유전자 알고리즘을 이용한 뮤테이션 테스팅의 테스트 데이터 자동 생성

정 인 상† · 창 병 모††

요 약

소프트웨어 테스팅의 중요 목표 중의 하나는 '좋은' 테스트 데이터 집합을 생성하는 것으로 이는 매우 어렵고 시간이 걸리는 작업이다. 본 논문은 소프트웨어 테스팅을 위한 자동 테스트 데이터 집합 생성에 유전자 알고리즘을 적용하는 방법을 제시하며 자동 테스트 데이터 생성에서 유전자 알고리즘의 효용성을 보이기 위해 뮤테이션 테스팅을 도입한다. 본 연구는 테스트 데이터 생성 과정이 테스트 대상 프로그램의 구현에 대한 지식을 필요로하지 않는다는 점에서 다른 방법들과 다르다. 또한, 제안된 방법의 효용성을 보이기 위하여 몇 가지 실험을 통해서 블랙박스 테스트 생성 기법인 랜덤 테스팅과 비교한다.

키워드 : 테스트 데이터, 뮤테이션, 유전자 알고리즘

Automatic Test Data Generation for Mutation Testing Using Genetic Algorithms

In sang Jung† · Byeong-Mo Chang††

ABSTRACT

One key goal of software testing is to generate a 'good' test data set, which is considered as the most difficult and time-consuming task. This paper discusses how genetic algorithms can be used for automatic generation of test data set for software testing. We employ mutation testing to show the effectiveness of genetic algorithms (GAs) in automatic test data generation. The approach presented in this paper is different from others in that the test generation process requires no knowledge of implementation details of a program under test. In addition, we have conducted some experiments and compared our approach with random testing which is also regarded as a black-box test generation technique to show its effectiveness.

Key word : test data, mutation, genetic algorithms

1. 서 론

테스팅은 소프트웨어 품질 향상을 위한 필수적인 단계라고 할 수 있으나 많은 시간과 컴퓨팅 자원을 필요로 한다. 따라서, 소프트웨어 테스팅에 투입되는 비용을 줄이기 위해 테스트 데이터를 자동으로 생성하는 것은 매우 중요하다. 지금까지 테스트 데이터 생성을 위한 대부분의 연구들은 구현 정보에 기반하여 테스트 데이터를 선택 혹은 생성하는 화이트박스 테스팅에 초점을 맞추고 있다[2,5]. 보통 이러한 기법들은 프로그램 코드를 분석하여 제약식(constraint system)을 구성하고 그 해를 구함으로써 필요한 테스트 데이터를 구한다. 이러한 접근법은 테스트 데이터 생성에 있어서 사람과의 상호작용을 줄일 수 있으나 동시에 소프트웨어 테스트 비용을 높이는 요인이 된다.

본 논문에서는 테스트 데이터 자동 생성에 유전자 알고리즘을 적용하는 방법을 제시한다. 최근에 이러한 시도가 있었으나 이러한 연구들은 분기 커버(branch coverage)와 같은 구조적 테스팅을 기초로 하고 있으며 각 테스트 데이터에 대해서 프로그램 내의 경로의 수행 횟수 결정을 위한 프로그램 분석을 필요로 한다[15,9].

본 논문에서는 테스트 데이터 자동 생성을 위해 유전자 알고리즘을 사용하는 이전의 연구를 기반으로 하나 테스트 데이터 생성을 위해 프로그램 분석 및 구현에 대한 지식 없이 오직 프로그램의 입출력 관계만 이용한다는 점에서 이전의 연구들과 다르며 효율적인 테스트 데이터 생성이 가능하도록 뮤테이션 테스팅(mutation testing)[12]을 적용한다. 뮤테이션 테스팅의 주요 목표는 주어진 프로그램 P를 P의 뮤턴트 프로그램들과 구별할 수 있는 테스트 데이

※ 본 연구는 1997년 한국과학재단 박사후 연수 프로그램(postdoctorial program)과 한국과학재단 목적기초연구(2000-1-30300-009- 3) 지원으로 수행되었음.
† 정 회 원 : 한성대학교 정보전산학부 교수
†† 정 회 원 : 숙명여자대학교 정보과학부 교수
논문접수 : 2000년 8월 31일, 심사완료 : 2001년 1월 26일

터 집합을 찾는 것이다. P와 어떤 면에서 다른 프로그램을 P의 뮤턴트(mutant) 프로그램이라고 한다[4]. 따라서 좋은 테스트 데이터 집합을 찾기 위해서는 프로그램 분석이 필요한 것이 아니라 한 프로그램과 그 뮤턴트들의 출력들을 비교 검토해야 한다.

