韓國컴퓨터情報學會 論文誌
第15卷 第5號, 2010. 5.

2010-15-5-1-5

제약 만족 최적화 문제의 해결을 위한 지역 탐색과 제약 프로그래밍의 결합

황 준 하*

An Integration of Local Search and Constraint Programming for Solving Constraint Satisfaction Optimization Problems

Junha Hwang*

요 약

제약 만족 최적화 문제는 복잡한 제약 조건을 포함하는 동시에 비용을 최소화하는 최적화 문제로 정의된다. 지역 탐색과 제약 프로그래밍은 각각 이와 같은 문제의 해결을 위한 도구로서 활용되어 왔다. 본 논문에서는 탐색 성능 향상을 위해 지역 탐색과 제약 프로그래밍을 결합하는 방안을 제시하고 있다. 기본적으로 대상 문제의 해결을 위해 지역 탐색을 사용한다. 그러나 지역 탐색만을 사용할 경우 제약 조건을 모두 만족하는 실행 가능한 이웃해를 생성하는 것이 매우 힘들어진다. 따라서 본 논문에서는 이웃해 생성을 위한 도구로 제약 프로그래밍을 도입하였다. 가중치가 부여된 *N*-Queens 문제를 대상으로 한 실험 결과, 본 논문에서 제시한 방법을 통해 탐색 성능을 획기적으로 향상시킬 수 있음을 확인할 수 있었다.

Abstract

Constraint satisfaction optimization problem is a kind of optimization problem involving cost minimization as well as complex constraints. Local search and constraint programming respectively have been used for solving such problems. In this paper, I propose a method to integrate local search and constraint programming to improve search performance. Basically, local search is used to solve the given problem. However, it is very difficult to find a feasible neighbor satisfying all the constraints when we use only local search. Therefore, I introduced constraint programming as a tool for neighbor generation. Through the experimental results using weighted *N*-Queens problems, I confirmed that the proposed method can significantly improve search performance.

▸ Keyword : 제약 만족 최적화 문제(Constraint Satisfaction Optimization Problem), 지역 탐색(Local Search), 제약 프로그래밍(Constraint Programming)

• 제1저자 : 황준하
• 투고일 : 2010. 04. 05, 심사일 : 2010. 04. 15, 게재확정일 : 2010. 04. 19.
* 금오공과대학교 컴퓨터공학부 부교수
※ 본 논문은 금오공과대학교 학술연구비에 의하여 연구된 논문임.

Ⅰ. 서 론

제약 만족 문제(Constraint Satisfaction Problem)는 변수(variables), 도메인(domain), 제약조건(constraints)으로 표현될 수 있다. 변수는 값을 결정해야 할 변수들을 의미하고 도메인은 각 변수들이 가질 수 있는 값들을 의미하며 제약조건은 하나의 변수에 대한 제약 또는 변수들 사이의 관계를 의미한다. 결국 제약 만족 문제는 모든 제약조건을 만족하는 변수들의 값을 찾는 문제로 요약될 수 있다[1]. 한편 최적화 문제(Optimization Problem) 역시 많은 경우에 있어서 변수들의 값을 결정하는 문제로 표현될 수 있는데, 대상 문제에서 요구하는 달성 목표를 목적 함수(Objective Function)로 표현하고 목적 함수의 값을 최대화 또는 최소화 할 수 있는 변수들의 값을 결정하는 것이다[2].

제약 만족 문제를 해결하기 위한 대표적인 기법으로는 제약 프로그래밍(Constraint Programming)이 있다. 제약 프로그래밍은 깊이 우선 탐색(Depth-first Search)을 기반으로 변수의 값을 하나씩 순차적으로 결정하되 하나의 변수값이 결정될 때마다 제약조건을 통해 다른 변수의 도메인을 축소시킴으로써 해를 보다 빨리 찾을 수 있도록 유도한다. 최적화 문제를 해결하기 위한 대표적인 기법으로는 지역 탐색(Local Search)이 있다. 지역 탐색은 하나의 완전한 해로부터 출발하여 현재해를 조금씩 개선시켜 나가는 방식을 사용한다.

제약 만족 최적화 문제(Constraint Satisfaction Optimization Problem, CSOP)는 제약 만족 문제로 표현되는 문제들 중 목적 함수를 포함하는 문제라 할 수 있다[1]. 즉, 복잡한 제약조건을 모두 만족하면서 주어진 목적 함수의 값이 최대화 또는 최소화될 수 있는 변수들의 값을 결정해야 한다. 이를 위해서도 기존의 제약 프로그래밍과 지역 탐색이 각각 활용될 수 있다. 제약 프로그래밍의 경우 사실상 모든 탐색 공간을 탐색함으로써 최적해의 도출을 보장할 수 있지만 해의 개선 속도가 느리며 최적해를 보장하기 위해 너무 많은 시간이 소요될 수 있다는 단점이 있다. 반면에 지역 탐색의 경우 보다 짧은 시간 내에 준최적해의 도출이 가능하지만, 최적해의 도출을 보장할 수 없으며 복잡한 제약조건을 포함하는 경우 제약조건을 모두 만족하는 해의 도출 자체가 어려워지는 경향이 있다.

본 논문에서는 제약 만족 최적화 문제의 해결 방안으로서 지역 탐색과 제약 프로그래밍을 결합하는 방안을 제시하고 있다. 전체적인 구조는 지역 탐색 중 단순 언덕오르기 탐색(Simple Hill-climbing Search)을 기반으로 한다. 먼저 초기해 생성 시 제약 프로그래밍을 적용한다. 그리고 이웃해 생성 시 일정 개수의 변수들의 값을 변경하게 되는데 이를 위해서도 제약 프로그래밍을 적용하고 있다. 이와 같은 방법을 통해 기존의 지역 탐색의 단점인 제약조건을 만족하는 해에 대한 도출의 어려움을 해소할 수 있도록 하였다.

