http://dx.doi.org/10.11627/jkise.2012.35.4.142 pISSN 2005-0461
eISSN 2287-7975

Release Planning in Software Product Lines Using a Genetic Algorithm

Jaewook Yoo†

Department of Business Administration, Dong-A University

유전자 알고리듬을 이용한 소프트웨어 제품라인의 출시 계획 수립

유 재 욱†

동아대학교 경영대학 경영학과 조교수

Release planning for incremental software development is to select and assign features in sequence of releases along a specified planning horizon. It includes the technical precedence inherent in the features, the conflicting priorities as determined by the representative stakeholders, and the balance between required and available resources. The complexity of this consideration is getting more complicated when planning releases in software product lines. The problem is formulated as a precedence-constrained multiple 0-1 knapsack problem. In this research a genetic algorithm is developed for solving the release planning problems in software product lines as well as tests for the proposed solution methodology are conducted using data generated randomly.

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

1. 서 론

소프트웨어 제품 라인(Software Product Lines)은 그 제품라인 안에서 공용되는 플랫폼과 그 플랫폼을 기반으로 하는 다수의 파생제품들로 이루어져 있다. 소프트웨어 제품 라인 개념은 소프트웨어 개발에 있어서 중요한 패러다임(paradigm)이며, 이를 통하여 소프트웨어 개발 회사들은 제품 출시까지의 시간(time-to-market), 비용, 생산성, 그 밖의 주요 지표들을 향상시키려 노력하고 있다[1].

Received 22 October 2012; Accepted 4 December 2012
† Corresponding Author : jyoo@dau.ac.kr

소프트웨어를 점진적으로 개발(incremental software development)하는 데 있어서 소프트웨어 출시를 계획하는 것은 소프트웨어를 이루고 있는 기능들(features)을 고려중인 일정기간 내에 다수의 출시시점(release) 중 적절한 출시시점에 할당하는 계획을 수립하는 것이다. 이 때 만족해야 할 제약들은 기능간 기술적 구현 순서, 소프트웨어 이해관계자들 각각의 기능들에 대한 우선순위, 개발 인적자원 등이다. 이러한 출시계획이 없다면 소프트웨어 제품라인을 이루고 있는 주요기능들은 적절한 시기에 출시되지 못할 수도 있고, 또는 너무 일찍 출시되어 시장에서 불필요하게 되거나 외면당할 수 도 있다. 이는 고객의 불만족, 시간과 예산의 초과, 시장점유의 상실 등을 야기시킨다[4].

기존의 소프트웨어 출시 계획과 달리, 소프트웨어 제품라인에 있어서 소프트웨어 출시 계획은 여러 가지 새롭게 고려해야 할 사항들을 수반 한다. 특히, 플랫폼을

포함하여 파생제품들이 다수라는 것과 이들을 개발하는 팀도 하나가 아니라 플랫폼과 그 파생제품에 따라 다수라는 것이다.

점진적 소프트웨어 개발 패러다임에서 한 개 소프트웨어 출시계획 문제를 정형화된 모델로 모형화하고 이를 풀기 위한 해법을 제시한 연구는 활발히 진행되어 왔으나[2, 3, 6], 소프트웨어 제품라인에 대한 소프트웨어 출시계획의 정형화된 연구는 많지 않다. 또한, 기존의 소프트웨어 출시 계획에 대한 해법이 실제 산업에서 사용 가능할 정도로 큰 크기의 문제에 적용 가능한지를 분석한 연구도 많이 수행되지 않았다[6].

Ullah and Ruhe[8]가 소프트웨어 제품라인의 소프트웨어 출시계획 문제에 대하여 정형화된 모형과 해답을 제시하였지만, 한 개 제품의 출시 계획을 수립하기 위해 기존에 개발된 의사결정 시스템을 이용하는 수준이었고, 그들이 제시한 해답은 최적(optimal)해를 보장하지 못했다. Taborda[7]가 제시한 출시 행렬(release matrix)은 전체적인 관점에서 소프트웨어 제품라인의 출시계획 관리를 다루었다. 그러나 최적해를 구하기 위한 모델과 해법을 제시하지는 않았다. Yoo[9]는 소프트웨어 제품라인의 출시계획을 우선순위 제약하의 다수 0-1 배낭문제로 수리모형화하고 이를 풀기 위하여 동적계획법(dynamic programming)을 개발하고 수치예제를 제시하였으나, 이 알고리듬의 성능을 테스트하기 위한 실험을 수행하지 않았다.

