19,1				14,1		0,0	14,7	
26		89,4						51,5			56,9		26,3		14,5			94,6				76,2		47,8	14,7	0,0	
27			82,7			32,4	93,6	37,9		91,6			11,0				74,1		99,5					60,6			0,0

Структура сети связи



Расчёт

3.1 Расчёт интенсивностей производимого в узлах сети трафика

Для расчёта интенсивности трафика необходимо умножить число абонентов в узле на y0

$$y_i = N_i \times y_0$$

Интенсивность трафика от узлов (yi):

n	y_i
1	520.7
2	987.7
3	498.1
4	752.1
5	877.9
6	705.8
7	100.3
8	500.7
9	791.2
10	113.3
11	159.6
12	301.9
13	775.1
14	423.9
15	466.4
16	390.8
17	405.8

n	y_i
18	264.6
19	470.9
20	222.7
21	698.6
22	245.9
23	872.5
24	906.1
25	409.0
26	557.7
27	933.3

3.2 Расчёт коэффициентов распределения трафика

Для определения вектора коэффициентов распределения трафика по направлениям необходимо разделить интенсивность трафика, генерируемого данным узлом, на суммарную интенсивность трафика, создаваемую всеми узлами сети.

$$k_{ij} = y_j / \Sigma y_i$$

n	k_ij
1	0.036279
2	0.068817
3	0.034705
4	0.052402
5	0.061167
6	0.049176
7	0.006988
8	0.034886

n	k_ij
9	0.055126
10	0.007894
11	0.011120
12	0.021035
13	0.054004
14	0.029535
15	0.032496
16	0.027229
17	0.028274
18	0.018436
19	0.032809
20	0.015516
21	0.048674
22	0.017133
23	0.060790
24	0.063131
25	0.028497
26	0.038857
27	0.065027

3.3 Расчёт интенсивностей трафика в направлениях связи

Матрица интенсивностей трафика в направлениях связи рассчитывается по формуле

$$\mathbf{Y} = [y_{i,j}], \quad i,j = 1,\ldots,n$$
$$y_{i,j} = k_{i,j} y_i, \quad i,j = 1,\ldots,n$$

Матрица интенсивностей трафика в направлениях связи Y

3. Матрица интенсивностей трафика

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	18,891	35,833	18,071	27,286	31,849	25,606	3,639	18,165	28,704	4,110	5,790	10,953	28,120	15,379	16,921	14,178	14,722	9,599	17,084	8,079	25,345	8,921	31,654	32,873	14,838	20,233	33,859
2	35,833	67,970	34,278	51,757	60,414	48,571	6,902	34,457	54,448	7,797	10,983	20,776	53,340	29,171	32,096	26,894	27,926	18,209	32,406	15,326	48,075	16,922	60,043	62,355	28,146	38,379	64,227
3	18,071	34,278	17,286	26,101	30,467	24,494	3,481	17,377	27,458	3,932	5,539	10,477	26,899	14,711	16,186	13,563	14,083	9,183	16,342	7,729	24,245	8,534	30,280	31,446	14,194	19,355	32,390
4	27,286	51,757	26,101	39,411	46,003	36,985	5,256	26,238	41,460	5,937	8,363	15,820	40,617	22,213	24,440	20,479	21,265	13,865	24,676	11,670	36,608	12,886	45,720	47,481	21,432	29,224	48,906
5	31,849	60,414	30,467	46,003	53,698	43,171	6,135	30,626	48,395	6,930	9,762	18,466	47,410	25,929	28,528	23,904	24,821	16,185	28,803	13,622	42,731	15,041	53,368	55,423	25,017	34,113	57,087
6	25,606	48,571	24,494	36,985	43,171	34,708	4,932	24,622	38,908	5,572	7,848	14,846	38,116	20,846	22,936	19,218	19,956	13,012	23,157	10,951	34,354	12,092	42,906	44,558	20,113	27,425	45,896
7	3,639	6,902	3,481	5,256	6,135	4,932	0,701	3,499	5,529	0,792	1,115	2,110	5,417	2,962	3,259	2,731	2,836	1,849	3,291	1,556	4,882	1,718	6,097	6,332	2,858	3,897	6,522
8	18,165	34,457	17,377	26,238	30,626	24,622	3,499	17,467	27,602	3,953	5,568	10,532	27,040	14,788	16,271	13,633	14,157	9,231	16,428	7,769	24,371	8,578	30,438	31,610	14,268	19,456	32,559
9	28,704	54,448	27,458	41,460	48,395	38,908	5,529	27,602	43,616	6,246	8,798	16,643	42,728	23,368	25,711	21,543	22,370	14,586	25,959	12,277	38,511	13,555	48,097	49,950	22,546	30,744	51,449
10	4,110	7,797	3,932	5,937	6,930	5,572	0,792	3,953	6,246	0,894	1,260	2,383	6,119	3,346	3,682	3,085	3,203	2,089	3,717	1,758	5,515	1,941	6,888	7,153	3,229	4,403	7,368
11	5,790	10,983	5,539	8,363	9,762	7,848	1,115	5,568	8,798	1,260	1,775	3,357	8,619	4,714	5,186	4,346	4,512	2,942	5,236	2,476	7,768	2,734	9,702	10,076	4,548	6,202	10,378
12	10,953	20,776	10,477	15,820	18,466	14,846	2,110	10,532	16,643	2,383	3,357	6,350	16,304	8,917	9,810	8,220	8,536	5,566	9,905	4,684	14,695	5,172	18,353	19,059	8,603	11,731	19,632
13	28,120	53,340	26,899	40,617	47,410	38,116	5,417	27,040	42,728	6,119	8,619	16,304	41,859	22,892	25,188	21,105	21,915	14,289	25,431	12,027	37,727	13,280	47,119	48,933	22,088	30,118	50,402
14	15,379	29,171	14,711	22,213	25,929	20,846	2,962	14,788	23,368	3,346	4,714	8,917	22,892	12,520	13,775	11,542	11,985	7,815	13,908	6,577	20,633	7,263	25,769	26,761	12,080	16,472	27,565
15	16,921	32,096	16,186	24,440	28,528	22,936	3,259	16,271	25,711	3,682	5,186	9,810	25,188	13,775	15,156	12,699	13,187	8,598	15,302	7,237	22,702	7,991	28,353	29,444	13,291	18,123	30,328
16	14,178	26,894	13,563	20,479	23,904	19,218	2,731	13,633	21,543	3,085	4,346	8,220	21,105	11,542	12,699	10,641	11,049	7,205	12,822	6,064	19,022	6,695	23,757	24,672	11,136	15,185	25,412
17	14,722	27,926	14,083	21,265	24,821	19,956	2,836	14,157	22,370	3,203	4,512	8,536	21,915	11,985	13,187	11,049	11,473	7,481	13,314	6,297	19,752	6,952	24,669	25,619	11,564	15,768	26,388
18	9,599	18,209	9,183	13,865	16,185	13,012	1,849	9,231	14,586	2,089	2,942	5,566	14,289	7,815	8,598	7,205	7,481	4,878	8,681	4,106	12,879	4,533	16,085	16,705	7,540	10,282	17,206
19	17,084	32,406	16,342	24,676	28,803	23,157	3,291	16,428	25,959	3,717	5,236	9,905	25,431	13,908	15,302	12,822	13,314	8,681	15,450	7,307	22,921	8,068	28,626	29,729	13,419	18,298	30,621
20	8,079	15,326	7,729	11,670	13,622	10,951	1,556	7,769	12,277	1,758	2,476	4,684	12,027	6,577	7,237	6,064	6,297	4,106	7,307	3,455	10,840	3,815	13,538	14,059	6,346	8,653	14,481
21	25,345	48,075	24,245	36,608	42,731	34,354	4,882	24,371	38,511	5,515	7,768	14,695	37,727	20,633	22,702	19,022	19,752	12,879	22,921	10,840	34,004	11,969	42,468	44,104	19,908	27,146	45,428
22	8,921	16,922	8,534	12,886	15,041	12,092	1,718	8,578	13,555	1,941	2,734	5,172	13,280	7,263	7,991	6,695	6,952	4,533	8,068	3,815	11,969	4,213	14,948	15,524	7,007	9,555	15,990
23	31,654	60,043	30,280	45,720	53,368	42,906	6,097	30,438	48,097	6,888	9,702	18,353	47,119	25,769	28,353	23,757	24,669	16,085	28,626	13,538	42,468	14,948	53,040	55,082	24,863	33,903	56,736
24	32,873	62,355	31,446	47,481	55,423	44,558	6,332	31,610	49,950	7,153	10,076	19,059	48,933	26,761	29,444	24,672	25,619	16,705	29,729	14,059	44,104	15,524	55,082	57,203	25,821	35,208	58,921
25	14,838	28,146	14,194	21,432	25,017	20,113	2,858	14,268	22,546	3,229	4,548	8,603	22,088	12,080	13,291	11,136	11,564	7,540	13,419	6,346	19,908	7,007	24,863	25,821	11,655	15,893	26,596
26	20,233	38,379	19,355	29,224	34,113	27,425	3,897	19,456	30,744	4,403	6,202	11,731	30,118	16,472	18,123	15,185	15,768	10,282	18,298	8,653	27,146	9,555	33,903	35,208	15,893	21,671	36,265
27	33,859	64,227	32,390	48,906	57,087	45,896	6,522	32,559	51,449	7,368	10,378	19,632	50,402	27,565	30,328	25,412	26,388	17,206	30,621	14,481	45,428	15,990	56,736	58,921	26,596	36,265	60,689

