

A Greedy Genetic Algorithm for Release Planning in Software Product Lines

Jaewook Yoo[†]

Department of Business Administration, Dong-A University

소프트웨어 제품라인의 출시 계획 수립을 위한 탐욕 유전자 알고리즘

유재욱[†]

동아대학교 경영대학 경영학과 조교수

Release planning in a software product line (SPL) is to select and assign the features of the multiple software products in the SPL in sequence of releases along a specified planning horizon satisfying the numerous constraints regarding technical precedence, conflicting priorities for features, and available resources. A greedy genetic algorithm is designed to solve the problems of release planning in SPL which is formulated as a precedence-constrained multiple 0-1 knapsack problem. To be guaranteed to obtain feasible solutions after the crossover and mutation operation, a greedy-like heuristic is developed as a repair operator and reflected into the genetic algorithm. The performance of the proposed solution methodology in this research is tested using a fractional factorial experimental design as well as compared with the performance of a genetic algorithm developed for the software release planning. The comparison shows that the solution approach proposed in this research yields better result than the genetic algorithm.

Keywords : Release Planning, Software Product Line, Precedence-constrained Multiple 0-1 Knapsack Problem, Greedy Genetic Algorithm

1. 서 론

소프트웨어를 이해관계자나 시장에 방출할 계획을 수립하는 것은 해당 소프트웨어의 기능들을 개발하여 계획 중인 기간 내에 여러 개의 출시시점 중 가장 적절한 출시시점에 할당을 계획하는 일이다. 이 때 고려해야 할 제약들은 기능 간 개발 구현 우선순위, 소프트웨어 이해관계자들(stakeholders)이 각 기능들에 대하여 갖는 중요도, 제한된 개발 인적자원 등이다. 이러한 출시계획이 없다면 소프트웨어를

이루고 있는 주요기능들이 적절한 시점에 출시되지 못하여 시장에서 고객들로부터 외면 당할 수 있다. 이와 같이 시기가 적절치 못한 출시는 고객의 불만족, 시간과 예산의 초과, 시장점유의 상실 등을 야기하여 기업에 큰 손실을 줄 수 있다[5].

소프트웨어 제품 라인(Software Product Line)은 그 제품 라인 안에서 공용되는 플랫폼과 그 플랫폼을 기반으로 파생되는 다수의 파생제품들(Product Variants)로 이루어져 있다. 소프트웨어 개발에 있어서 소프트웨어 제품 라인 개념은 중요한 패러다임(paradigm)이며, 그 이유는 플랫폼의 재사용(reuse)을 통한 비용, 품질, 출시까지의 시간(time-to-market)을 향상시킬 수 있기 때문이다[1].

Received 1 July 2013; Finally Revised 19 August 2013;
Accepted 21 August 2013

[†] Corresponding Author : jyoo@dau.ac.kr

소프트웨어 제품라인의 출시 계획은 소프트웨어 제품 기능간의 특성 뿐만 아니라, 개발환경도 잘 이해하여 출시 계획 수립 시 고려되어야 한다.

소프트웨어를 점진적으로 개발하는 환경에서 한 개 소프트웨어 출시계획 문제를 수리모형화하고 이를 풀기 위한 해법을 제시한 연구는 상당부분 진행되어 왔으나[2, 3, 8], 소프트웨어 제품라인의 개발환경에서 소프트웨어 출시계획을 정형화된 방법으로 연구한 논문은 그리 많지 않다. 또한, 소프트웨어 출시 계획에 대한 해법이 실제 산업에서 발생하는 큰 규모의 문제에 적용 가능한지를 분석한 연구도 활발히 수행되지 못했다[8].

Ullah and Ruhe[10]가 소프트웨어 제품라인의 소프트웨어 출시계획 문제에 대하여 정형화된 모형을 제시하였지만, 그 해법은 한 개 제품의 출시 계획을 수립하기 위해 기존에 개발된 의사결정 시스템을 이용하는 수준이었다. Taborda[9]는 출시행렬(release matrix)을 이용하여 전체적인 관점에서 소프트웨어 제품라인의 출시계획을 다루었다. Yoo는 소프트웨어 제품라인의 출시 계획을 우선순위 제약하의 다수 0-1 배낭문제로 수리 모형화하고 그 해법으로 동적계획법(dynamic programming)을 개발하였고[11], 또한 유전자 알고리즘을 제시하여 이 알고리즘이 실제 산업의 규모가 큰 문제에 적용 가능한 지를 알기 위하여 성능을 테스트하고 분석하였다[12].

본 연구의 핵심 기여부분은 Yoo[12]가 제시한 유전자 알고리즘을 소프트웨어 제품라인의 출시 계획수립 모형인 우선순위 제약하의 다수 0-1 배낭문제에 보다 잘 작동할 수 있도록 개선하고, Yoo[12]가 제시한 알고리즘과 본 연구에서 제시한 알고리즘의 성능을 비교 분석한 것이 되었다.

