균형 있는 이웃 해 생성 전략을 통한 타부 탐색

전대석, 전향신, 권기호
성균관대학교 정보통신 공학부
e-mail:blade@sphinx.skku.ac.kr, pianochs1@hotmail.com
khkwon@yurim.skku.ac.kr

Tabu Search using Balanced Neighborhood Production Strategy

Dae-Seuk Jeon, Hyang-Sin Jeon, Kye-Ho Kwon
School of Information and Communication Engineering,
Sungkyunkwan University

요 약

타부 탐색은 타부 전략 기법과 최급 강하 알고리즘이 결합된 알고리즘이다. 이는 한번 방문한 해는 다시 방문하지 않음으로써 지역 최적해에 수렴하지 않고 새로운 방향으로 움직이게 하여 공간 탐색 능력 효율을 높인다. 그러나 기존의 타부 탐색에서 이웃 해를 생성하는 방법에 따라 성능이 많이 좌우된다. 좋지 않은 이웃 해를 생성하는 탐색에서는 얻고자 하는 최적해에 수렴하는 시간이 많이 걸린다. 따라서 이웃 해를 생성할 때 해밍 거리를 고려하여 균형 있는 이웃 해를 생성하고, 해 공간을 탐색함으로써 우수한 최적해를 얻게 됨을 본 논문에서는 보여주고 있다. 이는 다양성도 보장되므로 최적해에 수렴해 가는 속도 또한 빠른 것을 보여주고 있다.

1. 서론

산업 전반에 많은 최적화 문제들이 있지만, 오늘날 컴퓨터 성능이 놀라울 정도로 급속하게 발전을 하였음에도 불구하고, 그러한 문제들을 주어진 시간 내에 얻고자하는 최적해를 찾는 것은 아직까지 불가능하다. 그렇기 때문에 최적해를 찾는 것보다 오히려 최적해에 가장 근접한 해를 찾고자 하는 노력을 많이 해왔다.

타부 탐색은 이러한 노력 중에 개발된 기법으로써 복잡하고 복합적인 문제를 해결하기 위해 많이 이용하는 메타 휴리스틱 기법 중에 하나이다.

타부 탐색은 다른 메타 휴리스틱 기법들과 마찬가지로 지역 최적해에 빠지지 않도록 인간의 기억 과정을 흉내내어 탐색과정에서 단기 메모리를 이용하여 이미 방문했던 지역에 대해 재방문을 방지하는 타부 전략 기법과 최급 강하 알고리즘[1]과 결합한 알고리즘이다. 최근의 보고에 따르면 많은 문제에 대해 좋은 결과를 보였다.

그러나 타부 탐색은 복잡한 해 탐색 공간을 가지고 있는 문제에서 좋은 성능을 보여주지만, 이웃 해를 어떻게 생성해내는지에 따라서 전역 탐색 능력이 낮아지므로 최적해를 찾는데 많은 시간이 요구될 뿐만 아니라, 성능이 크게 좌우되는 단점을 가지고 있다. 따라서 이웃 해 생성에 대한 방법을 생각해 볼 필요가 있다.

본 논문에서는 해들간에 해밍 거리를 일정하게 변화를 주는 균형 있는 이웃 해 생성 전략을 제안하고, 이 전략을 타부 탐색과 결합하여 기존의 타부 탐색의 성능을 높여보고자 한다.

본 논문은 2장에서 일반적인 타부 탐색에 대하여 기술하고 3장에서는 본 논문에서 제안한 균형 있는 이웃 해 생성 전략과 결합한 타부 탐색에 대해 살펴보고, 이를 4장에서 시뮬레이션을 통해 결과를 검증하고자한다. 마지막으로 5장에서 시뮬레이션의 결과를 통해 결론을 맺고자 한다.

2. 관련 연구

타부 탐색[2]은 Glover(1977)에 의해 제안된 공간

탐색 방법으로서 최적화 문제에 적용 가능하도록 개량되어졌다[3]. 일반적인 탐색 방법들과 달리 타부 리스트라는 단기 메모리를 이용하여 이미 방문한 적이 있는 해들에 대해 재방문을 방지함으로써 공감 탐색 효율을 높이게 된다. 그리고 탐색 과정에서 타부 리스트에 있지 않고 방문한 해들 중 제일 좋은 움직임을 가진 좋은 해를 선택함으로서 지역 최적해에서 이동을 가능하게 하여 새로운 방향으로 탐색을 진행할 수 있다. 뿐만 아니라 각 반복 단계에서 이웃 해 중에서 가장 좋은 해가 현재의 해보다 우수한가에 대한 평가를 하지 않고, 타부 리스트에 없다면 그 해를 무조건 대체시킨다. 이렇게 함으로서 지역 최적해에 머무르지 않게 된다[3][4].

