<학술논문> **DOI:10.3795/KSME-A.2010.34.2.191**

비스플라인 분지한계법 기반의 전역최적화 알고리즘 개발

박 상 근*†
* 충주대학교 기계공학과

Development of a Branch-and-Bound Global Optimization Based on B-spline Approximation

Sangkun Park*†
*Dept. of Mechanical Engineering, Chungju National Univ.

(Received October 7, 2009 ; Revised December 22, 2009; Accepted December 22, 2009)

Key Words : Global Optimization(전역 최적화), Branch-and-Bound(분지한계법), B-spline Approximation(비스플라인 근사법)

초록: 본 연구는 비스플라인 근사기법을 사용한 분지한계법 기반의 새로운 전역 최적화 알고리즘에 관한 것이다. 본 연구에서는 알고리즘 구성 요소 및 이들의 구현 내용에 관한 상세히 설명한다. 핵심 요소로서, 상호 분리되는 부공간으로의 설계 공간의 분할 작업이 있고, 이들 분할 부공간의 한계값 계산 작업이 있는데, 이들 모두는 실수형 비스플라인 볼륨모델에 의해 구현된다. 본 연구 알고리즘은 다양한 테스트 문제들을 가지고 해의 정확성, 함수호출 회수, 알고리즘 수행시간, 메모리 사용량, 알고리즘 수렴성 등 그 계산 성능들을 평가한다. 이러한 평가 결과는 제안 알고리즘이 직관에 의존하지 않는 완전 알고리즘이며, 대용량의 최적화 문제에도 높은 가능성이 있음을 보여주는 것이다.

Abstract: This paper presents a new global optimization algorithm based on the branch-and-bound principle using B-spline approximation techniques. It describes the algorithmic components and details on their implementation. The key components include the subdivision of a design space into mutually disjoint subspaces and the bound calculation of the subspaces, which are all established by a real-valued B-spline volume model. The proposed approach was demonstrated with various test problems to reveal computational performances such as the solution accuracy, number of function evaluations, running time, memory usage, and algorithm convergence. The results showed that the proposed algorithm is complete without using heuristics and has a good possibility for application in large-scale NP-hard optimization.

1. 서 론

본 연구에서 다루는 최적화 문제는 아래와 같이 실수형 설계변수에 의해 정의되는 비선형 최적화 문제로서 설계변수의 한계범위가 지정된 전역 최적화[1,2] 문제이다.

$$\text{Minimize} \quad f(\mathbf{x})$$
$$\text{subject to} \quad \mathbf{x} \in [\mathbf{x}_{\min}, \mathbf{x}_{\max}] \quad (\text{or} \quad \mathbf{x}_{\min} < \mathbf{x} < \mathbf{x}_{\max})$$

여기서 $\mathbf{x}$ 는 설계변수 벡터를 가리키며, $[\mathbf{x}_{\min}, \mathbf{x}_{\max}]$는 설계변수 한계범위(또는 탐색공간)를 의미한다. 그리고 $f(\mathbf{x})$는 일반적인 비볼록(non-convex) 비선형 목적함수를 나타낸다.

† Corresponding Author, skpark@cjnu.ac.kr

위의 비선형 최적화 문제의 해를 구하기 위하여, 본 연구에서 제시하는 전역 최적화 알고리즘은 다양한 응용분야에서 기본 프레임 혹은 원리로서 폭넓게 인정받고 있는 분지한계법(branch-and-bound) 방식[3~6]에 기반을 두고 있다. 이 방식은 주어진 설계공간(design space)을 반복해서 작은 크기의 부공간(subspace)으로 분할하고, 분할된 각 부공간에서 현재까지 찾은 최선해(best solution) 보다 개선된 해를 찾을 수 없다면 바로 조기에 그 부공간을 삭제하여 탐색 공간을 줄여나가는 방식으로 해를 탐색하는 방법론(기본틀)을 말한다. 여기서 부공간의 삭제는 부공간에서의 목적함수 상한값(upper bound)과 하한값(lower bound) 계산 결과를 바탕으로 최선해와의 비교평가를 통하여 결정하며, 분할이란 마치 나무에서 새 가지가 뻗어나오 듯, 자료

구조 측면에서 트리(tree) 구조의 현재 노드(node) 밑에 자식 노드들이 생성됨을 말한다. 결국, 분할을 통하여 새 가지(노드)가 만들어지고, 이렇게 분할된 각 노드(부공간)에서 상한값과 하한값을 계산하며, 최선해와의 비교를 통하여 삭제여부를 결정한다.

본 연구에서는 이러한 분할 및 상하한값 계산을 비스플라인 근사모델[7]을 사용하여 효과적으로 수행하는 비스플라인 근사모델 기반의 알고리즘을 제시하고자 한다. 여기서 효과적이란 구현 알고리즘의 성능평가 결과를 근거로 판단하는데, 이를 위한 적용 예제로서 전역 최적화 분야에서 난제로서 흔히 사용되는 테스트 문제들을 사용하며, 알고리즘 성능평가 기준으로서 해의 정확성, 함수호출 회수, 알고리즘 수행시간, 메모리 사용량, 그리고 알고리즘 수렴성 등을 사용한다. 또한 기존의 타 연구 결과와의 비교를 통하여 본 연구 알고리즘의 우수성을 보이고자 한다. 결국, 이러한 난제들에 관한 다양한 측면에서의 성능분석 및 타기법과의 비교를 통하여, 본 연구 알고리즘에서 구한 해가 전역 최적해로서 적절하며 비스플라인 근사모델에 의한 계산 방식이 분지한계법 구현에 효과적임을 보이고자 한다.

먼저 본 연구 2 장에서 비스플라인 볼륨모델 및 근사기법에 관해 기술하고, 3 장은 제안 알고리즘에 관하여 상세히 설명하며, 4 장은 적용예제로서 알고리즘 성능분석, 기존 방법과의 비교 등에 관해 기술한다. 끝으로 5 장 결론은 본 연구 알고리즘의 특징 및 향후 계획에 관해 서술한다.