유전자 알고리즘은 해의 적합성을 평가할 수 있는 적합성 함수(fitness function)를 필요로 한다. 다행히 뮤테이션 테스팅에서는 그 목표 자체를 최적화해야 할 적합성 함수로 표현할 수 있으며 테스트 데이터 집합의 품질을 뮤테이션 점수라고 불리는 척도를 이용하여 평가할 수 있다. 뮤테이션 테스팅은 테스트할 대상 프로그램의 뮤테이션 점수를 극대화하는 테스트 데이터 집합을 찾는 것을 목표로 한다. 뮤테이션 점수는 매우 간단한 함수로 이를 이용하여 뮤턴트 프로그램의 기능적 행동을 측정할 수 있으며 유전자 알고리즘에서 최적화(이 경우 최대화)해야 할 적합성 함수로 직접 사용할 수 있다.

본 논문에서는 제안된 유전자 알고리즘 기반 테스팅의 효용성을 보이기 위해 랜덤 테스팅과 비교한다. 이를 위해서 자주 사용되는 세 개의 프로그램을 검토하며 유전자 알고리즘을 이용하여 생성된 테스트 데이터의 효용성을 평가하기 위해서 뮤테이션 기준 및 두개의 변형된 뮤테이션 기준들(제약 뮤테이션 및 무작위로 선택된 10% 뮤테이션)을 사용한다.

본 논문의 구성은 다음과 같다. 2절에서는 유전자 알고리즘을 소개하고 3절에서는 뮤테이션 테스팅을 위한 자동 테스트 데이터 생성에 유전자 알고리즘을 적용하는 방법에 대해 설명한다. 4절에서는 실험 결과를 설명하며 5절은 결론과 향후 연구에 대해서 논한다.

2 유전자 알고리즘

유전자 알고리즘은 자연세계의 진화과정을 시뮬레이션함으로써 복잡한 실세계의 문제를 해결하고자 하는 계산모델로 구조가 간단하고 방법이 일반적이어서 응용범위가 매우 넓으며, 특히 적응적 탐색과 학습 및 최적화를 통한 공학적인 문제의 해결에 많이 이용되고 있다[8]. 유전자 알고리즘은 특정 문제에 대한 가능한 해를 간단한 염색체 같은 자료 구조로 표현하고 그 해에 근접하도록 선택(selection), 재조합(recombination), 변이(mutation)와 같은 연산자들을 적용하는 것이다.

(그림 1)은 유전자 알고리즘 실행의 전형적인 과정을 요약해서 기술하고 있다. 유전자 알고리즘 실행에서 첫 단계는 초기화 단계로 이 단계에서는 무작위로 초기 모집단 즉 $P(0)$을 생성하는 것이다. 이 모집단의 각 원소는 염색체라고 불리는 이진 스트링으로 표현된다. 유전자 알고리즘에서는 처음 시작할 때 초기 모집단이 현재 모집단($P(t)$)이 된

```
Genetic Algorithm()
    begin
        t ← 0;
        Initialize(P(t));
        Evaluate(P(t));
        while (not Finished())
        begin
            t ←t+1;
            Select P(t) from P(t−1);
            Recombine P(t);
            Mutate P(t);
            Evaluate(P(t));
        endwhile
        solution ← BestOf(P(t))
    end
```

(그림 1) 유전자 알고리즘 실행을 위한 전형적인 과정

다. 이 모집단에 선택 연산(selection), 재조합 연산(recombination) 및 변이 연산(mutatation)을 적용하여 다음 모집단($P(t+1)$)을 생성한다. 현재 모집단에서 다음 모집단까지의 이 과정을 한 세대라고 하며 다음 세대에서는 이 모집단이 다시 현재 모집단이 되어 동일한 과정을 반복한다. 이러한 반복 과정은 생성된 모집단이 문제에 대한 해에 수렴할 때 종료하게 된다.

한 세대 동안 거치는 과정을 자세히 살펴보면 다음과 같다. 선택 연산은 자연 선택(natural selection) 현상을 모형화한 연산으로 적합성 함수를 이용하여 각 염색체의 적합도를 산정한 후에 적합도가 높은 염색체들만을 선택한다. 선택 연산 후 재조합 연산(recombination)을 할 수 있다. 재조합은 교차(crossover)라고 부르는 유전자 연산을 이용하여 수행된다. 교차는 염색체 쌍 즉 스트링 쌍에 대해서 이 스트링들을 재조합하여 두개의 새로운 스트링을 형성하고 이들을 다음 모집단에 포함시킨다. 재조합은 스트링 재조합 방식에 따라 일점 교차(one-point crossover), 이점 교차(two- point crossover), 균일 교차(uniform crossover) 등의 많은 변형이 존재한다. 예를 들어 일점 교차 연산은 무작위로 스트링 내의 한 지점을 선택하고 선택된 지점의 뒷부분이 서로 교환된다. 모집단에 두 개의 스트링 v_1과 v_2가 있고 5번째 비트 뒤가 교차점으로 선택되었다고 가정해 보자.

$$v_1 = (0\,0\,1\,0\,1 \mid 1\,0\,1),\quad v_2 = (1\,1\,1\,0\,0 \mid 1\,1\,0).$$

그러면 결과 스트링은 다음과 같다.