3.4 Расчёт кратчайших расстояний и маршрутов между узлами сети

Для построения данной матрицы использован **алгоритм Флойда-Уоршелла**.

Получена **матрица кратчайших расстояний**:

4. Матрица кратчайших путей (расстояния)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0	7,1	29	19,1	45,8	50,6	26,4	19,3	36,2	21,4	18,4	26,8	35,8	32,5	29,3	42,3	31,2	2,5	21,2	31,9	22,3	13,2	40,7	13,5	29,3	43,8	46,8
2	7,1	0	31,9	12	38,7	50,2	19,3	18,4	29,1	14,3	21,3	19,7	34,9	25,4	22,2	35,2	34	9,6	14,1	24,8	25,2	20,3	39,8	20,6	33,2	36,7	45,9
3	29	31,9	0	40	27,4	60,5	38,4	22,1	49,8	36,3	10,6	38	17,1	48,9	40,5	42,9	23,4	31,5	17,8	41,3	14,5	15,8	12,2	27,9	26,3	41	28,1
4	19,1	12	40	0	45,6	57,1	26,2	26,5	29,2	21,2	29,4	26,6	43	13,4	29,1	42,1	42,2	21,6	22,2	31,7	33,3	32,3	43	32,3	28,9	43,6	54
5	45,8	38,7	27,4	45,6	0	39,3	19,4	26,8	22,4	24,4	38	19,8	10,3	37	22,3	24,3	29,4	48,3	31,1	13,9	41,9	43,2	15,2	32,6	29,3	36,6	21,3
6	50,6	50,2	60,5	57,1	39,3	0	30,9	42,9	34,4	35,9	54,4	31,3	43,4	48,5	33,8	35,8	62,5	53,1	47,2	25,4	58,3	59,6	48,3	37,1	54,9	48,3	32,4
7	26,4	19,3	38,4	26,2	19,4	30,9	0	23	14,5	5	34,5	0,4	21,3	17,6	2,9	15,9	40,4	28,9	27,3	5,5	38,4	39,6	26,2	28,8	24	17,4	32,3
8	19,3	18,4	22,1	26,5	26,8	42,9	23	0	37,5	28	11,5	22,6	16,5	39,9	20,1	38,9	24,3	21,8	4,3	28,5	15,4	16,7	21,4	5,8	23,4	34,6	27,5
9	36,2	29,1	49,8	29,2	22,4	34,4	14,5	37,5	0	19,5	41,6	14,9	32,7	32,1	17,4	19,4	51,8	38,7	41,8	9	45,5	46,8	37,6	43,3	38,5	31,9	43,7
10	21,4	14,3	36,3	21,2	24,4	35,9	5	28	19,5	0	35,6	5,4	26,3	12,6	7,9	20,9	45,4	23,9	28,4	10,5	34	34,6	31,2	33,8	29	22,4	37,3
11	18,4	21,3	10,6	29,4	38	54,4	34,5	11,5	41,6	35,6	0	34,1	27,7	42,8	31,6	35	12,8	20,9	7,2	40	3,9	5,2	22,8	17,3	26,3	41	38,7
12	26,8	19,7	38	26,6	19,8	31,3	0,4	22,6	14,9	5,4	34,1	0	20,9	18	2,5	16,3	40	29,3	26,9	5,9	38	39,3	25,8	28,4	23,6	17	31,9
13	35,8	34,9	17,1	43	10,3	43,4	21,3	16,5	32,7	26,3	27,7	20,9	0	38,9	23,4	34,6	19,1	38,3	20,8	24,2	31,6	32,9	4,9	22,3	19	26,3	11
14	32,5	25,4	48,9	13,4	37	48,5	17,6	39,9	32,1	12,6	42,8	18	38,9	0	20,5	33,5	55,6	35	35,6	23,1	46,6	45,7	43,7	45,7	41,6	35	49,9
15	29,3	22,2	40,5	29,1	22,3	33,8	2,9	20,1	17,4	7,9	31,6	2,5	23,4	20,5	0	18,8	42,5	31,8	24,4	8,4	35,5	36,8	28,3	25,9	21,1	14,5	34,4
16	42,3	35,2	42,9	42,1	24,3	35,8	15,9	38,9	19,4	20,9	35	16,3	34,6	33,5	18,8	0	47,8	44,8	42,2	10,4	31,1	40,2	39,5	44,7	39,9	33,3	45,6
17	31,2	34	23,4	42,2	29,4	62,5	40,4	24,3	51,8	45,4	12,8	40	19,1	55,6	42,5	47,8	0	33,7	20	43,3	16,7	18	15,8	30,1	29,9	44,6	30,1
18	2,5	9,6	31,5	21,6	48,3	53,1	28,9	21,8	38,7	23,9	20,9	29,3	38,3	35	31,8	44,8	33,7	0	23,7	34,4	24,8	15,7	40,9	16	26,8	41,5	49,3
19	21,2	14,1	17,8	22,2	31,1	47,2	27,3	4,3	41,8	28,4	7,2	26,9	20,8	35,6	24,4	42,2	20	23,7	0	32,8	11,1	12,4	25,7	10,1	19,1	33,8	31,8
20	31,9	24,8	41,3	31,7	13,9	25,4	5,5	28,5	9	10,5	40	5,9	24,2	23,1	8,4	10,4	43,3	34,4	32,8	0	41,5	45,1	29,1	34,3	29,5	22,9	35,2
21	22,3	25,2	14,5	33,3	41,9	58,3	38,4	15,4	45,5	34	3,9	38	31,6	46,6	35,5	31,1	16,7	24,8	11,1	41,5	0	9,1	26,7	21,2	30,2	44,9	42,6
22	13,2	20,3	15,8	32,3	43,2	59,6	39,6	16,7	46,8	34,6	5,2	39,3	32,9	45,7	36,8	40,2	18	15,7	12,4	45,1	9,1	0	28	22,5	31,5	46,2	43,9
23	40,7	39,8	12,2	43	15,2	48,3	26,2	21,4	37,6	31,2	22,8	25,8	4,9	43,7	28,3	39,5	15,8	40,9	25,7	29,1	26,7	28	0	27,2	14,1	28,8	15,9
24	13,5	20,6	27,9	32,3	32,6	37,1	28,8	5,8	43,3	33,8	17,3	28,4	22,3	45,7	25,9	44,7	30,1	16	10,1	34,3	21,2	22,5	27,2	0	29,2	40,4	33,3
25	29,3	33,2	26,3	28,9	29,3	54,9	24	23,4	38,5	29	26,3	23,6	19	41,6	21,1	39,9	29,9	26,8	19,1	29,5	30,2	31,5	14,1	29,2	0	14,7	30
26	43,8	36,7	41	43,6	36,6	48,3	17,4	34,6	31,9	22,4	41	17	26,3	35	14,5	33,3	44,6	41,5	33,8	22,9	44,9	46,2	28,8	40,4	14,7	0	37,3
27	46,8	45,9	28,1	54	21,3	32,4	32,3	27,5	43,7	37,3	38,7	31,9	11	49,9	34,4	45,6	30,1	49,3	31,8	35,2	42,6	43,9	15,9	33,3	30	37,3	0

