Super-reliable Loss Networks

Elena N. Petrova * Sergei Yu. Popov †

May 24, 1994

This paper continues the series of publications entering into the framework of the collaboration between the project MEVAL and the Laboratory of Large Random Systems, as specified in the agreement between INRIA and MGU (Moscow State University).

Editors: Guy Fayolle and Vadim Malyshev.

Abstract

We consider a loss network [1] consisting of independently working servers, each one having an input Poissonian stream of customers. If a customer arrives to an occupied server x, it can be put to one of the empty servers from some finite set Q_x. If all servers from Q_x are occupied, the customer is lost. We calculate the loss probability for some particular cases of such system in the regime when it is small.

*Postal address: Institute for Problems of Information Transmission, Russian Academy of Sciences, Ermolovoy, 19, Moscow, 101447, Russia; E-mail: petrova@ippi.msk.su

†Postal address: Laboratory of Large Random Systems, Faculty of Mechanics and Mathematics, Moscow State University, 119899, Moscow, Russia; E-mail: popov@llrs.math.msu.su

Réseaux superfiables avec pertes E.N.Petrova S.Yu.Popov 4 Mai 1994

Ce rapport continue la série de publications entrant dans le cadre de la collaboration entre le projet MEVAL et le LLRS de l'Université de Moscou.

Editors: Guy Fayolle and Vadim Malyshev.

Résumé

Nous considérons un réseau de serveurs [1] avec pertes. Les noeuds (serveurs) travaillent indépendamment et sont soumis à des arrivées de clients selon un Processus de Poisson. Si un client arrive à un serveur occupé, il peut être redirigé vers un des serveurs vides, appartenant à un ensemble fini Q_x. Si tous les serveurs de Q_x sont occupés, le client est perdu. Nous calculons la probabilité de perte pour certains systèmes particuliers, lorsque ces probabilités sont faibles.

1 Introduction

This report adjoins the paper [3], where a general theory of reliable loss networks was developed. Under some circumstances the probability of loosing the message should be negligible (or should not exceed probabilities of other improbable events).

In this paper, contrary to the cited above, some less general models are considered. The goal is to give more explicit formulas, using stationary techniques of cluster expansion. Note that in the above cited paper the non-stationary techniques was used.

The second goal was to write a program (using the programming language C) to calculate these small loss probabilities. The same program could be written for a realistic network if all necessary data are available. Note that writing a program which could be used for all imaginable networks is impossible and useless job.

Consider a network which is described by a set V of servers, $x \in V$ is called an x–server (or x–station). Calls requesting x–server arrive as a Poisson stream (called x–stream) of rate λ_x. These streams are independent for different x. A call of x–stream arriving at time t occupies immediately x–server if it is empty. If x–server is occupied, this call is immediately put to an empty server chosen according to some deterministic rule among the elements of some finite set Q_x . If all servers from Q_x are busy then the call is lost. We assume that Q_x are fixed and the rule is fixed and depends only on x. A customer of x–stream served on y–station is called a (x, y)–type customer. The serving time is independent of earlier arrival times and serving times and is exponentially distributed with some rate $\mu(x, y)$.

It is convenient to consider a graph structure on V. One way is to draw a directed link $l = (x, y)$ from $x \in V$ to $y \in V$ iff $y \in Q_x$.

There are other examples where the network is represented as follows. Here V is the set of nodes of some non oriented connected graph $G = (V, L)$, L is a set of links. We say that there exists a *path* connecting the nodes $x \in V$ and $y \in V$ if there is a sequence of nodes $x_1, \dots , x_n \in V$ such that $x_1 = x$, $x_n = y$, $(x_i, x_{i+1}) \in L$ for each $i = 1, ..., n-1$; n is called *the length* of a path. The *distance* $d(x, y)$ between nodes x and y is the minimal length of a path connecting x and y. We define $Q_x = Q_x(d)$ as the set of nodes which are at a distance not greater than d from node x, where d is some fixed positive number.

Denote $q_{\text{loss}}(x) = q(\lambda_x, \mu(x, y), d)$ the loss probability of an arriving customer at a given moment in the stationary regime for given rates λ_x, $x \in V$, $\mu(x, y)$, $x \in V$, $y \in Q_x$, and fixed parameter d . To calculate q_{loss} we have to consider the probability that all y-stations from Q_x are occupied.

Example. Independent system.

If $d < 1$ then obviously $Q_x = \{x\}$ and a customer arriving to an occupied server is immediately lost. So, we have independent queues of $M \mid M \mid 1$ type. Loss probability is given by well known formula [1]:

$$q_x = \lambda_x / (\mu_x + \lambda_x) \tag{1}$$

Our aim is to calculate the loss probability in the stationary regime under assumption that it is small and that a set of servers, where a call can be transmitted, is not empty.

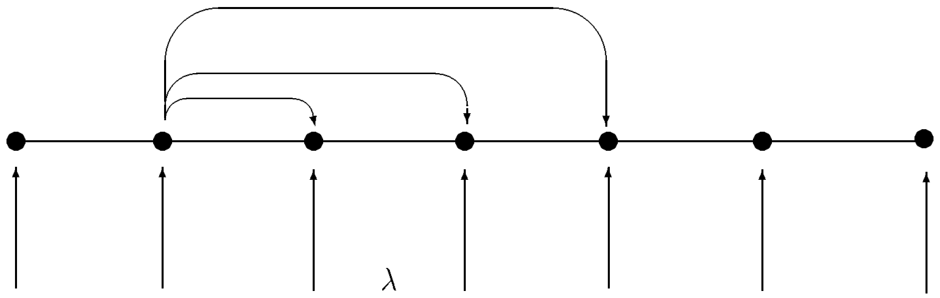

Figure 1: A chain of servers

2 A chain of servers.

2.1 Main Equation.

We consider now the case when graph G is a segment $V = [-N, N] \cap Z^1$ of a one-dimensional lattice , i.e. the servers are arranged in a line and the distance between two neighboring servers is equal to one.

This system is good to demonstrate the technique because of simplicity of notations with respect to the case of an arbitrary graph.

We assume that all input streams have the same rate and that a customer is lost if it can not find an empty node within a distance d to the right of its arrival node. This means that

$$\lambda_x \equiv \lambda; \quad Q_x = \{x+1,\ x+2,\ \cdots,\ x+d\}$$

Let us fix the following rule of searching for an empty node. If a call of x-stream arrives and x is occupied, then the system looks for nearest to x empty node to the right.

Let us assume that the serving rate of a call accepted at the arrival node does not depend on the node. We assume also that the serving rate of a transmitted call does not depend neither on the call, nor on the accepting station, so

$$\mu(x,y) = \begin{cases} \mu_1 & \text{if } y = x \\ \mu_2 & \text{if } y \neq x \end{cases} \tag{2}$$

Let us write $s_x = 0$ if the node x is empty, $s_x = 1$ if the node x is occupied by a call from x-stream and $s_x = 2$ if the node x is occupied by a call from y-stream for some $y \in \{x-1, \cdots, x-d\}$.

As arriving streams are Poissonian and serving times are independent exponential, we have a continuous time Markov chain $\xi_t = \{\xi_t(x),\ x \in V\}$ with state space $\{0,1,2\}^V$.

The loss probability

$$q_{loss} = P(s_0 \neq 0,\ s_1 \neq 0, \cdots, s_d \neq 0) \tag{3}$$

In order to calculate the loss probability we shall write down the Kolmogorov equations [2].

Let $h(s_V, s'_V), s_V \neq s'_V$, be transition rates for this Markov chain from the state s'_V to s_V so that

$$h(s_V, s'_V)dt = P(\xi_{t+dt}(x) = s_x, x \in V | \xi_t(x) = s'_x, x \in V) \tag{4}$$

Denote $p_t(s_V)$ the probability that our Markov chain is at the state s_V at time t. Then the Kolmogorov equations are:

$$p_{t+dt}(s_V) = p_t(s_V) + \sum_{s'_V} h(s_V, s'_V)p_t(s'_V)dt$$

where

$$h(s_V, s_V) = -\sum_{s'_V : s'_V \neq s_V} h(s'_V, s_V)$$

For our model only one node can change at a time, i.e. for $s_V \neq s'_V$ the rates $h(s_V, s'_V)$ can be different from 0 only if s_V and s'_V differ exactly at one point. Denote $h^s_x(s'_V)$ the rate of transition from s'_V to s_V, where $s(y) = s'(y)$ for $y \neq x$ and $s(x) = s_x \neq s'(x)$.

Denote also $\delta_s(s')$ the Kroneker symbol. From (4) one gets

$$P(\xi_{t+dt}(x) = s|\vec{\xi}_t) = (1 - \delta_s(\xi_t(x)))dt h^s_x(\vec{\xi}_t) + \delta_s(\xi_t(x))(1 - dt \sum_{s' : s' \neq s} h^{s'}_x(\vec{\xi}_t))$$

Taking expectation we get

$$p_{t+dt}(x; s) = \sum_{s_V : s_x \neq s} h^s_x(s_V)p_t(s_V)dt + p_t(x; s) - dt \sum_{s_V : s_x = s} \sum_{s' \neq s} h^{s'}_x(s_V)p_t(s_V)$$

where

$$p_{t+dt}(x; s) = P(\xi_{t+dt}(x) = s).$$

For the stationary measure

$$0 = \sum_{s_V : s_x \neq s} h^s_x(s_V)p(s_V) - \sum_{s_V : s_x = s} \sum_{s' \neq s} h^{s'}_x(s_V)p(s_V) \tag{5}$$

For any sets $A, B, C, D \subset V$ denote $p_t(1_A 2_B 0_C *_D)$ the probability that the nodes of C are empty, there is a μ_1-customer at each node of A, there is a μ_2-customer at each node of B and at the same time the nodes of D are occupied.

