KIPS Tr. Software and Data Eng.
Vol.7, No.1 pp.1~8 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2018.7.1.1

Applying Meta-Heuristic Algorithm based on Slicing Input Variables to Support Automated Test Data Generation

Hyorin Choi† · Byungjeong Lee††

ABSTRACT

Software testing is important to determine the reliability of the system, a task that requires a lot of effort and cost. Model-based testing has been proposed as a way to reduce these costs by automating test designs from models that regularly represent system requirements. For each path of model to generate an input value to perform a test, meta-heuristic technique is used to find the test data. In this paper, we propose an automatic test data generation method using a slicing method and a priority policy, and suppress unnecessary computation by excluding variables not related to target path. And then, experimental results show that the proposed method generates test data more effectively than conventional method.

Keywords : Model-Based Testing, Automated Test Data Generation, Slicing Technique, Meta-Heuristic Algorithm

테스트 데이터 자동 생성을 위한 입력 변수 슬라이싱 기반 메타-휴리스틱 알고리즘 적용 방법

최 효 린† · 이 병 정††

요 약

소프트웨어 테스트는 시스템의 신뢰도를 판단하는 중요한 작업이지만, 많은 노력과 비용이 요구된다. 모델 기반 테스트는 시스템 요구사항을 정형적으로 표현한 모델로부터 테스트 설계를 자동화함으로써 이러한 비용을 줄이기 위한 방안으로 제안되었다. 모델의 각 경로마다 입력 값을 생성하여 테스트를 수행하는데, 이 때, 적절한 입력 값을 찾기 위해 메타-휴리스틱 기법을 사용한다. 본 논문은 슬라이싱 기법과 우선순위 정책을 적용한 테스트 데이터 자동 생성 기법을 제안하며, 목적 경로와 관련이 없는 변수를 제외하여 불필요한 계산을 억제한다. 실험을 통해 기존의 기법보다 효과적으로 테스트 데이터를 생성함을 보인다.

키워드 : 모델 기반 테스트, 테스트 데이터 자동 생성, 슬라이싱 기법, 메타-휴리스틱 알고리즘

1. 서 론

개발된 소프트웨어가 요구사항에 맞게 잘 동작하는지 확인하는 것은 매우 중요한 작업으로, 시스템의 신뢰도를 판단하기 위한 기준이 된다. 그러나 개발된 시스템의 충분한 테스트를 수행하기 위해선 전체 개발 비용의 40%를 차지할 정도로 막대한 노력과 비용이 필요하다[1].

이러한 비용을 증대시키는 요인으로 비정형적인 언어로 작성된 요구사항으로 인한 테스트 설계의 어려움이 있다. 모델 기반 테스트(Model-based Testing, MBT)[2]는 시스템 요구사항을 UML, OCL 등 정형화된 표준 언어로 작성한 모델을 기반으로 테스트 설계 및 테스트 데이터 생성을 자동화하는 방안으로써 제안되었다. 모델을 분석하여 실행 가능한 경로를 찾고, 설정한 테스트 커버리지(Test Coverage)[2]에 따라 목적 경로를 생성한 후, 시스템을 임의의 입력 값으로 수행시켰을 때 탐색되는 경로와 일치시키는 과정을 통해 테스트를 수행한다. 각 입력 값을 생성하는 방법으로는 메타-휴리스틱(Meta-Heuristic)[3] 기법을 사용하는데, 대표적인 기법으로 유전 알고리즘(Genetic Algorithm, GA), 담금질 기법(Simulated Annealing) 등이 있다.

그러나 테스트하려는 시스템이 크고 복잡할수록 기본적인

※ 이 논문은 2016년도 서울시립대학교 연구년교수 연구비에 의하여 연구되었음.
† 준 회 원 : 서울시립대학교 컴퓨터과학과 석사
†† 정 회 원 : 서울시립대학교 컴퓨터과학과 교수
Manuscript Received : August 1, 2017
Accepted : September 6, 2017
* Corresponding Author : Byungjeong Lee(bjlee@uos.ac.kr)

알고리즘으로는 테스트 데이터를 생성하기 위해 많은 연산 시간을 필요로 하며, 지역 해 문제[3]에 쉽게 빠지게 되는 문제가 있다. 이를 극복하기 위해 기존의 메타-휴리스틱 기법을 개선하는 많은 연구들[4]이 제안되었지만, 이는 알고리즘 자체의 복잡도를 증가시켜 테스트 대상 시스템(System Under Test, SUT)에 대한 최적화 노력이 필요하다는 문제를 야기한다[5, 9].

본 논문은 앞서 언급된 문제들을 회피하면서, 동시에 성능 향상을 위한 새로운 방안을 제시한다. 제안하는 방법은 시스템의 실행 경로의 각 분기마다 관련된 변수들을 슬라이싱 기법[5]을 통해 추출하여, 이를 입력 값 조정 과정의 휴리스틱 계산에 사용함으로써 불필요한 연산을 줄여 알고리즘의 성능 향상을 꾀한다. 또한, 지역 해 문제를 최대한 회피하기 위해 3단계의 우선순위 정책을 적용하였으며, 실험을 통해 제안하는 방법이 기존 알고리즘보다 효과적으로 테스트 데이터를 생성할 수 있음을 보여준다. 본 논문이 기여하는 바는 다음과 같다.