2. 비스플라인 근사모델

2.1 비스플라인 볼륨모델

비스플라인 볼륨모델은 기존 NURBS[8]의 매개변수 공간을 임의의 D 차원으로 확장하고, 조정점의 차원을 임의의 K 개로 확장한 매개변수형 볼륨 표현모델이다. 본 연구에서는 매개변수 공간을 설계변수 공간으로 사용하고, 조정점 벡터를 목적함수 스칼라 값으로 사용하여, 식 (1)과 같이, D 차원 설계변수 공간 $\mathbf{x}$ 에서 1 차원 스칼라 목적함수로 매핑하는 비스플라인 표현모델을 정의한다.

$$f(x_1,\cdots,x_D)=\sum_{i_1=0}^{n_1-1}\cdots\sum_{i_D=0}^{n_D-1} f_{i_1\cdots i_D}N_{i_1}^{k_1}(x_1)\cdots N_{i_D}^{k_D}(x_D) \quad (1)$$

여기서 x_j 는 j 번째 설계변수이고, $f_{i_1\cdots i_D}$ 는 목적함수의 조정값(control value)이며, n_j 와 k_j 는 x_j 방향으로의 조정점의 개수 및 차수(order)이고, $N_{i_j}^{k_j}(u_j)$ 는 차수가 k_j 인 정규화된 비스플라인 기저함수로서 아래의 절점 벡터(knot vector) 상에서 정의된다. 즉,

x_1 방향 절점 벡터, $\mathbf{T}_1=\left\{t_{i_1}^{(1)}\right\}_{i_1=0}^{i_1=n_1+k_1-1}$

……

x_D 방향 절점 벡터, $\mathbf{T}_D=\left\{t_{i_D}^{(D)}\right\}_{i_D=0}^{i_D=n_D+k_D-1}$

2.2 비스플라인 근사기법

순서없이 산만하게 분산된 샘플점과 이에 해당하는 함수값 데이터가 주어졌을 때, 이를 식 (1)에 의해 근사적으로 표현하는 근사모델 생성기법을 소개하면 다음과 같다. 먼저 간결한 수식 전개를 위하여 식 (1)를 식 (2)와 같이 작성한다.

$$f(\mathbf{x})=\sum_{J=0}^{N_c-1} f_J\phi_J(\mathbf{x}) \quad (2)$$

여기서 $\mathbf{x}=(x_1,\cdots,x_D)$ 는 D 차원의 설계변수 벡터이고, $N_c=n_1\times\cdots\times n_D$ 은 조정값의 개수이며, J 는 조정값의 인덱스(index)로서 $f_{i_1\cdots i_D}$ 의 아래 첨자로부터 다음과 같이 구한다.

$$J=i_1+(n_1)\times i_2+(n_1\times n_2)\times i_3+\cdots+(n_1\times\cdots\times n_{D-1})\times i_D$$

그리고 $\phi_J(\mathbf{x})$ 는 각 방향의 비스플라인 기저함수들의 곱이다. 즉,

$$\phi_J(\mathbf{x})=N_{i_1}^{k_1}(x_1)\cdots N_{i_D}^{k_D}(x_D) \quad (3)$$

식 (2)를 가지고 주어진 분산형 데이터를 근사적으로 표현하기 위하여 다음과 같은 행렬식 형태의 선형대수 방정식을 유도한다. 즉, N_p 개의 샘플점 $\mathbf{P}_i=(p_i^1,\cdots,p_i^D)$ $(i=0,\ldots,N_p-1)$와 해당하는 함수값 Q_i 을 식 (2)에 대입하여 정리하면 아래의 식 (4)와 같다.

$$\{Q_I\}=[\phi_{IJ}=\phi_J(\mathbf{P}_I)]\{f_J\} \quad (4)$$

여기서 $(I=0,\cdots,N_p-1)$, $(J=0,\cdots,N_c-1)$이며, $[\phi_{IJ}]$는 $N_p\times N_c$ 행렬이다.

위의 식 (4)에서 $N_p > N_c$ 인 경우, 최소자승법을 적용하면 아래의 식 (5)가 유도되고, 여기서 미지수 $\{f_J\}$을 결정한다.

$$\{f_J\}=([\phi_{IJ}]^T[\phi_{IJ}])^{-1}[\phi_{IJ}]^T\{Q_I\} \quad (5)$$

그리고 $N_p < N_c$ 인 경우, Pseudo-inverse 기법에 의

해 식 (6)가 유도되고, 마찬가지로 미지수 $\{f_J\}$을 결정한다.

$$\{f_J\} = [\phi_{IJ}]^T ([\phi_{IJ}][\phi_{IJ}]^T)^{-1}\{Q_I\} \quad (6)$$

식 (5)의 $[\phi_{IJ}]^T[\phi_{IJ}]$ 와 식 (6)의 $[\phi_{IJ}][\phi_{IJ}]^T$ 는 각각 대칭 행렬로서 양의 정부호 행렬(positive definite matrix)이므로 유일해를 구할 수 있다. 여기서 사용된 e 방향의 절점 벡터 $\mathbf{T}_e = \{t_j^{(e)}\}_{j=0}^{j=n_e+k_e-1}$ $(e = 1, \cdots, D)$는 다음과 같이 계산된다.[8,9]

▶ 절점벡터 앞부분 $(j = 0, \cdots, k_e - 1)$

$$t_0^{(e)} = \cdots = t_{k_e-1}^{(e)} = p_{\min}^{e}$$

▶ 절점벡터 뒷부분 $(j = n_e, \cdots, n_e + k_e - 1)$

$$t_{n_e}^{(e)} = \cdots = t_{n_e+k_e-1}^{(e)} = p_{\max}^{e}$$

▶ 절점벡터 중간부분 $(j = k_e, \cdots, n_e - 1)$

$$\mathrm{d} = (N_p - 1) / (n_e - k_e + 1)$$
$$i = \mathrm{int}\,((k_e - 1 + j) \times \mathrm{d})$$
$$\alpha = (k_e - 1 + j) \times \mathrm{d} - i$$
$$t_j^{(e)} = (1 - \alpha) \cdot p_i^e + \alpha \cdot p_{i+1}^e$$

여기서 $p_{\min}^{e}$ 와 $p_{\max}^{e}$ 는 $p_{\min}^{e} \le p_i^e \le p_{\max}^{e}$ $(i = 0, \cdots, N_p - 1)$가 만족되도록 사용자가 설정할 수 있다.

A Branch-and-Bound Global Optimization

Input f(**x**) // objective function
$\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$ // search domain (or feasible domain)
ε_f and ε_x // tolerances for convergence test
N // no. of sample points
M // no. of splitting nodes along each axis
Output **x*** // searched solution

```
Algorithm Global_Minimize( inputs, output )
Xk = Initialize( )
while( Xk ≠ 0 ) do {
   Branch_and_Bound( Xk )
   Xk = Search_and_Prune( Xk )
   Xk = Search_and_Prune( X0 )
}
x* = FindSolution( )   // output
Terminate( )
```

Fig. 1 Overall view of the proposed algorithm

3. Branch-and-Bound 최적화 알고리즘

3.1 알고리즘 스케치

본 연구 제안 알고리즘은 Fig. 1 과 같이 반복수행 순환구조(while-do) 안에 두 가지 알고리즘이 순차적으로 수행된다. 본 알고리즘의 입력자료는 4.3 절을 참고바라며, 출력은 입력된 허용공차 범위 안에서 도출된 근사 전역해이다. 알고리즘 수행순서대로 작업 내용을 살펴보면 다음과 같다.

○ 초기화 (Fig. 2)

본 알고리즘에 사용되는 전역변수 및 파라미터를 설정하며, 탐색트리 구축을 위한 최초 노드인 root 노드(X^0)를 생성한다.