본 논문의 제 2장에서는 소프트웨어 제품라인의 출시 계획 문제의 수리모형을 요약하여 설명하고, 제 3장에서는 제 2장에서 제시한 수리모형의 해법 절차로써 탐욕 유전자 알고리즘을 제시한다. 제 4장에서는 탐욕 유전자 알고리즘을 계산 실험을 수행하여 그 성능을 분석하고, 제 5장에서는 본 연구의 결론과 미래연구분야를 제시한다.

2. 수리모형

소프트웨어 제품라인을 이루고 있는 모든 기능들은 두 가지 종류로 분류된다. 첫 번째는 플랫폼을 이루고 있는 핵심자산기능(core asset feature)이고, 두 번째는 그 플랫폼을 기반으로 하는 각각의 파생제품들의 제품 특성기능(product specific feature)이다. 본 연구에서는 소프트웨어 제품라인의 모든 기능들은 이와 같이 두 가지 종류 중 하나로 이미 결정되어 있다고 가정한다.

소프트웨어 제품 라인 of 플랫폼 또는 제품 k 를 형성하고 있는 모든 기능들의 집합을 $F(k)$ 로 정의하자 : $UF(k) = \{1, 2, \dots, I\}$. 이 기능들은 출시를 고려 중인 계획 기간 내 총 T 번의 출시 순서 중 하나의 출시 시점에 할당되어 시장에 나갈 수 있다. 이것은 의사결정변수 x_{it} 에 의해 결정된다: 기능 i 가 t 번째 출시에 할당되면 $x_{it}=1$ 이고, 기능 i 가 t 번째 출시되지 못하면 $x_{it}=0$ 이다.

본 연구의 수리모형의 목적은 출시 시점에 따른 가능한 개발자원, 기능간 개발 우선순위, 기능선택 제약 식을 만족 하면서 출시를 계획 기간 내 다수의 출시 시점에 소프트웨어 제품라인 내 기능들을 할당하는 것인데, 이 때 목적식 $P(x)$ 를 최대화하는 출시 계획 x 를 찾는 것이다. 여기서 $P(x)$ 의 값은 각 출시 및 이해관계자들의 중요도, 이해관계자가 인식하는 각 기능의 긴급도와 가치정도 등과 같은 요소들에 의해 좌우되며 다음과 같이 정의된다.

$$P(x) = \sum_t \sum_{i \in F(k)} p_{it} x_{it}$$

여기서,

$$p_{it} = \xi(t) [\sum_q \lambda(q) \times value(q, i) \times urgency(q, i)] \quad (1)$$

식 (1)에서 $\xi(t)$ 는 출시 t 의 중요도이고, 이 중요도는 계획된 총 출시에 대해서 1로 정규화된(normalized to 1)된다. 소프트웨어 이해관계자들(stakeholders)의 중요도는 소프트웨어 출시 계획을 수립하는 데 매우 중요한 요인이다. 소프트웨어 이해관계자들의 집합을 $S = \{1, 2, \dots, q, \dots, Q\}$ 라고 가정하자. 이해관계자 q 는 상대 중요도 $\lambda(q) \in \{1, 2, \dots, 9\}$ 를 할당 받을 수 있다. $\lambda(q)=1$ 은 상대 중요도의 최저 값을 가리키고, $\lambda(q)=9$ 는 최고 값을 의미한다. 각각의 이해관계자들은 모든 기능들에게 두 가지 기준으로 점수를 줄 수 있다. 첫 번째 기준은 이해관계자 q 에게 기능 i 가 얼마나 가치가 있는 가를 보여주는 가치정도 $value(q, i)$ 이고 두 번째 기준은 얼마나 긴급한가를 보여주는 긴급정도 $urgency(q, i)$ 이다. 만점은 9점이다.

본 연구에서 다루어질 소프트웨어 제품라인의 출시 계획 수립 문제는 아래 문제 (P)에서 목적식 (2)와 제약식 (3)~(6)으로 우선순위 제약하의 다수 0-1 배낭문제로 수리 모형화된다.

문제 (P)

$$\max \sum_{t=1}^T \sum_{i \in F(k)} p_{it} x_{it} \quad (2)$$

$$s.t. \sum_{i \in F(k)} w_{it} x_{it} \leq C_{ht} \quad \text{for all } k, t \quad (3)$$

$$\sum_{t=1}^T (T+1-t)(x_{it} - x_{jt}) \geq 0 \quad \text{for all } i, j \quad (4)$$

$$\sum_{t=1}^T x_{it} \leq 1 \quad \text{for all } i \quad (5)$$

$$x_{it} \in \{0, 1\} \quad \text{for all } i, t \quad (6)$$

문제(P)에서 최대화 목적 식 (2)는 식 (1)로 정의된다. 자원제약 식 (3)은 플랫폼 또는 제품 k 를 이루고 있는 기능들을 구현하기 위해서 필요한 자원은 출시 t 에서 플랫폼 또는 제품 k 의 개발팀이 보유한 자원을 초과할 수 없다는 것을 보여준다. C_{kt} 는 t 번째 출시하게 될 플랫폼 또는 제품 k 개발팀의 가용한 개발 자원들이다. 이것은 실제 소프트웨어 개발업체들이 개발조직을 구성할 때 소프트웨어 제품라인에 맞게 조직을 구성하고 있는 것을 식 (3)에 반영한 것이다. w_{it} 는 기능 i 를 t 번째에 출시 하기 위하여 구현하는 데 소요되는 개발 자원의 양이다. 우선순위 제약 식 (4)는 전체 출시 계획 기간에 걸쳐 소프트웨어 제품라인의 기능간 개발 우선순위를 보장한다. 기능선택 제약 식 (5)는 각 기능들이 전체 출시 계획 기간 중 하나의 특정 출시에 할당되거나, 출시를 지연하도록 한다; 제약 식 (6)은 의사결정변수의 0-1 정수 조건을 의미한다.

