# 유전알고리즘을 통한 지수귀문도 풀이

2005/봄 4541.681A 유전알고리즘 과제

신재호 ⟨netj@ropas.snu.ac.kr⟩ (2004-23580)

2005년 6월 3일

## 제 1 절 문제 정의

지수귀문도는 조선시대 영의정을 지냈던 학자 최석정(1646-1715)이 고안한 마방진의 일종으로 동양 전래의 수학과 철학을 담은 숫자놀이이다. 육각형들이 맞물려 거북등처럼 생긴 모양에 각 육각형들의 꼭지점에 적힌 숫자들을 더한 값이 모두 같은 것을 지수귀문도라고 한다. 각 꼭지점에는 1부터 꼭지점 총수까지 서로 다른 수가 한 번씩 나타나야 한다. 그림 1은 1부터 24까지의 숫자를 사용하여 만든 지수귀문도이다.

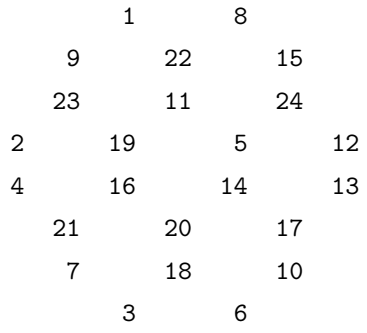


그림 1: 1부터 24까지 사용한 지수귀문도

최적화 연구실의 조사 결과 [2]에 따르면 같은 문제라도 다양한 해가 존재한다. 16짜리 지수귀문도의 경우 임의로 만든 $2^{32}$개의 순열에서도 140,915 개의 해가 발견되었으며 통계적 추측에 따르면 16! 전체에 대해서는 약 6억 개의 서로 다른 해가 존재할 것으로 추정된다. 30짜리의 경우 각 육각형의 합이 79부터 107까지인 다양한 해들이 발견되었다. 특수한 형태의 지수귀문도에 대해서는 해를 찾

는 특수한 방법이 알려져 있지만, 일반적인 형태에 대한 방법은 아직 모른다. 지수귀문도는 어려운 조합적 최적화 문제의 좋은 예이다.

# 제 2 절 해결 방법

## 2.1 혼합형 유전 알고리즘

**개요** 지수귀문도를 최적화 문제로 표현하여 유전 알고리즘을 이용하여 풀기 위해 다음과 같은 적합도 함수 $f$를 사용한다.

$$f = \frac{\mu}{NM} - \sigma^2$$

여기서

| | | | |
|---|---|---|---|
| 육각형의 합들의 평균 | $\mu$ | $=$ | $\sum_{i=1}^{M} H_i/M$ |
| 육각형들의 합의 표준편차 | $\sigma^2$ | $=$ | $\sum_{i=1}^{M}(H_i-\mu)^2/M$ |
| 꼭지점의 총 개수 | $N$ | | |
| 육각형들의 총 개수 | $M$ | | |
| $i$번째 육각형의 합 | $H_i$ | | |

위와 같은 적합도 함수를 사용하는 유전알고리즘의 전체적인 구조는 다음과 같다.

1. 한 세대의 크기 $P$는 512이며 매 세대마다 절반인 $K = 256$개 개의 자손을 만들어 교체한다.
2. 자손을 만드는 데 참여할 두 부모는 품질 비례 룰렛휠 방식을 사용하여 선택한다.
3. 2차원 내추럴 교차를 사용하여 자손을 생성한다.
4. $N/3$개의 유전자를 골라서 1/2의 확률로 그 값을 1만큼 높이거나 낮추는 변이를 적용한다.
5. 변이까지 마친 후에 중복된 숫자가 나타나지 않도록 수선을 해준다.
6. 지역최적화를 적용한다.
7. 이렇게 만든 $K$개의 자손들을 현재 세대 중에서 나쁜 해들 $K$개와 교체한다. 단, 나쁜 해들을 고를 때 $5/K$의 확률로 살려두기도 한다.

8. 위의 과정을 아래 조건이 만족될 때까지 반복한다.

   - 육각형들의 합의 분산이 0인 해를 찾았고
     - 그 해가 가장 적합도가 높거나
     - 가장 적합도가 높은 해의 나이가 충분히 많은 경우(가령, $> 100P/N$),
   - 가장 적합도가 높은 해가 $2000P/N$번 넘게 반복되거나,
   - 제한시간(24의 경우 1분, 54는 5분, 80은 15분, 110은 30분, 224는 두 시간)이 지날 때까지.

**선택 및 교체** 자손을 만드는 데 참여할 두 부모는 품질 비례 룰렛휠 방식으로 선택한다. 앞에서 언급한 함수 $f$를 통해 나온 적합도를 직접 사용하지 않고 아래와 같이 선택압 상수 $sp$로 보정하여 $[1, sp]$ 구간 내로 분포시킨다. 이렇게 보정한 적합도를 기준으로 선택 및 교체를 진행한다.

$$f'_i = (sp - 1)\frac{f_i - f_w}{f_b - f_w} + 1$$

여기서

$$\begin{aligned} f_i &: i\text{번째 해의 적합도} \\ f_w &: \text{세대에서 가장 낮은 적합도} \\ f_b &: \text{세대에서 가장 높은 적합도} \\ sp &: \text{선택압 조절 상수} \end{aligned}$$

선택압 상수 $sp$는 3으로 주어 가장 높은 적합도(3)가 가장 낮은 해(1)의 3배가 되도록 보정한다. 이렇게 보정한 적합도 $f'$을 가지고 $f'$의 크기에 비례해서 선택이 되도록, 0부터 $\sum f'$ 사이의 임의의 난수를 발생시켜 $\sum_{i=0}^{j} f'_i$이 그 난수보다 커지는 가장 작은 $j$를 찾아 $j$ 번째 염색체를 선택한다. 이러한 방법으로 서로 다른 두 부모를 선택한다.

만들어질 자손들을 교체할 해들을 고르기 위해서 $f'$를 기준으로 나쁜 해를 $K$개를 고른다. 이 때 너무 빠른 수렴을 막고 나쁜 해들에게도 기회를 주기 위해서, 세대 중에서 세대 내에서 $f'$이 작은 $K$개의 해들 중에서 $5/K$의 확률로 건너뛰어 그보다 높은 해들을 교체 대상으로 대신한다.

**교차** 2차원 *내추럴 교차(natural crossover)*를 사용한다. 꼭지점 단위로 전체 그림을 두 부분으로 가르기 위해 세로로 된 자름선을 정한다. 자름선을 기준으로 왼쪽은 첫 번째 부모로부터, 그리고 오른쪽은 두 번째 부모로부터 물려받는다.

가장 윗줄에서부터 내려오면서 현재 자름선의 위치를 양옆으로 3칸 이내로 움직이며 자름선을 그어나간다. 자름선은 각 줄의 가장 왼쪽 꼭지점의 왼쪽이나 가장 오른쪽 꼭지점의 오른쪽에도 올 수 있으므로 한쪽 부모에게서만 물려받는 것도 가능하다.

지수귀문도의 실제 그림을 바탕으로 교차를 하므로 각 꼭지점이 염색체의 어느 유전자로 표현했는가에 사실상 아무런 영향을 받지 않는다는 장점이 있다. 그러나 자름선을 3칸씩 가파르게 이동시키더라도 완전히 수평으로 자르는 것이 불가능하기 때문에 수평으로 하는 자름선을 하나 더 사용하는 등의 방법이 보완되어야 할 것이라 생각한다.

**변이** 변이는 각 유전자를 1/3의 확률로 그 값을 1만큼 높이거나 낮춘다. [1]에서 사용하는 방식과 비슷하다. 두 꼭지점을 골라 바꾸기도 하고 임의의 육각형을 골라 돌리기도 하였으나 교란이 너무 강해서 공간탐색을 방해하는 것으로 보여 제외시켰다.

**수선** 교차와 변이 후에 같은 숫자를 가지는 유전자가 염색체 내에 있을 수 있으므로 중복되는 숫자를 찾아 그 *순서를 보존하면서 재배열*을 한다. 처음에는 사용되지 않은 숫자를 임의로 골라 바꿔주는 작업을 하였는데, 이것이 해집단을 매우 불안정하게 만들어 문제공간의 탐색에 방해가 되는 사실을 알 수 있었다.

## 2.2 지수귀문도의 육각형 모양 분석

여기서 다루려는 모든 지수귀문도는 위아래와 양옆으로 선대칭이며, 맨 위와 아래에 $A$개의 육각형이 있고 가운데에는 $B$개의 육각형이 있다 ($A \leq B$). 윗부분에서 각 줄에 있는 육각형의 수는 그 윗줄보다 하나씩 늘어나며 아랫부분에서는 하나씩 줄어든다. 모두 $2(B-A)+1$줄이 있으며 각 줄에서는 최소 $A$개, 최대 $B$개의 육각형이 있다. $i \in \{0, 1, \cdots, 2(B-A)\}$로 줄을 나타내고 $j \in \{0, 1, \cdots, B - |B-A-i| - 1\}$로 그 줄에서 몇 번째 육각형인지를 나타내면 $i, j$를 가지고 지수귀문도의 모든 육각형들을 지칭할 수 있다. (그림 2 참조)

지수귀문도에서 한 육각형의 각 꼭지점들이 인접한 어떤 육각형의 꼭지점들과 겹치는지 파악하였다. 우선 $i, j$로 지칭한 한 육각형에서 각 꼭지점은 왼쪽 위부터 시계방향으로 차례로 0, 1, 2, 3, 4, 5로 가리킨다. (그림 3 참조) 한 꼭지점은 최대 3개의 인접한 육각형과 공유될 수 있다. 이어질 내용에서 육각형 $i, j$의 $k$번째 꼭지점은 $i_1, j_1$과 $i_2, j_2$가 각각 유효한 육각형을 가리킬 때 $i_1, j_1$의 $k_1$번째와 $i_2, j_2$의 $k_2$번째와 같다고 하여 그 관계를 파악해보았다.

