GA를 이용한 NVP 신뢰도 분석에 관한 연구

신 경 애[†] · 한 판 암[††]

요 약

컴퓨터 시스템의 성능을 분석하고 평가하는 방법에는 결함허용(fault tolerance)과 결함회피(fault avoidance) 기법이 있다. 소프트웨어 신뢰성을 향상시키기 위하여 소프트웨어 결함허용 기법중에서 가장 객관적이고 정량적으로 평가받는 것이 NVP(N-Version Programming)기법이다. 이 기법에서 신뢰도를 추정하는 모델로 이항분포를 사용하는데 이 추정 모델은 각 컴포넌트의 신뢰도의 값들이 동일하다는 한계점이 있었다. 본 논문에서는 기존 모델의 문제점을 해결하기 위하여 GA (Genetic Algorithms)를 적용하였다. GA를 적용하여 최적화 시뮬레이터를 구현하고 시뮬레이션을 수행해서 비교 분석 및 평가하였다. 그 결과 전체 시스템의 신뢰도를 일정 수준 이상 유지하면서 각 컴포넌트 신뢰도를 최적화 할 수 있었고, 또한 시스템 신뢰도에 가장 적합한 최적의 수를 추정할 수 있었다.

A Study on Analysis of NVP Reliability Using Genetic Algorithms

Kyeong-Ae Shin[†] · Pan-Am Han[††]

ABSTRACT

There are the fault tolerance technology and the fault avoidance technology to analyze and evaluate the performance of computer system. To improve the relibility of software, The N-Version Programming(NVP) technology is known to be the most objective and quantitive. However, when discrete probability distribution is used as estimation model, the values of it's component reliability should be same. In this paper, to resolve this problem, we adapted the genetic algorithms to NVP technology and implement the optimized simulate. And then the results were analyzed and estimated. Through this study, we could optimaze the reliability of each component and estimate the optimum count in the system reliability.

1. 서 론

소프트웨어 신뢰성을 향상시키는 방법에는 결함허용과 결함회피 기법이 있다. 결함회피는 소프트웨어 생명주기 동안에 결함이 없도록 소프트웨어를 설계하는 것을 뜻하며 결함허용은 소프트웨어 생명주기 동안에 결함이 발생해도 올바른 결과를 산출할 수 있도록 설계하는 것을 의미한다. 그런데 매우 복잡하면서 대규모이거나 고 신뢰도를 요구하는 소프트웨어를 완전하게 개발하는 것은 거의 불가능 하기 때문에 대부분 어느 정도 결함을 허용하는 소프트웨어 결함허용 기법을 사용한다. 이러한 소프트웨어 결함허용 기법을 이용하여 신뢰도를 향상시킬 수 있다.

소프트웨어 결함허용에는 복구블럭, NVP 기법들이 있는데 그 중에서 가장 객관적이고 정량적이라고 평가받는 방법이 NVP 기법이다. NVP 기법은 하나의 요구사양을 가지고 N개의 설계 팀이 서로 독립된 상태에

† 정 회 원 : 동주대학 전자계산과 교수
†† 종신회원 : 경남대학교 컴퓨터공학과 교수
논문접수 : 1998년 7월 24일, 심사완료 : 1998년 12월 8일

서 여러개의 프로그램을 개발하는 것을 말한다. 즉, 같은 내용의 프로그램이 N개이다.

그런데 일반적으로 NVP 기법에서 신뢰도를 추정할 때 추정 모델로 이항분포를 사용하여 전체 시스템 신뢰도를 추정하였다. 이 추정 모델에서의 문제점은 각 버전들의 컴포넌트 신뢰도가 동일하다는 것이다[2][3]. 이러한 문제점을 해결하기 위해서 본 연구에서는 GA를 적용하여 신뢰도를 추정 한다.

2. 결함허용 기법

2.1 결함허용 기법의 개념

결함허용 기법은 하드웨어, 소프트웨어 또는 정보의 결함이 발생하더라도 주어진 기능을 수행하여 올바른 결과를 산출할 수 있도록 하는 방법을 의미한다. 결함허용 시스템에는 결함탐지(fault detection), 결함위치(fault location), 결함분리 또는 봉쇄(fault isolation or containment), 결함 복구 및 시스템 재구성(fault recovery and reconfiguration) 등과 같은 구성 요소를 갖추어 설계해야 한다[1].

2.2 결함허용 기법의 종류

컴퓨터 시스템에서 결함허용 기법의 종류는 다음과 같다[12].

첫째, 하드웨어 결함허용 기법에는 삼중복(triple modular redundancy ; TMR), 와치독 타이머(Watchdog timer), 페어 및 스패어(Pair and spare), 중복비교(Duplication with comparison), 대기여분(Stand-by sparing) 등이 있다.

둘째, 정보 결함허용 기법에는 패리티 코드(Parity code), m 오브 n 코드, 검사합(Checksum), 베그 코드(Berger code), 해밍오류정정 코드(Hamming error correcting code) 등이 있다.