Матрица следующего узла

Матрица следующих узлов (Next)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0	2	22	2	2	24	2	24	2	2	22	2	24	2	2	2	22	18	2	2	22	22	24	24	18	2	24
2	1	0	19	4	10	10	10	19	9	10	19	10	19	4	10	10	17	1	19	10	19	1	19	1	19	10	19
3	11	11	0	11	23	23	23	11	23	10	11	23	23	10	23	16	11	11	11	23	11	11	23	11	23	23	23
4	2	2	19	0	10	10	10	19	9	10	19	10	19	14	10	10	19	2	19	10	19	2	25	19	25	25	19
5	20	20	13	20	0	20	20	13	9	20	13	20	13	20	20	20	13	20	13	20	13	13	13	13	13	13	13
6	24	20	27	20	20	0	20	24	20	20	24	20	27	20	20	20	27	24	24	20	24	24	27	24	20	20	27
7	10	10	12	10	20	20	0	12	20	10	12	12	12	10	12	20	12	10	12	20	12	10	12	12	12	12	12
8	24	19	19	19	13	24	15	0	15	15	19	15	13	19	15	15	19	24	19	15	19	19	13	24	19	15	13
9	2	2	5	4	5	20	20	15	0	20	11	20	5	20	15	20	5	2	15	20	11	11	5	15	15	15	5
10	2	2	3	4	7	7	7	7	7	0	2	7	7	14	7	7	7	2	2	7	21	2	7	7	7	7	7
11	22	19	3	19	3	19	19	19	9	19	0	19	3	19	19	21	17	22	19	19	21	22	3	19	19	19	3
12	7	7	13	7	7	7	7	15	7	7	15	0	13	7	15	7	13	7	15	7	15	15	13	15	15	15	13
13	8	8	23	8	5	27	12	8	5	12	23	12	0	12	12	5	17	8	8	5	23	23	23	8	23	26	27
14	4	4	10	4	10	10	10	4	10	10	4	10	10	0	10	10	4	4	4	10	10	4	23	4	10	10	10
15	12	12	12	12	12	12	12	8	9	12	8	12	12	12	0	12	12	12	8	12	8	8	12	8	25	26	12
16	20	20	3	20	20	20	20	20	20	20	21	20	20	20	20	0	21	20	21	20	21	21	20	20	20	20	20
17	11	2	11	11	13	13	13	11	13	13	11	13	13	11	13	11	0	11	11	13	11	11	23	11	23	23	13
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	25	1	25	25	1
19	2	2	11	4	8	8	8	8	8	2	11	8	8	4	8	11	11	2	0	8	11	11	8	8	25	25	8
20	7	7	5	7	5	6	7	7	9	7	7	7	5	7	7	16	5	7	7	0	16	7	5	7	7	7	5
21	11	11	11	11	11	11	11	11	11	10	11	11	11	10	11	16	11	11	11	16	0	11	11	11	11	11	11
22	1	1	11	1	11	11	1	11	11	1	11	11	11	1	11	11	11	1	11	1	11	0	11	11	11	11	11
23	13	13	3	25	13	13	13	13	13	13	3	13	13	14	13	13	17	25	13	13	3	3	0	13	25	25	13
24	1	1	8	8	8	6	8	8	8	8	8	8	8	8	8	8	8	1	8	8	8	8	8	0	8	8	8
25	18	19	23	4	23	15	15	19	15	15	19	15	23	15	15	15	23	18	19	15	19	19	23	19	0	26	23
26	15	15	25	25	13	15	15	15	15	15	25	15	13	15	15	15	25	25	25	15	25	25	25	15	25	0	13
27	13	13	13	13	13	6	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	0

Матрица содержит номера узлов, выступающих в качестве промежуточных точек на маршруте между парой узлов, обозначенной строкой и столбцом.

3.5 Расчёт интенсивностей нагрузок на линиях связи

Необходимо последовательно обработать всю матрицу интенсивностей трафика по направлениям. Для каждого направления маршрут определяется с использованием матрицы кратчайших путей, после чего соответствующее значение интенсивности прибавляется к каждой линии связи, входящей в найденный маршрут.

Матрица нагрузок:

5. Матрица нагрузок на линии связи Y_tilde

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0	449,901	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240,653	0	0	0	154,413	0	303,074	0	0	0
2	449,901	0	0	165,421	0	0	0	0	97,7382	425,166	0	0	0	0	0	0	27,9259	0	396,696	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	18,6433	471,272	0	0	0	0	13,5625	0	0	0	0	0	0	509,485	0	0	0	0
4	0	165,421	0	0	0	0	0	0	41,4602	166,59	0	0	0	153,997	0	0	0	0	352,411	0	0	0	0	0	96,3771	0	0
5	0	0	0	0	0	0	0	0	240,498	0	0	0	771,118	0	0	0	0	0	0	445,482	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	314,474	0	0	0	185,25	0	0	171,368
7	0	0	0	0	0	0	0	0	0	830,589	0	692,492	0	0	0	0	0	0	0	777,53	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	844,763	0	417,932	0	0	0	1020,19	0	0	0	0	947,005	0	0	0
9	0	97,7382	0	41,4602	240,498	0	0	0	0	0	60,8645	0	0	0	182,511	0	0	0	0	124,513	0	0	0	0	0	0	0
10	0	425,166	18,6433	166,59	0	0	830,589	0	0	0	0	0	0	231,614	0	0	0	0	0	0	26,1477	0	0	0	0	0	0
11	0	0	471,272	0	0	0	0	0	60,8645	0	0	0	0	0	0	0	164,892	0	675,39	0	643,499	280,1	0	0	0	0	0
12	0	0	0	0	0	0	692,492	0	0	0	0	0	284,93	0	670,829	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	771,118	0	0	844,763	0	0	0	284,93	0	0	0	0	149,508	0	0	0	0	0	1003,29	0	0	100,496	952,187
14	0	0	0	153,997	0	0	0	0	0	231,614	0	0	0	0	0	0	0	0	0	0	0	0	25,769	0	0	0	0
15	0	0	0	0	0	0	0	417,932	182,511	0	0	670,829	0	0	0	0	0	0	0	0	0	0	0	0	100,203	249,909	0
16	0	0	13,5625	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	323,502	64,774	0	0	0	0	0	0
17	0	27,9259	0	0	0	0	0	0	0	0	164,892	0	149,508	0	0	0	0	0	0	0	0	0	52,0008	0	0	0	0
18	240,653	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48,7451	0	0
19	0	396,696	0	352,411	0	0	0	1020,19	0	0	675,39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	174,317	0
20	0	0	0	0	445,482	314,474	777,53	0	124,513	0	0	0	0	0	0	323,502	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	26,1477	643,499	0	0	0	0	64,774	0	0	0	0	0	0	0	0	0	0	0
22	154,413	0	0	0	0	0	0	0	0	0	280,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	509,485	0	0	0	0	0	0	0	0	0	1003,29	25,769	0	0	52,0008	0	0	0	0	0	0	0	255,153	0	0
24	303,074	0	0	0	0	185,25	0	947,005	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	96,3771	0	0	0	0	0	0	0	0	0	0	100,203	0	0	48,7451	174,317	0	0	0	255,153	0	0	185,624	0
26	0	0	0	0	0	0	0	0	0	0	0	0	100,496	0	249,909	0	0	0	0	0	0	0	0	0	185,624	0	0
27	0	0	0	0	0	171,368	0	0	0	0	0	0	952,187	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3.6 Расчёт количества потоков в линиях связи