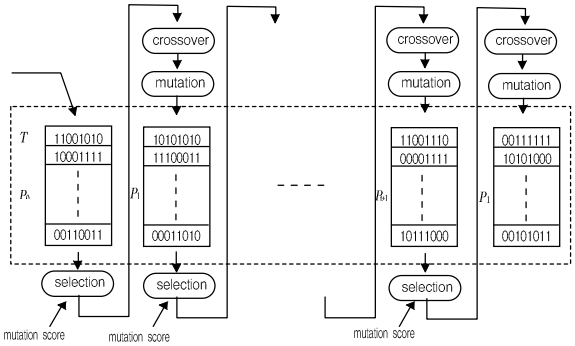
$$v_1 = (0\,0\,1\,0\,1 \mid 1\,1\,0),\quad v_2 = (1\,1\,1\,0\,0 \mid 1\,0\,1).$$

교차 연산 후에 변이 연산을 적용할 수 있다. 이 연산은 비트 패턴에 변이를 도입함으로써 염색체를 구성하는 유전자(gene)를 변형시키는 역할을 수행한다. 즉 염색체가 0과 1로 구성된 이진 스트링으로 구성되어 있으므로 변이 연산

자는 1를 0으로 또는 0을 1로 변형하여 시스템이 지역 최적점에 고착되어 있는 것을 막을 수 있다. 선택, 재조합, 변이 과정이 끝난 후 다음 모집단을 평가할 수 있다. 이러한 평가, 선택, 재조합, 변이 과정이 유전자 알고리즘의 실행에서 한 세대를 형성한다.

3. 유전자 알고리즘을 이용한 뮤테이션 테스팅을 위한 테스트 생성

이 절에서는 뮤테이션 테스팅을 개괄적으로 기술하고 이를 위한 유전자 알고리즘을 이용한 테스트 데이터 자동 생성에 대해서 기술한다.



(그림 2) 유전자 알고리즘을 이용한 테스트 생성 과정

3.1 뮤테이션 테스팅

뮤테이션 테스팅은 결함-기반 테스팅(fault-basedtesting) 개념을 기반으로 하는 테스트 기준 중에 하나로 두 가지 원리 즉 유능한 프로그래머 가정과 커플링 효과를 기초로 하고 있다. 첫번째 가정은 유능한 프로그래머는 거의 오류가 없는 프로그램을 작성하는 경향이 있다는 것이고 커플링 효과는 모든 간단한 결함을 검출할 수 있는 테스트 데이터 집합은 보다 복잡한 결함들도 검출할 수 있다는 사실을 의미한다.

뮤테이션 테스팅은 테스트할 프로그램의 변형인 뮤턴트(mutant)들을 생성하는 것으로 시작된다. 뮤턴트 프로그램은 하나의 결함을 포함한 프로그램으로 간주할 수 있다. 원래 프로그램 P에 구문적 변화를 주는 뮤턴트 연산자를 사용하여 결함을 삽입한다. 예를 들어 연산자 '≤'를 '<'로 대치함으로써 프로그램의 뮤턴트를 만드는 뮤턴트 연산자를 생각해 보자. 조건식 "if $x \leq y$"를 포함하는 프로그램에 이 연산자를 적용하면 "if $x < y$"으로 대치하여 뮤턴트 프로그램을 만들 수 있다.

뮤테이션 테스팅의 목표는 프로그램 P에 뮤턴트 연산자를 적용하여 만든 모든 뮤턴트들을 P로부터 구별할 수 있는 테스트 데이터 집합 T을 찾는 것이다. 어떤 뮤턴트 M_i에 대해서 P와 다른 출력을 내는 테스트 데이터가 T에 존재하면 이 테스트 데이터는 M_i를 죽였다고 말하며 이는 이 테스트 데이터가 뮤턴트 M_i에 포함된 결함을 검출할 수 있음을 의미한다. 죽은(dead) 뮤턴트는 뮤테이션 테스팅 과정에서 더 이상 고려할 필요가 없다.

따라서 뮤테이션 테스팅에서 좋은 테스트 데이터 집합이란 대상 프로그램에서 생성될 수 있는 뮤턴트 프로그램들을 가능한 한 많이 구별할 수 있는 테스트 데이터들을 많이 포함하는 집합이라 할 수 있다. 한편 구문적으로는 다르지만 기능적으로는 원래 프로그램과 같은 뮤턴트가 존재할 수 있는데 이러한 동등한(equivalent) 뮤턴트들은 테스트 집합 생성 과정에서 고려할 필요가 없다. 이러한 개념에 기반하여 뮤테이션 테스팅에서 뮤테이션 점수(MS)라고 불리는 척도를 사용하여 테스트 데이터 집합의 질을 평가하는데 이는 동등하지 않는 뮤턴트들로부터 구별될 수 있는 뮤턴트들의 비율이다. 즉

$$MS(P,T) = \frac{\#\ of\ Dead}{\#\ of\ Mutants\ -\ \#\ of\ Equivalent}.$$

테스트 데이터 집합 T와 프로그램 P에 대해서 뮤테이션 점수가 100%라면 T는 뮤테이션 기준에 대해서 적합하다고 볼 수 있다. 뮤테이션 점수가 100%인 테스트 데이터 집합은 모든 뮤턴트 프로그램들을 원래의 프로그램과 구분할 수 있는 테스트 데이터들을 포함하는 집합을 의미한다.