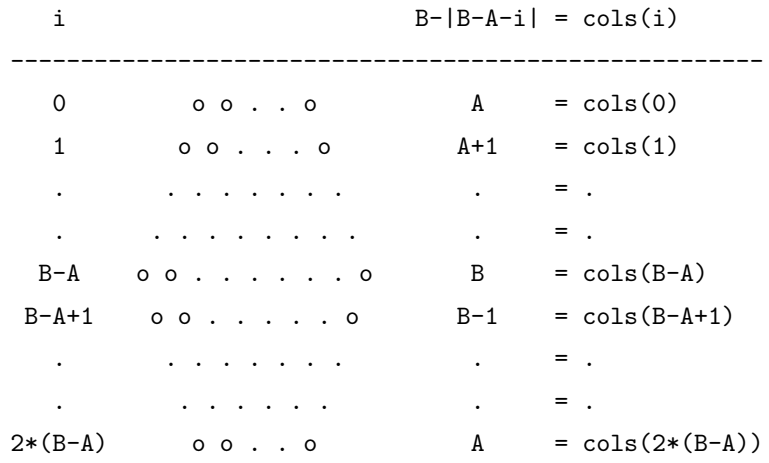


그림 2: 지수귀문도에서 각 줄과 육각형의 개수


그림 3: 꼭지점 번호

$i < B - A$인 경우,

| $k$ | $i_1$ | $j_1$ | $k_1$ | $i_2$ | $j_2$ | $k_2$ |
|---|---|---|---|---|---|---|
| 0 | $i-1$ | $j-1$ | 4 | $i+0$ | $j-1$ | 2 |
| 1 | $i-1$ | $j+0$ | 2 | $i-1$ | $j-1$ | 3 |
| 2 | $i+0$ | $j+1$ | 0 | $i-1$ | $j+0$ | 4 |
| 3 | $i+1$ | $j+1$ | 1 | $i+0$ | $j+1$ | 5 |
| 4 | $i+1$ | $j+0$ | 2 | $i+1$ | $j+1$ | 0 |
| 5 | $i+0$ | $j-1$ | 3 | $i+1$ | $j+0$ | 1 |

$i = B - A$인 경우,

| $k$ | $i_1$ | $j_1$ | $k_1$ | $i_2$ | $j_2$ | $k_2$ |
|---|---|---|---|---|---|---|
| 0 | $i-1$ | $j-1$ | 4 | $i+0$ | $j-1$ | 2 |
| 1 | $i-1$ | $j+0$ | 2 | $i-1$ | $j-1$ | 3 |
| 2 | $i+0$ | $j+1$ | 0 | $i-1$ | $j+0$ | 4 |
| 3 | $i+1$ | $j+0$ | 1 | $i+0$ | $j+1$ | 5 |
| 4 | $i+1$ | $j-1$ | 2 | $i+1$ | $j+0$ | 0 |
| 5 | $i+0$ | $j-1$ | 3 | $i+1$ | $j-1$ | 1 |

$i > B - A$인 경우,

| $k$ | $i_1$ | $j_1$ | $k_1$ | $i_2$ | $j_2$ | $k_2$ |
|---|---|---|---|---|---|---|
| 0 | $i-1$ | $j+0$ | 4 | $i+0$ | $j-1$ | 2 |
| 1 | $i-1$ | $j+1$ | 2 | $i-1$ | $j+0$ | 3 |
| 2 | $i+0$ | $j+1$ | 0 | $i-1$ | $j+1$ | 4 |
| 3 | $i+1$ | $j+0$ | 1 | $i+0$ | $j+1$ | 5 |
| 4 | $i+1$ | $j-1$ | 2 | $i+1$ | $j+0$ | 0 |
| 5 | $i+0$ | $j-1$ | 3 | $i+1$ | $j-1$ | 1 |

세 경우를 한꺼번에 표현하면,

| $k$ | $i_1$ | $j_1$ | $L$ | 0 | $H$ | $k_1$ | $i_2$ | $j_2$ | $L$ | 0 | $H$ | $k_2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $i-1$ | $j$ | $-1$ | $-1$ | $+0$ | 4 | $i+0$ | $j$ | $-1$ | $-1$ | $-1$ | 2 |
| 1 | $i-1$ | $j$ | $+0$ | $+0$ | $+1$ | 2 | $i-1$ | $j$ | $-1$ | $-1$ | $+0$ | 3 |
| 2 | $i+0$ | $j$ | $+1$ | $+1$ | $+1$ | 0 | $i-1$ | $j$ | $+0$ | $+0$ | $+1$ | 4 |
| 3 | $i+1$ | $j$ | $+1$ | $+0$ | $+0$ | 1 | $i+0$ | $j$ | $+1$ | $+1$ | $+1$ | 5 |
| 4 | $i+1$ | $j$ | $+0$ | $-1$ | $-1$ | 2 | $i+1$ | $j$ | $+1$ | $+0$ | $+0$ | 0 |
| 5 | $i+0$ | $j$ | $-1$ | $-1$ | $-1$ | 3 | $i+1$ | $j$ | $+0$ | $-1$ | $-1$ | 1 |

이를 $i = B - A$일 때 $j$와 현재 $i$가 가리키는 줄이 지수귀문도의 가운데 줄보다 위인지 아래인지를 나타내는 $L$, $H$를 가지고 나타내면 표 1과 같은 관계를 얻을 수 있다.

지수귀문도의 육각형 모양에 대한 분석 결과를 바탕으로, 사실상 지수귀문도 자체를 하나의 자연스러운 염색체로 생각할 수 있게 되었다. 표 1을 이용하여 어떠한 유전자 배치에 대해서도 적용이 가능한 일반적인 연산을 구현하는 것이 가능하다. 지수귀문도에서 각 육각형의 꼭지점이 유전자의 어디에 위치하는지를 대응시키는 배열을 만들어두면 되기 때문이다. 유전자 배치에 의존하는 연산들을

$$L = (i < B - A)\ ?\ 1 : 0 \quad H = (i > B - A)\ ?\ 1 : 0$$

| $k$ | $i_1$ | $j_1$ | $k_1$ | $i_2$ | $j_2$ | $k_2$ |
|---|---|---|---|---|---|---|
| 0 | $i-1$ | $j-1+H$ | 4 | $i+0$ | $j-1+0$ | 2 |
| 1 | $i-1$ | $j+0+H$ | 2 | $i-1$ | $j-1+H$ | 3 |
| 2 | $i+0$ | $j+1+0$ | 0 | $i-1$ | $j+0+H$ | 4 |
| 3 | $i+1$ | $j+0+L$ | 1 | $i+0$ | $j+1+0$ | 5 |
| 4 | $i+1$ | $j-1+L$ | 2 | $i+1$ | $j+0+L$ | 0 |
| 5 | $i+0$ | $j-1+0$ | 3 | $i+1$ | $j-1+L$ | 1 |

표 1: 지수귀문도에서 각 꼭지점을 공유하는 육각형들

특정한 유전자 배치로부터 독립시킴으로써, 육각형들의 합 구하기, 2차원 교차, 교란 등을 쉽게 구현할 수 있었다. 더불어 유전자 배치를 쉽게 바꾸어 수행을 할 수 있는 유연한 체계를 갖출 수 있었다.

이런 유연한 체계를 가지고 유전자 배치 실험도 해볼 수 있을 것이다. 표 1을 이용하면 임의의 유전자 배치도 쉽게 만들어낼 수 있다. 각 꼭지점들이 실제로 저장되는 배열을 1차원 염색체로 보고, N점 교차 등 유전자 배치와 밀접한 관계가 있는 연산과 가장 잘 어울리는 좋은 유전자 배치가 무엇인지 찾는 데에 큰 도움이 될 수 있을 것이라 생각한다. 더불어 유전알고리즘 수행 도중에 유전자 배치를 바꾸어가며 좋은 스키마들의 생존 확률을 높이는 것도 가능해진다.

**표현** 현재는 가장 왼쪽 위 육각형부터 오른쪽으로 가고 줄 별로 내려가면서, 각 육각형의 왼쪽 위 꼭지점부터 시계방향으로 돌면서 검사하여, 유전자 위치가 할당이 되지 않았으면 순서대로 번호를 주는 식으로 유전자 배치를 한다. 이 과정에서 표 1을 이용하여 같은 꼭지점을 나타내는 $i$, $j$, $k$에 모두 동일한 유전자 번호를 할당할 수 있다. 가령, 24짜리의 경우에는 $A = 2, B = 3$으로 그림 4과 같으며, 54, 80, 110, 224의 경우 $A = 3$과 각각 $B = 5, 6, 7, 9$으로 놓고 같은 방식으로 유전자 배치를 하였다.

