한국지능시스템학회 논문지 2011, Vol. 21, No. 3, pp. 401-406
DOI : 10.5391/JKIIS.2011.21.3.401

Harmony Search 알고리즘의 수렴성 개선에 관한 연구

Study on Improvement of Convergence in Harmony Search Algorithms

이상경 · 고광은 · 심귀보*
SangKyung Lee, Kwang-Enu Ko and Kwee-Bo Sim*

중앙대학교 전자전기공학부

요 약

복잡해진 최적화문제를 전통적인 방법보다 효율적으로 해결하기위해 유전알고리즘이나 개미군집화, 하모니서치알고리즘과 같은 다양한 메타휴리스틱이 개발되었다. 그 중에서 하모니 서치알고리즘이 다른 메타휴리스틱알고리즘보다 좋은 결과를 보이고 있다. 하모니 서치 알고리즘은 음악을 작곡할 때 아름다운 소리를 내는 하모니를 찾는 과정을 모방했다. 성능은 하모니 메모리에서 선택하는 비율인 HMCR값과 하모니 메모리에서 선택된 값의 조정 비율을 결정하는 PAR값에 따라 달라지는 것으로 알려져 있다. 다르게 말하면 두 변수의 기반이 되는 하모니 메모리의 사용방법의 문제로 볼 수 있다. 본 논문은 설정한 기간 동안 더 좋은 최적해를 찾지 못할 경우 하모니 메모리의 일부를 좋은 하모니로 구성되게 수정하는 방법을 제안했다. 테스트 함수를 이용한 검증 실험결과에서 하모니 메모리를 수정할 경우 정확도 변화가 적어 신뢰성 있는 정확도를 보였으며, Iteration이 짧더라도 최적값에 근접한 값을 찾았다.

키워드 : 하모니서치알고리즘, 최적화 연산, 메타휴리스틱, 전역 최적해, 지역 최적해

Abstract

In order to solve a complex optimization problem more efficiently than traditional approaches, various meta-heuristic algorithms such as genetic algorithm, ant-colony algorithm, and harmony search algorithm have been extensively researched. Compared with other meta-heuristic algorithm, harmony search algorithm shows a better result to resolve the complex optimization issues. Harmony search algorithm is inspired by the improvision process of musician for most suitable harmony. In general, the performance of harmony search algorithm is determined by the value of harmony memory considering rate, and pitch adjust rate. In this paper, modified harmony search algorithm is proposed in order to derive best harmony. If the optimal solution of a specific problem can not be found for a certain period of time, a part of original harmony memory is updated as the selected suitable harmonies. Experimental results using test function demonstrate that the updated harmony memory can induce the approximation of reliable optimal solution in the short iteration, because of a few change of fitness.

Key Words : Harmony search algorithm, optimization, Meta Heuristic, global optimum, local minimum

1. 서 론

최적화 문제를 해결하는 방법으로 자연을 모방한 메타휴리스틱(Meta-Heuristic)알고리즘이 많이 사용되고 있다. 비용을 절감과 효율성을 만족하는 값을 찾아내는 최적화 문제는 컴퓨터 연산량 증가와 함께 많은 발전과 다양한 분야에 활용 되고 있다. 하지만 문제의 복잡도가 증가하면서 기존의 해결방법은 지역 최적 해에 머무르거나 계산해야할 양이 기하급수적으로 증가하는 등의 한계에 부딪치게 되었다. 예를 들어 변수가 x가 n개 있을 때 검색할 해가 각각 k개씩 주어지면 kn개가 된다. 해결하기 위해 유전자 알고리즘(Genetic algorithm, GA)[1], 개미군집화 알고리즘(Ant Colony Optimization, ACO)[2], 입자군집 최적화(Particle Swarm Optimization, PSO)[3], 하모니 서치 알고리즘(Harmony Search algorithm, HS)[4]등이 제안되어 좋은 성능을 나타내고 있다.

이와 같은 일반적인 메타휴리스틱알고리즘은 한정된 시간 안에 주어진 문제를 모델링한 목적함수를 만족하는 최적해를 탐색하는 것을 목적으로 한다[5]. 하지만 랜덤함수를 기반으로 하기 때문에 항상 가장 좋은 최적 해를 찾는다는 보장을 할 수 없다. 메타휴리스틱 알고리즘은 초기 해집합을 랜덤함수를 이용해 주어진 범위 내에서 랜덤하게 초기화 후 좋은 값을 선택해 업데이트하는 과정을 거친다. 선택된 값을 이용하여 각각의 알고리즘이 제공하는 연산자에 따라 해를 생성한다. 생성된 해의 적합도 함수, 평가함수에 적용

접수일자 : 2011년 3월 19일
완료일자 : 2011년 4월 29일
* 교신 저자
본 논문은 본 학회 2011년도 춘계학술대회에서 선정된 우수논문입니다.
본 논문은 한국연구재단 중견연구자지원사업(No. 2010-0029226)에서 지원하여 연구하였습니다. 연구비 지원에 감사드립니다.