셋째, 소프트웨어 결함허용 기법에는 검사시점(Check pointing), 복구블럭(Recovery block), 분산복구블럭(Distributed recovery block), 자기 검사 프로그래밍(N Self-checking Programming), NVP 등이 있다.

2.3 소프트웨어 결함허용 기법의 성능 평가

Hudak은 소프트웨어 결함허용 기법의 성능 평가를 <표 1>과 같이 비교 실험하였다[8]. 이 결과에 의하면 NVP 기법이 다른 소프트웨어 결함 허용 기법에 비해 우수한 것으로 평가되었다.

〈표 1〉 소프트웨어 결함허용기법의 성능 비교
〈Table 1〉 Comparision of performance of software fault tolerance

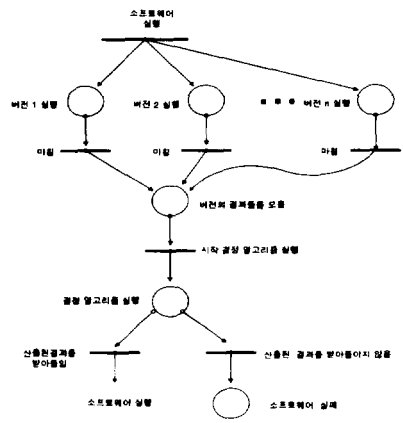
FT	ED	ER	Ab	Cr	Av	MTTF	MTTF L
NVP	5	1		1	1	1	1
Recovery Block	2	4	3	5	4	4	5
Concurrent Error Detection	4		1	2	4	4	4
Algorithmoc FT	1	2	3	2	3	4	4
Baseline				5	5	3	5

* <표 1>에서 1, 2, 3, 4, 5의 값은 1이 가장 좋고 5가 가장 나쁜 값이다.
* ED : error detection, ER : error recovery,
Ab : aborting, Cr : correctness
Av : availability, MTTF : mean time of failure,
MTTF L : MTTF w/o luck

3. NVP 기법과 GA

3.1 NVP 기법

소프트웨어 결함허용 방법중의 하나가 NVP 기법이다. 이는 소프트웨어 모듈을 N번 설계하고 코드화하여 이 모듈들에 의해 생성된 N개의 결과를 비교(Voting)하는 것이다[5]. 다시 말하면 NVP 기법은 하나의 요구사양을 가지고 N개의 설계 팀이 서로 독립된 상태에서 프로그램을 개발하는 것을 뜻한다. 즉, 여러개 설계를 하고 그 설계 결과를 가지고 하나의 결정 알고리즘이 정확한 값을 판정하도록 관리하는 시스템이다. NVP 기법을 N-버전 비교 또는 병렬 중복(Parallel redundancy[4], 또한 멀티버전 프로그래밍(Multiversion programming)[5]이라고도 한다. NVP 기법의 수행과정은 (그림 1)과 같다[11].



(그림 1) NVP기법의 처리과정
(Fig. 1) Procedure of NVP technology

(1) NVP 기법 성능 평가

NVP 기법의 성능을 평가하는데 가장 중요한 사항은 비용, 신뢰도 등이다[1][12].

① 비용 평가

NVP 기법에서 비용 평가를 하기 위해서는 다음과 같이 가정한다.

- 시스템은 N개의 단위 프로그램으로 구성되어 있다.
- 단위 프로그램이나 시스템은 두 가지 상태 즉, 성공과 실패만 존재한다.
- 단위 프로그램은 상호 독립되어 있다.

이때 전체 시스템 비용 $T(N)$은 식 (1)과 같다.

$$T(N) = c \cdot N + d \cdot (1 - R(N)) \quad (1)$$

여기서 c는 단위 프로그램 1개당 개발비용이고, d는 릴리즈 이후에 발생하는 시스템 고장 1개당 발생하는 비용이다. 그리고 $R(N)$는 시스템 신뢰도이다. 즉, d의 비용이 c에 비하여 큰 경우는 높은 신뢰도를 요구하는 시스템이며, 시스템 전체 비용은 달성하고자 하는 신뢰도나 결함율에 따라 그 차가 크게 나타날 수 있다. 그리고 c, N, d의 크기를 고정시키고 $R(N)$을 증가시키면 전체 비용 $T(N)$은 감소한다. 그러므로 개발 단계에서 신뢰도만 향상시킬 수 있다면 전체 비용을 줄이는 효과를 가져온다.

② 신뢰도 추정

NVP 기법에서 신뢰도를 추정 하기 위해서 다음과 같이 가정한다.

- 결함은 독립적으로 발생한다.
- 각 단위 프로그램의 신뢰도는 같다.
- $[\frac{N}{2}+1]$개 이상의 단위 프로그램이 정상이면 시스템은 결함이 없다.

이때 신뢰도 추정 $R(N)$은 식(2)와 같다.

$$R(N) = \sum_{j=k}^{N} {}_N C_j \cdot P^j \cdot (1-P)^{N-j}, \quad k = [\frac{N}{2}+1] \quad (2)$$

여기서 ${}_N C_j \cdot P^j \cdot (1-P)^{N-j}$ 는 N개중에서 j개만 성공할 확률이다. 즉, N개 중에서 j개의 단위 프로그램이 정상일 때 시스템이 작동할 확률을 말한다.