- 본 논문은 모델에 기반한 테스트 데이터 자동 생성을 지원함으로써 테스트 비용 억제에 효과적이다.
- 실험을 통해 제안하는 방법이 다른 알고리즘보다 효율적으로 테스트 데이터를 생성함을 보인다.

2장에서는 관련 연구에 대해 기술하고, 3장에서는 제안하는 방법에 대해 설명한다. 4장 실험을 통해 다른 알고리즘과의 비교를 보고, 제안하는 방법의 효과성을 확인한다. 5장에선 본 논문에 대한 토의를 수행하였으며, 마지막으로 6장에서 결론을 내린다.

2. 관련 연구

이 장에서는 본 논문과 관련된 연구에 대해 기술한다.

먼저, 테스트 커버리지를 만족하기 위한 테스트 데이터를 자동으로 생성하는 방법으로 현재의 입력 값을 조정하면서 시스템의 특정 경로를 탐색하는 검색 기반 테스트(Search based Testing)[6, 7] 기법이 소개된다. Miller and Spooner [6]는 시스템의 경로에 기반을 둔 테스트 설계 기법을 최초로 제안하였으며, 이후 Korel[7]에 의해 휴리스틱 계산을 위한 적합도 평가(Fitness Function) 기법 등으로 확장되었다. 이는 본 연구에서 메타 휴리스틱 기법을 적용하기 위한 기반이 되며, 목표 커버리지를 만족하는 경로를 선택하여 메타-휴리스틱 알고리즘의 방향을 결정한다는 점에서 효율적으로 테스트 데이터를 생성할 수 있는 방법이다.

Eugenia[8]은 메타-휴리스틱 기법을 적용하여 테스트 데이터를 생성할 때, 지역 해 문제를 빠르게 벗어나기 위해 타부 탐색(Tabu Search) 기법을 사용하여 기존의 검색 기반 테스트 데이터 생성 기법을 개선하는 방법을 제안하였다. 탐색하는 과정 동안 타부 제약(Tabu Contraint)을 지정하여, 제한된 입력 도메인을 탐색함으로써 전역 해 집합을 찾을 확률을 높이는 방법이다. 그러나 개발자가 직접 해당 경로에 대한 타부 제약을 설계해야 한다는 점에서 현업에서 활용되기에 많은 노력이 요구되는 한계가 있다.

M. Harman[9]은 휴리스틱 기법을 사용하여 테스트 데이터를 생성할 때, 보다 효과적으로 계산하기 위해 코드 정적 분석을 기반으로 입력 값의 범위를 줄이는 방법을 제안하였다. 관련성이 적은 부분에 대해선 불필요한 계산을 억제한다는 점에서 본 논문과 유사한 방법이지만, 실제 시스템에서 수행되는 기능의 동작에 대해서는 고려하지 않아 가능한 모든 경로를 탐색하여 목적 경로로 지정하게 되는 문제가 있으며, 소스 코드에 의존적인 한계가 있다.

이외에도, 정적 분석 기법을 사용하는 방법으로 R. Khan [10]은 입력 도메인을 기반으로 K-Means 클러스터링 알고리즘을 통해 테스트 데이터의 추정 값을 찾고, 해당 값의 범위 내에서만 휴리스틱 계산을 수행하여 커버리지를 달성하는 방법을 제안하였다. 각 경로에 맞는 적정한 값의 범위를 먼저 추정한다는 점에서 효과적인 계산이 가능하지만, 추가적인 전처리 과정이 필요하며 복잡한 경로일 경우 기본적인 GA와 차이가 없어지는 한계가 있다.

Ayari[11]는 개미의 군집화를 모방한 알고리즘(Ant Colony Algorithm)을 적용한 테스트 데이터 자동 생성 기법을 제안하였다. 시스템의 각 경로는 종료 조건에 따라 간단하게 수행할 수 있는 경로와 여러 개의 중첩된 조건을 통해 수행하는 경로를 동시에 가지는데, 이들 중 오류가 발생할 확률이 높은 경로에 대해 휴리스틱 계산 과정에서 많은 자원(Ant)를 할당하여 시도해 보는 기법이다. 이는 특정 경로에 제한을 두지 않으며 시스템 전체적으로 결함이 발생하는 부분을 찾는 데에 효과적이지만, 탐색 과정에서 소스 코드에 기반을 둠으로써 프로그래밍 언어에 의존적이며 인터페이스 등 상호 작용을 고려하지 않아 일반적이지 않은 제어 흐름의 테스트 데이터를 생성하게 되는 문제가 있다.