하여 가장 나쁜 해를 버리고, 업데이트하는 과정을 통해 최적의 해를 찾는다.

기존의 메타휴리스틱 알고리즘은 탐색 구간을 설정 후 초기 해 집합을 랜덤하게 구성하고 연산자를 랜덤선택한다. 초기 최적해를 모르기 때문에 초기 해집합을 랜덤구성 후 좋은 값을 찾으면 업데이트해간다. 연산자 역시 가장 좋은 최적값을 찾는 기준 설정이 힘들기 때문에 랜덤선택을 한다. 하지만 랜덤선태과 랜덤 초기해구성은 수렴속도와 정확도에 좋지 않은 영향을 준다. 예를 들어, 메모리를 개선해 가는 GA의 유전자 풀의 경우 하나의 해로 업데이트하는 과정이 랜덤 함수에 의존적이므로 수렴시간이 오래 걸리거나 지역해로 수렴할 가능성이 있다. 또한, GA의 빌딩블럭 문제는 유전자 풀에 좋은 해가 잘 관리 되지 못하게 한다[6].

본 논문에서는 이와 같은 랜덤 선택과 초기 해집합 랜덤 구성의 문제 해결을 위하여 HS를 기반으로 하는 새로운 메타휴리스틱 알고리즘에 대하여 제안하고자 한다. 최적 값인 harmony생성을 위한 연산자 선택과 초기 해 집합 랜덤구성은 해집합인 하모니메모리(Harmony Memory, HM)를 기반으로 하기 때문에 HM의 구성이 중요하다. 따라서 나쁜 값들로 HM이 구성되어 발견 가능성 낮다면 HM의 수정이 필요하다. 하지만 발견 가능성은 최적 값을 모르기 때문에 직접 HM 구성을 판단하기는 힘들다. 하지만 계속 더 좋은 harmony를 찾지 못한다면 구성이 나쁘다고 판단이 가능하다. 작곡가의 인내심의 한계를 모방해 계속 좋은 하모니를 찾지 못할 때 허용 가능한 기간을 설정하고, 이 기간을 초과하게 되면 HM을 수정하는 방법을 HS에 적용했다[7].

논문의 구성은 2장에서 기본이 되는 알고리즘인 HS에 대해 설명 후 3장에서 제안하는 내용에 대해 다루었다. 4장에서는 테스트 함수를 이용해 제안하는 알고리즘의 성능을 실험 통해 검증했다. 마지막으로 5장에서 결론과 향후 연구로 마무리 했다.

2. Harmony search algorithm 소개

연주자가 청중에게 감동을 주기위해 현장에서 좋은 harmony를 찾아 연주하는 즉흥 연주를 모방한 알고리즘이다. 몇 명의 연주자가 악보 없이 연주를 하기 때문에 서로 조화를 이루는 규칙이 존재한다. HS는 이 규칙을 모방해 아름다운 harmony를 찾듯이 해를 검색한다[1][8].

최적화연산은 연주자가 자신의 악기를 이용해 최고의 harmony를 찾아 청중에게 최고의 감동을 주는 것과 같다. 음을 찾을 범위는 악기에 제한되므로 찾을 변수의 범위로 볼 수 있다. 악기에서 하나의 음을 선택해 소리를 내는 연주자는 설계변수 또는 결정변수가 된다. 이때 나는 소리들이 모여 하나의 harmony를 이루므로 해집합 HM이 된다. 청중에게 큰 감동을 준 harmony가 오래 기억되듯이 해 벡터(Solution vector)도 목적함수에 대입했을 때 좋은 값이면 HM에 저장된다. 마지막으로 경험 많은 작곡가가 좋은 화음을 잘 만들 듯이 연산을 반복할수록 더 좋은 해를 생성하면서 최적 값에 접근한다.

나쁜 harmony는 버리고 새로 찾은 좋은 harmony들을 악보에 기록하듯 HM에 기록한다. 작곡가는 악보에 기록한 후 연주나 새로 작곡할 때 참고해 다시 연주하거나 편곡을 한다. HS 역시 악보와 같은 역할을 하는 HM을 가지고 있어 좋은 해를 재사용 할 수 있으며, 다음과 같이 행렬로 표현할 수 있다.

$$HM=\left[\begin{array}{cccc|c} x_{(1,1)} & x_{(1,2)} & \cdots & x_{(1,n)} & f(x_1) \\ x_{(2,1)} & x_{(2,2)} & \cdots & x_{(2,n)} & f(x_2) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{(hms,1)} & x_{(hms,2)} & \cdots & x_{(hms,n)} & f(x_{hms}) \end{array}\right] \quad (1)$$

여기서 왼쪽 열에 있는 x는 오른쪽 목적 함수인 $f(x_i)$의 변수를 의미한다. 이때 n은 변수의 개수이며, hms (Harmony Memory Size)는 HM에 최대 저장 가능한 harmony의 개수를 나타낸다. 초기 해는 주어진 범위 내에서 랜덤하게 생성해 구성한다.

