GUI Test Suite Generation using Ant Colony Optimization with EFG Traversal Criterion

1. Introduction

최근의 많은 프로그램들이 그래픽 유저 인터페이스(이하 GUI)를 사용함에 따라, GUI 테스트는 날이 갈수록 중요해지고 있다. 현대 GUI는 매우 interactive하기 때문에, 모든 경우의 수에 대해서 테스트하는 것은 불가능에 가깝다. 대신 가능한 이벤트 경로 중 유의미하면서 다른 이벤트 경로들을 포함할 수 있는 것을 찾아내어 테스트하는 방식을 사용하고 있다.

보통 GUI 테스트 케이스를 만드는 것은 사람의 반복적인 노동 작업으로 여겨진다. 의미 있는 테스트 케이스를 만드는 과정을 자동화하는 것이 어렵기 때문이다. 그렇기에 GUI 테스트 케이스 생성 자동화는 상당히 도전적인 주제로 생각되었다.

GUI를 테스팅하기 위한 수많은 criteria가 존재하고, test case를 criteria에 최적화하여 생성하는 방법론이 제시되었다. 우리는 하나의 GUI 프레임워크에 대해 criteria를 변형 적용해 보면서 GUI test case를 생성해보기로 하였다. 주어진 EFG로부터 Ant Colony Optimization을 이용하여 GUI test case를 형성하고, 그 결과를 비교 분석하는 것을 프로젝트 목표로 한다.

2. Preliminaries

애플리케이션의 복잡한 GUI를 이벤트를 기준으로 추상화한 것을 이벤트 흐름 그래프(Event Flow Graph)라고 한다. 이벤트 흐름 그래프는 GUI 이벤트를 정점으로, 이벤트의 선행 여부를 간선으로 가지는 유향 간선 그래프(directed edge graph)이다. 예를 들어, a → b가 그래프에 포함된다는 것은 이벤트 a가 실행된 이후 b가 실행 가능하다는 의미이다.

한편 EFG 그래프를 cover하는 테스트 케이스를 만드는 것이 완전히(complete) 더 좋은 GUI 테스팅을 가능하게 하는 것은 아니다. 예로, 문서 편집 프로그램의 EFG에서 (시작) → Edit → Paste와 같은 이벤트 sequence가 허용된다. Copy 명령을 실행하지 않고 Paste를 실행하는 것은 오류를 일으키지 않지만(feasible), 의미가 없는(not meaningful) 이벤트이다. 그럼에도 불구하고, EFG 그래프를 덮는(cover) 것은 '대략적으로' 대부분의 이벤트 순열들을 검증한다는 데에서 충분한 의미를 가지고, GUI로부터 complete한 EFG를 생성하는 것은 test

case generation 문제를 해결할 수 있도록 돕는다.