소프트웨어 출시 계획 관련 연구 분야에서 본 연구의 기여는 소프트웨어 제품라인의 출시 계획 문제를 수리모형화한 우선순위 제약하의 다수 0-1 배낭문제를 풀기 위하여 유전자 알고리듬을 적용하고, 이 알고리듬의 타당성을 위하여 성능을 테스트하고 분석한 것이라 하겠다.

본 논문의 제 2장에서는 소프트웨어 제품라인의 출시계획 문제의 수리모형을 소개하고, 제 3장에서는 제 2장에서 언급한 수리모형의 해법 절차로써 유전자 알고리듬을 제시한다. 제 4장에서는 유전자 알고리듬의 성능을 분석하기 위하여 계산 실험을 하고, 제 5장에서는 본 논문의 결론을 짓는다.

2. 수리모형

소프트웨어 제품라인에는 두 가지 종류의 기능들이 있다. 첫 번째는 하나의 플랫폼을 이루고 있는 핵심자산기능(core asset feature)이고, 두 번째는 그 플랫폼을 공통으로 하는 파생제품들의 제품특성기능(product specific feature)이다.

본 연구에서 제시되는 모형에서는 이 들 두 가지 기능들이 이미 결정되어 있고, 핵심자산기능들은 플랫폼의 형태로 소프트웨어 제품라인에 속한 모든 제품들에서 공통으로 구성되어 있고, 제품특성기능들은 소프트웨어 제품라인의 각 제품들의 특성에 맞게 구성되어 있다고 가정한다.

따라서, 플랫폼이나 제품 k를 형성하고 있는 모든 기능들의 집합은 $F(k)$로 정의된다 : $\underset{k}{U}F(k)=\{1, 2, \cdots, I\}$. 이들 기능들은 출시 계획 기간 내 총 T번의 출시 순서 중 하나의 출시 시점에 할당되어 시장에 나갈 수 있다. 이것은 의사결정변수 x_{it}에 의해 결정된다 : 기능 i가 t번째 출시에 할당되면 $x_{it}=1$이고, 기능 i의 출시가 연기되면 $x_{it}=0$이다. 수리모형은 기능들의 출시 순서를 정의하는 기능간 우선순위관계를 반영한다.

C_{kt}는 t번째 출시하게 될 플랫폼 또는 제품 k개발팀의 가용한 개발 자원들이다. 이 제약식은 실제 소프트웨어 개발업체들이 개발조직을 구성할 때 소프트웨어 제품라인에 맞게 구성하고 있는 것을 수리모형에 반영한 것이다. w_{it}는 기능 i를 t번째에 출시하기 위하여 구현하는 데 소요되는 개발 자원의 양이다.

소프트웨어 이해관계자들(stakeholders)은 소프트웨어 출시 계획을 수립하는 데 매우 중요하다. 소프트웨어 이해관계자들의 집합을 $S = \{S(1), S(2), \cdots, S(q)\}$라고 가정하자. 이해관계자 q는 상대 중요도 $\lambda(q)\in\{1, 2, \cdots, 9\}$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. 만점은 9점이다. 첫 번째 기준은 이해관계자 q에게 기능 i가 얼마나 가치가 있는 가를 보여주는 가치도 $value(q, i)$이고 두 번째 기준은 얼마나 긴급한가를 보여주는 긴급도 $urgency(q, i)$이다.