Требуемое число потоков рассчитывается из формулы Эрланга и необходимого процента обслуженных заявок

$$\mathbf{V} = [\mathbf{v}_{i,j}], \quad i,j = 1,\ldots,n$$
$$\mathbf{v}_{i,j} = \arg\min_{\mathbf{v}_{i,j}} \left|\rho\left(\widetilde{\mathbf{y}}_{i,j}, \mathbf{v}_{i,j}\right) - \boldsymbol{\rho}_0\right|,$$
$$\rho\left(\widetilde{\mathbf{y}}_{i,j}, \mathbf{v}_{i,j}\right) \le \boldsymbol{\rho}_0$$

$$p\left(\widetilde{\mathbf{y}}_{i,j}, \mathbf{v}_{i,j}\right) = \frac{\widetilde{\mathbf{y}}_{i,j}^{\mathbf{v}_{i,j}} / \quad \mathbf{v}_{i,j}!}{\sum_{k=0}^{\mathbf{v}_{i,j}} \frac{\widetilde{\mathbf{y}}_{i,j}^{k}}{k!}}$$

По формуле Эрланга B с p_0 = 0,02 получена матрица **количества потоков**:

6. Матрица требуемого числа каналов V[i,j]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0	464	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	0	0	0	168	0	318	0	0	0
2	464	0	0	180	0	0	0	0	111	440	0	0	0	0	0	0	37	0	411	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	27	485	0	0	0	0	21	0	0	0	0	0	0	523	0	0	0	0
4	0	180	0	0	0	0	0	0	52	181	0	0	0	168	0	0	0	0	367	0	0	0	0	0	109	0	0
5	0	0	0	0	0	0	0	0	255	0	0	0	782	0	0	0	0	0	0	460	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	329	0	0	0	200	0	0	186
7	0	0	0	0	0	0	0	0	0	841	0	705	0	0	0	0	0	0	0	789	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	855	0	432	0	0	0	1028	0	0	0	0	956	0	0	0
9	0	111	0	52	255	0	0	0	0	0	72	0	0	0	197	0	0	0	0	138	0	0	0	0	0	0	0
10	0	440	27	181	0	0	841	0	0	0	0	0	0	246	0	0	0	0	0	0	35	0	0	0	0	0	0
11	0	0	485	0	0	0	0	0	72	0	0	0	0	0	0	0	179	0	688	0	656	295	0	0	0	0	0
12	0	0	0	0	0	0	705	0	0	0	0	0	300	0	683	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	782	0	0	855	0	0	0	300	0	0	0	0	163	0	0	0	0	0	1012	0	0	113	961
14	0	0	0	168	0	0	0	0	0	246	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	0	0
15	0	0	0	0	0	0	0	432	197	0	0	683	0	0	0	0	0	0	0	0	0	0	0	0	113	265	0
16	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	338	76	0	0	0	0	0	0
17	0	37	0	0	0	0	0	0	0	0	179	0	163	0	0	0	0	0	0	0	0	0	63	0	0	0	0
18	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0
19	0	411	0	367	0	0	0	1028	0	0	688	0	0	0	0	0	0	0	0	0	0	0	0	0	189	0	0
20	0	0	0	0	460	329	789	0	138	0	0	0	0	0	0	338	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	35	656	0	0	0	0	76	0	0	0	0	0	0	0	0	0	0	0
22	168	0	0	0	0	0	0	0	0	0	295	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	523	0	0	0	0	0	0	0	0	0	1012	35	0	0	63	0	0	0	0	0	0	0	270	0	0
24	318	0	0	0	0	200	0	956	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	109	0	0	0	0	0	0	0	0	0	0	113	0	0	60	189	0	0	0	270	0	0	200	0
26	0	0	0	0	0	0	0	0	0	0	0	0	113	0	265	0	0	0	0	0	0	0	0	0	200	0	0
27	0	0	0	0	0	186	0	0	0	0	0	0	961	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3.7 Расчёт интенсивностей трафика ПД для линий связи

Для определения интенсивности трафика на линиях связи необходимо умножить базовую интенсивность $a0$ на матрицу количества потоков.

$a_{ij} = v_{ij} \times a_0 = v_{ij} \times 85600$ бит/с:

7. Матрица интенсивности трафика ПД A[i,j] (бит/с)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0	39718400	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21828000	0	0	0	14380800	0	27220800	0	0	0
2	39718400	0	0	15408000	0	0	0	0	9501600	37664000	0	0	0	0	0	0	3167200	0	35181600	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	2311200	41516000	0	0	0	0	1797600	0	0	0	0	0	0	4476880	0	0	0	0
4	0	15408000	0	0	0	0	0	0	4451200	15493600	0	0	0	14380800	0	0	0	0	31415200	0	0	0	0	0	9330400	0	0
5	0	0	0	0	0	0	0	0	21828000	0	0	0	66939200	0	0	0	0	0	0	39376000	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28162400	0	0	0	17120000	0	0	15921600
7	0	0	0	0	0	0	0	0	0	71989600	0	60348000	0	0	0	0	0	0	0	67538400	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	73188000	0	36979200	0	0	0	87996800	0	0	0	0	81833600	0	0	0
9	0	9501600	0	4451200	21828000	0	0	0	0	0	6163200	0	0	0	16863200	0	0	0	0	11812800	0	0	0	0	0	0	0
10	0	37664000	2311200	15493600	0	0	71989600	0	0	0	0	0	0	21057600	0	0	0	0	0	0	2996000	0	0	0	0	0	0
11	0	0	41516000	0	0	0	0	0	6163200	0	0	0	0	0	0	0	15322400	0	58892800	0	56153600	25252000	0	0	0	0	0
12	0	0	0	0	0	0	60348000	0	0	0	0	0	25680000	0	58464800	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	66939200	0	0	0	73188000	0	0	0	0	25680000	0	0	0	13952800	0	0	0	0	0	86627200	0	0	9672800	82261600
14	0	0	0	14380800	0	0	0	0	0	21057600	0	0	0	0	0	0	0	0	0	0	0	0	2996000	0	0	0	0
15	0	0	0	0	0	0	0	36979200	16863200	0	0	58464800	0	0	0	0	0	0	0	0	0	0	0	0	0	9672800	22684000
16	0	0	1797600	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28932800	6505600	0	0	0	0	0	0
17	0	3167200	0	0	0	0	0	0	0	0	15322400	0	13952800	0	0	0	0	0	0	0	0	0	5392800	0	0	0	0
18	21828000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5136000	0	0	0
19	0	35181600	0	31415200	0	0	0	87996800	0	0	58892800	0	0	0	0	0	0	0	0	0	0	0	0	16178400	0	0	0
20	0	0	0	0	39376000	28162400	67538400	0	11812800	0	0	0	0	0	0	28932800	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	2996000	56153600	0	0	0	0	6505600	0	0	0	0	0	0	0	0	0	0	0
22	14380800	0	0	0	0	0	0	0	0	0	25252000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	44768800	0	0	0	0	0	0	0	0	0	86627200	2996000	0	0	5392800	0	0	0	0	0	0	0	23112000	0	0
24	27220800	0	0	0	17120000	0	81833600	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	9330400	0	0	0	0	0	0	0	0	0	0	9672800	0	0	5136000	16178400	0	0	0	23112000	0	0	17120000	0
26	0	0	0	0	0	0	0	0	0	0	0	0	9672800	0	22684000	0	0	0	0	0	0	0	0	0	17120000	0	0
27	0	0	0	0	0	15921600	0	0	0	0	0	0	82261600	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3.8 Расчёт пропускной способности линий связи