○ 공간 분할 및 상하한값 계산 (Fig. 3)

현재노드(X^k)의 탐색공간 안에 존재하는 목적함수 분포를 비스플라인 근사모델($A^k(X)$)로 표현하고, 이를 분할한 후, 분할된 부공간(자식노드)의 상하한값을 계산한다. (상세 내용은 3.2 절 참조바람)

```
Algorithm Initialize( )
{
  // initialize the global variables and parameters
  n ← 1        // n = no. of nodes generated
  m ← MD       // m = no. of child nodes
               // D = no. of design variables
  // make a root node for the tree construction
  create a root node, X0
  XL(X0) ← xmin and XU(X0) ← xmax
  LB(X0) ← –∞ and UB(X0) ← ∞
  Xbest ← X0 and Xk ← X0
  return Xk
}
```

Fig. 2 Algorithm for the initialization of global variables and parameters

```
Algorithm Branch_and_Bound( Node Xk )
{
   do Bspline_Approximation( Xk, Ak(X) )
   do Subdivide_Space( Ak(X), {Ai(X)} )
   do Compute_Bounds( Xk, {Ai(X)} )
}
```

Fig. 3 Algorithm for node branching and its bound calculation

```
Algorithm Search_and_Prune( Node X^k )
{
  do Recaluate_ParentBound( X^k )
  do Search_BestNode( X^k, f^best )
  do Prune_Nodes( X^k, f^best )
  do Find_NextNode( X^k, X^next )
  do Test_Convergence( X^next )
  return X^next
}
```

Fig. 4 Algorithm for node searching and pruning

```
Algorithm FindSolution( )
{
  return x* = 0.5*( XL(X^best) + XU(X^best) )
}
```

Fig. 5 Algorithm for finding the optimal solution from X^{best}

```
Algorithm Bspline_Approximation( Node X^k , Model
A^k(X) )
{
  // generate sample points
  By LHS method, we generate sample points, x_i
  (i=0,...,N-1) existing in the current space {X^k}

  // evaluate objective function at the sample points
  We obtain the values of the objective function f(x) at
  the sample points, i.e., y_i=f(x_i)

  // build an approximate model to supply bounds
  information
  We generate a B-spline hypervolume, A^k(X), with the
  (x_i, y_i) (i=0,...,N-1)
}
```

Fig. 6 Algorithm for B-spline approximation

○ 탐색트리 및 노드삭제 (Fig. 4)

현재노드 및 그 아래의 트리구조에서 상한값이 최소인 최선노드(X^{best})를 찾고, 이 최소 상한값보다 높은 하한값을 갖는 노드를 삭제한다. 그리고 다음의 탐색공간을 선정하기 위하여 최소의 하한값을 갖는 다음노드(X^{Next})를 탐색한다. 마지막으로 다음노드의 수렴여부를 확인하고 만약 수렴했다면 0(null)을 반환한다. (상세내용은 3.3 절 참조바람)

```
Algorithm Subdivide_Space( Model A^k(X), ModelSet
{A_i(X)} )
{
  // we split A^k(X) into A_i(X) (i=0,...,m-1)
  insert A^k(X) to A_List.
  for each e = 1,...,D, do {
    for each A(X) in A_List, do {
      subdivide A(X) to S^i(X) (i=0,...,M-1) uniformly
      along the e-th axis.
      insert all S^i(X) to S_List.
    }
    A_List ← S_List  and clear the S_List
  }
  {A_i(X)} ← A_List  // A_i(X) is the i-th element of A_List.
}
```

Fig. 7 Algorithm for subdividing B-spline approximate model covered by X^k

```
Algorithm Compute_Bounds( Node X^k , ModelSet
{A_i(X)} )
{
  for each i = 0,...,m-1, do {
    create the child node X^{n+I} below X^k
    compute XL(X^{n+i}) and XU(X^{n+i}) from A_i(X)
    compute LB(X^{n+i}) and UB(X^{n+i}) from A_i(X)
    delete A_i(X)
  }  // it holds that {X^k} = {X^n} U {X^{n+1}} U ... {X^{n+m-1}}
  n ← n+m
}
```

Fig. 8 Algorithm for computing lower and upper bounds of each child subspace

○ 최선노드에서 최적해 계산 (Fig. 5)

현재까지 탐색된 최선노드를 가지고 최적해를 구한다. 여기서 국부 최적화 기법을 사용하여 해를 구할 수 있으나, 간단하게 최선노드의 최소점(XL)과 최대점(XU)을 평균하여 계산한다.

3.2 공간 분할 및 상하한값 계산

○ 비스플라인 근사모델의 생성 (Fig. 6)

LHS 방법[10]을 사용하여 샘플점을 추출하고, 각 샘플점에서의 목적함수 값을 구한 다음, 이를 2.2 절의 비스플라인 근사기법을 사용하여 근사모델을 생성한다.

○ 근사모델의 분할 (Fig. 7)

각 설계변수 축을 따라 M 개의 분할 공간이 생성되도록 균등하게 근사모델을 분할한다. 여기서 생성되는 분할 근사모델의 개수는, D 차원의 설계공간일 경우, $m = M^D$ 이다. 그리고 비스플라인 모델의 분할은 절점삽입(knot insertion)[8]을 통해 구현된다.

○ 분할된 자식노드의 상하한값 계산 (Fig. 8)

분할된 각 근사모델로부터 현재노드 밑의 자식노드를 생성하며, 동시에 자식노드의 탐색공간(XL, XU) 및 상하한값(LB, UB)을 계산한다. 여기서 탐색공간은 근사모델의 절점벡터가 정의하는 파라미터 공간이며, 상하한값은 근사모델의 조정값들 중에서 최대값을 상한값으로, 최소값을 하한값으로 설정한다. 그 근거는 식 (1)과 같은 비스플라인 모델에는 convex hull[8]이라는 성질이 있기 때문이다.

3.3 탐색트리 및 노드삭제

○ 부모노드의 상하한값 재계산 (Fig. 9)

입력노드(X^k)의 상하한값 변화에 영향을 받는 부모노드의 상하한값을 재계산한다. 이는 상하한값의 차이를 좁히기 위함인데 그 차이가 작을수록 반복수행 회수가 감소하여 빠른 수렴 및 정밀해를 기대할 수 있기 때문이다.

```
Algorithm Recaluate_ParentBound( Node X^k )
{
  repeat until X^k ≠ X^0    // X^0 denotes the root node
  {
    X^parent ← the parent of X^k
    X^jth_child ← the j-th child of X^parent
    UB(X^parent) = max( UB(X^1st_child), . . . , UB(X^last_child) )
    LB(X^parent) = min( LB(X^1st_child), . . . , LB(X^last_child) )
    X^k ← X^parent
  }
}
```

Fig. 9 Algorithm for updating all the parents for tighter bounding

```
Algorithm Search_BestNode( Node X^k, float f^best )
{
  f^best ← UB(X^best)
  for each leaf node under X^k
    if UB(X^leaf) < f^best
    then f^best ← UB(X^leaf) and X^best ← X^leaf
}
```

Fig. 10 Algorithm for searching the best node

○ 최선노드의 탐색 (Fig. 10)

입력노드(X^k) 아래의 모든 leaf 노드 각각에 대해, leaf 노드의 상한값이 현재까지 구한 최선노드(X^{best})의 상한값보다 작다면 그 leaf 노드를 최선노도로 설정한다.

