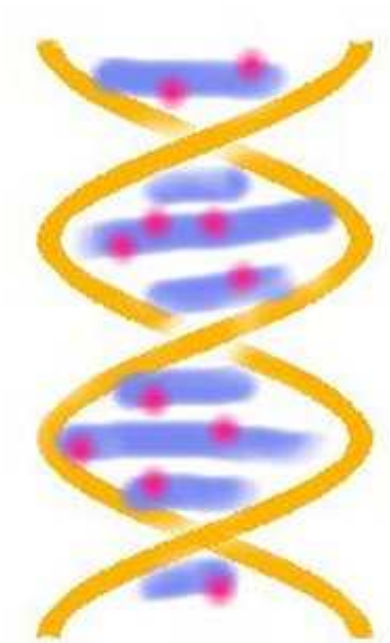
진화연산 알고리듬 툴박스

Genetic Algorithm Toolbox



사용자 지침서

목 차

제 1 장 개요(Introduction)

진화연산 알고리즘은 자연계의 생명체 중 환경에 잘 적응한 개체가 좀더 많은 자손을 남길 수 있다는 자연선택 과정과 자연계의 생명체의 설계도와 같은 유전자의 변화를 통해서 좋은 방향으로 발전해 나간다는 자연진화의 과정을 모방하여 컴퓨터로 모의 수행을 하는 최적화 알고리즘의 하나이다. 즉, 실세계의 문제를 풀기 위해 잠재적인 해들을 컴퓨터 상에서 코딩된 개체로 나타내고, 여러 개의 개체들을 모아 개체군을 형성한 뒤, 세대를 거듭하면서 이들의 유전 정보를 서로 교환하거나 새로운 유전 정보를 부여하면서 적자 생존의 법칙에 따라 모의 진화를 시킴으로써, 주어진 문제에 대한 최적의 해를 찾는 계산 모델이다.

진화연산 알고리즘은 Michigan 대학의 John Holland에 의해서 개발되었는데, 이것은 이진수의 형태로 표현된 유전자와 적합도들로 이루어진 개체들의 집합을 이용해서 최적화 과정을 수행하는 알고리즘이다. 기존의 최적화 알고리즘은 최적화 하고자 하는 목적 함수를 미분해서 탐색을 수행하는 반면, 진화연산 알고리즘은 선택 연산자, 교차 연산자, 돌연변이 연산자와 적합도를 이용해서 탐색을 수행한다. 이러한 방법은 목적 함수의 미분과정이나 특별한 수학적 연산을 필요로 하지 않는다. 또한, 진화연산 알고리즘은 점에 의한 탐색이 아니라 개체들이 모여 이루는 개체군에 의한 병렬적인 탐색이라는 점에서 기존의 최적화 알고리즘과 다르다. 한 세대의 개체군에 속한 개체들은 진화를 거듭하면서, 이전 세대까지 축적된 정보를 서로 교환하고 새로운 영역으로의 탐색을 시도한다. 탐색의 방향이나 영역이 초기값에 의해서 결정되지 않고 세대마다 확률적으로 결정되므로 지역 최소점에 빠질 가능성이 적어 전역 최적화가 가능한 알고리즘으로 알려져 있다.

제 2 장 기본 구성요소(Basic Components)

진화연산 알고리즘은 자연계의 진화 현상을 모방하여 컴퓨터 상에서 실험하는 방법을 사용해서 주어진 문제에 대한 최적의 해를 이끌어내는 알고리즘이다. 따라서 진화연산 알고리즘의 가장 기본이 되는 구성요소는 자연계 진화의 기본 요소와 매우 비슷하다.

먼저 풀고자하는 문제 즉 환경이 존재해야 한다. 환경이란 그 자체로서의 의미보다는 그 환경 속에 살고 있는 개체가 얼마나 잘 적응해서 살고 있는지, 얼마나 적합한지를 나타내는 값을 정하는 역할을 주로 한다. 이러한 값을 진화연산 알고리즘에서는 적합도라고 한다. 또 풀고자하는 문제를 정의하는 진화연산 알고리즘의 환경을 적합도 함수라고 한다.

다음으로, 앞서 정의한 환경 속에서 살고 있는 개체를 생각할 수 있는데, 적합도 함수의 관점에서 보면 함수의 가능한 해라고 생각할 수 있다. 각 개체들은 염색체의 형태로 표현되어 있어야 하는데, 진화연산 알고리즘에는 염색체의 표현을 이진수 문자열의 형태를 빌어서 나타낸다. 그밖에 다른 진화 알고리즘인 진화 전략이나 진화 프로그래밍에서는 실수 벡터, 혹은 트리(tree)의 형태를 사용하기도 한다. 개체를 염색체의 형태로 바꾸는 것을 인코딩(encoding)이라고 하는데, 보통 염색체의 인코딩 방법에 따라 적용시킬 수 있는 진화 연산자들이 다르고 문제를 풀어내는 능력이나 진화의 양상도 크게 달라지므로 해를 구하는데 있어서 인코딩 문제는 매우 중요한 부분을 차지한다.

염색체의 형태로 나타내어진 개체들은 진화연산 알고리즘이 적용될 수 있도록 개체들의 군을 이루는데 이를 개체군이라 부른다. 개체군은 진화연산 알고리즘이 다른 탐색 알고리즘과 가장 크게 구별되는 부분이다. 다른 탐색 알고리즘이 점에 의한 탐색이라고 말한다면 진화연산 알고리즘은 군에 의한 즉 개체군에 의한 탐색이라고 말할 수 있다. 즉, 개체군에 속하는 잠재적인 해를 나타내는 많은 개체들이 진화연산 알고리즘에서 사용하는 진화 연산자들을 사용해서 서로의 유전 정보를 교환하고 때로는 새로운 유전 정보를 획득함으로써 효율적으로 전역 최적점(global optimum)을 향해 탐색해 나갈 수 있는 것이다.

위에서 잠시 언급한 진화 연산자는 진화연산 알고리즘이 전역 최적점(global optimum)을 향해서 탐색을 해 나갈 수 있게 해주는 도구이다. 다시 말하면, 염색체의 형태로 인코딩된 개체의 유전자를 조작하는 연산자를

적용시킴으로써 진화연산 알고리즘이 개체군이 제시한 유익한 정보를 바탕으로 해공간의 다른 부분을 탐색할 수 있게 해준다. 많이 사용하는 진화연산자는 개체군들 중 유익한 정보를 가지고 있는 개체들을 선별하는 선택연산자, 두 개체의 유전 정보를 교환하기 위한 교차 연산자와 개체군 내의 모든 개체들에 존재하지 않는 새로운 유전 정보를 부여하기 위한 돌연변이 연산자가 있다. 이들 연산자는 확률이라는 정보와 각 개체들의 환경에 대한 적합도를 바탕으로 적용된다.

그밖에 생각할 수 있는 구성요소는 언제 진화연산 알고리즘을 끝마치겠는가를 결정하는 종료조건을 들 수 있다. 진화의 과정을 살펴보면 정체되는 시기와 진화가 급격히 이루어지는 시기가 있다. 보통의 시간은 정체기에 해당하지만, 이러한 시간을 거쳐서 축적된 변이의 가능성이 특정시기에 폭발해서 급격한 진화가 이루어진다. 이러한 현상은 진화연산 알고리즘에서도 쉽게 관찰할 수 있는데 종료조건을 너무 엄격하게 만들면 충분한 진화 과정을 볼 수 없게 되는 결과를 낳게 되고, 좀 느슨하게 만들면 시간이 많이 걸리는 문제가 생긴다. 따라서 적당한 종료조건을 만드는 것도 매우 중요한 문제이다.

제 3 장 개체의 표현
(Representation of Individual)

모든 진화연산 알고리즘에서는 문제의 가능한 해를 직접적으로 탐색하지 않고 염색체의 형태로 바꾸는 인코딩 과정을 거쳐 코딩된 개체에 대해서 진화 연산자를 적용시킴으로써 탐색을 수행한다. 인코딩에 의해서 해를 나타내므로 주어진 문제를 어떤 식으로 코딩하느냐에 따라서, 해공간의 형태가 바뀌게 되고, 적용할 수 있는 진화 연산자의 종류가 달라지게 되며, 진화 연산자에 의해서 만들어진 탐색 영역도 달라지게 된다. 따라서 효율적인 탐색을 위해서는 개체의 표현 방법이 매우 중요하다.

진화연산 알고리즘에서는 목적 함수의 변수들을 특정한 범위의 실수 값으로 사상할 수 있는 고정된 길이의 이진 문자열로 염색체를 표현한다. 그리고, 진화 전략이나 진화 프로그래밍에서는 실수 값을 가지는 벡터 혹은 트리(tree)의 형태를 띤 염색체를 사용하여 목적 함수의 변수 즉 개체를 나타낸다. 또, 개체의 염색체에 목적 함수의 변수뿐만 아니라 적응시키고자 하는 진화연산 알고리즘 자체의 매개 변수를 포함시킬 수도 있다. 실수 값을 직접 사용하는 경우는 변수들이 취할 수 있는 값의 범위가 제한되지 않고 연속적인 값을 가질 수 있는 장점이 있으나 진화 연산자의 정의가 다소 복잡해질 수 있다.

이진 문자열이나 실수 값의 벡터 외에 문제에 따라서는 행렬 혹은 퍼지 논리의 소속함수나 신경 회로망과 같이 복잡한 자료구조 자체가 개체를 의미할 수도 있다. 이 같은 경우는 코딩과 디코딩 과정이 간단해지지만 자료구조에 따른 새로운 진화 연산자를 정의하여야 하고, 진화연산 알고리즘의 수렴성이나 견실성 등을 보장하기가 힘들어진다. 따라서, 이와 같은 자료구조를 직접 사용하지 않고 이진 문자열이나 실수의 벡터로 코딩하는 과정을 거친 염색체의 형태에 대하여 진화 연산자를 적용시키고 적합도를 산정하려면, 다시 염색체 형태의 개체로부터 원래의 자료 구조를 구성하는 과정을 거쳐야 한다. 본 툴박스에서는 복잡한 자료구조를 사용하지는 않고 일반적으로 사용되고 있는 이진 문자열의 형태와 실수의 형태를 사용해서 염색체를 구성하였다.

이진 문자열의 형태를 취하는 개체의 유전자는 다음과 같이 표현된다. 각 개체의 앞부분은 유전자 코드에 해당되며 마지막 열에 적합도를 덧붙인

다. 이러한 개체들을 모아서 개체군(Population)을 만들어 낸 후 진화연산 알고리즘을 적용시키게 된다.

```
                유전자코드      적합도
Gene1=[ 1 0 1 1 0 1 ,  1.56 ];
Gene2=[ 1 1 0 0 0 0 ,  2.58 ];
Gene3=[ 0 0 1 1 1 0 ,  1.97 ];
                :
                :
Population=[ Gene1; Gene2; Gene3; ····];
```

실수의 형태를 취하는 경우는 아래와 같이 표현된다.

```
Gene1=[ 12  123   56 ,  2.67];
Gene2=[111   96   11 ,  12.5];
Gene3=[ 4    36  100 ,  5.66];
                :
                :
Population=[Gene1; Gene2; Gene3; ...];
```

실수의 형태를 취하는 유전자에 진화연산 알고리즘을 적용시키기 위해서는 이진 문자열과는 다른 진화 연산자가 필요하다.

제 4 장 적합도(Fitness)

진화연산 알고리즘에서는 개체의 성능을 다른 개체와 비교하기 위하여, 혹은 개체가 얼마나 진화연산 알고리즘이 적용되고 있는 가상의 환경에 잘 적응하고 있는지를 나타내는 척도로서 적합도(fitness)를 모든 개체에 부여한다. 적합도를 산정하는 적합도 함수는 진화연산 알고리즘이 적용되고 있는 가상의 환경이라고 할 수 있으며, 최적화 하고자 하는 목적 함수에 기반하여 만들어지지만 제약 조건이나 다른 주관적 평가 함수를 포함할 수도 있다. 즉, 적합도의 산정이 문제의 표현에 따라서 개체의 전체 성능에 기반할 수도 있고, 단지 부분적인 산정이 될 수도 있으며 경우에 따라서는 정성적인 지식까지 포함될 수 있다.