본 논문에서 제시한 방안의 유효성을 검증하기 위해 가중치가 부여된 *N*-Queens 문제를 사용하였다. 다양한 규모의 문제에 대한 실험 결과, 본 논문에서 제시한 방법을 통해 제약 프로그래밍이나 지역 탐색을 단독으로 사용할 때보다 탐색 성능을 획기적으로 향상시킬 수 있음을 확인할 수 있었다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 지역 탐색 기법들과 제약 프로그래밍의 결합에 관한 기존 연구에 대해 살펴본다. 그리고 Ⅲ장에서는 제약 만족 최적화 문제의 해결을 위한 지역 탐색과 제약 프로그래밍의 일반적인 적용 방식에 대해 각각 설명하며 아울러 본 논문에서 제시하는 결합 방안에 대해 설명한다. Ⅳ장에서는 대상 문제인 가중치가 부여된 *N*-Queens 문제에 대해 설명한 후 실험 결과를 분석한다. 마지막으로 Ⅴ장에서 결론 및 향후 과제에 대해 설명한다.

Ⅱ. 관련 연구

제약 만족 최적화 문제를 위한 지역 탐색과 제약 프로그래밍의 가장 단순한 결합 방식은 [3]과 같다. 이 연구에서는 초기해 생성을 위해 제약 프로그래밍을 적용하였다. 그리고 반복적으로 지역 탐색을 수행하게 되는데 이때 이웃해 생성을 위해 일부 변수들의 값을 변경하게 되며, 생성된 이웃해의 유효성을 검사하기 위해 제약 프로그래밍이 동원된다. 그런데 대부분 지역 탐색의 경우 이웃해를 무작위로 생성하게 되는데, 이때 제약조건을 만족하는 이웃해의 생성이 쉽다면 이 방법이 효과적일 수 있다. 그러나 복잡한 제약조건을 포함하는 대부분의 문제에 있어서 무작위로 생성된 이웃해는 모든 제약조건을 만족하기가 어려우며 이에 따라 다음 해로의 이동 자체가 매우 어려워지게 된다.

기존 연구 [4]에서는 무선 네트워크의 설계 문제를 해결하기 위해 지역 탐색과 제약 프로그래밍을 활용하였다. 그러나 이 연구에서는 대상 문제를 두 개의 부문제로 나누어 접근하되 하나는 지역 탐색을 적용하고 다른 하나는 제약 프로그래밍을 적용하고 있기 때문에 두 기법을 밀접하게 결합했다고 보기는 어려운 상황이다. 기존 연구 [5]에서도 차량 일정계획 시스템을 개발하기 위해 제약 프로그래밍과 지역 탐색의 일종인 타부 탐색(Tabu Search)[6]을 사용하고 있으나 이 역시 문제를 분리하여 별도의 문제에 각각의 기법을 적용하고 있

다. [7] 또한 복잡한 제약 만족 최적화 문제인 간호사 일정계획 문제를 해결하기 위해 원문제를 2단계로 나누어 각각 제약 프로그래밍과 가변 이웃해 탐색(Variable Neighborhood Search)[8]을 적용하였다. 이와 같이 하나의 응용 문제 내의 일부 문제를 해결하기 위해 두 가지 기법이 각각 적용된 사례는 꽤 많은 편이다.

지역 탐색과 제약 프로그래밍을 보다 밀접하게 결합한 연구로는 다음과 같은 연구들이 있다. 기존 연구 [9]에서는 개미 군집 최적화(Ant Colony Optimization)[10] 수행 시 개미가 경로를 하나씩 결정할 때마다 제약 전파를 통해 다음 경로들에 유효한 경로만을 남김으로써 보다 빨리 유효한 경로를 찾을 수 있도록 하였다. 개미 군집 최적화는 기본적으로 점진적으로 해를 구성하는 방법을 취하므로 변수의 값이 한 번에 하나씩 결정되기 때문에 제약 프로그래밍과의 결합이 효과적이라 할 수 있다. 그러나 개미 군집 최적화의 특성 상 주로 대상 문제가 그래프에 의해 표현되는 경우에 한하여 적용이 적합할 것으로 판단된다. [11]에서는 시간표 문제 해결을 위해 입자 군집 최적화(Particle Swarm Optimization)를 기반으로 하여 제약 프로그래밍을 활용하고 있는데, 제약 프로그래밍은 후보해의 유효성 검사 및 수정을 위해 동원되고 있어 본 논문의 방법보다는 다소 소극적인 역할을 담당하고 있는 것으로 판단된다. [12]에서는 지역 탐색을 기반으로 탐색이 불필요한 이웃해를 제거하기 위해 제약 프로그래밍의 제약 전파를 사용하였으나 기본적으로 최적화 문제가 아닌 제약 만족 문제를 대상으로 하고 있다. [13]에서는 비선형 최적화 문제의 해결을 위해 정수계획법과 지역 탐색을 결합한 바 있지만 본 논문의 대상 문제인 제약 만족 최적화 문제보다는 일반적인 최적화 문제에 적합한 방법을 제시하였다.

본 논문에서는 지역 탐색과 제약 프로그래밍을 밀접하게 결합하는 한 가지 방안을 제시하고 있는데, 지역 탐색 기법들 중 단순 언덕오르기 탐색의 효율을 향상시키기 위해 단순 언덕오르기 탐색의 세부 요소로 제약 프로그래밍을 활용하고 있다. 이는 [3]을 확대 발전시킨 것으로서 초기해 뿐만 아니라 탐색 도중에도 제약 프로그래밍을 적극적으로 활용하고 있다. 본 방법은 단순 언덕오르기 탐색의 특성 상 [9]의 개미 군집 최적화보다 적용 범위가 훨씬 넓을 것으로 판단된다.

III. 지역 탐색과 제약 프로그래밍의 적용

본 장에서는 먼저 제약 만족 최적화 문제를 해결하기 위한 기존의 제약 프로그래밍 및 지역 탐색에 대해 각각 설명하고 마지막으로 본 논문에서 제시하고 있는 두 기법의 결합 방안에 대해 설명한다.

1. 제약 프로그래밍

제약 만족 문제를 위한 제약 프로그래밍은 깊이 우선 탐색을 기반으로 제약 전파와 백트랙킹을 사용한다. 제약 만족 최적화 문제를 위한 탐색 역시 동일한 기법을 사용하되 지금까지 도출된 가장 좋은 해의 정보를 사용하여 불필요한 탐색 공간을 줄일 수 있도록 노력하며, 이와 같은 기법을 Branch & Bound라 부른다[1].