다음에 나오는 연산자는 좋은 harmony로 HM을 개선하기위해 harmony를 생성하는 방법이다. 기본적으로 사용하는 연산자는 랜덤선택과 HM을 이용한 방법으로 나뉜다.

2.1 랜덤선택(Random Selection)

HM을 초기화하는 것처럼 주어진 음의 범위 내에서 랜덤하게 해를 생성하는 방법이다. 모든 영역을 검사하는 것은 비효율적이기 때문에 랜덤하게 검색해서 찾은 좋은 해를 다음에 나오는 연산자를 이용해 자세히 찾는다. 지역 최적해에 빠졌을 때도 다른 영역으로 유도한다. 하지만, 임의로 하나의 해를 선택하기 때문에 신뢰도가 낮다. 따라서 너무 많은 선택 확률을 주어 해를 찾을 가능성과 효율성이 나빠지지 않도록 주의해야한다.

선택될 확률은 HM을 사용할 확률(Harmony Memory Considering Rate, HMCR)을 뺀 나머지 확률이므로 (1-HMCR)이 된다.

2.2 Harmony memory를 수정하는 방법

좋은 해가 저장되어 있는 HM의 값을 사용해 해를 찾는 방법이다. 이는 값을 가져와 그대로 사용 방법과 조정 후 사용하는 방법으로 나뉜다.

2.2.1 기억회상(Memory Consideration)

악보에 기록된 음을 하나 선택해 그대로 가져다 사용하는 기억회상이다. 악보에 기록된 좋은 음을 조합해 더 좋은 화음을 만드는 원리이다. HM에서 값을 가져와 변수 값으로 사용한다. 선택될 확률은 앞서 이야기한 HMCR에 의해 결정 되며 0부터 1사이 값을 갖는다. 많이 선택되는 값은 랜덤선택 문제 때문에 보통 0.7에서 0.95값을 사용한다.

2.2.2 피치 조정(Pitch Adjustment)

악보에서 선택된 음을 위나 아래로 조정해 사용하는 방법이다. 기억 회상법을 통해 선택된 음이 조금 더 나은 소리를 낼 수 있도록 피치를 조정해 보정하는 역할을 한다. 이는 HM에서 가져온 값을 위나 아래로 △pa만큼 이동 시키는 역할 한다. 이동 범위는 이산 문제 인 경우 이산 간격으로 설정하면 된다. 하지만 연속적인 변수의 경우 너무 세밀한 값을 설정하면 검색할 범위가 많아진다. 반대로 너무 크게 하면 사이에 존재하는 값은 반영하지 못 하므로 가변 가능하도록 설정 해 사용한다.

가장 좋은 값 주변도 검색할 수 있으므로 더 좋은 해의 발견 가능성을 높여준다. 랜덤함수는 해가 높은 지역을 구분할 수 있는 기준 없이 해를 생성하고 결정한다. 하지만 피치조정은 현재 가장 좋은 harmony의 집합인 HM을 기준으로 한다. 이 값들의 일부(또는 전부)는 최적 해에 가까운 값일 가능성이 높다. 따라서 가능성 높은 해의 근처도 탐색할 수 있어 발견 가능도 같이 커진다. 그리고 검색 범위도

축소되므로 효율성도 좋아진다.

하나의 HM뿐이지만 피치 조정된 2개의 가상 HM이 생성된다. HM 값은 하나이지만 피치 조정된 2개의 값이 더 존재해 가상의 HM이 2개 더 존재한다. 즉, 실재로는 3개의 HM을 사용하는 것이 되어 적은 공간으로도 가능성 높은 해집합을 많이 확보 할 수 있다. 이러한 메모리 사용은 다른 알고리즘에 없는 HS만이 갖는 장점이다.

피치조정이 선택될 확률은 PAR(Pitch Adjusting Rate)에 의해 결정된다. 기억 회상 중 일부가 피치 조정되므로 (1-PAR)로 표현되며 0부터 1까지 갖는다. 비율이 높으면 랜덤선택과 비슷하게 의미 없는 해를 많이 생성하므로 보통 0.01에서 0.3사이 값으로 설정한다. 위나 아래로 이동할 확률은 특별한 정의 없이 0.5로 고정해 사용한다.

3. 작곡가의 인내심을 모방한 Harmony search algorithm 제안

HS의 성능은 HMCR과 PAR값에 영향을 받는 것으로 알려져 있다. (GA나 ACO, PSO와 같은 대부분의 메타휴리스틱 알고리즘들은 초기 해집합의 구성이 나쁘더라도 좋은 해를 계속 업데이트해가면 최종적으로 가장 좋은 해로 수렴한다는 가정을 바탕으로 한다. 다시 말하면 해를 생성하는 중요한 매개변수(Parameter)인 HMCR과 PAR의 설정이 중요하다고 할 수 있다. 따라서 자동으로 매개변수를 설정을 자동으로 하거나 PSO를 적용(HS-PSO)해 적합도가 좋은 해의 선택확률을 높이는 등 관련 연구들이 많이 진행되고 있다[9][10]. 하지만 초기해가 나쁘면 수렴하는 시간도 길어져 주어진 시간 내에 최적 해를 찾지 못 한다.

