Естественные модели параллельных вычислений

Ершов Н.М., Попова Н.Н.

ershovnm@gmail.com
popova@cs.msu.su

30 июня 2012

Естественные вычисления

- Теория *естественных вычислений* (Natural Computing), относительно новый раздел науки, образовавшийся на стыке математики, информатики и естественных наук (прежде всего биологии).
- Тематика естественных вычислений включает в себя как уже классические разделы (клеточные автоматы, искусственные нейронные сети), так и относительно новые, появившиеся в последние 10-20 лет (ДНК-вычисления, роевой интеллект).
- Несмотря на биологическое «происхождение» большинства таких моделей, они находят широчайшее применение практически во всех областях, связанных с компьютерной обработкой данных: при моделировании в физике, химии, биологии, экономике и т. п.; в интеллектуальном анализе данных (Data Mining); при распознавании образов; для управления различными сложными системами и т. д.

История естественных вычислений

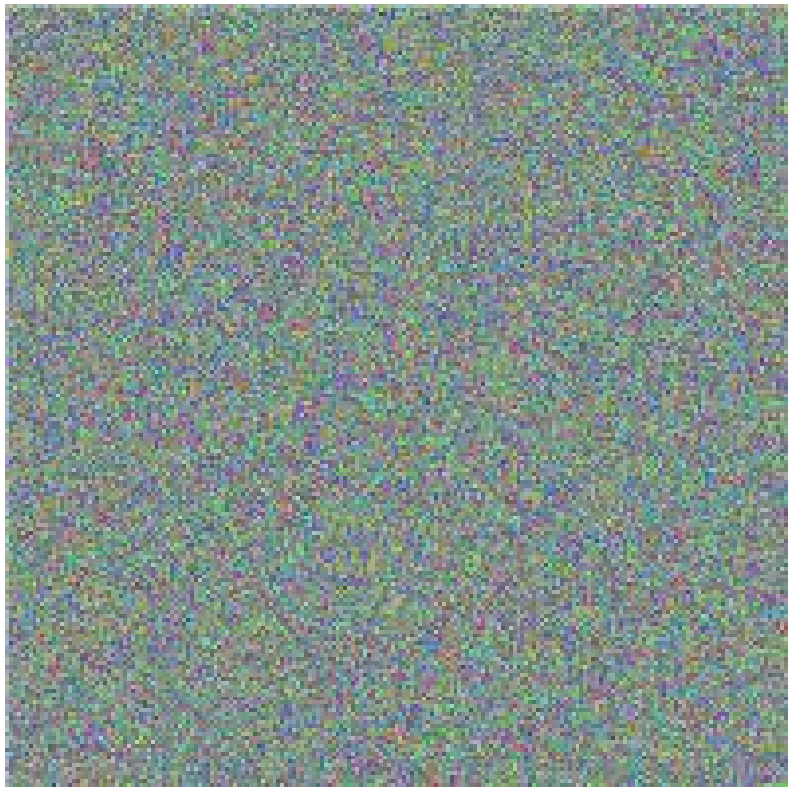
- Несмотря на то, что как раздел науки теория естественных вычислений все еще продолжает формироваться, история ее отдельных подразделов насчитатывает уже десятки лет.
- В частности, одним из авторов исторически первой модели естественных вычислений был не кто иной, как Джон фон Нейман.
- В 1940-х годах им была предложена модель клеточных автоматов, целью было моделирование процессов самовоспроизведения в живой природе.
- Таким образом, фон Нейман оказывается автором сразу двух алгоритмических парадигм — архитектуры фон Неймана *последовательного* двоичного компьютера и высоко *параллельных* алгоритмически универсальных (как это было показано позже) клеточных автоматов.

1951

Клеточные автоматы

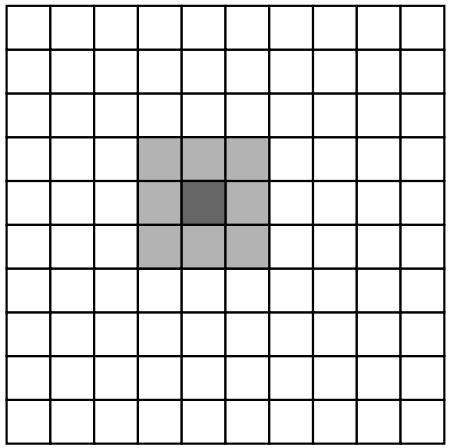
Джон фон Нейман

Станислав Улам

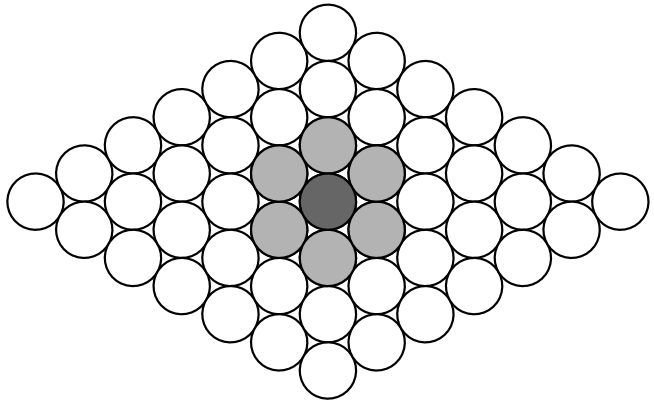


Клеточные пространства

Одномерная решетка



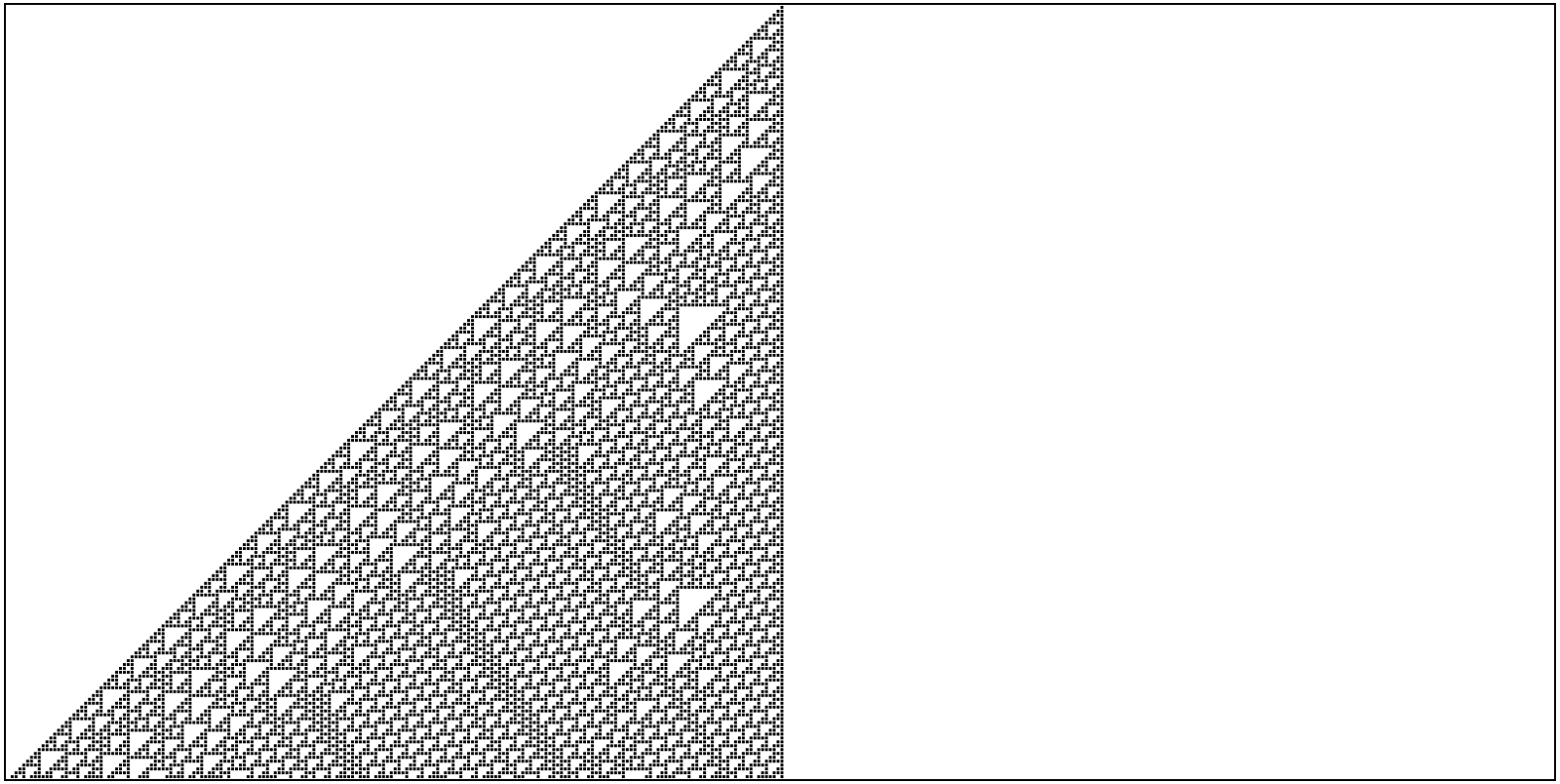
Двумерная прямоугольная решетка



Двумерная гексагональная решетка

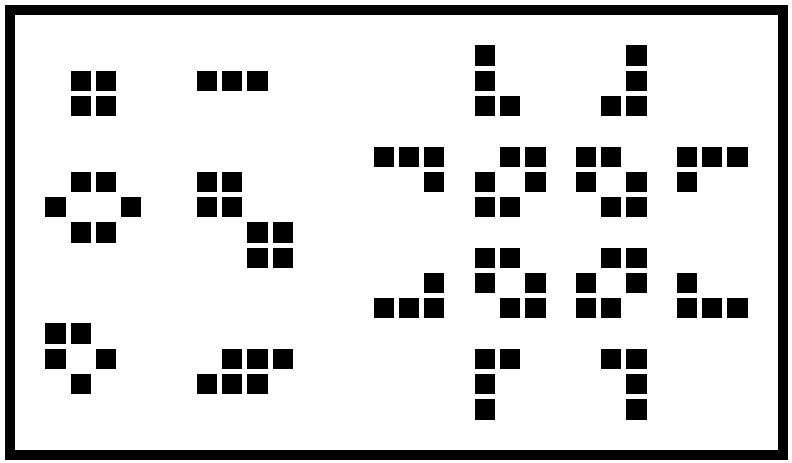
Светло-серым цветом показаны локальные окрестности выделенных (темно-серым цветом) клеток.

Элементарный автомат №110

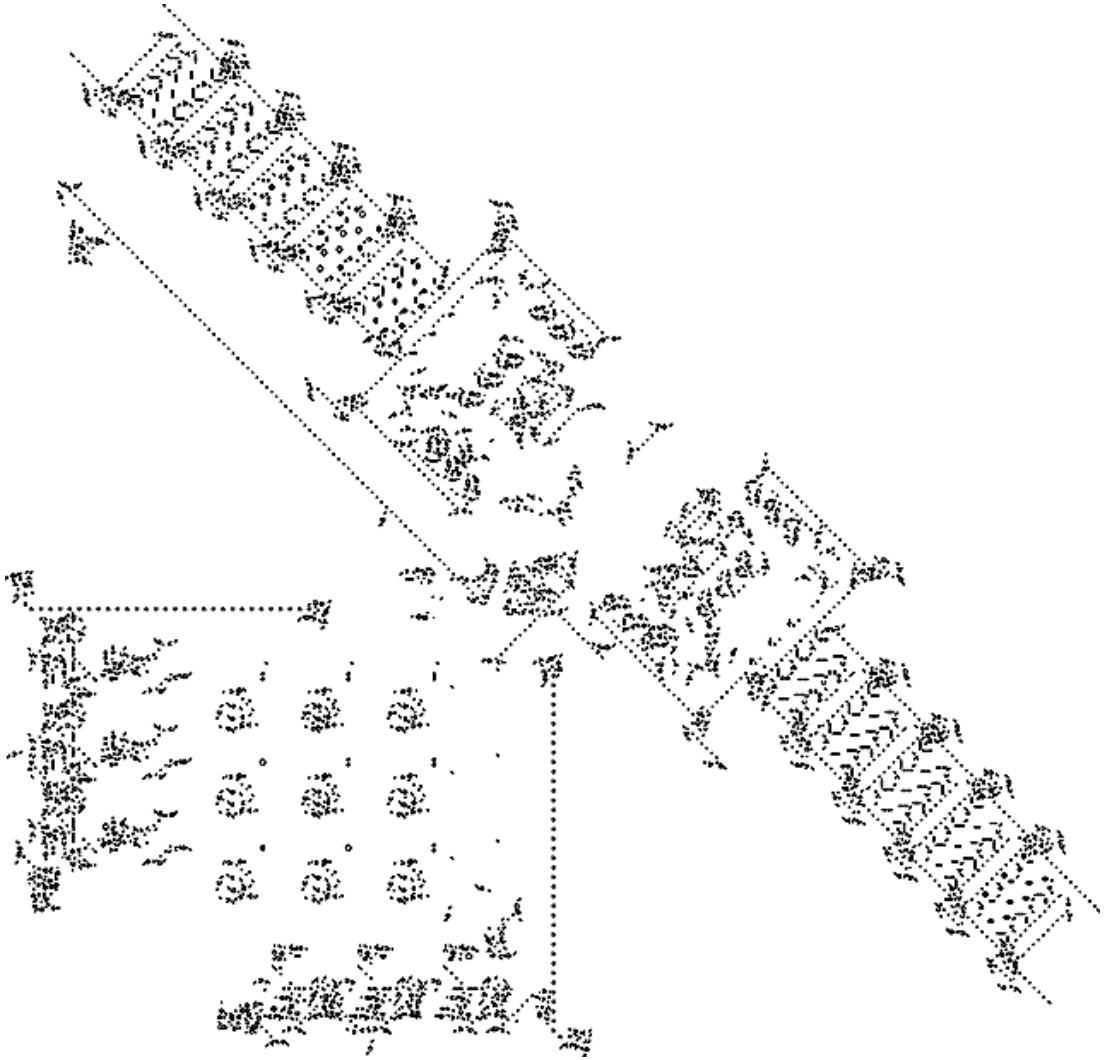


Игра «Жизнь»

- Эта игра представляет собой двоичный двумерный клеточный автомат с окрестностью Мура ранга $r = 1$, придуманный английским математиком Джоном Конвеем в 1970 году.
- Два состояния клеток в этом автомате интерпретируются биологическим образом:
  - 1 — «живая» клетка (или заселенная), изображается черным (закрашенным) квадратом;
  - 0 — «мертвая» клетка (или пустая), изображается белым (незакрашенным) квадратом.