3. 제안된 알고리즘

3.1 유전자 알고리즘

제 2장에서 설명된 수리모형을 풀기 위하여 유전자 알고리즘(Genetic Algorithm)을 적용하였다. Yoo[12]가 제시한 알고리즘과 상이한 부분은 단계 1의 초기 모집단(initial population)을 구하는 방법과 단계 7의 수리 오퍼레이터(repair operator) 부분이다.

초기 모집단 구하는 방법을 수정한 이유로 Yoo[12]가 제시한 방법은 문제(P)를 선형계획 모형으로 완화(Linear Programming-Relaxed)하여 구한 선형 최적해를 이용하여 초기 모집단을 구하는 것이었다. 이 방법은 Raidl[6]이 제시한 방법을 활용한 것인데, 이렇게 구한 초기 모집단은 그 품질이 우수하여 유전자 알고리즘이 빠르게 특정 해를 찾을 수 있도록 돕는다. 그러나, Kapsalis et al.[4]와 Reeves [7]에 의하면, 휴리스틱을 이용하여 얻은 품질이 좋은 염색체로 이루어진 초기 모집단은 유전자 알고리즘이 보다 좋은 해를 찾을 수 있는 데도 불구하고 특정 방향으로 편중되어 초기에 특정 해에 수렴하게 하는 단점이 있다는 것을 발견하였다. 그 이유로 초기 모집단을 이루고 있는 염색체들의 다양성이 부족한 것에 있다고 하였다. 본 연구에서는 이와 같은 유전자 알고리즘이 특정한 방향으로 편중되어 초기에 특정해에 도달하는 현상을 보완하여 보다 우수한 해를 얻고자 난수를 발생하여 무작위 방식으로 다양한 염

색체로 이루어진 초기 모집단을 구하고자 한다. 단계 1에서 이와 같은 초기 모집단을 구하는 방법을 설명한다.

수리 오퍼레이터의 역할은 염색체 교체와 염색체 돌연변이 오퍼레이터를 수행 후 염색체가 불가능해질 경우 이들을 가능해로 만드는 것이 그 역할이다. [12]에서는 0-1 정수 모형인 문제(P)를 선형계획 모형으로 완화하여 구한 해를 기반으로, 이들 해를 오름차순으로 정렬한 후 작은 값을 갖는 변수부터 점검하여 그 변수가 문제(P)의 제약식을 위반하는 경우 0으로 놓아 가능해를 찾도록 하였다. 이 방법은 단지 불가능해에서 가능해를 구하는 데 주력하였다. 본 연구에서는 보다 우수한 가능해를 얻고자 우선순위 제약하의 다수 0-1 배낭문제로 모델링된 문제(P)에 잘 작동하는 탐욕스러운 휴리스틱(greedy-like heuristic)을 개발하여 수리오퍼레이터에 반영한다. 수리오퍼레이터의 상세 내용을 제 3.2절에 설명한다.

다음은 위의 내용을 반영한 탐욕 유전자 알고리즘을 단계별로 설명한 것이다.

단계 1: 초기 모집단 구하기

0과 1사이의 균등분포(uniform distribution)를 따르는 난수를 발생하여 무작위 순위로 염색체(chromosome)내 유전자에 할당한다. 그 값이 0.5 이상이면 무작위로 선택된 의사결정변수(유전자)에 그 값을 1로 두고, 0.5 미만이면 그 값을 0으로 둔다. 염색체의 크기는 각 실험의 문제크기에 맞게 만들고, 한 개의 모집단내 염색체의 개수는 모든 실험에 동일하게 100개를 발생한다. 이 때 염색체는 모두가 가능해 이어야 한다.

단계 2: 적합도 평가(fitness evaluation)하기

단계 1에서 구한 초기 모집단의 모든 염색체의 적합도를 구한다. 각 염색체는 문제(P)의 가능해이므로 목적식에 가능해를 대입하여 목적식 값을 구하여 모집단의 적합도를 평가할 수 있다.

단계 3: 알고리즘의 종료 결정하기

현재 모집단의 염색체의 90% 이상이 같은 적합도 값을 갖거나 세대(generation)의 수가 실험마다 지정된 한계의 수를 넘어가면 알고리즘을 종료한다. 이 두 조건 중 어느 하나라도 만족하지 못하면 다음 단계를 수행한다.

단계 4: 염색체 선택(selection)하기

현재의 모집단에서 무작위로 두 개의 염색체(부모염색체)를 선택한다.