○ 노드 삭제 (Fig. 11)

입력노드 및 그 아래의 트리구조에서 해가 존재치 않는 모든 자식노드를 삭제한다. 즉 삭제노드의 하한값은 최선노드의 상한값보다 크다.

○ 다음노드의 탐색 (Fig. 12)

새로운 반복수행을 위한 다음노드(X^{next})를 탐색

```
Algorithm Prune_Nodes( Node X^k, float f^best )
{
  // prune the nodes having no solution in its space
  if LB(X^k) > f^best, then remove X^k and its all children
  else {
    for each child node X^child under X^k   // DFS is used
      if LB(X^child) > f^best
      then remove X^child and its all children
  }
}
```

Fig. 11 Algorithm for node pruning

```
Algorithm Find_NextNode( Node X^k, Node X^next )
{
  f^next ← LB(X^best)
  if X^k is 0, then X^k ← the parent of X^k
  for each leaf node under X^k
    if LB(X^leaf) < f^next
    then f^next ← LB(X^leaf) and X^next ← X^leaf
}
```

Fig. 12 Algorithm for selecting the next node which new iteration starts from

```
Algorithm Test_Convergence( Node X^k )
{
  // test of convergence (X^0 means the root node)
  if (UB(X^k) – LB(X^k)) / (UB(X^0) – LB(X^0)) < ε_f
    and XU(X^k) – XL(X^k) < ε_x
  then X^best ← X^k and X^k ← 0
}
```

Fig. 13 Algorithm for convergence test

한다. 입력노드 아래의 모든 leaf 노드들 중에 하한값이 가장 작은 노드를 찾아 이를 다음노드로 설정한다. (만약 입력노드가 이전 과정에서 삭제된 경우라면, 그 부모노드를 입력노드로 설정한다.)

○ 수렴여부의 판단 (Fig. 13)

입력노드의 수렴여부를 Fig. 13 과 같이 판단한다. 만약 수렴되었다면 입력노드를 최선노드로 설정하고 현재노드를 0(null)으로 설정한다.

4. 적용예제

4.1 테스트 함수

본 연구에서 제안하는 알고리즘의 성능을 평가하기 위하여 최적화 분야에서 테스트 함수로 흔히 사용되는 다음의 목적함수를 사용하였다.

$$f(\mathbf{x}) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{2}\sum_{i=1}^{2} x_i^2}} - e^{-\frac{1}{2}\sum_{i=1}^{2}\cos(2\pi x_i)} \quad (7)$$

$$f(\mathbf{x}) = \sum_{i=1}^{2}\frac{x_i^2}{4000} - \prod_{i=1}^{2}\cos(x_i/\sqrt{i}) + 1 \quad (8)$$

$$f(\mathbf{x}) = -\sum_{i=1}^{2}\sin(x_i)(\sin(i\,x_i^2/\pi))^{2m};\ m = 10 \quad (9)$$

$$f(\mathbf{x}) = 20 + \sum_{i=1}^{2}(x_i^2 - 10\cos(2\pi x_i)) \quad (10)$$

$$f(\mathbf{x}) = 418.9829 * 2 - \sum_{i=1}^{2}(x_i \sin\sqrt{|x_i|}) \quad (11)$$

$$f(\mathbf{x}) = (1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2 + 6x_1x_2+3x_2^2)) * (30+(2x_1-3x_2)^2(18-32x_1 + 12x_1^2+48x_2-36x_1x_2+27x_2^2)) \quad (12)$$

$$f(\mathbf{x}) = (x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1-\frac{1}{8\pi})\cos(x_1) + 10 \quad (13)$$

$$f(\mathbf{x}) = (4 - 2.1x_1^2 + \frac{1}{3}x_1^4)x_1^2 + x_1x_2 + (-4+4x_2^2)x_2^2 \quad (14)$$

$$f(\mathbf{x}) = \left(\sum_{i=1}^{5} i\cos((i+1)x_1+i)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2+i)\right) \quad (15)$$

각 테스트 함수의 함수명과 탐색공간은 Table 1 과 같으며, 목적함수의 복잡성은 Fig. 14 와 같다. 이들 함수들은 비선형성이 크고 국부 최소점이 많아 수치계산 측면에서 다루기 어려운 함수들이다.

식 (7) ~ (11)은 제안하는 최적화 알고리즘의 성능평가를 위한 예제로 사용되며, 식 (12) ~ (15)은 기존 타 방법과의 비교평가를 위한 예제로 사용된다. 그리고 식 (9)와 (14)는 사용자 입력 파라미터의 변화에 따른 수치결과를 분석하기 위한 예제로 활용된다.

Table 1 Test functions used as objective function for unconstrained global minimization

Label	Test function	Definition	Search domain
AC	Ackley	Eq. (7)	$[-15, 30]^2$
GR	Griewank	Eq. (8)	$[-600, 600]^2$
MI	Michalewics	Eq. (9)	$[0, \pi]^2$
RA	Rastrigin	Eq. (10)	$[-5.12, 5.12]^2$
SC	Schwefel	Eq. (11)	$[-500, 500]^2$
GP	Goldstein & Price	Eq. (12)	$[-2, 2]^2$
BR	Branin	Eq. (13)	$[-5, 10]$ x $[0, 15]$
C6	Six-hump Camel	Eq. (14)	$[-5, 5]^2$
SH	Shubert	Eq. (15)	$[-10, 10]^2$

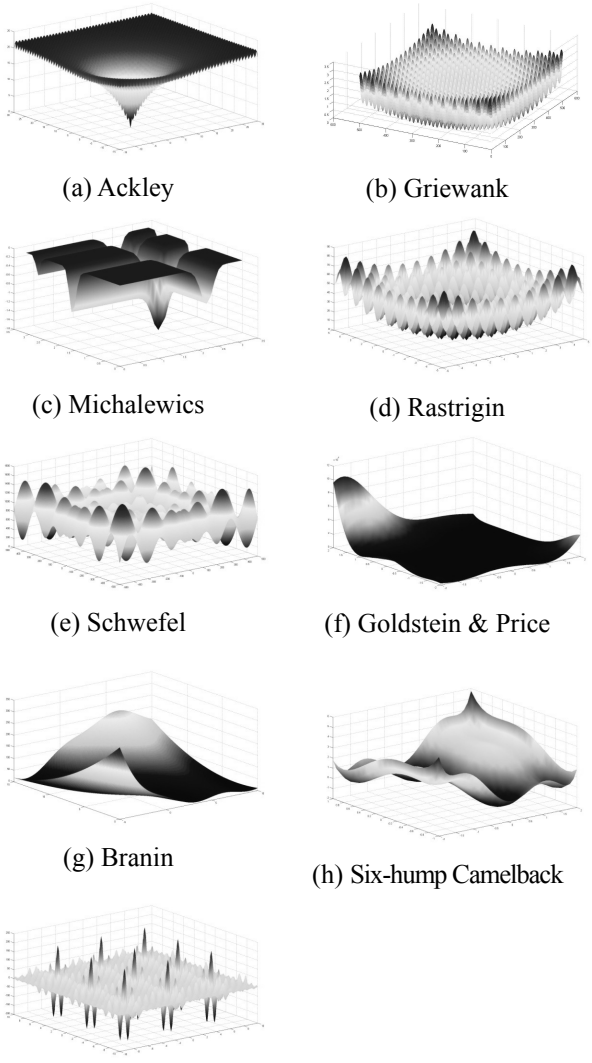


Fig. 14 Function graph of test functions

4.2 수치결과 및 알고리즘 성능분석

테스트 함수 식 (7) ~ (11)에 대해, 본 최적화 알

고리즘의 수치결과 및 이에 근거한 성능평가 결과는 Table 2 ~ 3 과 같다. 성능평가를 위해 본 연구에서 사용한 평가기준 및 수치결과는 다음과 같다.