Реализация машины Тьюринга в игре «Жизнь»



1957

Искусственные нейронные сети

Фрэнк Розенблатт

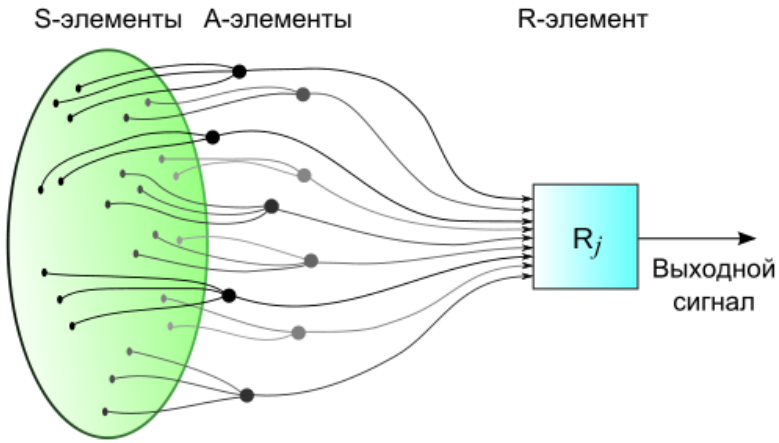


Схема строения нейрона

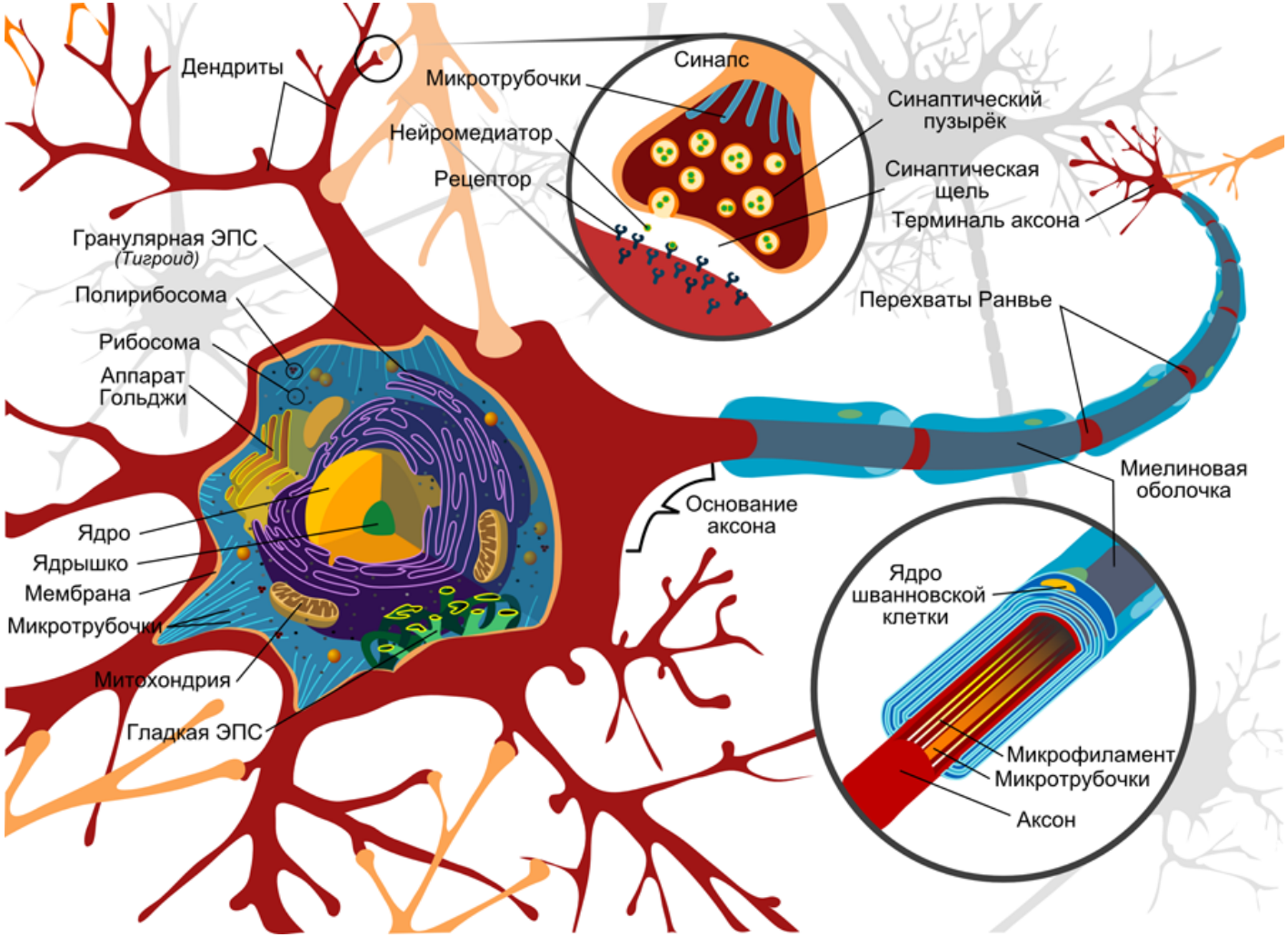
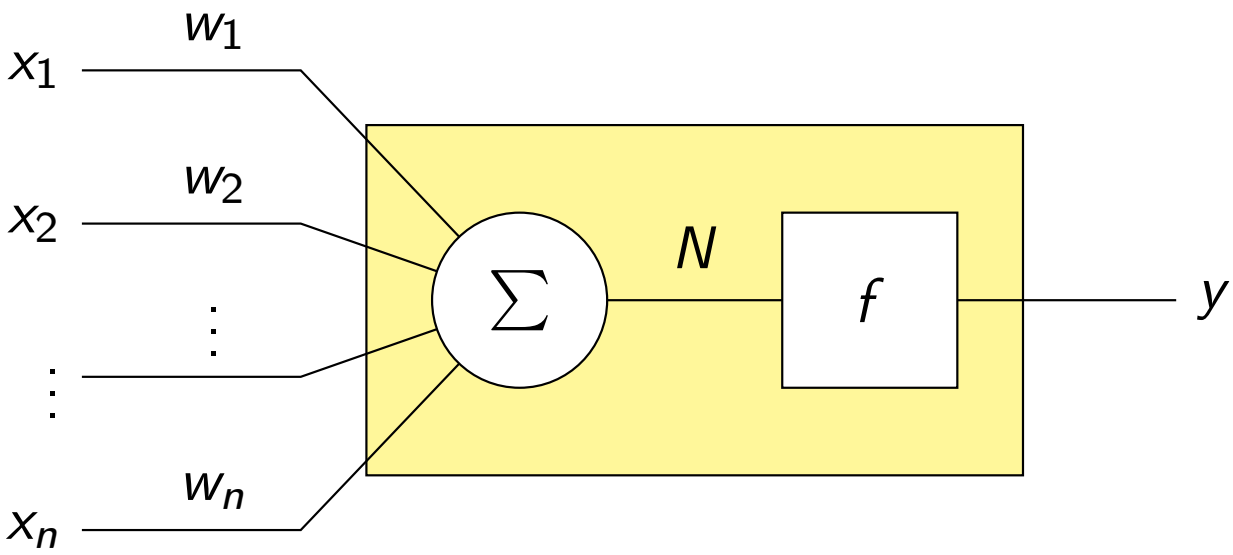
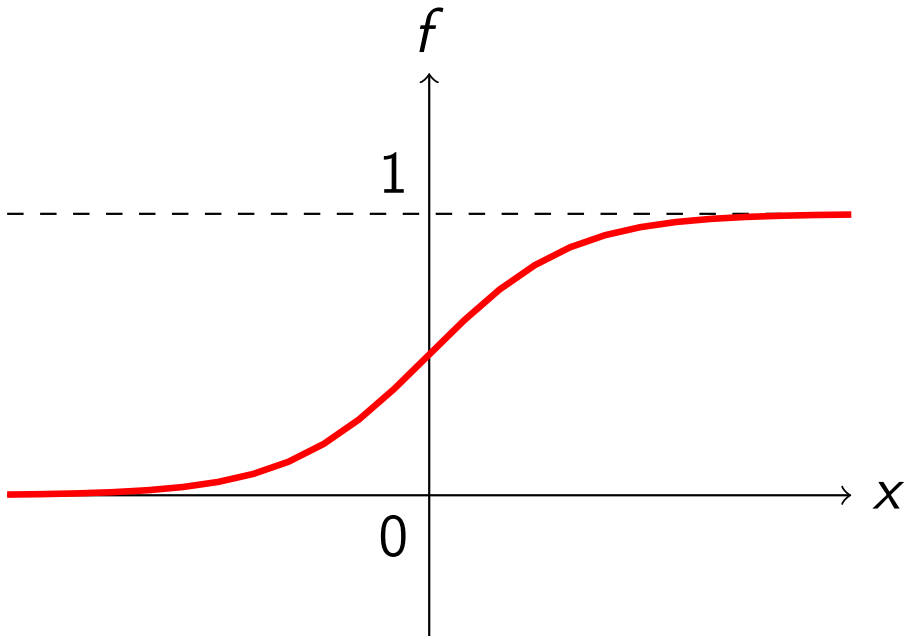