타부 탐색에서 탐색할 해의 공간을 S라 할 때 최초의 해를 S_0라고 하고 S_{cur}과 S_{best}에 S_0를 복사한다. 그리고 이웃 해의 집합을 N[]라고 할 때 이웃 해의 집합 중 가장 좋은 해인 $N[]_{best}$가 S_{best}보다 좋다면 이를 복사를 한다. 그리고 타부 리스트에 삽입을 한다. 만약 $N[]_{best}$가 S_{best}보다 좋지 않으나 타부 리스트에 들지 않는 $N[]_{best}$는 S_{cur}이 된다. 그리고 타부 리스트에 삽입한다. 만약 생성한 이웃 해 $N[]_{all}$이 타부 리스트에 있다면 타부 리스트를 수정을 하거나 N[]를 수정을 하거나 아니면 더 이상의 해의 개선이 없다면 알고리즘을 중지시킨다. 아래의 그림 1은 타부 탐색 알고리즘이다[3][4].

Tabu_Search
1. 단기 메모리 초기화
2. 최초 해 S_0 생성
3. $S_{cur} = S_{best} = S_0$
4. 정지 조건일 때까지 반복
   1′. 이웃해 집합 N[] 생성
   2′. $N[]_{all}$이 단기 메모리에 있으면
      1″. 반복문을 탈출
   3′. If S_{best}보다 $N[]_{best}$가 좋으면
      1″. $S_{best} = N[]_{best}$
      2″. 단기 메모리에 $N[]_{best}$ 삽입
   4′. else
      1″. $S_{cur} = N[]_{best}$
      2″. 단기 메모리에 $N[]_{best}$ 삽입
5. 종료할 있는 조건이면 종료,
   그렇지 않으면 4단계로 움직임

그림 1 타부 탐색 알고리즘

3. 균형 있는 이웃해 생성 전략과 결합한 타부 탐색

본 논문에서 제안하는 균형 있는 이웃해 생성 전략(BNPS)는 반복 과정에서 선택된 S와 S의 이웃해 N_1~N_{10}을 발생할 때 일정한 해밍거리로 이웃 해들간의 변화를 똑같이 하여 발생한다. 예를 들어 그림 2와 같이 해 S는 모든 이웃 해 N_1~N_{10} 해밍 거리를 계산해보면 1이 나온다.

S : 1010101010 :	해밍거리
N1 : 1010101011 :	1
N2 : 1010101000 :	1
N3 : 1010101110 :	1
N4 : 1010100010 :	1
N5 : 1010111010 :	1
N6 : 1010001010 :	1
N7 : 1011101010 :	1
N8 : 1000101010 :	1
N9 : 1110101010 :	1
N10: 0010101010 :	1

그림 2 이웃해 생성의 예

BNPS
1. 단기 메모리 초기화
2. 최초 해 S_0 생성
3. $S_{cur} = S_{best} = S_0$
4. 정지 조건일 때까지 반복
   1′. **해밍거리 일정한 이웃해 집합 N[] 생성**
   2′. $N[]_{all}$이 단기 메모리에 있으면
      1″. 반복문을 탈출
   3′. If S_{best}보다 $N[]_{best}$가 좋으면
      1″. $S_{best} = N[]_{best}$
      2″. 단기 메모리에 $N[]_{best}$ 삽입
   4′. else
      1″. $S_{cur} = N[]_{best}$
      2″. 단기 메모리에 $N[]_{best}$ 삽입
5. 종료할 있는 조건이면 종료,
   그렇지 않으면 4단계로 움직임

그림 3 BNPS

그림 3은 기존 타부 탐색 알고리즘에 비교해서 이웃해 생성부분만 바뀐 것이다. 해 S와 형태의 변

화는 비록 한 비트의 차이로 작다고 하지만 실제로 해 S와 이웃 해 N_1~N_{10}은 변화는 작을 수도 있고 클 수도 있는 것이기 때문에 알 수가 없다. 즉 실질적 서로 연관성이 전혀 없다. 따라서 다양성을 보장이 되는 것이다. 다시 말해서 전역 탐색능력이 향상되는 것을 의미한다.

