Journal of the Korean Institute of Industrial Engineers
Vol. 31, No. 1, pp. 10-19, March 2005.

진화 알고리듬을 위한 새로운 트리 표현 방법

석상문† · 안병하

광주과학기술원 기전공학과

A New Tree Representation for Evolutionary Algorithms

Sang-Moon, Soak · Byung-Ha, Ahn

Department of Mechatronics, Gwangju Institute of Science and Technology, Gwangju, 500-712

The minimum spanning tree (MST) problem is one of the traditional optimization problems. Unlike the MST, the degree constrained minimum spanning tree (DCMST) of a graph cannot, in general, be found using a polynomial time algorithm. So, finding the DCMST of a graph is a well-known NP-hard problem of importance in communications network design, road network design and other network-related problems.

So, it seems to be natural to use evolutionary algorithms for solving DCMST. Especially, when applying an evolutionary algorithm to spanning tree problems, a representation and search operators should be considered simultaneously.

This paper introduces a new tree representation scheme and a genetic operator for solving combinatorial tree problem using evolutionary algorithms. We performed empirical comparisons with other tree representations on several test instances and could confirm that the proposed method is superior to other tree representations. Even it is superior to edge set representation which is known as the best algorithm.

***Keywords*:** evolutionary algorithms, spanning tree problem, degree constrained minimum spanning tree problem

1. 서 론

먼저, 방향성이 없는 완전하게 연결된 그래프(undirected compete graph)를 $G=(V, E)$라고 하자. 여기서, $V=\{1, 2, ..., N\}$는 N개의 노드(Node)들의 집합을 의미하며, $E=\{1, 2, ..., M\}$는 M개의 에지(Edge)들의 집합을 의미한다. 만약 그래프에 있는 모든 노드들을 연결하는 최소 길이를 가지는 트리(Tree)를 만드는 문제를 고려한다면 이 문제는 널리 알려진 최소걸침나무(minimum spanning tree; MST) 문제가 된다. 최소걸침나무문제의 경우 기존에 알려진Prim 알고리듬이나Kruskal 알고리듬을 이용하여 쉽게 해결이 가능하다. 하지만 이러한 최소걸침나무문제에 약간의 제약 사항을 부가하게 되면 그 제약 조건에 따라 다양한 문제들로 분류되는데(Gen and Cheng, 1997), 본 논문에서는 각 노드가 차수(degree)에 관한 제약을 가지는 차수 제약 최소걸침나무(Degree Constrained Minimum Spanning Tree; DCMST) 문제를 다룬다.

이러한 DCMST 문제는 통신 네트워크 설계, 도로망 설계, 수송 네트워크 설계 및 다양한 네트워크 관련 문제들에 적용이 가능한 NP-hard 문제로 알려져 있다(Garey and Johnson, 1979). 예를 들어 도로망을 설계할 경우 어떤 교차로가 가질 수 있는 도로의 수가 교통 정체 때문에 제약을 받는다든지 또는 네트워크 설계 시 어떤 서브의 용량제약 때문에 서비스 가능한 클라이언트의 수에 제약을 받는다든지 하는 등의 다양한 현실 문제들에서 응용이 가능하다.

DCMST 문제는 처음에 Narula and Ho(1980)가 3가지 polynomial time 알고리듬을 이용해서 해결한 이후 많은 다양한 해결 방법들이 소개되었는데 최근에 가장 두드러지는 접근법은 진화 알고리듬(Evolutionary algorithms; EAs)을 이용하는 방법

본 연구는 Brain Korea 21 사업의 지원으로 수행되었음.
† 연락저자 : 석상문, 500-712 광주광역시 북구 오룡동 1번지 광주과학기술원 기전공학과 시스템통합연구실, Fax : 062-970-2384,
E-mail : soakbong@gist.ac.kr
2004년 5월 접수; 2004년 9월 수정본 접수; 2004년 9월 게재 확정,

들이다. 이 문제의 복잡성을 고려해 본다면 진화 알고리듬을 이용해서 해결하려는 시도는 어쩌면 당연한 시도일지도 모른다.

진화 알고리듬을 어떤 문제에 적용하기 위해서 가장 기본적으로 고려되어야 하는 사항은 해를 표현하는 방법과 표현법에 맞는 새로운 연산자를 개발하는 일이다. 그 이유는 해를 표현하는 표현법 및 연산자에 의해서 알고리듬의 성능이 크게 좌우되기 때문이다.