목적 함수에 의하여 계산된 적합도는 우수한 개체가 큰 값을 가지도록 하기 위해서 재조정할 수 있다. 목적 함수를 최대로 만드는 해를 찾는 문제에서는 목적 함수의 값을 그대로 적합도로 이용하여도 되지만, 목적 함수를 최소화로 만드는 해를 찾는 문제에서는 우수한 해일수록 작은 값의 목적 함수를 가지므로 적합도는 목적 함수에 의해서 계산된 값에 역수를 취하거나 부호를 바꾸는 등의 작업이 필요하다.

진화연산 알고리즘마다 택하고 있는 선택 연산자에 따라서 적합도가 의미 있는 수치가 되도록 목적 함수의 값을 조정하는 경우가 많다. 진화 전략이나 진화 프로그래밍에서는 선택 연산자의 정의에 따라서 이와 같은 재조정을 필요로 하지 않기 때문에 목적 함수의 값을 그대로 적합도로 사용하지만, 유전자 알고리즘의 경우는 적합도가 선택 연산자를 적용시키기 위한 확률로 사용되는 경우가 많으므로 보통 목적 함수의 값을 다른 값으로 사상시킨다. 이러한 작업을 적합도 조정(fitness scaling)이라고 한다. 그 중에서 선형 적합도 조정은 다음 식과 같이 수행할 수 있다.

$$f_{new} = a \cdot f + b$$

(a, b는 조정 상수)

```
CEMTool>> f=[2:5:0.1];
CEMTool>> a=0.7;       // 조정상수(scale constant)
CEMTool>> b=0;         // 조정상수(scale constant)
CEMTool>> newf=a*f+b;
```

```
CEMTool>> plot(f,newf)
CEMTool>> mean(f)
            3.5000
CEMTool>> mean(newf)
            2.4500
```

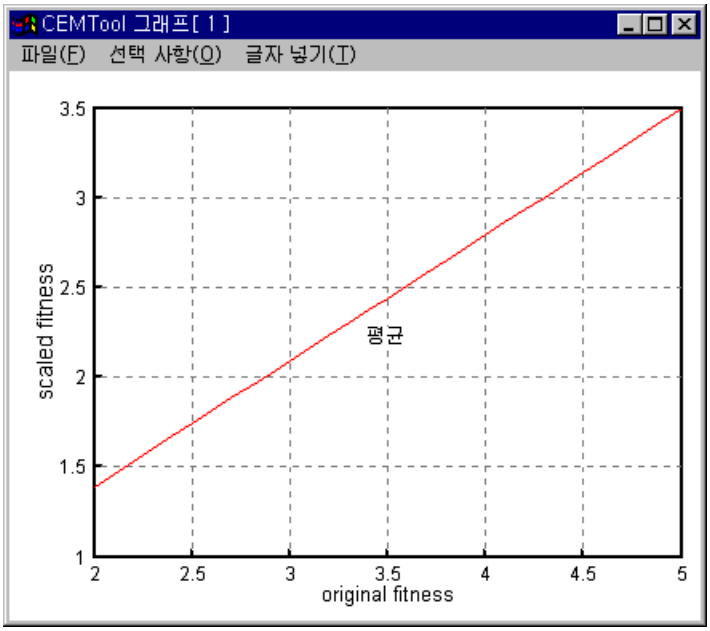


그림 1. 조정상수를 이용한 선형 적합도 조정

선형 적합도 조정 이외에도 개체들의 순위에 의해서 개체들의 적합도를 정해주는 방법이 있다. 그 중에서 Baker linear ranking을 예로 들어보자. Baker linear ranking 방법에서는 아래 식을 바탕으로 순위에 따라서 새롭게 적합도를 할당한다. 전체 적합도의 합은 1이어서 적합도 값을 곧바로 확률로 사용할 수 있다.

$$NewFit(i=\text{순위}) = \frac{1}{N}\left(\eta_{max} - (\eta_{max} - \eta_{min}) \times \frac{i-1}{N-1}\right)$$

Baker linear ranking은 “bak_rank” 함수로 구현되어 있다. 다음은 이 함수의 실행 예이다.

```
CEMTool>> f=[1:100:1]';          // 적합도의 순위를 1에서 100까지 만듦
CEMTool>> opt=1.5;
CEMTool>> nf=bak_rank(f,opt); // Baker linear ranking 적용
CEMTool>> plot(f,nf)
```

적합도의 순위가 어떤 식으로 바뀌는 지를 그래프로 도시하면 그림 2와 같다.

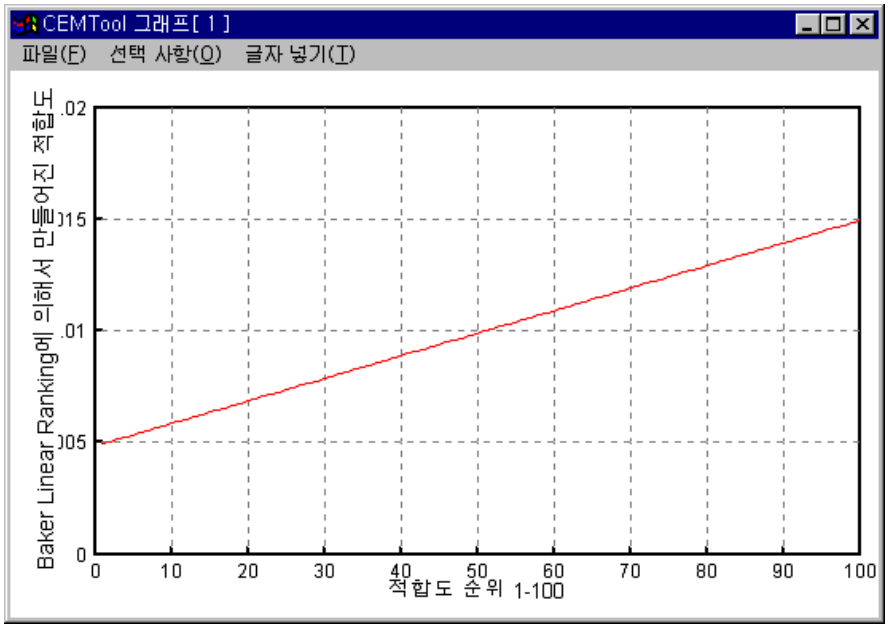


그림 2. Baker linear ranking에 의해서 새롭게 할당된 적합도, x축: 적합도 순위(1위-100위) y축: 새롭게 할당된 적합도

본 툴박스에서는 Baker linear ranking 이외에 simple linear ranking, Whitley linear ranking, uniform linear ranking 방법을 지원한다. simple linear ranking은 “s_rank", Whitley linear ranking은 ”whit_rank", uniform linear ranking은 "uni_rank" 함수로 구현되어 있다.

제 5 장 진화 연산자(Genetic Operators)

5.1 선택 연산자(selection operator)

자연계의 생물체는 적자 생존의 법칙에 따른 자연 선택을 통하여 세대가 지날수록 환경에 적합한 개체들이 많이 살아남게 된다. 따라서 선택 연산자는 진화연산 알고리즘이 이와 같은 자연 선택에 기반하여 개체군을 이용하여 탐색을 시도하는데 있어서 매우 중요한 연산자이다. 즉, 선택을 통해서, 적합도에 근거하여 우수한 개체들이 다음 세대에 더 많은 자손을 남기도록 하고 개체군의 적합도의 평균이 세대를 거듭하면서 증가하도록 하여 최적점으로 접근해 갈 수 있게 되기 때문이다. 하지만, 진화연산 알고리즘에서는 자연계의 자연 선택을 직접 구현하지는 못하므로, 목적 함수를 이용하여 구한 적합도를 바탕으로 하여 명시적인 선택기준을 나타내는 선택 연산자를 정의하게 된다.

진화연산 알고리즘에서는 선택을 통하서 개체군을 변형시켜나가는 과정에서 선택 강제정도(selection pressure)를 정의할 수 있다. 이것은 정량적인 수치로 표현되는 것이 아닌 상대적인 정도를 나타낸다. 선택 강제 정도가 높으면 우수한 개체가 다음 세대에 살아남을 가능성을 많이 주고, 선택 강제 정도가 낮으면 열등한 개체도 바로 절멸하지 않고 존재할 수 있는 기회를 준다. 문제에 따라서는 이 선택 강제 정도를 정하는 것이 중요한데, 만약 해공간이 단일 모드(single mode)이면 선택 강제 정도가 비교적 강해도 전역 최적점을 찾기가 어렵지 않지만, 해공간이 다중 모드(multi mode)일 때는 선택 강제 정도를 높게 할 경우 전역 최적점으로의 수렴이 이루어지지 않고 지역 최소점으로 수렴하는 경우가 있다. 이는 개체군이 작을 경우 적합도가 작은 개체가 빨리 사라지게 되면 개체군 내에 모든 해 공간을 탐색할 수 있는 다양한 유전 형질을 가지는 개체가 존재하지 않게 되어 더 이상의 잠재된 해 공간을 탐색하기가 어려워지기 때문이다. 즉, 선택 강제 정도를 높일 경우 적합도가 상대적으로 조금 높은 지역 최소점에 해당하는 개체들이 개체군을 장악하여 진화연산 알고리즘의 모든 개체들이 지역 최소점으로 조기 수렴해 버릴 가능성이 높다.

조기 수렴 현상을 배제하기 위하여 4장에서 설명한 적합도 조정(fitness scaling), 적합도 순위 재설정 방법(linear ranking method)을 사용하여 적합도를 새롭게 할당하는 방법을 사용할 수 있다. 이 방법들로 적합도가 아

주 낮은 개체의 선택 확률을 높여주고, 적합도가 아주 높은 개체의 선택 확률을 낮춰줄 수 있다.

선택 연산자의 종류에는 비례 선택법(proportionate selection or simple selection), 토너먼트 선택법(tournament selection), 지역 선택법(local selection), 절단 선택법(truncation selection), 균등 선택법(stochastic universal selection)이 있고 본 툴박스에서 각각을 함수로 구현하였다. 구현된 함수의 이름은 각각 “s_sel", "tour_sel", "loc_sel", "trun_sel", "stoc_sel"이다. 이러한 선택 연산자들은 적합도의 할당법과 함께 다양하게 변형하여 사용될 수 있다.

비례 선택법(proportionate selection or simple selection)은 개체의 적합도에 비례하여 다음 세대의 개체군에 존재할 자손의 수를 결정하는 것이 기본 원칙이다. 가장 기본이 되는 방법으로 전체 적합도의 합에 대한 각각의 적합도의 비를 선택의 확률로 사용한다. 적합도가 높을수록 선택될 확률이 높아지지만 적합도가 가장 낮은 개체는 선택될 확률이 거의 0에 가깝다. 비례 선택법에서 각 개체가 선택될 확률을 살펴보자.

```
CEMTool>>f=[1 2 3 4]; // 적합도를 임의로 할당
CEMTool>>sumf=sum(f); // 적합도의 전체 합으로 각각의 적합도 나눔
CEMTool>>sumf
            10

CEMTool>>newf=f/sumf
     newf  =
            0.1000     0.2000     0.3000     0.4000
```

가장 적합도가 낮은 개체가 선택될 확률은 0.1, 가장 적합도가 높은 개체가 선택될 확률은 0.4로 결정되었다. 이런 식으로 개체 각각의 선택 확률을 정하고, 임의로 발생시킨 난수를 통해 하나의 개체를 선택하는 방법으로 개체군의 크기만큼 개체를 선택한다. 이러한 방법은 가장 일반적으로 적용되는 방식이며, 아무리 적합도가 낮은 개체라도 경우에 따라서는 하나 이상의 개체를 복제할 수가 있게 되어 선택 강제 정도는 상대적으로 낮아지지만, 적합도가 높은 개체일수록 선택 확률이 크기 때문에 선택될 횟수가 많아져서 선택 연산자의 원래 목적에 어긋나지 않는다.