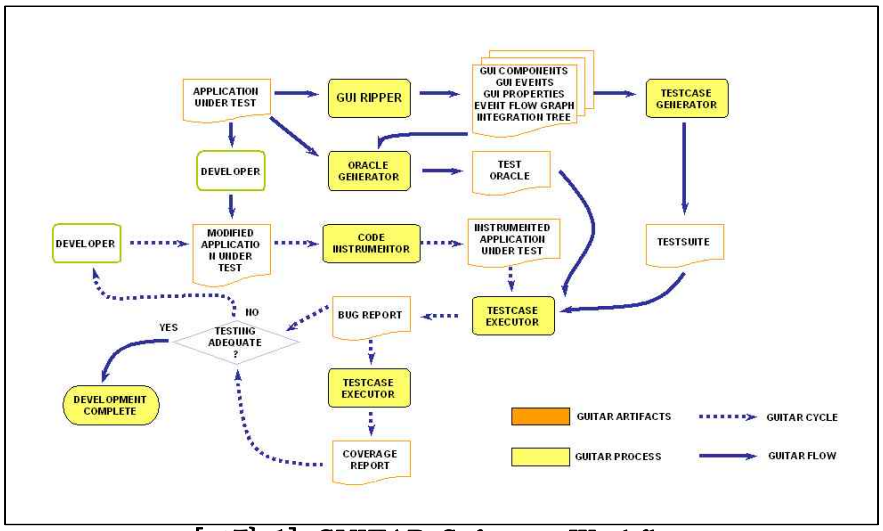
3. Related Works

Memon et al[1]은 다양한 기준의 GUI testing coverage criteria를 소개하였다. 이 중에는 EFG coverage(node, edge의 커버율), fixed length event coverage 등의 intra-component coverage와 invocation coverage, invocation-termination coverage 등의 inter-component coverage를 포함하고 있다. 이들은 논문을 통해 위의 coverage 기준들을 측정하는 방안을 소개하고 있다.

McMaster, Memon[2]은 Maximum call tree(이하 MCT)를 사용한 GUI 테스팅 기준을 제시한다. 이들은 call tree는 하나의 함수를 실행할 때 파생되는 함수들의 hierarchy로 정의하며, call tree의 단말 노드를 최대화하는 criteria로 test sequence를 생성하는 방법을 소개하고 있다.

Bauersfeld et al[3], [4]는 위의 MCT criteria를 바탕으로 Ant Colony Optimization(이하 ACO)를 적용하여 test sequence를 생성하는 방법을 선보였다. 이들은 random sequence generation에 비해 ACO가 MCT criteria에 부합하는 sequence를 생성할 수 있음을 보였다.

B. Nguyen et al[5]이 구현한 GUITAR:An Innovative Tool for Automated Testing of GUI-Driven Software는 다양한 모델을 기반으로 한 GUI 테스트 tool이다. 이는 GUI Ripper, Testcase Generator와 같은 tool chain을 바탕으로 새로운 workflow[그림 1]를 만들고, GUI 테스트를 수행할 수 있는 다양한 측정 도구를 연결할 수 있게 제작되었다.



[그림 1] GUITAR Software Workflow

[그림 1]의 작업 흐름에서 볼 수 있듯이, 개발자는 GUITAR process tool chain으로 다양한

GUI 테스트 기능을 실행할 수 있다. GUITAR의 주요한 기능으로 AUT를 통해 GUI를 분석하고 해당 GUI의 label, value, name들을 xml형태로 제공하는 것, 그리고 분석된 GUI 데이터를 바탕으로 EFG 를 제공하는 것 등이 있다.

4. Algorithms

기본적으로 Bauersfeld et al[3]이 제시한 ACO를 바탕으로 알고리즘을 설계하였다. 하지만 fitness를 계산하는 데 사용한 metric에서 차이가 있다. 우리는 MCT metric을 사용하지 않고, EFG를 탐색하면서 sequence에 이웃한 이벤트 수를 바탕으로 fitness를 계산한다. 뻗어나갈 가지가 많은 test case는 더 많은 code를 탐색할 수 있기 때문이다.

Pseudocode 1. ACORun(popSize, generation, seqLen, filename)

```
Output: Sequence that has maximum fitness value
begin
        g ← EventFlowGraph(filename) ;
        p ← Pheromone() ;
        best ← [] ;
        bestValue = -1 ;
        antList ← Ant[popSize] ;
        for gen = 1 to generation do
                for i = 1 to popSize do
                        antList[i].traverse(seqLen, g, p) ;
                        antList[i].calculateFitness(g) ;
                        if bestValue < antList[i].fitness then
                                bestValue ← antList[i].fitness ;
                                best ← antList[i].sequence ;
                p.update(antList)
        return best
end
```

Pseudocode 1.의 ACORun은 세대 수(generation)만큼 ACO를 실행하며, 각 세대마다 ant들이 탐색한 sequence의 fitness를 비교하여 가장 큰 fitness를 갖는 sequence를 반환한다.