본 연구의 수리모형의 목적은 가용한 개발자원, 기능간 우선순위, 기능선택 제약 식 등을 만족하면서 출시 계획 기간내 다수의 출시 시점에 소프트웨어 제품라인을 이루고 있는 기능들의 출시를 목적식 $P(x)$를 최대화는 출시 계획 x를 찾는 것이다. 여기서 $P(x)$의 값은 각 출시의 중요도, 이해관계자들의 중요도, 이해관계자가 인식하는 각 기능의 긴급도와 가치도 등과 같은 여러 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 t의 중요도이고, 이 중요도는 계획된 총 출시에 대해서 1로 정규화된(normalized to 1)

된다. 본 연구에서 다루어질 소프트웨어 제품라인의 출시 계획수립 문제는 아래 문제 (*P*)에서 목적식과 제약식 (2)~제약식 (6)으로 우선순위 제약하의 다수 0-1 배낭문제로 수리 모형화된다.

문제 (*P*)

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

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

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

$$\sum_{t=1}^{T} x_{it} \le 1 \qquad \text{for all } i \tag{5}$$

$$X_{it} \in \{0, 1\} \qquad \text{for all } i,\ t \tag{6}$$

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. 우선순위 제약 식 (4)는 전체 출시 계획 기간에 걸쳐 소프트웨어 제품라인의 기능간 우선순위를 보장한다. 기능선택 제약 식 (5)는 각 기능들이 전체 출시 계획 기간의 어느 한 출시에 할당되거나, 출시를 지연하도록 한다. 제약 식 (6)은 의사결정변수의 0-1 정수 조건을 의미한다.

3. 제안된 알고리듬

3.1 유전자 알고리듬

본 연구에서 제시한 수리모형을 풀기 위하여 유전자 알고리듬(Genetic Algorithm)을 적용하였다. 다음은 유전자 알고리듬을 단계별로 설명한 것이다.

단계 1 : 초기 모집단(initial population) 구하기

문제 (*P*)의 0-1 정수계획 모델을 선형계획 모델 문제로 완화(LP-relaxed)하여 구한 최적 해($x_{P[j]}^{LP}$)를 이용하여 염색체가 100개인 초기 모집단의 해를 구한다(상세 설명은 제 3.2절 참조).

단계 2 : 적합도 평가(fitness evaluation) 하기

단계 1에서 구한 초기 모집단의 모든 염색체(chromosome)의 적합도를 구한다. 각 염색체는 문제 (*P*)의 가능해이므로 목적식에 가능해를 대입하여 해를 구하여 모집단의 적합도를 알 수 있다.

단계 3 : 알고리듬의 종료 결정하기

현재 모집단의 염색체의 90%가 같은 적합도 값을 같거나 세대(generation)의 수가 이미 지정된 한계의 수를 넘어가면 알고리듬을 종료한다. 이 두 가지 조건을 모두 만족하지 못하면 다음 단계를 수행한다.

단계 4 : 염색체 선택(selection)하기

현재의 모집단에서 랜덤하게 두 개의 염색체(부모염색체)를 선택한다.

단계 5 : 염색체 교체(crossover)하기

부모 염색체로 선택된 두 개의 염색체를 교체 확률이 0.9(Pc = 0.9)인 유니폼 교체(uniform crossover)를 수행한다.

단계 6 : 염색체 돌연변이(mutation)하기

돌연변이 확률을 1/n(여기서, n은 변수의 수)으로 하여 염색체 변이를 수행한다. 교체와 돌연변이단계 수행 후 염색체가 비 가용(infeasible)인 경우 단계 1에서 구한 $x_{P[j]}^{LP}$를 활용하여 가용(feasible)하게 만든다(상세 설명은 3.2 참조). 단계 2로 간다.

<Table 1>은 본 연구에서 제시한 유전자 알고리듬의 특성을 정리한 것이다.