3. 테스트 데이터 생성 방법

이 장에서는 본 논문이 제안하는 테스트 데이터 생성 방법에 대해 기술한다. 이해를 돕기 위해 간단한 구조를 가지는 예시 함수를 사용하도록 한다. 사용할 예시 함수는 Fig. 1과 같이 정리하며, 소스 코드는 Java로 작성되었고 모델은 UML의 활동 다이어그램(Activity Diagram)으로 표현하였다. 예시 함수 foo는 int 형태의 값 3개를 매개 변수로 받아 각 변수 i, j, k에 초기화하고, 조건식을 통해 변수 i, j의 값이 양수이면 변수 k에 추가하는 과정을 수행한다. 수행된 결과로 k의 값이 0 이상이면 k, 아닐 경우 0을 return한다.

```
1  int foo(int input1, int
2  input2, int input3){
3     int i = input1;
4     int j = input2;
5     int k = input3;
6     if( i > 0 ){ //조건 1
7        k = k + i;
8        if ( j > 0 ){//조건 2
9          k = k + j;
10       }
11    }
12    if( k > 0 ){//조건 3
13       return k;
14    }
15    return 0;
16 }
```

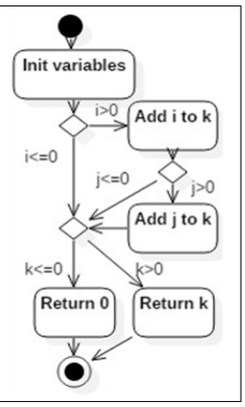

Fig. 1. Structure of an Example Function

3.1 입력 변수 슬라이싱

각 조건을 지나는 식을 '참(T)', '거짓(F)', 그리고 달성하기 위한 경로에 따라 조건 판단을 수행하지 않는 식을 '고려하지 않음(X)'이라고 표현할 때, 주어진 모델을 기반으로 생성할 수 있는 모든 경로는 표 1과 같다.

Table 1. All Executable Paths of Fig. 1.

Label	Path	Label	Path
P1	{ F, X, F }	P4	{ T, T, F }
P2	{ F, X, T }	P5	{ T, T, T }
P3	{ T, F, T }	P6	{ T, F, F }

만약, 분기 커버리지(Branch coverage)[2]를 달성하는 것이 목표이면, 목적 경로로써 {P1, P3, P5} 이나 {P2, P3, P4} 등을 선택할 수 있다. 제안하는 테스트 데이터 생성은 개발자가 설정한 목적 경로에 대해 적합한 입력 값을 찾기 위한 과정이다. 우선, 초기 입력 값을 임의로 생성한 후 이를 기반으로 테스트 대상 시스템을 실행시키면, 각 입력 값들이 탐색하는 경로를 실행 경로(Execution path)로 생성한다. 각 실행 경로가 목적 경로와 비교하여 얼마나 적합한지 평가한 후 입력 값의 변화를 통해 상이한 분기를 맞춰 나가야 한다. 예를 들면, 목적 경로가 'P3', 현재 실행 경로가 'P6'일 때, 상이한 분기점은 1개며, 조건 3식에 참(T)인 입력 값을 찾을 때까지 현재의 입력 값을 점진적으로 변화시킨다.

일반적인 유전 알고리즘에서는 입력으로 받은 변수 3개를 모두 고려하여, 'P3'에 맞는 입력 값을 찾는다. 그러나 각 조건에서 활용되는 변수를 '조건 변수'라 했을 때, 실제로 'P3'에서 조건 3의 조건 변수 'k'에 영향을 줄 수 있는 입력 값은 Line 5에서 사용된 'input3'과, Line 7에서 사용된 'input1'이다. 'input2'는 해당 분기를 해결하는 데에 영향을 주지 않는데, 이를 휴리스틱 계산에서 제외함으로써 성능 향상을 기대할 수 있다. 따라서 위와 같은 상황에서는 {input1, input3}을 연관된 '입력 값 집합(Input Value Set, IVS)'으로 생성하고, 이를 휴리스틱 계산에 활용한다면 보다 빠르게 전역 해를 찾음으로써 테스트 데이터 생성이 가능하다.

3.2 우선순위 정책

우선순위 정책이란 현재 실행 경로에서 목적 경로에 도달하기 위해 필요한 입력 변수 집합의 범위를 선택하는 방법이다. 실제 시스템에서 조건 변수와 입력 값과의 의존성을 슬라이싱 기법만으로 판단하는 것은 IVS가 전체 입력 값과 동일해 지거나, 제한된 변인으로 인한 지역 해 문제에 빠지게 되는 문제가 있다. 이를 극복하기 위한 방법으로 본 논문은 IVS 선택을 위한 3단계의 우선순위를 제안한다. 일정 반복 횟수 이상 더 좋은 해를 찾지 못한다면 사용되는 IVS가 목적 경로를 해결하기에 적절하지 않다고 보고, 더 다양한 변수를 고려하기 위해 IVS의 범위를 확장한다. 제안하는 단계는 Table 2에서 정리한다.