○ 해의 정확성

해의 정확성은 아래와 같이 최적해의 정확도 및 그 위치에서의 함수값 정밀도에 의해 계산된다.

▶ 최적해의 정확도 (절대오차 개념)

Solution accuracy = $\| \mathbf{x}^* - \mathbf{x}^{opt} \|$

▶ 최적 함수값의 정밀도 (상대오차 개념)

Objective accuracy = $\| f(\mathbf{x}^*) - f(\mathbf{x}^{opt}) \| / \max(\| f(\mathbf{x}^{opt}) \|, \tau)$

여기서 $\mathbf{x}^{opt}$ 는 최적해(참값)이며, $\mathbf{x}^*$는 본 알고리즘에서 찾은 최적해(근사값)이다. ($\tau = 0.1$)

Table 2 의 수치결과를 살펴보면, 최적해의 정확도는 평균 0.0018 (탐색공간의 크기를 기준으로 볼 때 평균 0.0065%에 해당함)이며, 함수값의 정밀도는 0.11%이다.

○ 함수호출 회수

함수호출 회수란 본 알고리즘의 종료 때까지 호출된 목적함수의 계산(evaluation) 회수로서, 본 연구의 경우 최적해를 구하기 위해 사용된 총 샘플점의 개수와 같다. 오랜 시간을 요하는 CAE 시뮬레이션의 경우에 전체 성능을 결정하므로, 최적화 분야에서 매우 중요시 다룬다. 각 테스트 함수의 호출 회수는 Table 2 와 같이 평균 267 회이며, 샘플점의 개수가 많을수록, 수렴공차 값이 작을수록 증가하는 경향을 갖는다. (4.3 절 참고 바람)

○ 알고리즘 수행시간

전체 소요시간에서 테스트 함수 계산 시간을 뺀 시간으로 정의한다. 테스트 함수의 복잡성에 의해 수행시간이 좌우될 수 있으므로 순수하게 알고리즘만의 복잡성을 확인하기 위함이다. 수치결과는 Table 2 와 같이 평균 0.4 초이다.

○ 메모리 사용량

메모리 사용량은 아래와 같이 생성된 노드 수에 비례하며, 노드 1 개가 차지하는 메모리 할당량에 비례한다. 즉,

Memory size = (# nodes) x (node size)

(# nodes) = 1 + (# splitting nodes)^D x (# branch)

(node size) = 16 + 8 x D + 4 x (# splitting nodes)^D

여기서 (# node)는 노드개수, (#splitting node)는 공간분할 시 각 설계축의 균등분할 수, (# branch)는 알고리즘 종료 시까지 수행된 분할 회수(혹은 반복수행 회수)를 나타낸다. 참고로 16, 8, 4 는 본 연구 노드 자료구조에서 비롯된 것이다.

Table 2 Performance results produced by the proposed algorithm for test functions

Performance		AC	GR	MI	RA	SC
global SOL	$\mathbf{x}^{opt}$	0.0 0.0	0.0 0.0	2.2029 1.5708	0.0 0.0	420.9687 420.9687
	$f(\mathbf{x}^{opt})$	0.0	0.0	-1.8013	0.0	0.0
SOL found	$\mathbf{x}^*$	0.000127 0.000127	-0.00229 -0.00229	2.2043 1.5708	0.0 0.0	420.9668 420.9724
	$f(\mathbf{x}^*)$	0.000509	0.000004	-1.8013	0.0	0.000028
solution accuracy (a)		0.00018	0.0032	0.0014	0.0	0.0042
objective accuracy(%) (b)		0.509	0.004	0.0016	0.000	0.028
# function calls		70	666	105	305	190
run-time(ms) (c)		141	922	172	453	313
tolerance (d)		1.0e-3	1.0e-2	1.0e-2	1.0e-3	1.0e-2
#splitting nodes		3	4	3	3	3
# sample points		5	9	7	5	5

(a) Computed by $\| \mathbf{x}^* - \mathbf{x}^{opt} \|$

(b) Computed by $\|f(\mathbf{x}^*) - f(\mathbf{x}^{opt})\| / \max(\|f(\mathbf{x}^{opt})\|, 0.1)$

(c) Measured in milliseconds. It excludes the function evaluation times.

(d) Used for stopping criterion.

Table 3 Memory usage of the proposed method

	AC	GR	MI	RA	SC
# branch	14	74	15	61	38
# total nodes	127	1185	136	550	343
memory size, KB (without pruning)	8.4	111.1	9.0	36.5	22.8
# pruning nodes	108	1159	116	524	319
pruning ratio (%)	85.0	97.8	85.3	95.3	93.0
memory size, KB (with pruning)	1.3	2.4	1.3	1.7	1.6

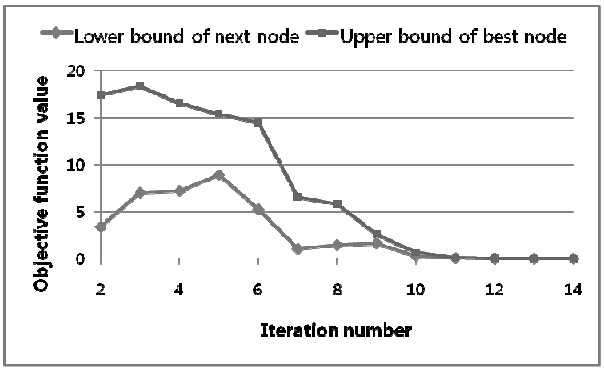
Table 3 과 같이 노드를 삭제한 경우에 평균 1.7 KB, 무삭제의 경우 37.6 KB 로서, 노드 삭제가 많이 일어났음을 알 수 있다. 이는 최선해와의 비교를 통해 많은 공간이 삭제되었음을 의미하며 본 연구의 최적해가 전역해임을 보여주는 것이다.