그림 1은 변수 3개(x_1, x_2, x_3)로 이루어져 있고 각각 0과 1의 값을 도메인으로 가지고 있는 제약 만족 최적화 문제에 대해 깊이 우선 탐색 방식으로 수행되는 Branch & Bound 탐색의 예를 보인 것이다. 각 노드의 값은 그 때까지 결정된 값들에 의해 계산된 비용을 의미하고 각 분기(branch) 위의 값은 해당 변수에 할당된 값을 의미한다. 각 노드에서의 최소 비용 f는 $(g + h)$로 구할 수 있는데 g는 그 때까지 확정된 비용을 의미하고 h는 그 노드로부터 마지막 노드가 결정될 때까지 추가될 것으로 예상되는 최소 비용을 의미한다. 그렇다면 그림 1에서와 같이 특정 노드의 예상 최소 비용 f가 현재까지 찾은 가장 좋은 해의 비용(bound)보다 같거나 큰 경우, 그 노드 이하의 탐색은 불필요한 탐색이 되므로 이하의 탐색을 생략할 수 있다. 결과적으로 불필요한 탐색 공간을 줄임으로써 보다 빠르게 더 좋은 해의 도출이 가능하게 된다. 현재 노드로부터 추가될 최소 비용 h는 대상 문제에 따라서는 계산이 어려운 경우도 있으며 이 경우 h의 값은 0이 된다. 대상 문제에 따라서는 아직 결정되지 않은 각 변수의 도메인들을 기반으로 계산이 가능한 경우도 있다.

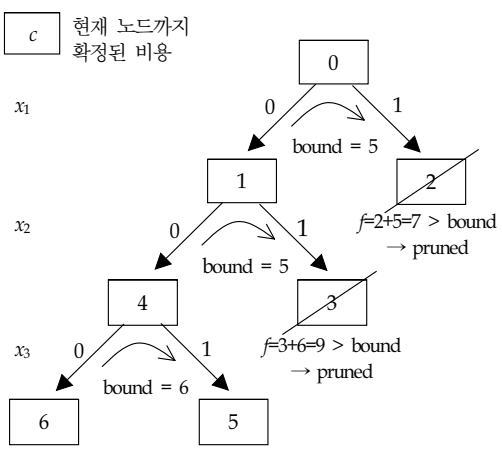


그림 1. Branch & Bound 탐색의 예
Fig. 1. An Example of Branch & Bound

제약 프로그래밍은 탐색 공간을 모두 열거하는 방식으로서 최적해의 도출을 보장한다. 그러나 제약 전파 및 Branch & Bound를 통해 불필요한 탐색 공간을 효과적으로 줄일 수 있다 하더라도 결국 모든 경우를 고려해야 하기 때문에 문제의 규모가 조금만 커지더라도 최적해가 도출되기까지 과도한 시간을 필요로 하게 된다.

제약 만족 문제 해결 시 제약 프로그래밍의 적용에 있어서 변수 선택 순서 및 변수값 선택 순서가 탐색 성능에 큰 영향을 미치는 것으로 알려져 있다. 대상 문제에 적합한 변수 선택 및 변수값 선택 방법이 존재하지 않는다면 일반적으로 변수의 선택 방법으로는 현재 도메인에 남아 있는 값의 개수가 가장 적은 변수를 우선적으로 선택하는 most constrained variable이 효과적이며, 변수값 선택 방법으로는 다른 변수의 도메인을 가장 적게 축소시키는 least constraining value가 효과적인 것으로 알려져 있다. 실험에 의하면 제약 만족 최적화 문제인 본 논문의 대상 문제에 있어서도 변수 선택 방법으로는 most constrained variable이 효과적인 것으로 보이나 변수값 선택 방법으로는 최소값을 우선적으로 배정하는 것이 효과적인 것으로 보인다.

본 연구에서는 제약 프로그래밍 구현을 위해 ILOG Solver 6.7을 사용하였다[14]. ILOG Solver는 상용 제약 프로그래밍 라이브러리로서 현재 학술 및 상업적 용도로 가장 많이 사용되고 있다. 제약 프로그래밍과 관련된 모든 실험에 있어서 변수 선택 방법으로는 most constrained variable을 사용하였고 변수값 선택 방법으로는 남아있는 값들 중 최소값을 우선적으로 배정하는 방식을 취하였다.

2. 지역 탐색

지역 탐색은 하나의 완전한 해로부터 출발하여 이웃해를 반복적으로 탐색함으로써 해를 개선시켜 나가는 방식으로서 언덕오르기 탐색(Hill-Climbing Search), 단순 언덕오르기 탐색, 시뮬레이티드 어닐링(Simulated Annealing), 타부 탐색 등이 있다. 언덕오르기 탐색과 타부 탐색은 모든 이웃해들 중 가장 좋은 해를 다음 해로 선택하는 최급 상승(steepest ascent) 전략을 기반으로 하고 있는 반면에 단순 언덕오르기 탐색과 시뮬레이티드 어닐링은 현재해로부터 무작위로 하나의 이웃해를 생성한 후 현재해보다 좋으면 이동하는 최우선 향상(first improvement) 전략을 기반으로 하고 있다[15].