Then (5) becomes (we put $s = 1$ for the first equation in (6) and $s = 2$ for the second one)

$$0 = \lambda p(0_x) - \mu_1 p(1_x)$$

$$0 = \lambda \sum_{k=1}^{d} p(*_{[x-k,x-1]} 0_x) - \mu_2 p(2_x) =$$

$$\lambda \sum_{k=1}^{d} \Big(p(*_{[x-k,x-1]}) - p(*_{[x-k,x]}) \Big) - \mu_2 p(2_x). \tag{6}$$

Similarly for any finite dimensional probabilities

$$P(\xi_{t+dt}(x) = s_x, x \in A | \vec{\xi}_t) =$$

$$\sum_{x\in A} (1 - \delta_{s_x}(\xi_t(x))) \left(\prod_{y: y \neq x, y\in A} \delta_{s_y}(\xi_t(y))\right) h_x^{s_x}(\vec{\xi}_t) dt + \tag{7}$$

$$\left(\prod_{x\in A} \delta_{s_x}(\xi_t(x))\right) \left(1 - dt \sum_{x\in A} \sum_{s'_x \neq s_x} h_x^{s'_x}(\vec{\xi}_t)\right).$$

Taking expectation in (7):

$$p_{t+dt}(1_A 2_B) = \lambda \sum_{x\in A} p_t(1_{A-\{x\}} 2_B 0_x) dt + p_t(1_A 2_B) +$$

$$\lambda \sum_{y\in B} \sum_{k=1}^{d} p_t \left(1_A 2_{B-\{y\}} 0_y *_{[y-k,y-1]}\right) dt - (\mu_1 |A| + \mu_2 |B|) p_t(1_A 2_B) dt, \tag{8}$$

where $| A |$ is the number of elements in the set A. From (8)

$$\frac{\partial p_t(1_A 2_B)}{\partial t} = \lambda \sum_{x\in A} p_t(1_{A-\{x\}} 2_B 0_x) +$$

$$\lambda \sum_{y\in B} \sum_{k=1}^{d} p_t \left(1_A 2_{B-\{y\}} 0_y *_{[y-k,y-1]}\right) - (\mu_1 |A| + \mu_2 |B|) p_t(1_A 2_B)$$

This gives for the stationary regime:

$$(\mu_1 |A| + \mu_2 |B|) p(1_A 2_B) =$$

$$\lambda \sum_{x\in A} \left(p(1_{A-\{x\}} 2_B) - p(1_{A-\{x\}} 2_B *_x)\right) +$$

$$\lambda \sum_{y\in B} \sum_{k=1}^{d} \left(p\left(1_A 2_{B-\{y\}} *_{[y-k,y-1]}\right) - p\left(1_A 2_{B-\{y\}} *_{[y-k,y]}\right)\right) \tag{9}$$

Finally, substituting (9) into (3) we get our main formula:

$$q_{\text{loss}} = \sum_{\substack{A,B:\, A\cap B=\emptyset \\ A\cup B=[0,d]}} (\mu_1 |A| + \mu_2 |B|)^{-1} \Bigg[\lambda \sum_{x\in A} \left(p(1_{A-\{x\}} 2_B) - p(1_{A-\{x\}} 2_B *_x)\right) +$$

$$\lambda \sum_{y\in B} \sum_{k=1}^{d} \left(p\left(1_A 2_{B-\{y\}} *_{[y-k,y-1]}\right) - p\left(1_A 2_{B-\{y\}} *_{[y-k,y]}\right)\right)\Bigg] \tag{10}$$

2.2 Expansion

Iterating (9) we can get expansion of $p(1_A 2_B)$ in λ. The technique of such expansions, called cluster expansions, was developed in [4]. Now we shall describe the expansion.

Denote α an ordered pair of finite non intersecting subsets of V, $\alpha = \{A, B\}$. We say that $\alpha = \emptyset$ if $A = \emptyset, B = \emptyset$. Denote also

$$\mathrm{supp}\alpha = A \cup B, \quad p_\alpha = p(1_A 2_B), \quad k_\alpha = \mu_1 \mid A \mid + \mu_2 \mid B \mid .$$

Then we may rewrite (9) as

$$p_\alpha = \lambda k_\alpha^{-1} \sum_{x \in \mathrm{supp}\alpha} \sum_{\alpha'} \theta(\alpha') p_{\alpha'} \tag{11}$$

where the internal summation is over all $\alpha' = \{A', B'\}$ that can be constructed from α for any given x by the following algorithm.

1. If x belongs to A, then there are three possibilities:

   a) $A' = A \setminus \{x\},\ B' = B,\ \theta(\alpha') = 1;$

   b) $A' = A \setminus \{x\},\ B' = B \cup \{x\},\ \theta(\alpha') = -1;$

   c) $A' = A,\ B' = B,\ \theta(\alpha') = -1.$

2. If x belongs to B, then fix an arbitrary k, $k = 1, \ldots, d$. For any fixed k two of the following possibilities can take place:

   d)$\mathrm{supp}\alpha' = A \cup (B \setminus \{x\}) \cup [x-k, x-1],\ \ A' \supseteq A,\ \ B' \supseteq B \setminus \{x\},\ \ \theta(\alpha') = 1;$

   e) $\mathrm{supp}\alpha' = A \cup B \cup [x-k, x],\ \ A' \supseteq A,\ \ B' \supseteq B \setminus \{x\},\ \ \theta(\alpha') = -1.$

Definition. We define *a cluster* as a finite sequence

$$\Gamma = (\alpha_1, x_1, \alpha_2, x_2, \ldots, \alpha_N, x_N, \alpha_{N+1})$$

where α_i is a pair of non intersecting finite subsets of V, $x_i \in \mathrm{supp}\alpha_i$, for any i α_{i+1} is constructed from α_i, x_i by the algorithm described above and $\alpha_{N+1} = \emptyset$. $N = N(\Gamma)$ is called the length of Γ.

Iterating (11) we may write down formally the following series:

$$p_\alpha = \sum_{N=|\mathrm{supp}\alpha|}^{\infty} \lambda^N \sum_{\Gamma : N(\Gamma)=N} a_\Gamma \;\; p(0) \tag{12}$$

where the internal summation is over all clusters Γ with fixed length N and such that $\alpha_1(\Gamma) = \alpha$ and

$$a_\Gamma = \prod_{i=1}^{N(\Gamma)} k_{\alpha_i}^{-1} \theta(\alpha_i), \quad \theta(\alpha_1) = 1 \tag{13}$$

Theorem 1 *The series (12) absolutely converges for small enough λ.*

Proof. Let us fix some positive N and consider all clusters of length N. Denote $\mu = min(\mu_1, \mu_2)$. Then

$$k_{\alpha_i}^{-1} \leq (\mu \mid \mathrm{supp}\alpha_i \mid)^{-1}$$

and

$$a(\Gamma) \leq \mu^{-N} \prod_{i=1}^{N} (\mid \mathrm{supp}\alpha_i \mid)^{-1}$$

Consider the algorithm of constructing α'. If case 1 takes place then only three different α' can be constructed. If case 2 takes place, then the number of possibilities to fix k does not exceed d. For fixed k consider the case d). In order to construct α' we need to point out for each vertex from $[x-k, x-1] \cap (V \setminus (A \cup B))$ whether it belongs to A' or to B'. So, the number of different possibilities does not exceed

$$d2^{|[x-k,x-1]\cap(V\setminus(A\cup B))|} \leq d2^d.$$

Similarly, if case 2e) takes place then the number of possibilities does not exceed $d2^{d+1}$. Then performing consequently summation in (12) over x_N, α_N, x_{N-1},...,α_2, x_1 we obviously get that

$$\sum_{\Gamma: N(\Gamma)=N} a_\Gamma \leq d^N 2^{N(d+1)} \mu^{-N}. \tag{14}$$

So, the right hand side of (12) does not exceed

$$\sum_{N=|\mathrm{supp}\alpha|}^{\infty} (\frac{\lambda}{\mu})^N 2^{N(d+1)} d^N \tag{15}$$

and this series converges for small enough $\frac{\lambda}{\mu}$.

Substituting (12) into (10) we get expansion for q_{loss}:

$$q_{\mathrm{loss}} = \sum_{\alpha:\ \mathrm{supp}\alpha=[0,d]} \sum_{N=d+1}^{\infty} \sum_{\substack{\Gamma:\ \alpha_1(\Gamma)=\alpha, \\ N(\Gamma)=N}} a_\Gamma\ p(0). \tag{16}$$

2.3 Algorithm.

Our aim here is to calculate the sum of terms of expansion (16) corresponding to $N = d+1$. Then in the case of small enough λ putting $p(0) = 1$ as the first approximation (we know, that $p(0) = 1 - p(1) - p(2) = 1 - o(\lambda)$) this gives us the main term of expansion of the loss probability in λ.

To simplify calculations we need some properties of functions $p(1_A 2_B)$.

Proposition 2 *For small* λ $p(1_A 2_B)$ *is of order at least* $\lambda^{|A|+|B|}$.

Proof. We use induction on $|A| + |B|$. From (6) it is easy to see that the proposition is true for one dimensional probabilities $p(1)$ and $p(2)$. In particular,

$$p(1) = \frac{\lambda}{\mu_1} p(0) = \frac{\lambda}{\mu_1} + o(\lambda) \tag{17}$$

The step of induction follows from (9).

Proposition 3 *If there exists $y_0 \in B$ such that ($y_0 - 1) \notin A \cup B$ then $p(1_A 2_B)$ is at least of order $\lambda^{|A|+|B|+1}$.*

Proof. We use induction once more. The basis of induction $p(2_x) = O(\lambda^2)$ follows from Proposition 1 and the second equation of (6) if we note that $p(*_{[x-k,x-1]})$ is at least of order of λ. The step of induction follows from (9). In fact, consider the r.h.s. of (9). First,

$$p(1_{A-x} 2_B *_x) \leq p(1_{A-x} 2_B) \leq O(\lambda^{|A|+|B|})$$

by induction assumption. In the second summation, if $y \neq y_0$ then the set $A \cup (B - y)$ satisfies the induction assumption. So, for any k

$$p(1_A 2_{B-\{y\}} *_{[y-k,y-1]}) \leq p(1_A 2_{B-\{y\}}) \leq O(\lambda^{|A|+|B|})$$

If $y = y_0$, then the set $A \cup (B - \{y_0\}) \cup [y_0 - k, y_0 - 1]$ consists at least of $|A| + |B|$ points and

$$p(1_A 2_{B-\{y_0\}} *_{[y_0-k,y_0-1]}) \sim O(\lambda^{|A|+|B|})$$

by Proposition 1. Finally, by Proposition 1

$$p(1_A 2_{B-\{y\}} *_{[y-k,y]}) \leq O(\lambda^{|A|+|B|})$$

Putting these inequalities in the r.h.s. of (9) completes the proof.