여러 가지 좋은 작곡 기술이 있어도 작곡가의 능력에 따라 악보에서 찾을 수 있는 곡은 정해져 있다. 악보도 음들의 집합이므로 작곡 가능한 경우의 수가 다르다. 경우에 따라 범위가 좁거나 너무 넓어 작곡하기 힘든 악보들이 발생한다. 이러한 악보들은 그림1과 같이 작곡가의 판단에 의해 좋은 harmony를 찾을 수 없다고 결정되면 악보를 수정한다. 수정하는 방법으로는 좋은 harmony만 선택하거나 새로운 작곡과 같은 방법들을 사용한다.

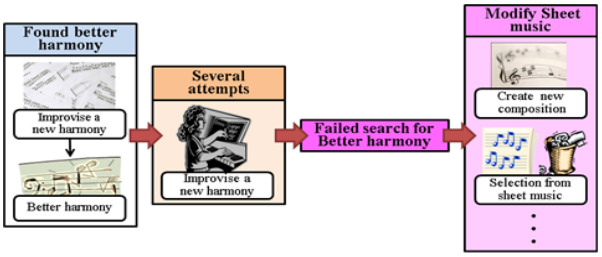

그림 1. 작곡 과정
Fig. 1. Composition Process

작곡가 인내심의 한계를 모방해 악보를 악보에 해당하는 HM을 수정하는 방법을 제안했다. 가장 좋은 해를 알 수 없기 때문에 기준이나 관련 변수 설정이 어렵다. 경험이 부족한 작곡가와 같으므로 HM 수정하는 순간을 바로 판단하기 힘들다. 따라서 판단하기 힘든 작곡가는 계속 하게 되다. 하지만 인내심의 한계가 있기 때문에 더 이상 좋은 곡을 작곡을 하지 못하면 발견 가능성이 낮다고 판단하게 된다. 그리고 악보를 수정을 통해 같은 시간에 더 많은 작곡을 할 수 있다. 이를 모방해 인내심과 같은 제한된 시도 횟수를 설정했다. 실행 중 제한된 시도횟수를 초과했지만 더 좋은 해를 발견하지 못하면 HM을 수정하도록 했다.

3.1 알고리즘 순서도

제한된 시도 횟수를 설정위한 "Attempt"와 "Endurance" 개념을 HS에 추가했다. "Attempt" 더 좋은 harmony를 발견하지 못했지만 찾기 위해 노력 하는 시도 횟수를 의미한다. 적합도가 나쁜 무의미한 harmony를 생성할 때 증가하며, 더 좋은 harmony를 발견하면 0으로 초기화 된다. "Endurance"는 계속 적합도가 나쁜 해가 생성될 경우 최대로 허용할 수 harmony의 개수이다. 열심히 작곡했지만 더 좋은 harmony를 발견하지 못 했을 때 작곡가가 견딜 수 인내심의 한계와 같다. 계속 새로운 harmony를 찾지 못했을 때 harmony 생성을 잠시 멈추고 HM 수정과 "Attempt 값을 초기화할 수 있도록 한다.

예를 들어 적합도가 나쁜 harmony의 최대 허용 개수가 200개 일 때 좋은 harmony를 iteration이 1,000번째에서 찾은 후 1,201번째까지 하나도 못 찾았다면, Endurance는 200 되고 Attempt는 201이 된다. 이때 Attempt값이 201이 되어 Endurance를 초과해 HM은 수정되고, Attempt값은 0으로 초기화 된다

아래 설명할 제안하는 알고리즘의 동작은 그림 2와 같다. harmony를 생성하고 HM에 업데이트하는 기본적인 방법은 같다. 하지만 최적해를 계속 찾지 못할 경우 HM을 수정하는 부분이 추가 되었다.