(2) 신뢰도 추정을 위한 수학적 모델링

지금까지 대부분의 NVP 기법을 이용한 연구에서는 각 단위 프로그램의 신뢰도가 같다고 가정 했기 때문에 신뢰도의 추정 모델로 이항 분포를 사용할 수 있었다. 그러나 현실적으로 컴포넌트의 신뢰도가 다른 경우가 거의 대부분이다. 따라서 본 논문에서는 기존의 NVP 기법에서 사용하던 가정을 다소 완화 시켰다. 본 논문에서 사용된 가정은 다음과 같다.

- 결함은 독립적으로 발생한다.
- 각 단위 프로그램의 신뢰도는 다르다.
- 단위 프로그램이나 시스템은 두가지 상태 즉, 성공과 실패만 존재한다.
- $[\frac{N}{2}+1]$개 이상의 단위 프로그램이 정상이면 시스템은 결함이 없다.

여기서 각 단위 프로그램의 상태가 성공이거나 실패만 존재하기 때문에 각 프로그램의 신뢰도는 각 프로그램이 성공할 확률이 P_i인 베루누이 분포를 따른다. 즉, 식 (3)과 같다.

$$R_i(N) = P_i^{x_i} \cdot (1-P_i)^{1-x_i}, \quad x_i = 0, 1 \quad (3)$$

여기서

$$x_i = \begin{cases} 1 : \text{프로그램이 작동하면} \\ 0 : \text{프로그램이 작동하지 않으면} \end{cases}$$

이다.

가정에서 $[\frac{N}{2}+1]$개 이상의 단위 프로그램이 작동하면 시스템은 결함이 없다고 가정하였기 때문에 시스템의 신뢰도 $R(N)$는 식(4)와 같이 사용할 수 있다.

$$R(N) = \sum_{j=k}^{N} {}_N C_j \prod_{i=1}^{j} P_i^{x_i} \cdot (1-P_i)^{1-x_i} \quad (4)$$

여기서 k는 $[\frac{N}{2}+1]$, N는 버전의 수이다.

3.2 GA의 기본개념

GA는 생물의 진화 메카니즘을 모방한 탐색 알고리즘이다. 또한 생물 진화의 원리로부터 착안된 것으로서 확률적 탐색이나 학습 및 최적화를 위한 한가지 기법이다[12]. 이 GA는 Holland에 의해 1975년에 처음 소개되었으며 포겔(Fogel)은 진화방식의 모형화를 시도하여 간단한 유한 상태 시스템의 최적화를 수행하였다[6][11]. 그 이후 지속적인 연구가 진행되고 있지만 본

격적으로 수행된 것은 최근의 일이며 아직 초보단계에 있는 학문이다[12].

(1) 3단계 유전자 오퍼레이션

GA의 기본은 Generate-and-test형(type)을 가지는 알고리즘이다. 즉, 문제를 문자열로 변환(즉, 유전자형에 대응)하고 문자열의 집단을 생성한다. 그리고 문자열을 평가하여 평가값이 높은 집단을 선택하여 남도록 한다. 한편 선택된 집단에 대하여 오퍼레이션을 적용하여 새로운 비트열을 생성한다. 또한 비트열을 복제하는 기능을 갖는 자기재생(copy)과 두 개의 비트열을 가지고 서로 부분적인 교환을 수행하여 새로운 비트열을 만들어 내는 교차, 비트열을 복제할 때 확률적으로 오류를 만들어내는 돌연변이 등이 있다. 즉, GA는 선택(selection), 교차(crossover), 돌연변이(mutation)와 같은 3단계 유전자 오퍼레이션(genetic operations)을 사용한다[4].

(2) 기본적인 GA

GA의 처리 순서는 (그림 2)와 같다. (그림 2)에서 보면 제일 먼저 초기집단(initial population)을 생성한다. 초기집단은 일반적으로 결정된 개체수의 염색체를 임의로 생성한다. 주로 초기집단의 크기(size)는 30개에서 100개 사이다. 이와 같이 초기집단이 생성되고 나면 각각의 개체에 대한 적응도 평가(fitness evaluation)를 한다. 적응도 평가의 기본적인 개념은 보다 좋은 개체가 높은 적응도의 평가를 받도록 하는 것이다. 그리고 각각의 개체에 대한 적응도가 결정되면 그것을 기초로 하여 교차시키는 조작을 한다. 이때 기본적으로 적응도가 높은 개체가 보다 많은 자손을 남기도록 한다. 이와 같이 함으로써 보다 좋은 개체를 형성하는 유전자가 확산하게 된다. 한편 선택교차를 수행할 개체의 쌍이 결정되면 염색체의 교차를 수행한다. 교차방법도 여러 가지가 있지만 기본적으로 쌍방의 염색체로부터 일부분씩 취하여 자손의 염색체를 만든다. 마지막으로 돌연변이를 수행한다. 돌연변이는 어떤 확률로 염색체의 일부 값을 변경시키는 조작이다. (그림 2)의 조작이 완료되면 새로운 세대의 개체 집단을 또 생성하게 된다.