단계 5: 염색체 교체(crossover)하기

부모 염색체로 선택된 두 개의 염색체를 교체 확률이

0.9($P_c = 0.9$)인 균등 교체(uniform crossover)를 수행한다.

단계 6: 염색체 돌연변이(mutation)하기

돌연변이 확률을 $1/n$ (여기서, n 은 변수의 개수)로 하여 염색체 변이를 수행한다. 교체와 돌연변이 단계 수행 후 염색체가 불가능해(infeasible solution)인 경우 단계 7을 수행한다. 가능해인 경우는 단계 2로 간다.

단계 7: 불가능해 염색체를 가능해 염색체로 만들기

탐욕스러운 휴리스틱으로 구성된 수리 오퍼레이터를 실행하여 불가능해 염색체를 가능해 염색체로 만든다(상세 설명은 제 3.2절 참조). 단계 2로 간다.

3.2 수리 오퍼레이터(Repair Operator)

유전자 알고리즘에서 염색체 교체(crossover)와 염색체 돌연변이 (mutation)를 수행 후 만들어진 염색체는 불가능해(infeasible)일 가능성이 높다. 왜냐하면, 이 염색체가 본 논문의 문제(P)의 다수의 제약식을 모두 만족시킬 가능성이 높지 않기 때문이다. 염색체의 불 가능해를 가능해로 만들어주기 위하여 탐욕스러운 휴리스틱을 개발하여 염색체 교체 및 염색체 돌연변이 수행 후 불 가능해인 염색체에 적용한다.

탐욕스러운 휴리스틱을 보다 효율적으로 개발하기 위하여 유사 효용 비(pseudo-utility ratio)라는 개념을 활용한다. 본 연구에서 이 비는 문제(P)의 의사결정변수 x_{it} 의 개발 자원 제약식 (3)의 계수 w_{it} 대비 목적식 (2)의 계수 p_{it} 의 비 $u_{it} = \frac{p_{it}}{w_{it}}$ 로 정의한다. 이 비가 클수록 해당 의사결정변수가 가능해에 속하게 될 가능성이 높다.

수리 오퍼레이터는 u_{it} 를 기준으로 각 의사결정변수를 자손 해(child solution)에 포함시킬 것인지, 배제시킬 것인지를 고려하여 설계된다. 본 연구에서 수리 오퍼레이터는 두 가지 단계로 이루어진다. 첫 번째 단계는 불가능해에서 가능해를 찾는 것이고, 이 단계를 DROP 단계라고 한다. 두 번째 단계는 첫 번째 단계에서 나온 가능해를 보다 우수한 해로 향상시키는 것이며 ADD 단계라고 한다.

첫 번째는 불가능해인 염색체의 각 의사결정변수를 u_{it} 의 오름차순으로 검토한다. 검토 대상은 그 값이 1인 의사결정변수 x_{jt} 이며, 검토 항목은 x_{jt} 에 선행되어 구현되어 있어야 할 기능 x_{it} 의 값이 무엇인지, w_{jt} 를 포함하여 현재까지 누적된 자원 값 W_{kt} 가 C_{kt} 를 초과하였는지 또는 아닌지의 여부이다. 이들을 점검하여 해의 가능성(feasibility)이 깨지면 이 때 해당 의사결정변수의 값을 1에서 0으로 바꾼다.

두 번째 단계는 첫 번째 단계와는 역으로 실행하는 것

인데, 각 의사결정변수를 u_{it} 의 내림차순으로 검토한다. 검토 대상인 의사결정변수 x_{jt} 는 그 값이 0인 것이며, 검토 항목은 DROP 단계에서와 같다. 해의 가능성이 깨지지 않는 한 의사결정변수의 값을 0에서 1로 바꾼다.

두 단계는 각각의 출시 t 에 대해서 행하여지며, 수리 오퍼레이터를 보다 효과적으로 실행하기 위해서는 실행 전 의사결정변수를 u_{it} 의 내림차순으로 나열한다.

이와 같이 수리 오퍼레이터는 염색체 교체(crossover)와 염색체 돌연변이(mutation)를 수행 후 만들어진 염색체가 불가능해일 경우, 이를 가능해로 만드는 것을 보장한다. <Figure 1>은 위에 설명한 두 단계의 탐욕스러운 휴리스틱으로 이루어진 수리오퍼레이터를 보여준다.

Algorithm. Repair operator for precedence constrained 0-1 multiple knapsack problem:

Let: W_{kt} = the accumulated resources for constraint t in X_{kt} , an infeasible solution for platform/product k at release t , which is obtained through crossover and mutation operator.

```

for  $t = 1$  to  $m$ 
  for  $j = n$  to 1 do/DROP phase/
    if  $(x_{jt} = 1)$  and  $(x_{it} = 0, \forall (v_i, v_j) \in E)$ 
      then
        set  $x_{jt} \leftarrow 0$ ;
        set  $W_{kt} \leftarrow W_{kt} - w_{jt}$ ;
      elseif  $(x_{jt} = 1)$  and  $(W_{kt} > C_{kt} \text{ any } k)$  then
        set  $x_{jt} \leftarrow 0$ ;
        set  $W_{kt} \leftarrow W_{kt} - w_{jt}$ ;
      endif
    endfor
  for  $j = 1$  to  $n$  do/ADD phase/
    if  $(x_{jt} = 0)$  and  $(W_{kt} + w_{jt} \leq C_{kt} \forall k)$  and
       $(x_{jt} = 1, \forall (v_i, v_j) \in E)$  then
      set  $x_{jt} \leftarrow 1$ ;
      set  $W_{kt} \leftarrow W_{kt} + w_{jt}$ ;
    end if
  end for
end for