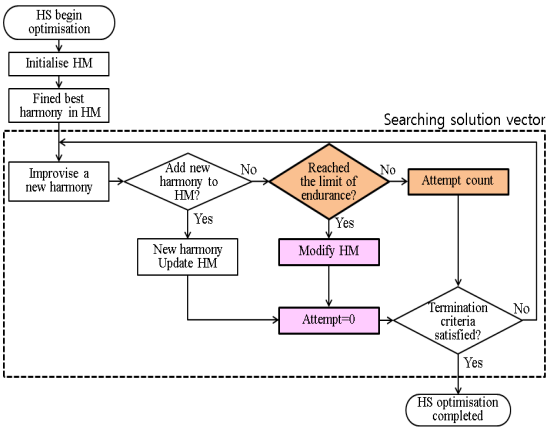

그림 2. 제안하는 알고리즘 순서도
Fig. 2. Suggest algorithm flowchart

1) HM을 랜덤함수를 이용해 초기 harmony집합을 생성함
2) 초기해 집합 중 가장 좋은 harmony를 처음 찾을 harmony의 기준으로 설정함
3) harmony를 생성함
4) 더 좋은 harmony를 찾으면 HM에 업데이트하고 "attempt"값을 '0'으로 초기화한 후 7)으로 이동
5) 더 좋은 harmony를 못 찾을 경우 "Attempt"값이 최대 허용 개수인 "Endurance"를 초과하면, HM을 수정하고 "Attempt"값을 '0'으로 초기화한 후 7)으로 이동 함.
6) 초과 하지 않은 경우 "Attemtp" 값을 증가 시키고 다음 항목으로 이동함.
7) 종료 조건을 만족할 때 까지 3)부터 6)을 반복함.

3.2 HM을 수정하는 방법 제안

초기화되는 HM의 개수는 유지 비율(Harmony Memory Preservation Rate, HMPR)에 의해 결정된다. 수정하지 않고 유지 하게 될 HM내의 harmony 개수를 의미하며, 0부터 최대 개수인 hms까지 설정할 수 있다. 예를 들어 HM의 hms=100일 때 HMPR이 70으로 설정되면 70개는 그대로 유지되고 나머지 30개는 앞서 소개한 수정 방법에 의해 재설정 된다. harmony 유지 비율(Harmony Memory, HMPR)을 두어 일부만 수정할 수 있도록 했다. 최적해를 찾는 중간에 HM 전체를 모두 수정하는 것은 찾은 좋은 harmony를 버리고 처음부터 시작하는 것 비효율적이다.

첫 번째로 새로 작곡하는 것과 같이 가장 좋은 harmony의 피치 조정해 HM을 수정하는 방법이다. 해를 생성했을 때와 같은 피치 조정을 사용하지만, HM의 harmony 대신 현재 가장 좋은 harmony를 사용한다. 실제 harmony를 생성할 때는 피치 조정 값인 $\triangle_{pa}$의 2배인 $2\triangle_{pa}$가 되므로 가능성 높은 지역을 더 넓게 탐색할 수 있다. 그리고 하나의 해로 HM이 수렴했을 때 랜덤선택과 같이 다른 곳으로 유도한다.

두 번째로 가장 나쁜 harmony를 좋은 harmony로 업데이트하는 방법이다. 현재 가장 좋은 harmony로 HM의 수정해 선택확률을 높게 하며, 하나의 해로 빠르게 수렴하게 돕는다. HS-PSO에서 좋은 harmony가 선택될 확률을 높이는 것과 비슷한 역할을 한다.

위 2가지방법 중 첫 번째 방법이 우선순위가 높다. 좋은 harmony를 빨리 찾게 하기 위해 그림 3와 같이 현재가장 좋은 harmony로 업데이트 한다. 이후 HM이 현재 가장 좋은 harmony로 수렴하게 되면 현재 가장 좋은 harmony를 피치 조정해 근처 최적 값을 넓게 검색한다.

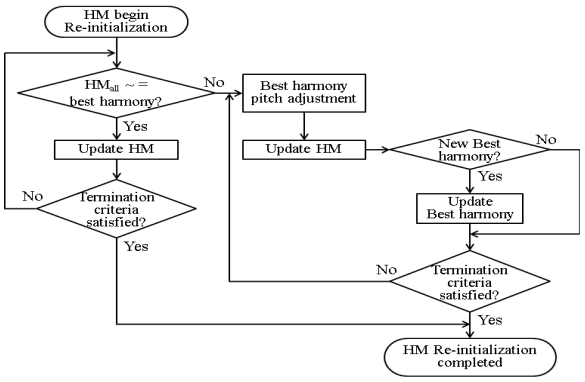

그림 3. HM 수정 순서도
Fig. 3. Modification HM flowchart

4. 테스트 함수 이용한 알고리즘 검증

검증 실험은 식 2의 방정식과 4개의 HM 값을 사용했다. 테스트 함수는 쉬운 분석을 위해 간단한 방정식을 사용했다. 하지만 해의 범위를 $1 \le x_i \le 60$으로 넓게 설정했다.

$$f(x_i) = (x_1 - 36)^2 + (x_2 - 1)^4 + (x_3 - 13)^2 + (x_4 - 21)^2 + (x_5 - 18)^2 + 3 \quad (2)$$

제안하는 알고리즘의(AHS)의 매개변수는 Endurance를 60으로 하고, HMPR은 70으로 고정해 사용했다. 나머지 기본적인 매개 변수인 HMCR, PAR, Iteration은 값의 변화에 따른 성능 변화를 확인하기 위해 2~3가지 값을 선택했다.