이러한 새로운 집단(new population)에 대하여 또다시 적응도 평가, 선택, 교차, 돌연변이를 수행함으로써 다음의 새로운 세대를 만들어 간다. 즉, 이러한 사이클을 반복함으로써 환경에 대응하는 평가값이 높은 비트열을 만들어 내어 비트열 집단 전체의 평가값을 향상시켜가는 것이 GA이다[6].

```
step1  : Procedure Initial population
step2  : Evaluate fitness function of Initial population
stpe3  : Loop fitness function < 90%
step4  : selection execute
step5  : crossover execute
step6  : mutation execute
step7  : Procedure New population
step8  : Evaluate fitness function of New population
step9  : step3
step10 : end
```

(그림 2) GA 처리과정
(Fig. 2) Procedures of genetic algorithms

(3) 스키마타 개념 및 정리

스키마타(schemata)는 어떤 비트열 위치에서 유사성을 가진 비트열의 부분집합을 나타내는 공통적인 특성 또는 틀(templates)이다. 즉, 염색체가 1차원의 비트열로 표현되어 있을 경우에 그 중에서 의미있는 패턴이 발생한다. 이때 패턴을 스키마타라고 부른다. 그리고 스키마타를 스키마(schema) 또는 유사성 틀(similarity template)이라고도 부른다[18].

예를 들면, 01110, 01111, 01110, 11111일 때 스키마타는 *111*가 된다. *111*에서 첫 번째와 다섯 번째 비트는 0 또는 1이 되어도 상관이 없고 두 번째, 세 번째, 네 번째 비트는 반드시 1이 되어야 한다. 이런 비트열들의 공통적인 특성을 스키마타라 한다. 스키마타는 패턴이 어느 정도로 다음 세대에 살아 남을지를 보여주는 정리라고 볼 수 있다[18].

4. 최적화 시뮬레이터

4.1 시뮬레이터 환경

본 논문에서 최적화 시뮬레이터에 사용된 환경은 다음과 같다.

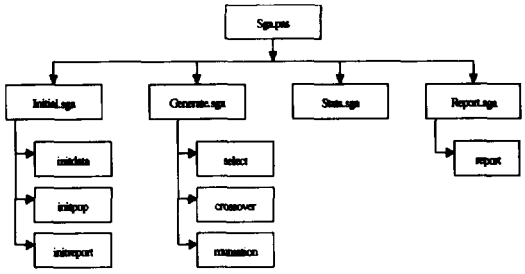
① IBM-PC환경에서 구현했다.

② 시뮬레이터에 사용된 언어는 windows용 turbo pascal 1.5 version 이다.

4.2 시뮬레이터 프로그램 구성도

최적화 시뮬레이터 전체적인 프로그램 구성도는 (그

림 3)과 같다.



(그림 3) 프로그램 구성도
(Fig. 3) Structures of program

4.3 시뮬레이터 구성 프로그램 기능

(그림 3)의 각 프로그램별 기능은 다음과 같다.

① Sga.pas 파일 : 최적화 시뮬레이터의 주 프로그램이다. 즉, 전체 시스템 신뢰도를 만족할 때까지 연속적으로 새로운 세대를 만들어가는 프로그램이다.

② Initial.pas 파일 : 초기세대를 생성하는 파일이다. 즉, 초기세대는 0과 1로 일련의 비트열(string)을 랜덤하게 생성하며 각 비트열은 컴포넌트의 상태를 나타낸다.
이때 각 비트마다 가중값을 가지게 되는데 이것은 해당위치에 존재하는 컴포넌트의 신뢰도에 해당한다. 이때 사용되는 함수는 initdata, initpop, initreport 등이다.

③ generate.pas 파일 : 유전자 알고리즘의 3단계 연산자인 선택, 교차, 돌연변이를 통하여 새로운 세대를 생성하는 파일이다. 이때 사용하는 함수는 select, crossover, mutation 등이다.

④ stats.pas 파일 : 각 세대의 최고, 최저, 평균값을 구하는 파일이다. 각 세대의 전체 시스템 신뢰도를 구하는 파일이다.

⑤ report.pas 파일 : 각 세대의 결과를 테이블의 형태로 표현하는 파일이다. 이때 사용하는 함수는 report 이다.

5. GA 시뮬레이션

5.1 GA 시뮬레이션 적용 조건

본 논문에서는 GA를 적용하기 위하여 다음과 같이 가정하였다.

① 7개의 버전을 가지고 적용하였다.

② 전체 시스템 신뢰도를 0.9로 하였다.

③ 각 컴포넌트들의 결함은 독립적으로 발생한다.

④ 각 컴포넌트들의 신뢰도는 다르다.

⑤ 각 컴포넌트들은 두 가지 상태를 가진다.
예를 들면 0은 결과 불일치, 1은 결과 일치

⑥ 컴포넌트들은 각 상태 비트마다 특정 신뢰도 값을 가진다.
예를 들면 0.5, 0.5666666667, 0.6333333333, 0.7, 0.7666666667, 0.8333333333, 0.9

⑦ N개 버전 중 $[\frac{N}{2}+1]$개 이상은 일치하는 것으로 한다.