Pseudocode 2. Ant.traverse(seqLength, graph, pheromone)

```
Output: Assign ant's traversal sequence according to pheromone
begin
        self.sequence ← [graph.initNode()] ;
        for i = 2 to seqLength do
                fromNode ← self.sequence[i-1] ;
                self.sequence[i] ← selectAction(fromNode, graph, pheromone) ;
end
```

Pseudocode 2.에서는 ant가 EFG와 그래프 위에 뿌려진 pheromone을 따라 test sequence를 생성한다. 각 ant가 다음 action을 정하는 방식은 -pseudo random proportional rule을 적용하였다.

Pseudocode 3. Ant.calculateFitness(graph)

```
Output: Assign ant's fitness value according to its sequence
begin
        isVisit ← [ False ] * graph.size ;  // Node visit flag
        self.fitness ← 0 ;
        for i = 1 to self.sequence.length do
                fromNode ← self.sequence[i] ;
                for j = 1 to graph.size do
                        if (graph.Matrix[fromNode][j]) == 1 and (~isVisit[j]) then
                                self.fitness ← self.fitness + 1 ;
                                isVisit[j] ← True ;
end
```

Pseudocode 3.은 ant의 새로운 fitness를 계산한다. Ant의 sequence를 따라 탐색하면서 각 이벤트의 이웃 이벤트 개수를 합한다. $(a \rightarrow b \rightarrow a \rightarrow b \cdots)$와 같이 sequence가 동일한 이웃을 가진 EFG의 일부에서 맴돌 가능성이 있기 때문에, 같은 이웃 이벤트는 한번만 세었다.

$$\tau[i][j] \leftarrow (1-\alpha) \times \tau[i][j] + \alpha \times Avg\{\text{ant}.fitness \mid (i \rightarrow j) \in \text{ant}.sequence\} \qquad \textbf{[식 1]}$$

Pheromone의 업데이트 함수로 [식 1]을 사용하였으며, 갱신하고자 하는 pheromone의 edge가 포함된 ant. sequence들의 fitness를 평균하여 적용한다.

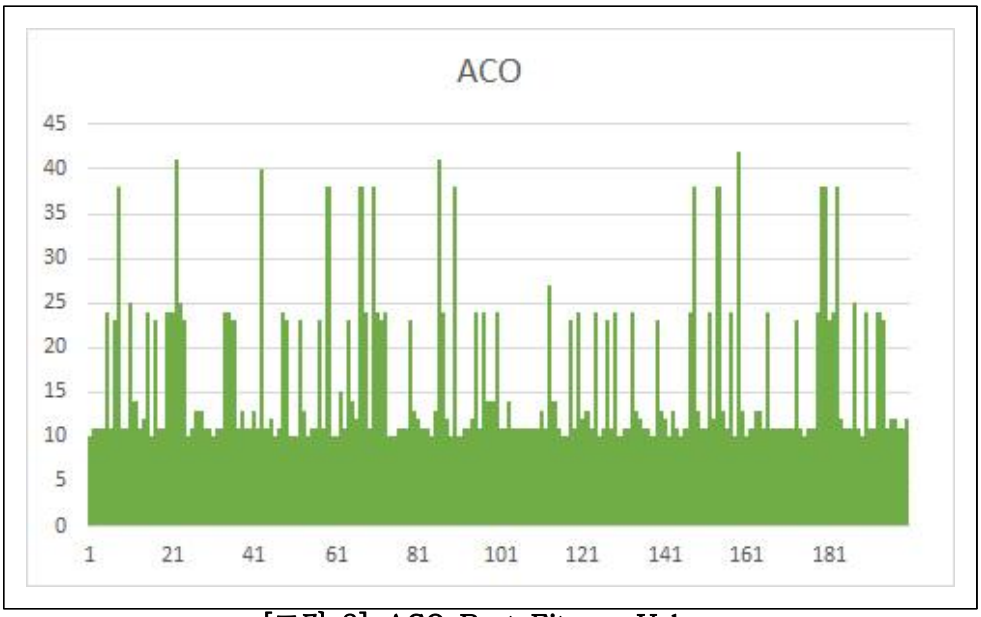
5. Evaluation

실험에 사용된 parameter의 값은 다음 [표 1]과 같다.