그리고 다양성을 위해서 비트 여러 개를 바꿔 보는 것을 생각 해볼 수 있다. 그러나 비트가 많이 바뀐다고 다양성이 풍부해지는 것도 아니고 비트 하나가 바뀐다고 적은 변화를 가져다 주는 것은 아니다. 그렇지만 이러한 방법 역시 다양성을 보장 해주는 것만은 확실하다.

4. 시뮬레이션

본 장에서는 몇 가지 함수를 통해 해밍거리를 일정하게 하여 이웃 해들의 변화에 대해 타부 탐색의 성능 평가를 하였다.

다양한 최적화 문제 중에서 파라미터 최적화 문제는 타부 탐색을 적용하기 쉽다[5]. 첫 번째 함수는 파라미터 x1과 x2를 조절하여 f(x1, x2)의 최대값을 구하는 문제를 사용하여 일반적인 타부 탐색과 본 논문에서 제안한 균형 있는 이웃해 생성 전략과 결합한 타부 탐색의 성능 평가부터 해보았다[6][7].

$$f_1(x_1, x_2) = 21.5 + \sin(4\pi x_1) + x_2\sin(20\pi x_2)$$

$$-3.0 \le x_1 \le 12.1,\ 4.1 \le x_2 \le 5.8$$

$$f_2(x_i) = \sum_{i=1}^{3} x_i^2$$

$$-5.12 \le x_i \le 5.12$$

$$f_3(x_i) = 100(x_1^2 - x_2)^2$$

$$-2.048 \le x_i \le 2.-48$$

$$f_4(_i) = \sum_{I=0}^{5} integer(x_i)$$

$$-5.12 \le x_i \le 5.12$$

그림 4 파라미터 최적화 함수

두 번째, 세 번째, 네 번째 함수들은 마찬가지로 함수 값이 최대가 되는 x_i를 구하는 것이다. 첫 번째 함수는 유효 자리를 소수점 아래 6자리로 하였고 나머지 함수들은 소수점 둘째 자리까지 구하였다.