5.2 적용 방법 및 분석

(1) 초기세대

초기세대는 임의로 선정하였다. 초기세대는 <표 2>와 같다. 이 <표 2>에서 살펴 보면 7개 비트 스트링의 형태 중에서 1의 개수가 4개인 것을 선택하였다. 왜냐하면 가정 ⑦에서 N개 버전 중에서 $[\frac{N}{2}+1]$개 이상 일치하는 것으로 하기 때문에 7개에서 4개가 일치한다는 것은 전체 시스템 신뢰도를 일정 수준 이상 유지하는 컴포넌트 개수가 되기 때문이다. 그리고 전체 시스템 신뢰도를 구하기 위하여 일반화 수식을 적응도 함수(fitness function)로 사용하였다. 적응도 함수의 전체 시스템 신뢰도는 식 (5)와 같다.

$$f(x)=R(N)=\prod_{i=1}^{k} P_i^{x_i}\cdot(1-P_i)^{1-x_i} \qquad (5)$$

여기서 k는 $[\frac{N}{2}+1]$, N은 버전의 수, x_i는 각 컴포넌트 상태값이다. 즉, 0 또는 1이다. 또한 Pi는 각 컴포넌트의 신뢰도의 값이다. 그리고 $P_i^{x_i}$는 성공할 확률, $(1-P_i)^{1-x_i}$는 실패할 확률이다. 그리고 Pselect는 선택될 확률을 나타내는데 식 (6)과 같다.

$$\text{Pselect} = \frac{fi}{\sum f} \qquad (6)$$

여기서 $\sum f$는 $f(x)$의 합을 나타내고, fi는 각각의 적응도 함수값이다. 그리고 Actual count는 다음세대(next population)에 존재할 스트링의 반복 개수를 나타내는 것으로 Actual count값은 Expected count값을

반올림 한 값으로 사용하였다. Expected count는 식 (7)과 같다.

$$\text{Expected count} = \frac{fi}{f} \tag{7}$$

여기서 f는 $f(x)$의 평균을 나타내고, fi는 각각의 적응도 함수값이다. 초기세대의 전체 시스템 신뢰도의 값은 0.411이다. 즉, 초기세대(1세대)는 전체 시스템 신뢰도가 약 41%이다.

〈표 2〉 GA 초기 세대 적용 과정
〈Table 2〉 Adopting process of initial population of genetic algorithms

string No.	initial population							f(x)	Pselect	Expected count	Actual count
	0.5	0.57	0.63	0.7	0.77	0.83	0.9				
1	1	1	1	0	1	0	1	0.0062	0.0099	0.296	0
2	1	0	1	1	0	1	1	0.0168	0.0292	0.876	1
3	1	1	0	1	1	0	1	0.0084	0.0136	0.040	0
4	1	1	1	0	0	1	1	0.0094	0.0158	4.119	4
5	0	1	1	1	1	1	1	0.0722	0.0809	2.723	3
6	1	1	0	1	0	1	1	0.0127	0.0218	0.053	0
7	1	0	0	1	1	1	1	0.0320	0.0583	1.748	2
8	0	1	1	1	1	1	0	0.0080	0.0153	0.457	0
9	0	1	1	1	1	1	0	0.0080	0.0153	0.457	0
10	1	1	1	0	0	1	1	0.0094	0.0158	0.474	0
11	1	1	0	1	0	1	1	0.0127	0.0218	0.653	1
12	1	0	1	1	1	0	1	0.0110	0.0183	0.548	1
13	1	0	1	1	1	0	1	0.0110	0.0183	0.548	1
14	1	1	1	0	1	1	0	0.0034	0.0062	0.548	1
15	1	1	1	0	1	1	0	0.0034	0.0062	0.186	0
16	0	1	0	1	1	1	1	0.0418	0.0772	2.317	2
17	1	0	0	1	1	1	1	0.0320	0.0582	1.748	2
18	1	1	1	0	1	0	1	0.0062	0.0099	0.296	0
19	0	1	1	1	0	1	1	0.0220	0.0387	1.162	1
20	1	1	0	0	1	1	1	0.0179	0.0316	0.946	1
21	1	1	0	1	1	1	1	0.0418	0.0772	2.317	2
22	1	1	0	1	1	0	1	0.0084	0.0136	0.408	0
23	1	0	1	1	1	0	1	0.0110	0.0183	0.548	1
24	1	1	1	0	0	1	1	0.0094	0.0158	0.474	0
25	0	1	1	1	1	1	1	0.0722	0.1373	4.119	4
26	0	1	1	1	1	1	0	0.0080	0.0153	0.457	0
27	1	1	1	0	1	1	0	0.0034	0.0062	0.186	0
28	1	1	0	1	1	1	1	0.0418	0.0772	2.313	2
29	1	1	1	1	0	1	0	0.0024	0.0043	0.129	0
30	1	1	1	0	0	1	1	0.0094	0.0158	0.474	1
Sum								0.8922	1.0000	30	30
Average								0.0297	0.0333	1	1
Max								0.0722	0.0722	4.119	4