토너먼트 선택법(tournament selection)은 현재의 개체군에서 무작위로 k

개의 개체를 선택한 후 그 중에서 가장 적합도가 높은 개체를 선택하는 방법으로, 개체군이 다 채워질 때까지 계속 같은 시행을 반복한다. k가 클수록 선택 강도는 커지게 된다.

지역 선택법(local selection)은 특정 개체 주변의 것들만 선택에 참여시키는 방법으로, 개체 사이의 거리가 d 이하인 것들 중 적합도가 높은 것을 선택하는데, 이 작업을 개체군이 다 채워질 때까지 반복한다.

그밖에 절단 선택법(truncation selection)은 적합도가 특정한 값 이상인 것들만 선택 연산자를 적용시키는 방법이고, 균등 선택법(stochastic universal sampling)은 모든 개체를 동일한 확률로 선택하는 방법이다.

다음으로는 선택을 통하여 얼마나 많은 개체를 다음 세대에 교체시킬 것인가 하는 문제를 살펴보자. 이에 대해 각 진화연산 알고리즘마다 취하고 있는 방법이 약간씩 다르다. 이진수를 사용하는 유전자 알고리즘에서는 보통 부모 세대의 개체군이 자손에 의하여 세대마다 완전히 교체되는 비중복(non-overlapping) 선택을 하지만, 세대 간격(generation gap)을 정해 놓고 세대 간격만큼 부모와 자식 개체를 중복시킬 수도 있다. 본 툴박스에서는 비중복 선택법을 통해서 진화연산 알고리즘의 선택 연산자를 구현하였다.

비중복(non-overlapping) 선택을 사용하지만, 단 하나의 개체만은 가장 적합도가 높은 개체로 유지시키는 선택법이 있는데 이를 엘리트 기법(elitest method)이라 한다. 이러한 기법은 현재까지 발견된 가장 우수한 개체를 확률적인 선택 혹은 진화 연산자에 의해서 희생되는 것을 막을 수 있다.

5.2 교차 연산자 (crossover operator)

교차 연산자는 유전자 형태로 표현된 해를 전역 최적점으로 수렴하도록 하기 위해 적용하는 중요한 진화 연산자이다. 개체군에 존재하는 개체들이 가지고 있는 정보를 교차 연산자를 통하여 서로 교환함으로써 해공간에서의 새로운 영역으로 탐색을 시도하게 된다. 일반적으로 교차는 개체 사이에서 유전자의 일부분을 맞교환함으로써 이루어지는데 코딩된 유전자의 형태에 따라서 다양한 교차 연산자가 정의될 수 있다.

이진 문자열의 형태로 유전자가 코딩이 되는 유전자 알고리즘의 경우에서는 가장 쉽게 생각할 수 있는 교차 연산자가 1-point 교차 연산자(simple crossover)로서 다음과 같이 동작한다.

```
CEMTool>> m1=[1 0 0 1 1 0 1];
CEMTool>> m2=[0 0 1 1 0 1 1];
CEMTool>> [c1,c2]=s_xover(m1,m2,0.7)

     c1  =
           0     0     1     1  |  1     0     1

     c2  =
           1     0     0     1  |  0     1     1
```

교차 위치

1-point 교차 연산자는 임의의 한 지점을 선택하여 그 이후의 유전자를 다른 개체와 맞바꾸는 형식으로 동작한다. 이와 비슷한 방법으로 2-point 교차 연산자가 있다. 이는 임의의 두 지점을 선택하여 그 사이의 유전자를 다른 개체와 교환하는 것으로 정의할 수 있다. 또한, 좀 더 일반적으로 n-point 교차 연산자를 정의할 수 있으며 이는 n 개의 교환 지점을 선택하고 그 사이의 유전자를 상대방의 해당 유전자와 바꾸는 것이다. 개체군이 작을수록 다양한 유전 정보의 존재가 필요하므로, n이 커질수록 한정된 정보량을 극복하는데 도움이 된다. 이 연산자는 "m_xover" 함수로 구현되어 있다.

그 외에 균일 교차 연산자(uniform crossover)라는 것이 있는데 이는 각각의 비트에 대하여 두 개의 부모 개체로부터 임의의 값을 가져오는 것으로 부모 개체의 정보에 대해 좀 더 파괴적인 결과를 가져오게 된다. 개체들의 유전자를 사용해서 어느 정도 진화연산 알고리즘을 적용시키게 되면, 높은 적합도를 보이는 각각의 개체들은 유용한 유전 정보에 대해서 개체 자신만의 군집(building block)을 만들게 되는데, 이런 정보가 진화 연산자를 통해서 파괴되지 않고 다음 세대에 전달되어야한다. 그러나 균일 교차 연산자의 경우는 각각의 비트에 대해서 임의로 교차를 시행하기 때문에 이러한 유용한 정보가 파괴될 확률이 높다. 하지만, 각각의 비트의 정보가 독립적이고 중요한 형태로 코딩된 유전자의 경우 효율적일 수도 있다.

실수의 형태로 코딩된 유전자에게 적용되는 교차 연산자에 대해서 알아보자. 실수의 형태를 가진 유전자에서는 교차를 재결합(recombination)으로

표현하는데 이산 재결합(discrete recombination), 중간 재결합(intermediate recombination), 선 재결합(line recombination) 등의 방법이 있다. 이중 중간 재결합(intermediate recombination)에 대해서 살펴보자. 이 방법에서는 각각의 자식(child) 개체들이 다음 식을 통해서 만들어진다.

$$Child1 = Mate1 + \alpha_1 \times (Mate2 - Mate1)$$
$$Child2 = Mate1 + \alpha_2 \times (Mate2 - Mate1)$$

여기서 α_1, α_2는 비례 상수(scaling factor)로서 미리 정해진 범위(Range)에 의해 [-Range 1+Range]까지의 범위에서 균일하게 분포하는 임의의 수로 선택한다.

```
CEMTool>> m1=[12 25 5];
CEMTool>> m2=[123 4 34];
CEMTool>> opt=[0.5 0.5];   //Range와 재결합이 일어날 확률을 정의한다.
CEMTool>> [c1 c2]=int_re(m1,m2,opt)
     c1  =
            81.7853    33.7568     0.0796
     c2  =
           137.5377     6.7074    34.8350
```

그밖에 이산 재결합 방법은 “dis_re", 중간 재결합 방법은 ”int_re", 선 재결합 방법은 “line_re" 함수로 구현되어 있다.

5.3 돌연변이 연산자 (mutation operator)

교차 연산자에 의해서 개체들 사이에 유전 정보가 서로 교환되지만 모든 해공간을 탐색하기 위한 유전 정보가 현재의 개체군내의 개체들에 들어 있지 않다면 교차 연산자를 아무리 적용시키더라도 더 이상의 탐색이 이루어질 수 없다. 따라서 새로운 유전 형질을 부여할 수 있는 돌연변이 연산자는 필수적이며 어떤 진화연산 알고리즘에서는 교차 연산자 없이 돌연변이 연산자만으로 탐색을 시도하기도 한다.

유전자 알고리즘에서는 보통 하나의 비트를 주어진 확률에 따라서 다른 값으로 바꾸는 것으로 돌연변이 연산자를 정의한다. 가장 간단한 형태의 돌연변이 연산자는 “s_mut” 함수로 구현되어 있으며, 다음과 같이 돌연변

이 확률과 시행회수의 두 가지 변수에 의해 동작한다.

```
CEMTool>> m=[1 0 1 0 ];
CEMTool>> opt=[0.5 1]; // 돌연변이 확률 0.5, 시행회수 1
CEMTool>> c=s_mut(m,opt)
     c  =
           1         1         1         0
                     ↑
          돌연변이 연산이 일어난 지점
```

제 6 장 알고리즘의 흐름(Procedures)

진화연산 알고리즘의 흐름은 알고리즘의 종류에 따라서 매우 다양하게 정의할 수 있다. 여기서는 가장 기본이 되는 Holland와 Goldberg가 사용한 진화연산 알고리즘으로 전체 흐름을 알아보겠다.

① 개체군을 초기화시킨다.
② 개체들 각각의 적합도를 계산한다.
③ 적합도 할당 전략에 따라 적합도를 다시 정해준다(선택사항).
④ 적합도에 따라서 선택 연산자(selection operator)를 적용한다.
⑤ 선택 연산자를 통해서 선택된 개체들에게 교차 연산자를 적용한다.
⑥ 개체들에게 돌연변이 연산자를 적용한다.
⑦ 개체들 각각의 적합도를 계산한다.
⑧ 종료조건과 비교해서 일치하면 진화연산 알고리즘을 끝내고, 일치하지 않으면 단계 3으로 간다.

셈툴 진화연산 알고리즘 툴박스의 함수들을 이용해서 위의 과정을 수행해 보자. 먼저 단계 1의 개체군 초기화를 수행한다.

```
CEMTool>> [Pop ChromLen]=init_sga(PopSize,Bounds,"fdemo1",Options);
```

툴박스에서는 “init_sga”라는 함수를 이용해서 개체군을 초기화시키는데, 전체 개체들의 크기(PopSize), 탐색범위(Bounds), 적합도함수, 옵션을 이용해서 개체를 생성시킨다. 옵션에는 정밀도가 지정된다. 이제 생성된 개체군을 최대 진화 세대수(MaxGen)만큼 진화시킨다. 루프는 다음 문장을 통해 구성할 수 있다.

```
for (Generation=1; Generation<=MaxGen; Generation=Generation+1)
{
```

루프 내에서는 우선 각 개체의 적합도를 계산한다. 여기서는 가장 간단한 형태의 진화연산 알고리즘을 적용하고, 적합도를 특정 규칙에 따라서 새로이 할당하지는 않는다.

```
[x v]=FitFunc(x,Options);
```

계산된 적합도를 바탕으로 진화 연산자를 이용하여 개체군의 크기만큼 새로운 개체들을 생성시킨다. 이는 위의 단계 4, 5, 6에 해당되며 다음의 루프 내에서 수행된다.

```
while (NewPopNum<=PopSize)
{
```

개체군들의 적합도를 바탕으로 2개의 개체를 선택한다.

```
  [Mate1, Mate2]=s_sel(Pop(:,TotalLen),[]);
```

선택된 두 개체의 정보를 교차 연산자를 사용해서 상호 교환한다.
이 과정을 통하여 새로운 2개의 개체(Child1, Child2)가 생성된다.

```
  [Child1,Child2]=s_xover(Pop(Mate1,:),Pop(Mate2,:),XoverOpt);
}
```

교차 연산으로 새롭게 생성된 개체들에 돌연변이 연산자를 적용해서 개체들에게 들어있지 않는 새로운 유전정보를 만들어 낸다.

```
for (cnt=1;cnt<=PopSize;cnt=cnt+1)
  Pop(cnt,:)=s_mut(Pop(cnt,:),MutOpt);
}
```

알고리즘의 종료조건인 최대 진화 세대수를 만족하게 되면 진화연산 알고리즘을 통해서 만들어진 가장 훌륭한 개체를 선택한다.

```
BestIndi=sel_best(Pop,[])
```

이 과정을 모두 묶으면 가장 간단한 형태의 진화연산 알고리즘을 얻는다.

```
[Pop ChromLen]=init_sga(PopSize,Bounds,"fdemo1",Options);
for (Generation=1;Generation<=MaxGen;Generation=Generation+1)
```

```
{
   [x v]=FitFunc(x,Options);
   while (NewPopNum<=PopSize)
   {
      [Mate1, Mate2]=s_sel(Pop(:,TotalLen),[]);
      [Child1,Child2]=s_xover(Pop(Mate1,:),Pop(Mate2,:),XoverOpt);
   }
   for (cnt=1;cnt<=PopSize;cnt=cnt+1)
      Pop(cnt,:)=   s_mut(Pop(cnt,:),MutOpt);
}
BestIndi=sel_best(Pop,[])
```