따라서 본 논문에서는 새로운 트리 표현 방법(tree representation method)을 제안하고, 이를 이용해서 유전자형(genotype)을 표현형(phenotype) 으로 바꾸는 효율적인 decoding 방법을 소개한다. 그리고 제안한 표현법을 이용해서 효율적으로 우수한 해를 찾아내는 유전 연산자 또한 제안한다. 제안하는 유전 연산자는 부모 세대의 우수한 형질을 보존함과 동시에 네트워크 정보를 이용함으로써 보다 우수한 해를 찾아 낼 수 있었다.

본 논문의 구성은 다음과 같다. 제2장에서 기존의 트리 표현 방법들에 대해 간단하게 소개하고, 제3장에서는 제안하는 새로운 표현법과 트리 구성 룰(tree construction rule)에 대해 설명한다. 그리고 제4장에서는 본 논문에서 사용된 유전연산자들을 소개하고, 제5장에서는 다양한 실험 데이터를 통한 실험 결과를 보여준다. 그리고 마지막으로 제6장에서 결론을 맺는다.

2. 기존의 트리 표현 방법들

그 동안 많은 연구자들이 트리 네트워크 문제(tree network problem)를 진화 알고리듬을 이용해서 해결하기 위해 다양한 표현 방법에 대한 연구를 수행해 왔다.

Dengiz *et al*.(1997)은 에지 표현법(edge representation)을 사용하였는데, 이는 모든 가능한 에지들을 열거하고 그 중에서 N-1 개의 에지를 선택한다. 이러한 에지 기반 표현법들은 선택된 N-1개의 에지가 트리를 형성할 가능성이 매우 낮다는 단점을 지니고 있다. 따라서 유효한 트리를 만들어 내기 위한 repair 과정을 필요로 한다. 그리고 Piggott and Suraweera(1995)는 특성 벡터(characteristic vectors)를 이용해서 트리를 나타내었다. 이 또한 항상 트리를 표현하는 것이 아니기 때문에 repair를 위한 부가적인 계산이 필요하며 표현법이 해의 지역성(locality) 및 상속성(heritability) 과 같은 진화 알고리듬의 중요한 특성을 반영하지 못한다는 단점을 지니고 있다. Abuali *et al*.(1995)은 determinant coding을 제안하였는데, 이 표현법은 선행 벡터 표현법(predecessor vector representation)(Li, 2001; Raidl and Drexel, 2000) 과 아주 유사하며 이 또한 항상 유효한 트리를 만들어 내는 것이 아니기 때문에 repair 과정을 필요로 한다. 기존의 대부분의 repair 과정을 필요로 하는 방법들의 경우 repair 과정으로 인한 부모해가 지니고 있는 우수한 형질의 손실이나 임의 탐색(random search)과 같은 탐색의 방향성 결여로 좋은 결과를 나타내지는 못하였다.

이에 반해서 항상 유효한 트리를 만들어 내는 방법으로 가장 일반적인 트리 표현법은 Prüfer 수를 이용하는 방법이다. 이는 Cayley's 정리에 의해서 N 노드를 가진 완전하게 연결된 그래프 상에는 N^{N-2} 개의 가능한 트리가 존재하는데, Prüer 수는 이러한 모든 트리 표현이 가능하고, 트리를 나타내기 위해 단지 $N-2$개의 숫자만을 필요로 한다는 장점을 지니고 있기 때문이다(Gen and Cheng, 1997; Zhou and Gen, 1998; Krishnamoorthy *et. al*, 2001). 또한 이 방법은 일반적인 교차연산자(crossover operator)나 변이 연산자(mutation operator)를 사용하는 데 제약을 받지 않는다. 하지만 여러 논문들에서 지적한 바와 같이(Gottlie *et. al*, 2001; Raidl and Julstrom, 2003) Prüfer 수의 약간의 변화가 전혀 다른 트리를 형성한다거나(lack of locality) 네트워크가 지니고 있는 어떤 정보를 이용하기가 어렵다는 단점을 지니고 있다. 또한 해 표현이 Prüfer 수와 이의 보수(complement)를 유지하는 두 개의 배열로 분리되어 있기 때문에 효과적인 유전연산자의 개발이 아주 어렵다.