## 2.3 지역최적화 및 휴리스틱

**꼭지점 바꾸기로 적합도 올리기** 새로 만든 염색체를 세대로 집어넣기 전에 두 유전자들을 바꿔치기하여 적합도를 최대로 만드는 지역최적화를 하였다. 즉, 매

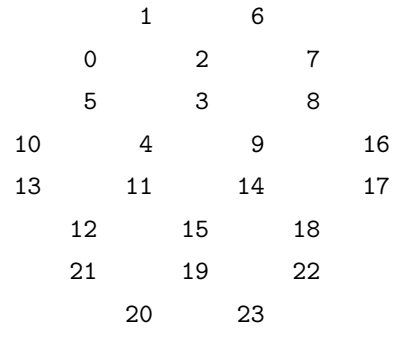


그림 4: 24짜리의 유전자 배치

자손 염색체가 만들어지고 교체를 하기 직전에 각 유전자들의 자리를 바꿨을 때 적합도가 더 높아지는가를 확인하여 그럴 경우에는 바꿔주었다. 이 과정을 모든 유전자 쌍들에 대해서 더 이상 적합도가 높아질 수 없을 때까지 진행하였다.

이 방법은 한 염색체의 모든 유전자 쌍마다 적합도를 다시 계산해야 한다. 새 적합도를 계산하기 위해서는 모든 육각형들의 합의 평균 $\mu$와 제곱의 합의 평균 $\bar{H}_i^2$을 구해야 한다. 그러나 한 쌍의 유전자를 바꿨을 경우만 고려하므로, 사실상 이 두 유전자가 속한 육각형을 제외하고는 합과 제곱의 합이 변하지 않는다.

불필요한 계산은 가급적 건너뛰고 변화되는 양만을 고려하여 적합도를 효율적으로 계산한다. 한 유전자, 즉 한 꼭지점이 공유될 수 있는 육각형은 최대 3개이다. 따라서 두 유전자를 서로 바꿔치기 했을 때, 최대 6개의 육각형의 합만이 변할 수 있다. 이 사실을 이용하여 육각형들의 합과 제곱의 합을 변화량만을 가지고 다음과 같이 효율적으로 계산할 수 있다.

위의 지역최적화를 보다 효율적으로 하기 위하여 적합도의 개선 정도에 한계를 두거나 인접한 바꿔치기를 하였다. 전자를 통해서 아주 나쁜 해들을 불필요하게 최적화 하는 것을 방지할 수 있었다. 그런데 [1]에서 소개하는 *인접한 바꿔치기 (consecutive exchange)*를 사용할 경우, 위와 같이 최대한 바꿔치기를 하는 것과 비슷한 효과를 내면서도 효율이 더 좋아 이를 채용하였다.

**나이** 각 해들을 시간이 지남에 따라 적합도를 떨어뜨려 보다 넓은 탐색이 가능하게 하였다. 모든 해의 나이는 한 세대가 지남에 따라 1씩 증가한다. 해의 나이가 $a$일 때, 그 해의 보정된 적합도 $f'$을 다음과 같이 보정하여 나이가 반영된 적합도 $f''$을 얻는다.

$$f'' = f' - \frac{N}{10P}a$$

$f'$ 대신에 $f''$을 이용하여 선택 및 교체를 진행하였다.

**교란** 세대가 수렴하는 문제를 해결하기 위해 수렴 정도가 심하면 해들을 교란 시키는 방법을 썼다. 세대의 분산의 평균과 가장 좋은 해의 분산의 차이가 0.005보다 작은 것이 $K/2$회 이상 반복되면, 세대 중 1/4을 선택하여 다음과 같은 두 교란 방법을 적용한다. 첫 번째는 임의로 두 유전자를 골라 바꿔치기 하는 것이다. 유전자 쌍을 최대 둘까지 골라서 바꿔치기 해준다. 두 번째는 육각형을 임의로 선택하여 회전을 시키는 방법이다. 선택한 육각형의 각 꼭지점을 1에서 5칸 시계방향으로 회전시킨다. 회전 역시 최대 두 육각형까지 한다.

**유전자 재배치** 유전자 배열을 무작위로 시작하여 가장 좋은 해의 개선이 별로 없을 경우에 새로운 유전자 배치를 가지고 진행하는 방법을 사용해보았다. 새 배치를 임의로 생성한 뒤에 기존 배치에 맞게 저장된 해들을 새 배치에 맞게 바꾸고 진행한다. 그러나 유전자 배치에 독립적인 2차원 교차를 사용하기 때문에 이는 별로 의미가 없어서 심각하게 사용하지는 않았다.

# 제 3 절 구현

C언어로 구현하였으며 중복되는 계산이 최소가 되도록 심혈을 기울여 만들었다. 반복되는 다수의 코드는 매크로를 이용하여 간소화 하였다. `alarm()`을 이용하여 전체 수행 시간 제한을 두었으며, `SIGINT`와 `SIGQUIT`을 받아서 현재 세대의 상황과 가장 좋은 해를 출력하도록 만들어 성능에 저하를 주지 않으면서도 진행상황을 점검하기 쉽게 하였다. pthread 라이브러리를 이용하여 각 자손을 만들어내는 부분처럼 독립적인 계산은 3개의 일꾼 쓰레드가 담당하게 하여 병렬화 하였다.

## 3.1 가속하기

계산을 가속하기 위한 몇몇 아이디어를 사용하였다. 새로운 자손을 부모와 교체할 때 적합도, 평균, 분산 등 항상 전체를 새로 계산하지 않고 변하는 양만큼을 가감하여 기록해 계산양이 최소가 되도록 하였다. 자손과 세대 내의 해 사이의 포인터 바꿔치기만으로 교체를 하여 염색체의 복사로 생기는 비효율을 최대한 줄였다. 표준편차를 계산할 때 효율적으로 하기 위해서 분산을 다음과 같이 제곱의 평균과 평균의 제곱을 이용하여 계산하였다.

$$\sigma^2 = \bar{H_i^2} - \bar{H_i}^2$$

**변이 가속** 한 염색체 내에서 1/3의 확률로 유전자를 골라 변이를 시키기 위해서 $N/3$개를 뽑아 변이를 시켰다. 항상 서로 다른 $N/3$개의 유전자가 선택되도록

하기 위하여 다음 유전자의 위치를 난수로 발생시킬 때, 뽑아야할 남은 개수 만큼을 반드시 남겨두고, 하나를 뽑은 다음에는 반드시 그 다음 칸에서부터 선택하도록 하였다. $N$개의 해들 각각에 대해서 난수를 발생시켜 1/3의 확률로 가감을 하는 경우 반드시 $N$번의 반복이 필요한 반면, 이렇게 할 경우 $N/3$번의 반복만 하면 되므로 3배가 빨라진다.

**바꾸기로 적합도 올리기 가속** 적합도를 비교해가면서 각 꼭지점 쌍을 바꾸는 지역최적화에서 매번 적합도를 계산해야 한다. 그런데 바뀌는 꼭지점은 둘 뿐이므로 대부분의 육각형의 합은 변하지 않는다. 따라서 적합도 계산에 필요한 육각형들의 합 및 제곱의 합을 매번 다시 계산하는 것을 대단히 비효율적이다. 두 꼭지점을 바꿀 경우 합이 변하는 육각형들만 다시 계산하면서 지역최적화를 가속할 수 있는 방법을 고안하였다.

우선, 바꿔치기 했을 때의 적합도를 알아볼 두 유전자 $x$와 $y$를 생각하자. $x$를 꼭지점으로 가지는 육각형들의 꼭지점의 합을 각각 $H_i$라 하고, $y$를 가지는 육각형들을 $H_j$라 하자. 적합도를 계산하기 위해서 평균과 분산이 필요하므로, 염색체가 나타내는 모든 육각형들의 합 $S$와 제곱의 합 $SS$가 필요하다. $x$나 $y$가 속하지 않은 육각형들은 그 합 및 제곱이 변하지 않으므로, $x$나 $y$가 속한 육각형들만을 고려해도 된다. 즉, $S$와 $SS$를 다음과 같이 놓고 이들의 변화만을 고려해도 된다.

$$\begin{aligned} \delta &= y - x \\ S &= \sum H_i + \sum H_j \\ SS &= \sum H_i^2 + \sum H_j^2 \end{aligned}$$

이제, $H_i$들에 들어있는 $x$를 $H_j$들에 들어있는 $y$와 바꾸면, 각 $H_i$는 $y-x$만큼 커지고, $H_j$는 $x-y$만큼 커진다. $y-x$를 $\delta$로 놓으면, 관계된 모든 육각형들의 새 합 $S'$와 새 제곱의 합 $SS'$은 다음과 같다.

$$\begin{aligned} S' &= \sum(H_i + \delta) + \sum(H_j - \delta) \\ SS' &= \sum(H_i + \delta)^2 + \sum(H_j - \delta)^2 \end{aligned}$$

한 꼭지점을 공유하는 육각형은 최대 3개이지만, 둘레 쪽으로 가면 2개 또는 1개까지 될 수 있다. $x$를 공유하는 육각형 $H_i$들이 $m$개이고, $y$를 공유하는 육각형 $H_j$들이 $n$개라고 해보자. 만약 $x$와 $y$가 인접한 경우라면, 이들을 공유하는 육각형이 겹칠 수도 있다, 즉 원래부터 같은 육각형에 속한 꼭지점인 경우다. 이 경우에는 바꿔치기를 하더라도 해당 육각형의 합이나 제곱의 합은 변하지 않는다. 이렇게 겹치는 육각형의 수는 $c$개라고 해보자. 그러면 $S'$와 $SS'$는 다음과 같이 원