제 7 장 예제 (Examples)

7.1 함수의 최대값 찾기

함수 최대값을 찾는 문제를 진화연산 툴박스를 이용해서 풀어보자. 최대값을 찾고자 함수를 그림 3에 나타내었다.

$$f(x) = x + 10\sin(5x) + 7\cos(4x), \quad 0 \le x \le 6$$

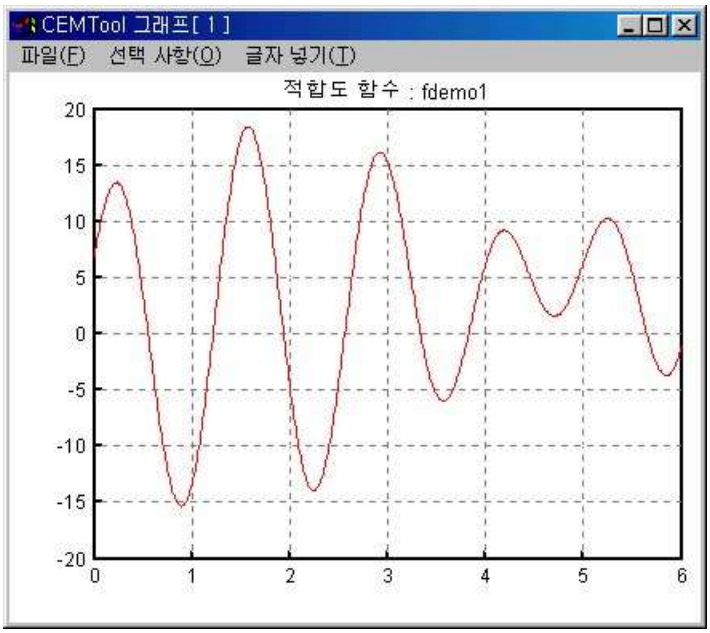


그림 3. 최대값을 찾고자 하는 함수

주어진 구간에서 이 함수는 $x \fallingdotseq 1.5736$에서 최대값으로 18.5722를 갖는다.

툴박스를 이용하여 최대값을 찾는 방법을 살펴보자. 먼저 진화연산 알고리즘에 사용될 매개 변수들을 지정한다.

```
CEMTool>> MaxGen=20;          // 최대 진화 회수
CEMTool>> MaxPop=10;          // 개체군의 크기
CEMTool>> Bounds=[0 6];       // 개체가 나타내는 값의 범위
CEMTool>> XoverOpt=[0.5];     // 교차율
CEMTool>> MutOpt=[0.1 10];    // 돌연변이율과 회수
```

```
CEMTool>> Options=[0.01];   // 정밀도
```

실제 연산은 "sga" 함수에 의해 진행된다. "sga"는 토너먼트 선택, 1-point 교차, 단순 돌연변이 연산자를 이용하여 최적화를 수행한다.

```
CEMTool>> best=sga(MaxPop, MaxGen, Bounds, "fdemo1",
                   XoverOpt, MutOpt, Options);
```

그림 4는 알고리즘 수행 과정에서 개체군의 최고 적합도와 평균 적합도를 나타낸 것이다. 개체군이 크지 않고 문제가 다중 모드이므로 평균 적합도가 감소하는 부분도 생기는 것을 볼 수 있다.

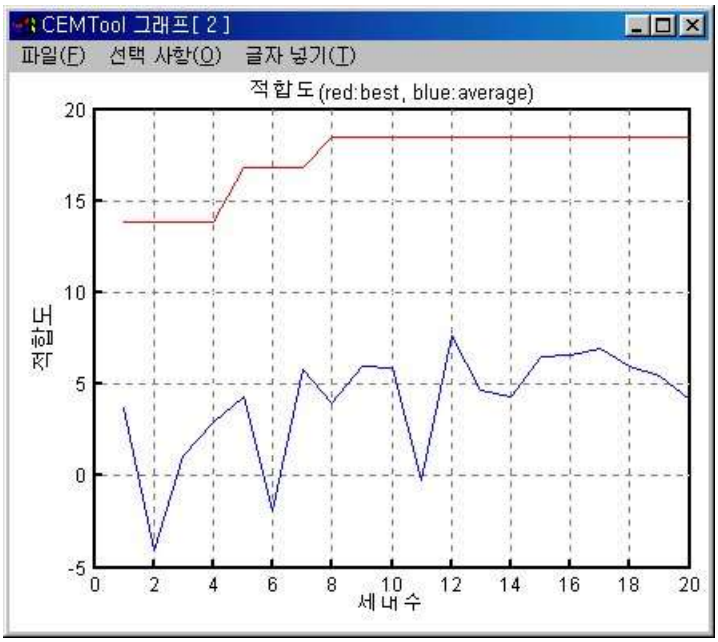


그림 4. 적합도 추이

"sga" 함수가 구한 최적의 개체는 다음과 같다.

```
best = [0   1   0   0   0   0   1   1   0   1   18.5691]
```

이것을 실수값으로 고쳐보면 x=1.5777일 때 최대값 18.5691이다. 문제의 함수가 18.5736에서 최대값을 가지므로, 오차는 1.5777-1.5736=0.0041이다. 이 값은 조건으로 준 정밀도 0.01보다 작으므로 개체가 표현할 수 있는 범위 내에서는 최적의 해를 찾았다고 할 수 있다. 그림 5는 최적해의 위치이다.

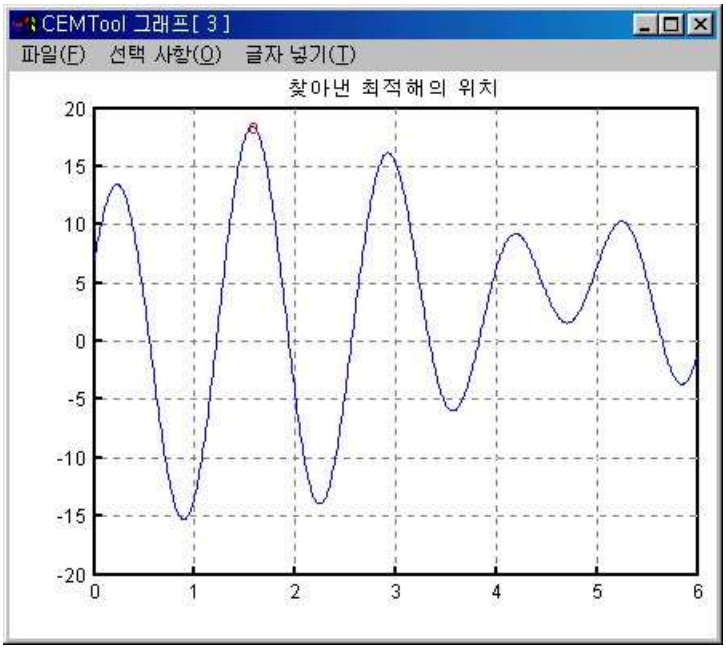


그림 5. 최적해의 위치

함수 최적화 문제(demo_sga.cem)

```
MaxGen=20;
MaxPop=10;
Bounds=[0 6];
XoverOpt=[0.5];
MutOpt=[0.1 10];
Options=[0.01 0];
best=sga(MaxPop,MaxGen,Bounds,"fdemo1",XoverOpt,MutOpt,Options);

[a b]=size(best);
bestx=bin2float(best,Bounds,b-1);
besty=best(1;b);

disp=["     최적해 : (" num2str(bestx) ", " num2str(besty) ")"];
msgprint(" ");
msgprint(disp);

x=0:6:0.01;
```

```
[x y]=fdemo1(x);
figure;
plot(x,y,"b",bestx,besty,"ro");
title("찾아낸 최적해의 위치");
```

7.2 그래프 분할 문제(graph partitioning problem)

7.2.1 개요

그래프 분할 문제는 노드와 노드사이의 연결로 구성되어 있는 그래프를, 노드사이의 연결(브랜치)을 가장 적게 자르면서 여러 부분으로 분할하는 문제이다. 간단한 형태의 예를 통해서 그래프 분할 문제를 이해해 보자. 그림 6의 사각형 모양 그래프에서 사각형을 2개의 노드(꼭지점)를 포함하도록 2부분으로 나눌 수 있는 방법은 대각선 방향의 노드(꼭지점)를 중심으로 나누는 방법과 오른쪽과 왼쪽으로 나누는 방법 두 가지가 있다. 첫 번째 방법은 모두 4군데의 브랜치를 자르면서 나누고 있고, 두 번째 방법은 2군데의 브랜치를 자르면서 나누는 것을 알 수 있다. 이러한 형태의 그래프에서는 최적의 해가 두 번째 방법, 즉 오른쪽과 왼쪽으로 나누는 방법이다.

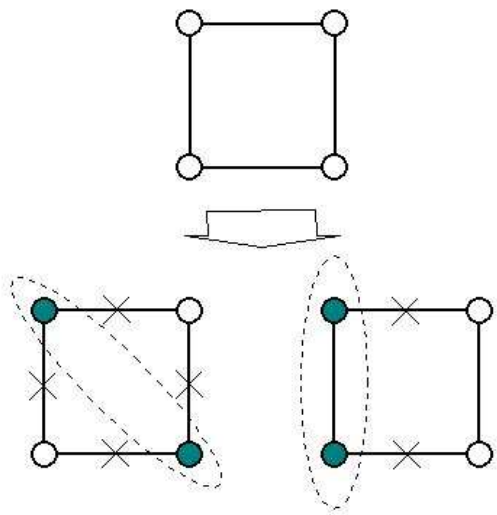


그림 6. 그래프 분할 문제의 간단한 예

이와 같이 간단한 형태의 그래프에서는 직관적으로 최적의 해를 알 수 있지만 그래프가 매우 복잡해질 경우에는 간단히 풀 수 없는 문제가 된다. 이러한 특성 때문에 그래프 분할 문제는 컴퓨터의 성능을 평가하는 데 활용되고 있는 대표적인 문제이다.

본 예제에서는 그림 7의 48개의 노드를 가지는 그래프를 두 부분으로 나누는 문제를 셈툴 진화연산 알고리즘 툴박스를 이용해서 푸는 방법을 보인다. 최적의 해는 한가운데를 세로로 잘라서 오른쪽과 왼쪽 부분으로 나누는 것이다. 이 경우 브랜치를 자르는 회수는 0이다.

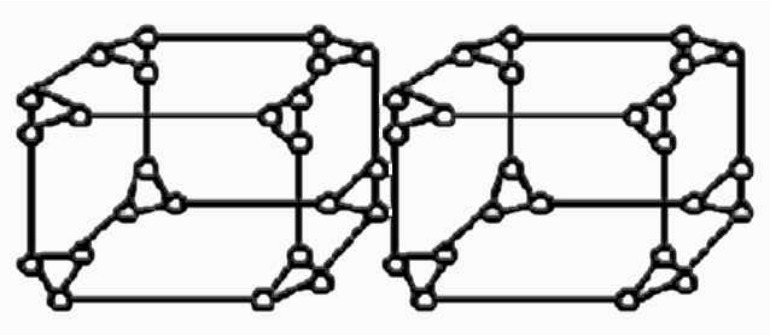


그림 7. 48개의 노드를 가진 그래프

7.2.2 진화연산 툴박스의 그래프 분할 문제 적용

본 예제에 사용할 진화 연산자의 종류와 파라미터들은 다음과 같다.