Rothlauf *et al*.(2002)과 Schindler *et al*.(2002) 등에 의해 소개된 Network Random Keys(NetKeys) 표현법은 기존의 TSP를 해결하기 위해서 Bean(1994)이 처음 소개한 Radom Key 방법을 트리 표현으로 확장한 방법이다. NetKey방법은 기존의 최소걸침나무문제를 해결하기 위한 방법인 Kruskal의 알고리듬과 매우 유사하다. 하지만 Kruskal의 알고리듬은 트리를 구성하기 위해 각 에지들의 비용 또는 거리를 이용하는 반면에 NetKey 방법에서는 랜덤하게 각 에지에 부여된 벡터 값을 이용한다. 하지만 NetKey 방법 역시 Kruskal의 방법과 똑같은 단점을 지니고 있다. 즉, 네트워크의 규모가 큰 경우 많은 계산 시간 및 메모리를 필요로 하는데, 예를 들어 100개의 노드를 가진 완전하게 연결된 네트워크(complete network)의 경우, N(N- 1)/2개의 에지를 가지기 때문에, NetKey 방법을 이용해서 각각의 해를 표현하기 위해서는 길이가 L = 100(100-1)/2 = 4950인 bit string을 필요로 한다. 그리고 길이가 L인 bit string을 정렬하기 위해서 매번 $O(L\log L)$ 의 계산 시간을 필요로 한다. 또한 유전연산자를 설계할 때 NetKey 표현법의 경우 부모세대가 지니고 있는 유전 정보나 네트워크 정보를 이용할 수가 없기 때문에 효율적인 연산자를 개발하기가 아주 어렵다. 그리고 이 방법의 경우 전체 N(N-1)/2개의 에지들 중에서 N-1개의 에지만을 사용하기 때문에 네트워크의 크기가 증가할수록 성능은 떨어진다.

항상 유효한 트리를 만들어 내는 또 다른 방법으로는 Raidl and Julstrom(2003)이 에지의 집합에 기반하여 개발한 Edge Set 표현법이 있다. Raidl 등은 이 표현법을 DCMST 문제에 적용하여 아주 우수한 결과를 보여주었다. 현재까지 나와 있는 알고리듬들 중에서 이 표현법을 이용한 진화 알고리듬이 DCMST를 해결하는 가장 우수한 방법으로 알려져 있다.

이 표현법은 트리를 구성하는 에지들을 이용해서 해를 표현하는 아주 직접적인 해 표현법이다. 따라서 이러한 형태를 가지는 초기해를 형성하기 위해 기존의 MST를 해결하는 알고리

듬인 Prim의 알고리듬과 Kruskal의 알고리듬에 차수제약을 추가하여 트리를 형성하는 에지들을 찾아내고 이 에지들을 모두 연결함으로써 초기해를 만들어낸 다음, 이렇게 만들어진 초기해에 여러 가지 연산자들을 적용하여 해를 진화시킨다. 기존 방법들과의 차이점은 모든 연산자들이 항상 에지 단위로 움직이게 된다는 것이다. 그리고 Raidl and Julstrom(2003)은 이 방법을 좀더 개선시키기 위해서 휴리스틱 초기해 생성자(heuristic initialization operator), 휴리스틱 에지 교차 연산자(heuristic edge crossover)와 휴리스틱 에지 삽입 변이 연산자(heuristic edge insertion mutation)와 같은 다양한 휴리스틱 연산자들을 제안하였다.

본 논문에서는 이 표현법을 성능 비교를 위한 벤치마크(benchmark) 알고리듬으로 사용하여 제안하는 방법의 우수성을 보여준다.

3. 제안하는 트리 표현 방법

주어진 어떤 문제에 대한 해를 찾기 위해서 진화 알고리듬을 개발할 때 가장 중요한 부분은 어떻게 해를 표현할 것인가 하는 encoding 방법을 결정하는 부분이다. 본 논문에서는 어떤 그래프로부터 유효한 에지들을 추출함으로써 트리를 생성하는 새로운 트리 표현법을 소개한다.

우선 <Figure 1>과 같은 걸침나무를 고려해 보자. 트리는 9개의 노드와 8개의 에지를 가지고 있다. 즉, 모든 트리는 정확하게 N-1개의 에지로 구성된다.