래의 $S$, $SS$와 함께 그 변화량을 가지고 나타낼 수 있다.

$$\begin{aligned} S' &= \textstyle\sum^{m-c}(H_i+\delta) + \sum^{n-c}(H_j-\delta) \\ &= \textstyle\sum H_i + \sum^{m-c}\delta + \sum H_j - \sum^{n-c}\delta \\ &= \textstyle\sum H_i + \sum H_j + \sum^{m-c}\delta - \sum^{n-c}\delta \\ &= S + \textstyle\sum^{m-c}\delta - \sum^{n-c}\delta \\ &= S + (m-c)\delta - (n-c)\delta \\ &= S + (m-n)\delta \end{aligned}$$

$$\begin{aligned} SS' &= \textstyle\sum^{m-c}(H_i+\delta)^2 + \sum^{n-c}(H_j-\delta)^2 \\ &= \textstyle\sum^{m-c}(H_i^2+2\delta H_i+\delta^2) + \sum^{n-c}(H_j^2-2\delta H_j+\delta^2) \\ &= \textstyle\sum H_i^2 + 2\delta\sum H_i + \sum^{m-c}\delta^2 + \sum H_j^2 - 2\delta\sum H_j + \sum^{n-c}\delta^2 \\ &= \textstyle\sum H_i^2 + \sum H_j^2 + 2\delta\sum H_i - 2\delta\sum H_j + \sum^{m-c}\delta^2 + \sum^{n-c}\delta^2 \\ &= SS + 2(\textstyle\sum H_i - \sum H_j)\delta + \sum^{m+n-2c}\delta^2 \\ &= SS + (2(\textstyle\sum H_i - \sum H_j) + (m+n-2c)\delta)\delta \end{aligned}$$

따라서 기존의 합 $S$와 제곱의 합 $SS$, 그리고 현재 유전자 $x$와 $y$에 관계된 $n, m, c, \sum H_i, \sum H_j$를 알면, 새 합 $S'$과 제곱의 합 $SS'$을 바로 알 수 있으며 적합도 또한 바로 따라 나온다. 만약 $S'$와 $SS'$으로부터 나오는 적합도가 더 크다면 바꿔치기를 한다. 바꿔치기를 하면 관계된 육각형들의 합이 바뀌므로, $H_i$와 $H_j$들만 각각 $\delta$를 가감해주면 된다. 따라서, 염색체가 나타내는 모든 육각형들의 합을 맨 처음에만 기록해두고 위와 같이 매 바꿔치기에서 변화되는 육각형들만 조정해가면서 지역최적화를 진행할 수 있다. 매번 적합도를 계산할 때 필요한 모든 육각형들의 합과 제곱의 합도 마찬가지로 처음에만 계산을 해두고, 매 바꿔치기마다 그 변화량 $(m-n)\delta$와 $(2(\sum H_i - \sum H_j) + (m+n-2c)\delta)\delta$ 값만을 계속 더해주면서 그 변화를 유지해가면 된다.

## 제 4 절 실험 결과

앞서 설명한 구현을 가지고 Pentium4 3.2GHz, 4GB 메모리를 가진 기계에서 Linux 2.6 커널 위에서 실험하였다. 문제 풀이 실험의 결과는 표 2에 나와있다. 24짜리의 경우는 엄청나게 빠른 속도로 해를 찾으며 54 이상의 경우도 최대 적합도가 10 이상씩 올라갔음을 확인할 수 있다.

54, 80, 110짜리에서 찾은 가장 좋은 해들의 그림은 각각 5, 6, 7과 같다. 224 이상에서는 아직 해를 찾지 못했다. 80짜리 및 110짜리의 해를 찾는 실험 하나를 골라 세대가 지남에 따라서 세대 내의 해들의 평균 적합도와 최대 적합도, 평

균 분산과 가장 좋은 해의 분산이 어떻게 변하는지를 그림 8 및 9과 같이 그려보았다. 5번째 이전은 분산이 너무 크고, 1000번째 이후는 변화를 관찰하기 어려워 5번부터 1000번 세대 사이만 그렸다.

| $N$ | 해 수 | 평균 적합도 | 최대 적합도 | 평균 수행 시간 | 평균 세대 수 |
|---|---|---|---|---|---|
| 24 | 3228 | 76.87825 | 85.00000 | 0.03s | 12.72 |
| 54 | 198 | 178.62932 | 190.00000 | 18.90s | 819.55 |
| 80 | 184 | 268.61760 | 281.00000 | 248.62s | 6419.08 |
| 110 | 104 | 372.23091 | 381.00000 | 1049.55s | 19756.41 |
| 224 | 1 | 744.90783 | 744.90783 | 14293.58s | 89660.00 |

표 2: 실험 결과

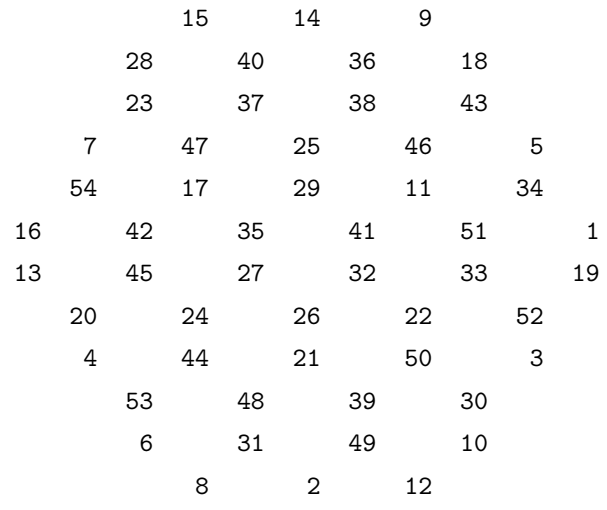


그림 5: 54짜리에서 찾은 가장 좋은 해. (합 190)

## 제 5 절 결론 및 고찰

순수 유전알고리즘만 가지고도 지수귀문도의 꼭지점 수가 24인 경우에는 대부분 답을 쉽게 찾아낸다. 그러나 54짜리부터는 세대 안의 해들이 분산을 빠른 속도로 낮추어 지역해로 이동은 하지만, 공간 특성상 별로 매력이 크지 않은 한 지역해에 걸려서 수렴한 후 실제 해로 쉽게 올라가지 못한다. 적합도를 고려하여 두 꼭지점을 바꾸는 지역최적화 및 나이, 교란 등의 휴리스틱을 사용하여 순수 유전 알고리

```
                19       3      14
            22      53      60      13
            47      69      63      65
        12      71      33      66       5
        39      37      28      51      30
     6      75      43      40      64      21
    56      35      50      72      31      57
15      70      41      48      23      78       1
11      74      27      54      25      80      20
    55      34      61      59      44      45
    17      77      36      42      73       7
        24      46      29      38      32
         4      68      26      79      10
            62      76      67      49
             8      58      52      16
                 9       2      18
```

그림 6: 80짜리에서 찾은 가장 좋은 해. (합 281)

즘만으로는 찾지 못하는 54짜리 이상에서도 해를 구할 수 있었다. 54짜리는 190, 80짜리는 281, 110짜리는 381의 합을 갖는 지수귀문도를 찾았다.

## 5.1 고찰

온갖 방법을 통해서 해 찾기에 도전해보았지만 번번히 실패했다. 그러다가 무작위로 하던 수선을, 순서를 보존하는 방식으로 바꾸어보니 해를 매우 잘 찾기 시작했다. 이것은 유전 알고리즘이 랜덤하게 해를 찾는 것이 아니라 교차에 매우 의존하여, 정말로 유전을 통해서 문제공간을 탐색하는 것이라는 사실을 암시한다고 생각한다. 뿐만 아니라 지역최적화, 변이 등 다른 유전 알고리즘 연산자도 마찬가지로 유전이 제대로 되지 않게 교란을 시키면 유전 알고리즘 전체가 제대로 동작하지 않는다는 사실을 발견했다. 유전 알고리즘이 무작위로 공간 탐색하는 것과 다를 바가 없을거라는 편견이 있었는데, 본 과제를 통해서 정말로 *유전*을 통해서 문제를 접근하는 알고리즘임을 확인할 수 있었다.