교차 연산
교차 방법 : Multi-point Crossover
교차 확률 : 0.7
교차 회수 : 4

돌연변이 연산
돌연변이 방법 : Simple Mutation
돌연변이 확률 : 0.05
돌연변이 회수 : 20

선택 연산
선택 방법 : Tournament selection

토너먼트 크기 : 5

적합도 함수 : “gp_fit”

실제 셈툴에서 실행하면, 다음과 같은 출력을 얻을 수 있다.

```
CEMTool>>demo_gp(50,10)

Genetic Algorithm Cemtool Toolbox Demo
      graph partitioning problem

  total node : 48
  optimal fitness : 0

  optimal gene
  111111111111111111111111000000000000000000000000
  or
  000000000000000000000000111111111111111111111111

  Crossover Option
     XoverOpt  =
      0.7000    4.0000
  Mutation Option
     MutOpt  =
      0.0500   20.0000

  Generation->
           1
  Generation->
           2
  Generation->
           3
  Generation->
           4
         :
         :
```

해를 찾는 과정을 임의 탐색(random search) 방법과 비교한 것이 그림 8

이다.

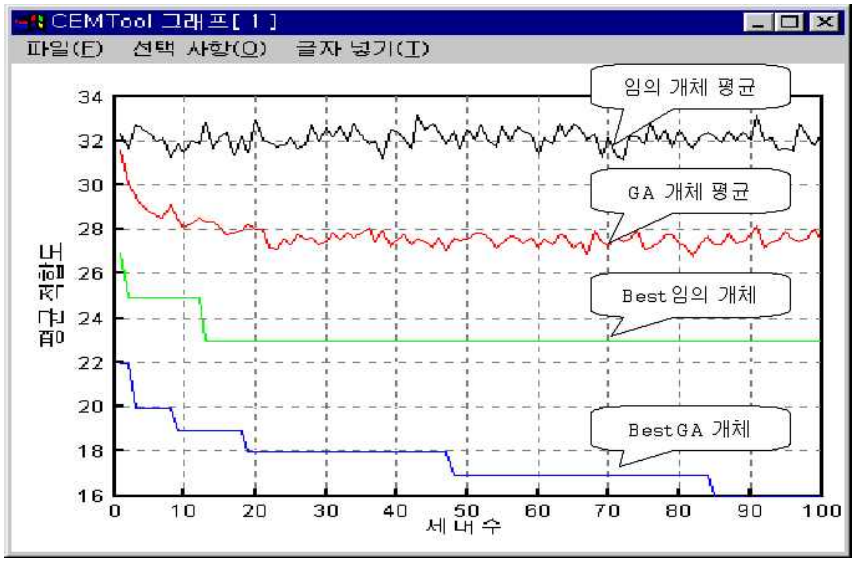


그림 8. 그래프 분할 문제 실험 결과

실험에서 사용한 목적 함수와 전체 코드는 다음과 같다.

48개의 노드를 가진 그래프 분할 문제의 목적함수(gp_fit.cem)

```
-------------------------------------------------------------
function;
v<> Pop;

gp24=[ 1  2; 25 26;  1  3; 25 27;  2 11; 26 35;  2  3; 26 27;
      11  2; 35 26; 11 12; 35 36; 11 10; 35 34;  9 10; 33 34;
       8  1; 32 31;  7  9; 31 33;  8  9; 32 33;  8 23; 32 47;
       7  6; 31 30;  6  4; 30 28;  6  5; 30 29;  4  3; 28 27;
      17  1; 41 25; 17 16; 41 40; 17 18; 41 42; 16 15; 40 39;
      15 13; 39 37; 13 14; 37 38; 13 12; 27 36; 15 14; 39 38;
      14 24; 38 48; 24 23; 48 47; 24 22; 48 46; 22 21; 46 45;
      21 19; 45 43; 19 20; 43 44; 19 18; 43 42; 18 17; 42 41 ];
ss=size(gp24);
pp=size(Pop);
PopSize=pp(1);
v=zeros(PopSize,1);
for (jj=1;jj<=PopSize;jj=jj+1)
{
```

```
   temp=0;
   for (ii=1;ii<=ss(1);ii=ii+1)
   {
      if( (Pop(jj;gp24(ii;1))==1)&&(Pop(jj;gp24(ii;2))==0) )
         temp=temp+1;
      else if ((Pop(jj;gp24(ii;1))==0)&&(Pop(jj;gp24(ii;2))==1) )
         temp=temp+1;
   }
   Pop(jj;pp(2))=temp;
   v(jj)=temp;
}
return
```

전체 알고리즘 코드(demo_gp.cem)

```
function
BestIndi <> PopSize,MaxGen;

msgprint("");
msgprint("");
msgprint("Genetic Algorithm Cemtool Toolbox Demo ");
msgprint("    graph partitioning problem");
msgprint(" ");
msgprint(" ");
msgprint("total node : 48");
msgprint("optimal fitness : 0 ");
msgprint(" ");
msgprint("optimal gene ");
msgprint("111111111111111111111111000000000000000000000000");
msgprint("or");
msgprint("000000000000000000000000111111111111111111111111");
msgprint(" ");
msgprint(" ");

msgprint("Crossover Option");
XoverOpt=[0.7 4]
msgprint(" ");
msgprint("Mutation Option");
MutOpt=[0.05 20]

AvgFit=zeros(MaxGen,1);
AvgRFit=zeros(MaxGen,1);

VarFit=zeros(MaxGen,1);
AvgVar=zeros(MaxGen,1);
```

```
BestFit=zeros(MaxGen,1);
BestRFit=zeros(MaxGen,1);

[Pop]=gpinit(PopSize);
RanPop=gpinit(PopSize);
GenPop=Pop;

TotalLen=48+1;
nn=ceil(PopSize*60*0.01);

for (Generation=1;Generation<=MaxGen;Generation=Generation+1)
{
   NewPopNum=1;

   v=gp_fit(Pop);

   Pop(:,TotalLen)=70-v;
   //[vv nnn]=sel_indis(Pop,50);

   while (NewPopNum<=(PopSize-nn))
   {
      for (ii=1;ii<100;ii=ii+1)
      {
         //simple selection을 사용합니다.
         //[Mate1, Mate2]=s_sel(Pop(:,TotalLen),[]);
         //tournament selection을 사용합니다.
         [Mate1, Mate2]=tour_sel(Pop(:,TotalLen),[5]);
         //truncation selection을 사용합니다.
         //[Mate1, Mate2]=trun_sel(Pop(:,TotalLen),[70]);

         if (hamdis(Pop(Mate1,:),Pop(Mate2,:))<5) break;
      }
      //simple crossover를 사용합니다.
      [Child1,Child2]=m_xover(Pop(Mate1,:),Pop(Mate2,:),XoverOpt);

      NewPop(NewPopNum,:)=Child1;
      if (NewPopNum==(PopSize-nn)) break;
      NewPopNum=NewPopNum+1;
      NewPop(NewPopNum,:)=Child2;
      NewPopNum=NewPopNum+1;
   }

   //Pop(:,TotalLen-1)=NewPop(:,TotalLen-1);
   Pop(1:(PopSize-nn),:)=NewPop;

   for (cnt=1;cnt<=(PopSize-nn);cnt=cnt+1)
```

```
   {
      //simple mutation을 사용합니다.
      Pop(cnt,:)=s_mut(Pop(cnt,:),MutOpt);
   }
   v=gp_fit(Pop);
   Pop(:,TotalLen)=70-v;
   // 적합도의 평균을 계산해낸다.
   AvgFit(Generation)=mean(v);
   // 적합도의 편차를 계산해낸다.
   AvgVar(Generation)=std(v);

   // 가장 훌륭한 개체를 찾는다.
   BestIndi=sel_best(Pop,[]);
   BestFit(Generation)=70-BestIndi(TotalLen);

   [vv nnn]=sel_indis(Pop,60);

   Pop((PopSize-nn+1):PopSize,:)=vv;
   Pop(PopSize,:)=BestIndi;

   // random search
   rv=gp_fit(RanPop);
   RanPop(:,TotalLen)=rv;
   BestRIndi=sel_best(RanPop,[]);
   RanPop=gpinit(PopSize);
   RanPop(PopSize,:)=BestRIndi;

   // 적합도의 평균을 계산한다.
   AvgRFit(Generation)=mean(rv);
   BestRFit(Generation)=70-BestRIndi(TotalLen);

   pg=1:Generation;
   plot(pg,AvgFit(pg),"r",pg,AvgRFit(pg),"B",pg,BestFit(pg),"b",
        pg,BestRFit(pg),"g");
   xtitle("세대수 red:AvgFit black:AvgRFit blue:BestFit green:BestRFit");
   ytitle("평균 적합도");

   msgprint("Generation->");
   Generation
}

ResPop=Pop;
BestIndi=sel_best(Pop,[]);

return
-------------------------------------------------------------
```

Reference

[1] Baker, J., "Adaptive Selection Methods for Genetic Algorithms," Proc. First ICGA, Jul. pp. 101-111, 1987

[2] Jong, D., An analysis of the behaviour of a class of genetic adaptive systems, 1975

[3] Holland, J., Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975

[4] Goldberg, D., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc., 1989

[5] Voigt, H. M.,Born, J. and Santibanez-Koref, I., "Modeling and Simulation of Distributed Evolutionary Search Processes for Function Optimization," in PPSN1, pp. 373-380, 1991

[6] Goldberg, D. E. and Deb, K., "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms," in FGA1, pp.69-93, 1991

[7] Blickle, T. and Thiele, L., "A Comparison of Selection Schemes used in Genetic Algorithms," TIK Report Nr. 11, December, 1995

[8] Crow, J. F. and Kimura, M., An introduction of population genetics theory, New York: Harper and Row, 1970

[9] Whitley, D., "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials Is Best," Proc. Third ICGA, June 1989, pp.116-121, 1989

[10] Hoffmeister, F. and Back. T., "Genetic Algorithms and Evolution Strategies: Similarities and Differences," Proc. First International Conference on PPSN, pp. 447-461, 1990

[11] Mullenbein, H. and Schierkamp-Voosen, "D, Predictive Models for the Breeder Genetic Algorithm: Continuous parameter optimization," Evolutionary Computation GEA toolbox manual by Hartmut Pohheim

부록 A : 함수 목록(List of Functions)

적합도 함수		
함수 이름	설명	쓰이는 곳
fde1	첫 번째 De Jong 함수	4장
fde2	두 번째 De Jong 함수	4장
fde3	세 번째 De Jong 함수	4장
fde4	네 번째 De Jong 함수	4장
fde5	다섯 번째 De Jong 함수	4장
plot_fun	적합도 함수 도시	4,6장

적합도 할당 전략		
함수 이름	설명	쓰이는 곳
s_rank	선형 순위 재조정	4장
back_rank	Baker 순위 재조정	4장
whit_rank	Whitley 순위 재조정	4장
uni_rank	균일 순위 재조정	4장

진화 연산자		
함수 이름	설명	쓰이는 곳
s_sel	비례 선택법	5.1절
loc_sel	지역 선택법	5.1절
sto_sel	균등 선택법	5.1절
tour_sel	토너먼트 선택법	5.1절
trun_sel	절단 선택법	5.1절
s_xover	1-point 교차	5.2절
m_xover	multi-point 교차	5.2절
dis_re	이산 재결합	5.2절
int_re	중간 재결합	5.2절
line_re	선 재결합	5.2절
s_mut	이진 돌연변이	5.3절

알고리즘 수행 관련 함수		
함수 이름	설명	쓰이는 곳
init_sga	simple GA 초기화	6장
sga	simple GA	6장
sel_best	최적의 개체를 선택	6장
sel_indis	적합도가 좋은 개체를 여러 개 선택	6장

공통 함수		
함수 이름	설명	쓰이는 곳
bin2float	2진수를 실수로 변환	2,3장
cal_bit	필요한 비트 수 계산	2,3장
hamdis	Hamming 거리 계산	7장

fde1

▷ 목적

1번째 De Jong 함수

▷ 문법

[x val]=fde1(x,Options)

입력

x : 입력된 값

Options : 쓰이지 않는다.