이 걸침나무는 에지의 집합, *EdgeSet*{(1, 2),(1, 3),(1, 4),(1, 5),(1, 6),(1, 7),(1, 8),(1, 9)}로 구성된다. 이 에지들을 모두 차례대로 연결하면(1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9)과 같은 string이 된다. 즉, 이러한 형태의string을 형성할 수 있다면, 그

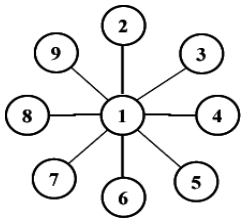
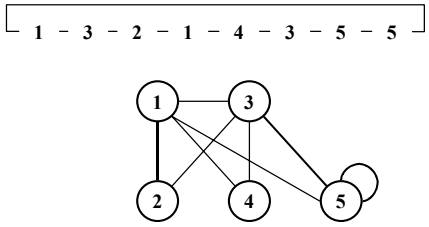


Figure 1. Star Tree.

리고 이를 통해서 합당한 에지들을 추출할 수 있는 방법이 있다면 <Figure 1>과 같은 트리를 형성할 수 있음을 의미한다.



Graph

Figure 2. Generated Graph.

예를 들어, $(N-1)\times 2$의 길이를 가지는 string을 만들어 보자. 여기서 N은 5이고 차수제약(d)은 3이라고 가정한다. 단, 모든 노드들은 적어도 한 번씩은 나타나야 하며, 최대(d-1) 번까지 나타날 수 있다. 생성된 string은 (1, 3, 2, 1, 4, 3, 5, 5)이고 이를 차례대로 연결하면 <Figure 2>와 같은 그래프가 된다. 그럼 그래프로부터 어떻게 트리를 만들어 낼 수 있을까? 다음으로 decoding 방법을 설명한다.

앞에서 언급한 것처럼, 만약 우리가 decoding 과정에서 효과적으로 유효한 에지들을 그래프로부터 추출해 낼 수 있다면,

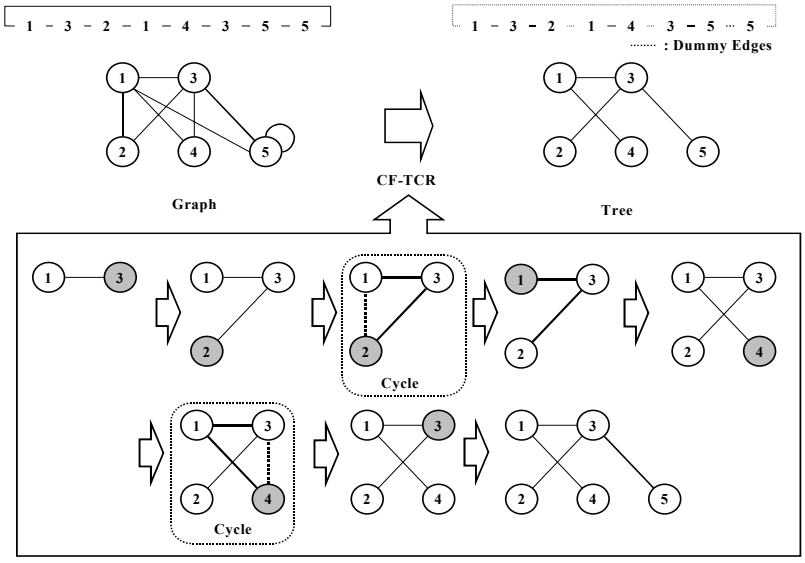


Figure. 3 Cycle Free TCR.

항상 유효한 트리를 만들어 낼 수 있을 것이다. 이를 위해 본 논문에서는 2가지 트리 구성 룰(Tree Construction Rule; TCR)을 소개한다.

첫 번째 방법은 사이클(cycle)을 형성하지 않는 방식으로 트리를 구성하는 Cycle-Free TCR(CF-TCR)이다. 이 방법은 Soak 등이 그들의 논문(Soak *et. al*, 2004)에서 제안한 방법으로 우선, string의 가장 왼쪽에서부터 i번째 gene과 i+1번째 gene을 고려한다. 이 경우 i번째 gene은 첫 번째 노드를 제외하고 항상 이전에 나왔던 노드이다. 따라서 i+1번째 gene이 이전에 선택되었는지를 확인하고 선택된 적이 없으면 i번째 gene과 i+1번째 gene을 연결하여 에지를 형성하는 방식으로 트리를 구성한다. 이 방법을 설명하는 그림은 <Figure 3>과 같다. <Figure 3>에서 색깔을 가진 노드는 다음에 에지로 연결되기 위해 기다리는 노드를 의미한다. 그리고 <Figure 4>는 CF-TCR의 pseu- docode를 나타낸다. 여기서 $SN\{\cdot\}$과 $EN\{\cdot\}$는 각각 선택된 노드의 집합과 선택된 에지의 집합을 나타내고, $E_i(x, y)$는 i 번째로 선택된 에지 (x, y)를 의미한다.

```
Begin
  Set d(degree), SN{ · }←φ, ES{ · }←φ;
  i←1 and k←1;
  SN{ · }←gene_i;
  while i ≠ N−1 do
    if gene_{k+1} ∉ SN{ · } then
      SN{ · }←gene_{k+1};
      ES{ · }←E_i(gene_k, gene_{k+1});
      i←i+1;
    k←k+1;
  end
end
```

Figure 4. Pseudocode of CF-TCR.