<Table 1> Characteristics of the GA

Encoding	bit string
Selection	Tournament(k = 2)
Mutation	Bitwise(p_m = 1/n)
Recombination	Uniform crossover(p_c = 0.9)
Population size	100
Termination	After 10^5 generations or 90% of the chromosomes in the population has the same fitness value
Improvement	LP-optimized initialization, repair operator

3.2 유전자 알고리듬의 향상

본 연구에서 적용된 유전자 알고리듬의 특징으로는 알고리듬이 보다 신속하게 우수한 해를 찾을 수 있도록 우수한 초기 모집단(population)을 구하는 것이다. Raidl [5]은 다수제약식을 갖는 배낭문제를 풀기 위한 유전자 알고리듬을 적용하였는데 우수한 초기 모집단을 구하기 위해서 0-1 정수 모델인 원 문제를 선형계획 모델 문제로 완화하여 구한 해를 이용하였다. Raidl이 제시한 기법과 같이 본 연구의 수리모형인 문제 (*P*)를 선형계획 모델 문제로 완화(LP-relaxed)하여 구한 최적 해($x_{P[j]}^{LP}$)를 이용하여 우수 초기 모집단을 구한다.

이를 위하여 먼저 모든 i, t에 대하여 x_{it}를 0으로 놓는다. 전체 의사결정변수의 수가 n개 라고 가정하였을 때, 색인 1에서 n까지 인 순열 P를 랜덤하게 n개 발생시켜 의사결정변수 x_{it}를 순열 P의 색인 값에 맞게 배열한다. 이때, 그 배열의 각 변수 $x_{P[j]}$에 문제 (P)를 선형계획모델로 완화(LP-relaxed)하여 구한 해 $x^{LP}_{P[j]}$를 할당한다. 난수(random number) $R_j(0 \leq R_j \leq 1)$를 발생시켜 $x^{LP}_{P[j]} > R_j$ 이면, $x_{P[j]}$의 값을 1로 할당하고, 이것이 문제 (P)의 제약식을 위반하면 0의 값을 갖게 한다.

이와 같이하여 100개의 가용 해를 갖는 초기 모집단을 구한다. <Figure 1>은 위에 설명된 초기 모집단을 구하는 과정을 알고리듬으로 표현한 것이다.

Procedure Initialize x_{ij} :
$x_{ij} \forall i, j \quad \leftarrow 0$;
$P \leftarrow$ random permutation of {1, 2, ⋯, n};
for $j \leftarrow 1$ **to** n **do**
  if $x^{LP}_{P[j]} > R_j$ **then**
    $x_{P[j]} \leftarrow 1$;
    if any constraint is violated **then**
      $x_{P[j]} \leftarrow 0$;
Done;

<Figure 1> Algorithm for Generating an Improved Initial Solution

제 3.1장에서 제시된 교체와 돌연변이 과정을 거친 후 비 가용해가 발생하였을 때 적합도 평가를 수행하기 전에 이를 가용해로 만든다. 이를 위해 색인이 1에서 n까지의 순열 P를 발생시켜 의사결정변수 x_{it}를 랜덤하게 발생된 순열 P대로 배열한다. 이 배열을 선형계획 모형의 해들 ($x^{LP}_{P[j]}$)의 오름차순으로 정렬시킨다. 같은 값을 갖는 색인이 있다면 랜덤하게 섞는다. 1의 값을 갖는 모든 $x_{P[j]}$가 어떤 제약 식을 만족시키지 못하면 0의 값을 갖게 된다. 최악의 경우 모든 변수 값이 0의 값을 갖게 될 수도 있다. <Figure 2>는 위에 설명한 비 가용해를 가능해로 만드는 알고리듬을 보여준다.

Procedure Repair x_{ij} :
$P \leftarrow$ random permutation of {1, 2, ⋯, n} with $x^{LP}_{P[j]} \leq x^{LP}_{P[j+1]}$ (j = 1, ⋯, n-1);
for $j \leftarrow 1$ **to** n **do**
  if $x_{P[j]} = 1$ **and** any constraint is violated **then** $x_{P[j]} \leftarrow 0$;
Done;

<Figure 2> Algorithm for Making a Solution Feasible

4. 실험 조건 및 결과

본 연구에서 제시된 유전자 알고리듬의 성능실험을 위하여 알고리듬을 Matlab(2012a 버전)으로 코딩하였다. 인텔코어 i3-2100 CPU(3.10 GHz)를 장착한 1.90 GB 램(RAM)을 탑재한 PC가 실험에 사용되어졌다. 제안된 알고리듬의 작동상태 (behavior)를 파악하기 위하여 다음과 같이 4개 요인에 각 요인이 2개의 값을 갖는 조합으로 알고리듬이 적용될 문제를 발생시켰다.