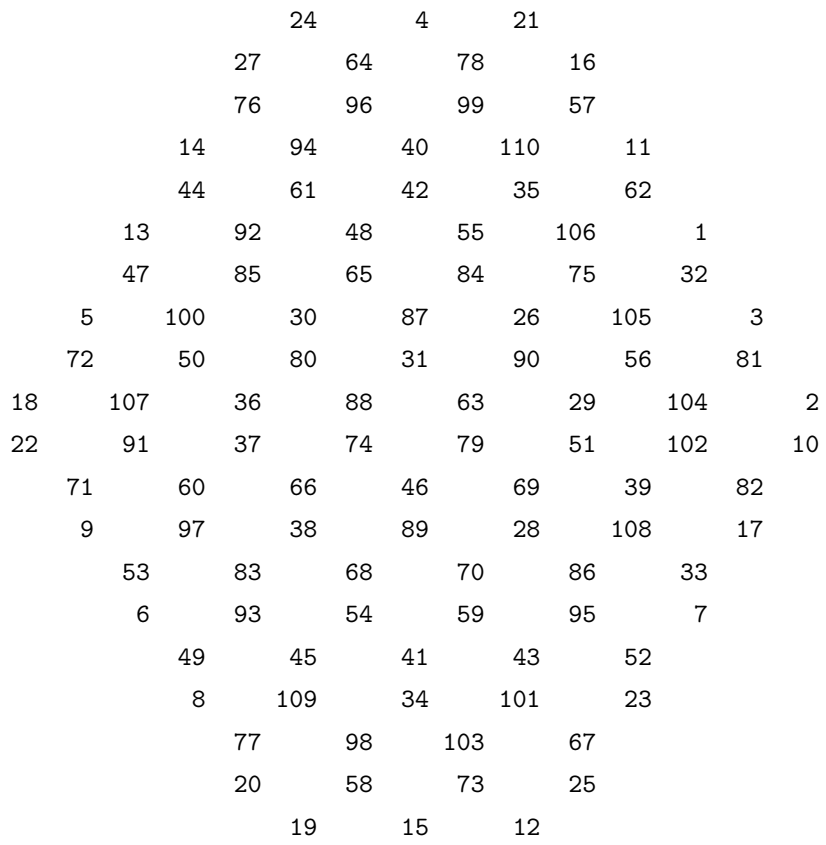


그림 7: 110짜리에서 찾은 가장 좋은 해. (합 381)

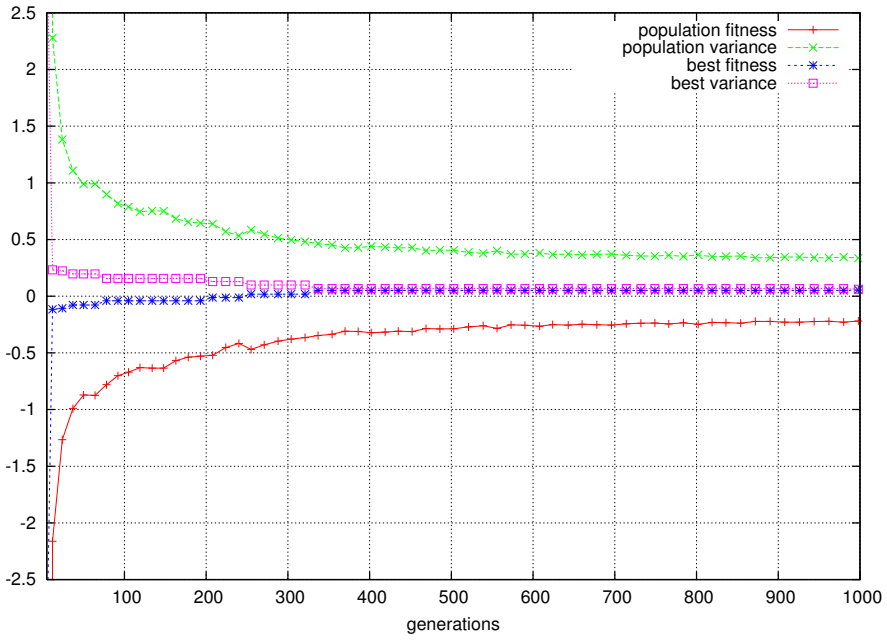


그림 8: 80짜리를 풀 때 세대 변화

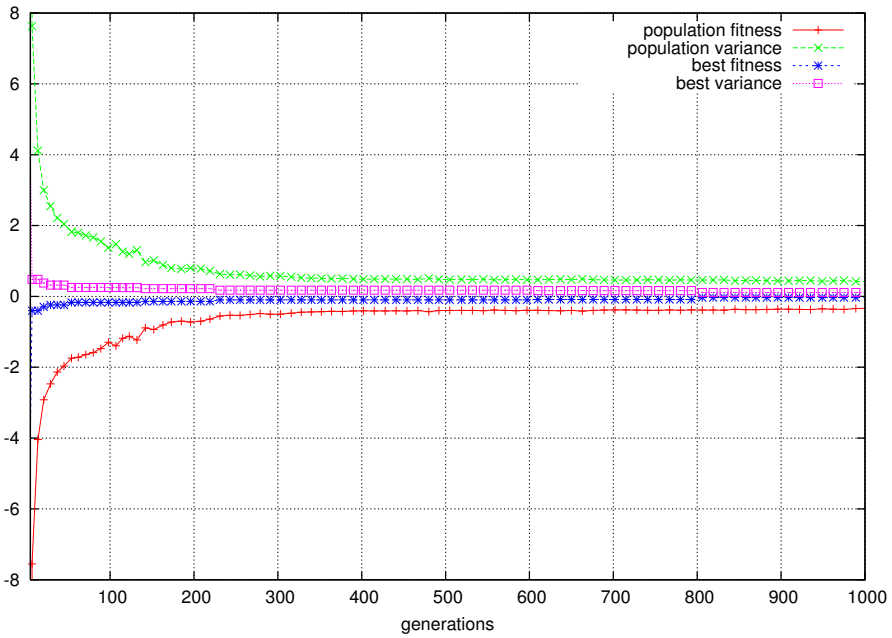


그림 9: 110짜리를 풀 때 세대 변화

### 5.2 앞으로의 계획

2차원 교차를 수평 뿐만 아니라 수직으로도 자르면 좀 더 다양한 공간을 탐색할 수 있을 것으로 보인다. 유전자 배치 실험을 통해 좋은 스키마가 보존되는 유전자 배치를 찾아보는 것도 좋겠다. 보통 한 두 개의 육각형의 합을 맞추지 못해서 허덕이는데, 이를 해결할 수 있는 지역최적화를 추가하면 큰 도움이 될 것으로 보인다. 적합도 함수를 고르기가 매우 까다로운데 합과 분산 각각을 최적화하기위한 복수의 목적함수를 두어 진행하는 것도 좋을 것 같다.

## 참고 문헌


[1] Heemahn Choe, Sung-Soon Choi, and Byung Ro Moon. A hybrid genetic algorithm for the hexagonal tortoise problem. In Erick Cantú-Paz, James A. Foster, Kalyanmoy Deb, Lawrence Davis, Rajkumar Roy, Una-May O'Reilly, Hans-Georg Beyer, Russell K. Standish, Graham Kendall, Stewart W. Wilson, Mark Harman, Joachim Wegener, Dipankar Dasgupta, Mitchell A. Potter, Alan C. Schultz, Kathryn A. Dowsland, Natasa Jonoska, and Julian F. Miller, editors, *GECCO*, volume 2723 of *Lecture Notes in Computer Science*, pages 850–861. Springer, 2003.

[2] 문병로. *유전알고리즘*. 두양사, 2003.