Using these propositions we see that the main term in (10) gives the summation over such A, B that $0 \in A$.

Definition.

We say that a pair of sets (A, B) satisfies *$l-$condition* if for any $y \in B \quad y-1 \in A \cup B$. Denote r_x the distance from $x \in B$ to the nearest to x node in the left which does not belong to $A \cup B$.

Proposition 4 *Let $A \cup B$ be connected and let (A, B) satisfy l-condition. Then for small λ*

$$(\mu_1 |A| + \mu_2 |B|) p(1_A 2_B) =$$
$$\lambda \left(\sum_{x \in A : x+1 \in A} p(1_{A-\{x\}} 2_B) + \sum_{y \in B : y+1 \in A} (r_y - 1) p(1_A 2_{B-\{y\}}) \right) + o(\lambda^{|A|+|B|}) \tag{18}$$

Here the first summation is over the nodes serving "their own" customers such that the nearest node on its right is serving "its own" customer too. The second summation is over the nodes serving a transmitted customer such that its right nearest neighbour is serving a μ_1–customer. The proof follows from (9) and Propositions 1 and 2.

Lemma 5 *Let $A \cup B$ be disconnected. Let D_1 and D_2 be its components in a sense that there is no path from D_1 to D_2 belonging completely to $A \cup B$ (D_1 and D_2 can be disconnected in their turn). Denote $A_1 = A \cap D_1$, $B_1 = B \cap D_1$, $A_2 = A \cap D_2$, $B_2 = B \cap D_2$. Let (A, B) satisfy l-condition. Then*

$$p(1_A 2_B) = p(1_{A_1} 2_{B_1}) p(1_{A_2} 2_{B_2}) + O(\lambda^{|A|+|B|+1}) \tag{19}$$

Proof. We proceed again by induction on $n = |A| + |B|$. Denote D the minimal connected set, containing $A \cup B$, $|D| = k$. Let us numerate the nodes of D from 1 to k. As our system is spatially homogeneous we denote $p(1_A 2_B) = p_{i_1 i_2 \dots i_k}$ where each i_m takes values 1,2 or $\diamond$ depending on whether the corresponding node belongs to A, B, or does not belong to $A \cup B$.

For $n = 2$ only $p_{1\diamond\dots\diamond 1}$ satisfies l-condition. ¿¿From (9) using the first equation of (6) we have for any $k > 2$

$$p_{\underbrace{1\,\diamond\,\dots\,\diamond\,1}_{k}} = \frac{\lambda}{\mu_1} p_1 = \frac{\lambda^2}{\mu_1^2} (1 + o(1)) = p_1^2 + o(\lambda^2)$$

Suppose that (19) is true for any A, B such that $|A| + |B| \leq n$. Consider the probability $p_{1i_1\dots i_k}$ where $i_k = 1$ or 2 and there exists $m < k$ such that $i_m = \diamond$. Suppose also that exactly n among k positions i_j are different from $\diamond$. We put $A = \{j : i_j = 1\} \cup \{0\}$, $B = \{j : i_j = 2\}$, $A_1 = \{j < m : i_j = 1\} \cup \{0\}$, $A_2 = \{j > m : i_j = 1\}$, $B_1 = \{j < m : i_j = 2\}$, $B_2 = \{j > m : i_j = 2\}$. Then according to (9) and Proposition 1 we have

$$\begin{aligned}(\mu_1|A| + \mu_2|B|)p_{1i_1\dots i_k} = \lambda \Bigg[& p_{i_1\dots i_k} + \sum_{x\in A_1} p^{(x)}_{1i_1\dots i_k} + \sum_{y\in B_1} (r_y - 1)p^{(y)}_{1i_1\dots i_k} + \\ & \sum_{x\in A_2} p^{(x)}_{1i_1\dots i_k} + \sum_{y\in B_2} (r_y - 1)p^{(y)}_{1i_1\dots i_k} + p_{1i_1\dots i_{k-1}} \Bigg] + o(\lambda^{n+1})\end{aligned} \tag{20}$$

Here we denoted $p^{(y)}_{1i_1\dots i_k} = p_{1i_1\dots\diamond\dots i_k}$ the probability which is obtained from $p_{1i_1\dots i_k}$ by changing $i_y = 2$ to $i_y = \diamond$. Now we may apply the induction assumption to each term in the r.h.s. of (20). We get:

$$\begin{aligned}\lambda \Bigg[& p_{i_1\dots i_{m-1}} p_{i_{m+1}\dots i_k} + \sum_{x\in A_1} p^{(x)}_{1i_1\dots i_{m-1}} p_{i_{m+1}\dots i_k} + \\ & \sum_{y\in B_1} (r_y - 1)p^{(y)}_{1i_1\dots i_{m-1}} p_{i_{m+1}\dots i_k} + \sum_{x\in A_2} p_{1i_1\dots i_{m-1}} p^{(x)}_{i_{m+1}\dots i_k} + \\ & \sum_{y\in B_2} (r_y - 1)p_{1i_1\dots i_{m-1}} p^{(y)}_{i_{m+1}\dots i_k} + p_{1i_1\dots i_{m-1}} p^{(y)}_{i_{m+1}\dots i_k} \Bigg] + o(\lambda^{n+1})\end{aligned} \tag{21}$$

Grouping together first three terms in (21) and using (9) and Proposition 1 we see that their sum is equal to

$$(\mu_1|A_1| + \mu_2|B_1|)\, p_{1i_1\dots i_{m-1}} p_{i_{m+1}\dots i_k} + o(\lambda^{n+1})$$

Similarly the second three terms in (21) give us

$$(\mu_1|A_2| + \mu_2|B_2|)\, p_{1i_1\dots i_{m-1}} p_{i_{m+1}\dots i_k} + o(\lambda^{n+1})$$

Combining these expressions together we finally get

$$(\mu_1|A| + \mu_2|B|)\, p_{1i_1\dots i_{m-1}} p_{i_{m+1}\dots i_k} + o(\lambda^{n+1})$$

This finishes the proof.

Using Propositions 1,2 and 3 we rewrite (10) as follows:

$$q_{\text{loss}} = \sum_{\substack{A,B:\, A\cap B=\emptyset\\ A\cup B=[0,d],\, 0\in A}} (\mu_1|A|+\mu_2|B|)^{-1}\lambda$$
$$\left(\sum_{\substack{x\in A:\\ x+1\in A}} p(1_{A-\{x\}}2_B) + \sum_{y\in B\,:y+1\in A} yp(1_A 2_{B-\{y\}})\right) + o(\lambda^{d+1}) \tag{22}$$

Summarizing (22), (18) and Lemma 1 we represent q_{loss} as follows:

$$q_{\text{loss}} = C(d,\mu_1,\mu_2)\lambda^{d+1} + o(\lambda^{d+1}), \tag{23}$$

where

$$C(d,\mu_1,\mu_2) = \sum_{\substack{\alpha:\, \text{supp}\alpha=[0,d]\\ A(\alpha)\ni 0}} C^{(\alpha)}(d,\mu_1,\mu_2) \tag{24}$$

Here the summation is over all pairs $\alpha = \{A, B\}$ of nonintersecting subsets of V such that $A\cup B = [0,d]$ and $0 \in A$ and the coefficients $C^{(\alpha)}(d,\mu_1,\mu_2)$ can be calculated recurrently with the help of the following formula:

$$C^{(\alpha)}(d,\mu_1,\mu_2) = k_\alpha^{-1} \sum_{\substack{x\in\text{supp}\alpha:\\ x+1\in A(\alpha)}} f_x C^{(\alpha_1)}(d,\mu_1,\mu_2) C^{(\alpha_2)}(d,\mu_1,\mu_2) \tag{25}$$

where

$$f_x = \begin{cases} 1 & \text{if } x\in A,\ x+1\in A; \\ r_x - 1 & \text{if } x\in B,\ x+1\in A; \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

2.4 Program and calculations.

The algorithm described above is realized on IBM PC AT-386 by the Program NETWORK using C-language. It calculates recurrently the coefficient $C(d,\mu_1,\mu_2)$ at $(d+1)-$power of λ of the expansion (23) of q_{loss} using (24) and (25).

Consider a pair (A, B) such that $A\cup B$ is connected, denote $k = |A| + |B|$. Let us numerate the nodes of $A\cup B$ from 1 to k in their natural order. Let us associate with (A, B) a binary $k-$vector v in the following way: $v_i = 0$ if $i \in A$, $v_i = 1$ if $i \in B$. Denote J the binary number corresponding to the vector v. The main function PROB(k, nul, num) calculates the coefficient $c_{i_1\ldots i_k}$ at the main term of the expansion of probability $p_{i_1\ldots i_k}$ in λ where $i_1 = 1$, $i_j = 1$ or 2 for any $j > 1$, $k = |A| + |B|$, $nul = |A|$, $num = J$ using the following formula based on (24) and (25):

$$c_{i_1\ldots i_k} = (\mu_1|A|+\mu_2|B|)^{-1} \sum_{m=1}^{k} f_m c_{i_1\ldots i_{m-1}} c_{i_{m+1}\ldots i_k}$$

where

$$f_m = \begin{cases} 1 & \text{if } i_m = 1,\ i_{m+1} = 1; \\ m-1 & \text{if } i_m = 2,\ i_{m+1} = 1; \\ 0 & \text{otherwise} \end{cases}$$

The results of calculations are stored in a special array $mes[\ ]$. Next the program calculates $C(d, \mu_1, \mu_2)$ using the formula:

$$C(d, \mu_1, \mu_2) = \sum_J PROB(d+1, a, J) \tag{27}$$

where the summation is over all binary numbers J from 0 to $2^d - 1$, a is the number of units i.e. $a = |A|$. The program does not use shift operations and digital operations in order to simplify the procedure of its generalization to the case of network with different types of calls. Text of the program is given in Appendix 1.