```

<Figure 1> Repair Operator for Precedence Constrained 0-1 Multiple Knapsack Problem

4. 실험 조건 및 결과

본 연구에서 제시된 유전자 알고리즘의 성능실험을 위하여 알고리즘을 Matlab(2012a 버전)으로 코딩하였다. 인텔코어 i3-2100 CPU(3.10 GHz)와 1.90 GB 램(RAM)을 탑재한 PC가 실험에 사용되었다. 제안된 알고리즘의 작동상태(behavior) 및 성능을 파악하기 위하여 다음과 같이 4개 요인에 각 요인이 2개의 값을 갖는 조합으로 알고리즘이 적용될 문제를 발생시켰다[12].

- (i) 제품라인 내 기능의 총수 $\in \{70, 100\}$
- (ii) 출시 횟수 $\in \{3, 5\}$

- (iii) 제품라인 내 플랫폼/제품의 총수 $\in \{5, 10\}$
- (iv) 기능개발 우선순위 제약식 규모 $\in \{3, 5\}$

기능 개발 우선순위 제약식 규모의 차이에 따라 본 연구에서 제시한 유전자 알고리즘의 성능을 보기 위하여 기능개발 우선순위 제약식의 규모를 구분하였다. 기능개발 우선순위 제약식 규모의 3과 5의 의미는 한 기능에 개발 우선순위로 연계된 기능의 수를 3개 또는 5개로 구별하여 기능개발 우선순위 제약식 규모의 크기에 차이를 주었다.

제약식 (3)의 계수 w_{it} 로는 3,000과 10,000사이의 숫자를 랜덤하게 발생시켜서 갖도록 하였고, 다음 출시의 제약식 계수 $w_{i,t+1}$ 에는 물가 상승을 등을 고려하여 이전 출시의 제약식 계수에 1.05배를 하여 갖도록 하였다. 목적식 계수 p_{it} 는 경험적으로 이 계수에 대응되는 제약식 계수 w_{it} 에 20배에서 40배 사이의 값을 랜덤하게 발생시켜서 갖도록 하였으며, 기능간의 개발 우선순위도 랜덤하게 발생시켰다[12].

위 조합의 총 수가 16개로 너무 많아 최소한의 실험 횟수로 원하고자 하는 분석결과를 얻고자 실험계획 기법인 부분요인설계(fractional factorial design)를 적용하였다. 본 연구

의 실험에는 4개의 요인(factor)을 고려하고, 각 요인은 2개의 레벨(level)이 있는 것으로 설계하였으므로, 8개의 실험 조건이 실험계획의 직교배열(orthogonal array)로 부터 얻어진다. <Table 1>는 직교배열 OA(8, 4, 2, 3)을 나타낸다. 이 표기에서 8은 실험의 런(run) 수를, 4는 요인 수, 2는 레벨의 수, 3은 강도(strength)를 나타낸다[12].

<Table 1>의 행은 실험 조건을 보여주며, 열은 요인을 의미하는 데 실험을 통하여 이들의 효과가 분석된다. <Table 1>

<Table 1> OA(8, 4, 2, 3)

0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

<Table 2> Experiment Results

Feature	Release	Platform/Product	Precedence-constraints scale	Generation	CPU time (second)		Fitness evaluation	
					GA	Greedy GA	GA	Greedy GA
70	3	5	3	10^2	0.77	1.88	662327	665486
				10^3	6.98	16.55	727586	768458
				10^4	62.27	132.56	812101	838316
70	3	10	5	10^2	0.95	1.97	295236	283042
				10^3	8.18	17.12	328485	320625
				10^4	79.81	150.91	345076	352070
70	5	5	5	10^2	1.82	3.66	840714	577707
				10^3	19.19	28.73	1003684	807739
				10^4	188.53	214.11	1180061	1280206
70	5	10	3	10^2	1.63	3.36	1221946	935337
				10^3	16.41	25.74	1325531	1147314
				10^4	147.55	188.91	1457829	1512342
100	3	5	5	10^2	1.95	3.48	990629	939208
				10^3	18.77	30.66	1147209	1166718
				10^4	169.83	196.39	1287525	1654068
100	3	10	3	10^2	1.55	2.86	1600176	1346240
				10^3	14.66	23.54	1725971	1662661
				10^4	126.39	162.04	1840051	1963687
100	5	5	3	10^2	3.25	5.13	1827707	1274815
				10^3	34.14	36.72	1907777	1716975
				10^4	305.11	251.94	2062997	2058399
100	5	10	5	10^2	5.91	6.99	947308	679081
				10^3	59.21	48.45	1121987	943097
				10^4	586.80	329.74	1244945	1609779