# 부록 A 프로그램 소스코드

```
/*****************************************************************************
 * 4541.681A Genetic Algorithm (2005/Spring) 지수귀문도                      *
 * 2004-23580  Jaeho Shin <netj@ropas.snu.ac.kr>                              *
 * Created: 2005-04-26                                                        *
 *****************************************************************************/

#ifndef A
#define A 3
#endif
#ifndef B
#define B 8
#endif
#if A>B
#error "A must be less than or equal to B"
#endif
#define SUM(n) ((n)*((n)+1)/2)
#define M (2*(SUM(B-1) - SUM(A-1)) + B)
#define N (4*(SUM(B) - SUM(A-1)) + 2*(B-A+1))

#define P 512
#define K (P * 1 / 2)
#define F(mean, var) ((mean) / M / N -(var))
#define Fopt(mean, var) ((mean) - sqrt((var)* N))
#define SELPRESS 3
#define MUTRATE N/4
#define REPLACE_WORST K/5
//#define XOVER_CUTS 2
#define XOVER_2DCUTMOVE 3

//#define OPTFULL
//#define OPTMAXIMPROVE 1.5
//#define OPTAPPLY K/4

//#define ENCODING_RANDOM

//#define EXPLOSION P/4
#define EXPLOSION_DIFF 0.005
#define EXPLOSION_THRESHOLD K/2
#define SWAPRATE 2
#define HEXSPINRATE 2

#define DONE \
    (solfound > 0 && (fbestidx == var0idx || gbest_age > P * 100 / N) \
     || gbest_age > P * 2000 / N)

#ifdef DEBUG
#define debug(fmt...) fprintf(stderr, "## " fmt)
#else
#define debug(fmt...)
#endif
#define message(fmt...) fprintf(stderr, "# " fmt)


#include <stdio.h>
#include <stdlib.h>
```

```
56  #include <string.h>
57  #include <math.h>
58  #include <unistd.h>
59  #include <signal.h>
60  #include <sys/types.h>
61  #include <sys/resource.h>
62  #include <sys/time.h>
63  #include <time.h>
64
65
66  typedef short idx_t;
67  typedef short gene;
68  typedef gene *chromo;
69  typedef gene Chromo[N];
70  typedef double fitness_t;
71  typedef struct _evaluation {
72      fitness_t fitness, var, mean;
73      int age;
74  } Evaluation;
75  typedef Evaluation *evaluation;
76
77  #define new_chromosome() malloc(N * sizeof(gene))
78  #define copy_chromosome(dest, src) memcpy(dest, src, N * sizeof(gene))
79  #define new_evaluation() malloc(sizeof(Evaluation))
80
81  int generations = 0,
82      solfound = 0;
83  chromo pop[P], offspring[K];
84  evaluation eval[P], offspring_eval[K];
85  fitness_t fbest = -1.0/0.0, fworst = 1.0/0.0,
86            fsum = 0, varsum = 0, meansum = 0;
87  int find_worst = 0,
88      find_best = 0,
89      fbestidx = -1,
90      var0idx = -1;
91  int parent1[K], parent2[K];
92
93  fitness_t gbest = -1.0/0.0;
94  int gbest_age = 1;
95
96  #ifdef ENCODING_RANDOM
97  int encoding_age = 1;
98  #endif
99
100 fitness_t ffsum = 0, ff[P];
101 int rank[P];
102
103 #ifdef EXPLOSION
104 int explosion_countdown = EXPLOSION_THRESHOLD;
105 #endif
106
107 #define rnd_one() \
108     (((double)rand())/((double)RAND_MAX + 1.0))
109 #define rnd_f(ub) ((double)(ub) * ((rnd_one() + rnd_one()) / 2.0))
110 #define rnd(n) ((unsigned int)rnd_f(n))
111 #define min(a, b) (((a) < (b)) ? (a) : (b))
112 #define max(a, b) (((a) > (b)) ? (a) : (b))
```

```
113
114
115 /*****************************************************************************
116  * Hexagon Structure                                                         *
117  *****************************************************************************/
118
119 #define prepare_C(i) \
120         C = B - ((i < B-A) ? B-A - i : i - (B-A))
121 #define prepare_LH(i) L = (i < B-A) ? 1 : 0, \
122                       H = (i > B-A) ? 1 : 0
123 #define prepare_LHC(i) \
124         prepare_LH(i), prepare_C(i)
125 /* NOTE: Following variables must be defined within the same scope:
126  *       idx_t i,j,C; char L,H;  */
127
128 #define for_each_i \
129         for (i=0, C=A, \
130                 prepare_LH(i); \
131                 i<=2*(B-A); \
132                 i++, C += L ? +1 : -1, \
133                 prepare_LH(i) \
134             )
135 #define for_each_j \
136         for (j=0; j<C; j++)
137 #define for_each_ij for_each_i for_each_j
138
139 idx_t idx[2*(B-A)+1][B][6];
140 idx_t shares[N][3][3];
141 char numshares[N];
142 /* filling idx */
143 void initialize_encoding() {
144     /* initialize idx for gene # assignment */
145     idx_t i,j,C; char L,H,k;
146     for_each_ij {
147         for (k=0; k<6; k++)
148             idx[i][j][k] = N+1;
149     }
150     int gene_idx_cnt = 0;
151 #ifdef ENCODING_RANDOM
152     int gene_idx[N] = {0};
153     int next_gene_idx() {
154         int c = rnd(N - gene_idx_cnt--);
155         int i = rnd(N);
156         while (c >= 0) {
157             while (gene_idx[i])
158                 i = (i+1) % N;
159             c--;
160         }
161         gene_idx[i] = 1;
162         return i;
163     }
164 #else
165 #define next_gene_idx()  gene_idx_cnt++
166     // TODO: There must be a better encoding..
167 #endif
168     /* assign gene indexes */
169     for_each_ij {
```

```
#define if_exists(i0, j0) \
        if (0<=i0 && i0<=2*(B-A) && 0<=j0 && j0<C+(i0-i)*(L?1:H?-1:-(i0-i)))
#define assign(i,j,k) \
        numshares[gene_idx]++, \
        idx[i][j][k] = gene_idx, \
        share = shares[gene_idx][s++], \
        share[0] = i, share[1] = j, share[2] = k
#define share(k, i1, j1, k1, i2, j2, k2) \
        if (idx[i][j][k] > N) { \
            int gene_idx = next_gene_idx(); \
            idx_t *share; char s = 0; \
            numshares[gene_idx] = 0; \
                                assign(i   ,j   ,k ); \
            if_exists(i+i1, j+j1) assign(i+i1,j+j1,k1); \
            if_exists(i+i2, j+j2) assign(i+i2,j+j2,k2); \
        }
        share(0,  -1, -1+H, 4,  00, -1+0, 2);
        share(1,  -1, 00+H, 2,  -1, -1+H, 3);
        share(2,  00, +1+0, 0,  -1, 00+H, 4);
        share(3,  +1, 00+L, 1,  00, +1+0, 5);
        share(4,  +1, -1+L, 2,  +1, 00+L, 0);
        share(5,  00, -1+0, 3,  +1, -1+L, 1);
#undef share
#undef assign
#undef if_exists
#undef next_gene_idx
    }
}

void print_chromo(FILE *out, chromo c) {
    idx_t i, j, C;
#define print(fmt...) fprintf(out, fmt)
#define indent() for (j=C; j<B; j++) print("    ");
    for (C=A, i=0; C<=B; C++, i++) {
        indent(); for_each_j print("    %3d ", c[idx[i][j][1]]); print("\n");
        indent(); for_each_j print("%3d     ", c[idx[i][j][0]]);
        print("%3d\n", c[idx[i][j-1][2]]);
    }
    for (C=B, i=B-A; C>=A; C--, i++) {
        indent(); for_each_j print("%3d     ", c[idx[i][j][5]]);
        print("%3d\n", c[idx[i][j-1][3]]);
        indent(); for_each_j print("    %3d ", c[idx[i][j][4]]); print("\n");
    }
#undef print
#undef indent
}

/* sum of all points in hexagon i,j */
inline int hexsum(chromo c, idx_t i, idx_t j) {
    int s = 0;
    char k;
    idx_t *ij = idx[i][j];
    for (k=0; k<6; k++)
        s += c[ij[k]];
    return s;
}
```

```
227 /* evaluation of variance, mean, fitness for a given chromosome */
228 void evaluate(chromo c, evaluation e) {
229     int s, sum = 0, sqsum = 0;
230     idx_t i,j,C; char L,H;
231     for_each_ij {
232         s = hexsum(c,i,j);
233         sum += s;
234         sqsum += s*s;
235     }
236     fitness_t mean   = (fitness_t)sum / M,
237               sqmean = (fitness_t)sqsum / M;
238     e->mean     = mean;
239     e->var      = sqmean - mean*mean;
240     e->fitness  = F(mean, e->var);
241 }
242
243 inline fitness_t fitness(int sum, int sqsum) {
244     fitness_t mean = (fitness_t)sum / M;
245     return Fopt(mean, (fitness_t)sqsum / M - mean*mean);
246 }
247
248 /* exhaustive local optimization */
249 inline void exhaustive_opt(chromo c) {
250     idx_t i,j,C, tmp, g1,g2, *share;
251     char L,H, s,s1,s2, m,n,cm;
252     int hsum[2*(B-A)+1][B], p,q;
253     int sqsum = 0, sum = 0, d, newsum, newsqsum;
254     for_each_ij {
255         d = hexsum(c,i,j);
256         hsum[i][j] = d;
257         sum += d;
258         sqsum += d*d;
259     }
260     fitness_t orig = fitness(sum, sqsum), new;
261 #ifdef OPTMAXIMPROVE
262     fitness_t init = orig;
263 #endif
264 #ifndef OPTFULL
265     /* build inverse map */
266     idx_t inv[N+1], g;
267     for (g=0; g<N; g++)
268         inv[c[g]] = g;
269 #endif
270 #define prepare(S, h, g) \
271         S = numshares[g]; \
272         for (h=0,s=0; s<S; s++) \
273             share = shares[g][s], h += hsum[share[0]][share[1]];
274     char changed;
275     do {
276         changed = 0;
277 #ifdef OPTFULL
278         /* for every pair of genes */
279         for (g1=0; g1<N-1; g1++) {
280             prepare(m,p, g1);
281             for (g2=g1+1; g2<N; g2++)
282 #else
283         /* for each consecutive gene values */
```

```
284         g1 = inv[1];
285         prepare(m,p, g1);
286 #define d 1
287         for (g=2;  g<=N;  g1=g2,m=n,p=q,g++) {
288             g2 = inv[g];
289 #endif
290             /* try swapping */
291             {
292                 prepare(n,q, g2);
293                 /* find common ones */
294                 for (cm=0, s1=0; s1<m; s1++)
295                     for (s2=0; s2<n; s2++)
296                         if (shares[g1][s1][0] == shares[g2][s2][0] &&
297                                 shares[g1][s1][1] == shares[g2][s2][1])
298                             cm++;
299                 if (m+n-2*cm > 0) {
300                     /* compute the difference of fitness when g1, g2 are swapped */