Picture 1 shows results of calculations for the case $d = 10$ (the maximum possible value of d is 12) for μ_1 varying from 1.2 to 1.9 and μ_2 =1.0, 1.5 and 2.0.

3 System with several types of calls

Consider now a system serving customers of m different types. For any node x customers of type i form a Poissonian stream with intensity λ_i not depending on x, all streams are independent. Let $\mu_1^{(i)}$ be the serving rate of x-customer of type i at the node x and $\mu_2^{(i)}$ the serving rate of transmitted customer of type i, $i = 1, \ldots, m$.

Denote $Q_x \subset V$ the set of nodes where x-customer can be transmitted and R_x the set of nodes from which a customer can be transmitted to the node x. For example, for the chain of serves described in Section 2

$$Q_x = \{x+1, x+2, \ldots, x+d\}, \quad R_x = \{x-1, x-2, \ldots, x-d\}.$$

For each x a deterministic rule of searching of the free server for an x-customer is given. It means that some numeration of elements of the set Q_x is given and the system is looking for an empty server from Q_x with minimal number. It is convenient to think that a family of order relations $\overset{x}{\prec}$, each on its own set Q_x, is given, x varying in V.

We put $\lambda_i = a_i\lambda$, supposing that λ is small . Similarly to the previous section for any $i = 1, \ldots, m$ $\xi_t(x) = i$ means that there is an x-customer of type i at time t at the node x and $\xi_t(x) = i + m$ means that there is a transmitted customer of type i at the node x at time t. Symbol $*$ means that the corresponding server is occupied, 0 - that it is free, $\diamondsuit$ means that its state is arbitrary.

As in the previous section we may derive Kolmogorov equations for finite dimensional probabilities. For any $x \in Q_y$ denote $Q_y^-(x) \subseteq Q_y$ the set of nodes from Q_y which are less than x according to the given order $\overset{y}{\prec}$. For a fragment of a graph shown on Figure 2, $R_x = \{y, z_6, z_7, z_8\}$, $Q_y = \{z_1, z_2, x, z_3, z_4, z_5\}$, $Q_y^-(x) = \{z_1, z_2\}$ and the numbers on the edges correspond to the order relation on Q_y. Then

$$P(\xi(x) = i)\mu_1^{(i)} = \lambda_i P(\xi(x) = 0), \quad i = 1, \ldots, m; \tag{28}$$

$$P(\xi(x) = i)\mu_2^{(i-m)} = \lambda_{i-m} \sum_{y \in R_x} p(*_y \ *_{Q_y^-(x)} \ 0_x), \quad i = m+1, \ldots, 2m. \tag{29}$$

To present higher order equations we need some definitions. Let us consider a collection α of $2m$ non intersecting subsets of V:

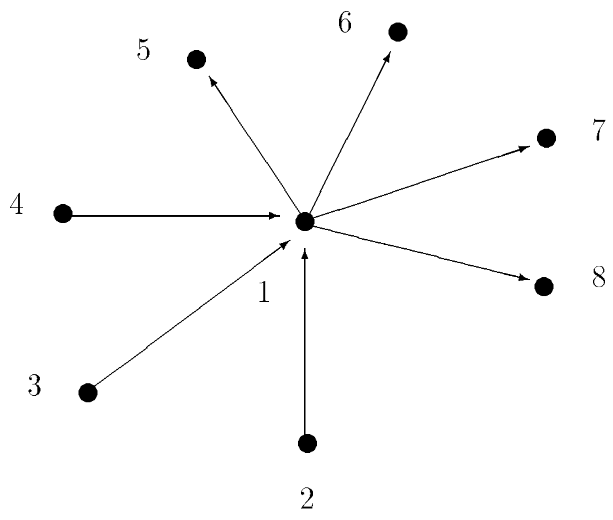

Figure 2: Fragment of a graph.

$\alpha = (A_1, A_2, \ldots, A_m, B_1, \ldots, B_m)$. Denote supp $\alpha = (\cup_i A_i) \cup (\cup_i B_i)$. We use notation s_α for the following configuration:

$$\{\xi(x) = i \text{ for } x \in A_i, \quad \xi(x) = i + m \text{ for } x \in B_i, \quad i = 1, \ldots, m\}$$

and $p(s_\alpha)$ for the corresponding probability. For any $x \in$ supp α denote also $\alpha - x$ a collection obtained from α by subtracting x from the corresponding set, $s_\alpha^{i,x}$ configuration obtained from s_α by changing its value at the node x to $s(x) = i$ and $s_\alpha^{\diamond,x} = s_{\alpha - x}$ the restriction of configuration s_α to the set supp $\alpha - \{x\}$. Then

$$p(s_\alpha)(\sum_{i=1}^{m} \mu_1^{(i)} \mid A_i \mid + \sum_{i=1}^{m} \mu_2^{(i)} \mid B_i \mid) =$$

$$\sum_{i=1}^{m} \lambda_i [\sum_{x \in A_i} p(s_\alpha^{0,x}) + \sum_{y \in B_i} \sum_{z \in R_y} p(s_\alpha^{0,y} \ *_z *_{Q_z^-(y)})]. \tag{30}$$

Iterating (30) we may obtain cluster expansion for probabilities $p(s_\alpha)$ similarly to (12), (13). The proof of the convergence of this expansion is similar to the proof of Theorem 1. We do not describe it here in order to avoid cumbersome notations. The loss probability for the node x

$$q_{\text{loss}}(x) = p(*_{x \cup Q_x}) = \sum_{s_{x \cup Q_x}} p(s_{x \cup Q_x}) \tag{31}$$

where the summation is over all configurations $s_{x \cup Q_x}$ on $x \cup Q_x$ such that for any $y \in (x \cup Q_x)$ $s(y) \in \{1, \ldots, 2m\}$.

The same arguments as in Section 1 show that

$$p(s_\alpha) = C(s_\alpha)(\lambda^{|\mathrm{supp}\alpha|}) + o(\lambda^{|\mathrm{supp}\ \alpha|}) \tag{32}$$

if $s_\alpha(y) \in \{1, \ldots, 2m\}$ for any $y \in \mathrm{supp}\ \alpha$. Hence,

$$q_{\mathrm{loss}}(x) = C_x \lambda^{|x \cup Q_x|} + o(\lambda^{|x \cup Q_x|}) \tag{33}$$

Using (32) we can represent (30) as follows:

$$p(s_\alpha)(\sum_{i=1}^{m} \mu_1^{(i)} \mid A_i \mid + \sum_{i=1}^{m} \mu_2^{(i)} \mid B_i \mid) =$$

$$\sum_{i=1}^{m} \lambda_i [\sum_{x \in A_i} p(s_\alpha^{0,x}) + \sum_{y \in B_i} \sum_{z \in R_y \cap \mathrm{supp}\ \alpha} p(s_\alpha^{0,y})] + o(\lambda^{|\mathrm{supp}\alpha|}) \tag{34}$$

where the last summation is over all nodes z from supp α which can transmit a customer to y and such that $Q_z^-(y) \subseteq \mathrm{supp}\ \alpha$, i.e. all nodes that could accept a customer from z before it is transmitted to y are occupied. Note that

$$p(s_\alpha^{0,x}) = p(s_{\alpha - x}) - \sum_{i=1}^{2m} p(s_\alpha^{i,x})$$

Using this fact and (32) we can rewrite (34) as follows:

$$p(s_\alpha)(\sum_{i=1}^{m} \mu_1^{(i)} \mid A_i \mid + \sum_{i=1}^{m} \mu_2^{(i)} \mid B_i \mid) =$$

$$\sum_{i=1}^{m} \lambda_i [\sum_{x \in A_i} p(s_\alpha^{\diamond,x}) + \sum_{y \in B_i} \sum_{z \in R_y \cap \mathrm{supp}\ \alpha} p(s_\alpha^{\diamond,y})] + o(\lambda^{|\mathrm{supp}\alpha|}) \tag{35}$$

Let $\gamma(\alpha, y)$ denote the number of nodes $z \in \mathrm{supp}\ \alpha$ such that $Q_z^-(y) \subseteq \mathrm{supp}\ \alpha$. Let us define

$$G(s_\alpha, x) = \begin{cases} a_{s_\alpha(x)} & \text{if } s_\alpha(x) \in \{1, \ldots, m\}; \\ a_{s_\alpha(x)-m}\gamma(\alpha, x) & \text{if } s_\alpha(x) \in \{m+1, \ldots, 2m\}. \end{cases} \tag{36}$$

Then (35) becomes

$$p(s_\alpha) = \lambda[(\sum_{i=1}^{m} \mu_1^{(i)} \mid A_i \mid + \sum_{i=1}^{m} \mu_2^{(i)} \mid B_i \mid)]^{-1}$$

$$\sum_{x \in \mathrm{supp}\ \alpha} G(s_\alpha, x) p(s_{\alpha - x}) + o(\lambda^{|\mathrm{supp}\ \alpha|}). \tag{37}$$

Program ANYGRAPH is based on (37), (36). It calculates C_x (see (33)) using (31) and the recurrence relation:

$$C(s_\alpha) = [(\sum_{i=1}^{m} \mu_1^{(i)} \mid A_i \mid + \sum_{i=1}^{m} \mu_2^{(i)} \mid B_i \mid)]^{-1}$$

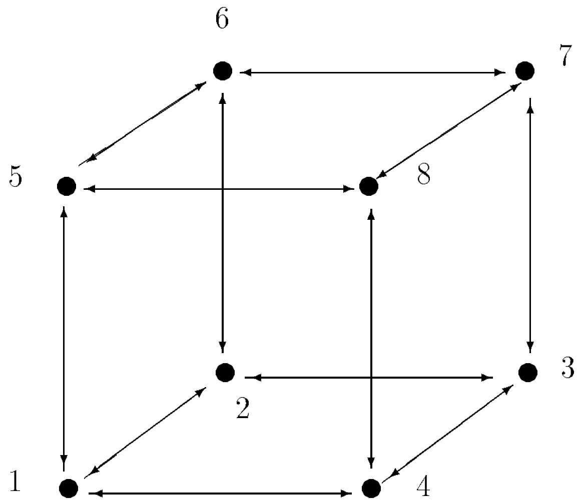

Figure 3: A cube

Server	Looking order
1	2 5 4
2	1 3 6
3	4 7 2
4	3 1 8
5	6 8 1
6	5 2 7
7	8 6 3
8	7 4 5

Table 1: Network 1

$$\sum_{x \in \mathrm{supp}\ \alpha} G(s_\alpha, x) C(s_{\alpha - x}). \tag{38}$$

Program ANYGRAPH is written on C for IBM PC. It contains a graph editor and four main functions for calculations. Function CURR( ) checks the condition $z \cup Q_z^-(y) \subseteq \mathrm{supp}\ \alpha$ (see(34)). Function PKTH( ) calculates the value of $G(s_\alpha, x)$. Function PROB realizes formula (38) and function COEF calculates C_x according to (33). A graph editor permits the user to input and edit graphs and parameters of the system in interactive regime. Text of the program is given in Appendix 2. The maximal possible size of Q_x is 9. This program has been applied to the calculation of the loss probability for the following systems.