우선순위 선택은 기본적으로 1단계부터 시작한다. 조건 변수와 직접적으로 연관성이 있는 입력 변수 집합이며, 위와 같은 상황에서는 조건 변수 k에 직접 할당된 입력 변수 값인 input3이 1단계 입력 변수 집합이다. 다음 단계로는 제안하는 슬라이싱 기법을 적용한 IVS를 선택하는 단계로, 현재의 실행 경로에서 목적 경로와 상이한 분기점으로부터 슬라이싱 기법을 통해 조건 변수에 변화를 주는 변수를 IVS로 생성한다. 위와 같은 상황에서는 'P6'에서 조건 3의 k값에 영향을 줄 수 있는 입력 변수는 직접적인 영향을 주는 1단계에 Line 7의 변수 i가 추가적으로 포함된다. 이는 각각 Line 3, 5에서 입력 변수 input1, input3로 인해 변화되는 값이며 이를 IVS로 가진다. 마지막으로 3단계는 2단계의 IVS으로도 적절한 테스트 데이터를 찾을 수 없을 때, 목적 경로와 관련된 입력 변수를 고려하는 단계이다. 이는 기존 GA방식과 동일하며, 가능한 모든 경우를 다 수행함으로써 지역 해 문제를 벗어나는 데에 사용된다.

Table 2. Priority Steps for Selecting IVS

Step	Description	Example
Level 1	IVS directly associated with the condition variable	{ input3 }
Level 2	IVS that affects change of k value by slicing the current execution path	{ input1, input3 }
Level 3	All IVSs related to the Target path	{ input1, input2, input 3 }

해당 목적 경로의 해를 찾기 위해 알고리즘을 반복한 횟수를 ITARGET, 더 나은 해를 찾지 못하는 상태로 반복한 횟수를 IFROZEN이라 할 때, 우선순위 단계를 판단하기 위한 조건식은 다음과 같이 정리한다.

$$F(p) = \frac{I_{TARGET} - I_{FROZEN}}{I_{TARGET}} \quad (1)$$

F(p)는 우선순위를 선택하기 위한 계산에 사용되는 식으로, 총 반복 횟수(ITARGET) 대비 지역 해 문제에 빠진 후 반복한 횟수(IFROZEN)의 비율을 구한 후, 설정한 임계치 값에 따라 단계를 선택한다. 이는 Equation (2)와 같이 정리한다.

$$IVS_{LEVEL} = \begin{cases} 1, & I_{FROZEN} < R \\ 2, & I_{FROZEN} > R,\ \theta < \mathrm{F(p)} \\ 3, & I_{FROZEN} > R,\ \theta \geq \mathrm{F(p)} \end{cases} \quad (2)$$

각 단계는 일정한 값 R에 따라 설정된다. 지역 해 문제에 빠지지 않은 경우는 아직 해당 IVS가 효과적으로 탐색하고 있음을 의미하므로, 1단계를 선택한다. 하지만, 일정 반복(R) 이상 더 좋아지지 않은 상태에 있으면 다음 단계로 넘어가기 위해 F(p) 계산을 수행하며, 미리 설정한 임계치 값 θ에 따라 결과 값이 낮으면 2단계, 높으면 3단계를 적용한다.

3.3 진화 알고리즘 적용 방법

이 절에서는 본 기법을 데이터 조정을 위해 진화 알고리즘에 적용하는 방법을 기술한다. 입력 값으로 Table 4의 설정 파라미터들을 받으며 모든 목적 경로에 대한 입력 변수의 해 집합 S를 반환하는 알고리즘으로, 3.1절과 3.2절에서 기술한 슬라이싱 기법 및 우선순위 정책을 GA에 적용하여 의사 코드로 나타내었으며 Fig. 2와 같이 정리한다.

```
    Input the preset values shown in Table 4.
    Output all solution set S for all TP
1   Generate all target paths TP(m) = {TP1,…,TPm}
2   Generate an initial population P(n) = {P1, …, Pn}
3   Loop while not all solution set S found yet
4     Generate all execution paths X(n) = {x1, …, xn}
5     Find a solution xj from X
6     If xj is fit for TPi Then
7       Add xj to S, Continue;
8     End If
9     Set input variable set IVSLEVEL with TPi
10    Create a childhood population CH(n)
11    For k = 1 to n Do
12      Select two solution xa, xb from X
13      xe ← Crossover(xa, xb, IVSLEVEL)
14      CHk ← Mutate(xe, IVSLEVEL)
15    End Do
16    Evaluate CH by using fitness function F
17    Replace(P, CH)
18  End Loop
    Terminate Algorithm
```

Fig. 2. Algorithm Applying Slicing Input Variables (SV)

정리한 의사 코드는 크게 세 부분으로 구성된다. 먼저, Line 1-2에서는 휴리스틱 계산을 위한 초기 설정 부분으로, 시스템의 목적 경로(TP)을 설정하고 임의의 입력 값으로 모집단 P를 생성한다. m은 생성된 목적 경로의 수, 그리고 n은 입력 값으로 받은 모집단의 크기이다.

Line 3부터는 모든 목적 경로에 대한 적절한 입력 값(S)을 찾을 때까지 반복문을 시작한다. Line 4-8까지는 생성된 입력 값 집합(P)으로 시스템을 실행시킨 경로 집합(X)에서 목적 경로와 일치하는 값(xj)을 S에 추가한다. 더 일치하는 경로가 없을 경우, 현재 생성된 P 값을 조정하기 위한 Line 9-17까지의 휴리스틱 계산을 시작한다.