CF-TCR의 경우 상대적으로 빨리 선택되는 에지에 의해서 뒤쪽에 있는 에지들이 종속되기 때문에 뒤쪽에 나와 있는 유전인자가 우수한 형질일지라도 고려가 되지 않는다는 단점을 지니고 있었다. 이는 CF-TCR이 사이클(cycle)을 회피하는 방법으로 트리(tree)를 형성하기 때문인데, 이를 개선하기 위해서 새로운 TCR을 제안한다.

제안하는 방법은 첫 번째 CF-TCR을 확장한 방법으로, 각 에지를 체크하는 과정은 CF-TCR과 같다. 하지만 에지를 형성하는 과정 중에 사이클(cycle)을 만들게 되면 CF-TCR의 경우 이를 회피하고 다음 에지를 고려하지만, 제안하는 방법의 경우 그 사이클을 찾아서 이를 형성하는 각 에지의 거리를 비교하고 가장 긴 에지를 제거하는 방식으로 트리를 형성한다. 만약 에지들의 거리값이 같을 경우에는 임의로 하나의 에지를 선택한다.

여기서 새롭게 제안하는 방법을 Cycle-Breaking TCR(CB-TCR)이라고 부른다. CB-TCR을 설명하는 그림은 <Figure 5>와 같고 pseudocode는 <Figure 6>과 같다. <Figure 6>에서 $C(n)$은 노드 n이 선택된 횟수를 나타낸다.

제안하는 encoding 방법은 기존의 *d*-based 표현법(Soak *et al*. 2004)이 비록 같은 크기의 네트워크일지라도 차수제약에 따라 해의 길이가 변하는 $N\times(d-1)$ 길이의 해 표현법을 사용한 반면에 제안하는 방법은 이를 개선하여 차수 제약에 상관없이 항상 해의 길이가 $(N-1)\times 2$로 일정한 표현법으로 기존의 방법에 비해 저장 공간(memory)의 크기 및 계산 시간(computation time) 측면에서 상당한 개선을 줄 수 있다.

또한 설명한 encoding 방법과 decoding 방법(CF-TCR과 CB-TCR)을 이용하면 모든 트리 네트워크 문제에 간단히 적용이 가능할 뿐만 아니라 모든 가능한 트리를 표현할 수 있다. 그리고 본 논문에서 제안하는 방법은 각각의 노드가 chromosome 내에서 $1 \le C(n) \le d-1$ 로 나타나야 한다는 조건을 만족시킬 경우 항상 유효한 트리를 만들어 낼 수 있는데, 이는 각 연산자를 적용할 때 항상 $C(n)$값을 고려하기 때문에 간단하게 해결

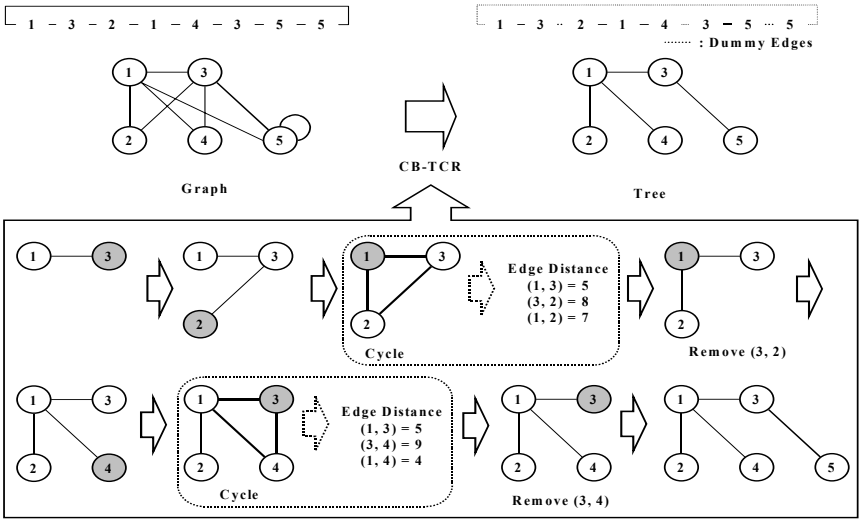


Figure 5. Cycle Breaking TCR.