(i) 제품라인 내 기능의 총수 ∈ {70, 100}
(ii) 출시 횟수 ∈ {3, 5}
(iii) 제품라인 내 플랫폼/제품의 총수 ∈ {5, 10}
(iv) 가용예산의 제약비율 α ∈ {0.25%, 0.5%}

가용예산의 제약비율 α는 $a\sum_i w_{it}$으로 예산이 제약된다.

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

위 조합의 총 수가 16개로 너무 많아 최소한의 실험 횟수로 원하고자 하는 분석결과를 얻고자 실험계획 기법인 부분요인설계(fractional factorial design)를 적용하였다. 본 연구의 실험에는 4개의 요인(factor)이 있고, 각 요인은 2개의 레벨(level)을 갖는 것으로 설계하였으므로, 8개의 실험조건이 실험계획의 직교배열(orthogonal array)로 부터 얻어진다. <Table 2>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.

<Table 2> 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>의 행은 실험 조건을 나타내며, 열은 요인을 의미하는 데 실험을 통하여 이들의 효과가 분석되어진

다. <Table 2> 내에 기재된 값 0과 1은 요인이 갖는 레벨을 보여준다. 각 행의 조합당 10개의 문제를 발생하였고 총 80개의 문제에 대하여 실험이 수행되어졌다. 본 실험에서 각 조합에 대하여 10개씩 총 80개 문제의 초기해와 10^3, 10^4, 10^5세대(generation)를 통하여 해가 얻어졌다. 알고리듬의 계산 성능은 계산시간(초 단위)으로 측정되었고 각 세대별 전 세대 해에 대비 현 세대 해의 향상률을 계산하였다. <Table 3>은 각 조합의 10개 문제에 대한 실험 결과의 평균값을 정리한 것이다.

문제의 크기는 변수의 수와 제약식 수의 함수로 볼 수 있다. 그러므로 기능의 수, 출시의 수, 플랫폼/제품의 수에 비례하여 문제의 크기가 증가하는 것을 알 수 있다. 기능의 수의 증가는 변수의 수와 문제 (P)의 우선순위 제약식 (4)와 기능선택 제약식 (5)의 수를 증가시키고, 출시 횟수는 변수의 수와 자원 제약식 (3)의 수를 증가시킨다. 또한, 플랫폼/제품 수의 증가는 자원 제약식 (3)의 수를 증가시킨다.

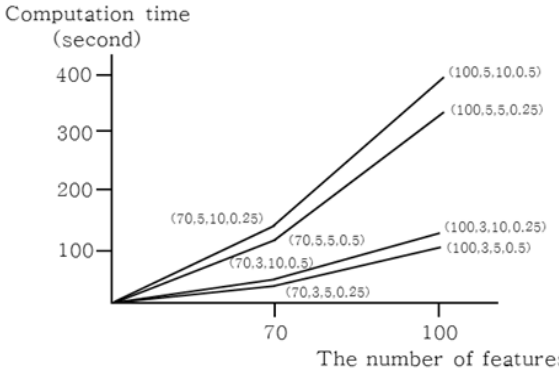
<Figure 3>~<Figure 6>은 8개 조합에 대한 알고리듬의 104 세대 적용 후 기능의 수, 출시의 수, 플랫폼/제품 수, 가용예산 제약비율의 증가에 따른 알고리듬의 계산시간을 보여준다. <Figure 3>~<Figure 5>를 보면 기능의 수, 출시의 수, 플래폼/제품 수의 증가는 알고리듬이 문제를 푸는 데 걸리는 계산 시간을 증가시키는 것을 알 수 있다. 특히, <Figure 3>, <Figure 4>에서 기능의 수와 출시의 수의 증가는 계산 시간을 기하급수적으로 증가시키는 것을 볼 수 있다. <Figure 5>에서는 플랫폼/제품 수의 증가가 계산 시간을 기하급수적으로는 아니나 비례하여 증가 시키는 것을 볼 수 있다.