○ 알고리즘 수렴성

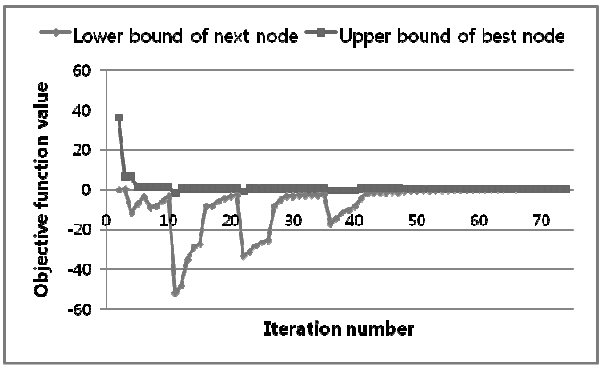
다음노드의 하한값과 최선노드의 상한값 변화 추이를 살펴봄으로써 알고리즘의 수렴성을 확인할 수 있다. 반복수행이 진행됨에 따라, Fig. 15 와 같이, 이 두 값은 그 차이가 0 에 수렴하며 하나의 최적값에 수렴함을 알 수 있다.

4.3 사용자 파라미터 영향도 분석

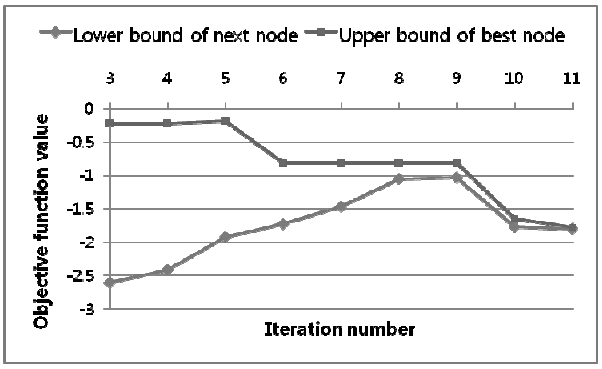
본 알고리즘 입력자료로서 목적함수 및 탐색공간 외에 아래와 같은 사용자 파라미터가 있다. 이 파라미터들은 알고리즘 성능과 직결되는데, 해의 정밀도 및 계산시간의 증감에 영향을 준다.



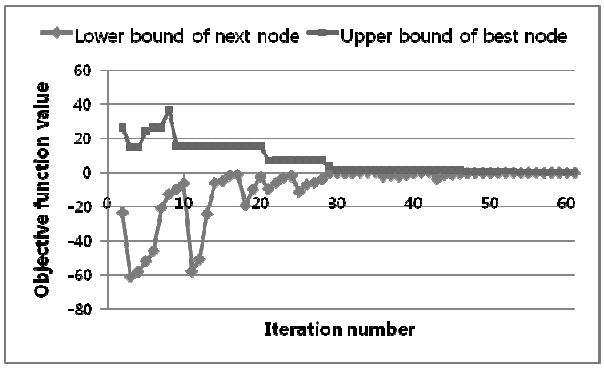
(a) Ackley function



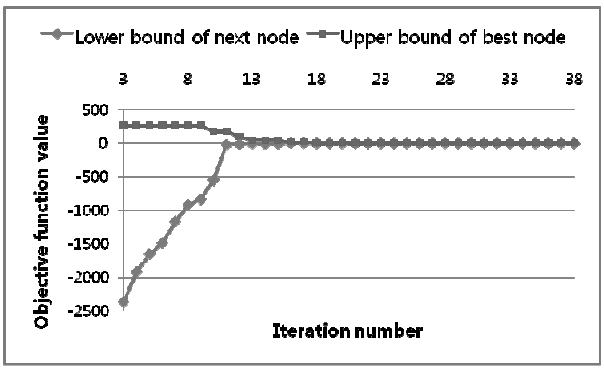
(b) Griewank function



(c) Michalewics function



(d) Rastrigin function



(e) Schwefel function

Fig. 15 Algorithm convergence of test functions

○ 수렴여부 판단 공차 (Fig. 16)

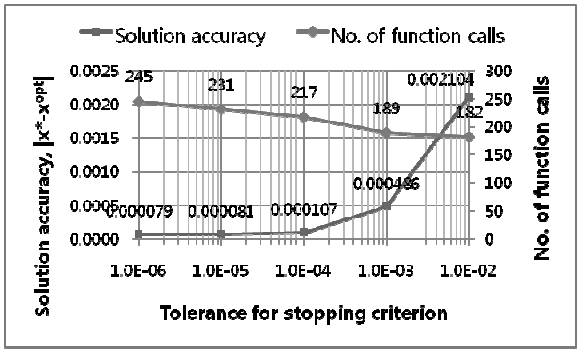
수렴여부를 판단하기 위한 공차로서 Fig. 13 과 같이 ε_f 는 목적함수 상하한값 차이에 관한 최대허용공차이며, ε_x 는 탐색공간의 크기에 관한 최대허용공차이다. 본 테스트 문제의 경우 이 두 값을 동일시 하였다. Fig. 16 과 같이 공차값이 커질수록 최적해의 정확도는 떨어지나, 함수호출 회수는 적어진다.

○ 샘플점의 개수 (Fig. 17)

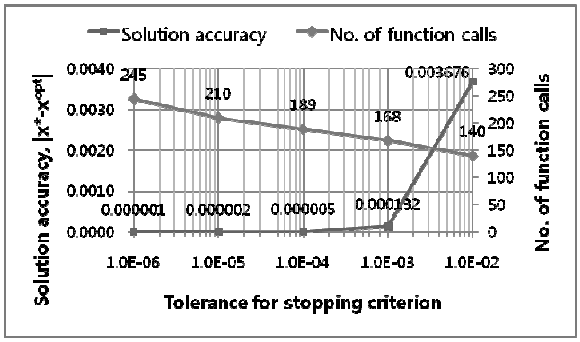
목적함수 근사모델 생성을 위한 샘플점의 개수(N)로서 LHS 방법에 의해 생성되는 추출점의 개수이다. Fig. 17 과 같이 샘플점의 개수가 커질수록 목적함수의 정밀도는 어떤 한계까지만 높아지나, 함수호출 회수는 꾸준히 증가한다.

○ 분할노드 개수 (Fig. 18)

설계변수 축을 따라 균등 분할되는 부공간의 개수(M)로서 각 축은 M 등분되므로 설계공간이 D 차원일 때 분할 부공간의 총 개수는 M^D 이다. Fig. 18 과 같이 분할노드 개수가 목적함수 정밀도 및 함수호출 회수의 증감에 아무 영향이 없음을 확인할 수 있다.