내에 기재된 값 0과 1은 요인이 갖는 레벨을 보여준다. 각 행의 조합당 10개의 문제를 발생하였고 총 80개의 문제에 대하여 실험이 수행되었다. 본 실험에서 각 조합에 대하여 10개씩 총 80개 문제의 10^2 , 10^3 , 10^4 세대(generation)를 통하여 해가 얻어졌다. Yoo[12]가 제시한 알고리즘과 본 연구에서 제시한 알고리즘의 성능을 비교, 분석하고자 두 개 알고리즘의 최대값과 그 값을 얻는 데 걸리는 계산시간을 측정하였다. <Table 2>는 두 개 알고리즘의 각 조합의 10개 문제에 대한 실험 결과의 평균값을 정리한 것이다. GA는 Yoo[12]가 제시한 유전자 알고리즘을 나타내고, Greedy GA는 본 논문에서 제시한 탐욕유전자 알고리즘을 의미한다.

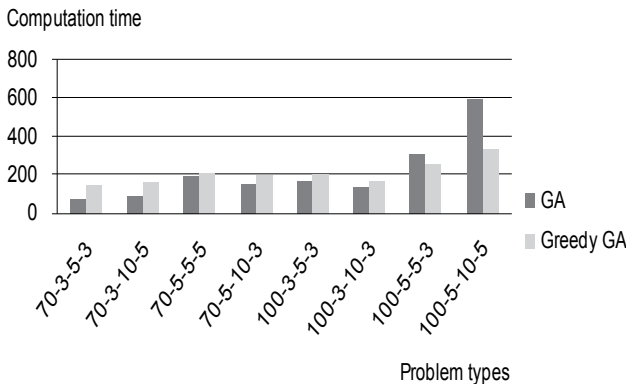
<Table 2>에서 얻어진 실험 결과 데이터를 기반으로 <Figure 2>는 8개 조합에 대한 두 개 알고리즘의 계산시간을 비교한 것이다. 8개 조합 중 앞의 6개 조합에서 GA가 Greedy GA보다 계산시간이 적게 걸리고, 나머지 두 조합(100-5-5-3과 100-5-10-5)에 대해서는 Greedy GA가 GA 대비 각각 17.4%, 43.8% 만큼 계산시간이 줄어든 것을 볼 수 있다. 그 원인은 문제의 크기가 일정 수준 이상 커지게 되면, Greedy GA의 수리오퍼레이터가 본 연구의 수리모형인 우선순위 제약하의 다수 0-1 배낭문제에 잘 작

동하여 가능해를 빨리 찾아주는 것으로 보여진다.

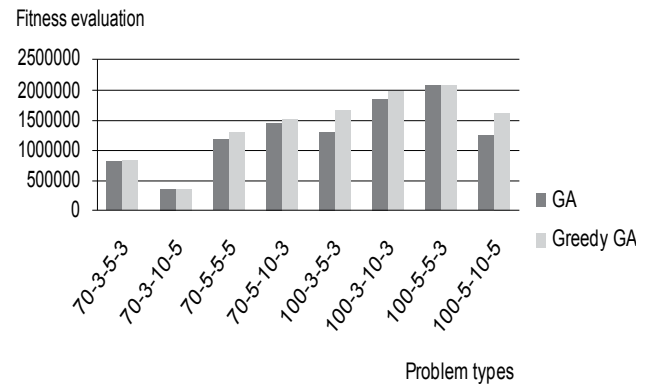
<Figure 3>에서는 GA와 Greedy GA가 8개 조합에서 찾은 각각의 최대값을 비교하여 보여준다. 100-5-5-3 조합을 제외하고 나머지 7개 조합에서 Greedy GA가 GA보다 평균적으로 11.7% 우수한 해를 찾아낸 것을 볼 수 있다. 100-5-5-3 조합의 경우조차 두 개 알고리즘의 최대값의 차이가 0.2%로 매우 미세했다. 그 원인은 수리오퍼레이터가 문제(P)에 잘 작동하여 우수한 가능해를 찾아 주었을 뿐만 아니라, 초기 모집단이 보다 다양한 염색체를 보유하여 우수한 가능해를 찾는 데 큰 영향을 준 것으로 예상된다.

<Figure 4>~<Figure 7>은 본 연구에서 제시한 Greedy GA가 기능의 수, 출시의 수, 플랫폼/제품 수, 기능간 개발 우선순위 제약 규모의 차이에 따른 10^4 세대를 적용한 경우, 최대해를 구하는 데 소요되는 계산시간을 보여준다. <Figure 4>, <Figure 5>, <Figure 7>에 볼 수 있듯이 기능 수, 출시횟수, 기능간 개발 우선순위 제약의 규모가 증가함에 따라 계산시간이 비례하여 증가하는 것을 볼 수 있다.