출력

val : 계산된 결과값

x : 입력된 값이 그대로 저장된다.

▷ 설명

$$val = f(\vec{x}) = \sum_{i=1}^{N}(x_i^2)$$

· Ref

De Jong(1975) : An analysis of the behaviour of a class of genetic adaptive systems

▷ 사용예제

```
CEMTool>> x=[-0.5:0.5:0.1];
CEMTool>> y=[-0.5:0.5:0.1];
CEMTool>> xy=[x' y'];
CEMTool>> [xy val]=fde1(xy)
     xy  =
           -0.5000   -0.5000
           -0.4000   -0.4000
           -0.3000   -0.3000
           -0.2000   -0.2000
           -0.1000   -0.1000
            0.0000    0.0000
            0.1000    0.1000
            0.2000    0.2000
            0.3000    0.3000
            0.4000    0.4000
            0.5000    0.5000

    val  =
            0.5000
            0.3200
            0.1800
```

0.0800
0.0200
0.0000
0.0200
0.0800
0.1800
0.3200
0.5000

▷ 관련함수

fde2, fde3, fde4, fde5

▷ 종류

진화연산 툴박스 함수

▷ 목적

2번째 De Jong 함수

▷ 문법

[x val]=fde2(x,Options)

입력

x : 입력된 값

Options : 쓰이지 않는다.

출력

val : 계산된 결과값

x : 입력된 값이 그대로 저장된다.

▷ 설명

$$val = f(\vec{x}) = \sum_{i=1}^{N-1}((100\times((x_i+1)-x_i^2)^2)+(x_i-1)^2)$$

· Ref

De Jong(1975) : An analysis of the behaviour of a class of genetic adaptive systems

▷ 사용예제

```
CEMTool>> x=[-0.5:0.5:0.1];
CEMTool>> y=[-0.5:0.5:0.1];
CEMTool>> xy=[x' y'];
CEMTool>> [xy val]=fde2(xy)
     xy  =
           -0.5000   -0.5000
           -0.4000   -0.4000
           -0.3000   -0.3000
           -0.2000   -0.2000
           -0.1000   -0.1000
            0.0000    0.0000
            0.1000    0.1000
            0.2000    0.2000
            0.3000    0.3000
            0.4000    0.4000
            0.5000    0.5000

    val  =
           58.5000
           33.3200
```

16.9000
7.2000
2.4200
1.0000
1.6200
3.2000
4.9000
6.1200
6.5000

▷ 관련함수

fde1, fde3, fde4, fde5

▷ 종류

진화연산 툴박스 함수

fde3

▷ 목적

3번째 De Jong 함수

▷ 문법

[x val]=fde3(x,Options)

입력

x : 입력된 값

Options : 쓰이지 않는다.

출력

val : 계산된 결과값

x : 입력된 값이 그대로 저장된다.

▷ 설명

$$val = f(\vec{x}) = \sum_{i=1}^{N} integer(x_i)$$

· Ref

De Jong(1975) : An analysis of the behaviour of a class of genetic adaptive systems

▷ 사용예제

```
CEMTool>> x=[-0.5:0.5:0.1];
CEMTool>> y=[-0.5:0.5:0.1];
CEMTool>> xy=[x' y'];
CEMTool>> [xy val]=fde3(xy)
     xy  =
           -0.5000   -0.5000
           -0.4000   -0.4000
           -0.3000   -0.3000
           -0.2000   -0.2000
           -0.1000   -0.1000
            0.0000    0.0000
            0.1000    0.1000
            0.2000    0.2000
            0.3000    0.3000
            0.4000    0.4000
            0.5000    0.5000

      val =
           -2
           -2
           -2
```

-2
-2
0
0
0
0
0
0

▷ 관련함수

fde1, fde2, fde4, fde5

▷ 종류

진화연산 툴박스 함수

fde4

▷ 목적

4번째 De Jong 함수

▷ 문법

[x val]=fde4(x,Options)

입력

x : 입력

Options : x의 차원

x=[1 2;3 4;5 6]이면 Options=2이어야 한다.

Options은 넣지 않아도 무방하다.

출력

val : 계산된 결과값

x : 입력된 값이 그대로 저장된다.

▷ 설명

$$val = f(\vec{x}) = \sum_{i=1}^{N} i \times x_i^4 + Gauss(0,1)$$

· Ref

De Jong(1975) : An analysis of the behaviour of a class of genetic adaptive systems

▷ 사용예제

```
CEMTool>> x=[-0.5:0.5:0.1];
CEMTool>> y=[-0.5:0.5:0.1];
CEMTool>> xy=[x' y'];
CEMTool>> [xy val]=fde4(xy)
     xy  =
           -0.5000   -0.5000
           -0.4000   -0.4000
           -0.3000   -0.3000
           -0.2000   -0.2000
           -0.1000   -0.1000
            0.0000    0.0000
            0.1000    0.1000
            0.2000    0.2000
            0.3000    0.3000
            0.4000    0.4000
            0.5000    0.5000

     val  =
```

0.6809
0.9537
0.7161
0.4139
1.0083
-1.0571
-0.6764
-0.0589
1.0898
0.3045
1.8221

▷ 관련함수

fde1, fde2, fde3, fde5

▷ 종류

진화연산 툴박스 함수

fde5

▷ 목적

5번째 De Jong 함수

▷ 문법

[x val]=fde5(x,Options)

x : 입력된 값
Options : 쓰이지 않는다.

출력
val : 계산된 결과값
x : 입력된 값이 그대로 저장된다.

▷ 설명

$$val = f(\vec{x}) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$$

· Ref

De Jong(1975) : An analysis of the behaviour of a class of genetic adaptive systems

▷ 사용예제

```
CEMTool>> x=[1 2;3 4;5 6;7 8];
CEMTool>> [a b]=fde5(x,[])
     a  =
           1         2
           3         4
           5         6
           7         8
     b  =
           1.1156
           1.0041
           1.0022
           1.0021
```

▷ 관련함수

fde1, fde2, fde3, fde4

▷ 종류

진화연산 툴박스 함수

plot_fun

▷ 목적
적합도 함수를 그래프로 도시한다.

▷ 문법
plot_fun(FuncName, Bounds, Options)

입력
FuncName : 적합도 함수의 이름
Bounds : 함수의 범위
Options : [Precision ..]

▷ 설명
적합도 함수를 주어진 범위 내에서 그래프로 나타낸다. 이 때 점을 구하는 간격은 옵션에서 정밀도로 준다.

▷ 사용예제
CEMTool>> plot_func("fdemo1", [0 6], [0.1]);

▷ 관련함수

▷ 종류
진화연산 툴박스 함수

s_rank

▷ 목적

simple linear ranking을 수행하는 함수이다.

▷ 문법

[NewFit]=s_rank(Fit,Option)

입력

Fit : 적합도 함수에 의해 계산된 적합도
ex> Fit=[1 ; 2; 3; 4]

Option : [SP]
SP : 상수 1 에서 2 까지

출력

NewFit : simple linear ranking 후의 새로운 적합도

▷ 설명

적합도를 순위를 매겨서 그 순위에 따라 정해진 최소값보다는 크게 할당한다.

$$NewFit(Position) = 2 - SP + \frac{2\,(SP-1)(Position-1)}{NumOfIndi - 1}$$

SP 값에 따라 최소값이 정해지고, 정해진 순위에 따라 새로운 적합도가 할당된다. 적합도가 가장 나쁜 개체도 선택의 기회가 2-SP 만큼은 주어진다.

▷ 사용예제

```
CEMTool>> OldFit=[1;2;3;4];
CEMTool>> Opt=[1.6];
CEMTool>> NewFit=s_rank(OldFit,Opt);
CEMTool>> OldFit
             1
             2
             3
             4

CEMTool>> NewFit
           0.4000
           0.8000
           1.2000
           1.6000
```

▷ 관련함수

bak_rank, whit_rank, uni_rank

▷ 종류
진화연산 툴박스 함수

bak_rank

▷ 목적

Baker's ranking을 수행하는 함수이다.

▷ 문법

[NewFit]=bak_rank(Fit,Option)

입력

Fit : 적합도 함수에 의해 계산된 적합도
ex> Fit=[1 ; 2; 3; 4]

Option : [EtaMax]
EtaMax : 상수 1 에서 2 까지
EtaMin = 2 - EtaMax

출력

NewFit : baker linear ranking이후의 새로운 적합도
단 결과는 확률의 형태로 normalize되어 나온다.

▷ 설명

$$NewFit(position) = \frac{1}{N} * \left(EtaMax - (EtaMax - EtaMin) * \frac{position - 1}{N - 1}\right)$$

EtaMax와 EtaMin 사이를 기존의 적합도에 따라 매긴 순번에 의해서 균등하게 배분해준다. 그럼으로써, 일정하게 제한된 범위 안에 적합도를 넣을 수 있다.

· Ref

J. Baker(1987) : Adaptive Selection Methods for Genetic Algorithms, Proc. First ICGA, Jul. pp. 101-111

▷ 사용예제

```
CEMTool>> OldFit=[1;2;3;4];
CEMTool>> Opt=[1.6];
CEMTool>> NewFit=bak_rank(OldFit,Opt);
CEMTool>> OldFit

             1
             2
             3
             4

CEMTool>>NewFit
            0.1000
            0.2000
```

0.3000
0.4000

▷ 관련함수

s_rank, whit_rank, uni_rank

▷ 종류

진화연산 툴박스 함수

whit_rank

▷ 목적

Whitley 's ranking을 수행하는 함수이다.

▷ 문법

NewFit]=whit_rank(Fit,Option)

입력

Fit : 적합도 함수에 의해 계산된 적합도
ex> Fit=[1 ; 2; 3; 4]

Option : [WhitleyA]
WhitleyA : 상수 1 에서 2 까지

출력

NewFit : whitley linear ranking이후의 새로운 적합도
결과는 확률의 형태로 정규화된다.

▷ 설명

$$NewFit(\lambda, a) = \frac{N}{2(a-1)} \times (a - \sqrt{(a^2 - 4\times(a-1)*\lambda)})$$

(a : 1에서 2까지의 상수, λ : random number on [0 1])

실제 계산된 적합도에 따라서 개체들을 선택할 경우 premature convergence, local search problem 등의 문제가 생기므로, 이러한 문제를 적합도를 새롭게 할당해서 해결하고자 하는 방법이다.

· Ref

D. Whitley(1989) : The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials Is Best, Proc. Third ICGA, June 1989, pp. 116-121

▷ 사용예제

```
CEMTool>> OldFit=[1;2;3;4];
CEMTool>> Opt=[1.6];
CEMTool>> NewFit=whit_rank(OldFit,Opt);
CEMTool>> OldFit

             1
             2
             3
             4

CEMTool>>NewFit
```

0.5339
1.9131
1.3703
1.0150

▷ 관련함수

s_rank, bak_rank, uni_rank

▷ 종류

진화연산 툴박스 함수

uni_rank

▷ 목적

적합도를 동일하게 할당해주는 함수이다.

▷ 문법

NewFit=uni_rank(Fit,Option)

입력

Fit : 적합도 함수에 의해 계산된 적합도
ex> Fit=[1 ; 2; 3; 4]

Option : 쓰이지 않는다.

출력

NewFit : uniform linear ranking이후의 새로운 적합도
결과는 확률의 형태로 정규화된다.

▷ 설명

$$NewFit(Ramda,a) = \frac{1}{N}$$

uniform linear ranking 은 적합도를 모두 동일하게 할당해주므로 모든 개체가 선택될 확률이 동일하게 된다.