제약 만족 최적화 문제를 위한 일반적인 단순 언덕오르기 탐색 알고리즘은 그림 2와 같으며 최대화 문제로 가정하였다. 지역 탐색에서는 제약조건을 그대로 처리하기 어렵기 때문에 제약조건을 목적함수의 일부로 포함시키게 되며 이는 그림 2의 식 (1)과 같이 표현될 수 있다. 제약조건을 위배한 경우를 점수화한 벌점(*Penalty*)을 원래 목적함수(*Obj*)에 추가하였는데, 최대화 문제일 경우 원래 목적함수에서 벌점을 빼면 된다. 이때 원래 목적함수와 벌점의 비율(a)을 결정해야 한다. a가 커지면 제약조건을 만족하는 해를 도출하기 위한 탐색의 성격이 강해지고 이 값이 작아지면 원래 목적함수 값을 최대화하기 위한 탐색의 성격이 강해진다. 이후의 탐색 과정은 일반적인 단순 언덕오르기 탐색과 동일하다. 초기해를 생성하여 현재해로 설정한 후 일부 변수들(k)의 값을 변경하여 이웃해를 만들고 이 해가 현재해보다 같거나 좋으면 이동하게 된다.

```
// SHC : Algorithm for maximization problems
Algorithm Simple_Hill_Climbing_Search
    x : Variable vector (Current solution).
    Obj : Objective function.
    Penalty : Penalty function for violation of constraints.
    TotalObj : Total objective function.
                TotalObj = Obj - a × Penalty ····· (1)
    k : The number of variables to be changed.
Begin
    Start with a random initial solution x
    While stopping condition is not met Do
        Generate a neighbor solution x* by changing
                                the value of k variables
        If TotalObj(x*) ≥ TotalObj(x) Then
            x = x*
    End While
    return x
End Begin
```

그림 2. CSOP를 위한 단순 언덕오르기 탐색
Fig. 2. Simple Hill-Climbing Search for CSOP

시뮬레이티드 어닐링 역시 단순 언덕오르기 탐색과 매우 유사하다. 단순 언덕오르기 탐색과 마찬가지로 이웃해 하나를 생성하고 현재해보다 더 좋거나 같다면 이웃해를 현재해로 설정한 후 탐색을 진행해 나간다. 그렇지만 단순 언덕오르기 탐색과는 달리 현재해보다 좋지 않다 하더라도 이동이 가능하다. 이때 이동 여부는 $e^{\triangle E/t}$의 확률에 따라 결정되는데 $\triangle E$는 해의 개선 정도인 ($TotalObj(x^*)$ - $TotalObj(x)$)을 의미한다. 즉, 좋지 않은 정도가 커지면 커질수록 이동 가능성은 낮아지게 된다. 그리고 온도를 의미하는 t는 탐색 초기에는 높은 값을 가지다가 탐색이 진행될수록 점점 감소하게 된다. 이로 인해 탐색 초기에는 좋지 않은 해로의 이동 확률이 높은 반면에 탐색이 진행될수록 이동 확률이 낮아지게 되며 결국 t 값이 0이 되면 단순 언덕오르기 탐색과 같아진다.

단순 언덕오르기 탐색의 경우 지역 최적해로부터 벗어나지

못하는 경우가 많은데, 시뮬레이티드 어닐링은 이를 보완할 수 있는 효과적인 알고리즘으로 알려져 있다. 본 연구에서는 제안한 기법의 성능을 검증하기 위해 제약 프로그래밍 외에 시뮬레이티드 어닐링을 비교 대상으로 활용하였다.

3. 지역 탐색과 제약 프로그래밍의 결합

본 논문에서 제안하는 제약 만족 최적화 문제의 해결을 위한 지역 탐색과 제약 프로그래밍의 결합 방법은 그림 3과 같이 단순 언덕오르기 탐색을 기반으로 하고 있다. 먼저 제약 프로그래밍을 통해 제약조건을 모두 만족하는 초기해를 도출한다. 그리고 이 해를 개선하기 위해 단순 언덕오르기 탐색 과정을 반복 수행하게 되는데 기존 단순 언덕오르기 탐색과는 달리 이웃해 생성 시 k개의 변수값을 무작위로 변경하는 것이 아니라 선택된 k개 이외의 변수값을 현재해와 동일하게 고정시킨 후 또 다시 제약 프로그래밍을 수행한다. 이와 같은 방법을 통해 제약조건을 모두 만족하는 해를 보다 쉽게 도출하게 된다. 이때 특별한 조건을 부여하지 않는다면 제약조건을 만족하는 해가 무작위로 생성되어 현재해보다 더 좋은 해의 도출이 어려워지게 된다. 이를 방지하기 위해 목적함수 값이 현재해의 목적함수 값보다 같거나 크다는 제약조건을 추가하였다. 따라서 매 반복 시마다 최소한 현재해에 비해 같거나 더 좋은 해가 도출된다.

```
// SHC+CP : Algorithm for maximization problems
Algorithm Local_Search_with_Constraint_Programming
    x : Variable vector (Current solution).
    Obj : Objective function.
    CurObj : Current objective function value.
    k : The number of variables to be selected.
Begin
    CP = Start a Constraint Programming
    Add all constraints to CP
    x = Make an initial solution with CP
    While stopping condition is not met Do
        CurObj = Obj(x)
        Select k variables randomly from x
        CP = Restart the Constraint Programming
        Add a constraint to CP
            (Fix values of unselected variables of x)
        Add a constraint to CP (Obj >= CurObj)
        x = Make a neighbor solution with CP
    End While
    return x
End Begin
```

그림 3. CSOP를 위한 지역 탐색과 제약 프로그래밍의 결합
Fig. 3. Integration of Local Search and Constraint Programming for CSOP

그림 3의 알고리즘에서 결정해야 할 파라미터는 단 한 가지로 이웃해 생성 시 값을 변경할 변수의 개수인 k의 값이다. k의 값이 작아지면 다른 지역 탐색과 마찬가지로 지역 최적해로부터 빠져나오기 힘들 수 있으며, k의 값이 커지면 제약 프로그래밍을 통해 해를 도출하는 데 어려움을 겪을 수 있다. 따라서 제약 프로그래밍을 통해 해를 쉽게 도출할 수 있는 범위 내에서 비교적 큰 수를 지정하는 것이 효과적일 것으로 판단된다.