Line 9에서는 현재 목적 경로에 대해 우선순위 정책을 적용하여 IVS를 선택한다. 그리고 Line 10에서 조정된 입력 값 집합을 저장하기 위해 새로운 입력 값 집합 CH를 생성한다. CH는 유전 알고리즘에서 사용되는 자식(Childhood) 개념으로, 부모 세대인 P로부터 우수한 유전자를 교차(Crossover) 또는 변이(Mutate) 연산을 통해 물려받는 과정을 통해 휴리스틱 계산을 수행한다. Line 11-15는 모집단의 크기(n) 만큼 데이터 조정 과정을 수행하며, 생성된 결과를 각 CH에 할당한다.

반복이 끝나면, Line 16에서 결과 집합 CH가 부모 집합 P보다 더 좋은 해를 가지는지 확인하기 위한 적합도 평가를 수행하는데, 사용하는 함수는 Korel[7]의 방법을 적용한다. 찾아낸 데이터가 P보다 더 좋은 해를 가질 경우, Line 17에서 P의 해당 값을 대체(Replace)하여 다시 재구성한다. 이러한 과정을 반복하여 모든 경로에 대한 입력 값을 찾으면, 알고리즘을 종료한다.

4. 실 험

4.1 실험 환경

본 실험은 휴리스틱 계산을 수행하는 여러 알고리즘들을 기반으로 제안하는 기법을 적용하였을 때와 비교하는 과정을 수행한다. 실험에 사용되는 알고리즘은 가장 기본적인 언덕오르기(Hill Climbing, HC)[2] 기법 이외에 유전 알고리즘(GA)[2, 3], 유전 담금질 기법(Genetic-Simulated Annealing, GSA)[3]이 사용되며, 이를 기반으로 본 논문이 제안하는 기법(GA/SV, GSA/SV)을 추가적으로 적용하였다. 테스트 대상 시스템으로는 총 6개의 시스템을 사용한다. 동일한 실험 환경을 조성하기 위해 모든 프로그램은 Java언어로 작성하였고, 실험 도구로 Eclipse를 사용하였다.

Table 3은 본 실험에 사용되는 SUT를 요약하여 나타내었다. S1은 날짜를 입력받아 윤년 여부와 요일 확인 등을 수행하는 validDate 함수이다. S2-S4는 삼각형의 형태를 판단하는 시스템으로, 테스트 데이터 생성에서 제안한 알고리즘의 효과성을 보이기 위해 사용된다[5, 13-15]. S5-S6 시스템은 오픈소스 프로젝트[16]로, 은행 업무 처리와 계좌 이체 등을 수행할 때 사용되는 checkPIN, validWiring 함수를 분석하여 활용하였다. 각 SUT의 분기 수(Branches)와 생성한 목적 경로, 매개변수(Parameters)의 형태를 정리하였으며, 근사 도메인 크기(Approximate domain size)란 매개변수를

Table 3. System Under Test (SUT)

Label	Function name	Branches	Target Paths	Parameter	Approximate domain size(10^n)
S1	validDate[12]	14	11	int: 4	7
S2	triangle2[13]	20	8	int: 3	14
S3	triangle3[5,14]	20	8	int: 3	14
S4	triangle4[15]	26	14	int: 3	14
S5	SWIFTNET_checkPIN[16]	8	13	int: 7	13
S6	SWIFTNET_validWiring[16]	13	18	int: 11	22

Table 4. Setting the Main Parameters of All Algorithm

Algorithm	Parameter	Value
GA, GSA, GA/SV, GSA/SV	Population size	N=50
	Crossover probability	CP=0.99
GSA, GSA/SV	Temperature	T=1000
	Neighborhood size	b=50
	Cooling ratio	r=0.95
GA/SV, GSA/SV	Priority level threshold	Θ=0.85

통해 생성될 수 있는 입력 값의 범위를 10의 배수로 표현한 값이다. 모든 실험은 Intel Xeon CPU X5550(x2), 2.67GHz, 12GB RAM에서 수행하였다.

Table 4는 각 알고리즘에서 사용되는 주요한 매개변수를 정리하였다. 유전 알고리즘을 사용하는 모든 기법들은 모집단 크기(Population size)와 교차 확률(Crossover Probability)을 가지며, 담금질 기법은 온도(Temperature)와 이웃 범위(Neighborhood size), 그리고 알고리즘 종료 조건을 위한 냉각 비율(Cooling ratio)을 사용한다. 또한, 본 기법은 우선순위 정책을 위한 임계값(Priority level threshold)이 있다. 설정한 값은 관련된 연구[3, 7, 17]에서 실험에 사용된 값을 참고하여 설정하였으며, 우선순위 임계값은 경험적으로 우수한 결과를 보인 값을 설정하였다.

4.2 실험 결과

실험은 총 3가지 관점에서 수행하였다. 첫 번째로 분기 커버리지의 90%를 달성하기 위해 알고리즘을 수행한 시간을 비교한다. 이는 동일한 SUT에서 수행 시간이 적을수록, 효과적인 알고리즘임을 의미한다. 총 20회 수행한 평균을 냈으며, 단위는 초(second), 소수점은 첫 번째 자리에서 생략한다. 결과는 Fig. 3과 같이 그래프로 정리하였다.