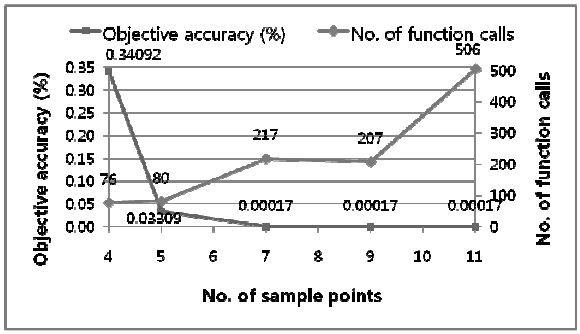


(a) Michalewics function

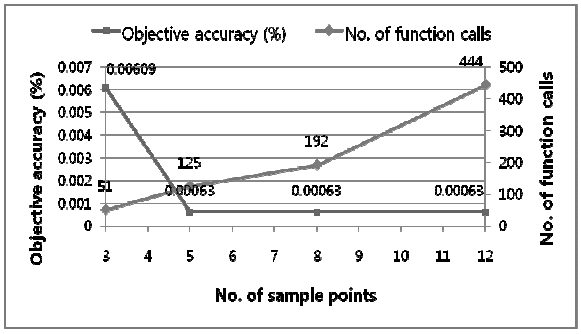


(b) Six-hump Camelback function

Fig. 16 Performance test of the convergence tolerance



(a) Michalewics function



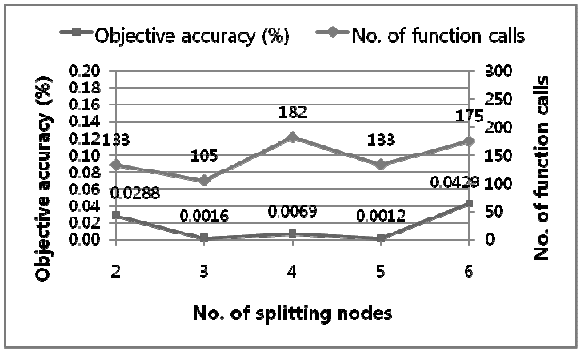
(b) Six-hump Camelback function

Fig. 17 Performance test of the number of sample points

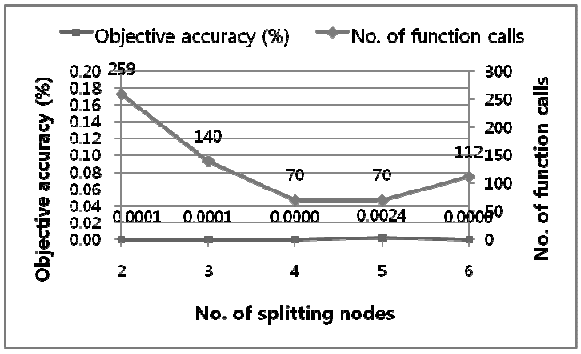
Table 4 Number of function calls of various methods compared to the proposed approach

Method	GP	BR	C6	SH
Bremmerman		250		
Mod. Bremmerman	300	160		
Zilinskas		5129		
Torn	2499	1558		
Gomulka-V.M.	1495	1318		
Price	2500	1800		
Fagiuoli	158	1600		
De Biase-Frontini	378	587		
Mockus	362	189		
Belisle et al.	4728	1846		
Boender et al.	398	235		
Snyman-Fatti	474		178	
Kostrowicki-Piela	120		120	
Yao			1132	
Pettunen	82	97	54	197
Stuckman	113	109	96	
Jones et al.	191	195	285	2967
Storn-Price	1018	1190	416	1371
MCS(1)	94	51	37	566
MCS(2)	194	57	44	48
Our approach	138	127*	88**	135

(1) Perturbed box bounds (see ref [11] for details)
(2) Unconstrained problem (see ref [11] for details)
*Average evaluations over 3 global minima (cf: 127= (168+78+136)/3)
**Average evaluations over 2 global minima (cf: 88 = (125+51)/2)



(a) Michalewics function



(b) Six-hump Camelback function

Fig. 18 Performance test of the number of splitting nodes

Table 5 Performance results of multivariable test function, Eq. (16)

No. of design variables	2	3	4	5	6	7	8	9
$\mathbf{x}^*$	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
$f(\mathbf{x}^*)$	0.000 000	0.000 000	0.000 056	0.000 071	0.000 000	0.000 209	0.003 599	0.017 626
$\|\| \mathbf{x}^* - \mathbf{x}^{opt} \|\|$	0.000	0.000	0.006	0.007	0.000	0.014	0.043	0.126
$\|\|f(\mathbf{x}^*) - f(\mathbf{x}^{opt})\|\|$	0.0E-6	0.0E-6	5.6E-5	7.1E-5	0.0E-6	2.1E-4	3.6E-3	1.8E-2
# function calls	33	48	117	168	300	576	1870	2880
run-time(sec)	0.016	0.078	0.203	1.079	6.765	44.92	490.4	2752.
# sample points	3	4	9	12	20	36	110	160

(a) (0.000000, 0.000000)
(b) (0.000000, 0.000000, 0.000000)
(c) (0.005758, -0.000339, -0.002710, 0.000677)
(d) (0.004742, -0.004742, -0.000339, -0.000339, -0.001016)
(e) (0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000)
(f) (0.013887, -0.002710, -0.000339, 0.000000, 0.000339, -0.000339, -0.000677)
(g) (-0.008129, -0.041322, -0.005081, -0.002710, 0.000000, -0.002710, 0.000000, 0.000000)
(h) (-0.123626, 0.014225, 0.006097, 0.000000, -0.008129, -0.013887, 0.000000, 0.000000, 0.006097)

Table 6 Performance results of multivariable test function, Eq. (17)

No. of design variables	2	3	4	5	6	7	8	9
$\mathbf{x}^*$	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
$f(\mathbf{x}^*)$	0.000 000	0.000 000	0.000 073	0.004 676	0.013 684	0.033 523	0.044 614	0.035 346
$\|\| \mathbf{x}^* - \mathbf{x}^{opt} \|\|$	0.000	0.000	0.008	0.068	0.113	0.180	0.211	0.188
$\|\|f(\mathbf{x}^*) - f(\mathbf{x}^{opt})\|\|$	0.0E-6	0.0E-6	7.3E-5	4.7E-3	1.4E-2	3.4E-2	4.5E-2	3.5E-2
# function calls	80	270	800	2158	4480	7200	8960	10540
run-time(sec)	0.047	0.141	2.031	32.63	383.0	3402.	4278.	13205
# sample points	10	30	80	166	320	480	560	620

(a) (0.000153, 0.000153)
(b) (0.000153, 0.000153, 0.000153)
(c) (0.003815, 0.003815, 0.003815, -0.005341)
(d) (-0.039074, -0.003124, 0.011311, 0.042980, -0.033848)
(e) (-0.039661, 0.043171, -0.010998, 0.081203, -0.041142, -0.031742)
(f) (0.020519, 0.093754, -0.063740, -0.127056, 0.041630, 0.020275, 0.031872)
(g) (0.123050, 0.014920, -0.032350, 0.037482, -0.031589, 0.102908, 0.037871, -0.117305)
(h) (-0.024569, -0.019987, 0.036230, -0.063807, 0.020956, -0.144533, 0.018465, 0.083029, 0.019869)

4.4 기존 방법과의 성능비교

함수호출 회수 측면에서 기존 방법과 비교한 결과가 Table 4 와 같다. 함수호출 회수에 의한 비교가 알고리즘의 질을 평가하기 위한 유일한 방법은 아니지만 목적함수 값을 추출하는데 상당한 시간과 비용을 요구하는 산업현장의 경우라면 의미 있다고 볼 수 있다. 한편, 동일한 문제를 여러 방법으로 그 해를 구할 때 입력조건 및 수렴조건 등에 따라 함수호출 회수가 달라진다. 비록 Table 4 의 모든 방법들이 모두 동일한 조건은 아니지만 간접적으로 알고리즘의 성능을 평가해 볼 수 있으리라 판단된다.