1. Consider 8 servers situated at the nodes of a cube and connected along the edges of the cube as it is shown on Figure 3 (network 1).

Table 1 gives the rule of choice of a free server among the neighbors.

Consider also 8 servers situated at the nodes of an octagon. Each server can transmit a call to one of two neighboring servers using the incident edges and also to the next server looking clockwise using a chord as it is shown on Figure 4 (network 2). The order of searching of a free server is the same for all servers. For the server 1 it is the following: 8,2,3. The results of calculations show that the loss probability for network 1

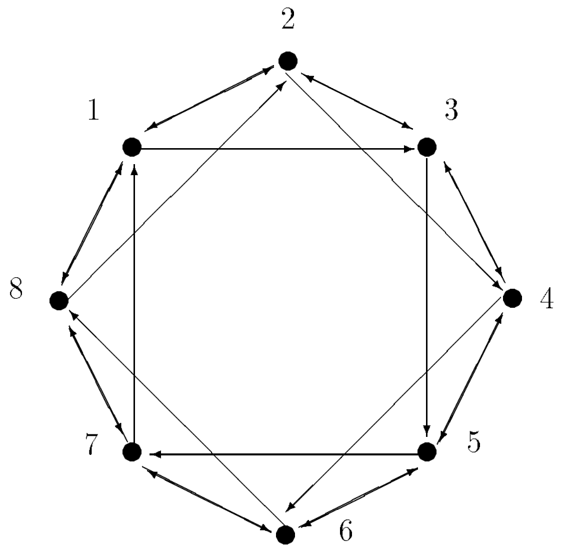

Figure 4: An octagon

is essentially less than for network 2. Thus, for $\mu_1 = 1$ and $\mu_2 = 3$ $C_1 = 2.03274$ for network 1 and $C_1 = 2.7122$ for network 2. For $\mu_1 = 2$ and $\mu_2 = 1$ $C_1 = 0.283135$ for network 1 and $C_1 = 0.483135$ for network 2.

2. Consider a system of servers arranged in a line. Any server can put its customer to the nearest free server situated not far than 3 units to the left or to the right (see Figure 5). Then for $\mu_1 = 2$, $\mu_2 = 1$ $C_1 = 0.9855$.

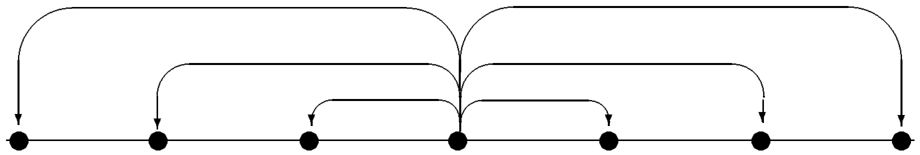

Figure 5: A line

3. For cubic lattice (each edge is two side oriented): for $\mu_1 = 2$, $\mu_2 = 1$ $C = 0.0634485$; for $\mu_1 = 1$, $\mu_2 = 3$ $C = 2.78032$.

4. For triangular lattice (each edge is two side oriented): for $\mu_1 = 2$, $\mu_2 = 1$ $C = 0.232754$; for $\mu_1 = 1$, $\mu_2 = 3$ $C = 5.50574$. In triangular and cubic lattices the size of Q_x is the same, but simulation shows that cubic lattice network is much more effective than the triangular.

5. For hexagonal lattice the result is the same as for the cube (network 1).

Both programs were written with participation of N.Kotovich.

4 Resume

Network. We consider a loss network which consists of a set of independently working nodes (servers). Customers of m different types arrive to each server, forming m input independent Poissonian streams. The intensity of each stream depends on the type of customers but does not depend on the server. A customer, being served at one node, leaves the system.

Discipline. If the node of arrival is empty then the arriving customer immediately occupies it. We call such customers *accepted.*

If the node of arrival is occupied, then arriving customer looks for an empty node among some distinguished nodes. For each node x of arrival this set Q_x of distinguished nodes is fixed once for all. The searching rule (i.e. the ordering of Q_x, one chooses the first empty node in this ordering) is deterministic and is also fixed for each Q_x. We assume that the search takes no time.

If a customer finds an empty server among the servers of this set, it is immediately accepted by this server. We call such a customer *transferred.* If all servers from the set Q_x of search are occupied, the customer is lost.

Parameters of the system. We assume that the serving rate depends on the type of a customer and on whether it was accepted at the node of its arrival or transferred to another node according to the searching rule. So, the system has the following parameters:

- Input intensities depending on the type of calls: $\lambda_i, \quad i = 1, \ldots, m$.
- Serving intensities of accepted customers depending on the type of customers: $\mu_1^{(i)}, \quad i = 1, \ldots, m$.
- Serving intensities of transferred customers depending on the type of customers: $\mu_2^{(i)}, \quad i = 1, \ldots, m$.
- A collection of sets Q_x (x running over all nodes of the system) where a customer can be transmitted in the case when the node of its arrival is occupied.

We suppose that input intensities are small and that they have a common small parameter:

$$\lambda_i = \lambda a_i, \quad i = 1, \ldots, m.$$

Result. We show that the loss probability q_x for any node x in the stationary regime is proportional to λ^{d_x+1}, $d_x = \mid Q_x \mid$ is the number of nodes where a customer can be transmitted in the case that the node x of its arrival is occupied:

$$q_x \approx C_x \lambda^{d_x+1}.$$

We present an algorithm of calculation the coefficient C_x. It is realized on IBM PC. The program is written on C-language and consists of two parts: a graph editor, convenient to describe a network under consideration and define all necessary parameters, and calculations itself. This program was applied to several models of networks. The results of calculations are described in Sections 2.4 and 3.

APPENDIX 1. Program NETWORK

```
#include <stdio.h>
#define MAXL 8190
#define DMAX 12
float mes[MAXL],mu1,mu2;
int d;
float coef(void);
 main()
 {
  printf("  network-2\n");
  printf("\n  d-?(d<=12) ");scanf("%d",&d);
  printf("\n  mu-1-? ");scanf("%f",&mu1);
  printf("\n  mu-2-? ");scanf("%f",&mu2);
  printf("\nC(%d,%g,%g)=%f",d,mu1,mu2,coef());
  }
/*********************************************/
 void decod(int k1,int p,int* dm) /*decimal p to bynary dm[]*/
 {
 int i,n1=p;
 for(i=k1;i;i--)
    {
    dm[i]=n1%2;
    n1/=2;
    }
 }
/*************************************************/
 int code(int k1,int l1,int* dm) /*bynary dm[] to decimal code*/
 {
 int i,co=0;
 for(i=l1;i<=l1+k1-1;i++)
    co=co*2+dm[i];
 return(co);
 }
/*************************************************/
 int dex(int n) /*2 power n*/
 {
 int i,n1=1;
 for(i=1;i<=n;i++)
    n1*=2;
 return(n1);
 }
/*************************************************/
```

```
float prob(int k,int nul,int num) /*P(0,i.1,...,i.(k-1))/(lambda pow k)*/
{
int qi,l,nl,fl,j,comb[DMAX];
float pr;
if (k==1) pr=1/mu1;
   else
     {
     l=dex(k-1)+num-1;
     if (mes[l]>=0) pr=mes[l];
        else
          {
          pr=0;
          nl=1;
          decod(k-1,num,comb);
          if (comb[1]==0) pr+=prob(k-1,nul-1,code(k-2,2,comb));
          if (comb[k-1]==0) pr+=prob(k-1,nul-1,code(k-2,1,comb));
             else
               pr+=prob(k-1,nul,code(k-2,1,comb))*(k-1);
          for(j=1;j<=k-2;j++)
             {
             if (comb[j]==0)
                {
                fl=qi=1;
                nl++;
                }
               else
                 {
                 qi=j;
                 fl=0;
                 }
             if (comb[j+1]==0)
                pr+=qi*prob(j,nl-fl,code(j-1,1,comb))*\
                prob(k-j-1,nul-nl,code(k-j-2,j+2,comb));
             }
          mes[l]=pr/=nul*mu1+(k-nul)*mu2;
          }
     }
return(pr);
}
/********************************************************/
int zero(int k,int p) /*number of nulls*/
{
int aaa[DMAX],i,z=0;
decod(k,p,aaa);
for(i=1;i<=k;i++)
```

```
   z+=aaa[i];
 return(k-z);
 }
/******************************************************/
 float coef() /*C(d,mu1,mu2)*/
 {
 int i,l;
 float c1=0;
 if(d==0)
   return(1/mu1);
 l=dex(d+1)-2;
 for(i=1;i<=l;i++)
    mes[i]=-1;
 l=(l+2)/2-1;
 for(i=0;i<=l;i++)
    c1+=prob(d+1,zero(d,i)+1,i);
 return(c1);
 }
```