<Table 3> Experiment Results

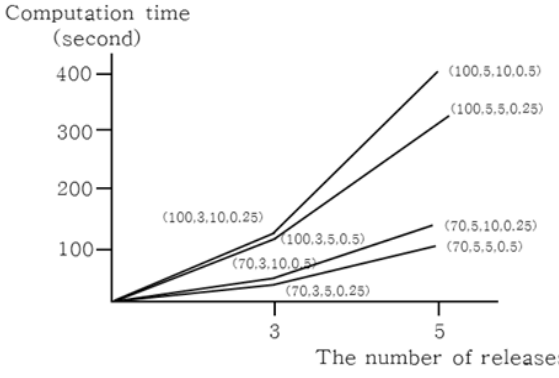
Feature	Release	Platform/Product	Tightness ratio	Generation	CPU time (second)	improvement rate (%)
70	3	5	0.25	initial solution	0	-
				10^3	5	89
				10^4	47	9
				10^5	433	2
70	3	10	0.5	initial solution	0	-
				10^3	6	61
				10^4	51	10
				10^5	455	7
70	5	5	0.5	initial solution	0	-
				10^3	15	30
				10^4	111	5
				10^5	1076	3
70	5	10	0.25	initial solution	0	-
				10^3	15	86
				10^4	133	16
				10^5	1068	1
100	3	5	0.5	initial solution	0	-
				10^3	15	123
				10^4	120	13
				10^5	1091	6
100	3	10	0.25	initial solution	0	-
				103	14	57
				104	123	12
				105	1141	1
100	5	5	0.25	initial solution	1	-
				10^3	37	86
				10^4	306	17
				10^5	2836	17
100	5	10	0.5	initial solution	1	-
				10^3	43	73
				10^4	381	4
				10^5	3182	2

그러나, 가용예산의 제약비율이 알고리듬의 계산시간과 비례관계에 있다고 보기 어렵다. <Figure 6>에서 보는 바와 같이 실험 조건 100-3-10-0.25(기능수-출시수-플랫폼/제품수-가용예산제약비율)와 70-5-10-0.25 각각의 계산시간(123초, 133초)이 실험 조건 100-3-5-0.5와 70-5-5-0.5 각각의 계산 시간(120초, 111초)보다 크기 때문이다. 이는 플랫폼/제품 수의 증가로 인한 제약식 수의 증가가 알고리듬의 이들 실험조건의 계산시간을 더욱 증가시키는 요인임을 알 수 있다.

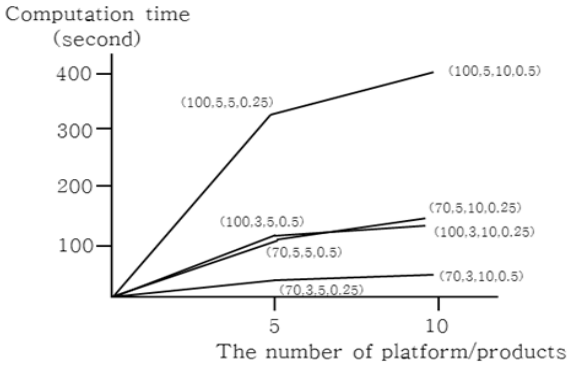
세대의 수가 클수록 해가 향상되었고, 각 실험조합의 초기해 대비 10^3세대 직후의 해가 전 실험에 걸쳐 30%에서 123%까지 향상되는 것으로 보아 우수한 초기 해를 얻기 위해 문제 (P)를 선형계획 모델 문제로 완화(LP-relaxed)하여 구한 최적 해($x_{P[j]}^{LP}$)를 이용하여 구한 초기 모집단이 어느 정도는 효과가 있었지만, 상당하지는 않아 보인다. 그 원인은 다수의 제약 식으로 인하여 선형계획 모델 문제로 완화(LP-relaxed)하여 구한 최적 해($x_{P[j]}^{LP}$)와 문제 (P)의 최적해 (x_{it})와의 간격이 크게 벌어진 것으로 보인다.