뮤테이션 테스팅에서 주문제는 프로그램에 대해서 생성될 수 있는 뮤턴트들의 수이다. n 줄의 프로그램으로부터 n^2개의 뮤턴트 프로그램이 생성될 수 있다[12]. 따라서 프로그램을 테스팅하는 동안에 고려해야 할 뮤턴트 프로그램들의 수를 줄이는 한가지 전략은 일부 뮤테이션 연산자들만을 고려하는 것이다. 이러한 방법을 제약 뮤테이션(constrained mutation) 테스팅이라고 하며 이 방법은 고려할 뮤테이션 연산자들로부터 "좋은" 부분집합을 선정하는 것이 매우 중요하다. 여기서 "좋다"는 의미는 결함 검출 능력을 희생하지 않으면서 실행 비용을 줄일 수 있는 것을 의미한다.

본 연구에서는 제약 뮤테이션 테스팅을 사용하여 실험을 통해서 뮤테이션 테스팅과 데이터 흐름 테스팅의 결함 검출 효과성을 비교하였다[12]. 이 선택적 뮤테이션 테스팅은 Mathur에 의해 제안되었으며 뮤턴트의 수보다는 오류 검출에 있어서 중요도를 기초로 하여 뮤턴트 연산자들을 선택한다[11]. 뮤테이션 테스팅의 또 다른 변형은 무작위로 각 뮤턴트 타입의 일부 뮤턴트들만을 선정하여 검토하고 나머지는 무시하는 것이다. 본 실험에서는 모든 가능한 뮤턴트의 10%만을 무작위로 선정하고 검토한다. 앞으로는 무작위 선정 10% 뮤테이션 기준을 단지 10% 기준이라고 한다.

3.2 유전자 알고리즘을 이용한 테스트 생성 과정

뮤테이션 테스팅을 위한 테스트 데이터 생성 목표는 뮤턴트들을 가능한 많이 구별할 수 있게 하는 테스트 데이터

를 생성하는 것이다. 유전자 알고리즘 기반 테스트 데이터 데이터 생성은 적합성 함수에서 이 성질을 반드시 반영하여 더 '적합한' 테스트 데이터가 테스트 데이터 집합에 포함되도록 해야 한다.

앞서 기술한 것처럼 각 염색체에 값을 배정하는 f_i로 나타내는 적합성 함수를 평가하는 중간 과정이 있다. 각 염색체가 테스트 데이터를 나타낸다고 가정하면 f_i는 테스트 데이터 집합 대신에 개별적인 테스트 데이터 t_i에 대해서 정의된 뮤테이션 점수

$$f_i(P, t_i) = \frac{\#\ of\ Dead}{\#\ of\ Mutants - \#\ of\ Equivalent}.$$

에 의해 계산될 수 있다는 것을 쉽게 알 수 있다. f_i는 프로그램 지식을 필요로 하지 않으며 어떤 뮤턴트 프로그램이 동등한 지 안다는 가정 하에서 테스트 프로그램을 분석하지 않고 테스트 데이터를 생성할 수 있게 한다는 점을 주목해야 한다. 많은 뮤턴트 프로그램들을 죽이는 테스트 데이터는 그렇지 않는 테스트 데이터에 비해 보다 높은 값을 반환한다. 이는 모집단의 원소들로 하여금 아직 살아 있는 뮤턴트들을 죽일 수 있는 가능성을 높여준다.

(그림 3)는 뮤테이션 테스팅을 위한 유전자 알고리즘 기반 테스트 데이터 생성 과정이다. 어떤 모집단 내의 각 염색체는 테스트 대상 프로그램에서 입력으로 필요로 하는 입력 벡터를 구성하는 테스트 데이터들이다. 초기 모집단은 무작위로 생성하고 다음 모집단들은 선택, 교차 및 변이 연산자를 적용하여 유도할 수 있다. 각 염색체를 위에서 정의된 f_i로 평가한 후에 선택 연산자는 현재 모집단에서 다음 교차 및 변이 연산에 참가할 두 염색체를 선택한다. 적합성 평가 과정에서 어떤 테스트 데이터 즉 염색체에 의해서 죽게된 뮤턴트들은 다음 평가에서는 무시된다.

```
TestSet Genetic Algorithm(Program P)
begin
        TestSet T ← 0;
        Generate Mutants for P;
        t ← 0;
        Initialize(P(t));
        T ← T ∪ P(t);
        Evaluate(P(t))*;
        while (not Finished())
        begin
                t ← t+1;
                Select P(t) from P(t-1);
                Recombine P(t);
                Mutate P(t);
                T ← T ∪ P(t);
                Evaluate(P(t))*;
        endwhile
        return T;
end
```