초기 HM과 랜덤함수 값은 동일한 값을 사용한다. 모든 HM 초기값에 따른 성능변화를 확인하기는 불가능하다. 그리고 실재 HS에선 초기 HM 값은 랜덤하게 설정하기 때문에 랜덤함수를 이용해 초기 HM값을 생성했다.

초기 HM을 수정하는 방법에 따른 결과를 비교하기위해 동일한 연산과정을 갖도록 했다. 연산과정을 결정하는 매개변수인 HMCR과 PAR값이 같은 Iteration에서 동일한 값을 갖는다. 예를 들어 Iteration=1000에서는 랜덤함수로 매개변수 HMCR=0.98가 생성되어 무작위 선택연산으로 '10'이라는 harmony가 생성되면, AHS와 OHS는 모두 같은 HMCR값인 0.98과 '10'이라는 harmony를 갖는다.

성능 분석을 위해 동일한 1000번 실행결과 사용했다. 값이 너무 적으면 랜덤함수를 기본을 하기 때문에 적은 값을 설정하면 값을 신뢰하기 힘들다. 따라서 충분히 많은 값을 생성하기위해 1000번 실행시켜 성능을 비교했다.

4.1 HMCR과 PAR의 값 변화에 대한 정확도 비교

HS에서 중요한 매개변수인 HMCR과 PAR값에 따른 변화를 비교했다. 각각의 값이 높거나 낮은 경우를 성능 변화를 확인하기위해 HMCR값은 0.9와 0.7을 선택하고, PAR값은 0.3과 0.5를 선택했다. 그리고 매개변수 외에 Iteration이 성능에 미치는 영향을 알기위해 8000, 4000, 3000으로 설정했다. 실험 결과는 그림 4과 같으며, 왼쪽 축에는 1000번 실행 후 해를 찾은 정확도를 백분율로 표시 했다. 기존 harmony 서치알고리즘은 최대 0.1%로 거의 찾지 못했지만, HM을 수정했을 때는 최대 약 70%정도를 높은 정확도를 나타냈다. 그래프에 나타난 특징을 살펴보면 다음과 같다.

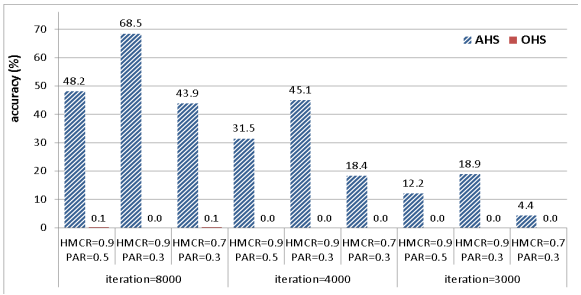

그림 4. HMCR과 PAR 값에 따른 정확도 비교
Fig. 4. accuracy comparison for HMCR and PAR

첫 번째로, 같은 Iteration 에서는 HMCR 값이 성능에 큰 영향을 미쳤다. 그림 4에서 Iteration이 3000, 4000일 때 PAR값에 따라 약 14%와 약 7%정도의 차이를 나타냈다. 하지만, HMCR값은 약 27%와 15%로 PAR보다도 약 2배 정도 높았다.

두 번째로 Iteration이 증가했을 때는 PAR값이 영향을 미쳤다. 그림 4에서 Iteration이 3000에서 4000과 8000으로 증가 했을 때 PAR이 0.5인 경우 최대 20% 밖에 증가하지 못했다. 하지만 PAR값이 '0.3' 인 경우 HMCR값이 0.7과 0.9인 경우 보두 각각 23%와 26%로 높았다. 특히 HMCR값이 0.7인 경우 HMCR값이 0.9이고 PAR은 0.5인 결과와 비교해보면 정확도 차이가 Iteration이 3000일 때 약 1.7배 차이에서 4000일 때 0.1배로 차이가 많이 감소했다.

마지막으로 위 두 가지 특징을 정리하면 HM을 사용하는 방법에 따라 정확도의 차이가 나타난다. HS를 이용해 최적해를 찾는 과정에서 기존 HS는 거의 찾지 못했을 때, 중간에 HM을 수정할 경우 최대 약 70%까지 정확도가 향상되었다. 앞에 첫 번째와 두 번째와 같이 PAR과 HMCR값은 HM을 사용하는 연산자의 선택비율을 조정 하므로 같은 의미로 볼 수 있다.

위 세 가지 특성은 여러 번 실행하더라도 신뢰성 있게 나타난다. 그림 5은 1000번 실행한 결과를 10번으로 나누어 정확도를 표시한 결과로 실행 할 때 마다 정확도가 일정하게 유지되고 있음을 보여준다. 갑자기 떨어지지 않고 평균보다 약 10% 정도의 변화 폭을 보이고 있다. 따라서 HM을 적절히 활용한다면 정확도를 많이 높일 수 있다.