APPENDIX 2. Program ANYGRAPH

```
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <bios.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#define MXGR 30
#define MXM 30
#define LNTH 16000
#define DMAX 9
float *mes;
float ci[MXM],mu1[MXM],mu2[MXM];
int d,ncur,mtp,msys,ngr=0,flag,mst=0;
int rad=9;
unsigned ai[DMAX],gmij[DMAX][DMAX];
struct node {
            int lth;
            int serv[DMAX];
            }grph[MXGR];
struct llk {
            int x1;
            int y1;
            int x2;
            int y2;
            int x3;
            int y3;
            int x4;
            int y4;
            int fl;
           }lnk[MXGR][MXGR];
struct pnt {
            int x;
            int y;
           }vrt[MXGR];
struct mrk {
            int x;
            int y;
            int num;
           }mark[MXGR][MXGR];
/***********************/
```

```
int init_mouse(void);
void vizi_cursor(void);
void unvizi_cursor(void);
void set_coor(int,int);
void set_bound(int min_x,int max_x,int min_y,int max_y);
int get_coor(int *x,int *y);
/***********************/
void inpgrph(void),inpdat(void),curr(void),outrez(void);
void dr_link(int,int,int);
void del_link(int,int);
void del_mark(int,int);
void mntxt(void);
int if_link(int,int,int,int);
int new_node(int,int);
void er_link(int,int);
void grmenu(void);
int if_node(int,int,int);
void del_node(int);
void del_all(void);
int check_gr(void);
void clr_txt(void);
void dr_cur(void),er_cur(void);
float coef(void),inpfl(int);
static union REGS regM;
static int   NumButton=0;

 main()
 {
 char d5[30];
 int ii;
 int key,key1;
 int item=1;
 int xx,yy;
 int grmd=2,grdr=VGA;
 mes=(float*)malloc(LNTH*sizeof(float));
 initgraph(&grdr,&grmd,"");
 setbkcolor(LIGHTGRAY);
 cleardevice();
 del_all();
 setfillstyle(1,CYAN);
 bar(180,10,460,80);
 setcolor(YELLOW);
 settextstyle(DEFAULT_FONT,HORIZ_DIR,3);
 outtextxy(217,30,"NETWORK-3");
 settextstyle(SMALL_FONT,HORIZ_DIR,1);
```

```
setcolor(BLUE);
rectangle(15,100,125,270);
rectangle(15,290,125,460);
rectangle(140,100,630,460);
line(140,430,630,430);
line(15,440,125,440);
line(141,440,629,440);
setcolor(BLUE);
line(18,306,122,306);
line(51,293,51,437);
line(88,293,88,437);
setcolor(RED);
outtextxy(19,297,"C(i)");
outtextxy(57,297,"Mu1");
outtextxy(95,297,"Mu2");
if(init_mouse()==0)
  {
  printf("NO MOUSE INSTALLED !!!");
  bioskey(0);
  exit(1);
  }
setfillstyle(1,LIGHTGRAY);
bar(141,101,629,439);
clr_txt();
mntxt();
setcolor(RED);
outtextxy(115,90+30*item,"&");
vizi_cursor();

while(1)
  {
  key=get_coor(&xx,&yy);
  if(key!=0)
     {
     if(key!=0)
         while((key1=get_coor(&xx,&yy))!=0)
            key|=key1;
     if(key==1)
         {
         if(15<xx && xx<125 && 100<yy && yy<270)
            {
            unvizi_cursor();
            setfillstyle(1,LIGHTGRAY);
            bar(114,101,124,269);
            vizi_cursor();
```

```
    item=(int)((yy-85)/30);
    clr_txt();
    switch(item)
        {

        case 1:{
               clr_txt();
               unvizi_cursor();
               outtextxy(150,450,"delete graph(y/n)?");
               if(getch()==121)
                  {
                  ngr=0;
                  del_all();
                  setfillstyle(1,LIGHTGRAY);
                  bar(141,101,629,439);
                  }
               vizi_cursor();
               clr_txt();
               break;
               }
        case 2:{
               grmenu();
               clr_txt();
               mntxt();
               break;
               }
        case 3:{
               inpdat();
               break;
               }
        case 4:{
               unvizi_cursor();
               setcolor(BLUE);
               outtextxy(150,450,"select the node");
               vizi_cursor();
               break;
               }
        }/* switch */
    unvizi_cursor();
    setcolor(BLUE);
    outtextxy(115,90+30*item,"&");
    vizi_cursor();
    }/* if */
 if(140<xx && xx<630 && 100<yy && yy<440)
    {
```

```
            if(item==4)
              {
              for(ii=0;ii<ngr;ii++)
                 {
                 if(if_node(ii,xx,yy))
                   {
                   unvizi_cursor();
                   setfillstyle(1,LIGHTGRAY);
                   bar(16,441,124,459);
                   setcolor(RED);
                   itoa(ii+1,d5,10);
                   outtextxy(19,450,d5);
                   outtextxy(35,450,": C=");
                   ncur=ii;
                   curr();
                   gcvt(coef(),6,d5);
                   outtextxy(70,450,d5);
                   vizi_cursor();
                   break;
                   }
                 }/* for */
              }/* if */
            }/* if */
         }/* if */

      }/* if */
   if(bioskey(1) && bioskey(0)==0x11B)
         break;
   if(item==5)
      break;

   }/* while */

 closegraph();
 restorecrtmode();
  }/*main*/
/***********************************************/

int decod(int p,int* dm,int* zero) /*decimal p to (2*m+1) dm[]*/
 {
 int i,n1=p,cz=0;
 for(i=d;i>=0;i--)
    {
    dm[i]=n1%msys;
    n1/=msys;
```

```
   if(dm[i]==0)
     {
     zero[cz]=i;
     cz++;
     }
   }
   return(cz);
}/*decod*/
/*****************************************************/
int dex(int n) /*2*m+1 power n*/
{
int i,n1=1;
for(i=1;i<=n;i++)
   n1*=msys;
return(n1);
} /*dex*/
/*****************************************************/
int pkth(int i1,int* zero,int nz) /*calculating p-i1 */
{
unsigned tp=0,tq;
int j,pth=0;
for(j=0;j<nz;j++)
   tp|=gmij[i1][zero[j]];
tq=ai[i1]&(~tp);
for(j=0;j<=d;j++)
   pth+=(tq>>j)&(1);
return(pth);
} /* pkth */
/************************************************************/
float prob(int num) /*P(0,i.1,...,i.d)/(lambda pow k)*/
{
int nzer,j,a,comb[DMAX],null[DMAX],mult=1;
float pr=0,sum=0;
flag=0;
nzer=decod(num,comb,null);
for(j=d;j>=0;j--)
   {
   a=comb[j];
   if((a>=1)&&(a<=mtp))
     {
     sum+=mu1[a];
     pr+=ci[a]*mes[num-mult*a];
     }
   if((a>=mtp+1)&&(a<=mtp+mtp))
     {
```

```
      sum+=mu2[a-mtp];
      pr+=ci[a-mtp]*pkth(j,null,nzer)*mes[num-mult*a];
      }
    mult*=msys;
    }
 mes[num]=pr/=sum;
 if(nzer)
   flag=1;
 return(pr);
 } /* prob */
/*******************************************************************/
 float coef() /*C(d,mu1,mu2)*/
 {
 int i,l;
 float c1=0,b;
 l=dex(d+1);
 mes[0]=1;
 for(i=1;i<=l-1;i++)
    {
    b=prob(i);
    if(flag==0)
      c1+=b;
    }
 return(c1);
 } /*coef */
/******************************************************/
 float inpfl(int color)
 {
 float p12=0,dc=1;
 char d2[24];
 int mcon,aa,bb,sff=0;
 unvizi_cursor();
 setfillstyle(1,color);
 dr_cur();
 zz1:mcon=getch();
 while((mcon>47 && mcon<57) || mcon==8 )
    {
    if(mcon==8)
      {
      er_cur();
      aa=getx();
      bb=gety();
      bar(aa-8,bb-4,aa,bb+6);
      moverel(-8,0);
      p12=floor(p12/10);
```

```
      dr_cur();
      goto nue;
      }
    mcon-=48;
    p12=p12*10+mcon;
    itoa(mcon,d2,10);
    er_cur();
    outtext(d2);
    dr_cur();
    nue:mcon=getch();
    }
if(mcon==13)
    {
    er_cur();
    vizi_cursor();
    return(p12);
    }
er_cur();
outtext(".");
dr_cur();
mcon=getch();
while(mcon!=13 && (mcon==8 || (mcon>47 && mcon<57)))
    {
    if(mcon==8)
      {
      er_cur();
      aa=getx();
      bb=gety();
      bar(aa-8,bb-4,aa,bb+6);
      moverel(-8,0);
      if(fabs(dc-1)<0.1)
        {
        dr_cur();
        sff=1;
        break;
        }
      p12=floor(p12/dc/10)*dc*10;
      dc*=10;
      dr_cur();
      goto ue;
      }
    mcon-=48;
    dc/=10;
    p12+=mcon*dc;
    itoa(mcon,d2,10);
```

```
   er_cur();
   outtext(d2);
   dr_cur();
   ue:mcon=getch();
   }
if(sff==1)
   {
   sff=0;
   goto zz1;
   }
er_cur();
vizi_cursor();
return(p12);
}
/**********************************************************/
void inpdat()
{
int i,ww;
clr_txt();
unvizi_cursor();
setcolor(LIGHTGRAY);
bar(16,307,50,439);
bar(52,307,87,439);
bar(89,307,124,439);
setcolor(RED);
outtextxy(145,450,"enter number of types of calls:");
moveto(410,450);
vizi_cursor();
mtp=(int)(inpfl(LIGHTGRAY));
msys=mtp*2+1;

for(i=1;i<=mtp;i++)
   {
   ww=306+11*i;
   moveto(17,ww);
   ci[i]=inpfl(LIGHTGRAY);

   moveto(53,ww);
   mu1[i]=inpfl(LIGHTGRAY);

   moveto(90,ww);
   mu2[i]=inpfl(LIGHTGRAY);
   }
} /* inpdat */
/************************************************************/
```

```
void curr()
{
unsigned flee;
int i,j,k,l,tra[DMAX],zz1,zz3;
d=grph[ncur].lth;
if((int)(pow(msys,d+1))>=LNTH)
  {
  unvizi_cursor();
  clr_txt();
  outtextxy(150,450,"sorry,lack of memory");
  getch();
  clr_txt();
  mst=1;
  vizi_cursor();
  goto qqq;
  }
for(i=1;i<=d;i++)
   tra[i]=grph[ncur].serv[i-1];
tra[0]=ncur;
for(i=0;i<=d;i++)
   {
   ai[i]=0;
   for(j=0;j<=d;j++)
      {
      flee=0;
      if(i==j)
        continue;
      for(k=0;k<grph[tra[j]].lth;k++)
         {
         zz3=0;zz1=grph[tra[j]].serv[k];
         for(l=0;l<=d;l++)
            {