Схема устройства искусственного нейрона

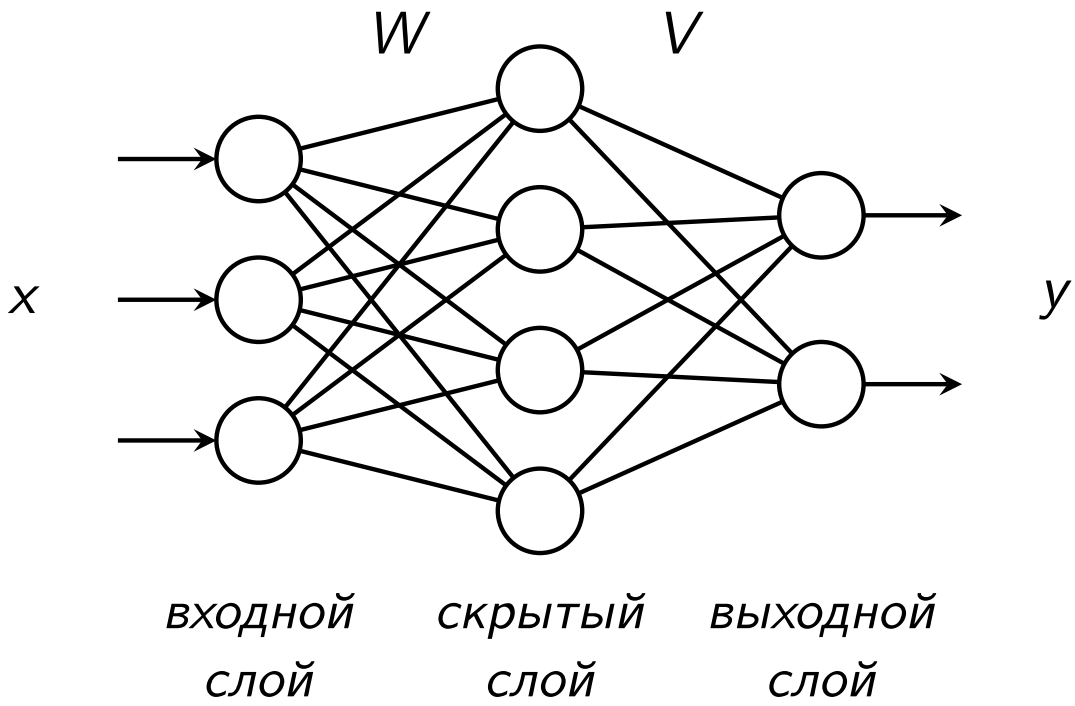


Активационная функция нейрона

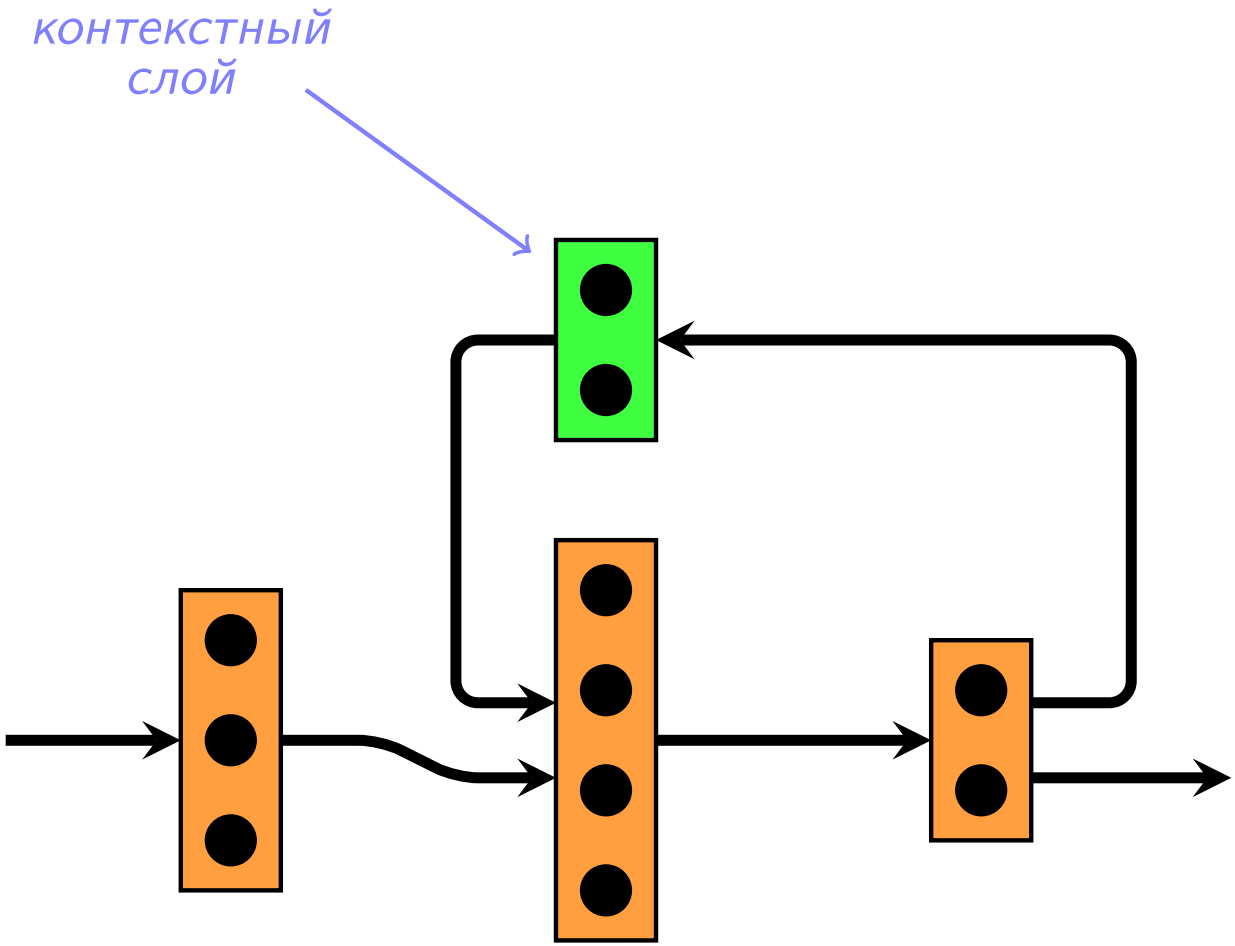


$$f(x) = \frac{1}{1 + e^{-kx}}.$$

Двуслойный перцептрон



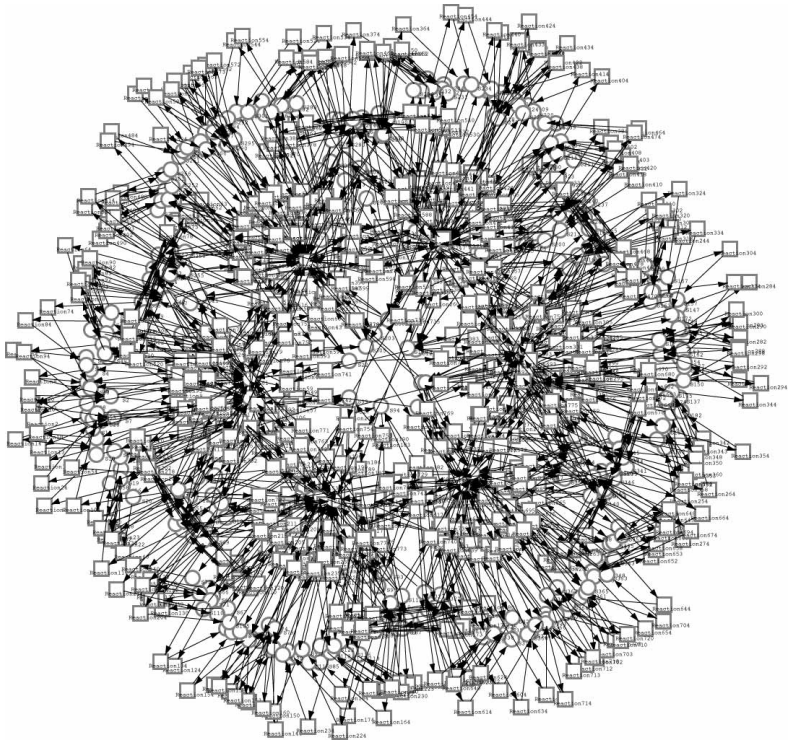
Реккурентная сеть Джордана



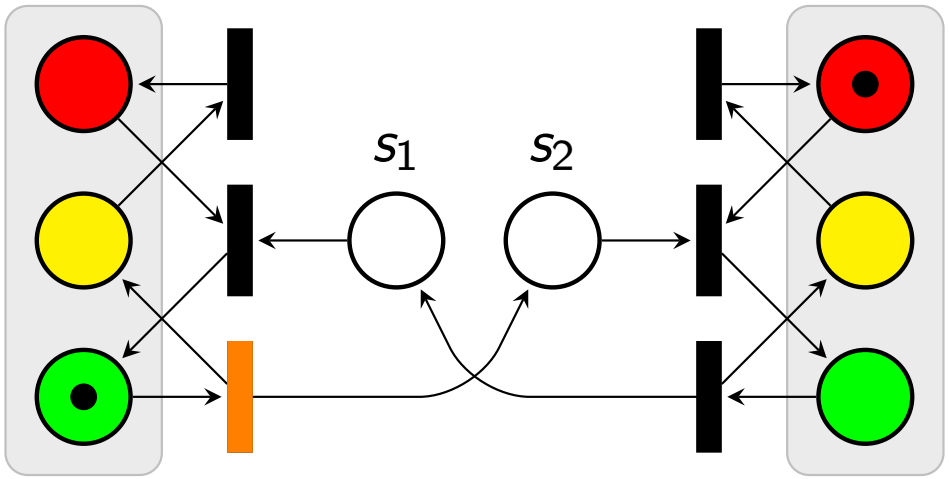
1962

Сети Петри

Карл Адам Петри

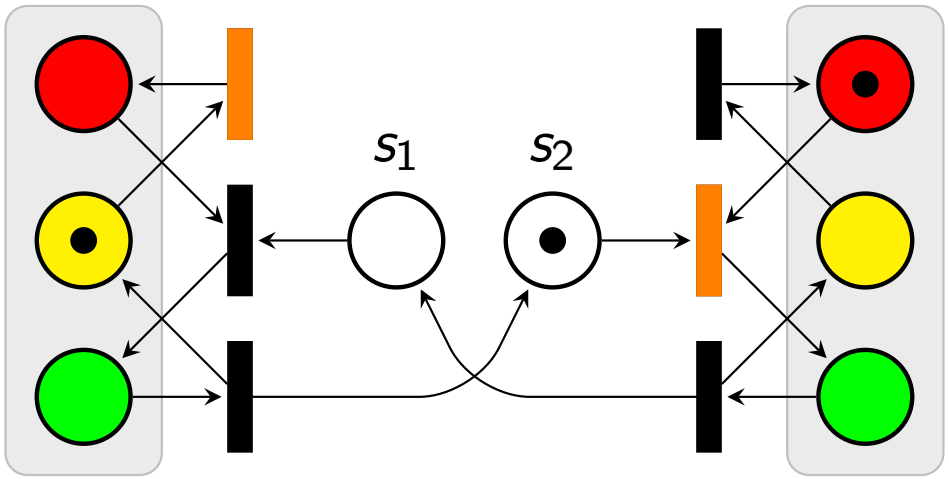


Модель двух светофоров



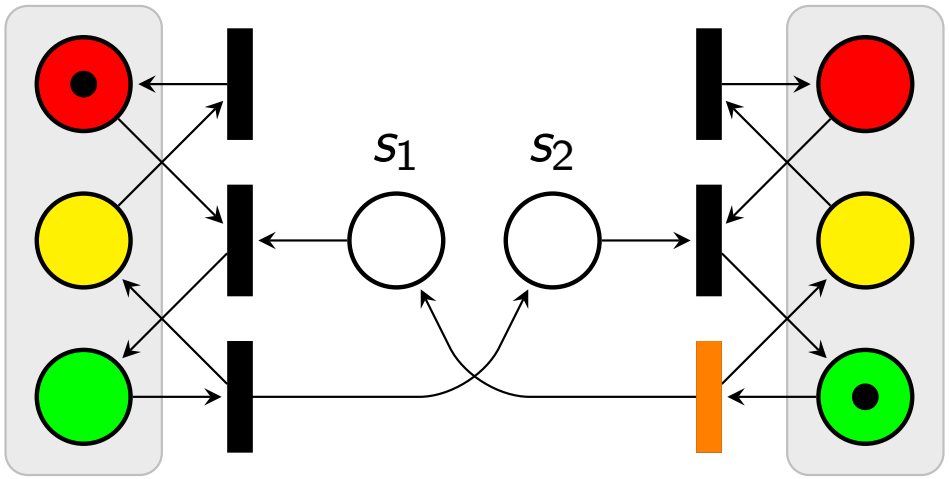
$t = 0$

Модель двух светофоров



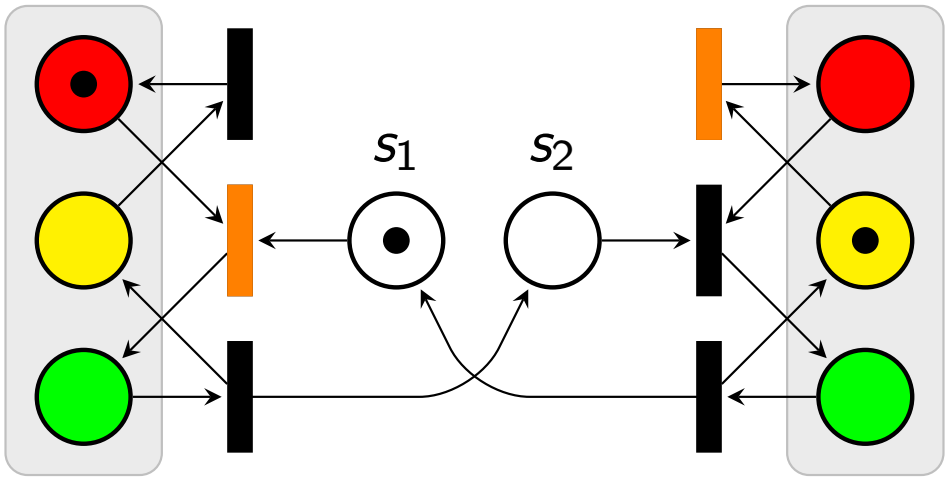
$$t = 1$$

Модель двух светофоров



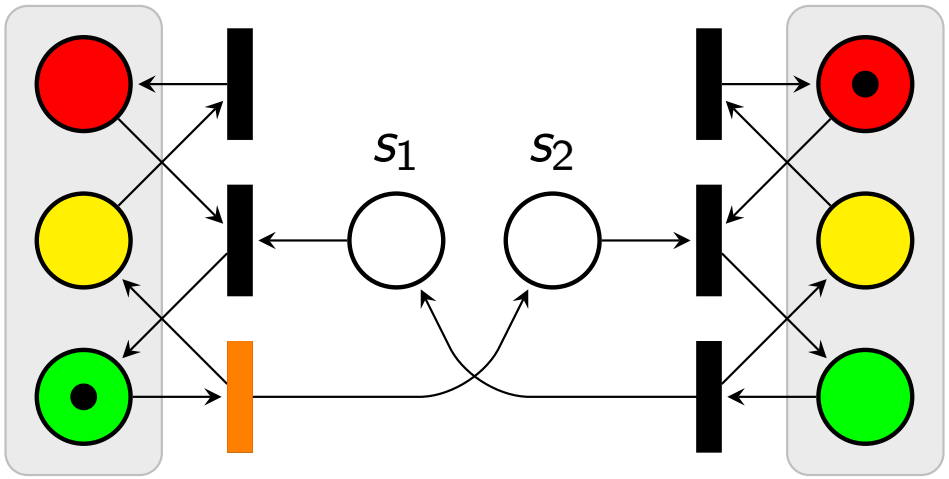
$t = 2$

Модель двух светофоров



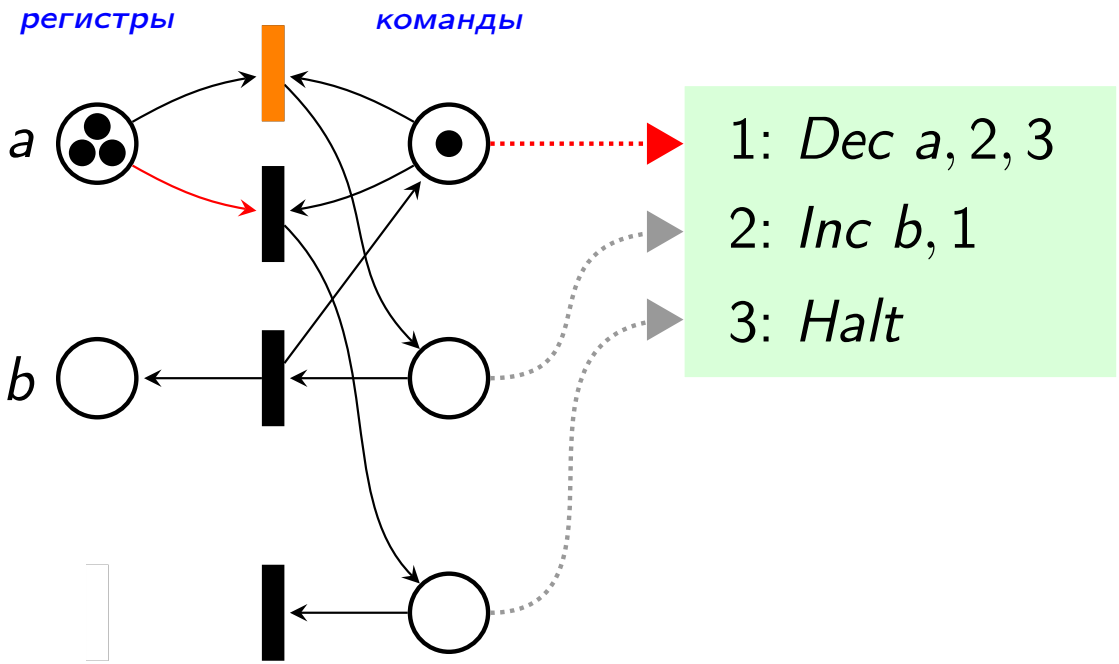
$t = 3$

Модель двух светофоров



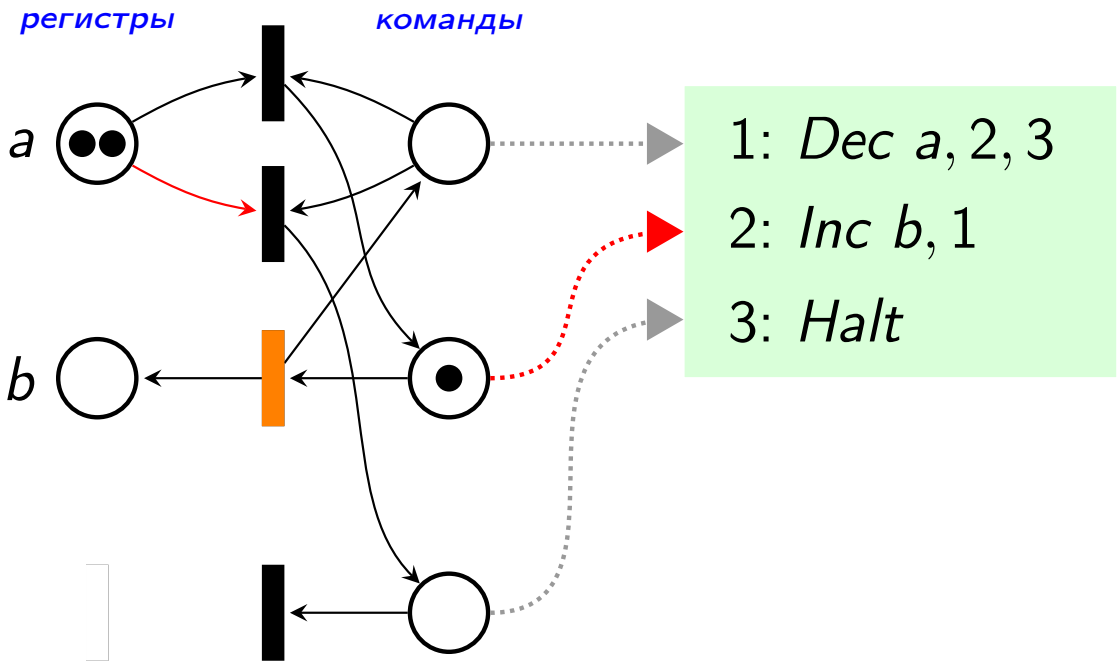
$t = 4$

Моделирование программы для машины Минского



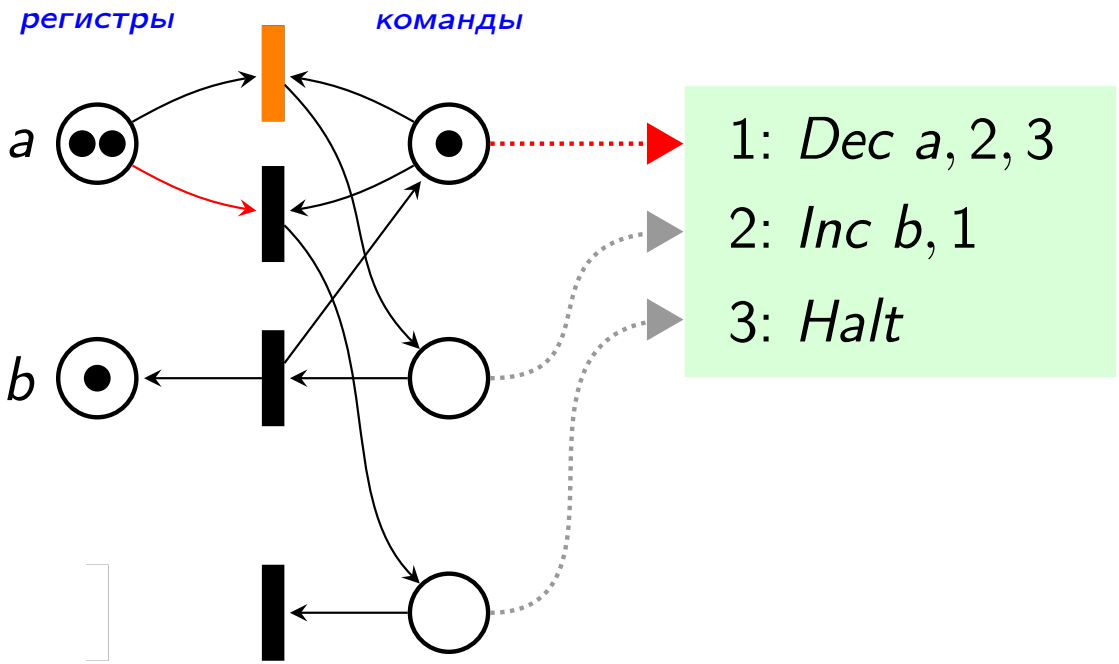
Программа копирования содержимого регистра a в регистр b.

Моделирование программы для машины Минского



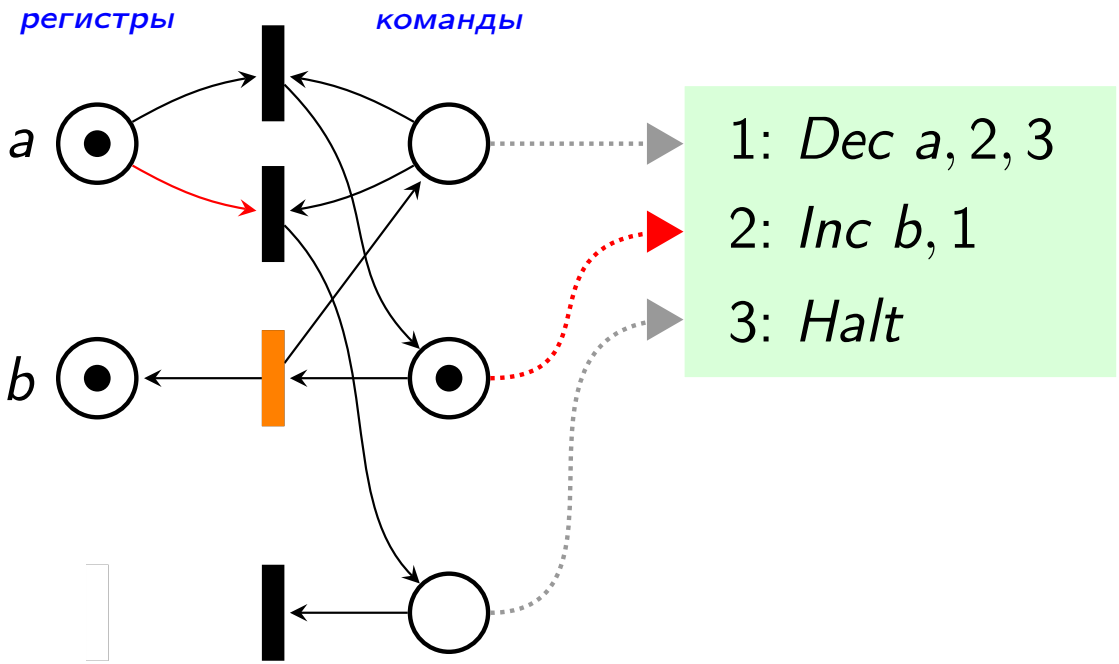
Программа копирования содержимого регистра a в регистр b.

Моделирование программы для машины Минского



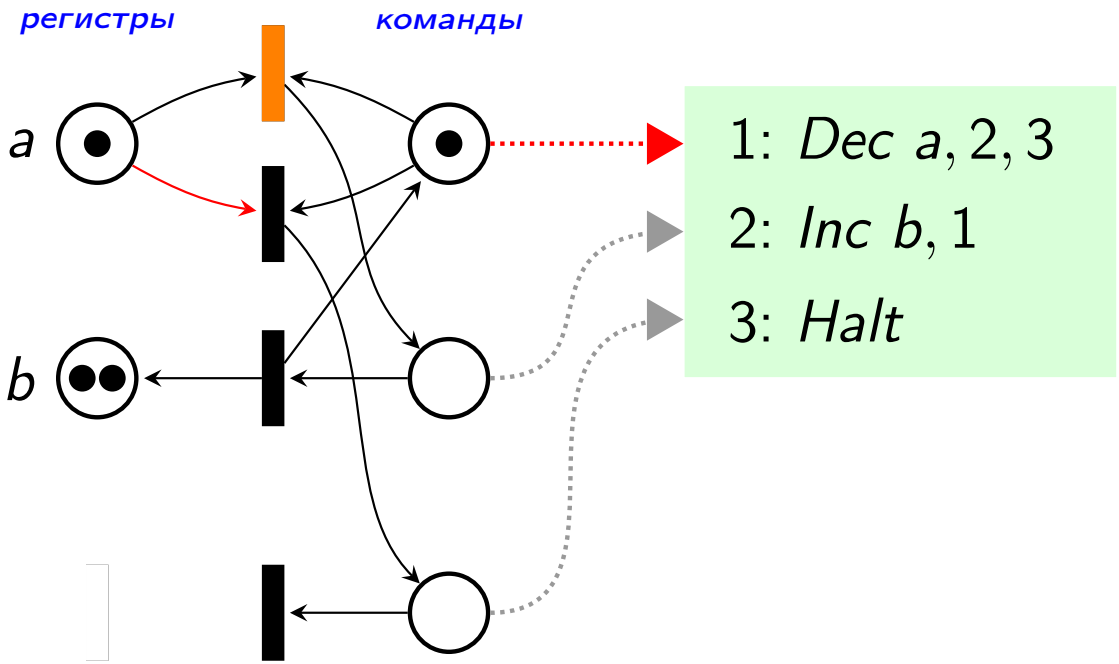
Программа копирования содержимого регистра a в регистр b.