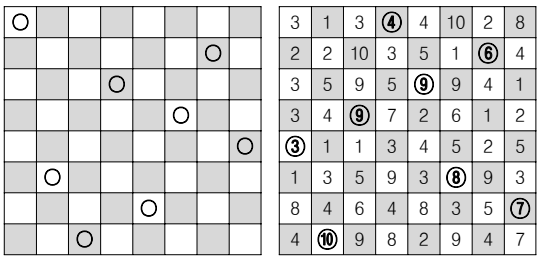
그림 3을 기반으로 하되 단순 언덕오르기 탐색이 아닌 다른 지역 탐색과 제약 프로그래밍을 결합한 방식을 고려해 볼 수 있다. 우선 이웃해 집합으로부터 가장 좋은 해를 선택하여 이동하는 언덕오르기 탐색을 고려해 볼 수 있는데, k개의 변수를 변경하는 알고리즘의 특성 상 이웃해 집합을 정의하기가 곤란해질 수 있다. k의 값이 조금만 커지더라도 k개를 선택하는 조합이 너무 많아지기 때문이다. 따라서 언덕오르기 탐색보다는 단순 언덕오르기 탐색이 보다 적합할 것으로 판단하였다. 또 한 가지 고려해 볼 수 있는 결합 방식은 시뮬레이티드 어닐링과 제약 프로그래밍을 결합하는 것이다. 시뮬레이티드 어닐링은 지역 최적해로부터 탈출하기 위해 현재해보다 좋지 않다 하더라도 확률적으로 이동이 가능하다. 그러나 k의 값이 충분히 크다면 지역 최적해에 빠질 위험은 줄어들게 된다. 그럼에도 불구하고 시뮬레이티드 어닐링을 적용한다면 불필요하게 나쁜 해로 이동하는 현상이 발생할 가능성이 높아질 것으로 예상된다. 더군다나 시뮬레이티드 어닐링을 적용할 경우 k의 값 외에도 어닐링 스케줄, 좋지 않은 해의 허용 수준 등 고려해야 할 파라미터가 많아져 대상 문제 또는 데이터에 따라 보다 섬세한 파라미터 조정이 필요하게 된다. 단순 언덕오르기 탐색의 경우 k의 값만 결정하면 되며 이 값 또한 제약 프로그래밍의 성능에 좌우되므로 대상 문제나 데이터에 덜 민감할 것으로 기대된다.

IV. 대상 문제 및 실험 결과

1. 대상 문제

본 논문에서 제시한 기법의 성능을 검증하기 위해 가중치가 부여된 N-Queens 문제를 사용하였다. 원래 N-Queens 문제는 N개의 Queen을 $N \times N$의 체스 보드에 위치시키되 모든 2개의 Queen 쌍들에 대해 수평, 수직, 대각선의 직선 방향에 위치시킬 수 없는 문제로 정의된다. 그림 4(a)는 8-Queens 문제에 대한 하나의 해를 나타낸 것이다. 본 논문에서는 N-Queens 문제를 변형하여 제약 만족 최적화 문제

로 변경하였다. 각 격자에 1부터 10까지의 값을 부여하고 Queen들이 위치한 격자의 값들의 합을 최대화하는 것이 목표이다. 따라서 대상 문제는 직선 방향에 위치시키지 않되 가중치의 합을 최대화하는 제약 만족 최적화 문제로 정의될 수 있다. 그림 4(b)는 가중치가 부여된 8-Queens 문제에 대한 최적해의 예를 보인 것이다.



(a) 원문제 (b) 가중치가 부여된 문제(*Obj*=56)

그림 4. 8-Queens 문제의 예
Fig. 4. An Example of 8-Queens Problem

N-Queens 문제에 대한 기본적인 모델링은 다음과 같다. 변수의 개수는 *N*개이고 각 변수는 0부터 (*N*-1)까지의 값을 도메인으로 가질 수 있다. 이는 제약조건을 만족하기 위해서는 필수적으로 하나의 행 당 하나의 Queen이 반드시 위치해야 하기 때문이다. 그리고 첫 번째 제약조건은 *N*개의 변수값은 모두 다르다는 것이며 이를 통해 동일한 열에 위치하는 쌍이 존재하지 않음을 보장할 수 있다. 두 번째 제약조건은 *i*번째 행의 변수값이 *a*이고 j번째 행의 변수값이 *b*일 경우 $(i-j) \neq (a-b)$이고 $(i-j) \neq (b-a)$이어야 한다는 것으로서 이를 통해 대각선 방향으로 위치하는 쌍이 존재하지 않음을 보장할 수 있다.

실험을 위해 사용된 가중치가 부여된 *N*-Queens 문제의 데이터는 표 1과 같다. 100-Queens로부터 5000-Queens에 이르기까지 다양한 규모의 문제를 사용하였으며 각 위치에 대한 가중치 값으로는 1부터 10까지의 값을 무작위로 부여하였다.

표 1. 실험 데이터
Table 1. Experimental Data

특징 / data	Queen의 개수	특징 / data	Queen의 개수
100-Q	100	2000-Q	2000
500-Q	500	3000-Q	3000
1000-Q	1000	5000-Q	5000

2. 실험 결과

실험은 Pentium D 3.4GHz, 2G RAM PC 상에서 수행되었으며, 모든 실험에 있어서 각 실험 당 수행 시간은 1시간으로 제한하였다. 1시간 정도면 본 논문에서 제시한 단순 언덕오르기 탐색과 제약 프로그래밍의 결합 기법 및 비교 대상 기법들의 특성을 파악하기에 충분한 시간으로 판단하였다.

먼저 본 논문에서 제시한 기법인 단순 언덕오르기 탐색과 제약 프로그래밍을 결합한 기법(SHC+CP)에 있어서 유일한 파라미터인 이웃해 생성 시 교체되는 변수의 개수인 k 값에 대한 영향을 살펴보았다. 실험 결과는 표 2와 같으며 모든 수치는 각 데이터 및 k 값 별로 5회 실험 후 평균을 취한 값이다.