· Ref

F. Hoffmeister and T. Back(1990) : Genetic Algorithms and Evolution Strategies:Similarities and Differences, Proc. First International Conference on PPSN, 1990, pp. 447-461

▷ 사용예제

```
CEMTool>> OldFit=[1;2;3;4];
CEMTool>> Opt=[];
CEMTool>> NewFit=uni_rank(OldFit,Opt);
CEMTool>> OldFit

            1
            2
            3
            4

CEMTool>> NewFit

       0.2500
       0.2500
```

0.2500
0.2500

▷ 관련함수
s_rank, bak_rank, whit_rank

▷ 종류
진화연산 툴박스 함수

s_sel

▷ 목적

simple linear selection을 수행하는 함수이다.

▷ 문법

[Mate1, Mate2]=s_sel(Fit,Option)

입력

Fit : 개체들의 적합도를 나타낸다
ex> Fit=[2;3;5;4;7;1];

Option : 쓰이지 않는다.

출력

Mate1, Mate2 : 선택된 개체의 번호를 나타낸다.

▷ 설명

개체의 적합도에 따라서 적합도가 높은 개체는 높은 확률로, 적합도가 낮은 개체는 낮은 확률로 선택의 기회를 주는 선택 연산자이다.

·Ref

D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc., 1989

▷ 사용예제

```
CEMTool>> oldfit=[2;3;5;4;7;1];
CEMTool>> opt=[];
CEMTool>> [sel1 sel2]=s_sel(oldfit,opt)

     sel1  =
              2

     sel2  =
              5
```

▷ 관련함수

loc_sel, stoc_sel, tour_sel, trun_sel

▷ 종류

진화연산 툴박스 함수

▷ 목적

일정한 거리보다 가까운 개체들만 사용해서 선택연산자를 적용한다.

▷ 문법

[Mate1, Mate2]=loc_sel(Fit,Option)

입력

Fit : 적합도 함수에 의해 계산된 적합도
ex> Fit=[1 ; 2; 3; 4]

Option : [Distance]
Distance : 선택하는 개체들의 범위

출력

Mate1,Mate2 : 선택된 개체의 번호

▷ 설명

local selection이란 아래 숫자가 개체를 나타낸다고 할 때,

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8
<- | ->

distance가 1이고 3을 중심으로 선택한다면 2,3,4 개체만 selection에 참여시키는 것이다. 2, 3, 4만을 가지고 selection operator를 적용시킬 때에는 s_selection 함수를 사용한다.

· Ref

H.M. Voigt, J. Born, and Santibanez-Koref, I.(1991) : Modeling and Simulation of Distributed Evolutionary Search Processes for Function Optimization. in PPSN1, pp. 373-380

▷ 사용예제

```
CEMTool>> Fit=[3;4;1;7;5;9;2];
CEMTool>> Opt=[1];
CEMTool>> [Sel1 Sel2]=loc_sel(Fit,Opt)

     Center=>
                 5

     Selected Subpopulation..
                 4        5        6

     Sel1  =
                 6
```

```
      Sel2  =
                   5
```

▷ 관련함수

s_sel, sto_sel, trun_sel, tour_sel

▷ 종류

진화연산 툴박스 함수

sto_sel

▷ 목적

universal stochastic selection(모든 개체에 동일한 확률로 선택연산자를 적용)을 수행하는 함수이다.

▷ 문법

[Mate1, Mate2]=sto_sel(Fit,Option)

입력

Fit : 개체들의 적합도를 나타낸다
ex> Fit=[2;3;5;4;7;1];

Option : 쓰이지 않는다.

출력

Mate1, Mate2 : 선택된 개체의 번호를 나타낸다.

▷ 설명

simple selection에서 개체의 적합도에 따라서 적합도가 높은 개체는 높은 확률로, 적합도가 낮은 개체는 낮은 확률로 선택의 기회가 주어지는 반면, universal stochastic selection은 모든 개체들에게 동일한 확률을 적용시켜 선택의 기회를 균등하게 준다.

▷ 사용예제

```
CEMTool>> oldfit=[2;3;5;4;7;1];
CEMTool>> opt=[];
CEMTool>> [sel1 sel2]=sto_sel(oldfit,opt)

    sel1  =
            4

    sel2  =
            2
```

▷ 관련함수

s_sel, tour_sel, loc_sel, tour_sel

▷ 종류

진화연산 툴박스 함수

tour_sel

▷ 목적

개체군에서 임의로 선택된 크기만큼의 개체만을 가지고 선택연산자를 적용한다.

▷ 문법

[Mate1, Mate2]=tour_sel(Fit,Option)

입력

Fit : 적합도 함수에 의해 계산된 적합도
ex> Fit=[1 ; 2; 3; 4]

Option : [Size]
Size : tournament에 참여하는 개체수

출력

Mate1,Mate2 : 선택된 개체의 번호

▷ 설명

초기에 주어진 크기(Size)만큼의 개체를 임의로 뽑아 sub-population을 만들어 선택연산자를 적용시킨다.

· Ref

D.E. Goldberg, and K. Deb(1991) : A Comparative Analysis of Selection Schemes Used in Genetic Algorithms, in FGA1, pp. 69-93

▷ 사용예제

```
CEMTool>> Fit=[3;4;1;7;5;9;2];
CEMTool>> Opt=[3];
CEMTool>> [Sel1 Sel2]=tour_sel(Fit,Opt)

     Selected subpopulation!!
               6         1         7

     Sel1  =
               6

     Sel2  =
               7
```

▷ 관련함수

s_sel, sto_sel, loc_sel, trun_sel

▷ 종류

진화연산 툴박스 함수

▷ 목적

truncation selection(적합도가 높은 개체만 선택의 후보가 됨)을 수행하는 함수이다.

▷ 문법

```
[Mate1, Mate2]=trun_sel(Fit,Option)
```

입력

```
Fit         : 적합도 함수에 의해 계산된 적합도
              ex> Fit=[ 1 ; 2; 3; 4]
Option      : [Threshold]
              Threshold : 참여시킬 상위 개체들의 비율(%)
```

출력

```
Mate1,Mate2 : 선택된 개체의 번호
```

▷ 설명

Threshold(%)밑에 있는 개체들은 selection에 참여하지 못하는 selection 방법이다.

· Ref

T. Blickle and L. Thiele(1995) : A Comparison of Selection Schemes used in Genetic Algorithms, TIK Report Nr. 11, December

J. Crow and M. Kimura(1970s) : An introduction of population genetics theory, New York: Harper and Row

▷ 사용예제

```
CEMTool>> Fit=[3 ; 4;1;7;5;9;2];
CEMTool>> Opt=[50];
CEMTool>> [Sel1 Sel2]=trun_sel(Fit,Opt)

     Sel1  =
              5

     Sel2  =
              5
```

▷ 관련함수

s_sel, sto_sel, tour_sel, loc_sel

▷ 종류

진화연산 툴박스 함수

▷ 목적

simple crossover를 수행하는 함수이다.

▷ 문법

[Child1,Child2]=s_xover(Mate1, Mate2, XoverOption)

입력

Mate1, Mate2 : 교차연산자가 수행되어야 할 부모 개체

XoverOption : 교차연산 확률

출력

Child1, Child2 : 교차연산이 수행된 다음 생성된 개체

▷ 설명

교차연산자를 가장 간단한 형태로 구현한 것으로, 두 개의 개체를 교차연산 확률에 따라 임의로 선택된 특정 부분을 중심으로 두 개체를 교차하여 새로운 두 개체를 생성한다.

```
Mate1  0 1 0 0 1 : 1 1 1
Mate2  1 1 1 0 0 : 0 1 1
              ↑
     교차연산이 수행되는 부분

교차연산이 수행된 이후의 결과
  Child1  0 1 0 0 1 : 0 1 1
  Child2  1 1 1 0 0 : 1 1 1
```

· Ref

J. Holland(1975) : Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor

D.E. Goldberg(1989) : Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc.

▷ 사용예제

```
CEMTool>> Mate1=[0 1 0 0 1 0 1];
CEMTool>> Mate2=[1 1 1 0 0 1 1];
CEMTool>> XoverOpt=[0.5];
CEMTool>> [Child1 Child2]=s_xover(Mate1,Mate2,XoverOpt)
    Child1  =
              1       1       0       0       1       0       1

                          ↕ 교차연산이 일어난 부분
    Child2  =
```

0 1 1 0 0 1 1

▷ 관련함수

s_mut

▷ 종류

진화연산 툴박스 함수

m_xover

▷ 목적

multi-point crossover를 수행하는 함수이다.

▷ 문법

[Child1,Child2]=m_xover(Mate1, Mate2, XoverOption)

입력

Mate1, Mate2 : 교차연산자가 수행되어야 할 부모 개체
XoverOption : [XoverProbability XoverNumber]
XoverProbability : 교차율
XoverNumber : 나누는 점의 수

출력

Child1, Child2 : 교차연산이 수행된 다음 생성된 개체

▷ 설명

Multi-point 교차 연산자는 두 개의 개체를 교차연산 확률에 따라, 임의로 선택된
여러 지점에서 잘라 교차하여 새로운 두 개체를 생성한다.

```
Mate1  0 : 1 0 0 1 : 1 1 1
Mate2  1 : 1 1 0 0 : 0 1 1
          ↑       ↑
      교차연산이 수행되는 부분

교차연산이 수행된 이후의 결과
Child1  1 1 0 0 1 : 0 1 1
Child2  0 1 1 0 0 : 1 1 1
```

Ref
J. Holland(1975) : Adaptation in Natural and Artificial Systems,
University of Michigan Press, Ann Arbor
D.E. Goldberg(1989) : Genetic Algorithms in Search, Optimization,
and Machine Learning, Addison-Wesley Publishing Company, Inc.

▷ 사용예제

```
CEMTool>> Mate1=[0 1 0 0 1 0 1 1 1 1 0 0 0 1];
CEMTool>> Mate2=[1 1 1 0 0 1 1 0 0 0 1 0 1 1];
CEMTool>> XoverOpt=[0.5 3];
CEMTool>> [Child1 Child2]=m_xover(Mate1,Mate2,XoverOpt)
    Child1  =
              0  0  1  1  0  0  0  0  0  0  0  0  0  1
    Child2  =
              1  1  0  0  1  1  1  1  1  1  1  1  1  0
```

▷ 관련함수
s_xover

▷ 종류
진화연산 툴박스 함수

dis_re

▷ 목적

discrete recombination을 수행하는 함수이다.

▷ 문법

[Child1,Child2]=dis_re(Mate1, Mate2, XoverOption)

입력

Mate1, Mate2 : 입력된 개체

XoverOption : [XoverProbability ...]

출력

Child1, Child2 : 새로 만들어진 개체

▷ 설명

두 개체사이의 값을 교환하는 방식으로 새로운 개체를 생성한다.

```
Mate1 :  12   25    5
Mate2 : 123    4   34
```

선택할 위치를 임의로 발생시킨다.

```
Sel1 :   2    2    1
Sel2 :   1    2    1
```

위의 부모 개체들에 XoverProbability를 적용해서 Child를 생성한다.

```
Child1 : 123    4    5
Child2 :  12    4    5
```

· Ref

H. Mullenbein and D. Schierkamp-Voosen : Predictive Models for the Breeder Genetic Algorithm: Continuous parameter optimization. Evolutionary Computation

GEA toolbox manual by Hartmut Pohheim

▷ 사용예제

```
CEMTool>> m1=[12 25 5];
CEMTool>> m2=[123 4 34];
CEMTool>> opt=[0.5];
CEMTool>> [c1 c2]=dis_re(m1,m2,opt)

    c1  =
          123        25         5
```

```
    c2  =
              123          4          34
```

▷ 관련함수
int_re, line_re

▷ 종류
진화연산 툴박스 함수

int_re

▷ 목적

intermediate recombination을 수행하는 함수이다.