            if(zz1==tra[l])
              {
              zz3=1;
              break;
              }
            }
         if(zz3==0)
           break;
         if(zz1==tra[i])
           {
           flee=1;
           break;
           }
```

```
          } /* k */
        ai[i]|=(flee<<j);
        } /* j */
    } /* i */
 for(i=0;i<=d;i++)
    {
    for(j=0;j<=d;j++)
        {
        if(j==i)
          continue;
        gmij[i][j]=1<<j;
        for(k=0;k<=d;k++)
            {
            if(k==i||k==j)
              continue;
            flee=0;
            for(l=0;l<grph[tra[k]].lth;l++)
                {
                if(tra[i]==grph[tra[k]].serv[l])
                  break;
                if(tra[j]==grph[tra[k]].serv[l])
                  {
                  flee=1;
                  break;
                  }
                }/* l */
            gmij[i][j]|=(flee<<k);
            }/* k */
        }/* j */
    }/* i */
 qqq:;
 }/* curr */
/***************************************************/



            /*  work with mouse */

/******************/
/****************/
int comm_mouse(int ax,int bx,int cx,int dx)
{
 regM.x.ax=ax;
 regM.x.bx=bx;
 regM.x.cx=cx;
```

```
 regM.x.dx=dx;
 int86(0x33,&regM,&regM);
 return(0);
}
/***************/
int init_mouse(void)
{
 if(NumButton) return(0xFF);
 comm_mouse(0,0,0,0);
 NumButton=regM.x.bx;
 return(regM.x.ax);
}
/***************/
void vizi_cursor(void)
{
 comm_mouse(1,0,0,0);
}
/**************/
void unvizi_cursor(void)
{
 comm_mouse(2,0,0,0);
}
/**************/
int get_coor(int *x,int *y)
{
 comm_mouse(3,0,0,0);
 *x=regM.x.cx;
 *y=regM.x.dx;
 return((regM.x.bx&7));   /* bx&1 - left bx&2 - right bx&4 - middle */
}
/**************/
void set_coor(int x,int y)
{
 comm_mouse(4,0,x,y);
}
/**************/
void set_bound(int min_x,int max_x,int min_y,int max_y)
{
 comm_mouse(7,0,min_x,max_x);
 comm_mouse(8,0,min_y,max_y);
}
/*******************************/

/********************************/
void dr_link(int n,int m,int col)
```

```
 {
 int x1,x2,y1,y2,f1,f2,f3,f4,g1,g2,g3,g4;
 float tt,root,a1,b1;
 int a2,b2;
 unvizi_cursor();
 x1=vrt[n].x;
 y1=vrt[n].y;
 x2=vrt[m].x;
 y2=vrt[m].y;
 root=sqrt(pow((x1-x2),2)+pow((y1-y2),2));
 tt=(rad+2)/root;
 a2=(int)(5*(y2-y1)/root);
 b2=(int)(-5*(x2-x1)/root);
 f1=lnk[n][m].x1=(int)(x1+(x2-x1)*tt)+a2;
 g1=lnk[n][m].y1=(int)(y1+(y2-y1)*tt)+b2;
 f2=lnk[n][m].x2=(int)(x1+(x2-x1)*(1-tt))+a2;
 g2=lnk[n][m].y2=(int)(y1+(y2-y1)*(1-tt))+b2;
 a1=x1+(x2-x1)*(1-tt-8/root);
 b1=y1+(y2-y1)*(1-tt-8/root);
 f3=lnk[n][m].x3=(int)(a1+2*(y2-y1)/root)+a2;
 g3=lnk[n][m].y3=(int)(b1-2*(x2-x1)/root)+b2;
 f4=lnk[n][m].x4=(int)(a1-2*(y2-y1)/root)+a2;
 g4=lnk[n][m].y4=(int)(b1+2*(x2-x1)/root)+b2;
 lnk[n][m].fl=1;
 setcolor(col);
 line(f1,g1,f2,g2);
 line(f3,g3,f2,g2);
 line(f4,g4,f2,g2);
 vizi_cursor();
 }/* dr_link */
/*******************************************/
void del_link(int n,int m)
 {
 int a,i;
 char d3[20];
 unvizi_cursor();
 er_link(n,m);
 if(mark[n][m].num==0)
    {
    vizi_cursor();
    return;
    }
 setfillstyle(SOLID_FILL,LIGHTGRAY);
 bar(mark[n][m].x,mark[n][m].y-6,mark[n][m].x+8,mark[n][m].y+6);
 a=mark[n][m].num;
```

```
 mark[n][m].num=0;
 grph[n].lth--;
 for(i=0;i<MXGR;i++)
    {
     if(lnk[n][i].fl==1 && mark[n][i].num>a)
         {
         del_mark(n,i);
         mark[n][i].num--;
         setcolor(RED);
         itoa(mark[n][i].num,d3,10);
         outtextxy(mark[n][i].x,mark[n][i].y,d3);
         }
    }
 vizi_cursor();
 }/* del_link */
/**********************************************/
int if_link(int n,int m,int x,int y)
 {
 int x1,x2,y1,y2;
 float rt,d;
 if(lnk[n][m].fl==0)
    return(0);
 x1=lnk[n][m].x1;
 y1=lnk[n][m].y1;
 x2=lnk[n][m].x2;
 y2=lnk[n][m].y2;
 rt=sqrt(pow((x1-x2),2)+pow((y1-y2),2));
 d=fabs((y2-y1)/rt*x-(x2-x1)/rt*y+y1/rt*x2-x1/rt*y2);
 if(d<4 && abs(x1-x)+abs(x2-x)<=abs(x1-x2)+3 && abs(y1-y)+abs(y2-y)<
=abs(y1-y2)+3)
     return(1);
   else
     return(0);
 }/* if_link */
/*****************************************************/
int new_node(int x,int y)
 {
 int i;
 char d3[20];
 unvizi_cursor();
 if(140+rad+1>x || x>630-rad-1 || 100+rad+1>y || y>440-rad-1)
    {
    vizi_cursor();
    return(0);
    }
```

```
 for(i=0;i<ngr;i++)
    {
     if(if_node(i,x,y))
       {
       vizi_cursor();
       return(0);
       }
    }
 ngr++;
 grph[ngr-1].lth=0;
 vrt[ngr-1].x=x;
 vrt[ngr-1].y=y;
 setcolor(BLUE);
 circle(x,y,rad);
 setcolor(YELLOW);
 itoa(ngr,d3,10);
 if(ngr>9)
    outtextxy(x-7,y-3,d3);
   else
    outtextxy(x-3,y-2,d3);
 vizi_cursor();
 return(1);
 }/* new_node */
/***********************************/
int if_node(int n,int x,int y)
 {
 if(sqrt(pow((vrt[n].x-x),2)+pow((vrt[n].y-y),2)) < 10)
     return(1);
   else
     return(0);
 }/* if_node */
/***********************************/
void del_node(int n)
 {
 char d3[20];
 int x,y,i,j;
 unvizi_cursor();
 x=vrt[n].x;
 y=vrt[n].y;
 setcolor(LIGHTGRAY);
 setfillstyle(1,LIGHTGRAY);
 fillellipse(x,y,rad,rad);
 for(i=0;i<ngr;i++)
    {
    if(lnk[n][i].fl)
```

```
      del_link(n,i);
    if(lnk[i][n].fl)
      del_link(i,n);
    }
 for(i=0;i<ngr-1;i++)
    for(j=0;j<ngr-1;j++)
      {
      if(i<n && j>=n)
         {
         lnk[i][j]=lnk[i][j+1];
         mark[i][j]=mark[i][j+1];
         }
      if(i>=n && j<n)
         {
         lnk[i][j]=lnk[i+1][j];
         mark[i][j]=mark[i+1][j];
         }
      if(i>=n && j>=n)
         {
         lnk[i][j]=lnk[i+1][j+1];
         mark[i][j]=mark[i+1][j+1];
         }
      }/* for */
 for(i=n;i<ngr-1;i++)
    {
    grph[i].lth=grph[i+1].lth;
    vrt[i]=vrt[i+1];

    setcolor(LIGHTGRAY);
    setfillstyle(1,LIGHTGRAY);
    fillellipse(vrt[i].x,vrt[i].y,rad-1,rad-1);
    setcolor(YELLOW);
    itoa(i+1,d3,10);
    if(i+1<10)
      outtextxy(vrt[i].x-3,vrt[i].y-2,d3);
     else
      outtextxy(vrt[i].x-7,vrt[i].y-3,d3);
    }
 ngr--;
 vizi_cursor();
 }/* del_node */
/********************************************/
int check_gr()
 {
 int i,j;
```

```
 for(i=0;i<ngr;i++)
    for(j=0;j<grph[i].lth;j++)
       grph[i].serv[j]=-1;
 for(i=0;i<MXGR;i++)
   for(j=0;j<MXGR;j++)
      if(lnk[i][j].fl==1 && mark[i][j].num>0 && mark[i][j].num<MXGR-1)
         grph[i].serv[mark[i][j].num-1]=j;
 for(i=0;i<ngr;i++)
    for(j=0;j<grph[i].lth;j++)
       if(grph[i].serv[j]<0)
          return(i+1);
 return(100);
 }/* check_gr */
/********************************************/
void mark_link(int n,int m)
 {
  int ke,ke1,x,y;
  del_mark(n,m);
  if(mark[n][m].num>0)
     grph[n].lth--;
  dr_link(n,m,BLUE);
  while(1)
     {
     ke=get_coor(&x,&y);
     if(ke!=0)
         {
         if(ke!=0) while((ke1=get_coor(&x,&y))!=0) ke|=ke1;
         if(ke==2)
            {
            moveto(x,y);
            setcolor(RED);
            mark[n][m].num=(int)(inpfl(LIGHTGRAY)+0.1);
            mark[n][m].x=x;
            mark[n][m].y=y;
            grph[n].lth++;
            dr_link(n,m,RED);
            break;
            }
         }
     }
 }/* mark_link */
/********************************************/
void del_mark(int n,int m)
 {
  unvizi_cursor();
```

```
  setfillstyle(1,LIGHTGRAY);
  if(mark[n][m].num<10)
      bar(mark[n][m].x,mark[n][m].y-5,mark[n][m].x+8,mark[n][m].y+6);
     else
      bar(mark[n][m].x,mark[n][m].y-5,mark[n][m].x+16,mark[n][m].y+6);
  setcolor(RED);
  vizi_cursor();
 }/* del_mark */
/**********************************************/
void grmenu()
 {
 int ky,ky1,x,y,a2,g5,h5,i,j,n3,m3,itm=1;
 int nerr;
 char d3[20];
 triam:setfillstyle(1,LIGHTGRAY);
 unvizi_cursor();
 bar(16,101,124,269);
 setcolor(BLUE);
 outtextxy(18,112,"new node");
 outtextxy(18,136,"delete node");
 outtextxy(18,160,"new link");
 outtextxy(18,184,"delete link");
 outtextxy(18,208,"mark link");
 outtextxy(18,232,"delete graph");
 outtextxy(18,258,"menu");
 outtextxy(115,87+itm*24,"#");
 vizi_cursor();
 while(1)
   {

   ky=get_coor(&x,&y);
   if(ky!=0)
     {
     if(ky!=0) while((ky1=get_coor(&x,&y))!=0) ky|=ky1;
     if(ky==1)
       {
       if(15<x && x<105 && 100<y && y<270)
         {
         unvizi_cursor();
         setfillstyle(1,LIGHTGRAY);
         bar(114,101,124,269);
         itm=(int)((y-88)/24);
         if(itm<1)
           itm=1;
           clr_txt();
```

```
          vizi_cursor();
          setcolor(BLUE);
          unvizi_cursor();
          switch(itm)
             {
             case 1:{
                    outtextxy(150,450,"use the LB to place the new
node");
                    break;
                    }
             case 2:{
                    outtextxy(150,450,"select the node you want to
delete");
                    break;
                    }
             case 3:{
                    outtextxy(150,450,"use the LB to select the
first node,");
                    break;
                    }
             case 4:{
                    outtextxy(150,450,"use the LB to select the
link you want to delete");
                    break;