Моделирование программы для машины Минского



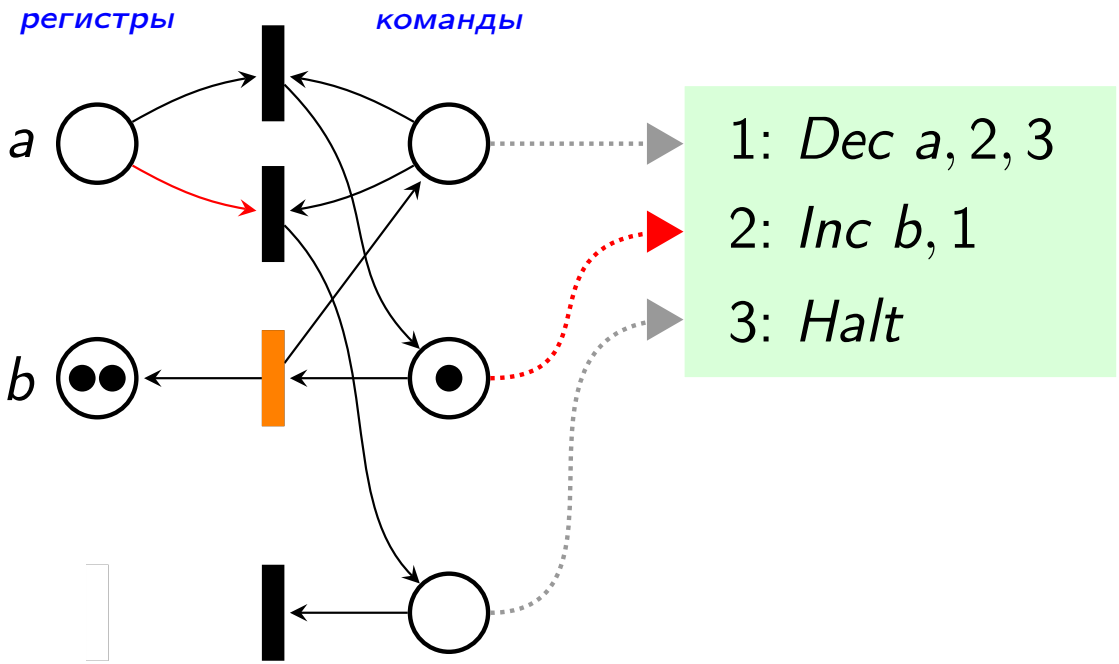
Программа копирования содержимого регистра a в регистр b.

Моделирование программы для машины Минского



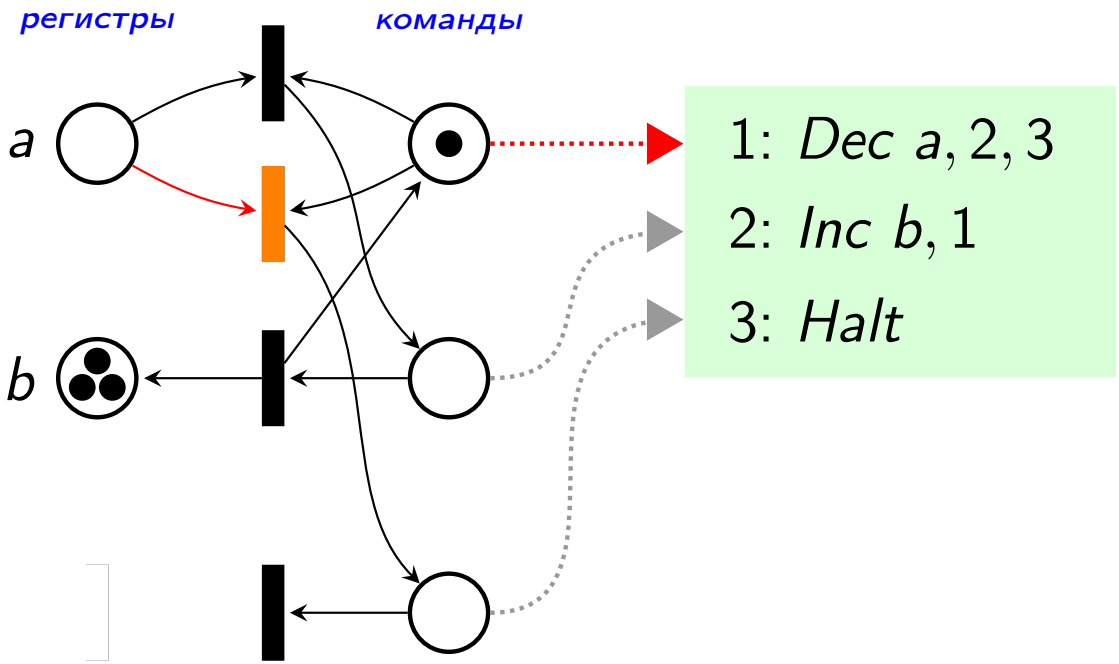
Программа копирования содержимого регистра a в регистр b.

Моделирование программы для машины Минского



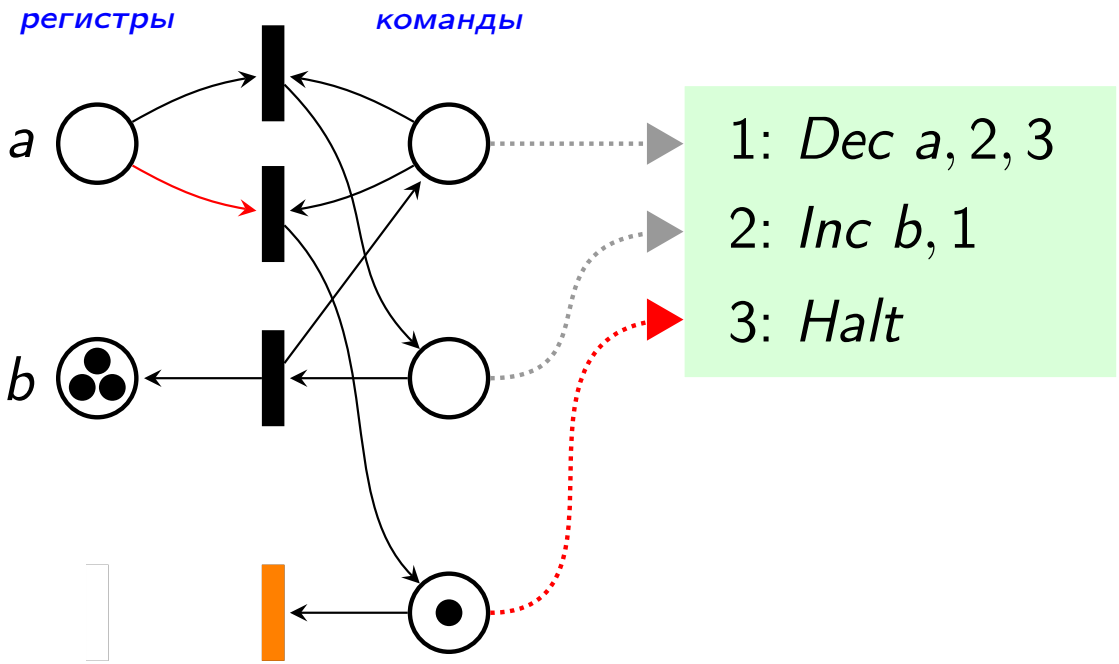
Программа копирования содержимого регистра a в регистр b.

Моделирование программы для машины Минского



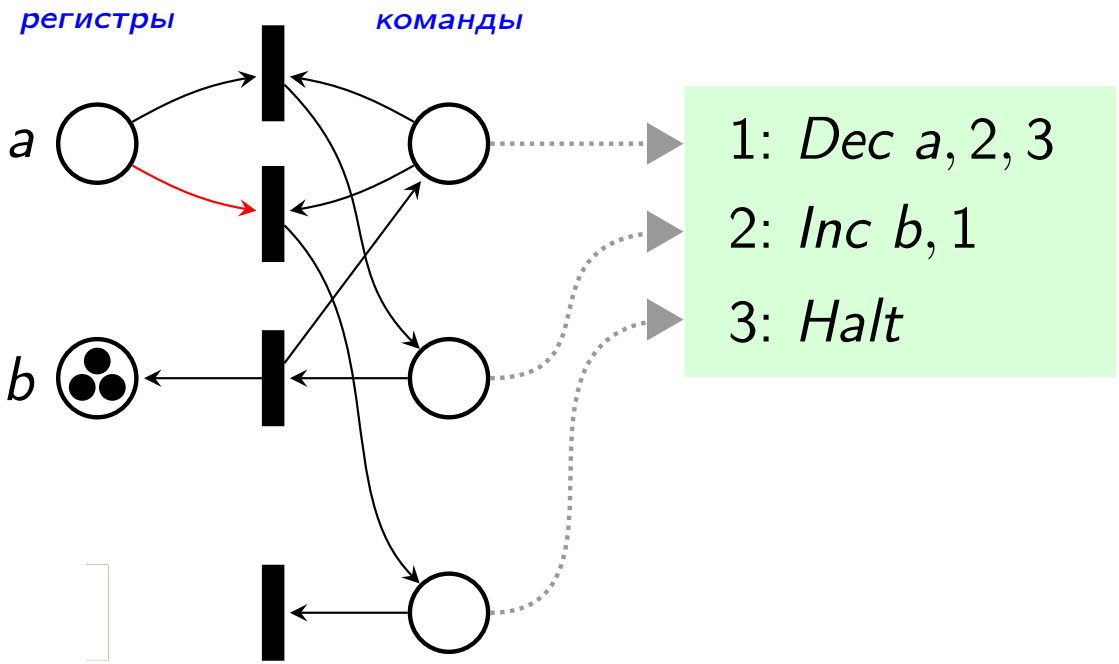
Программа копирования содержимого регистра a в регистр b.

Моделирование программы для машины Минского



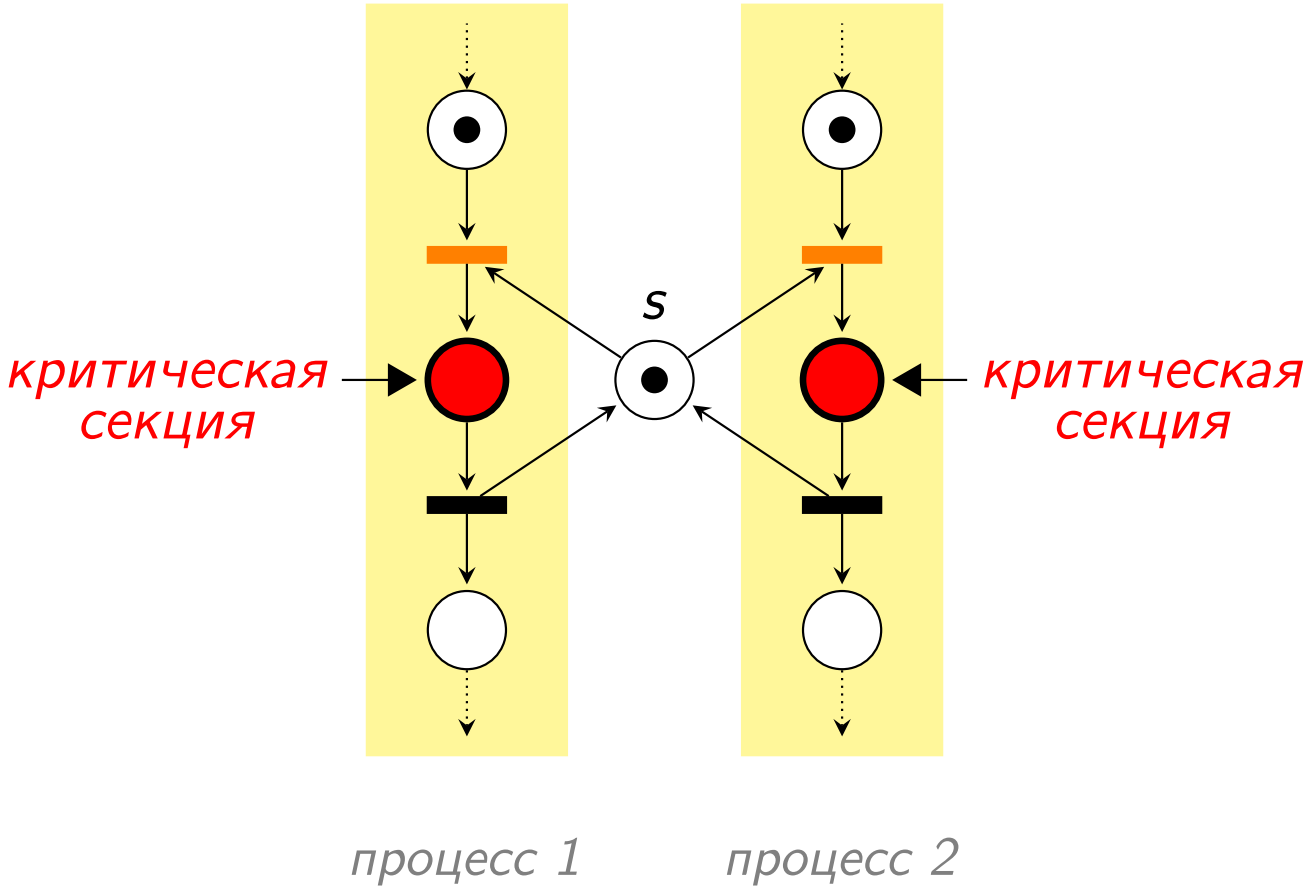
Программа копирования содержимого регистра a в регистр b.

Моделирование программы для машины Минского



Программа копирования содержимого регистра a в регистр b.

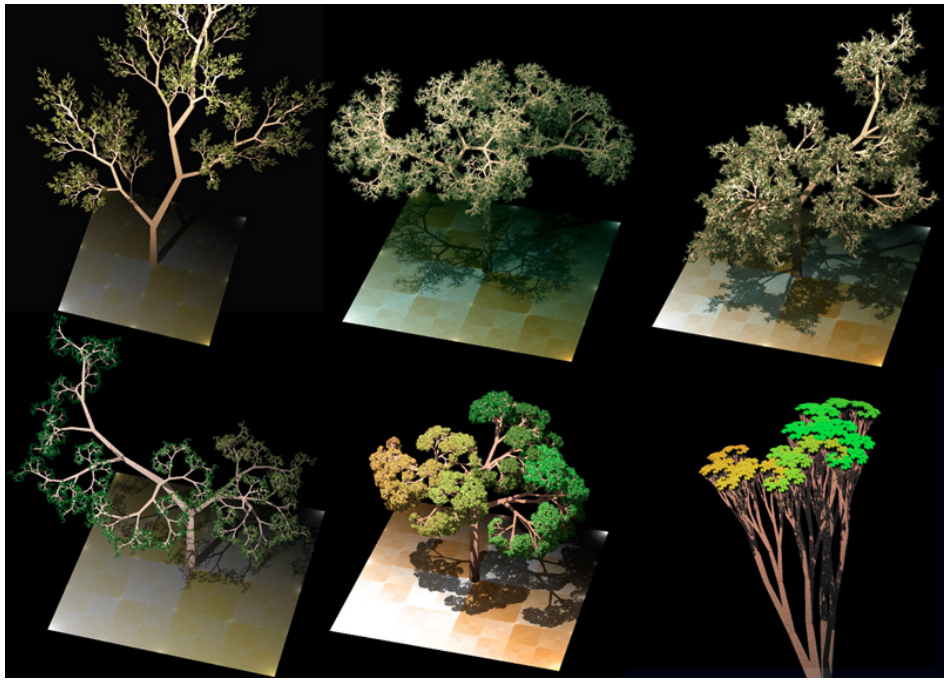
Моделирование параллельных процессов



1968

Системы Линденмайера

Аристид Линденмайер

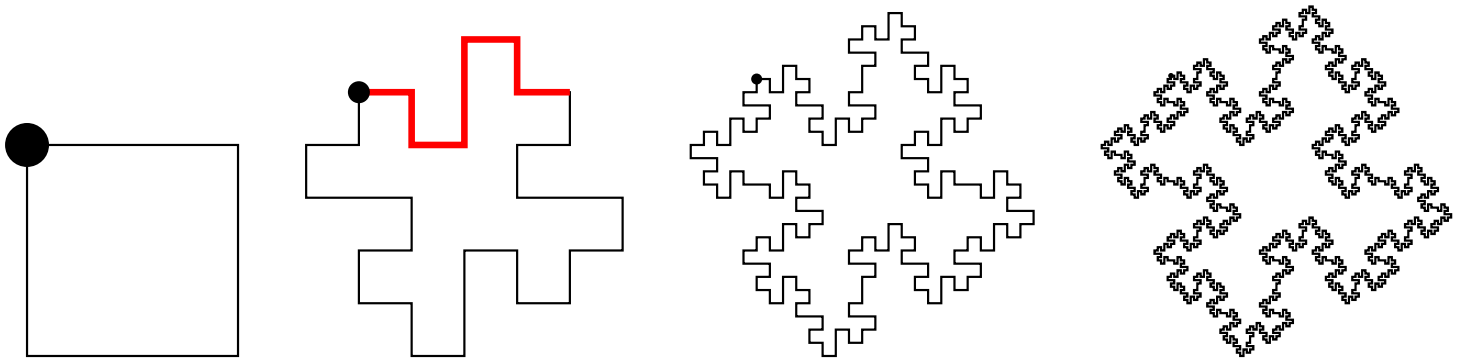


Генерация фракталов

- На рисунке приведены первые четыре шага эволюции L-системы с единственным правилом

$$R : F \to F - F + F + FF - F - F + F$$

для аксиомы $\omega_0 = F - F - F - F$.