결과를 살펴보면, 본 논문에서 제안하는 방법을 적용함으로써 GA나 GSA기법의 전반적인 수행 시간이 유의미한 수준으로 적어지는 것을 확인할 수 있다. 특히, GA의 경우 S1과 S4 시스템에서 기존의 GA와 비교해 각각 26%, 20% 이상 빨라졌으며, GSA의 경우에도 S6 시스템 비교였을 때 약 20% 이상 개선되었음을 확인할 수 있다. 눈여겨 볼만한 점은 S4의 경우 GSA기법이 GA기법보다 오히려 수행 시간이 많았는데, 이는 지역 해 문제로 인해 소비되는 시간이 많았기 때문이다. 하지만 제안하는 기법(GSA/SV)에서 우선순위 정책을 적용함으로써 이를 효과적으로 개선한 결과를 보였다. 언덕 오르기 알고리즘에서 100초를 초과하는 부분은 그림 상에서 표현하지 않았다.

또한, 본 기법은 시스템에서 사용되는 입력 변수의 수에 따라 성능에 영향을 받는다. 슬라이싱 기법을 통해 불필요한 계산을 줄임으로써 테스트 데이터를 생성하는 과정에서 성능 향상을 도모하는데, 이는 입력 변수의 수가 많을수록 더 큰 효과를 볼 수 있다. 위 실험 결과를 분석하면, 사용된 SUT에 따라 기존 알고리즘(GA, GSA)과 제안하는 기법(GA/SV, GSA/SV)이 적용한 수행 시간의 차이(%)를 비교하였을 때 Fig. 4와 같은 그래프로 나타낼 수 있다.

분석된 결과를 살펴보면, 입력 변수의 수가 많아질수록 차이가 점차 커지는 것을 알 수 있다. S2. S3. S4 시스템의 경우, 입력 변수는 3개이며 GA의 경우 제안하는 기법과의 차이가 약 15% 임을 보이는데, 본 실험에 사용된 SUT 중

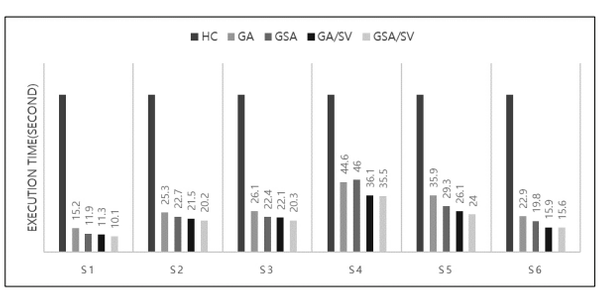

Fig. 3. Comparison of Execution Time

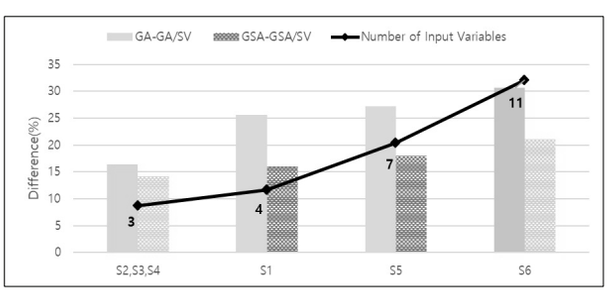

Fig. 4. Comparison of Execution Time Difference

Table 5. Maximum Coverage Achieved by Algorithm

SUT	HC	GA	GSA	GA/SV	GSA/SV
S1	100%(136s)	100%(43s)	100%(29s)	100%(20s)	100%(19s)
S2	100%	100%(77s)	100%(61s)	100%(45s)	100%(40s)
S3	100%	100%(86s)	100%(58s)	100%(53s)	100%(47s)
S4	77%	100%(113s)	100%(80s)	100%(77s)	100%(70s)
S5	100%	100%(101s)	100%(88s)	100%(80s)	100%(76s)
S6	100%(147s)	100%(85s)	100%(65s)	100%(58s)	100%(55s)

가장 많은 입력 변수를 가지는 S6 시스템은 약 30%로, 두 배 가까이 차이를 보인다. 이는 입력 변수가 많고 복잡한 시스템일수록 본 기법이 효과적으로 적용이 가능함을 의미한다. 또한, GA에 적용한 기법(GA/SV) 간의 차이가 GSA에 적용한 차이(GSA/SV)보다 더 많은 차이를 보이는데, 이는 두 알고리즘이 실행될 때 기본적으로 수행되는 작업에서 GSA가 GA보다 담금질 기법을 위한 시간이 추가적으로 소모되며, 다음 세대로 넘어가는 시간의 차이 때문에 우선순위 정책에 영향을 주기 때문이다.

두 번째로, 일정한 단위 시간마다 각 알고리즘의 평균 분기 커버리지를 비교한다. 시간의 단위는 5초, 커버리지는 4개의 SUT에 대한 평균 분기 커버리지 값으로 작성하였다. 결과는 Fig. 5와 같다.