Для вычисления требуемой пропускной способности линий связи к значениям матрицы трафика добавляется величина L / T_0

Матрица пропускных способностей (фрагмент, бит/с)

8. Пропускная способность линий B[i,j]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0	39734400	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21844000	0	0	0	14396800	0	27236800	0	0	0
2	39734400	0	0	15424000	0	0	0	0	9517600	37680000	0	0	0	0	0	0	3183200	0	35197600	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	2327200	41532000	0	0	0	0	1813600	0	0	0	0	0	0	44784800	0	0	0	0
4	0	15424000	0	0	0	0	0	0	4467200	15509600	0	0	0	14396800	0	0	0	0	31431200	0	0	0	0	0	9346400	0	0
5	0	0	0	0	0	0	0	0	21844000	0	0	0	66955200	0	0	0	0	0	0	39392000	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28178400	0	0	0	17136000	0	0	15937600
7	0	0	0	0	0	0	0	0	0	72005600	0	60364000	0	0	0	0	0	0	0	67554400	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	73204000	0	36995200	0	0	0	88012800	0	0	0	0	81849600	0	0	0
9	0	9517600	0	4467200	21844000	0	0	0	0	0	6179200	0	0	0	16879200	0	0	0	0	11828800	0	0	0	0	0	0	0
10	0	37680000	2327200	15509600	0	0	72005600	0	0	0	0	0	0	21073600	0	0	0	0	0	0	3012000	0	0	0	0	0	0
11	0	0	41532000	0	0	0	0	0	6179200	0	0	0	0	0	0	0	15338400	0	58908800	0	56169600	25268000	0	0	0	0	0
12	0	0	0	0	0	0	60364000	0	0	0	0	0	25696000	0	58480800	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	66955200	0	0	73204000	0	0	0	25696000	0	0	0	0	13968800	0	0	0	0	0	86643200	0	0	9688800	82277600
14	0	0	0	14396800	0	0	0	0	0	21073600	0	0	0	0	0	0	0	0	0	0	0	0	3012000	0	0	0	0
15	0	0	0	0	0	0	0	36995200	16879200	0	0	58480800	0	0	0	0	0	0	0	0	0	0	0	0	9688800	22700000	0
16	0	0	1813600	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28948800	6521600	0	0	0	0	0
17	0	3183200	0	0	0	0	0	0	0	0	15338400	0	13968800	0	0	0	0	0	0	0	0	0	5408800	0	0	0	0
18	21844000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5152000	0	0	0
19	0	35197600	0	31431200	0	0	0	88012800	0	0	58908800	0	0	0	0	0	0	0	0	0	0	0	0	0	16194400	0	0
20	0	0	0	0	39392000	28178400	67554400	0	11828800	0	0	0	0	0	0	28948800	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	3012000	56169600	0	0	0	0	6521600	0	0	0	0	0	0	0	0	0	0	0
22	14396800	0	0	0	0	0	0	0	0	0	25268000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	44784800	0	0	0	0	0	0	0	0	0	86643200	3012000	0	0	5408800	0	0	0	0	0	0	0	23128000	0	0
24	27236800	0	0	0	0	17136000	0	81849600	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	9346400	0	0	0	0	0	0	0	0	0	0	968800	0	0	5152000	16194400	0	0	0	23128000	0	0	17136000	0
26	0	0	0	0	0	0	0	0	0	0	0	0	9688800	0	22700000	0	0	0	0	0	0	0	0	0	17136000	0	0
27	0	0	0	0	0	15937600	0	0	0	0	0	0	82277600	0	0	0	0	0	0	0	0	0	0	0	0	0	0

4 Оптимизация пропускной способности линий связи

Оптимизация пропускной способности направлена на подбор таких значений для линий связи, при которых сквозная задержка стремится к наиболее часто встречающемуся значению - **50 мс**. Для достижения этой цели используется итерационный жадный алгоритм с шагом (dc = 10,000) бит/с.

Алгоритм:

1. Берём начальную матрицу пропускных способностей b_{ij} (из пункта 3.8).
2. Поочерёдно увеличиваем каждую линию на dc и считаем целевую функцию **$O = \Sigma\ (t_{ij} – 0{,}05)^2$** по всем парам абонентов.
3. Выбираем линию, давшую наибольшее уменьшение O.
4. Фиксируем прирост, повторяем до стабилизации O.