이런 측면에서 Table 4 의 결과를 살펴보면, 본 연구 알고리즘이 MCS 방법[11]과 비교 시 대등하며, 이 밖의 방법들과는 상대적으로 우위에 있음을 확인할 수 있다.

4.5 다변수 최적화 예제

알고리즘 성능평가를 위해 사용된 4.1 절의 테스트 함수는 모두 2 변수 문제이다. 설계변수 증가에 따른 본 알고리즘의 전역해 탐색 성능을 확인하기 위해 식 (16)의 Sum Squares 함수와 식 (17)의 Zakharov 함수를 D-차원 설계공간에서의 다변수 테스트 함수로 사용하였다. 식 (16)의 경우에 설정된 공차가 0.001, 분할노드 개수가 3 이며, 최적해 참값은 $\mathbf{x}^{opt} = (0,\ldots,0)$, $f(\mathbf{x}^{opt}) = 0$ 이고, 탐색공간은 $[-10, 10]^D$ 이다. 마찬가지로 식 (17)은 각각 0.00001, 4, $\mathbf{x}^{opt} = (0,\ldots,0)$, $f(\mathbf{x}^{opt}) = 0$, $[-5, 10]^D$ 이다.

$$f(x_1,\cdots,x_D) = \sum_{i=1}^{D} i \cdot x_i^2 \tag{16}$$

$$f(x_1,\cdots,x_D) = \sum_{i=1}^{D} x_i^2 + \left(\sum_{i=1}^{D} 0.5 \cdot i \cdot x_i\right)^2 + \left(\sum_{i=1}^{D} 0.5 \cdot i \cdot x_i\right)^4 \tag{17}$$

Table 5 와 6 은 이들 테스트 함수의 수치결과 및 이에 근거한 성능평가 결과를 보여주고 있다. 설계변수 개수가 증가함에 따라, 수행시간 및 함수호출 회수는 기하급수적으로 증가하며, 해의 정확성은 대략 선형적으로 떨어짐을 확인할 수 있다.

설계공간 차원의 증가는 탐색공간 크기의 증가를 의미하며, 확장된 넓은 공간에 대해 비스플라인 근사모델에 의한 특정 수준의 근사 정밀도를 확보하기 위해선 더 많은 함수호출이 필요할 것이다. 본 예제의 경우, 설계변수 증가에 따라 탐색공간의 크기가 기하급수적으로 증가하는데 예상대로 그만큼의 함수호출이 수행되었다. 그러나, 기대했던 일정 수준의 해의 정확성은 나타나지 않았다.

5. 결 론

본 연구에서 제안하는 branch-and-bound 기반의 전역최적화 알고리즘은 일반적인 비선형 문제에 관한 근

사해를 제공한다. 그리고 본 연구의 특징인 LHS 방법에 기반을 둔 비스플라인 근사모델 방식은 공간분할 및 상하한값 계산의 수치적 복잡성을 해결해 주며, 동시에 요구되는 정밀도의 전역해를 가능하게 해준다. 적용 예제를 통해 살펴본 알고리즘의 성능평가 결과와 기존 방법들과의 비교결과 자료는 이러한 본 연구 알고리즘의 우수성을 입증하는 것이라 볼 수 있다.

본 연구 알고리즘의 주요 특징을 살펴보면 다음과 같이 요약된다. 1) 고차원의 설계공간 안에서 전역 최적화가 가능하다. 2) 기본 방식이 branch-and-bound 방식이므로 수렴성이 보장되는 완전(complete) 알고리즘[12]이다. 3) 직관에 의존한 휴리스틱(heuristics)을 사용하지 않는다. 4) 목적함수의 1 차 이상 미분값을 요구하지 않으며, 국부 탐색 알고리즘 없이도 적절한 전역해를 제공한다. 5) 2 변수의 경우, 알고리즘 수행시간이 짧고, 메모리 사용량이 작으며, 함수호출 회수 역시 비교적 적다. 6) 설계변수 개수가 증가함에 따라 함수호출 회수는 기하급수적으로 증가한다.

추후 연구과제로서 1) 현재의 알고리즘은 정수형 변수를 지원치 못하며, 구속조건에 관한 처리가 없다. 2) 목적함수 계산시 잡음이 없다고 가정한다. 3) 쾌속의 근사 최적해 유도를 위해 목적함수에 영향력이 미비한 설계변수를 탐색 삭제한다. (비스플라인 모델의 convex hull 성질을 활용할 경우 쉽게 해결되리라 판단한다.) 4) 다중 목적함수의 표현 및 해법, 마지막으로 5) 균등 분할이 아닌 목적함수에 의존한 비균등 분할 등의 추가 연구가 필요하다.

참고문헌

(1) Pinter, J.D., 1996, *Global Optimization in Action*, Kluwer, Dordrecht.

(2) Floudas, C.A., 2000, *Deterministic Global Optimization: Theory, Methods and Applications*, Kluwer Academic Publishers.

(3) Neumaier, A., 2004, “Complete Search in Continuous Global Optimization and Constraint Satisfaction,” in: *Acta Numerica*, Cambridge Univ. Press, pp.1~99.

(4) Floudas, C.A., 2007, “Overview of aBB-based Approaches In Deterministic Global Optimization,” *Workshop on Global Optimization*, Imperial College London, 15-17 Dec.

(5) Al-Khayyal, F.A. and Sherali, H.D., 2000, “On finitely terminating branch-and-bound algorithms for some global optimization problems,” *SIAM J. Optimization*, Vol.10, pp.1049-1057.

(6) Audet, C., Hansen, P., Jaumard, B., and Savard, G., 2000, “A Branch and Cut Algorithm for Nonconvex Quadratically Constrained Quadratic Programming,” *Math. Programming*, Vol.87, pp.131-152.

(7) Park, S, 2009, “A Rational B-spline Hypervolume for Multidimensional Multivariate Modeling,” *Journal of Mechanical Science and Technology*, Vol.23, pp.1967~1981.

(8) Piegl, L. and Tiller, W., 1995, *The NURBS Book*, Springer-Verlag.

(9) De Boor, C., 1978, *A Practical Guide to Splines*, New York, Springer-Verlag.

(10) Stein, M., 1987, “Large Sample Properties of Simulations Using Latin Hypercube Sampling,” *Technometrics*, Vol.29, pp.143~151.

(11) Huyer, W. and Neumaier, A., 1999, "Global Optimization by Multilevel Coordinate Search," *Journal of Global Optimization*, Vol. 14, pp.331~355.

(12) Neumaier, A., 2004, “Complete Search in Continuous Global Optimization and Constraint Satisfaction,” pp.1-99 in: *Acta Numerica*, Cambridge Univ. Press.