▷ 문법

[Child1,Child2]=int_re(Mate1, Mate2, XoverOption)

입력

Mate1, Mate2 : 입력된 개체

XoverOption : [XoverProbability Range]

XoverProbability : 교차연산자가 실행될 확률

Range : 할당될 값의 상한선과 하한선을 제한하는 값

출력

Child1, Child2 : 새로 만들어진 개체

▷ 설명

두 개체의 값 주변의 값을 할당하여 새로운 개체를 생성한다.

$$Child1 = Mate1 + \alpha_1 \times (Mate2 - Mate1)$$
$$Child2 = Mate1 + \alpha_2 \times (Mate2 - Mate1)$$

α_1, α_2 : scaling factor. XoverOption에서 정해진 Range값에 의해 [-Range 1+Range]까지의 범위에서 uniform한 random variable에서 선택된다.

Mate1 :	12	25	5
Mate2 :	123	4	34

Range = 0.5 ([-0.5 1.5]중에서 선택된다)

Alpha1 :	0.5	1.1	-0.1
Alpha2 :	0.1	0.8	0.5
Child1 :	67.5	1.9	2.1
Child2 :	23.1	8.2	19.5

· Ref

H. Mullenbein and D. Schierkamp-Voosen : Predictive Models for the Breeder Genetic Algorithm: Continuous parameter optimization. Evolutionary Computation

GEA toolbox manual by Hartmut Pohheim

▷ 사용예제

```
CEMTool>> m1=[12 25 5];
```

```
CEMTool>> m2=[123 4 34];
CEMTool>> opt=[0.5 0.5];
CEMTool>> [c1 c2]=int_re(m1,m2,opt)

     c1  =
          81.7853    33.7568     0.0796

     c2  =
          137.5377     6.7074    34.8350
```

▷ 관련함수

dis_re, line_re

▷ 종류

진화연산 툴박스 함수

line_re

▷ 목적

line recombination을 수행하는 함수이다.

▷ 문법

[Child1,Child2]=line_re(Mate1, Mate2, XoverOption)

입력

Mate1, Mate2 : 입력된 개체

XoverOption : [XoverProbability]

XoverProbability : 교차연산자가 실행될 확률

출력

Child1, Child2 : 새로 만들어지 개체

▷ 설명

두 개체의 값 주변의 값을 할당하면서 새로운 개체를 생성한다.

$$Child1 = \alpha_1 \times (Mate1 + Mate2)$$
$$Child2 = \alpha_2 \times (Mate1 + Mate2)$$

α_1, α_2 : 0에서 1사이의 random한 값

Mate1 :	12	25	5
Mate2 :	123	4	34
Alpha1 :	0.5		
Alpha2 :	0.1		
Child1 :	67.5	14.5	19.5
Child2 :	23.1	22.9	7.9

· Ref

H. Mullenbein and D. Schierkamp-Voosen : Predictive Models for the Breeder Genetic Algorithm: Continuous parameter optimization. Evolutionary Computation

GEA toolbox manual by Hartmut Pohheim

▷ 사용예제

```
CEMTool>> m1=[12 25 5];
CEMTool>> m2=[123 4 34];
CEMTool>> opt=[0.5 0.5];
CEMTool>> [c1 c2]=line_re(m1,m2,opt)

     c1  =
```

```
          105.1882    7.3698   29.3465
    c2  =
          114.4803    5.6118   31.7741
```

▷ 관련함수

dis_re, int_re

▷ 종류

진화연산 툴박스 함수

s_mut

▷ 목적

simple mutation을 수행하는 함수이다.

▷ 문법

[Child]=s_mutation(Par,MutOption)

입력

Par : 개체를 나타낸다.

MutOption : [MutationProbability MutationNumber ...]

MutOpt : 돌연변이 연산자의 옵션

MutationProbability : 돌연변이가 일어날 확률

MutationNumber : 돌연변이 연산자의 시행횟수

출력

Child : 돌연변이 연산을 수행한 후의 개체

▷ 설명

simple mutation을 나타내고 돌연변이 연산자를 가장 간단한 형태로 구현한 함수이다. 돌연변이가 일어날 확률과 돌연변이 횟수를 정의해서 돌연변이를 일으킨다.

· Ref

J. Holland(1975) : Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor

D.E. Goldberg(1989) : Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc.

▷ 사용예제

```
CEMTool>> Par=[0 1 0 0 1 1 0]; // 부모 개체
CEMTool>> MutOpt=[0.2 4];      // 돌연변이 확률 0.2  횟수 4번
CEMTool>> Child=s_mut(Par,MutOpt)
   Child  =
            0         0         0         0         0         1         0
                        ↑                             ↑
            돌연변이가 발생된 위치          돌연변이가 발생된 위치
```

▷ 관련함수

s_xover

▷ 종류

진화연산 툴박스 함수

init_sga

▷ 목적

simple GA에 쓰일 유전자코드를 초기화하는 함수이다.

▷ 문법

[pop,ChromosomeLength]=init_sga(PopulationSize,Bounds,FitFunc,Options)

입력

PopSize : 개체군의 크기(개체수)
Bounds : [min max] 개체의 범위
FitFunc : 적합도함수의 이름
Options : [Precision Type]
Precision : 나누는 구간의 크기
Type : 쓰이지 않음
FitOpt : Fitness function의 Options

출력

pop : 새로 만들어진 개체들
ChromLen : 각 유전자의 코드 길이

▷ 설명

유전자 알고리즘을 사용하기 위해 개체군의 크기, 범위, 적합도 함수 등을 사용해서 개체군을 초기화한다.

· Ref

J. Holland(1975) : Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor
D.E. Goldberg(1989) : Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc.

▷ 사용예제

개체군의 크기는 3개 , 0과 1사이에 default precision 0.1, 나머지 파라미터들은 모두 default 값을 사용해서 초기화한 경우 아래와 같은 유전자 코드를 얻을 수 있다. 처음 4칸은 유전자 코드이고 마지막 칸은 기본 적합도 함수에서 계산된 적합도값을 나타낸다.

```
CEMTool>>init_sga(3,[0 1])

    1.0000    1.0000    0.0000    0.0000  -11.8956
    1.0000    0.0000    0.0000    0.0000    3.5717
    1.0000    1.0000    1.0000    1.0000  -14.8033
```

▷ 관련함수

▷ 종류

진화연산 툴박스 함수

▷ 목적

simple GA를 수행하는 함수이다.

▷ 문법

```
BestIndi=sga(PopSize,MaxGen,Bounds,FitFunc,XoverOpt,MutOpt,Options)

 입력
PopSize  : 개체의 개수
MaxGen   : 최대 진화 회수
Bounds   : [Min Max] 변수의 범위
FitFunc  : 적합도 함수 이름
XoverOpt : 교배 연산자 옵션
MutOpt   : [MutationProbability MutationNumber]
           MutationProbability : 돌연변이가 일어날 확률
           MutationNumber      : 돌연변이 연산자의 시행회수
           돌연변이가 일어날 확률, 돌연변이 연산자의 시행회수
Options  : [Precision Type]
           Precision : 정밀도
                       ex> Bounds가 [0 5]인 경우 정확도가 0.1이면
                           유전자의 크기는 5/0.1=50
           Type      : 쓰이지 않음

 출력
BestIndi : 최적의 개체
```

▷ 설명

진화연산 알고리즘을 구현한 예제로 simple crossver와 simple mutation 그리고 simple selection을 사용해서 구현되어 있다. 최대 개체수와 최대 진화 회수를 정해주면 진화연산 알고리즘을 수행해서 가장 훌륭한 개체를 리턴한다.

· Ref

J. Holland(1975) : Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor
D.E. Goldberg(1989) : Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc.

▷ 사용예제

```
CEMTool>> MaxGen=10;
CEMTool>> MaxPop=10;
CEMTool>> Bounds=[0 4];
CEMTool>> XoverOpt=[0.5];
CEMTool>> MutOpt=[0.1 10];
CEMTool>> Options=[0.1 0];
```

```
CEMTool>>sga(MaxPop,MaxGen,Bounds,"fdemo1",XoverOpt,MutOpt,Options)
 BestIndi =
            0.0000 1.0000 1.0000 0.0000 0.0000 1.0000 18.5380
```

▷ 관련함수

s_xover, s_mut, s_sel, sel_best

▷ 종류

진화연산 툴박스 함수

sel_best

▷ 목적
개체들 중 가장 적합도가 높은 것을 찾아내는 함수이다.

▷ 문법
val=sel_best(Pop,Option)

입력
Pop : 전체 개체들을 나타낸다 [유전자 코드 , 적합도]
ex> [0 1 1 0 24.5]
Option : 쓰이지 않는다.

출력
val : 찾아낸 가장 적합도가 높은 개체를 나타낸다.

▷ 설명
개체들 중 가장 적합도가 높은 것을 찾아서 리턴한다.

▷ 사용예제
```
CEMTool>> Pop=[1 0 1 22; 1 1 1 10 ; 0 0 0 5 ; 0 1 1 15];
CEMTool>> sel_best(Pop,[])

             1         0         1        22
```

▷ 관련함수
sga

▷ 종류
진화연산 툴박스 함수

sel_indis

▷ 목적
개체군에서 적합도가 높은 상위의 개체들을 찾는다.

▷ 문법
[val n]=sel_indis(Pop,Option)

입력
Pop : 전체 개체군 [유전자 코드 , 적합도]
Option : 선택할 개체의 수(%)

출력
val : 찾아낸 개체들
n : 찾아내 개체의 수

▷ 설명
개체들 중 가장 적합도가 높은 것을 찾아서 리턴한다.

▷ 사용예제
```
CEMTool>> Pop=[1 0 1 22; 1 1 1 10 ; 0 0 0 5 ; 0 1 1 15];
CEMTool>> [indis n]=sel_indis(Pop,50)

 indis  =
          0        1        1       15
          1        0        1       22
 n  =
          2
```

▷ 관련함수
sel_best

▷ 종류
진화연산 툴박스 함수

bin2float

▷ 목적

2진수를 실수로 바꿔주는 함수이다.

▷ 문법

```
fval=bin2float(BinVal, Bounds, BitLength)
```

입력

```
BinVal    : 2진수 값
Bounds    : [ min max ] 실수값의 범위
BitLength : BinVal의 전체 길이
```

출력

```
fval      : 변환된 실수값
```

▷ 설명

2진수를 주어진 범위에 따라 실수값으로 변환한다.

▷ 사용예제

```
CEMTool>> BinVal=[0 1 0 1];
CEMTool>> Bounds=[0 1];
CEMTool>> BitLength=[4];
CEMTool>> fval=bin2float(BinVal,Bounds,BitLength)
     fval  =
              0.3125
```

▷ 관련함수

▷ 종류

진화연산 툴박스 함수

cal_bit

▷ 목적

일정한 범위를 원하는 정확도로 나눌 경우, 이를 2진수의 형태로 표현했을 때 필요한 비트 수를 계산해 준다.

▷ 문법

```
bits=cal_bit(Bounds, Prec)
```

입력

Bounds : [min max] 값의 범위

Prec : 정확도. 범위가 [0 5]인 경우 정확도가 0.1이면 유전자의 크기는 5/0.1=50

출력

bits : 필요한 비트 수

▷ 설명

범위가 [0 5]이고 정확도가 0.1인 경우, 50칸으로 나누어지게 되고, 이를 2진수로 표현했을 경우 필요한 bit수는 2^6=64이므로 6 bit이 필요하게 된다.

▷ 사용예제

```
CEMTool>> a=cal_bit([0 5],0.1)
     a  =
           6
```

▷ 관련함수

▷ 종류

진화연산 툴박스 함수

hamdis

▷ 목적
Hamming Distance를 계산한다.

▷ 문법
v=hamdis(m1, m2)

입력
m1 : 2진 벡터
m2 : 2진 벡터

출력
v : Hamming distance

▷ 설명
두 이진 벡터에서 서로 다른 원소의 개수를 리턴한다.

▷ 사용예제
```
CEMTool>> a1=[ 0 0 1 1];
CEMTool>> a2=[ 1 1 1 1];
CEMTool>> hamdis(a1,a2)

            2
```

▷ 관련함수

▷ 종류
진화연산 툴박스 함수