결과를 살펴보면, 본 논문에서 제안하는 방법이 여타 알고리즘보다 커버리지를 빠르게 달성하고 있음을 확인할 수 있다. 특히 0~5초 구간에서 다른 알고리즘과 비교하여 약 20% 이상 차이를 보이는데, 이는 실험에 사용된 SUT의 각 경로에서 불필요한 연산을 제외하여 빠르게 전역 해를 찾았음을 의미한다. 또한, 본 실험에 사용된 기법들 중에서는 GSA/SV가 100%의 커버리지를 가장 먼저 달성하는 알고리즘으로 나타난다. GA와 GSA의 그래프는 유사한 형태를 보이지만, 10초 이후부터는 GSA가 좀 더 빠르게 커버리지를 달성하는 것을 볼 수 있다. 언덕 오르기 기법의 경우, 0~5초 구간에서 다른 알고리즘과 비슷한 수준의 커버리지를 달성하지만, 이는 초기 값을 임의의 값으로 생성하는 과정에서 다른 알고리즘과 크게 다르지 않기 때문이다. 시간이 지날수록 달성한 커버리지의 차이가 급격하며, 이는 언덕 오르기 알고리즘이 본 연구에는 적합하지 않음을 의미한다.

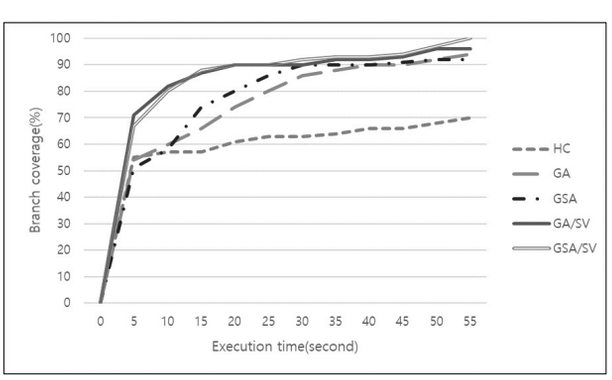

Fig. 5. Average Branch Coverage Comparison

마지막으로 각 알고리즘을 충분한 시간 동안 수행할 경우 달성하는 최대 분기 커버리지와 평균 수행 시간을 기록하였다. 총 180초 간 수행시키고, 이를 30회씩 반복한 후 평균 소요 시간(초)을 기록하여 Table 5에 정리하였다. 결과가 평균 160초 이상이면, 표기를 생략하였다.

정리된 결과를 살펴보면, 대부분의 알고리즘에서 100%를 달성할 수 있었음을 볼 수 있다. 이는 본 실험에 사용된 각 SUT의 목적 경로들이 모두 수행 가능한 경로임을 의미한다. 그러나 언덕 오르기 알고리즘에서 S4 시스템의 최대 커버리지를 77% 달성하는데, 이는 S4의 경로들이 특정한 조건을 만족하여야 수행이 가능한 목적 경로를 가지고 있기 때문에, 매 수행 마다 값이 일정한 방향으로 변화하는 알고리즘으로는 매우 많은 시간이 필요하기 때문이다.

5. 토 의

본 실험에서 수행된 모든 결과를 종합하면, 제안한 기법이 기존 메타-휴리스틱 기법보다 약 20% 이상 실행 시간을 단축하며, 입력 변수가 많은 시스템일수록 빠르게 테스트 데이터를 생성함을 확인하였다. 이를 통해 복잡하고 많은 변수들이 사용되는 현업의 시스템에서 테스트 데이터를 생성할 때, 보다 효과적으로 활용 가능하리라 기대된다.

또한, 본 실험에선 입력 변수로 Int 자료형을 사용하였다. 이는 적합도 평가 방법인 변수 간 분기 거리 계산을 용이하게 하기 위함으로, 분기 거리 계산이 가능한 자료형은 제한 없이 적용이 가능하다. 하지만 String 등 문자형을 사용하는 경우 해당 평가 방법만 사용하기엔 한계가 있으며, 사용할 자료형에 따라 추가적인 적합도 평가 방법을 적용한다면 이를 극복하리라 기대한다.

본 논문이 제안하는 방법은 기존의 메타-휴리스틱 알고리즘에 추가적으로 입력 변수 슬라이싱과 우선순위 정책을 적용함으로써 효과를 보였다. 이는 SUT에 대한 최적화 노력이 필요 없고, 휴리스틱 계산을 수행하는 다양한 알고리

즘에서 적용 가능함을 의미한다. 본 실험에서는 이를 증명하기 위해 GA 및 GSA에 각각 제안한 기법을 적용하여 유의미한 성능 향상을 보였다. 그러나 제안한 기법 중 우선순위 정책을 적용할 때, 실제로 효과를 보이는 경우 외에는 오히려 임계값 계산을 위해 시간을 소모하게 되는 한계가 있었다. 본 논문에서는 경험적으로 우수한 결과를 보이는 값을 임의로 지정하였지만, 복잡한 경로일수록 임계값을 조정하여 빠르게 다음 단계의 우선순위를 선택하도록 변경하는 방법을 통해 개선되리라 기대한다.