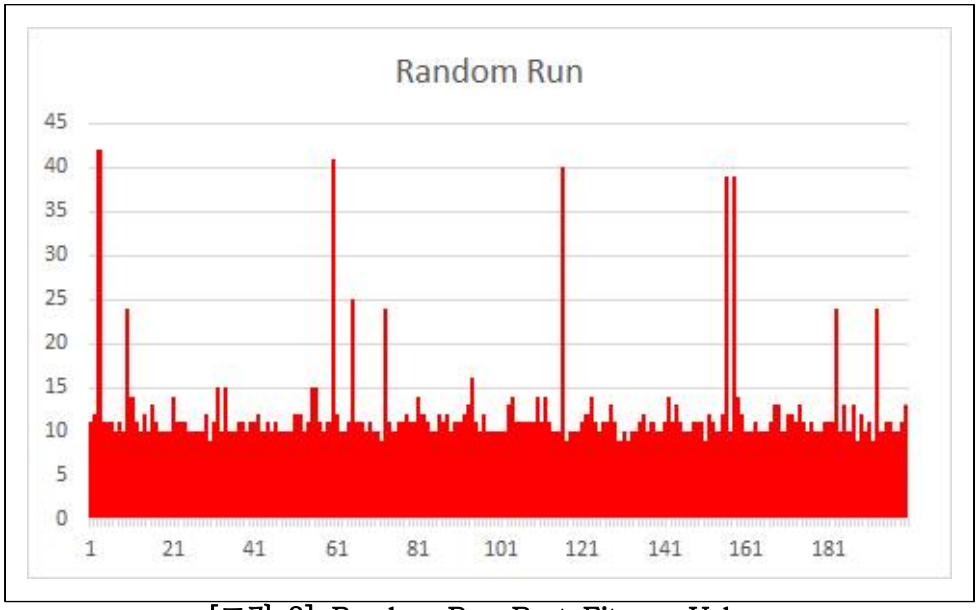
Parameter	Description	Value
popSize	Ant Population Size	20
generation	Number of Generation	200
seqLength	Sequence Length	5
ρ	Pseudo-Random Proportional Rule Selection Probability	0.7
α	Pheromone Evaporation Rate	0.5
initP	Pheromone Initial Value	10

[표 1] ACO Parameter Value

이를 GUITAR 1.2버전에서 기본적으로 제공된 EFG에 적용한 결과는 다음 [그림 2, 3]과 같다.