(2) 2세대와 9세대

2세대와 9세대는 <표 3> 및 <표 4>와 같다. <표 3>에서 Mating pool after reproduction은 초기세대의 Actual count를 가지고 생성하였고, Mate는 임의로 정해 주었다. 한편 Mate를 할 때 될 수 있으면 상위 비트가 0인 것은 1과 Mate를 시키도록 하였고, Cross-over site도 임의로 Mate를 시켜서 새로운 집단을 생성시켰다. 역시 초기세대와 마찬가지로 각각의 적응도 함수값, Pselect, Expected count, Actual count를 구했다. <표 5>에서 나타난 바와 같이 2세대의 전체 시스템 신뢰도의 값은 0.5083이다.

한편 3세대에서 9세대까지의 적용과정도 2세대와 동일한 방법으로 하였다.

〈표 3〉 GA 2세대 적용 과정
〈Table 3〉 Adopting process of second population of genetic algorithms

string No.	M.P. after Reproduction	Mate	Cross-over Site	New Population	f(x)	Pselect	Expected count	Actual count
1	1110101	23, 19	5	1011111	0.0552	0.0632	1.896	2
2	1011011	23, 19	5	0111001	0.0044	0.0041	0.125	0
3	1101101	17, 12	6	1001111	0.0320	0.0355	1.066	1
4	1110011	17, 12	6	1011101	0.0110	0.0111	0.334	0
5	0111111	5, 21	7	0111111	0.0722	0.0838	2.514	3
6	1101011	5, 21	7	1101111	0.0418	0.0471	1.414	1
7	1001111	24, 25	4	1110011	0.0094	0.0096	0.289	0
8	0111110	24, 25	4	0111011	0.0220	0.0236	0.709	1
9	0111110	17, 20	7	1001101	0.0064	0.0062	0.188	0
10	1110011	17, 20	7	1100111	0.0179	0.0192	0.577	1
11	1101011	30, 16	5	1110011	0.0094	0.0096	0.289	0
12	1011101	30, 16	5	1111111	0.0722	0.0838	2.514	3
13	1011101	26, 21	7	0111110	0.0080	0.0093	0.279	0
14	1110110	26, 21	7	1101111	0.0418	0.0471	1.414	1
15	1110110	25, 20	5	0111111	0.0722	0.0818	2.514	3
16	0101111	25, 20	5	1100111	0.0179	0.0192	0.577	1
17	1001111	13, 5	4	1011110	0.0061	0.0070	0.210	0
18	1110101	13, 5	4	0111101	0.0144	0.0147	0.443	0
19	0111011	19, 21	2	0101111	0.0418	0.0471	1.514	2
20	1100111	19, 21	2	1111011	0.0220	0.0236	0.709	1
21	1101111	7, 11	4	1001011	0.0097	0.0100	0.300	0
22	1101101	7, 11	4	1101111	0.0418	0.0471	1.414	1
23	1011101	28, 24	2	1110011	0.0094	0.0096	0.289	0
24	1110011	28, 24	2	1101111	0.0418	0.0471	1.414	1
25	0111111	2, 16	7	1011111	0.0552	0.0632	1.896	2
26	0111110	2, 16	7	0101111	0.0418	0.0471	1.414	1
27	1110110	28, 19	7	1101101	0.0084	0.0083	0.249	0
28	1101111	28, 19	7	0111011	0.0220	0.0236	0.709	1
29	1111010	16, 28	1	0101111	0.0418	0.0471	1.514	2
30	1110011	16, 28	1	1101111	0.0418	0.0471	1.514	2
Sum					1.1013	1.0000	30	30
Average					0.0369	0.0333	1	1
Max					0.0722	0.0838	2.514	3

〈표 4〉 GA 9세대 적용 과정
〈Table 4〉 Adopting process of nine population of genetic algorithms