6. 결 론

본 논문은 모델을 기반으로 테스트 설계를 수행하는 과정에서 각 목적 경로를 수행하는 입력 값을 자동으로 찾아내는 알고리즘을 제안한다. 기존에 사용되던 메타-휴리스틱 기법에 추가적으로 제안하는 두 기법을 적용하여, 불필요한 계산을 줄이는 과정을 통해 알고리즘 수행 시간을 단축하였으며, 실험을 통해 본 기법의 우수함을 보인다.

또한, 제안하는 방법이 입력 변수에 따라 알고리즘의 성능이 향상됨을 보였으며, 이는 곧 다양한 변수를 활용하는 현업의 시스템에서 본 기법을 효과적으로 적용할 수 있음을 의미한다. 향후에는 다양한 시스템에서 제안하는 방법을 적용하기 위해 추가적으로 필요한 요소를 분석하여 사용성을 고려한 도구로 개발할 예정이다.

References

[1] P. C. Jorgensen, “Software Testing: a Craftsman′s Approach,” CRC press, pp.3-76, 2013.

[2] M. Utting and B. Legeard, “Practical Model - Based Testing: A Tools Approach,” Morgan Kaufmann, 2010.

[3] P. R. Srivastava, V. Ramachandran, and M. Kumar, “Generation of test data using meta heuristic approach,” *TENCON 2008-2008 IEEE Region 10 Conference*, pp.1-6, 2008.

[4] D. B. Mishra, R. Mishra, K. N. Das, and A. A. Acharya, “A Systematic Review of Software Testing Using Evolutionary Techniques,” In *Proc. of Sixth International Conference on Soft Computing for Problem Solving*, Singapore, 2017.

[5] McMinn, Phil, “Search-based software test data generation: A survey,” Software Testing Verification and Reliability, 2004.

[6] W. Miller, and D. L. Spooner, “Automatic generation of floating-point test data,” In *Proc. of IEEE Transactions on Software Engineering*, Vol.2, No.3, pp.223-226, 1976.

[7] B. Korel, “Automated software test data generation,” *IEEE Transactions on Software Engineering*, pp.870-879, 1990.

[8] E. Díaz, J. Tuya, and R. Blanco, “Automated software testing using a metaheuristic technique based on tabu search,” In *Proc. of 18th IEEE International Conference on Automated Software Engineering*, pp.310-313, 2013.

[9] M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, and J. Wegener, “The impact of input domain reduction on search-based test data generation,” In *Proc. of the the 6th Joint Meeting of the European Software Engineering Conference*, ACM, 2007.

[10] R. Khan and M. Amjad, “Automatic generation of test cases for data flow test paths using K-means clustering and generic algorithm,” *International Journal of Applied Engineering Research*, Vol.11, No.1, pp.473-478, 2016.

[11] K. Ayari, S. Bouktif, and G. Antoniol, “Automatic mutation test input data generation via ant colony,” In *Proc. of the 9th Annual Conference on Genetic and Evolutionary Computation*, pp. 1074-1081, 2007.

[12] S. Back, H. Choi, J. W. Lee, and B. Lee, “Evolutionary Test Case Generation from UML-Diagram with Concurrency,” In *International Conference on Computer Science and its Applications*, pp.674-679, 2016.

[13] H. C. Wang, “A hybrid genetic algorithm for automatic test data generation,” Master′s thesis, Sun Yat-sen University, 2006.

[14] C. C. Michael, G. McGraw, and M. A. Schatz, “Generating software test data by evolution,” *IEEE Transactions on Software Engineering*, Vol.27, No.12, pp.1085-1110, 2001.

[15] B. F. Jones, H. H. Sthamer, and D. E. Eyres, “Automatic structural testing using genetic algorithms,” *Software Engineering Journal*, Vol.11, No.5, pp.299-306, 1996.

[16] F. A. Aderohunmu, G. Paci, and L. Benini, “SWIFTNET: A data acquisition protocol for fast-reactive monitoring applications,” *SIES 2013 8th IEEE International Symposium*, pp.93-96, 2013.

[17] C. C. Michael, G. E. McGraw, M. A. Schatz, “Genetic algorithms for dynamic test data generation,” In *Proc. of Automated Software Engineering, 12th IEEE International Conference*, pp.307-308, 1997.

최 효 린

http://orcid.org/0000-0002-8642-7421

e-mail : yoinoichr2015@uos.ac.kr

2015년 한국교통대학교 소프트웨어전공 (학사)

2017년 서울시립학교 컴퓨터과학과(석사)

관심분야 : 소프트웨어 테스트, 소프트웨어 진화

이 병 정

http://orcid.org/0000-0002-2750-7608

e-mail : bjlee@uos.ac.kr

1990년 서울대학교 계산통계학과(학사)

1998년 서울대학교 전산과학과(석사)

2002년 서울대학교 전기·컴퓨터공학부 (박사)

1990년~1998년 ㈜ SK하이닉스 연구원

2002년~현 재 서울시립대학교 컴퓨터과학과 교수

관심분야 : 소프트웨어 테스트, 소프트웨어 진화, 소프트웨어공학