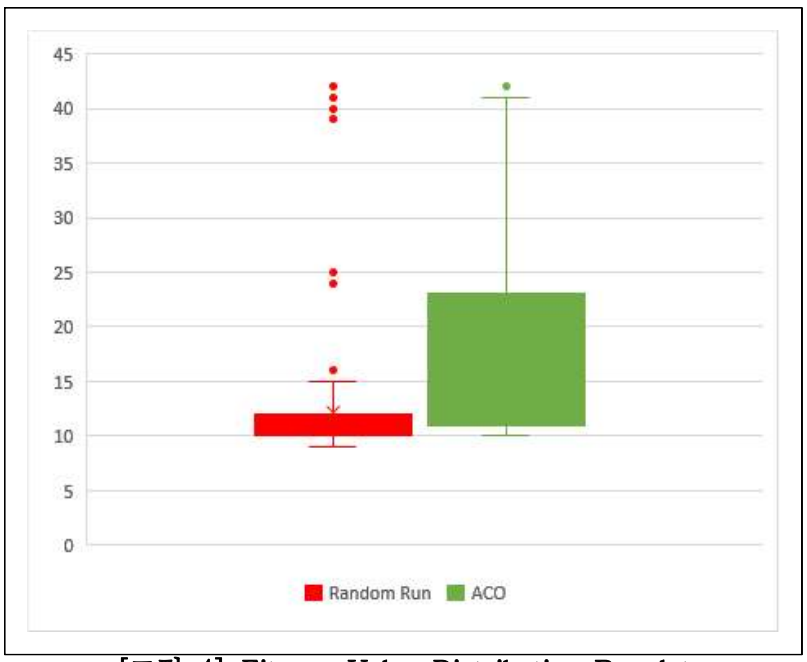


[그림 2] ACO Best Fitness Values



[그림 3] Random Run Best Fitness Values

ACO를 적용했을 때의 fitness 분포가 random run에 비해 평균적으로 높은 것을 볼 수 있다. 이는 [그림 4]를 통해 자세히 확인할 수 있다.



[그림 4] Fitness Value Distribution Boxplot

ACO가 얻어낸 test sequence들이 random generation된 sequence들 보다 더 나은 fitness를 가짐을 확인할 수 있었다.

6. Conclusion

GUITAR를 사용하여 Java GUI application에서 추출한 Event Flow Graph를 기반으로 하여, Ant Colony Optimization으로 높은 event coverage를 가지는 sequence를 생성할 수 있었다.

임의의 GUI application을 method level에서 분석할 수 있는 툴을 직접 만들기에는 시간과 여유가 부족했다. 물론 GUI testing이 가능한 툴과 프로젝트가 다수 공개되어 있지만, 모두 event level에서의 테스트를 지원하여 MCT와 같이 의미있는 데이터를 추출하기 어렵거나, 오랫동안 코드 revision 없이 방치되어 있어 정상적인 실행이 불가능했다. 따라서, 애초에 의도했던 method level 수준에서의 데이터를 이용하는 방식에서, application에서 EFG를 추출한 다음 이를 바탕으로 optimization을 진행하는 방식으로 선회했다. EFG를 추출하기 위해서 GUITAR [5]를 사용하였다. 하지만 배포된 GUITAR 1.2버전은 java application, 그중에서도 .class 파일에만 사용할 수 있었기 때문에 테스트할 수 있는 application에 한계가 있었다. 조금이라도 복잡한 대부분의 소프트웨어나 프로젝트는 .class 파일을 제공하지 않기 때문에 테스트에 사용할 수 없었다.

테스트용 application으로는 GUITAR에서 제공하는 sample application을 사용하였다. 이 application이 단순한 구조를 가지고 있어, optimization 과정에서 sequence들의 fitness value가 maximum을 달성하는 경우가 자주 발생하였고 효과를 검증하기 어려운 면이 있었다. 하지만 random run과 비교했을 때 maximum fitness를 자주 찾거나, 평균 fitness가 높은 등 확실한 향상을 보이고 있기 때문에, 더 복잡한 application을 테스트 대상으로 삼을 때 더 큰 효과가 나타날 것으로 본다.

Reference

[1] A. M. Memon, M. L. Soffa, and M. E. Pollack, “Coverage criteria for GUI testing,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 26, no. 5, p. 256, Sep. 2001.

[2] S. McMaster and A. Memon, “Call Stack Coverage for GUI Test-Suite Reduction,” in *2006 17th International Symposium on Software Reliability Engineering*, 2006, no. 1, pp. 33-44.

[3] S. Bauersfeld, S. Wappler, and J. Wegener, “A Metaheuristic Approach to Test Sequence Generation for Applications with a GUI,” 2011, pp. 173-187.

[4] S. Bauersfeld, S. Wappler, and J. Wegener, “An approach to automatic input sequence generation for GUI testing using ant colony optimization,” in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation - GECCO ’11*, 2011, p. 251.

[5] B. Nguyen, B. Robbins, I. Banerjee, and A. Memon, “GUITAR: An Innovative Tool for Automated Testing of GUI-Driven Software”, in *Automated Software Engineering*, vol. 21, no. 1, p. 65-105, Mar. 2014.