                    }
             case 5:{
                    outtextxy(160,450,"use the LB to select the link,");

                    break;
                    }
             case 6:{
                    clr_txt();
                    unvizi_cursor();
                    outtextxy(150,450,"delete graph(y/n)?");
                    if(getch()==121)
                       {
                       ngr=0;
                       del_all();
                       setfillstyle(1,LIGHTGRAY);
                       bar(141,101,629,439);
                       }
                    vizi_cursor();
                    clr_txt();
                    break;
                    }
```

```
                }
            setcolor(BLUE);
            outtextxy(115,87+itm*24,"#");
            vizi_cursor();
            }
        if(140<x && x<630 && 100<y && y<440)
            {
            setcolor(LIGHTRED);
            switch(itm)
                {
                 case 1:{
                        if(new_node(x,y)==0)
                            {
                            clr_txt();
                            outtextxy(150,450,"can´t place new
node here; press any key");
                            bioskey(0);
                            clr_txt();
                            setcolor(BLUE);
                            outtextxy(150,450,"use the LB to place
the new node");

                            continue;
                            }
                        break;
                        }
                 case 2:{
                        for(i=0;i<ngr;i++)
                           {
                           if(if_node(i,x,y))
                               del_node(i);
                           }
                        break;
                        }
                 case 3:{
                        n3=-1;
                        for(i=0;i<ngr;i++)
                           {
                           if(if_node(i,x,y))
                              n3=i;
                           }
                        if(n3<0)
                             continue;
                          else
                            {
                            clr_txt();
```

```
                              setcolor(BLUE);
                              outtextxy(160,450,"and the RB to
select the second");
                              while(1)
                                  {

                                  ky=get_coor(&x,&y);
                                  if(ky!=0)
                                     {
                                     if(ky!=0)
while((ky1=get_coor(&x,&y))!=0) ky|=ky1;

                                     if(ky==2)
                                        {
                                        m3=-1;
                                        for(i=0;i<ngr;i++)
                                           if(if_node(i,x,y))
                                              m3=i;
                                        if(m3<0 || n3==m3)
                                              continue;
                                          else
                                            {
                                            dr_link(n3,m3,BLUE);
                                            break;
                                            }
                                        }
                                     }
                                  if(bioskey(1) && bioskey(0)==0x11B)
                                     break;
                                  }/* while */
                              clr_txt();
                              outtextxy(150,450,"use the LB to
select the first node,");
                              }
                          break;
                          }
                   case 4:{
                          a2=0;
                          for(i=0;i<ngr;i++)
                             for(j=0;j<ngr;j++)
                                if(if_link(i,j,x,y))
                                   {
                                   g5=i;
                                   h5=j;
                                   a2++;
```

```
                                }
                          if(a2==0)
                            continue;
                          if(a2>1)
                            {
                            clr_txt();
                            outtextxy(150,450,"please, select the
link carefully");

                            bioskey(0);
                            clr_txt();
                            setcolor(BLUE);
                            outtextxy(150,450,"use the LB to select
the link you want to delete");
                            continue;
                            }
                          del_link(g5,h5);
                          break;
                          }
                   case 5:{
                          a2=0;
                          for(i=0;i<ngr;i++)
                             for(j=0;j<ngr;j++)
                                if(if_link(i,j,x,y))
                                   {
                                   g5=i;
                                   h5=j;
                                   a2++;
                                   }
                          if(a2==0)
                            continue;
                          if(a2>1)
                            {
                            clr_txt();
                            outtextxy(150,450,"please, select the
link carefully");

                            bioskey(0);
                            clr_txt();
                            setcolor(BLUE);
                            outtextxy(160,450,"use the LB to select
the link,");

                            continue;
                            }
                          clr_txt();
                          setcolor(BLUE);
                          outtextxy(150,450,"and the RB to place the
```

```
 marker");
                          mark_link(g5,h5);
                          clr_txt();
                          setcolor(BLUE);
                          outtextxy(160,450,"use the LB to select
 the link,");
                          break;
                          }
                   case 6:{
                          }
                  }/* switch */
              }/* if */
          }/* if */
        if(itm==7)
           break;
        }/* if */
   if(bioskey(1) && bioskey(0)==0x11B)
      break;
   }/* while */
 if((nerr=check_gr())!=100)
   {
   clr_txt();
   setcolor(RED);
   itoa(nerr,d3,10);

   outtextxy(150,450,"the graph is not valid,error in vertex");
   outtextxy(461,450,d3);
   goto triam;
   }

 }/* grmenu */
/********************************************/
void clr_txt()
 {
 unvizi_cursor();
 setfillstyle(SOLID_FILL,LIGHTGRAY);
 bar(141,441,629,459);
 vizi_cursor();
 }/* clr_txt */
/******************************************/
void dr_cur()
 {
 int a,b;
 unvizi_cursor();
 a=getx();
```

```
 b=gety();
 setcolor(RED);
 lineto(a,b-2);
 lineto(a,b+7);
 moveto(a,b);
 vizi_cursor();
 }/* dr_cur */
/****************************************/
void er_cur()
 {
 int a,b;
 unvizi_cursor();
 a=getx();
 b=gety();
 setcolor(LIGHTGRAY);
 lineto(a,b-2);
 lineto(a,b+7);
 moveto(a,b);
 setcolor(RED);
 vizi_cursor();
 }/* er_cur */
/****************************************/
void mntxt()
 {
 unvizi_cursor();
 setfillstyle(1,LIGHTGRAY);
 bar(16,101,124,269);
 setcolor(BLUE);
 outtextxy(20,120,"del. graph");
 outtextxy(20,150,"edit graph");
 outtextxy(20,180,"intens.");
 outtextxy(20,210,"result");
 outtextxy(20,240,"quit");
 vizi_cursor();
 }/* mntxt */
/***************************************/
void er_link(int n,int m)
 {
 unvizi_cursor();
 setcolor(LIGHTGRAY);
 line(lnk[n][m].x1,lnk[n][m].y1,lnk[n][m].x2,lnk[n][m].y2);
 line(lnk[n][m].x3,lnk[n][m].y3,lnk[n][m].x2,lnk[n][m].y2);
 line(lnk[n][m].x4,lnk[n][m].y4,lnk[n][m].x2,lnk[n][m].y2);
 lnk[n][m].fl=0;
 vizi_cursor();
```

```
 }/* er_link */
/**************************************/
void del_all()
 {
 int ii,jj;
 for(ii=0;ii<MXGR;ii++)
    {
    grph[ii].lth=0;
    for(jj=0;jj<MXGR;jj++)
       {
       lnk[ii][jj].fl=0;
       mark[ii][jj].num=0;
       }
    }/* for */
 }/* del_all */
/**************************************/
```

References

[1] Kelly F.P. Loss Networks. *Annals of Applied Probability*, 1991, Vol.1, No.3, 319–378.

[2] Feller W. *An Introduction to Probability Theory and its Applications*, Vol.2. Wiley, New York (1971).

[3] Botvich D.D., Fayolle G., Malyshev V.A. Loss networks in thermodynamical limit. Rapport de Recherche. INRIA,1994. Proceedings of 11 International Conference on Analysis and Optimization of Systems. Discrete Event Systems. June 15-17, 1994, Ecole des Mines, Sophia-Antipolis (France).

[4] Malyshev V.A., Minlos R.A.*Gibbs Random Fields. Method of Cluster Expansions.* Kluwer, Dordrecht, 1985.