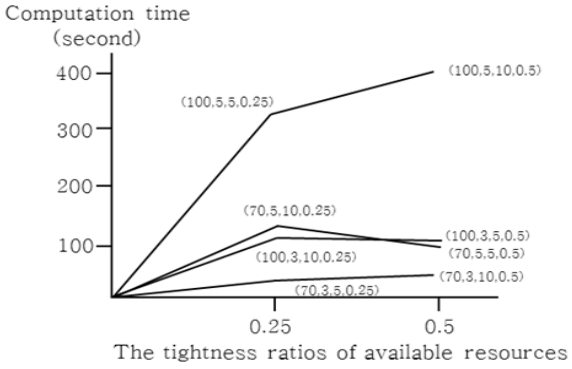
<Figure 3> The Number of Features vs. Computation Time



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



<Figure 5> The Number of Platform/products vs. Computation Time



<Figure 6> The Tightness Ratio of Available Resources vs. Computation Time

5. 결론 및 미래연구

우선순위 제약하의 다수 0-1 배낭문제로 수리 모형화된 소프트웨어 제품라인에 있어서 소프트웨어 출시 계획 문제를 유전자 알고리듬을 이용하여 그 해를 찾았다. 유전자 알고리듬의 특성은 <Table 1>과 같고, 유전자 알고리듬이 해를 보다 효율적으로 찾을 수 있도록 우선순위 제약하의 다수 0-1 배낭문제인 문제 (P)를 선형계획 모델 문제로 완화(LP-relaxed)하여 구한 최적 해($x_{P[j]}^{LP}$)를 이용하여 우수한 초기 모집단을 구하고자 시도하였다.

본 연구에서 제시된 유전자 알고리듬의 소프트웨어 제품라인에 있어서 소프트웨어 출시 문제에 대한 해를 찾는 성능을 분석하기 위하여 부분요인설계 (fractional factorial design)를 적용하여 직교배열 OA(8, 4, 2, 3) 실험을 계획하고 수행하였다.

실험을 통하여 문제의 크기가 증가할수록, 즉 변수의 수와 제약식의 수가 증가할수록 알고리듬이 해를 찾는데 걸리는 계산 시간이 증가하는 것을 알 수 있었다. 제

품라인 내 기능의 수, 출시의 수, 제품라인 내 플랫폼/제품 수의 증가는 알고리듬의 계산시간의 증가에 직접적인 영향을 준 반면, 가용예산의 제약비율은 알고리듬의 계산시간과 직접적인 관련이 없어 보였다.

세대의 수가 클수록 해는 향상되었고, 문제 (*P*)를 선형계획 모델 문제로 완화(LP-relaxed)하여 구한 최적 해 ($x_{P[j]}^{LP}$)를 이용한 초기 모집단으로 시작한 유전자 알고리듬의 성능은 크게 효과적이지는 않아 보였다.

본 연구에 기여할 만한 미래연구는 본 연구에서 논의된 우선순위 제약하의 다수 0-1 배낭문제에 보다 적합한 유전자 알고리듬의 특성을 찾는 것이다. 이 특성을 찾는 과정에서 본 연구에 적용된 부분요인설계(fractional factorial design)를 적용하여 직교배열 실험을 계획하고 수행하면 보다 효율적으로 우선순위 제약하의 다수 0-1배낭문제에 적합한 유전자 알고리듬의 특성을 찾을 수 있으리라 예상된다.

Acknowledgements

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, Brinkkemper, S., Diepen, G., and Versendaal, J., 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] vanden, J.M.V., Akker, Brinkkemper, S., Diepen, G., and Versendaal, J., Software Product Release Planning through Optimization and what-if analysis. *Information and Software Technology*, 2008, Vol. 50, No. 1-2, p 101-111.

[4] Penny, D., An Estimation-Based Management Framework for Enhancive Maintenance in Commercial Software Products. *Proceedings of the International Conference on Software Maintenance*, Montreal, Canada, 2002, p 122-130.

[5] 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.

[6] Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Saleem, S.B., and Shafiqfue, M.U., A Systematic review on strategic release planning models. *Information and Software Technology*, 2010. Vol. 52, No. 3, p 237-248.

[7] 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.

[8] 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.

[9] 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.