а) $t = 0$ б) $t = 1$ в) $t = 2$ г) $t = 3$

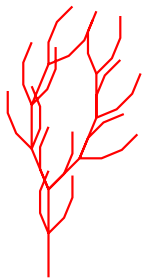
Ветвление

$$\begin{cases} R_1 : F \to FF - [-F + F + F] + \\ \qquad\qquad\qquad [+F - F - F] \\ \omega_0 = F, \delta = 22.5^\circ, n = 3. \end{cases}$$

Ветвление

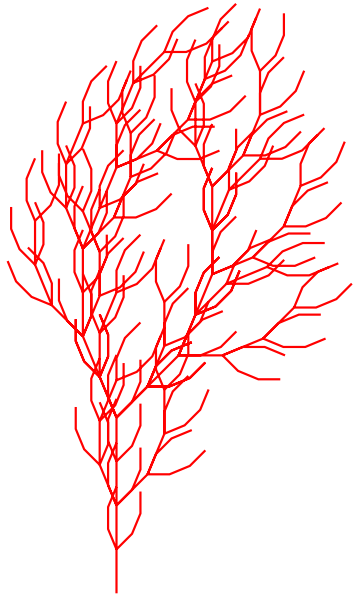
$$\begin{cases} R_1 : F \to FF - [-F + F + F] + \\ \qquad\qquad\qquad [+F - F - F] \\ \omega_0 = F, \delta = 22.5^\circ, n = 3. \end{cases}$$

Ветвление



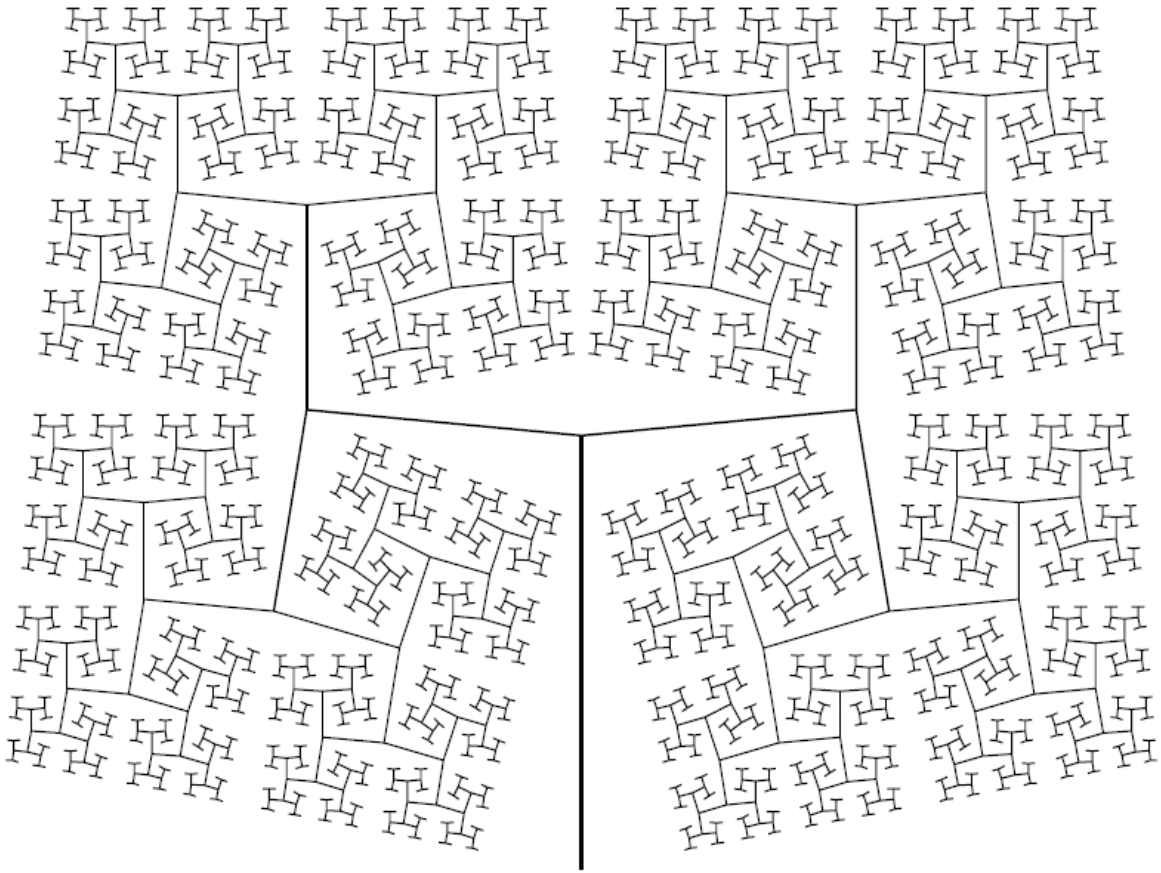
$$\begin{cases} R_1 : F \to FF - [-F + F + F] + \\ \qquad\qquad\qquad [+F - F - F] \\ \omega_0 = F, \delta = 22.5^\circ, n = 3. \end{cases}$$

Ветвление



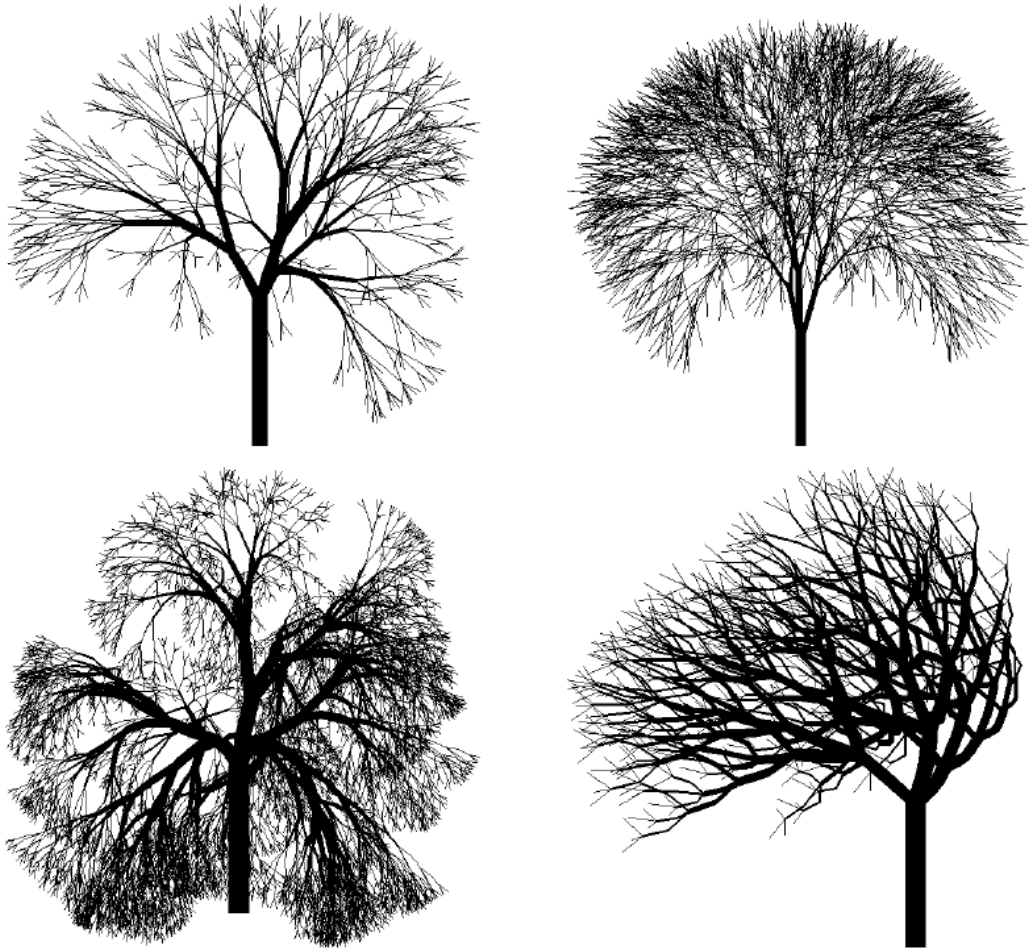
$$\begin{cases} R_1 : F \to FF - [-F + F + F] + \\ \qquad\qquad\qquad [+F - F - F] \\ \omega_0 = F, \delta = 22.5^\circ, n = 3. \end{cases}$$

Параметрические L–системы



$$\begin{aligned} \omega &: \quad A(1) \\ p_1 &: \quad A(s) \to F(s)[+A(s/R)][-A(s/R)] \end{aligned}$$

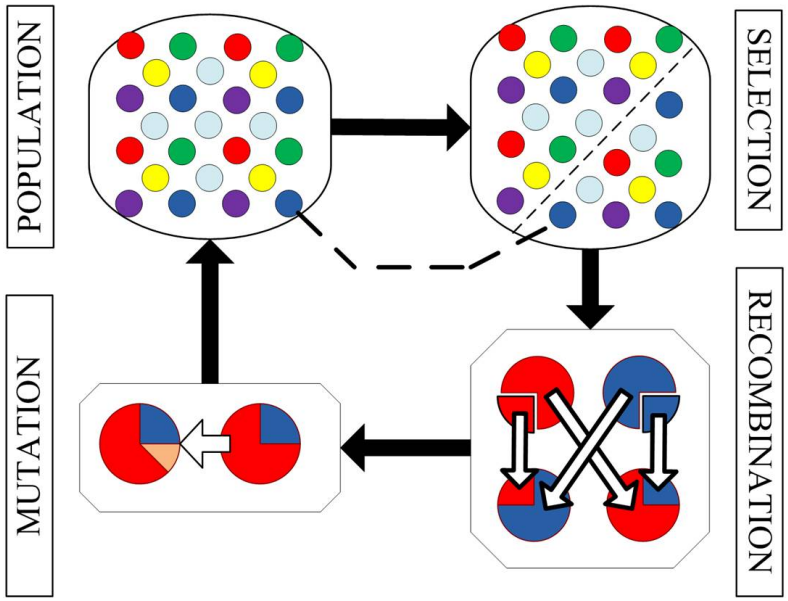
Моделирование процессов формообразования