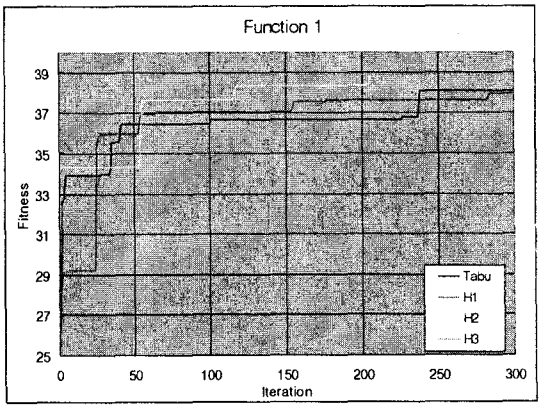

그림 5 함수 1의 결과

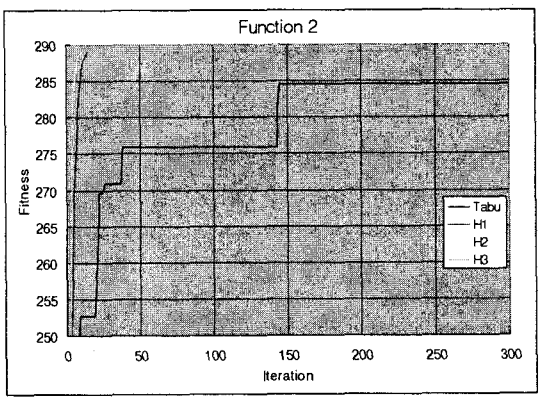

그림 6 함수 2의 결과

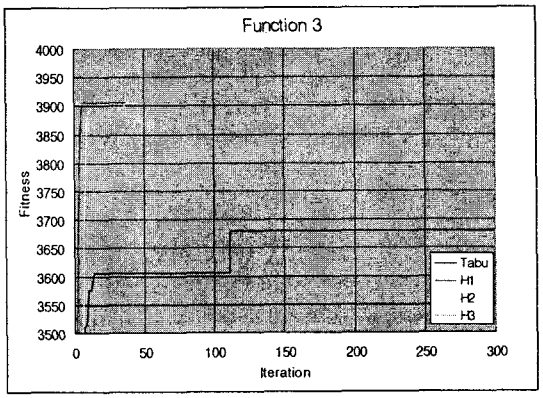

그림 7 함수 3의 결과

Tabu는 타부 탐색을 통한 결과이고 H1은 해밍거리를 1로 한 결과이고, H2는 해밍거리를 2로 한 결과이고, H3은 해밍거리를 3으로 한 그림이다. 그림 5에서 첫 번째 함수가 최적해로 수렴해 가는 모습을 보여주고 있다. 기존의 타부 탐색으로도 좋은 결과를 얻었지만 나머지 그림 6. 7, 8에서는 기존의 타부 탐색이 좋게 나오지 않은 것을 확인할 수가 있었다. 속도 면에서도 그림 6, 7, 8에서 보면 보다 빨리 수렴해감을 알 수가 있다.

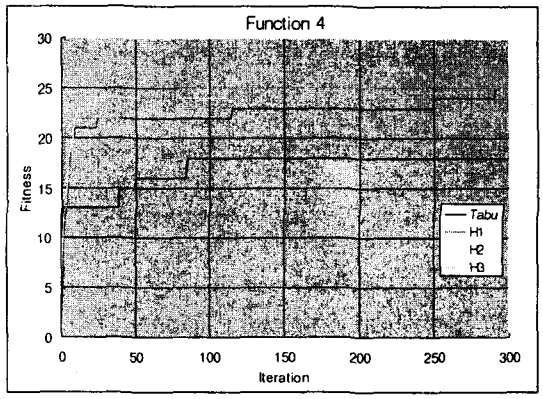

그림 8 함수 4의 결과

4. 결론 및 향후 연구과제

타부 탐색은 보통의 지역적 탐색법처럼 하나의 후보에서 시작하여 주어진 종료조건을 만족할 때까지 반복적으로 이웃해로 이동을 한다. 그러나 타부 탐색은 지역 최적해에 빠지는 것을 피하기 위한 전략들을 사용함으로써 지역적 탐색을 넘어선다. 하나의 해에서 이웃의 다른 해로 이동할 때, 비록 이전의 해보다 좋지 않더라도 이웃 해 중 가장 좋은 해로 이동을 하여 지역 최적해를 벗어나기 위한 이동을 시도한다. 그러나 다음의 이동에서 가장 좋은 해들을 계속 선택하다 보면 이전의 국지 최적해로 다시 돌아올 확률이 높으므로, 상당수의 이동이 이어지는 동안 지난번의 지역 최적해로의 회귀를 방지하는 수단이 필요하다. 이를 위해서 타부 탐색은 타부 리스트라는 단기 메모리를 이용한다[2][3].

본 논문에서는 기존의 이웃해 생성에 대한 전략이 고려되지 않은 타부 탐색보다 해밍거리를 일정하게 하여 균형 있게 이웃해 변화를 주어 생성을 하는 타부 탐색 결과가 우수함을 보여 주었다. 또한 실험의 결과 지역적 최적해에 빠지지 않고 보다 빠르게 수렴하는 것도 보여주었다.

기존 타부 탐색에서 다양한 이웃해 생성 전략이 결합된 알고리즘들이 존재한다. 향후 이런 알고리즘과 본 논문에서 제안한 알고리즘과 비교를 하고자 하고, 최적화가 필요 하는 시스템에서 응용하고자 한다.

참고문헌

[1] Patric R.J.Ostergard Constructing Covering Codes by Tabu Search

[2] Glover, F., "Tabu Search, Part I," ORSA Journal on Computing, Vol.1, No.3, pp.190-206, 1989.

[3] Glover, F., "Tabu Search, Part II," ORSA Journal on Computing, Vol.2, No.1, pp.4-32, 1989.

[4] F. Glover and M. Laguna, Tabu search, Kluwer Academic Publishers, 1977.

[5] D.E. Goldberg, "Genetic Algorithms in Search, Optimization & Machine Learning", Addison Wesley, 1989.

[6] Zbigniew Michalewicz, "Genetic Algorithms + Data Structure = Evolution Programs", Springer-Verlag, New York, Third Edition, 1996.

[7] Gen & Cheng, "Genetic Algorithms & Engineering Optimization", Wiley-Interscience, New York, 1999.