Матрица задержек между узлами после оптимизации

Задержки после оптимизации DL (мс)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0,0000	0,0138	0,0563	0,0424	0,0727	0,0485	0,0398	0,0380	0,0582	0,0289	0,0377	0,0508	0,0591	0,0634	0,0618	0,0683	0,0620	0,0127	0,0424	0,0516	0,0528	0,0211	0,0729	0,0242	0,0571	0,0828	0,0729
2	0,0138	0,0000	0,0599	0,0286	0,0589	0,0621	0,0261	0,0424	0,0444	0,0151	0,0413	0,0370	0,0634	0,0496	0,0480	0,0545	0,0615	0,0265	0,0286	0,0378	0,0564	0,0348	0,0772	0,0380	0,0571	0,0690	0,0772
3	0,0563	0,0599	0,0000	0,0555	0,0456	0,0775	0,0641	0,0451	0,0741	0,0444	0,0186	0,0531	0,0289	0,0630	0,0641	0,0615	0,0428	0,0690	0,0313	0,0666	0,0337	0,0353	0,0151	0,0589	0,0393	0,0604	0,0427
4	0,0424	0,0286	0,0555	0,0000	0,0723	0,0755	0,0395	0,0380	0,0615	0,0286	0,0369	0,0505	0,0591	0,0211	0,0614	0,0680	0,0612	0,0551	0,0242	0,0513	0,0520	0,0634	0,0687	0,0518	0,0444	0,0655	0,0729
5	0,0727	0,0589	0,0456	0,0723	0,0000	0,0453	0,0328	0,0377	0,0286	0,0438	0,0642	0,0438	0,0167	0,0624	0,0547	0,0377	0,0409	0,0854	0,0515	0,0211	0,0793	0,0808	0,0305	0,0515	0,0547	0,0611	0,0305
6	0,0485	0,0621	0,0775	0,0755	0,0453	0,0000	0,0360	0,0380	0,0590	0,0470	0,0645	0,0470	0,0486	0,0656	0,0579	0,0409	0,0728	0,0612	0,0518	0,0242	0,0796	0,0812	0,0624	0,0242	0,0822	0,0790	0,0348
7	0,0398	0,0261	0,0641	0,0395	0,0328	0,0360	0,0000	0,0386	0,0465	0,0110	0,0651	0,0110	0,0352	0,0296	0,0219	0,0284	0,0594	0,0525	0,0524	0,0118	0,0802	0,0609	0,0490	0,0524	0,0462	0,0430	0,0490
8	0,0380	0,0424	0,0451	0,0380	0,0377	0,0380	0,0386	0,0000	0,0514	0,0495	0,0265	0,0276	0,0211	0,0591	0,0167	0,0670	0,0507	0,0507	0,0138	0,0503	0,0416	0,0432	0,0348	0,0138	0,0424	0,0377	0,0348
9	0,0582	0,0444	0,0741	0,0615	0,0286	0,0590	0,0465	0,0514	0,0000	0,0575	0,0444	0,0575	0,0452	0,0761	0,0348	0,0514	0,0695	0,0709	0,0652	0,0348	0,0595	0,0611	0,0590	0,0652	0,0590	0,0558	0,0590
10	0,0289	0,0151	0,0444	0,0286	0,0438	0,0470	0,0110	0,0495	0,0575	0,0000	0,0564	0,0219	0,0462	0,0186	0,0329	0,0394	0,0704	0,0416	0,0437	0,0227	0,0444	0,0499	0,0600	0,0633	0,0571	0,0539	0,0600
11	0,0377	0,0413	0,0186	0,0369	0,0642	0,0645	0,0651	0,0265	0,0444	0,0564	0,0000	0,0541	0,0475	0,0580	0,0432	0,0499	0,0242	0,0504	0,0127	0,0768	0,0151	0,0167	0,0337	0,0403	0,0413	0,0623	0,0613
12	0,0508	0,0370	0,0531	0,0505	0,0438	0,0470	0,0110	0,0276	0,0575	0,0219	0,0541	0,0000	0,0242	0,0405	0,0110	0,0394	0,0485	0,0635	0,0414	0,0227	0,0692	0,0708	0,0380	0,0414	0,0352	0,0320	0,0380
13	0,0591	0,0634	0,0289	0,0591	0,0167	0,0486	0,0352	0,0211	0,0452	0,0462	0,0475	0,0242	0,0000	0,0648	0,0352	0,0544	0,0242	0,0718	0,0348	0,0377	0,0626	0,0642	0,0138	0,0348	0,0380	0,0444	0,0138
14	0,0634	0,0496	0,0630	0,0211	0,0624	0,0656	0,0296	0,0591	0,0761	0,0186	0,0580	0,0405	0,0648	0,0000	0,0515	0,0580	0,0822	0,0761	0,0453	0,0413	0,0630	0,0845	0,0615	0,0729	0,0757	0,0725	0,0786
15	0,0618	0,0480	0,0641	0,0614	0,0547	0,0579	0,0219	0,0167	0,0348	0,0329	0,0432	0,0110	0,0352	0,0515	0,0000	0,0503	0,0594	0,0745	0,0305	0,0337	0,0583	0,0598	0,0490	0,0305	0,0242	0,0211	0,0490
16	0,0683	0,0545	0,0615	0,0680	0,0377	0,0409	0,0284	0,0670	0,0514	0,0394	0,0499	0,0394	0,0544	0,0580	0,0503	0,0000	0,0741	0,0810	0,0626	0,0167	0,0348	0,0665	0,0682	0,0808	0,0746	0,0714	0,0682
17	0,0620	0,0615	0,0428	0,0612	0,0409	0,0728	0,0594	0,0507	0,0695	0,0704	0,0242	0,0485	0,0242	0,0822	0,0594	0,0741	0,0000	0,0747	0,0369	0,0620	0,0393	0,0409	0,0348	0,0645	0,0590	0,0801	0,0380
18	0,0127	0,0265	0,0690	0,0551	0,0854	0,0612	0,0525	0,0507	0,0709	0,0416	0,0504	0,0635	0,0718	0,0761	0,0745	0,0810	0,0747	0,0000	0,0551	0,0643	0,0655	0,0338	0,0687	0,0369	0,0444	0,0655	0,0856
19	0,0424	0,0286	0,0313	0,0242	0,0515	0,0518	0,0524	0,0138	0,0652	0,0437	0,0127	0,0414	0,0348	0,0453	0,0305	0,0626	0,0369	0,0551	0,0000	0,0641	0,0278	0,0294	0,0486	0,0276	0,0286	0,0496	0,0486
20	0,0516	0,0378	0,0666	0,0513	0,0211	0,0242	0,0118	0,0503	0,0348	0,0227	0,0768	0,0227	0,0377	0,0413	0,0337	0,0167	0,0620	0,0643	0,0641	0,0000	0,0514	0,0727	0,0515	0,0641	0,0579	0,0547	0,0515
21	0,0528	0,0564	0,0337	0,0520	0,0793	0,0796	0,0802	0,0416	0,0595	0,0444	0,0151	0,0692	0,0626	0,0630	0,0583	0,0348	0,0393	0,0655	0,0278	0,0514	0,0000	0,0318	0,0488	0,0554	0,0564	0,0774	0,0764
22	0,0211	0,0348	0,0353	0,0634	0,0808	0,0812	0,0609	0,0432	0,0611	0,0499	0,0167	0,0708	0,0642	0,0845	0,0598	0,0665	0,0409	0,0338	0,0294	0,0727	0,0318	0,0000	0,0504	0,0570	0,0579	0,0790	0,0780
23	0,0729	0,0772	0,0151	0,0687	0,0305	0,0624	0,0490	0,0348	0,0590	0,0600	0,0337	0,0380	0,0138	0,0615	0,0490	0,0682	0,0348	0,0687	0,0486	0,0515	0,0488	0,0504	0,0000	0,0486	0,0242	0,0453	0,0276
24	0,0242	0,0380	0,0589	0,0518	0,0515	0,0242	0,0524	0,0138	0,0652	0,0633	0,0403	0,0414	0,0348	0,0729	0,0305	0,0808	0,0645	0,0369	0,0276	0,0641	0,0554	0,0570	0,0486	0,0000	0,0562	0,0515	0,0486
25	0,0571	0,0571	0,0393	0,0444	0,0547	0,0822	0,0462	0,0424	0,0590	0,0571	0,0413	0,0352	0,0380	0,0757	0,0242	0,0746	0,0590	0,0444	0,0286	0,0579	0,0564	0,0579	0,0242	0,0562	0,0000	0,0211	0,0518
26	0,0828	0,0690	0,0604	0,0655	0,0611	0,0790	0,0430	0,0377	0,0558	0,0539	0,0623	0,0320	0,0444	0,0725	0,0211	0,0714	0,0801	0,0655	0,0496	0,0547	0,0774	0,0790	0,0453	0,0515	0,0211	0,0000	0,0582
27	0,0729	0,0772	0,0427	0,0729	0,0305	0,0348	0,0490	0,0348	0,0590	0,0600	0,0613	0,0380	0,0138	0,0786	0,0490	0,0682	0,0380	0,0856	0,0486	0,0515	0,0764	0,0780	0,0276	0,0486	0,0518	0,0582	0,0000

Среднее время задержки составляет 47,7 мс

Оптимизированная матрица пропускных способностей

9. Оптимизированная пропускная способность Bo

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0	39834400	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21954000	0	0	0	14456800	0	27286800	0	0	0
2	39834400	0	0	15464000	0	0	0	0	9537600	37770000	0	0	0	0	0	0	3193200	0	35237600	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	2347200	41602000	0	0	0	0	1823600	0	0	0	0	0	0	44874800	0	0	0	0
4	0	15464000	0	0	0	0	0	0	4477200	15549600	0	0	0	14456800	0	0	0	0	31481200	0	0	0	0	0	9366400	0	0
5	0	0	0	0	0	0	0	0	21884000	0	0	0	67035200	0	0	0	0	0	0	39452000	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28228400	0	0	0	17186000	0	0	15967600
7	0	0	0	0	0	0	0	0	0	72135600	0	60494000	0	0	0	0	0	0	0	67674400	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	73264000	0	37075200	0	0	0	88112800	0	0	0	0	81949600	0	0	0
9	0	9537600	0	4477200	21884000	0	0	0	0	0	6199200	0	0	0	16909200	0	0	0	0	11858800	0	0	0	0	0	0	0
10	0	37770000	2347200	15549600	0	0	72135600	0	0	0	0	0	0	21143600	0	0	0	0	0	0	3032000	0	0	0	0	0	0
11	0	0	41602000	0	0	0	0	0	6199200	0	0	0	0	0	0	0	15388400	0	59018800	0	56259600	25348000	0	0	0	0	0
12	0	0	0	0	0	0	60494000	0	0	0	0	0	25746000	0	58610800	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	67035200	0	0	73264000	0	0	0	25746000	0	0	0	0	14018800	0	0	0	0	0	86743200	0	0	9708800	82377600
14	0	0	0	14456800	0	0	0	0	21143600	0	0	0	0	0	0	0	0	0	0	0	0	0	3022000	0	0	0	0
15	0	0	0	0	0	0	0	37075200	16909200	0	0	58610800	0	0	0	0	0	0	0	0	0	0	0	0	9738800	22760000	0
16	0	0	1823600	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29028800	6551600	0	0	0	0	0	0
17	0	3193200	0	0	0	0	0	0	0	0	15388400	0	14018800	0	0	0	0	0	0	0	0	0	5438800	0	0	0	0
18	21954000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5172000	0	0
19	0	35237600	0	31481200	0	0	0	88112800	0	0	59018800	0	0	0	0	0	0	0	0	0	0	0	0	0	16234400	0	0
20	0	0	0	0	39452000	28228400	67674400	0	11858800	0	0	0	0	0	0	29028800	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	3032000	56259600	0	0	0	0	6551600	0	0	0	0	0	0	0	0	0	0	0
22	14456800	0	0	0	0	0	0	0	0	0	25348000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	44874800	0	0	0	0	0	0	0	0	0	86743200	3022000	0	0	5438800	0	0	0	0	0	0	0	23178000	0	0
24	27286800	0	0	0	0	17186000	0	81949600	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	9366400	0	0	0	0	0	0	0	0	0	0	9738800	0	0	5172000	16234400	0	0	0	23178000	0	0	17196000	0
26	0	0	0	0	0	0	0	0	0	0	0	0	9708800	0	22760000	0	0	0	0	0	0	0	0	0	17196000	0	0
27	0	0	0	0	0	15967600	0	0	0	0	0	0	82377600	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5. Выводы