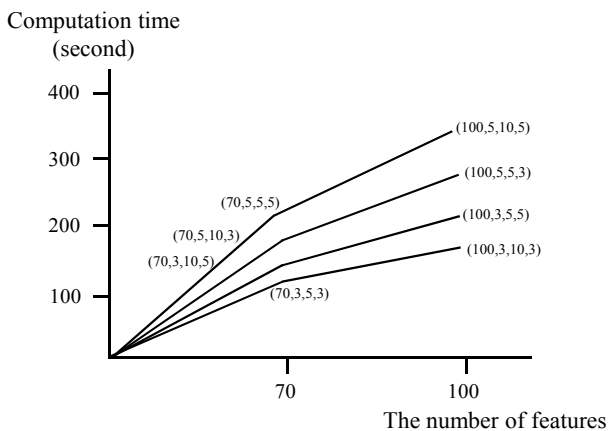
<Figure 6>에서는 플랫폼/제품 수의 증가가 Greedy GA의 계산시간을 반드시 증가시키지 않는 것을 볼 수 있다. 실험 조건 70-5-5-5와 70-5-10-3을 보면 플랫폼/제품 수가 5에



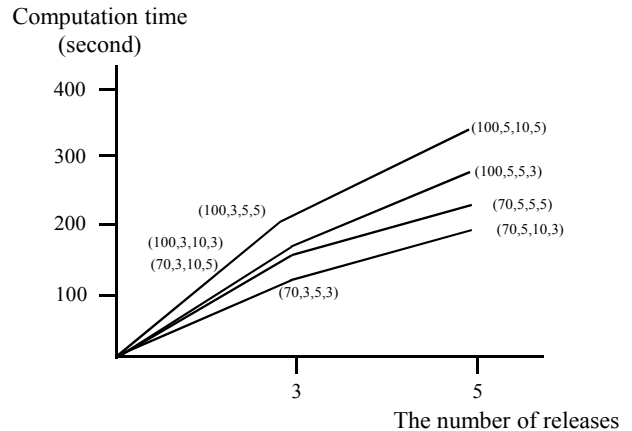
<Figure 2> Computation Time of GA and Greedy GA



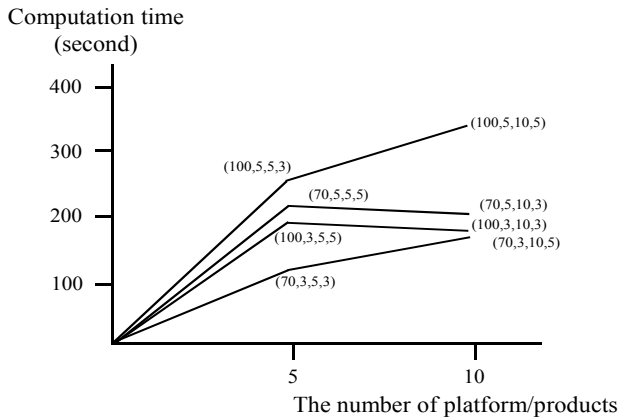
<Figure 3> Fitness Evaluation of GA and Greedy GA



<Figure 4> The Number of Features vs. Computation Time



<Figure 5> The Number of Releases vs. Computation Time



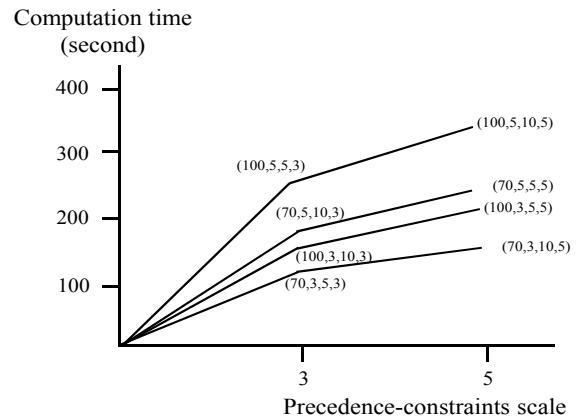
<Figure 6> The Number of Platform/Products vs. Computation Time

서 10으로 증가하였으나, 알고리즘의 계산시간이 214.11초에서 188.91초로 감소한 것을 보여준다. 이것은 플랫폼/제품 수보다 기능간 개발 우선순위의 제약 규모의 차이가 계산시간의 증감에 더 크게 영향을 준다는 것을 알 수 있다. 실험조건 100-3-5-5와 100-3-10-3도 이와 같은 이유로 설명될 수 있다.

5. 결론 및 미래 연구

본 연구는 소프트웨어 제품라인의 출시문제를 풀기 위하여 Yoo[12]가 제시한 유전자 알고리즘을 향상시키고자 수행되었다. Yoo[12]가 제시한 유전자 알고리즘과 본 연구에서 제시한 알고리즘의 구성에서 상이한 부분은 초기 모집단을 구하는 방법과 수리 오퍼레이터 부분이다. 또한, 본 논문에서 제시된 알고리즘의 작동상태(behavior) 및 성능을 파악하고 Yoo[12]가 제시한 알고리즘과의 비교를 위하여 실험계획 기법인 부분요인설계(fractional factorial design)를 적용하였다. 4개 요인에 각 요인이 2개의 값을 갖는 실험을 설계하여 직교배열로부터 얻어진 8 조합으로 각 조합 당 10문제를 발생시켜 실험을 수행하였다.