string No.	M.P. after Reproduction	Mate	Cross-over Site	New Population	f(x)	Pselect	Expected count	Actual count
1	0111010	8, 7	7	0111111	0.0722	0.0370	1.111	1
2	1011111	8, 7	7	0111111	0.0722	0.0370	1.111	1
3	0111111	28, 6	7	1111111	0.0722	0.0370	1.111	1
4	1101111	28, 6	7	0111111	0.0722	0.0370	1.111	1
5	1001111	7, 26	7	0101101	0.0084	0.0429	0.658	1
6	0111111	7, 26	7	0111111	0.0722	0.0370	1.111	1
7	0111111	13, 27	5	0111111	0.0722	0.0370	1.111	1
8	1111111	13, 27	5	0111111	0.0722	0.0370	1.111	1
9	0111111	2, 14	7	1011111	0.0552	0.0279	0.838	1
10	0111111	2, 14	7	1111111	0.0722	0.0370	1.111	1
11	1111110	28, 9	1	1111111	0.0722	0.0370	1.111	1
12	0111111	28, 9	1	0011111	0.0552	0.0279	0.838	1
13	0111111	29, 21	7	1111111	0.0722	0.0370	1.111	1
14	1111111	29, 21	7	1111111	0.0722	0.0370	1.111	1
15	1110111	2, 13	7	1011111	0.0552	0.0279	1.111	1
16	1111111	2, 13	7	0111111	0.0722	0.0370	1.111	1
17	0111111	25, 30	6	0111111	0.0722	0.0370	1.111	1
18	0111011	25, 30	6	1011111	0.0552	0.0279	0.838	1
19	0111111	6, 10	7	0111111	0.0722	0.0370	1.111	1
20	0111111	6, 10	7	0111111	0.0722	0.0370	1.111	1
21	1111111	6, 19	7	0111111	0.0722	0.0370	1.111	1
22	1011110	6, 19	7	0111111	0.0722	0.0370	1.111	1
23	0111111	7, 26	7	0111111	0.0722	0.0370	1.111	1
24	0111111	7, 26	7	0101111	0.0418	0.020	0.625	1
25	0111111	21, 10	7	1111111	0.0722	0.0370	1.111	1
26	0111111	21, 10	7	0111111	0.0722	0.0370	1.111	1
27	0111111	2, 17	7	1011111	0.0552	0.0279	0.838	1
28	1111111	2, 17	7	0111111	0.0722	0.0370	1.111	1
29	1111111	4, 16	7	1101111	0.0418	0.020	0.625	1
30	1011111	4, 16	7	1111111	0.0722	0.0370	1.111	1
Sum					1.9571	1.0000	30	30
Average					0.0652	0.0333	1	1
Max					0.0722	0.0370	1.111	1

(4) 시뮬레이션 결과

7개의 버전과 전체 시스템 신뢰도를 0.9했을 경우 각 세대별 최대값(max), 평균값(avg), 시스템 신뢰도의 값(f(x))은 <표 5>와 같다. <표 5>를 살펴 보면 세대를 거듭할수록 평균값과 전체 시스템 신뢰도의 값이 점점 더 커진다는 것을 알 수 있다. 이것은 전체 시스템 신뢰도의 값을 90%로 가정했을 때 가장 최적의 컴포넌트 신뢰도의 값이라는 것을 알 수 있다.

〈표 5〉 각 세대 시스템 신뢰도 값
〈Table 5〉 Values of system reliability of each generation

generation	max	avg	f(x)
1	0.0722	0.0297	0.4113
2	0.0722	0.0367	0.5083
3	0.0722	0.0429	0.5941
4	0.0722	0.0484	0.6703
5	0.0722	0.0548	0.7590
6	0.0722	0.0513	0.7105
7	0.0722	0.0574	0.7950
8	0.0722	0.0590	0.8172
9	0.0722	0.0652	0.903

(5) 비교 분석 및 평가

본 논문에서는 7개의 버전을 가지고 초기세대의 비트열을 만들었다. 그리고 가우스 함수에 의하여 초기세대의 비트열은 4개이상이 1로 구성되었다. 초기세대의 비트열은 임의로 구성했다. <표 5>에서 살펴본 결과 9세대까지만 해도 우리가 요구하는 전체 시스템 신뢰도를 구할 수 있었다. <표 2>, <표 3>, <표 4>에서 알 수 있는 것은 본 논문에서 설정한 환경에 적합한 비트열이 다음 세대에 생존할 확률이 높고 결국 수치의 합과 평균값이 점점 더 커지는 형태로 나타나게 된다. 이는 전체 시스템의 신뢰도가 점점 최대화 됨에 따라 더 적합함을 의미한다. 결국 전체 시스템 신뢰도 값은 계속 증가할 것이며 일정한 수준이 되면 수렴하는 경향을 띄게된다. <표 4>를 살펴 보게 되면 9세대에서 전체 시스템 신뢰도가 90%에 도달하게되는 것으로 나타나며 이때 9세대의 스케마타를 구해보면 ***1111로 표현할 수 있다. 이것은 7개중 4개의 컴포넌트가 일치하여야 하며 이때 각 컴포넌트의 신뢰도가 0.7, 0.7666666667, 0.8333333333, 0.9인 경우가 최적의 컴포넌트 신뢰도가 된다.

6. 결론 및 향후 연구 과제

본 논문에서는 기존의 NVP 신뢰도 추정모형의 한계점을 해결하기 위하여 NVP 시스템 신뢰도 추정모형에 GA를 적용하는 방안을 제안하였다.

이 GA를 이용한 추정 방안을 이용하여 각 컴포넌트 신뢰도들을 0.5~0.9까지 달리 지정하였다. 그 결과 컴포넌트 상태 비트 중에서 하위 비트에 1이 나타날수록 다음 세대에 살아남을 가능성이 높은 것으로 나타

났다. 이 특징은 높은 가중값을 가지는 컴포넌트가 많을수록 전체 시스템의 신뢰도가 향상되는 것을 의미한다. 그러므로 전체 시스템의 신뢰도를 일정 수준 이상 유지하면서 각 컴포넌트들에 대한 최적의 신뢰도를 추정함으로써 최적의 전체 시스템 개발 비용을 추정 할 수 있다.