표 2. SHC+CP의 *k*에 대한 실험 결과
Table 2. Experimental Results for *k* of SHC+CP

k / data	100-Q	500-Q	1000-Q	2000-Q	3000-Q	5000-Q
5	897	3830	5886	11053	16559	27343
10	960	4649	8412	12324	17049	27525
15	974	4813	9180	14668	18481	27886
20	983	4892	9463	15877	19962	28562
25	**991**	4933	**9616**	**16151**	**20434**	**28928**
30	990	**4947**	9605	15205	19512	28813
35	833	3315	5802	12170	17319	27873
50	679	2854	5410	10981	16601	27343

표 2에 의하면 모든 데이터에 있어서 유사한 경향을 보이고 있다. k가 25, 30, 20일 때 성능이 가장 뛰어나며 이를 기준으로 작으면 작을수록 그리고 크면 클수록 성능이 급격히 저하됨을 알 수 있다. k의 값이 너무 작으면 지역 최적해에 빠져 더 좋은 해를 도출할 기회가 줄어드는 것으로 판단된다. 그렇다고 무작정 k의 값이 커진다고 좋은 것은 아니다. k의 값이 너무 크게 되면 더 좋은 해를 찾을 기회도 늘어나지만 좋지 않은 해도 그만큼 많아지게 된다. 따라서 이웃해 생성 시 추가된 제약조건인 현재해에 비해 같거나 좋은 해를 찾는 것 자체가 제약 프로그래밍에 있어서 큰 부담으로 작용하기 때문에 단위 시간 당 반복 횟수가 매우 적어지게 되며 이에 따라 해의 개선 역시 매우 느려지게 된다. 이와 같은 경향은 문제 규모와는 크게 무관한 것으로 보인다. 물론 대상 문제에 따라 적절한 k 값은 달라질 수 있지만 최소한 본 논문의 대상 문제에 있어서는 문제 규모에 관계없이 20 내지 30의 값이 적절한 것으로 판단된다.

두 번째 실험에서는 본 논문에서 제시한 기법(SHC+CP)

과 시뮬레이티드 어닐링과 제약 프로그래밍을 결합한 기법(SA+CP)의 성능을 비교하기 위해 시간에 따라 그 때까지 도출된 가장 좋은 해의 변화 추세를 살펴보았다. 그림 5는 각 기법에 의한 시간대 별 가장 좋은 해의 전형적인 변화 추이를 나타낸 것이다. 각 수치는 {(*Obj*(SHC+CP) / *Obj*(SA+CP)) × 100}의 값으로서 이 값이 100보다 크다면 SHC+CP의 결과가 더 좋음을 의미하며 100보다 작다면 SA+CP의 결과가 더 좋음을 의미한다. 각 기법 별로 k의 값은 25로 동일하게 설정하였다. SA+CP의 경우 현재해보다 좋지 않은 해로의 이동이 가능하므로 SHC+CP와는 달리 이웃해 생성 시 "*Obj* 〉= *CurObj*"에 대한 제약조건이 필요없다. 그러나 이 제약조건을 포함하지 않을 경우 해의 개선이 매우 어려움을 확인하였으며 따라서 해당 제약조건을 "*Obj* 〉= *CurObj* - 2"와 같이 설정함으로써 해의 개선이 가능하면서도 현재해보다 좋지 않은 해가 도출될 수 있도록 하였다. 온도 감소 방법으로는 현재 온도에 0.999를 곱하는 방식을 사용하였다.

그림 5에서 보는 바와 같이 SHC+CP와 SA+CP 모두 제약 프로그래밍을 통해 동일한 초기해를 생성하므로 출발점은 100으로 동일하다. 그러나 이후 모든 데이터와 모든 시간 대에 있어서 100을 초과하는 값을 보이고 있으며 이로부터 SHC+CP가 SA+CP보다 우수하다는 결론을 내릴 수 있다. 이는 여러 개의 변수를 한꺼번에 변경하는 그 자체로써 다양성을 확보할 수 있기 때문에 SA+CP와 같이 다양한 탐색을 모색하기 위한 좋지 않은 해로의 이동은 불필요한 것으로 보인다.

그러나 데이터 별 변화 추이에 있어서는 다소 다른 결과를 보이고 있는데 크게 두 가지로 분류할 수 있다. 첫 번째는 100-Q, 500-Q, 1000-Q와 같이 상대적으로 규모가 작은 데이터로서 초반에 큰 차이를 보이다가 점점 그 격차가 줄어들고 있으며, 두 번째는 2000-Q, 3000-Q, 5000-Q와 같이 상대적으로 규모가 큰 데이터로서 시간이 갈수록 점점 더 격차가 벌어지고 있다. k의 크기가 동일하다 하더라도 규모가 큰 데이터가 규모가 작은 데이터보다는 제약조건 만족 여부를 위한 점검 시간 등에 있어서 더 많은 시간을 요구하기 때문에 다음 해로 한 번 이동하는 데 더 많은 시간이 소요되고 결과적으로 단위 시간 당 이동하는 횟수가 적어지게 된다. 따라서 데이터 규모가 작은 경우에는 탐색 초반에 많은 이동이 가능하므로 그때 발생하는 SA+CP의 불필요한 이동에 의한 격차가 더 크게 벌어지게 되고 탐색 후반에는 시뮬레이티드 어닐링의 특성 상 불필요한 해로의 이동이 적어지기 때문에 그나마 격차가 줄어들게 되는 것으로 보인다. 반면에 데이터 규모가 큰 경우에는 1시간을 수행했다 하더라도 이동 횟수가 적기 때문에 여전히 SA+CP에 의한 불필요한 이동이 계속해서 발생하는 상태이고 이로 인해 그 격차 역시 지속적으로 벌어지게 된다. 즉, 이들 데이터에 있어서 1시간이란 시점이 규모가 작은 데이터의 탐색 초반에 해당한다고 볼 수 있다. 이에 대한 증거로 1시간 동안 각 데이터에 대한 이웃해로의 이동 횟수가 약 (100-Q : 34,000회), (500-Q : 30,000회), (1000-Q : 12,000회), (2000-Q : 2,600회), (3000-Q : 1,100회), (5000-Q : 350회)로 2000-Q부터 급격하게 줄어듦을 확인할 수 있었다.