Yoo[12]가 제시한 유전자 알고리즘은 문제 (P)의 우수한 해를 신속하게 찾을 수 있도록 문제 (P)를 선형계획 모형으로 완화하여 구한 최적해를 이용하여 얻은 우량의 초기 모집단을 이용하였다. 우량 초기 모집단은 알고리즘을 특정 해에 빠르게 수렴하도록 하였다. 그 특정해의 우수성을 알아보기 위하여 본 연구에서는 보다 다양한 유전자를 지닌 염색체를 얻고자 0과 1사이의 균등분포(uniform distribution)를 따르는 난수를 발생하여 무작위 방식으로 초기 모집단을 구하여 본 논문에서 제시한 탐욕 유전자 알고리즘에 적용하였다. 탐욕 유전자 알고리즘이 찾은 해가 평균적으로 Yoo[12]가 제시한 알고리즘의 해 보다 11.7%



<Figure 7> The Scale of Precedence-Constraints of vs. Computation time

우수한 것을 알 수 있었다.

본 연구에서 제시한 탐욕 유전자 알고리즘에 염색체 교체와 염색체 돌연변이를 수행 후 염색체가 불가능해진 경우 이를 가능해로 만들기 위하여 문제(P)에 잘 작동하는 탐욕스러운 휴리스틱을 기반으로 하는 수리 오퍼레이터를 개발하여 알고리즘에 반영하였다. 이 수리 오퍼레이터가 문제의 크기가 일정 수준으로 커질 때 계산시간을 감소시키는 데 상당한 영향을 끼치는 것으로 분석되었다.

실험을 통하여 4개 요인 중 기능 수, 출시횟수, 기능간 개발 우선순위 제약의 규모가 증가함에 따라 본 논문의 탐욕 유전자 알고리즘의 계산시간이 비례하여 증가하는 것을 볼 수 있었으나, 플랫폼/제품 수의 증가는 알고리즘의 계산시간을 반드시 증가시킨다고 보기 어려웠다.

마지막으로, 미래 연구로써 본 연구에서 고려한 우선순위 제약하의 다수 0-1 배낭문제를 보다 효율적으로 풀기 위한 유전자 알고리즘 외에 다른 메타휴리스틱 개발에 관한 연구가 필요해 보이고, 이들 메타휴리스틱 간 성능비교도 흥미로운 연구분야가 될 것으로 보인다.

Acknowledgement

This work was supported by the Dong-A University research fund.

References

- [1] <http://www.sei.cmu.edu/productlines>.
- [2] J.M.V., vanden Akker, S. Brinkkemper, G. Diepen, and J. Versendaal, Determination of the Next Release of a Software Product: an approach using integer linear programming. *Proceeding of the 11th International Workshop on Requirements Engineering, Foundation*

- for Software Quality*, (REFSQ 2005), 2005, p 119-124.
- [3] J.M.V. vanden Akker, S. Brinkkemper, G. Diepen, and J. Versendaal, Software Product Release Planning through Optimization and what-if analysis. *Information and Software Technology*, 2008, Vol. 50, No. 1-2, p 101-111.
 - [4] Kapsalis, A., G.D., Smith, V.J., and Rayward-Smith, Solving the Graphical Stainer Tree Problem Using Genetic Algorithms. *Journal of the Operational Research Society*, 1993, Vol. 44, No. 4, p 397-406.
 - [5] Penny, D., An Estimation-Based Management Framework for Enhance Maintenance in Commercial Software Products. *Proceedings of the International Conference on Software Maintenance*, Montreal, Canada, 2002, p 122-130.
 - [6] Raidl, G.R., An Improved Genetic Algorithm for the Multiconstrained 0~1 Knapsack Problem. *Proceedings of the 5th IEEE International Conference on Evolutionary Computation*, Alaska, USA, 1998, p 207-211.
 - [7] Reeves, C.R., A Genetic Algorithm for Flowshop Sequencing. *Computers and Operations Research*, 1995, Vol. 22, No. 1, p 5-13.
 - [8] Svahnberg, M., T. Gorschek, R. Feldt, R. Torkar, S.B., and Saleem, M.U. Shafiqfue, A Systematic review on strategic release planning models. *Information and Software Technology*, 2010, Vol. 52, No. 3, p 237-248.
 - [9] Taborda, L., Generalized Release Planning for Product Line Architectures. *Proceedings of the SPLC, The Third Software Product Lines Conference*, Boston, USA, 2004, p 238-254.
 - [10] Ullah, M. and Ruhe, G., Towards Comprehensive Release Planning for Software Product Lines. *Proceedings of the First International Workshop on Software Product Management*, Minneapolis/St. Paul, Minnesota, USA, 2006, p 55-59.
 - [11] Yoo, J., An Exact Solution Approach for Release Planning of Software Product Lines. *Journal of the Society of Korea Industrial and Systems Engineering*, 2012, Vol. 35, No. 2, p 57-63.
 - [12] Yoo, J., Release Planning of Software Product Lines Using a Genetic Algorithm. *Journal of the Society of Korea Industrial and Systems Engineering*, 2012, Vol. 35, No. 4, p 142-148.