301 #ifdef OPTFULL
302                     d = c[g2] - c[g1];
303 #endif
304                     newsum = sum + d*(m-n);
305                     newsqsum = sqsum + d*(2*(p-q) + d*(m+n-2*cm));
306                     new = fitness(newsum, newsqsum);
307                     /* XXX: gcc -O2 bug.
308                      *      new > orig is true even when new == orig. */
309                     if (new > orig && new != orig
310                             && (newsqsum != sqsum || newsum != sum)) {
311                         /* if increased, then swap g1 and g2 */
312 #ifdef OPTFULL
313                         tmp = c[g1], c[g1] = c[g2], c[g2] = tmp;
314 #else
315                         c[g1] = g, c[g2] = g-1;
316 #endif
317 #ifdef OPTMAXIMPROVE
318                         if (new / init > OPTMAXIMPROVE)
319                             return;
320 #endif
321                         /* update relevant values */
322                         changed = 1;
323                         orig = new, sum = newsum, sqsum = newsqsum;
324 #define update_hsum(g, S, d) \
325                         for (s=0; s<S; s++) \
326                             share = shares[g][s], \
327                             hsum[share[0]][share[1]] += d;
328                         update_hsum(g1,m,+d);
329                         update_hsum(g2,n,-d);
330                         p += (m-cm)*d;
331 #ifndef OPTFULL
332                         q -= (n-cm)*d;
333                         inv[g-1] = g2, inv[g] = g1;
334                         g2 = g1;
335 #undef d
336 #endif
337                     }
338                 }
339             }
340         }
```

```
341     } while (changed);
342 #undef update_hsum
343 #undef prepare
344 }
345
346
347
348 /*****************************************************************************
349  * Output & Auxilliary procedures                                            *
350  *****************************************************************************/
351
352 void error(char *msg) {
353     message("error: %s\n", msg);
354     exit(4);
355 }
356
357 void print_pop(FILE *out, int n) {
358     evaluation e = eval[n];
359     if (e->var == 0)
360         fprintf(out, "found: ");
361     fprintf(out, "sum=%8.4f var=%8.4f fitness=%8.4f age=%d\n",
362             e->mean, e->var, e->fitness, e->age);
363     fprintf(out, "-->8--\n");
364     print_chromo(out, pop[n]);
365     fprintf(out, "--8<--\n");
366 #ifdef ENCODING_RANDOM
367     printf("ENCODING: age=%d\n", encoding_age);
368     idx_t i,j,k,C; char L,H;
369     Chromo _c; chromo c = &_c;
370     for_each_ij {
371         for (k=0; k<6; k++)
372             c[idx[i][j][k]] = idx[i][j][k];
373     }
374     printf("-->8--\n");
375     print_chromo(out, c);
376     printf("--8<--\n");
377 #endif
378 }
379
380 int best_pop() {
381     int n, b = 0;
382     fitness_t f = eval[b]->fitness;
383     for (n=1; n<P; n++)
384         if (eval[n]->fitness > f)
385             b = n, f = eval[n]->fitness;
386     return b;
387 }
388
389 #define print_summary(o, s, m, v, f) \
390     fprintf(o, s "m=%7.3f v=%5.3f f=%7.3f", m, v, f);
391 #define print_summary_pop(o) \
392     print_summary(o, "pop ", meansum / P, varsum / P, fsum / P)
393
394 void print_status(FILE *out) {
395     fprintf(out, "generation #%d: ", generations);
396     print_summary_pop(out);
397     fprintf(out, "\n");
```

```
398 }
399
400 void ping(int x) {
401     fprintf(stderr, "%d: ", generations);
402     print_summary_pop(stderr);
403     fprintf(stderr, " F=%5.3f/%d", ffsum/P, SELPRESS);
404     fprintf(stderr, "; ");
405     int b = best_pop();
406     print_summary(stderr, "best ",
407             eval[b]->mean, eval[b]->var, eval[b]->fitness);
408     fprintf(stderr, " a=%d", eval[b]->age);
409     fprintf(stderr, " F=%5.3f/%d", ff[b], SELPRESS);
410     fprintf(stderr, "\n");
411 }
412
413 void status(int x) {
414     FILE *out = (x == SIGQUIT) ? stderr : stdout;
415     /* show the best if it wasn't a solution */
416     int b = best_pop();
417     if (eval[b]->var != 0)
418         print_status(out),
419         fprintf(out, "best: "),
420         print_pop(out, b);
421 }
422
423 void ga_info() {
424     printf("jsgmd%d: M=%d A=%d B=%d P=%d K=%d", N, M, A, B, P, K);
425 #ifdef SELPRESS
426     printf(" sp=%d", SELPRESS);
427 #endif
428     printf(" xover="
429 #ifdef XOVER_CUTS
430             "%d", XOVER_CUTS
431 #else
432             "2D/%d", XOVER_2DCUTMOVE
433 #endif
434           );
435 #ifdef MUTRATE
436     printf(" mut=%d", MUTRATE);
437 #endif
438 #ifdef REPLACE_WORST
439     printf(" replace_worst=%d", REPLACE_WORST);
440 #endif
441     printf(" aging");
442 #ifdef OPTAPPLY
443     printf(" optapply=%d", OPTAPPLY);
444 #endif
445 #ifdef OPTMAXIMPROVE
446     printf(" optmax=%f", OPTMAXIMPROVE);
447 #endif
448 #ifdef OPTFULL
449     printf(" optfull");
450 #endif
451 #ifdef EXPLOSION
452     printf(" explosion=%d@%f*%d swap=%d hexspin=%d",
453             EXPLOSION, EXPLOSION_DIFF, EXPLOSION_THRESHOLD,
454             SWAPRATE, HEXSPINRATE);
```

```
#endif
    printf("\n");
}

struct timeval tbegin, tend;
void die(int x) {
    status(x);
    ping(x);
    gettimeofday(&tend, NULL);
    struct rusage usage;
    if (getrusage(RUSAGE_SELF, &usage) >= 0) {
#define tdiff(s, t) \
        (float)(t.tv_sec - s.tv_sec) + (float)(t.tv_usec - s.tv_usec) * 1e-6
#define ptime(t) printf("%fs", (float)t.tv_sec + (float)t.tv_usec * 1e-6)
        printf("execution time:");
        printf(" real %fs", tdiff(tbegin, tend));
        printf(" user "); ptime(usage.ru_utime);
        printf(" sys "); ptime(usage.ru_stime);
        printf("\n");
#undef ptime
#undef tdiff
    }
    exit(x);
}



/*************************************************************************
 * Operators for Genetic Algorithm                                       *
 *************************************************************************/

void xover(chromo p1, chromo p2, chromo o) {
#ifdef XOVER_CUTS
    // FIXME: don't use d.  use (N - i) for more uniform randomness
    idx_t i = 0, j, c = XOVER_CUTS, d = N / XOVER_CUTS;
    chromo p = p1;
    while (c > 0) {
        for (j=i, i+= rnd(d); j<=i; j++)
            o[j] = p[j];
        c--;
        i++;
        p = (p == p1) ? p2 : p1;
    }
    for (; i<N; i++)
        o[i] = p[i];
#else
    idx_t i,j,C, bnd,cut,g;
#define copy_from(p1, kpfx, cut) \
        for (; j<cut; j++) \
            g = idx[i][j/2][kpfx j%2], \
            o[g] = p1[g]
#define copy(kpfx) { \
        bnd = min(cut, 2*C); \
        copy_from(p1, kpfx, bnd); \
        copy_from(p2, kpfx, 2*C); \
        g = idx[i][C-1][kpfx 2]; \
        o[g] = ((cut > 2*C) ? p1 : p2)[g];  \
```

```
512         cut += rnd(2*XOVER_2DCUTMOVE+1) - XOVER_2DCUTMOVE; \
513         cut = min(max(0, cut), 2*C+1); \
514     }
515     cut = rnd(2*A+2);
516     for (C=A,i=0,j=0;   C<=B;   C++,i++,j=0) copy();
517     for (C=B,i=B-A;     C>=A;   C--,i++,j=0) copy(5-);
518 #undef copy
519 #undef copy_from
520 #endif
521 }
522
523 void mutate(chromo c) {
524     idx_t i, r = MUTRATE;
525     for (i=0; r>0; i++) {
526         r--;
527         i += rnd(N - i - r);
528         if (rnd(2))
529             c[i]++;
530         else
531             c[i]--;
532     }
533 }
534
535 void repair(chromo c) {
536     idx_t uses[N+2] = {0},
537           dups[N+2][N] = {0},
538           i,j,k,g;
539     /* count # of uses for each gene values, and remember their indexes */
540     for (i=0; i<N; i++)
541         dups[c[i]][uses[c[i]]++] = i;
542     /* fill consecutively according to dups and usedby */
543     for (g=1, i=0; i<N+2; i++)
544         for (k=uses[i]; k>0; k--) {
545             j = rnd(uses[i]);
546             while (dups[i][j] == N)
547                 j = (j+1) % uses[i];
548             c[dups[i][j]] = g++;
549             dups[i][j] = N;
550         }
551 }
552
553 inline void create(chromo c) {
554     idx_t i;
555     for (i=0; i<N; i++)
556         c[i] = 1+rnd(N);
557     repair(c);
558     exhaustive_opt(c);
559 }
560
561 void explode(chromo c) {
562     int m, n;
563 #ifdef SWAPRATE
564     /* swapping */
565     n = rnd(SWAPRATE) + 1;
566     for (m=0; m<n; m++) {
567         idx_t i = rnd(N),
568               j = (i + rnd(N-1)+1) % N,
```

```
569                 tmp = c[i];
570             c[i] = c[j];
571             c[j] = tmp;
572         }
573     #endif
574     #ifdef HEXSPINRATE
575         /* hexagon spinning */
576         n = rnd(HEXSPINRATE) + 1;
577         idx_t tmp[6];
578         for (m=0; m<n; m++) {
579             idx_t i = rnd(2*(B-A)+1),
580                   j = rnd(B - abs(B-A - i)),
581                   *ij = idx[i][j],
582                   k, d = rnd(5) + 1;
583             for (k=0; k<6; k++)
584                 tmp[(k+d)%6] = c[ij[k]];
585             for (k=0; k<6; k++)
586                 c[ij[k]] = tmp[k];
587         }
588     #endif
589         exhaustive_opt(c);
590     }
591
592
593
594     /*****************************************************************************
595      * Genetic Algorithm                                                         *
596      *****************************************************************************/
597