В ходе работы успешно рассчитаны требуемые пропускные способности линий связи, обеспечивающие заданное качество обслуживания. Реализованы алгоритмы определения и оптимизации пропускных способностей каналов, что позволило достичь целевого значения сквозной задержки. Таким образом, поставленная задача выполнена полностью.

6. Приложение

Код программы:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx


def load_data(path_matrix, path_distr, n1=20, n2=40, y0=0.1, n=31, L=200, a0=85600, T0=0.1, q=98
    data = pd.read_csv(path_matrix, delimiter=";", index_col=0, decimal=",")
    distribution = pd.read_csv(path_distr, delimiter=",", index_col=0, decimal=".")

    try:
        del distribution["Unnamed: 2"]
    except:
        pass

    print(distribution)
    sumAbonents = distribution.sum()["Абонентов"]
    print(data)

    return {
        "data": data,
        "distribution": distribution,
        "sumAbonents": sumAbonents,
        "n1": n1, "n2": n2, "y0": y0, "n": n, "L": L, "a0": a0, "T0": T0, "q": q,
        "Y_v0": Y_v0, "a_v0": a_v0, "a_IoT": a_IoT
    }


def intensity_traffic(state, out=False):
    """1. Интенсивность исходящего трафика от каждого из узлов сети y_i = N_i * y0"""
    dataIntencity = state["distribution"]["Абонентов"] * state["y0"]
    state["dataIntencity"] = dataIntencity

    if out:
        print("1. Интенсивность исходящего трафика от каждого из узлов сети:")
        print(dataIntencity)

    return dataIntencity


def distribution_coefficients(state, out=False):
    """2. Коэффициенты распределения трафика по направлениям связи kij = yj/sum(yi)"""
    sumIntencity = state["sumAbonents"] * state["y0"]
```

```
    dataKij = state["dataIntencity"] / sumIntencity
    state["dataKij"] = dataKij

    if out:
        print("2. Коэффициенты распределения трафика по направлениям связи")
        print(dataKij)

    return dataKij


def matrix_intensity_traffic(state, out=False):
    """3. Матрица интенсивностей трафика в направлениях связи"""
    # Формируем матрицу интенсивностей трафика Y[i,j] = y_i * k_ij
    matrixIntencity = pd.DataFrame(
        np.outer(state["dataIntencity"], state["dataKij"]),
        index=state["dataIntencity"].index,
        columns=state["dataKij"].index
    )
    state["matrixIntencity"] = matrixIntencity

    if out:
        print("3. Матрица интенсивностей трафика в направлениях связи")
        print(matrixIntencity)

    return matrixIntencity


def algorhitm_floid(state, out=False):
    """4. Матрица кратчайших маршрутов + матрица путей"""
    D = state["data"].copy()
    D.fillna(1e9, inplace=True)
    D.index = D.index.astype(int)
    D.columns = D.columns.astype(int)

    Next = D.copy()

    # Инициализация матрицы Next
    for i in D.index:
        for j in D.index:
            if i != j and D.loc[i,j] < 1e12:
                Next.loc[i,j] = j

    # Основной цикл алгоритма Флойда
```

```
    for k in D.index:  # промежуточная вершина
        for i in D.index:  # начальная вершина
            for j in D.index:  # конечная вершина
                if D.loc[i,k] + D.loc[k,j] < D.loc[i,j]:  # найден более короткий путь
                    D.loc[i,j] = D.loc[i,k] + D.loc[k,j]  # обновляем расстояние
                    Next.loc[i,j] = Next.loc[i,k]  # обновляем следующий узел на пути

    matrixNext = Next
    R = D

    matrixNext.index = matrixNext.index.astype(int)
    matrixNext.columns = matrixNext.columns.astype(int)

    state["R"] = R
    state["matrixNext"] = matrixNext

    if out:
        print("4. Матрица кратчайших маршрутов между вершинами графа + матрица путей")
        print("Кратчайшие маршруты:")
        print(R)
        print("Пути:")
        print(matrixNext)

    return R, matrixNext


def calc_link_load(state, out=False):
    """5. Матрица интенсивностей нагрузки на линии связи Y_tilde"""
    Y = state["matrixIntencity"]
    Next = state["matrixNext"]
    nodes = list(Y.index)  # узлы
    n = len(nodes)  # число узлов

    Y_tilde = pd.DataFrame(np.zeros((n,n)), index=nodes, columns=nodes)

    for i in nodes:
        for j in nodes:
            if i == j or Y.loc[i,j] == 0:
                continue
            path = [i]  # начинаем путь с i
            cur = i  # текущий узел
            while cur != j:  # идём по пути, пока не дойдём до j
                cur = Next.loc[cur,j]  # следующий узел на пути из cur в j
```

```
                path.append(cur)  # добавляем его в путь

            for u, v in zip(path[:-1], path[1:]):  # распределяем трафик по рёбрам
                Y_tilde.loc[u,v] += Y.loc[i,j]  # нагрузка на ребро u->v

    state["matrixNagruzka"] = Y_tilde

    if out:
        print("#5. Матрица интенсивностей нагрузок на линии связи")
        print(Y_tilde)

    return Y_tilde


def matrix_v(state, out=False):
    """6. Матрица потоков (число каналов)"""
    Y_tilde_sym = state["matrixNagruzka"].copy()
    line_load = pd.DataFrame(0.0, index=Y_tilde_sym.index, columns=Y_tilde_sym.columns)

    # Делаем матрицу нагрузок симметричной
    for i in Y_tilde_sym.index:
        for j in Y_tilde_sym.columns:
            if i >= j:
                continue
            total_load = Y_tilde_sym.loc[i, j]
            line_load.loc[i, j] = total_load
            line_load.loc[j, i] = total_load

    V_matrix = pd.DataFrame(0, index=line_load.index, columns=line_load.columns, dtype=int)

    # То, сколько нужно каналов, чтобы обслужить нагрузку с вероятностью блокировки ≤ 2%
    for i in line_load.index:
        for j in line_load.columns:
            if i == j:
                continue
            load = line_load.loc[i, j]
            if load > 0:
                v = find_min_channels(load, p_block=(1 - state["q"]/100), max_v=50000)
                V_matrix.loc[i, j] = v
                V_matrix.loc[j, i] = v

    state["V_matrix"] = V_matrix
    if out:
```

```
        print("6. Матрица потоков")
        print(V_matrix)
    return V_matrix


def matrix_a(state, out=False):
    """7. Матрица интенсивности трафика A[i,j]"""
    # A[i,j] = V[i,j] * a0
    A_matrix = state["V_matrix"].astype(float) * state["a0"]
    state["A_matrix"] = A_matrix
    if out:
        print("7. Матрица интенсивности трафика")
        print(A_matrix)
    return A_matrix


def matrix_b(state, out=False):
    """8. Матрица пропускных способностей B[i,j]"""
    # B[i,j] = A[i,j] + (L * 8) / T0
    B_matrix = pd.DataFrame(0, index=state["A_matrix"].index, columns=state["A_matrix"].columns,
    L_bits = state["L"] * 8
    for i in state["A_matrix"].index:
        for j in state["A_matrix"].columns:
            if state["A_matrix"].loc[i, j] == 0:
                continue
            B_matrix.loc[i, j] = state["A_matrix"].loc[i,j] + (L_bits / state["T0"])

    state["B_matrix"] = B_matrix
    if out:
        print("8. Матрица пропускных способностей")
        print(B_matrix)
    return B_matrix