1975

Генетические алгоритмы

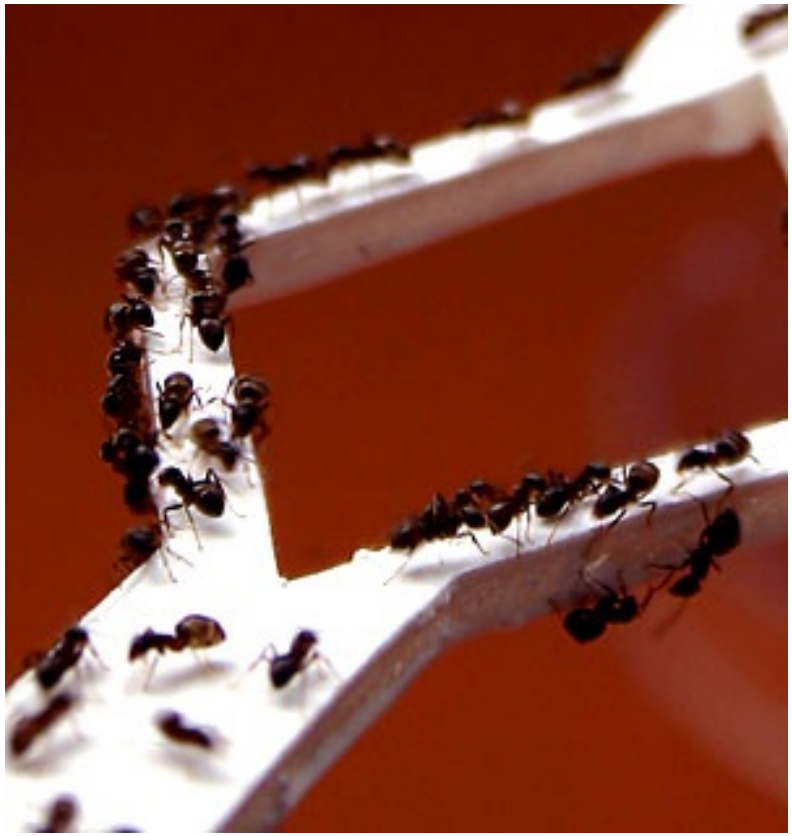
Джон Холланд



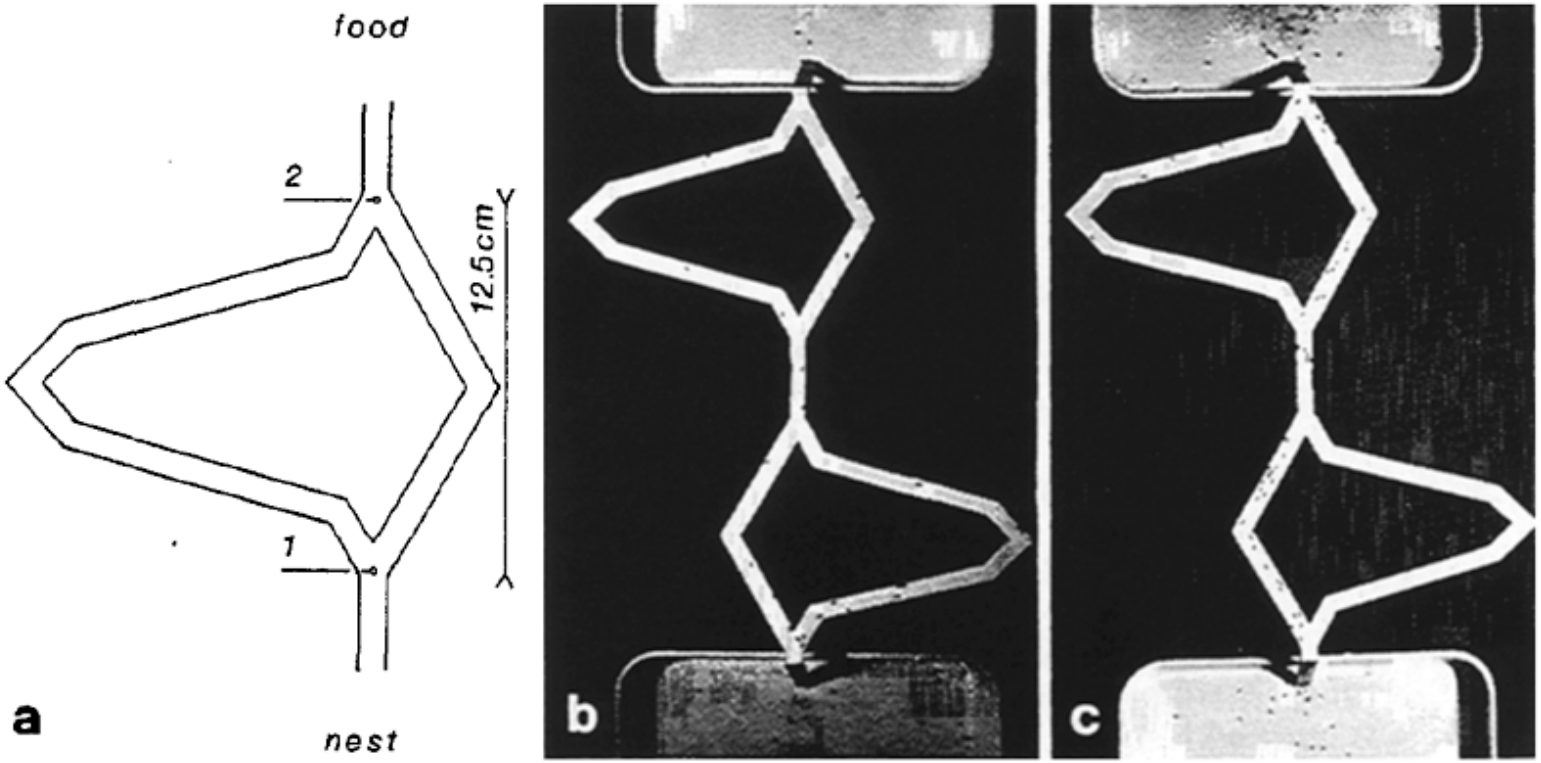
1991

Муравьиные алгоритмы

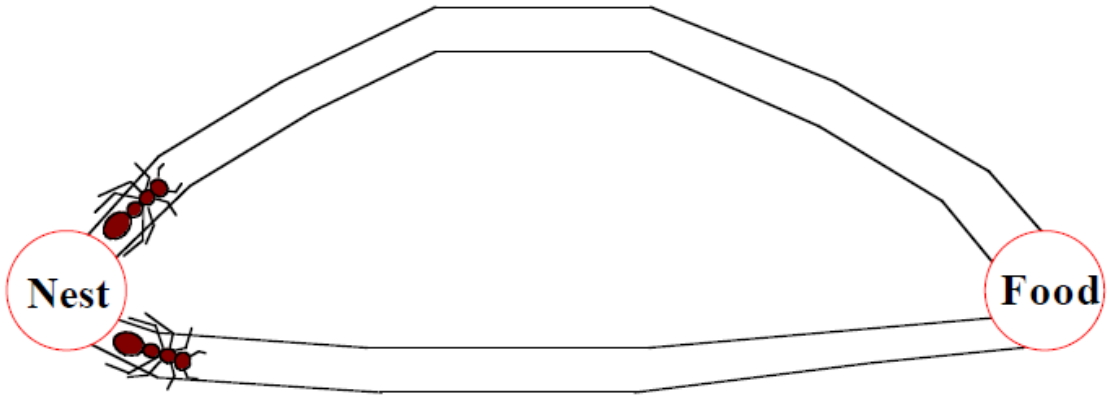
Марко Дориго



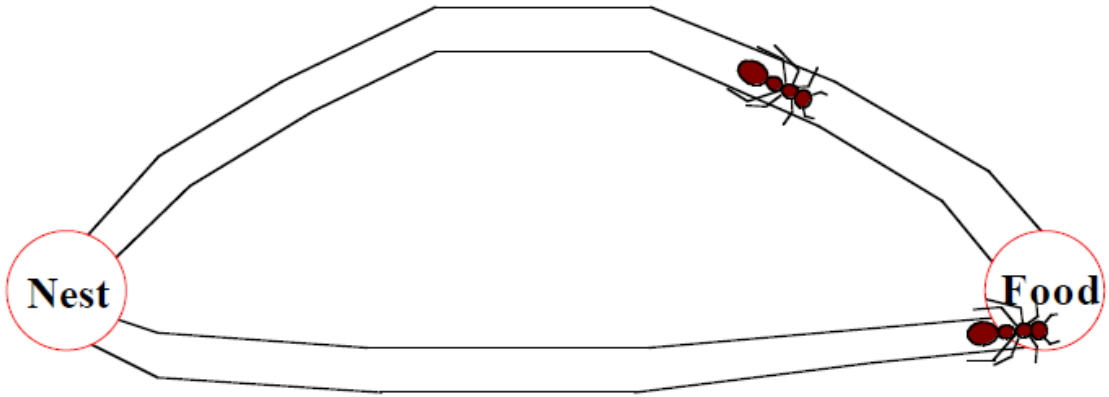
Эксперимент с двумя мостами



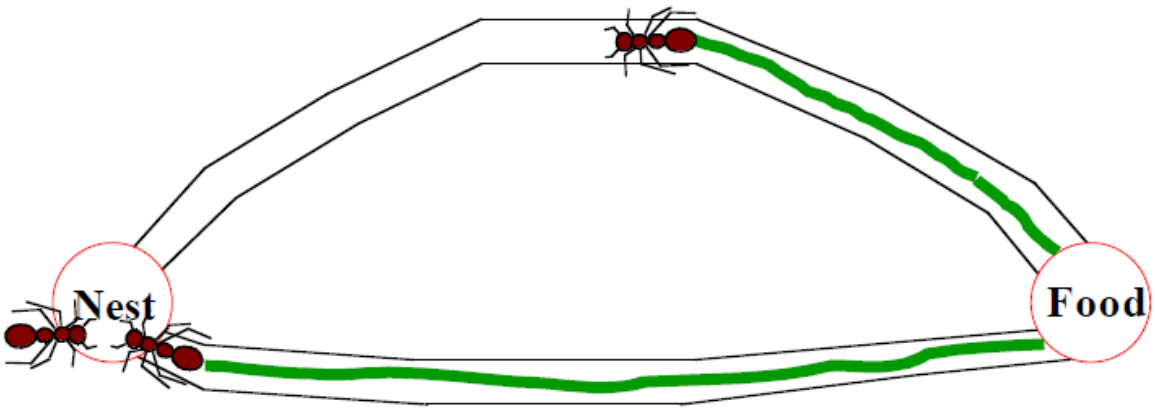
Эксперимент с двумя мостами



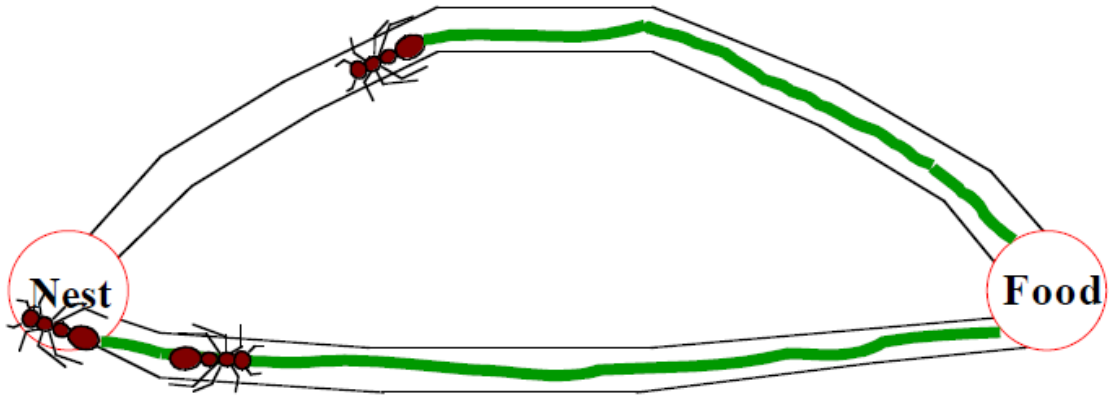
Эксперимент с двумя мостами



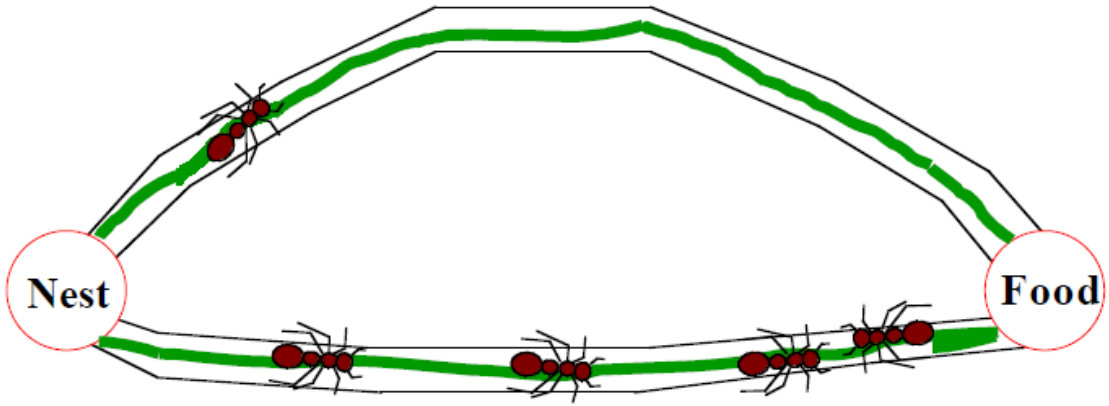
Эксперимент с двумя мостами



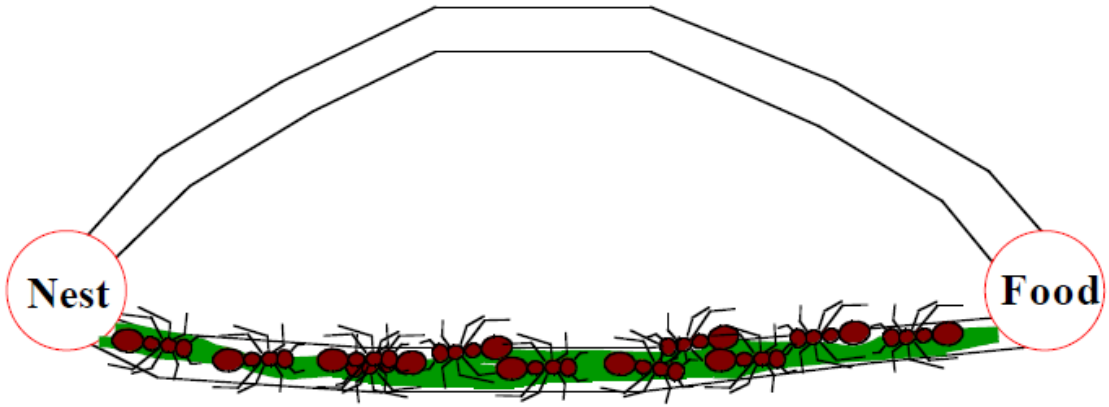
Эксперимент с двумя мостами



Эксперимент с двумя мостами



Эксперимент с двумя мостами

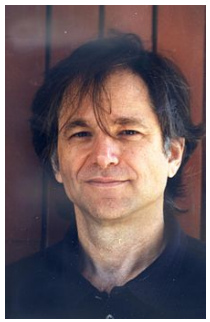


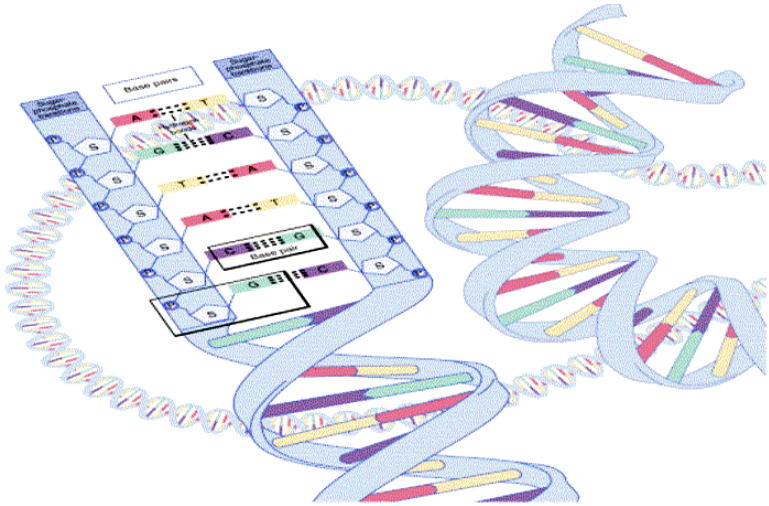
Мельница смерти

1994

ДНК–вычисления

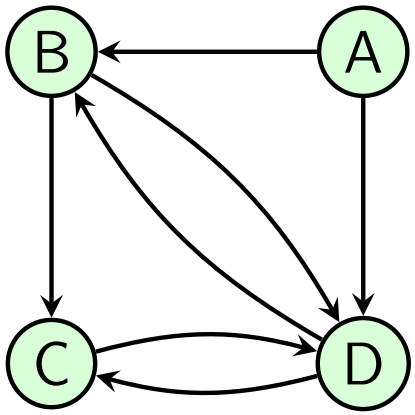
Леонард Адлеман





Пример работы алгоритма Адлемана

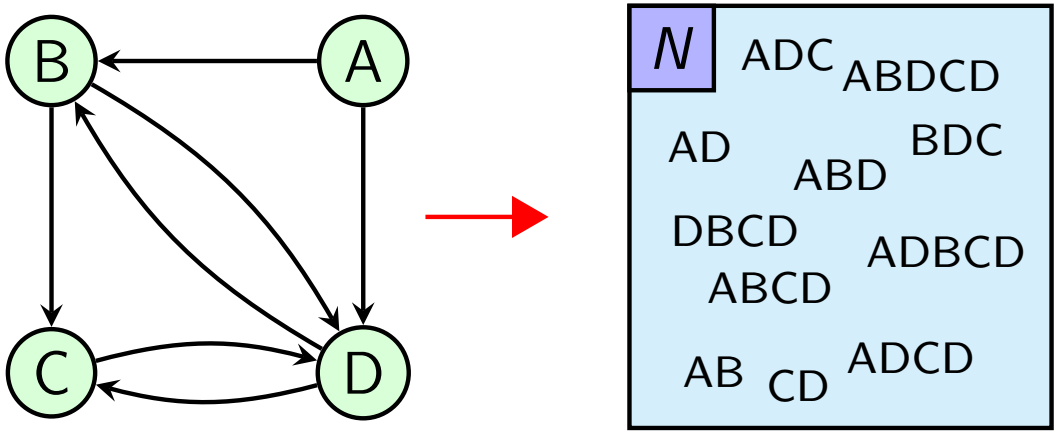
Требуется определить, имеется ли гамильтонов путь из вершины A в вершину D?



Генерация случайных путей в графе

Пример работы алгоритма Адлемана

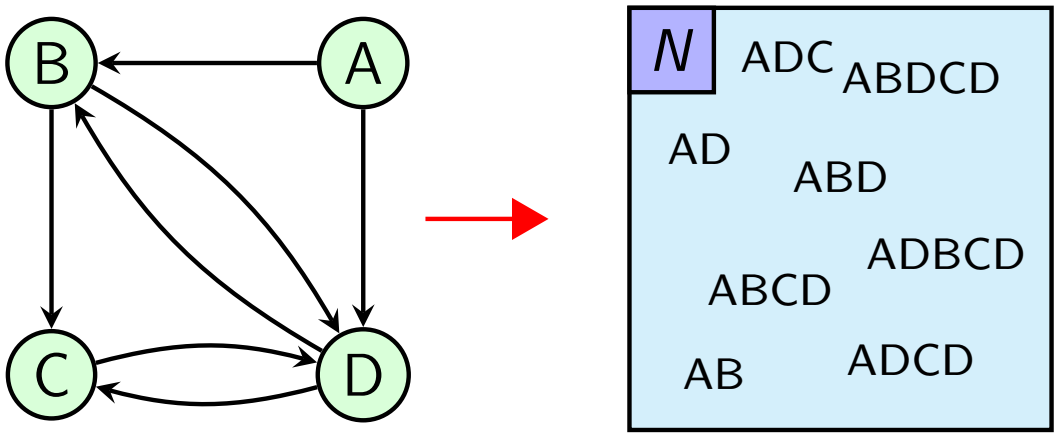
Требуется определить, имеется ли гамильтонов путь из вершины A в вершину D?



Отбрасывание путей, не начинающихся в A

Пример работы алгоритма Адлемана

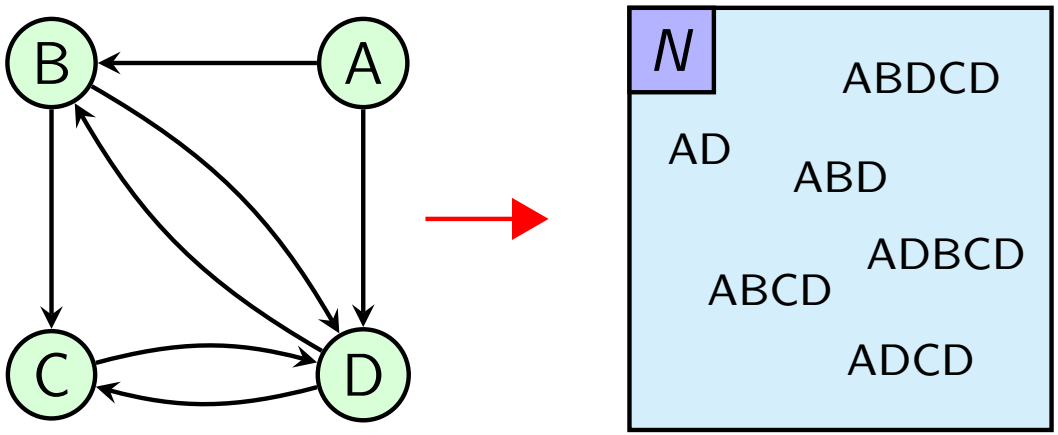
Требуется определить, имеется ли гамильтонов путь из вершины A в вершину D?