본 논문에서 얻은 결론은 다음과 같이 두가지로 요약할 수 있다.

첫째, 전체 시스템의 신뢰도를 일정 수준 이상 유지하면서 각 컴포넌트 신뢰도를 최적화 할 수 있다.

둘째, 시스템 신뢰도에 가장 적합한 최적의 수를 추정할 수 있다.

한편 본 논문에서 적용시킨 GA는 기초적인 단계이므로 보다 높은 신뢰성을 보장하기 위해 다양한 기법의 적용이 요구되며 아울러 이를 기반으로 한 시뮬레이터의 개발이 중요한 향후 연구과제이다.

참 고 문 헌

[1] 양승민, "결함허용 소프트웨어 신뢰도 모델링 기법", 정보과학회지, 제11권 3호, pp.48-57, 1993.

[2] 임유미, 박만곤, "N-버젼 프로그래밍 기법을 사용한 소프트웨어 신뢰도 모델의 구현", 96 추계 학술발표논문집, 제3권 2호, 한국정보처리학회, p.497, 1996.

[3] 황현숙, 장화식, 박만곤, "파스칼 보팅 절차에서 N 버젼 프로그래밍 소프트웨어 신뢰도 추정모형, 94년 춘계 학술발표 논문집, 제1권 1호, 한국정보처리학회, p.210, 1994.

[4] Ann Marie Neufelder, 'Ensuring Software Reliability,' Marcel Dekker, Inc., NewYork, 1993.

[5] Fevzi Belli and Piotr Jedrzejovicz, "An Approach to the Reliability Optimization of Software with Redundancy," IEEE Transactios on Software Engineering, Vol.17, No.3, 1991.

[6] Goldberg. D., 'Gentic Algorithms in Search Optimazation & Machine Learning,' Addision Wesley, 1989.

[7] Joanne Bechta Dugan, Michael R. Lyu, "System Reliability Analysis of an N-Version Programming Application," IEEE Transactions Reliability, Vol.43, No.4, 1994.

[8] John Hudak et al, "Evaluation & Comparision of Fault Tolerant Software Techniques," IEEE Transactions On Software Engineering, Vol.42, No.2, 1993.

[9] Karama Kanoun, Mohamed Kaaniche, Christian Beounes, Jean-Claude Laprie, Jean Arlat, "Reliability Growth of Fault-Tolerance Software," IEEE Reliability, Vol.42, No.2, 1993.

[10] Kishor. Trivedi, 'Reliability & Statistics With Reliability, Queuing, And Computer Science Applications', 1995.

[11] Michael R. Lyu, 'Software Fault Tolerance,' John Wiley & Sons, 1995.

[12] L.A. Belady, MM. Lehman, "A model of large program development," IBM System Journal, Vol.15, No.3, pp.225-252, 1976.

[13] Robert E. Dorsey and Walter J. Mayer, "Genetic Algorithms for Estimation Problems with Multiple Optima, Nondifferentiability, and Other Irregular Features," Journal of Business & Economic Statistics, Vol.13, No.1, pp.53-65, 1995.

[14] S. Hurley, L. Moutinho, N.M. Stephens, "Solving marketing optimization problems using genetic algorithms," European Journal of Marketing, Vol.29, No.4, pp. 39-56, 1995.

[15] Syswerda, G. and Palmucci, J. : "The Application of Genetic Algorithms to Resoure Scheduling," Proc. of ICGA-91, 1991.

[16] Whitely, D. and Hnson, T. : "Optimizing Neural Networks Using Faster, More Accurate Genetic Search," Proc. of ICGA-89, 1989.

[17] Whitely, D. , Starkweather, T. and Fuquay, D. : "Scheduling Problems and Traveling Salesman : The Genetic Edge Recombination Operator," Proc. of ICGA-89, 1989.

[18] Zbigniew Michalewicz, 'Genetic Algorithms + Data Structures=Evolution Programs,' Springer-Verlag, 1996.

신 경 애

e-mail : kashin@seokpa.dongju-c.ac.kr

1986년 한국방송대학교 경영학과 졸업(경영학사)

1991년 한국방송대학교 전자계산학과 졸업(이학사)

1989년 계명대학교 교육대학원 전자계산학과 졸업(교육학 석사)

1997년 경남대학교 대학원 컴퓨터공학과(박사과정 수료)

1983년~1989년 동주여자상업고등학교 재직

1990년~현재 동주대학 전자계산과 조교수

관심분야 : 소프트웨어 공학(소프트웨어 비용예측, 결함허용, 소프트웨어 신뢰도 분석)

한 판 암

e-mail : pahan@zeus.kyungnam.ac.kr

1969년 동국대학교 졸업

1975년 동국대학교 경영대학원 졸업(경영학 석사)

1989년 명지대학원 대학원 졸업(공학석사)

1992년 인천대학교 대학원 졸업(경영학 박사)

1980년~현재 경남대학교 공과대학 컴퓨터공학과 교수

관심분야 : 소프트웨어 품질관리 및 신뢰성, 소프트웨어 개발환경, 정보공학