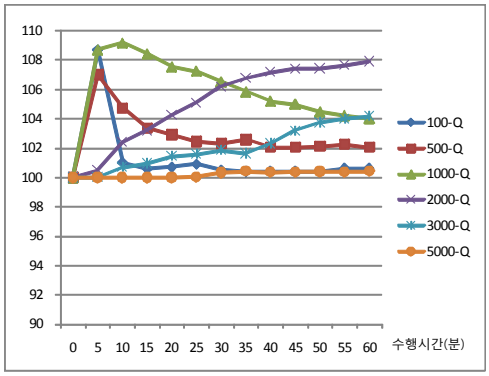


그림 5. SHC+CP와 SA+CP의 비교 실험 결과
Fig. 5. Experimental Results for Comparing SHC+CP with SA+CP

중요한 것은 앞서 설명한 바와 같이 본 실험을 통해 데이터의 규모나 시점에 관계없이 SHC+CP가 SA+CP보다 성능이 훨씬 우수하다는 결론을 내릴 수 있었다는 것이다.

표 3은 본 논문의 결론에 해당하는 실험 결과로서 제약 프로그래밍(CP), 시뮬레이티드 어닐링(SA) 그리고 본 논문에서 제안한 단순 언덕오르기 탐색과 제약 프로그래밍을 결합한 기법(SHC+CP)에 대한 비교 실험 결과를 나타낸 것이다. 모든 실험 결과는 각 기법 및 데이터 별로 5회 실험 후 평균값을 취한 것이다.

표 3. CP, SHC+CP, SA의 비교 실험 결과
Table 3. Experimental Results for Comparing CP, SHC+CP and SA

methods / data	CP	SHC+CP	SA
100-Q	727	**991**	987
500-Q	2961	**4933**	4898
1000-Q	5469	**9616**	Fail
2000-Q	11066	**16151**	Fail
3000-Q	16692	**20434**	Fail
5000-Q	27427	**28928**	Fail

SHC+CP의 경우 이웃해 생성 시 변경되는 변수의 개수인 k 값을 25로 설정했을 때의 결과를 제시한 것이다. SA의 경우에는 다양한 파라미터에 대한 결정이 필요하다. 첫 번째 파라미터인 온도 감소 방법으로는 현재 온도에 0.999를 곱하는 방법을 사용하였는데 온도 감소 방법이 전체적인 성능에 큰 영향을 미치지는 않았다. 두 번째로는 새로운 전체 목적함수(*TotalObj*)에 있어서 원래 목적함수인 가중치의 합(*Obj*)과 추가된 목적함수인 제약조건을 위배한 쌍들의 합(*Penalty*)에 대한 비율인 a 값을 결정해야 한다. a 값으로 0.1, 0.5, 1, 5, 10, 20, 30과 같이 다양한 값을 부여하여 실험을 실시해 보았다. a 값이 작을수록 원래 목적함수 값을 최대화하는 데 주력하게 되고 클수록 제약조건을 만족하기 위해 노력할 것이다. 그러나 a 값이 5인 경우를 제외하고는 주어진 시간 내에 제약조건을 만족하는 해 자체를 도출하는 것조차 어려웠다. 더군다나 a 값이 5인 경우라 할지라도 100-Q 또는 500-Q와 같이 상대적으로 규모가 작은 경우에만 가능하였다. 따라서 a 값은 5로 설정하여 실험을 실시하였다. 세 번째로는 이웃해 생성 시 값을 변경하는 변수의 개수(k)로서 1, 5, 10, 20과 같이 값을 달리하여 실험해 보았으나 1개를 변경하는 경우의 성능이 가장 뛰어났다. 이때 역시 문제 규모가 작은 경우에 한하여 제약조건을 만족하는 해의 도출이 가능하였으며 1000-Q 이상에 대해서는 해를 도출할 수 없었다. 결론적으로 표 3에 제시한 SA의 수치는 a 값은 5로, k 값은 1로 설정하여 실험한 결과이다.

표 3의 실험 결과에 의하면 문제 규모가 작은 경우에는 CP에 비해 SHC+CP와 SA의 성능이 월등히 뛰어남을 알 수 있으며, SA보다 SHC+CP가 다소 우세함을 알 수 있다. 문제 규모가 큰 경우에는 SHC+CP의 성능이 CP에 비해 월등히 좋은 반면에 SA는 제약조건을 모두 만족하는 실행가능한 해조차 도출하기 어려움을 알 수 있다. 이 실험을 통해 제약 만족 최적화 문제에 있어서 본 논문에서 제시한 방법의 우수성을 확인할 수 있으며, 특히 대규모 제약 만족 최적화 문제에 있어서 더욱 더 효과적임을 알 수 있다.

참고로 그림 6은 표 3에서 제시한 3가지 기법에 있어서 시간대 별로 전형적인 변화 추이를 나타낸 것으로서 500-Q 데이터에 대한 예이다. SA는 첫 번째 실행가능해가 도출되기까지 매우 많은 시간이 소요됨을 알 수 있는데 그만큼 제약조건을 모두 만족하는 해 자체를 도출하기가 매우 어렵다는 것을 의미한다. 그리고 모든 시간 대에 있어서 SHC+CP가 CP 또는 SA+CP보다 더 좋은 해를 도출할 수 있음을 알 수 있다. 따라서 실시간으로 보다 좋은 해를 요구하는 응용 문제에 있어서도 SHC+CP가 다른 기법에 비해 훨씬 효과적일 것으로 예상된다.