(그림 3) 유전자 알고리즘 기반 테스트 생성 과정

(그림 3)은 본 실험에서 사용할 유전자 알고리즘의 골격을 보여주고 있다. 우선 테스트할 프로그램의 뮤턴트들을 생성해야 한다. 이는 테스트에서 사용될 뮤턴트들의 종류를 선택한 후에 할 수 있다. 10% 기준을 테스트 기준으로 사용한다면 모든 가능한 뮤턴트들의 10%만을 생성할 필요가 있다. (그림 4)에서 기호 * 로 표시된 평가 단계는 $P(t)$내의 각 테스트 데이터의 적합성 평가와 관련되어 있다. 앞서 언급한 것처럼 각 테스트 데이터의 적합성은 뮤테이션 점수를 계산하여 알 수 있는데 이는 $P(t)$에 대해서 원래 프로그램과 (물론 살아있는) 그 뮤턴트들의 실행을 필요로 한다. (그림 3)의 과정은 전통적인 유전자 알고리즘과 약간 다르다. 전통적인 유전자 알고리즘에서는 각각의 스트링은 문제에 대한 가능한 해를 나타낸다. 이 연구에서 사용된 유전자 알고리즘은 각 모집단 즉 테스트 데이터 집합을 고려하여 각 세대의 끝에서 해의 일부를 생성한다. 테스트 생성 과정이 $t = n\,(n \geq 0)$에 수렴한다고 가정하면 테스트 집합 T는

$$T = \bigcup_{t=0}^{t=n} P(t).$$

4 실험결과

이 절에서는 유전자 알고리즘 기반 테스트 데이터 생성과 랜덤 테스트 데이터 생성(간단히 랜덤 테스팅)를 비교하여 그 결과에 대해 기술한다. 테스트 데이터 생성을 위해 많은 시스템에서 널리 사용되어 온 랜덤 테스팅은 이 논문에서 제안된 방법과는 달리 프로그램에 대한 지식을 요구하지 않는다. 몇몇 연구들은 랜덤 테스팅이 상당히 직관적이지만 어떤 환경하에서는 유효할 수 있다는 것을 보였다. 이러한 점때문에 우리는 비교를 위해서 랜덤 테스팅을 선택하였다. 본 실험에서는 뮤테이션 기준과 그 변형들(제약 뮤테이션과 10% 기준)를 사용하여 두 테스팅 방법을 비교한다.

4.1 예제 프로그램

본 실험에서는 아래에 3 가지 프로그램을 사용하였다.

- QUAD [14] : 이 프로그램은 a,b,c에 대하여 2차 방적식 $ax^2 + bx + c = 0$에 대한 정수 해를 출력한다. 이 프로그램은 반복을 포함하지 않지만 매우 비선형적인 성질을 갖는다.
- POSITION [12] : 이 프로그램은 두 개의 입력, 배열 a 및 max 값을 받아서 합이 max와 같거나 초과될 때까지 a의 원소들을 더한다. 합이 max와 같거나 초과되게 하는 원소가 존재하면 그 위치가 반환되고 그렇지 않으면 0이 반환된다. 이 프로그램은 약간의 결함을 가지고 있다.

● TRITYP [4] : 이 프로그램은 테스트 데이터 선택을 위한 전형적인 벤치마크로 삼각형 세 변의 길이를 나타내는 세개의 정수를 받아서 삼각형, 정삼각형, 직각삼각형, 이등변삼각형 여부를 결정한다.

본 실험에서는 테스팅 도구 Proteum [3]를 사용하였다. Proteum은 C 프로그램에 대한 뮤테이션 테스팅 적용을 지원하기 위해서 개발되었으며 [1]의 연산자들을 기초로 71개의 뮤테이션 연산자들을 제공한다. 프로그램과 테스트 집합이 주어지면 Proteum은 뮤턴트들의 집합을 생성하고 테스트 데이터 집합에 대해서 그들을 실행하여 뮤테이션 점수를 계산한다. 이 도구를 사용하는 동안 테스트하는 사람은 어떤 뮤턴트가 동등한지 결정할 책임이 있다.

<표 1>는 사용된 프로그램의 크기(LOC)와 뮤터트들의 수를 나열하고 있다. Proteum를 사용해서 얻어진 이 척도들은 실험에서 고려할 프로그램들의 상대적인 복잡도를 나타내고 있다.

<표 1> 프로그램 척도

프로그램	LOC	뮤턴트 프로그램 개수		
		100%	10%	제약
QUAD	12	693	61	74
POSITION	16	422	38	51
TRITYP	26	1539	164	196

4.2 뮤테이션 기준을 이용한 비교 결과

본 연구에서는 20 개의 테스트 데이터 집합을 모집단으로 랜덤 테스팅과 유전자 알고리즘 기반 테스팅에 대해서 실험하였고 각 프로그램에 대해서 5회식 반복하였다. 두 테스팅을 위한 종료 조건으로 뮤테이션 점수를 사용하여 100% 뮤테이션 점수에 달할 때까지 테스트를 생성하였다. 100% 뮤테이션 점수에 도달하지 못할 상황에서는 25세대를 초과하면 테스트 생성을 멈추게 하였다. 따라서 많아야 500개의 서로 다른 테스트들을 생성할 수 있는데 이는 단위 테스팅에 적절하다고 볼 수 있다.