# Вспомогательные функции

def erlangb(v: int, y: float) -> float:
    if y <= 0: return 0.0
    if v == 0: return 1.0
    inv_b = 1.0
    for i in range(1, v + 1):
        inv_b = 1.0 + inv_b * (i / y)
    return 1.0 / inv_b
```

```python
def find_min_channels(y: float, p_block: float = 0.02, max_v: int = 50000) -> int:
    # Ищем минимальное число каналов v, чтобы вероятность блокировки ≤ p_block
    # обратная формула Эрланга B
    if y <= 0: return 0
    if y < 1.0: return 1 # при малой нагрузке достаточно 1 канала
    low, high = 1, max_v
    while low < high:
        mid = (low + high) // 2
        if erlangb(mid, y) > p_block:
            low = mid + 1
        else:
            high = mid
    v = low
    while erlangb(v, y) > p_block:
        v += 1
    return v

# Основные функции для оптимизации
def build_paths(state):
    # Строим пути между всеми парами узлов на основе матрицы Next
    nodes = list(state["A_matrix"].index)
    n = len(nodes)
    next_hop = state["matrixNext"].values
    node_to_idx = {node: i for i, node in enumerate(nodes)} # карта узлов к индексам

    paths = [[None] * n for _ in range(n)] # paths[a][b] = список индексов узлов на пути из a в
    for a_idx in range(n):
        for b_idx in range(n):
            if a_idx == b_idx:
                continue
            i = a_idx
            path = [i]
            while i != b_idx:
                next_node = state["matrixNext"].iloc[nodes[i]-1, nodes[b_idx]-1]
                # Переводим номер узла в индекс в списке paths
                i = node_to_idx[next_node]
                path.append(i)
            paths[a_idx][b_idx] = path
    return paths, nodes, n
```

```python
def calc_dl(B, A_np, L_bits, paths, n):
    # Вычисляем матрицу задержек DL
    DEL = L_bits / (B - A_np)
    # L_bits / (B - A_np) — сколько секунд нужно, чтобы передать пакет по этой линии. B - A_np -
    DEL[np.isinf(DEL)] = 1e9
    DEL[DEL < 0] = 1e9
    DL = np.zeros((n, n), dtype=float)
    for i in range(n):
        for j in range(n):
            if i == j:
                continue
            route = paths[i][j]
            DL[i,j] = sum(DEL[route[k], route[k+1]] for k in range(len(route)-1)) # сумма задер
    return DL


def objective_function(B, A_np, L_bits, Topt, paths, n):
    DL = calc_dl(B, A_np, L_bits, paths, n)
    return np.sum((DL - Topt)**2)


def optimization(state, accuracy=0.001, out=False, out_graph_O=False):
    """9. Оптимизация"""
    Topt = state["T0"] / 2  # целевое значение задержки
    dc = 10_000  # шаг изменения пропускной способности
    L_bits = state["L"] * 8  # длина пакета в битах
    Bo = state["B_matrix"].copy()  # начальная матрица пропускных способностей
    A_np = state["A_matrix"].values  # матрица интенсивности трафика

    paths, nodes, n = build_paths(state)

    default_O = objective_function(state["B_matrix"].values, A_np, L_bits, Topt, paths, n)
    best_O = np.inf
    iteration = 0
    sp_o = []
    sp_iteration = []

    while True:
        iteration += 1
        base_O = objective_function(Bo.values, A_np, L_bits, Topt, paths, n)
        best_edge = None

        for i in nodes:
```

```
            for j in nodes:
                if i >= j or Bo.loc[i,j] <= 16000:
                    continue
                Bo.loc[i,j] += dc
                Bo.loc[j,i] += dc
                O_try = objective_function(Bo.values, A_np, L_bits, Topt, paths, n)
                if O_try < best_O:
                    best_O = O_try
                    best_edge = (i,j)
                Bo.loc[i,j] -= dc
                Bo.loc[j,i] -= dc

        if best_edge is None or (base_O - best_O) < accuracy:
            print("\nУлучшения не найдено — оптимизация завершена.")
            break

        i, j = best_edge
        Bo.loc[i,j] += dc
        Bo.loc[j,i] += dc
        sp_o.append(base_O)
        sp_iteration.append(iteration)

    state["Bo"] = Bo
    state["DL2"] = pd.DataFrame(calc_dl(Bo.values, A_np, L_bits, paths, n),
                                index=state["A_matrix"].index, columns=state["A_matrix"].columns

    if out:
        print("Оптимизированная матрица пропускных способностей:")
        print(f"Начальная O: {default_O:.3f}")
        print(f"Финальная O: {best_O:.3f}")
        print(Bo)

    if out_graph_O:
        plt.plot(sp_iteration, sp_o)
        plt.xlabel("Номер итерации")
        plt.ylabel("Значение целевой функции")
        plt.grid(True)
        plt.show()

    return Bo


# Сохранение результатов в Excel
```

```python
def save_to_excel(state, path="network_results_single_sheet.xlsx"):
    """Сохраняем всё в EXCEL таблицами"""
    excel_file = path

    with pd.ExcelWriter(excel_file, engine="xlsxwriter") as writer:
        start_row = 0

        # Список всех матриц для записи (берём из state)
        matrices = [
            ("1. Интенсивность исходящего трафика Y_i", state["dataIntencity"].to_frame(name="Y_
            ("2. Коэффициенты распределения трафика k_ij", state["dataKij"].to_frame(name="k_ij'
            ("3. Матрица интенсивностей трафика", state["matrixIntencity"]),
            ("4. Матрица кратчайших путей (расстояния)", state["R"]),
            ("Матрица следующих узлов (Next)", state["matrixNext"]),
            ("5. Матрица нагрузок на линии связи Y_tilde", state["matrixNagruzka"]),
            ("6. Матрица требуемого числа каналов V[i,j]", state["V_matrix"]),
            ("7. Матрица интенсивности трафика ПД A[i,j] (бит/с)", state["A_matrix"]),
            ("8. Пропускная способность линий B[i,j]", state["B_matrix"]),
            ("9. Оптимизированная пропускная способность Bo", state["Bo"]),
            ("Задержки после оптимизации DL (мс)", state["DL2"]),
        ]

        for title, df in matrices:
            # Записываем заголовок
            pd.DataFrame([title]).to_excel(writer, sheet_name="Results", index=False, header=Fa
            start_row += 1

            # Записываем матрицу
            df.to_excel(writer, sheet_name="Results", startrow=start_row)
            start_row += len(df) + 2  # +2: одна пустая строка + место под сумму

            # Сумма всех значений
            total = df.values.sum()
            sum_df = pd.DataFrame([["Сумма всех значений", total]])
            sum_df.to_excel(writer, sheet_name="Results", index=False, header=False, startrow=st
            start_row += 3  # отступ перед следующей матрицей

    print(f"Все результаты успешно сохранены в файл: {excel_file}")
    return 1


state = load_data("data/only_matrix.csv", "data/Distribution.csv")
```

```
intensity_traffic(state, out=True)
distribution_coefficients(state)
matrix_intensity_traffic(state)
algorhitm_floid(state)
calc_link_load(state)
matrix_v(state)
matrix_a(state)
matrix_b(state)
optimization(state, out=True, out_graph_O=True)
save_to_excel(state, path="DATA1.xlsx")
```