Отбрасывание путей, не заканчивающихся в D

Пример работы алгоритма Адлемана

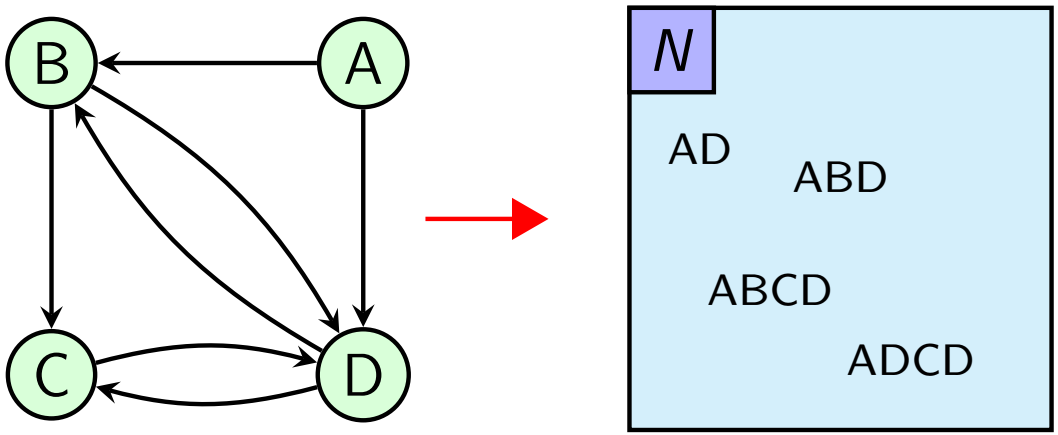
Требуется определить, имеется ли гамильтонов путь из вершины A в вершину D?



Отбрасывание путей длины большей 4

Пример работы алгоритма Адлемана

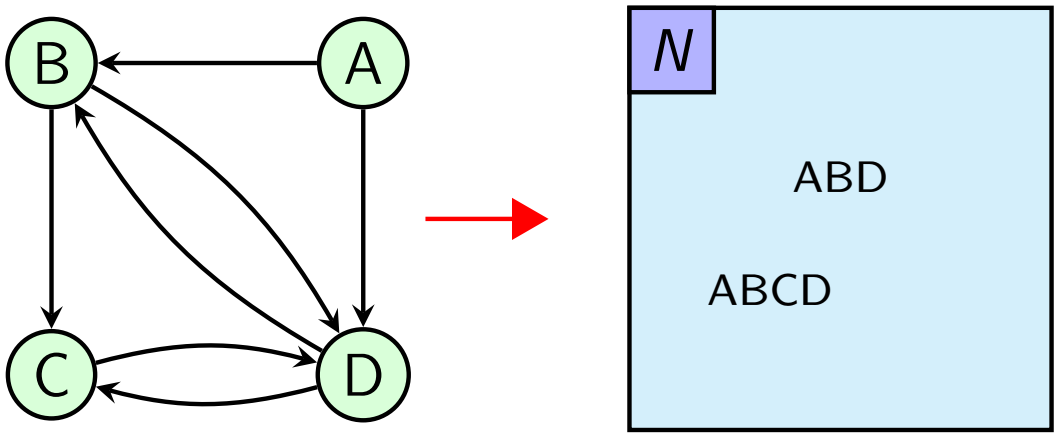
Требуется определить, имеется ли гамильтонов путь из вершины A в вершину D?



Отбрасывание путей, не содержащих вершины B

Пример работы алгоритма Адлемана

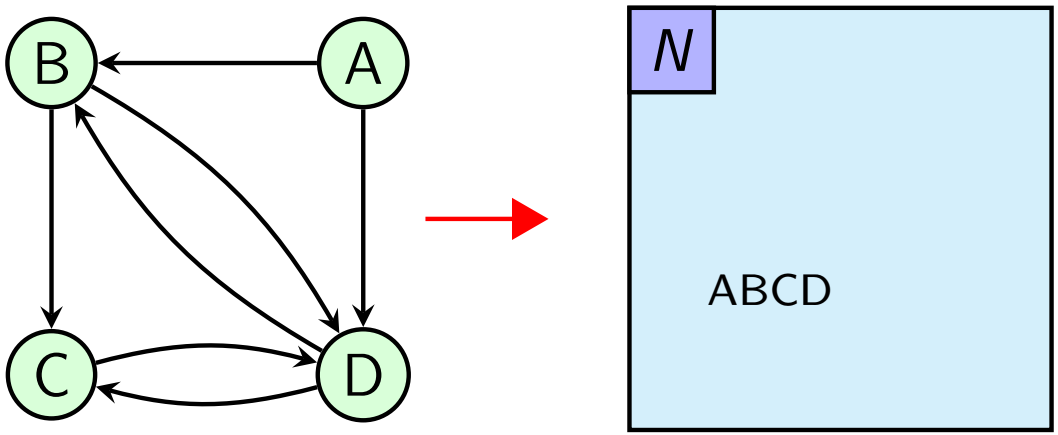
Требуется определить, имеется ли гамильтонов путь из вершины A в вершину D?



Отбрасывание путей, не содержащих вершины C

Пример работы алгоритма Адлемана

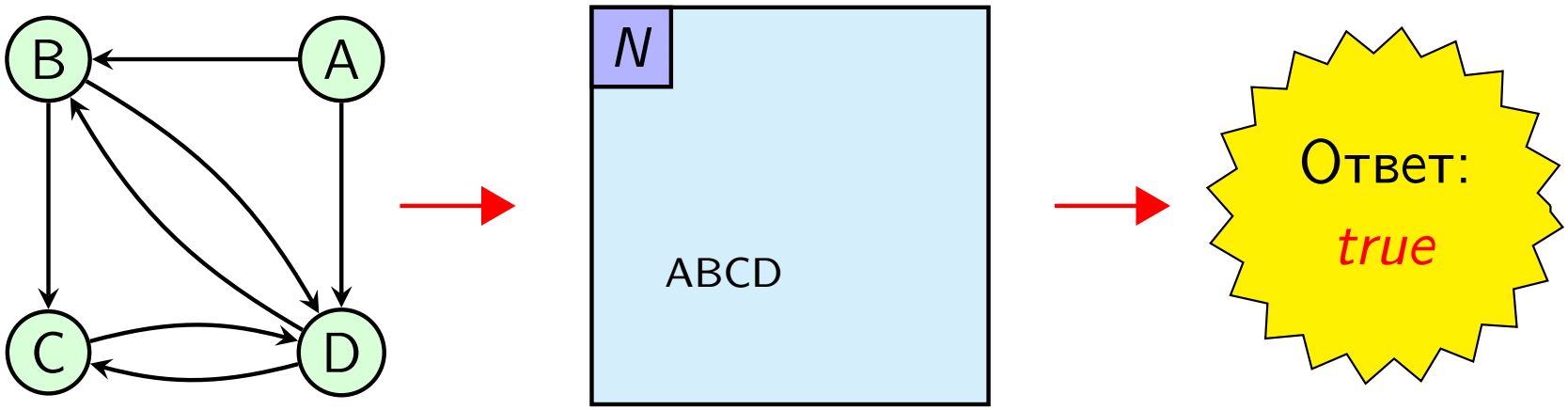
Требуется определить, имеется ли гамильтонов путь из вершины A в вершину D?



Остались ли в пробирке молекулы?

Пример работы алгоритма Адлемана

Требуется определить, имеется ли гамильтонов путь из вершины A в вершину D?



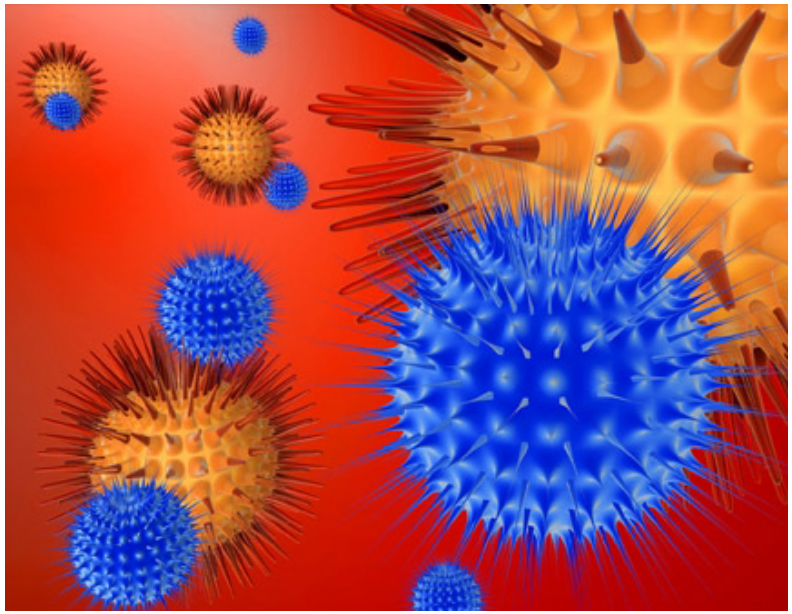
В графе есть гамильтонов путь из A в D

1994

Искусственные иммунные системы

Стефани Форрест

Джефри Кипхарт



1995

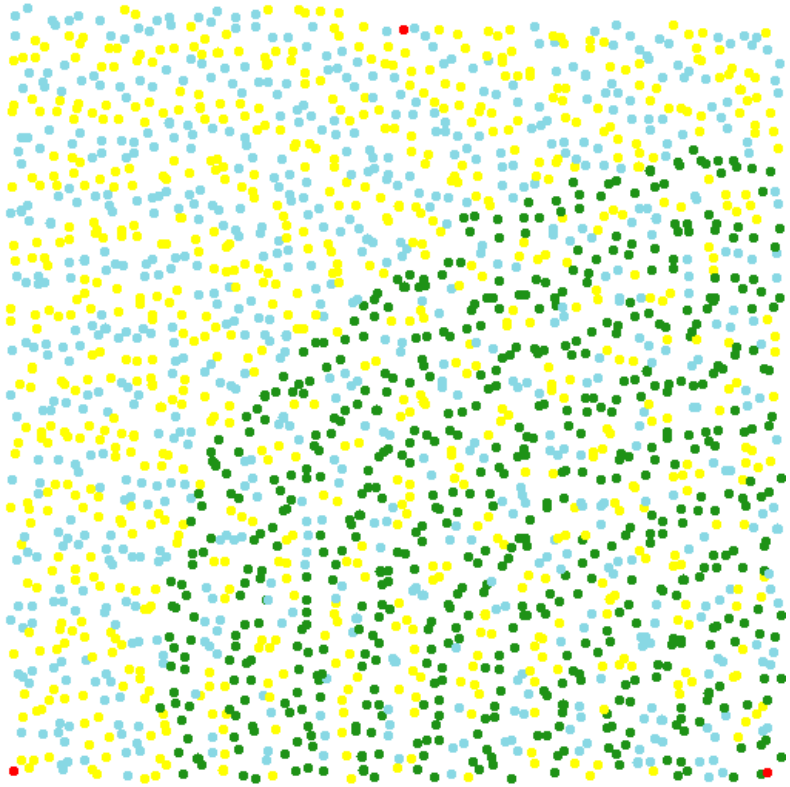
Метод роя частиц

Джеймс Кеннеди

1996

Аморфные вычисления

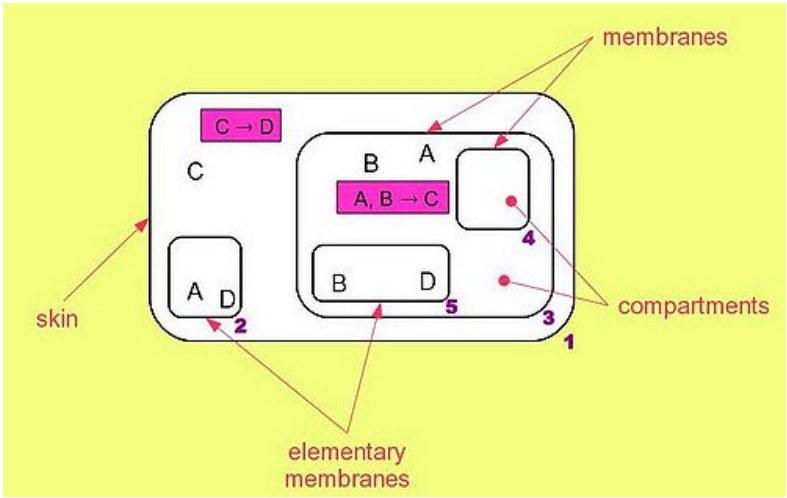
Хал Абельсон



1998

Мембранные системы

Георгий Паун



2002

Бактериальные алгоритмы

Кевин Пассино

2005

Пчелиные алгоритмы

Дук Труонг Фам

Приложения естественных вычислений

Область применения естественных моделей в настоящее время чрезвычайно широка. Краткий, наиболее общий список приложений естественных вычислений выглядит следующим образом:

- моделирование природных процессов и систем;
- исследование вычислительных возможностей естественных систем;
- решение вычислительных задач;
- разработка новых технологий.

Структура естественных вычислительных моделей

- Практически все естественные вычислительные модели, если рассматривать их на верхнем уровне абстракции, имеют схожую структуру — каждая такая модель состоит из большого числа (обычно однотипных) относительно простых элементов (объектов). Эти объекты обладают внутренним состоянием и способны менять это состояние путем взаимодействия друг с другом или с окружающей их средой.
- Выбор подходящих правил коммуникации позволяет настроить всю систему на решение некоторой вычислительной задачи — возникает некое системное свойство, которого нет ни у одного отдельного взятого объекта этой системы.

Массивно–параллельные вычисления

- Важным пунктом функционирования естественных систем является то, что все входящие в них объекты функционируют (взаимодействуют, меняют свои состояния) одновременно, т. е. *параллельно*.
- Следовательно, можно сказать, что в естественных вычислительных моделях производится параллельная (распределенная) обработка информации.
- Это делает такого рода модели весьма интересным объектом для реализации на современных суперкомпьютерах — *массивно–параллельных вычислительных системах*.

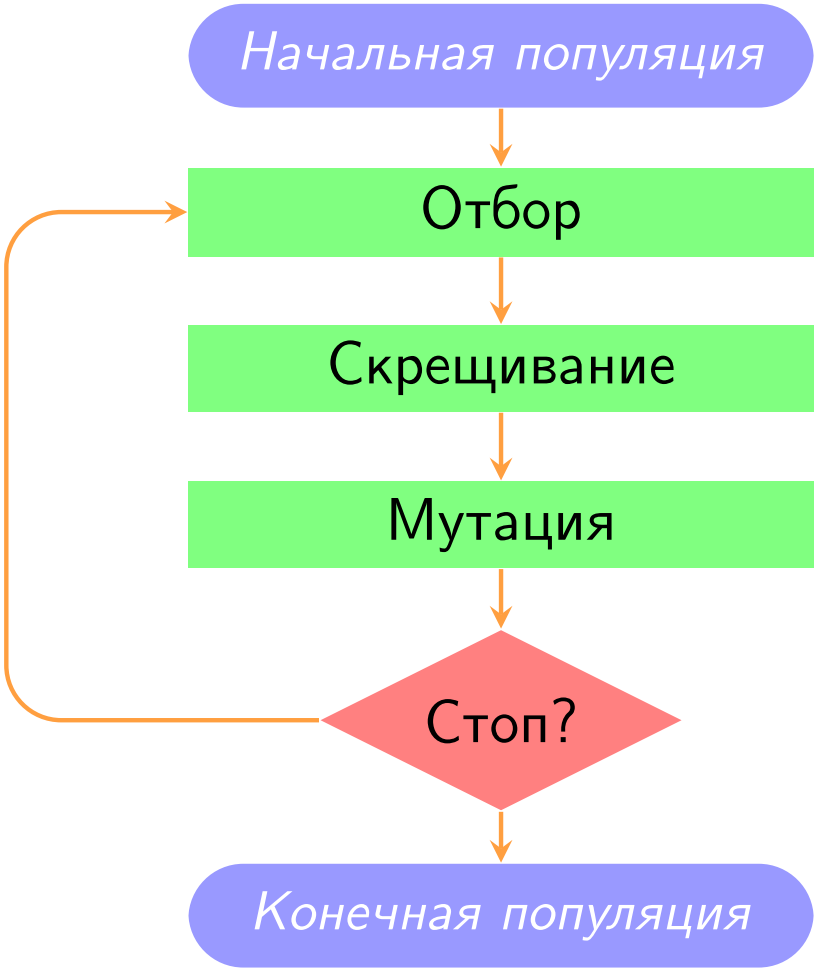
Параллельные алгоритмические модели

- Имеется еще одна точка соприкосновения естественных моделей и параллельных вычислений: многие из таких моделей (клеточные автоматы, нейронные сети, мембранные системы) могут рассматриваться в качестве параллельных алгоритмических моделей. Возможно, в будущем эти модели будут положены в основу построения новых архитектур вычислительных устройств.
- В настоящее время имеется ряд примеров аппаратной реализации некоторых естественных моделей: нейронные сети (например, первая искусственная нейронная сеть — перцептрон Розенблатта был реализован аппаратно); клеточные автоматы (в MIT были разработаны несколько моделей машин клеточных автоматов CAM), ДНК–вычисления (опыт Адлемана).