유전자 알고리즘에서는 비트 스트링 표현을 선택하는 것이 매우 중요하다. 여기서는 두 종류의 표현 즉 부호 비트를 갖는 이진 코드와 그레이 코드를 사용하였다. QUAD와 POSITION 프로그램에 대해서는 이진 표현이 그레이 코드보다 효과적임을 발견할 수 있었고 TRITYP에 대해서는 그레이 코드가 이진 표현보다 약간 좋았다. 우리들은 유전자 알고리즘 기반 테스팅에서 두 가지 다른 교차 연산 즉 일점 교차 연산 및 균일 교차 연산을 테스트하였는데 균일 교차 연산이 일점 교차 연산보다 TRITYP 프로그램에 대해 약간 좋은 결과를 주는 것을 볼 수 있었다. QUAD와 POSITION 프로그램에 대해서는 반대로 일점 교차 연산이 균일 교차 연산보다 좋은 결과를 주는 것을 볼 수 있었다

4.2.1 뮤테이션 기준을 이용한 평가

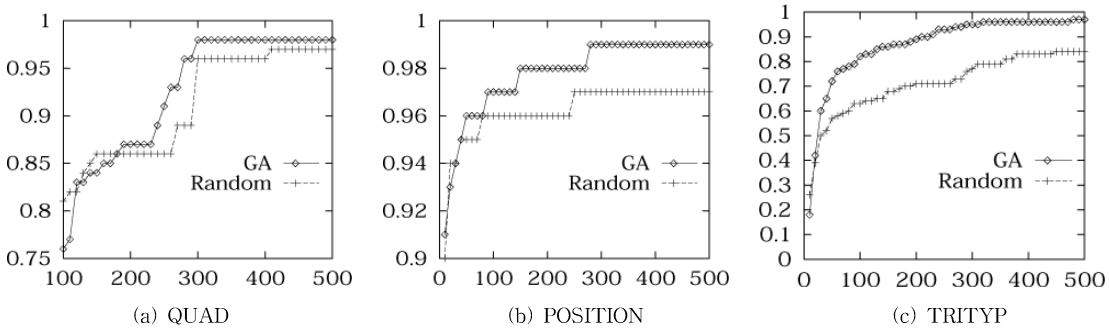
이 실험을 하기 위해서 Proteum에서 제공하는 모든 뮤테이션 연산자를 적용하여 실험 대상 프로그램 각각에 대해서 모든 가능한 뮤턴트들을 생성하였다. 두 테스팅 방법 각각에 대하여 <표 2>는 얻어진 뮤테이션 점수에 따라 최선, 최악, 평균 결과를 보여주고 있으며 (그림 4)는 테스트 수에 따른 평균 뮤테이션 점수를 보여주고 있다. x-축은 실행된 테스트 데이터의 수를 나타내고 y-축은 얻어진 뮤테이션 점수를 나타낸다.

<표 2> 뮤테이션 기준을 이용한 비교 결과

프로그램	유전자 알고리즘			무작위		
	최선	최악	평균	최선	최악	평균
QUAD	1.0	0.96	0.98	0.97	0.85	0.94
POSITION	1.0	0.97	0.99	0.99	0.94	0.97
TRITYP	1.0	0.90	0.97	0.91	0.80	0.84

위의 결과로부터 다음과 같은 사실을 관찰할 수 있다.

● 유전자 알고리즘은 사용된 모든 프로그램에 대해서 랜덤 테스팅보다 좋은 뮤테이션 점수를 주었다. 또한 유전자 알고리즘을 사용한 경우에는 모든 뮤턴트가 죽은 경우가 있었으나 랜덤 테스팅에 대해서는 그러한 경우가 없었다. 실제로 이는 사용자가 수작업으로 뮤테이션 시스템에서 95% 이상을 기록하는 테스트 데이터들을 생성하는 것은 매우 어렵고 시간이 많이 드는 작업이라는 사실을 상기 할 때 이 결과는 유전자 알고리즘 기반 테스팅이



(a) QUAD (b) POSITION (c) TRITYP

(그림 4) 뮤테이션 기준을 이용한 유전자 알고리즘과 랜덤 테스팅 비교

테스트 데이터 생성을 위해 매우 효과적임을 보여주고 있다.