598     inline void sync_evaluation(int n, evaluation new) {
599         evaluation old = eval[n];
600         /* bookkeep tracking values, e.g. sums, counts and indexes */
601         fsum -= old->fitness, fsum += new->fitness;
602         varsum -= old->var, varsum += new->var;
603         meansum -= old->mean, meansum += new->mean;
604         if (fbest < new->fitness) {
605             fbest = new->fitness;
606             fbestidx = n;
607         } else if (old->fitness == fbest)
608             find_best = 1;
609         if (fworst > new->fitness)
610             fworst = new->fitness;
611         else if (old->fitness == fworst)
612             find_worst = 1;
613     }
614
615     inline void check_solution(int n) {
616         /* check variance */
617         if (eval[n]->var == 0) {
618             if (var0idx == -1 || eval[var0idx]->fitness < eval[n]->fitness)
619                 var0idx = n;
620             print_status(stdout),
621             print_pop(stdout, n);
622             solfound++;
623         }
624     }
625
```

```
626 inline void update_evaluation(int n, evaluation new) {
627     sync_evaluation(n, new);
628     *eval[n] = *new;
629     check_solution(n);
630 }
631 
632 void initialize_population() {
633     Evaluation _e;
634     evaluation e = &_e;
635     int n;
636     for (n=0; n<P; n++) {
637         chromo c = pop[n];
638         e->age = eval[n]->age;
639         evaluate(c, e);
640         update_evaluation(n, e);
641     }
642 }
643 
644 inline void begin_generation() {
645     int n;
646 beginning_of_generation:
647     ffsum = 0;
648 #define ff(f) ((SELPRESS - 1) * (f - fworst) / (fbest - fworst) + 1)
649     for (n=0; n<P; n++) {
650         fitness_t f = ff(eval[n]->fitness)
651             - (fitness_t) eval[n]->age / N * M / P / 10;
652         ffsum += f;
653         ff[n] = f;
654     }
655 #undef ff
656 #ifdef REPLACE_WORST
657     int cmp(const void *a, const void *b) {
658         fitness_t fa = ff[*(int *)a],
659                   fb = ff[*(int *)b];
660         if (fa < fb)
661             return +1;
662         else if (fa > fb)
663             return -1;
664         else
665             return 0;
666     }
667     qsort(rank, P, sizeof(int), cmp);
668 #endif
669 #ifdef EXPLOSION
670     double diff = varsum / P - eval[fbestidx]->var;
671     if ((diff > 0 ? diff : -diff) < EXPLOSION_DIFF) {
672         if (explosion_countdown-- < 0) {
673             int i, r;
674             chromo o;
675             Evaluation _e; evaluation e = &_e;
676             for (i=1; i<P; i++) {
677                 if (rnd(EXPLOSION)) {
678                     r = rank[i];
679                     o = pop[r];
680                     explode(o);
681                     evaluate(o, e);
682                     e->age = 0;
```

```
                    update_evaluation(r, e);
                }
            }
            explosion_countdown = EXPLOSION_THRESHOLD;
            goto beginning_of_generation;
        }
    }
#endif
}

inline int pick() {
    fitness_t s = 0, pt = rnd_f(ffsum);
    int i;
    for (i=0; i<P; i++) {
        s += ff[i];
        if (s > pt)
            return i;
    }
    return rnd(P);
}

inline void generate_offspring(int n) {
    /* select parents */
    int p1, p2;
    p1 = pick();
    do {
        p2 = pick();
    } while (p1 == p2);
    parent1[n] = p1;
    parent2[n] = p2;
    /* and generate offspring */
    chromo o = offspring[n];
    xover(pop[p1], pop[p2], o);
    mutate(o);
    repair(o);
#ifdef OPTAPPLY
    if (rnd(OPTAPPLY))
#endif
        exhaustive_opt(o);
    evaluate(o, offspring_eval[n]);
    offspring_eval[n]->age = 0;
}

inline void generate_offsprings();

inline void replace_population() {
    int n;
    int r = P-1;
    for (n=0; n<K; n++) {
        int p1 = parent1[n],
            p2 = parent2[n],
            p = p1, q = p2;
#ifdef REPLACE_WORST
        q = rank[r--];
        if (!rnd(REPLACE_WORST))
            r--;
#else
```

```
740         if (ff[p1] < ff[p2])
741             p = p2, q = p1;
742 #endif
743         chromo c;
744         c = pop[q], pop[q] = offspring[n], offspring[n] = c;
745         sync_evaluation(q, offspring_eval[n]);
746         evaluation e;
747         e = eval[q], eval[q] = offspring_eval[n], offspring_eval[n] = e;
748         check_solution(q);
749     }
750 }
751
752 inline void end_generation() {
753     int n;
754 #ifdef ENCODING_RANDOM
755     if (encoding_age > P) {
756         idx_t idx2[2*(B-A)+1][B][6];
757         idx_t i,j,C; char L,H,k;
758         for_each_ij
759             for (k=0; k<6; k++)
760                 idx2[i][j][k] = idx[i][j][k];
761         initialize_encoding();
762         Chromo _orig; chromo orig = &_orig;
763         for (n=0; n<P; n++) {
764 #define move(c) \
765             copy_chromosome(orig, c); \
766             for_each_ij \
767                 for (k=0; k<6; k++) \
768                     c[idx[i][j][k]] = orig[idx2[i][j][k]];
769             move(pop[n]);
770 #undef move
771         }
772         initialize_population();
773         ga_info();
774         encoding_age = 0;
775     }
776 #endif
777
778     int x;
779 #define find(what, cmp) { \
780             x = 0, what = eval[0]->fitness; \
781             for (n=1; n<P; n++) \
782                 if (eval[n]->fitness cmp what) \
783                     x = n, what = eval[n]->fitness; \
784         }
785     if (find_best) {
786         find(fbest, >);
787         fbestidx = x;
788     }
789     if (find_worst)
790         find(fworst, <);
791 #undef find
792
793     for (n=0; n<P; n++)
794         eval[n]->age++;
795
796     gbest_age = (gbest != fbest) ? 1 : gbest_age + 1;
```

```
797     gbest = fbest;
798     eval[fbestidx]->age = gbest_age;
799 
800     generations++;
801 }
802 
803 void ga() {
804     while (! DONE) {
805         begin_generation();
806         generate_offsprings();
807         replace_population();
808         end_generation();
809     }
810 }
811 
812 #ifndef SINGLETHREAD
813 #include <pthread.h>
814 
815 #ifndef NUMTHREADS
816 #define NUMTHREADS 3
817 #endif
818 
819 pthread_mutex_t generated_mtx = PTHREAD_MUTEX_INITIALIZER;
820 pthread_cond_t generate_new_cond = PTHREAD_COND_INITIALIZER,
821                generation_ready_cond = PTHREAD_COND_INITIALIZER;
822 int generated = K;
823 int working = 0;
824 
825 void *offspring_generator(void *p) {
826     int n;
827     pthread_mutex_lock(&generated_mtx);
828     for (;;) {
829         /* wait until a new generation begins */
830         working++;
831         while (generated < K) {
832             /* assign an offspring # */
833             n = generated++;
834             pthread_mutex_unlock(&generated_mtx);
835             /* work in parallel */
836             generate_offspring(n);
837             pthread_mutex_lock(&generated_mtx);
838         }
839         working--;
840         /* signal new offsprings are all ready, if i'm the last worker */
841         if (working == 0)
842             pthread_cond_signal(&generation_ready_cond);
843         pthread_cond_wait(&generate_new_cond, &generated_mtx);
844     }
845     pthread_mutex_unlock(&generated_mtx);
846     return NULL;
847 }
848 
849 inline void generate_offsprings() {
850     generated = 0;
851     pthread_cond_broadcast(&generate_new_cond);
852     pthread_cond_wait(&generation_ready_cond, &generated_mtx);
853 }
```

```
854
855 inline void GA() {
856     int n;
857     pthread_t threads[NUMTHREADS];
858     pthread_mutex_trylock(&generated_mtx);
859     for (n=0; n<NUMTHREADS; n++)
860         pthread_create(&threads[n], NULL, offspring_generator, NULL);
861     debug("threads ready\n");
862     ga();
863     pthread_mutex_unlock(&generated_mtx);
864 }
865
866 #else
867 inline void generate_offsprings() {
868     int n;
869     for (n=0; n<K; n++)
870         generate_offspring(n);
871 }
872
873 #define GA() ga()
874
875 #endif
876
877
878
879
880 /*****************************************************************************
881  * and finally, the main()                                                   *
882  *****************************************************************************/
883
884 int main(int argc, char *argv[]) {
885     srand((int)argv);
886     gettimeofday(&tbegin, NULL);
887
888     int n;
889     for (n=0; n<P; n++) {
890         chromo c = new_chromosome();
891         create(c);
892         pop[n] = c;
893         rank[n] = n;
894     }
895     for (n=0; n<K; n++) offspring[n] = new_chromosome();
896     for (n=0; n<P; n++) {
897         evaluation e = new_evaluation();
898         e->age = 0;
899         eval[n] = e;
900     }
901     for (n=0; n<K; n++) offspring_eval[n] = new_evaluation();
902     initialize_encoding();
903     initialize_population();
904     debug("memory, data ready\n");
905
906 #ifdef T
907     alarm(T);
908     signal(SIGALRM, die);
909 #endif
910     signal(SIGHUP,  die);
```

```
911       signal(SIGTERM, die);
912       signal(SIGINT,  ping);
913       signal(SIGQUIT, status);
914       debug("signal handlers ready\n");
915
916       message("starting jsgmd%d\n", N);
917       ga_info();
918       ping(0);
919       GA();
920       die(solfound == 0);
921       return -1;
922   }
```