Параллельные генетические алгоритмы

- *Генетические алгоритмы* являются мощным методом решения сложных оптимизационных задач. Работа генетических алгоритмов (ГА) основана на моделировании ими эволюционного процесса с использованием таких генетических механизмов, как отбор, скрещивание и мутация.
- Генетическим алгоритмом рассматривается набор из сразу нескольких возможных решений задачи, такой набор называется *популяцией*.
- Решения «борются» за существование, причем выживают в этом процессе *отбора* наиболее приспособленные решения.
- В процессе своего развития члены популяции обмениваются информацией, используя механизм *скрещивания*.
- *Мутации* в ГА применяются, как правило, для предотвращения слишком быстрой их сходимости к локальным экстремумам решаемой задачи.

Схема работы генетического алгоритма



Задача непрерывной оптимизации

- Построим ГА, решающий задачу минимизации n-мерной функций, область определения которой является непрерывное n-мерное пространство R^n.
- В качестве примера возьмем следующую функцию

$$f(x) = \sum_{i=1}^{n} |x_i| \to \min$$

Это *унимодальная* функция, имеющая единственный минимум в начале координат $x = 0$.

- Решение будем искать в области $x_i \in [-100, 100]$ — это *начальная* область для поиска.

Кодирование решений

Каждое решение (отдельная особь популяции) будет естественным образом представляться вещественным массивом длины n, который мы обернем в простую структуру:

```
struct genom
{
    double* data; \\ геном
    int len; \\ длина генома (размерность задачи)
    genom() { data = 0; }
    ~genom() { if( data ) delete[] data; }
};
```

Создание нового генома

Создаем массив data и заполняем его случайными числами в заданном диапазоне:

```
double frand() // случайное число в интервале [0,1)
{
    return double(rand())/RAND_MAX;
}

void create(genom& A, int l)
{
    A.data = new double[l];
    A.len = l;
    for( int i=0; i<l; i++ )
        A.data[i] = -100+200*frand(); // случайный выбор гена
}
```

Функция качества

Определяем целевую функцию:

```
double eval(genom& A)
{
    double sum = 0;
    for( int i=0; i<A.len; i++ )
        sum += fabs(A.data[i]);
    return sum;
}
```

Отбор

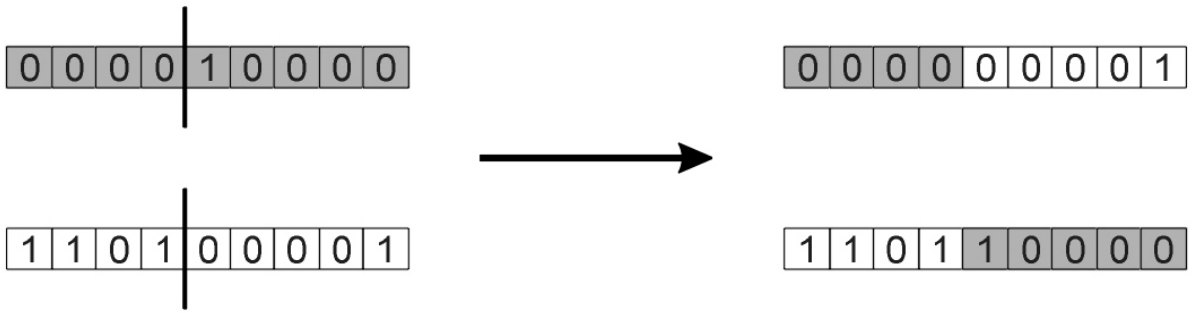
Для отбора будем использовать *турнирную* схему:

```
void copy(genom& A, genom& B) // победитель создает свою копию
{
    for( int i=0; i<A.len; i++ )
        B.data[i] = A.data[i];
}
void select(genom& A, genom& B) // отбор
{
    const double pwin = 0.75; // вероятность выживания лучшего в паре
    double fa = eval(A);
    double fb = eval(B);
    double p = frand();
    if( fa<fb && p<pwin || fa>fb && p>pwin )
        copy(A,B); // победил A
    else
        copy(B,A); // победил B
}
```

Скрещивание

Одноточечная схема скрещивания:

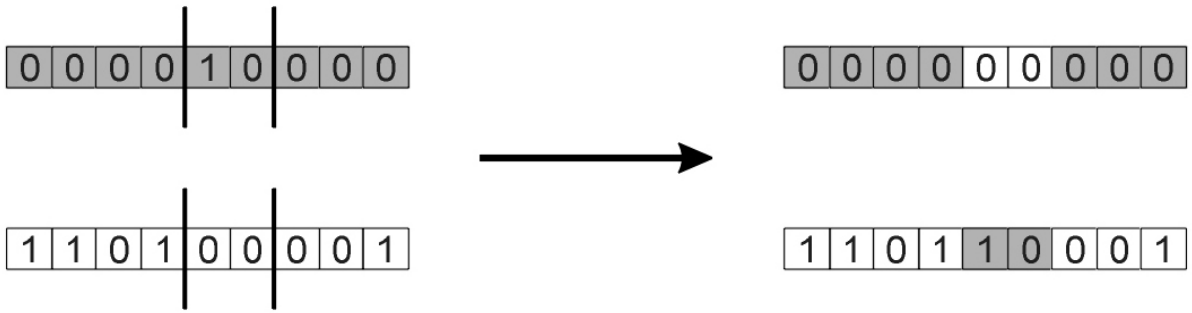
```
void crossover(genom& A, genom& B)
{
    int n = A.len;
    int k = rand()%(n-1); // точка скрещивания
    for( int i=k+1; i<n; i++ )
       swap(A.data[i],B.data[i]);
}
```



Скрещивание

Двухточечная схема скрещивания:

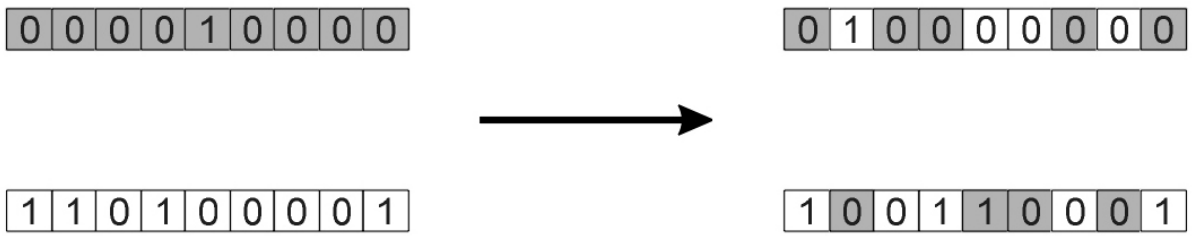
```
void crossover(genom& A, genom& B)
{
    int n = A.len;
    int k = rand()%(n-1); // первая точка скрещивания
    int l = k+rand()%(n-k-1); // вторая точка скрещивания
    for( int i=k+1; i<l; i++ )
        swap(A.data[i],B.data[i]);
}
```



Скрещивание

Равномерная схема скрещивания:

```
void crossover(genom& A, genom& B)
{
    const double pswap = 0.25; \\ вероятность обмена двумя генами
    int n = A.len;
    for( int i=0; i<n; i++ )
        if( frand()<pswap )
            swap(A.data[i],B.data[i]);
}
```



Мутация

Случайное возмущение решения:

```
void mutate(genom& A)
{
    const double pmutate = 0.1; // вероятность мутации гена
    double dx = 0.1; // величина мутации
    for( int i=0; i<A.len; i++ )
        if( frand()<pmutate )
            A.data[i] += dx*(2*frand()-1);
}
```

Перемешивание популяции

Популяцию будем представлять массивом, перемешивание применяется в процессе эволюции популяции:

```
// P --- популяция (массив геномов), size --- размер популяции
void shuffle(genom* P, int size)
{
    for( int i=0; i<size; i++ )
        swap(P[i].data,P[rand()%size].data); // обмен указателями
}
```

Основная функция (1)

```
int main()
{
    const double pselect = 0.5; // вероятность отбора
    const double pcross = 0.5; // вероятность скрещивания
    int size = 4096; // размер популяции
    int len = 100; // размерность задачи (длина генома)
    genom* P = new genom[size]; // создаем популяцию
    for( int i=0; i<size; i++ )
        create(P[i],len);
    const int tmax = 1000; // число итераций алгоритма
    // лучшее и среднее значения целевой функции на каждом шаге
    double* best = new double[tmax];
    double* average = new double[tmax];
```

Основная функция (2)

```
for( int t=0; t<tmax; t++ ) // основной цикл
{
    shuffle(P,size);
    for( i=0; i<size/2; i++ ) // отбор
        if( frand()<pselect )
            select(P[2*i],P[2*i+1]);
    shuffle(P,size);
    for( i=0; i<size/2; i++ ) // скрещивание
        if( frand()<pcross )
            crossover(P[2*i],P[2*i+1]);
    for( i=0; i<size; i++ ) // мутация
        mutate(P[i]);
```

Основная функция (3)

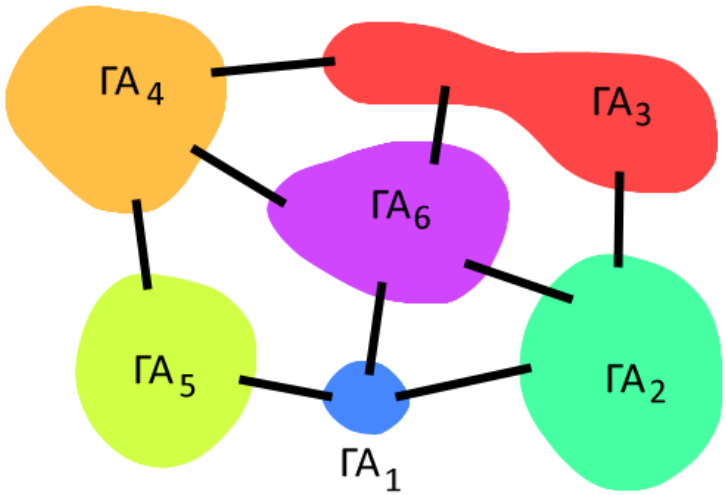
```
    // сбор статистики
    average[t] = 0;
    best[t] = eval(P[0]);
    for( i=0; i<size; i++ )
    {
        double f = eval(P[i]);
        average[t] += f/size;
        if( f<best[t] )
            best[t] = f;
    }
} // конец основного цикла
```

Основная функция (4)

```
    // печать статистики
    for( t=0; t<tmax; t+=10 )
        cout << t << " " << average[t] << " " << best[t] << endl;
    // освобождение памяти
    delete[] best;
    delete[] average;
    delete[] P;
    return 0;
} // конец функции main()
```

Островная модель

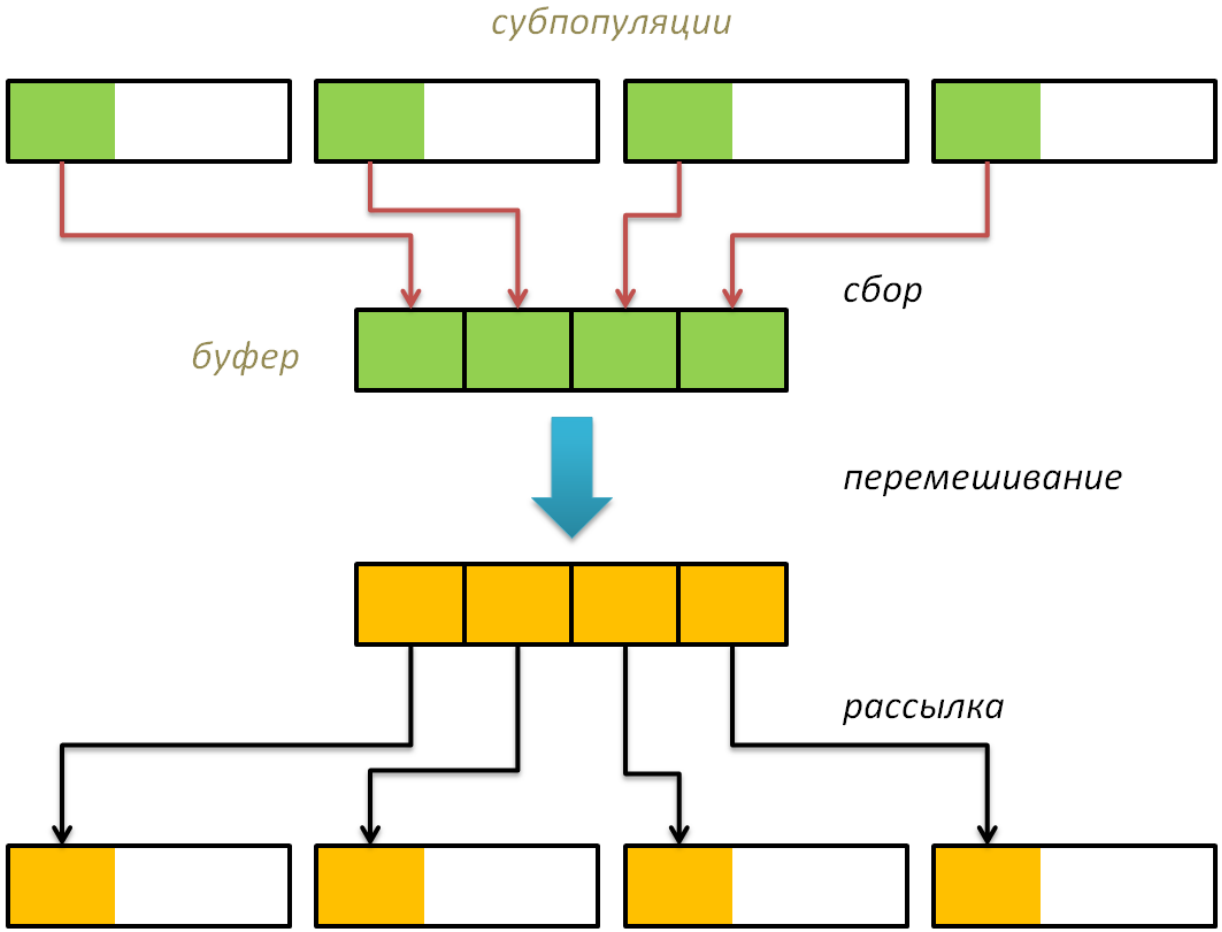
- Суть *островной модели* заключается в том, что вся популяция ГА разбивается на некоторое число субпопуляций, каждая из которых помещается на отдельный процессор вычислительной системы.
- Далее, каждая субпопуляция начинает развиваться независимо друг от друга, по обычным правилам ГА.
- Чтобы организовать обмен информации между различными субпопуляциями, вводится еще один оператор — *миграция*.



Централизованная миграция

- Пусть размер всей популяции равен *size*, а число процессоров (т.е. число субпопуляций) равно p. Тогда размер одной субпопуляции будет равен $size' = size/p$.
- Все субпопуляции развиваются отдельно, через каждые dt шагов алгоритма выполняется миграция.
- В централизованной схеме миграция выполнятся следующим образом:
  - каждая субпопуляция отправляет мастер-процессу часть $size'/fraction$ своих геномов,
  - мастер-процесс перемешивает полученные геномы,
  - и рассылает их обратно по субпопуляциям.

Централизованная миграция



MPI-реализация

- Нулевой процесс яаляется мастер-процессом, все остальные — рабочими.
- Миграция выполняется после первого перемешивания (перед отбором) через каждые dt шагов.
- Рабочие процессы при выполнении миграции пересылают геномы первых $size'/fraction$ особей на мастер-процесс, потом принимают такое же количество геномов от мастер-процесса и записывает их на старое место в массиве P.
- Мастер-процесс при выполнении миграции записывает в отдельный буфер свою порцию геномов, затем принимает от рабочих процессов их геномы, перемешивает буфер, рассылает геномы рабочим процессам, переписывает из буфера в массив P свою часть геномов.

Сбор и печать статистики

- Последним действием параллельного алгоритма является сбор статистики со всех процессов (массивы *best* и *average*) и формирование общей статистики по всей популяции.
- Таким образом, результатом работы одного запуска должен быть файл, в котором для каждого шага алгоритма указаны лучшее значение целевой функции и среднее по всей популяции значение целевой функции.

Практическое задание

- Задание включает в себя а) формулу минимизируемой функции; б) тип скрещивания (одноточечное, двухточечное, равномерное); в) значения основных параметров алгоритма.
- Основным результатом работы должны быть а) графики сходимости алгоритма для значений $p = 2, 4, 8, 16$; б) график зависимости времени работы T от числа процессоров p.
- Дополнительные результаты: а) реализация гибридной модели; б) анализ зависимости сходимости и времени работы от параметров *dt* и *fraction*; в) реализация распределенной модели миграции, например, на кольцевой топологии.

Спасибо за внимание!