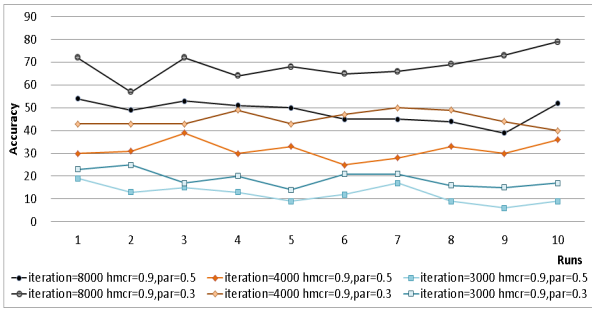

그림 5. HMCR과 PAR 값에 따른 정확도 변화
Fig. 5. Accuracy Changes for HMCR and PAR

4.2 적합도 분포

최적값이나 근처 값을 잘 찾는지 확인하기위해 적합도를 비교했다. 기존 HS 알고리즘은 거의 0%를 보여 제안한 알고리즘만 표시 했으며, 그림 4의 결과 중 Iteration은 가장 작은 3000을 선택했다. 나머지 매개변수는 가장 좋은 성능을 보인 HMCR=0.9, PAR=0.3의 결과 값을 사용했다.

기존 HS은 대부분 '50'에서 '200'사이에 넓게 분포했다. 최적해로 빠르게 수렴해주는 연산자 없으므로 그림 6과 같이 대부분 '50'부터 '200'사이에 넓게 분포했으며, '400'을 초과하는 값도 3개나 찾았다. 수렴하는 해의 적합도가 넓기 때문에 Iteration이 증가하더라도 그림 6과 같이 해를 잘 찾지 못한다. 이러한 문제는 HS뿐만 아니라 메모리를 수정하는 기능이 없는 다른 알고리즘에서도 비슷하게나 나타나날 것으로 판단된다.

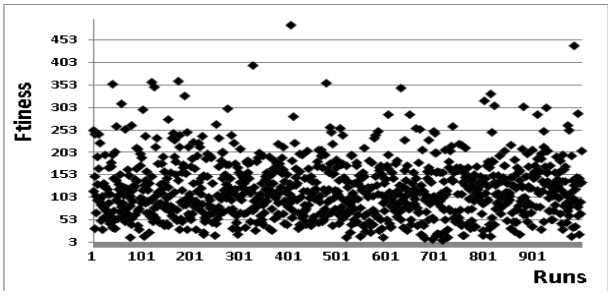

그림 6. HS의 적합도 분포
Fig. 6. Fitness distribution of the HS

제안하는 알고리즘은 최적 값이 대부분 '5'이하로 가깝게 위치했다. 해를 찾는 Iteration이 짧아 정확도가 그림 4과 같이 20%로 낮지만, 그림 7과 같이 대부분 최적 값인 근처인 '5'와 '4'를 찾았다. 최댓값도 '16'으로 그림 5의 결과에서 대부분 '50'이상에서 분포했던 것과 비교했을 때 분포가 확실히 많이 축소되었다. 그리고 그림 7의 결과를 통해 Iteration이 증가할수록 최적값인 '3'으로 이동해 해를 잘 찾음을 알 수 있으며, 이는 HM 수정을 통해 Iteration을 효율적으로 사용한다고 볼 수 있다.

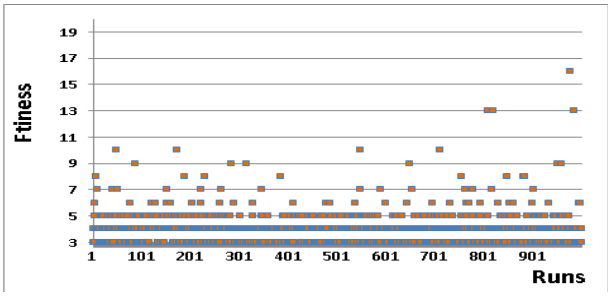

그림 7. 제안하는 HS 적합도 분포
Fig. 7. Fitness distribution of the Suggested HS

5. 결론과 향후 연구

기존의 연구들은 최적해에 가장 가까운 해를 주어진 시간 내에 찾기 위해 좋은 해를 생성하는 메타 휴리스틱 알고리즘이 많이 사용되고 있다. 하지만, 해를 랜덤으로 생성하기 때문에 경우에 따라 수렴 시간이 느려 주어진 시간 내에 해를 찾지 못한다.

작곡가의 인내심을 모방해 하모니서치알고리즘에 적용한 방법을 제안했다. 작곡가가 계속 좋은 최적해를 찾지 못했을 때 인내심의 한계를 초과하게 되면 작업의 효율성을 높이기 위해 악보를 수정한다. 이를 모방해 인내심과 같이 해를 찾지 못했을 때 허용 가능한 최대 Iteration을 설정했다. 이를 초과하게 되면 HM을 수정하게 되며, 현재 구성된 HM으로 계속 좋은 하모니를 찾기 힘들다는 것을 의미한다. 새로운 변수로 시도 횟수를 세는 "Attempt"와 "Attempt "의 최대 시도 횟수인 "Endurance", 메모리 설정 유지 비율을 나타내는 "HMPR"을 설정했다.