- 유전자 알고리즘 기반 테스팅은 다른 프로그램보다 TRITYP 프로그램에 대해서 훨씬 향상된 결과를 주고 있는데 이는 본 실험이 제한적이긴 하지만 유전자 알고리즘이 프로그램의 크기와 복잡도가 커감에 따라 더욱 효과적일 수 있다는 사실을 나타내고 있다.
- POSITION 프로그램은 두 테스팅 방법에 대해서 20개의 테스트 데이터로 높은 뮤테이션 점수에 빨리 도달하였다. 실험에 사용한 POSITION 프로그램은 예외상황을 처리하는 것과 같은 특별한 경우를 처리할 수 있는 구현 부분이 없다는 점을 유의할 필요가 있다. 즉, POSITION 프로그램은 다른 프로그램에 비해서 테스트가 용이함을 의미한다. Yin 등이 [16]에서 관찰한 것처럼 랜덤 테스팅은 테스트가 용이한 프로그램에 대해서는 좋은 결과를 줄 수 있으며 처음 몇 개의 테스트 데이터를 적용할 때 유전자 알고리즘 기반 테스팅과 랜덤 테스팅 사이에 별 차이가 없다. 그러나, 이 경우에도 100개의 테스트를 적용한 후에는 유전자 알고리즘이 랜덤 테스팅보다 약간 좋은 점수를 주었다.

4.2.2 뮤테이션의 변형 기준을 이용한 평가

이 실험은 제약 뮤테이션과 10% 기준은 동등하지 않는 뮤턴트들을 구별하는 능력을 5% 이내에서 희생하면서 검사할 뮤테이션의 수를 상당히 줄일 수 있다는 [13]의 선행 연구에 기반을 두고 있다. 즉 유전자 알고리즘 기반 테스팅이 뮤테이션의 두 변형에 대해서 랜덤 테스팅보다 효과적인 것을 보인다면 뮤테이션 테스팅과 유전자 알고리즘 관련 비용은 모든 가능한 뮤턴트들의 부분집합만 고려함으로써 크게 줄일 수 있을 것이다.

<표 3> 10% 기준을 이용한 비교 결과

프로그램	유전자 알고리즘			무작위		
	최선	최악	평균	최선	최악	평균
QUAD	1.0	0.98	0.99	1.0	0.95	0.97
POSITION	1.0	1.0	1.0	1.0	0.97	0.98
TRITYP	1.0	0.94	0.98	0.97	0.86	0.93

<표 3>와 <표 4>은 10% 뮤테이션과 제약 뮤테이션에 대해서 두 테스팅 방법에 의한 뮤테이션 점수를 나타내고 있다. 이 결과로부터 두 테스팅에 대해서 테스트 기준으로 10% 기준을 사용하는 것이 제약 뮤테이션보다 95% 이상의 뮤테이션 점수을 얻기 쉽다는 것은 알 수 있었다. 이는 본 실험에서 사용한 제약 뮤테이션의 특성에 기인한 것이다.

<표 4> 제약 뮤테이션을 이용한 비교 결과

프로그램	유전자 알고리즘			무작위		
	최선	최악	평균	최선	최악	평균
QUAD	1.0	0.96	0.98	0.97	0.91	0.94
POSITION	1.0	0.96	0.97	0.87	0.87	0.87
TRITYP	1.0	0.93	0.97	0.97	0.91	0.94

본 연구에서는 VDTR(absolute value insertion)과 ORRN(relational operator replacement) 연산자만을 검토하였다. 이 두 연산의 중요성은 [12]에서 나타나 있으며 이 논문은 이 두연산을 사용한 뮤테이션 테스팅이 10% 기준보다 좋은 결함 발견 능력을 갖는다는 것을 보이고 있다. VDTR과 ORRN 연산자는 교차될 식의 입력 도메인의 다른 부분 혹은 경계로부터 테스트 데이터를 생성하게 유도한다. 모든 경우에 유전자 알고리즘 기반 테스팅은 랜덤 테스팅 보다 좋은 결과를 준다는 것을 관찰할 수 있었다. 또한 제약 뮤테이션 기준에 대해 유전자 알고리즘 기반 테스팅을 사용하는 것이 10% 뮤테이션 기준에 대해 랜덤 테스팅을 하는 것보다 우월하다는 것을 알 수 있었다. 이는 유전자 알고리즘 기반 테스팅이 결함 검출 능력의 희생없이 뮤테이션 테스팅을 위한 작업 비용을 상당히 감소시켜준다는 것을 나타내고 있다. 결과적으로 제약 뮤테이션 기준은 유전자 알고리즘 기반 테스팅을 위한 하나의 좋은 뮤테이션 기준이 될 수 있다.

5 결론 및 향후 연구

본 논문에서는 테스트 데이터의 자동 생성을 위해 유전자 알고리즘을 이용한 방법을 제시하였다. 본 논문에서 제시한 방법은 프로그램에 대한 구현 내역을 요구하지 않는 '블랙박스' 테스팅에 기반하고 있으며 생성된 테스트 데이터의 효용성을 보이기 위하여 뮤테이션 테스팅을 도입하고 랜덤 테스팅과 비교하는 실험을 수행하였다. 실험 결과 유전자 알고리즘 기반 테스팅이 랜덤 테스팅보다 우월하다는 것을 알 수 있었다.