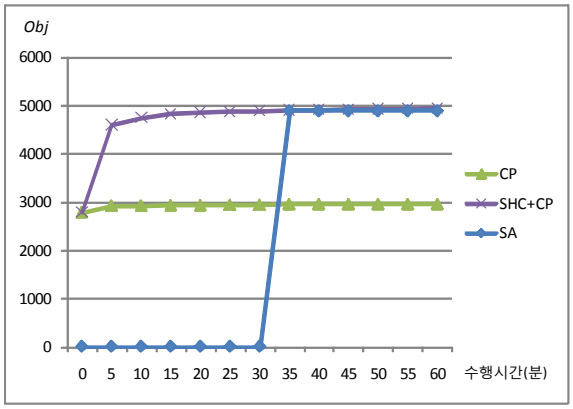


그림 6. CP, SHC+CP, SA의 탐색 추이
Fig. 6. Search Patterns of CP, SHC+CP, SA

V. 결론

본 논문에서는 제약 만족 최적화 문제를 해결하기 위한 지역 탐색과 제약 프로그래밍의 결합 방안으로서 단순하지만 매우 효과적인 방안을 제시하였다. 지역 탐색 기법들 중 단순 언덕오르기 탐색을 기반으로 하고, 초기해 생성과 이웃해 생성 시 제약 프로그래밍을 활용함으로써 제약조건을 모두 만족하는 해를 보다 쉽게 생성할 수 있도록 하였다. 또한 현재해와 같거나 더 좋은 해에 대한 요구 사항을 이웃해 생성을 위한 제약 프로그래밍의 제약조건으로 추가함으로써 보다 빨리 해를 개선할 수 있도록 하였다. 가중치가 부여된 N-Queens 문제에 대한 실험을 통해 본 논문에서 제시한 기법을 사용함으로써 제약 프로그래밍이나 지역 탐색을 단독으로 사용할 때보다 획기적으로 성능을 향상시킬 수 있음을 확인하였다.

향후 과제로는 다음 두 가지를 고려해 볼 수 있다. 첫 번째는 본 논문에서 제시한 기법 자체에 대한 보다 심도있는 연구로서 다른 최적화 알고리즘들과의 또 다른 형태의 결합을 시도하는 것이다. 물론 본 논문을 통해 시뮬레이티드 어닐링을 기반으로 할 때보다 본 논문에서 제시한 바와 같이 단순 언덕오르기 탐색을 기반으로 하는 것이 보다 성능이 뛰어남을 확인할 수 있었다. 그러나 시뮬레이티드 어닐링 외에도 유전 알고리즘[16] 등 또 다른 메타휴리스틱과의 결합이 가능하며 그 결과는 예측하기 힘들다. 이와 관련된 연구를 통해 새로운 형태의 더 좋은 알고리즘을 개발할 수도 있으며, 한편으로는 본 논문에서 제시한 방법의 우수성을 보다 확실히 입증할 수 있을 것이다. 두 번째는 실세계 문제로의 적용을 통해 본 논문에서 제시한 기법의 유용성을 입증하는 것이다. 대규모 일정계획 문제를 비롯한 다양한 실세계 문제들이 제약 만족 최적화 문제로

모델링될 수 있다. 본 논문에서 제시한 기법의 개념은 매우 단순하다. 따라서 대규모 일정계획 문제와 같은 복잡한 실세계 문제로의 적용이 비교적 쉬울 것으로 기대된다.

참고문헌

〔1〕 E. Tsang, "*Foundations of Constraint Satisfaction*," Academic Press Limited, pp.1-319, 1996.

〔2〕 C.R. Reeves, "*Modern Heuristic Techniques for Combinatorial Problems*," McGraw-Hill Book Company, pp.1-19, 1995.

〔3〕 B. De Backer, V. Furnon, and P. Shaw, "An Object Model for Meta-heuristic Search in Constraint Programming," Online Proceedings of CP-AI-OR'99, *http://www3.deis.unibo.it/Events/Deis/Workshops/cp-ai-or99.html*, February 1999.

〔4〕 Y. Pomerleau, S. Chamberland, and G. Pesant, "A Constraint Programming Approach for the Design Problem of Cellular Wireless Networks," Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering Canada, pp.881-884, May 2003.

〔5〕 김용환, 장용성, 유환주, "제약 프로그래밍과 메타휴리스틱을 활용한 차량 일정계획 시스템 개발에 관한 연구," 대한산업공학회/한국경영과학회 2002 춘계공동학술대회, 979-986쪽, 2002년 5월.

〔6〕 F. Glover, and M. Laguna, "*Tabu Search*," Kluwer Academic Publishers, pp.1-124, 1997.

〔7〕 R. Qu, and F. He, "A Hybrid Constraint Programming Approach for Nurse Rostering Problems," Applications and Innovations in Intelligent Systems XVI : Proceedings of AI-2008, pp.211-224, October 2008.

〔8〕 E.K. Burke, T. Curtois, G. Post, R. Qu, and B, Veltman, "A Hybrid Heuristic Ordering and Variable Neighbourhood Search for the Nurse Rostering Problem," European Journal of Operational Research, Vol. 188, No, 2, pp.330-341, April 2007.

〔9〕 M. Khichane, P. Albert, and C. Solnon, "Integration of aco In a Constraint Programming Language," Proceedings of the 6th International Conference on Ant Colony Optimization and Swarm Intelligence, pp.84-95, September 2008.

〔10〕 M. Dorigo, and C. Blum, "Ant Colony Optimization Theory: A Survey," Theoretical Computer Science, Vol. 344, No. 2-3, pp.243-278, Nov. 2005.

〔11〕 H.S. Fen, S. Deris, and S.Z.M. Hashim, "Incorporating of Constraint-Based Reasoning into Particle Swarm Optimization for University Timetabling Problem," Computer Science Letters, Vol. 1, No. 1, *http://www.issres.net*, June 2009.

〔12〕 S. Prestwich, "Generalized Graph Colouring by a Hybrid of Local Search and Constraint Programming," Discrete Applied Mathematics, Vol. 156, No. 2, pp.148-158, April 2007.

〔13〕 황준하, "비선형 최적화 문제의 해결을 위한 정수계획법과 이웃해 탐색 기법의 결합," 한국컴퓨터정보학회눈문지, 제 14권, 제 2호, 27-35쪽, 2009년 2월.

〔14〕 IBM ILOG Solver, "*User's Manual and Reference Manual*," Version 6.7, 2009.

〔15〕 S. Russell, and P. Norvig, "*Artificial Intelligence : A Modern Approach*," Prentice Hall, pp. 110-119, 2005.

〔16〕 강명주, "무향 Rural Postman Problem 해법을 위한 유전 알고리즘에서 그래프 변환에 의한 디코딩 알고리즘," 한국컴퓨터정보학회논문지, 제 12권, 제 2호, 2007년 2월.

저 자 소 개

황 준 하

2002년 : 부산대학교 컴퓨터공학과 공학박사

2002년~현재 : 금오공과대학교 컴퓨터공학부 교수

관심분야 : 인공지능, 최적화, 기계학습, 프로그래밍언어