현재 가장 좋은 하모니를 이용해 HM을 수정했다. HS을 이용해 찾은 값 중에서 현재 가장 좋은 가장 하모니는 최적값에 가깝거나 최적해 일 가능성이 높다. 따라서 그대로 업데이트해 선택확률을 높이는 방법과 피치를 조정해 넓은 영역을 검색하는 두 가지 방법을 사용했다.

실험 결과에서 신뢰성 있는 정확도와 최적값에 가까운 값을 잘 찾았다. 기존 하모니 서치알고리즘의 정확도는 거의 0%로 낮은 정확도를 보였으며, Iteration이 증가하더라도 성능의 변화가 없었다. 하지만 제시한 알고리즘은 최대 약 70%의 정확도를 보였으며, 근처 약 10% 내외에 위치해 신뢰성 있게 최적해를 찾았다. 최적해를 찾지 못했을 때는 Iteration이 짧더라도 적합도가 최대 '16'이하로 최적값 가까운 값을 찾았다. 따라서 HM을 사용하는 방법에 따라 성능이 향상됨을 확인했다.

향후 연구에서는 다양한 함수와 매개변수에 대한검증 실험이 필요하다. 1개의 테스트함수만을 사용했기 때문에 실재성능을 검증했다고 보기 어려우므로 다양한 함수를 이용한 실험을 진행이 필요하다. 그리고 하모니서치알고리즘에

서 중요한 매개변수인 HMCR과 PAR값의 영향에 따라 성능에 영향을 받으므로 가장 좋은 성능을 나타내는 값을 설정하기위한 실험이 요구된다.

참 고 문 헌

[1] Holland J. H., "Adaptation in Natural and Artificial Systems," MC; *University of Michigan Press*, 1975.

[2] Marco Dorigo, Vittorio Maniezzo, Alberto Colorni, "The Ant System: Optimization by a colony of cooperating agents," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 26, Issue 1, pp. 29-41, 1996.

[3] Kennedy, J., Eberhart, R., "Particle swarm optimization," Neural Networks,1995. *Proceedings., IEEE International Conference on*, vol. 4, pp. 1942-1948, 1995.

[4] Zong Woo Geem, Joong Hoon Kim, G.V. Loganathan, "new heuristic optimization algorithm: harmony search," *Simulation*, vol. 76, pp.60-68, 2001.

[5] L. Bianchi, M. Dorigo, L. M. Gambardella, W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Computing*, vol. 8, pp.239-287, 2009

[6] Beyer HG, "An Alternative Explanation for the Manner in which Genetic Algorithms Operate," *Biosystems*, vol. 41, pp. 1-15, 1997.

[7] 이상경, 고광은, 심귀보, "Harmony Search 알고리즘의 수렴성 개선에 관한 연구," *한국지능시스템학회 2011년도 춘계학술대회*, 제21권, 1호, pp.31-34, 2011.

[8] 김종우, 김원배, 우효섭, "첨단 최적화 기술과 토목공학상의 응용," *대한토목학회지*, 제55권, pp. 4-186, 2007.

[9] Z.W. Geem and K.B. Sim, "Parameter-setting-free harmony search algorithm," *Applied Mathematics and Computation*, vol. 217, 2010.

[10] M. G. H. Omran and M. Mahdavi, "Global-best harmony search," *Applied Mathematics and Computation*, vol. 198, 2005.

저 자 소 개

이상경(SangKyung Lee)

2010년 : 세명대학교 전자공학부 공학사
2010년 ~ 현재 : 중앙대학교 대학원 전자전기공학부 석사과정

관심분야 : Intention Robotic, Intelligent System, Machine Learning, Embedded System
Phone : 02-820-5319
Fax : 02-817-0553
E-mail : raon@wm.cau.ac.kr

고광은(Kwang-Eun Ko)

2007년 : 중앙대학교 전자전기공학부 공학사
2007년 ~ 현재 : 중앙대학교 대학원 전자전기공학부 석박사통합과정

관심분야 : Multi-Agent Robotic Systems (MARS), Machine Learning, Context Awareness, Emotion Recognition Systems
Phone : 02-820-5319
Fax : 02-817-0553
E-mail : kke@wm.cau.ac.kr

심귀보(Kwee-Bo Sim)

1990년 : The University of Tokyo 전자공학과 공학박사

[제21권 제2호(2011년 4월호) 참조]

1991년 ~ 현재 : 중앙대학교 전자전기공학부 교수
2006년 ~ 2007년 : 한국지능시스템학회 회장

E-mail : kbsim@cau.ac.kr
Homepage URL : http://alife.cau.ac.kr