본 논문에서 소개한 방법은 회귀 테스팅(regression testing) [7]에도 별 어려움이 없이 적용할 수 있으리라 기대된다. 이는 회귀 테스팅의 목표가 원래 프로그램을 변형된 프로그램과 구별하게 할 가능성이 큰 테스트데이터들을 선택 혹은 생성하는 것이기 때문이다.

참 고 문 헌

[1] H. Agrawal, R. A. DeMillo, R. Hataway, W. Hsu, W. Hsu, E. W. Krauser, R. J. Martin, and A. P. Mathur, "Design of Mutant Operators for C Programming Language," *Technical Report SERC-TR-41-P*, Software Engineering Research Center, Purdue Univ., March, 1989.

[2] L. A. Clarke, "A System to Generate Test Data and Symbolically Execute Programs," IEEE Transactions on Software Engineering, Vol.SE-2, pp.215-222, Sept. 1976.

[3] D. E. Delamaro, J. C. Maldonado, and A. P. Mathur, "Proteum-A Tool for the Assessment of Test Adequacy for C Programs-User's Guide," Technical Report SERC-TR-168-P, Software Engineering Research Center, Purdue

Univ., Apr., 1996.
[4] R. A. DeMillo, R. J. Lipton, and F. G. Sayward,"Hints on Test Data Selection : Help for the Practicing Programmar," in IEEE Computer, Vol.SE-11(4), pp.34-41, Apr. 1978.
[5] R. A. DeMillo and A. J. Offutt,"Constraint-Based Automatic Test Data Generation," IEEE Transactions on Software Engineering, Vol.17, No.9, pp.900-910, Sept. 1991.
[6] D. E. Goldberg,"Genetic Algorithms in Search, Optimization Machine Learning," Addison-Wesley, Reading, Massachusetts, 1989.
[7] J. Hartman and D. J. Robson, "Approaches to Regression Testing," in Proceedings of IEEE Conference on Software Maintenance, Los Alamitos, Calif., pp.368-372, 1988.
[8] J. H. Holland, "Adaption in Natural and Artificial Systems," Ann Arbor : The University of Michigan Press. 1975.
[9] B. F. Jones, H.-H. Sthamer and D. E. Eyres, "Automatic Structural Testing Using Genetic Algorithms," Software Engineering Journal, pp.299-306, Sept., 1996.
[10] Y. K. Malaiya, "Antirandom Testing : Getting the Most out of Black-Box Testing," *Technical Report CS-96-129*, Colorado State Univ., 1996.
[11] A. P. Mathur, "Performance, Effectiveness, and Reliability Issues in Software Testing," *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, pp.604-605, Tokyo, Japan, Sept., 1991.
[12] A. P. Mathur and W. E. Wong, "An Empirical Comparison of Mutation and Data Flow Based Test Adequacy Criteria," *Technical Report SERC-TR- 135-P*, Software Engineering Research Center, Purdue Univ., 1993.
[13] A. P. Mathur and W. E. Wong, "Evaluation of the Cost of Alternate Mutation Testing Strategies," *Technical Report SERC-TR-138-P*, Software Engineering Research Center, Purdue Univ., 1993.
[14] J. Voas, L. Morell, and K. Miller,"Predicting Where Faults Can Hide from Testing," *IEEE Software*, pp.41-48, March, 1991.
[15] A. L. Watkins, "A Tool for the Automatic Generation of Test Data Using Genetic Algorithms," in *Proceedings of Software Quality Conference*, Dundee, Scotland, 1995.
[16] H. Yin, Z. Lebne-Dengel and Y. K. Malaiya, "Automatic Test Generation using Checkpoint Encoding and Antirandom Testing," in *Proceedings of International Symposium on Software Reliability Engineering*, pp.84-95, Oct., 1997.

정 인 상

e-mail : insang@hansung.ac.kr
1987년 서울대학교 컴퓨터 공학과 학사
1989년 한국과학기술원 전산학과 석사
1993년 한국과학기술원 전산학과 박사
1993년~1994년 한국전자통신연구원 박사 후 연수연구원
1994년~1999년 한림대학교 부교수
1995년~1995년 영국 Durham 대학 Centre for Software Maintenance 방문연구원
1997년~1998년 미국 Purdue 대학교 방문 교수
1999년~현재 한성대학교 정보전산학부 부교수
관심분야 : 병렬 및 분산 프로그램 테스팅

창 병 모

e-mail : chang@cs.sookmyung.ac.kr
1988년 서울대학교 컴퓨터 공학과 학사
1990년 한국과학기술원 전산학과 석사
1994년 한국과학기술원 전산학과 박사
1994년~1995년 한국전자통신연구원 박사 후연수연구원
2000년~2000년 이탈리아 Verona 대학 방문 교수
1995년~현재 숙명여자대학교 부교수
관심분야 : 프로그램 분석, 객체-지향 프로그래밍, 컴파일러