이 가능하다. 또한 encoding 과정과 decoding 과정이 기존의 Prüer 수처럼 분리되어 이루어지는 것이 아니라 하나의 해 표현으로 가능하기 때문에 부모가 지닌 정보를 이용한 연산자 개발이 용이하다.

4. 유전연산자(Genetic Operators)

4.1 선택연산자(Selection Strategy)

본 논문에서는 Soak *et al.*(2004)이 소개한 토너먼트 선택전략인 Real World Tournament Selection(RWTS) 전략을 개선한 Improved Real World Tournament Selection(IRWTS) 전략을 이용하였다. 이미 그들의 논문에서 다양한 규모의DCMST 문제들의 실험을 통해 일반적인 binary 토너먼트 선택전략보다 RWTS가 우수함을 보여주었는데, 이는 널리 알려진 대로 binary 토너먼트의 경우 항상 해집단 내에서 임의의k개의 해(토너먼트 크기)를 뽑아 그들 중 가장 우수한 해를 다음 해집단에 포함시키게 되는데, 이 경우 이전 해집단에서 가장 우수했던 해뿐만 아니라 우수한 해들조차도 다음 해집단에 포함되지 않을 가능성이 있다.

따라서 이를 방지하기 위해 대개 엘리트 선택전략(Elitism)을 같이 사용하였다. 하지만 RWTS의 경우 해집단 내에 있는 모든 해들을 다 고려하기 때문에 binary 토너먼트의 이러한 단점을 극복할 수 있을 뿐만 아니라 선택전략 자체가 엘리트 선택전략의 특징을 함께 지니고 있어 보다 우수한 결과를 보여주었다. 또한 계산 시간(computa- tion time)뿐만 아니라 저장공간(memory)에서도 binary 토너먼트의 경우 다음 해집단을 형성하기 위한 중간 해집단(경쟁에서 승리한 해들을 저장)을 형성해야만 하지만 RWTS의 경우 경쟁에서 선택된 해들을 이전의 해집단에 그대로 차례대로 덮어쓰기 때문에 binary 토너먼트보다 더 적은 계산 기간과 저장 공간을 필요로 했다. 따라서 제안하는 IRWTS 또한 RWTS가 가지고 있는 이러한 장점들을 모두 가진다.

우선 RWTS에 관해서 간단히 설명한다. RWTS는 현실 세계의 토너먼트 경쟁을 응용한 전략으로 일반적으로 토너먼트 경쟁은 각 참가자들을 쌍으로 묶은 뒤 각각의 쌍이 경쟁자들과 경쟁을 해서 이긴 선수가 다음 토너먼트를 위한 후보 선수가 된다. 그리고 이 후보 선수들이 다시 쌍을 이루고 똑같은 경쟁을 반복 수행해서 결국 하나의 승자가 남을 때까지 반복한다. RWTS 또한 똑같은 방법으로 경쟁을 하지만 종료 조건은 하나의 승자가 남을 때까지가 아니라 population 크기만큼 해들이 모여졌을 때 종료한다는 것이다.

RWTS의 경우 <Figure 7>의 위쪽 그림과 같이 초기 토너먼트 경쟁을 하고 난 이후 맨 마지막의 승자는 항상 부전승으로 다음 단계의 경쟁에 참여를 하게 된다. 이를 개선하기 위해 본 논문에서는 홀수 개의 후보가 남아 있을 경우 맨 마지막 후보는 임의로 population 내에서 하나의 후보를 선택한 후, 이 후보와 경쟁을 하도록 하였다. 비록 제안하는 방법이 아주 현저한 해의 개선을 이끌어 내지는 못할지라도 해집단 내의 해의 다양성을 개선할 수 있다. <Figure 7>의 오른쪽 그림은 IRWTS를 보여준다.

```
Begin
  Set i←1, k←1, NS{ · } ← ∅, ES{ · } ← ∅ and C(n)←0, n ∈ V;
  NS( · )←gene_k and C(gene_k)←C(gene_k) + 1;
  while i ≠ N − 1 do
    if gene_{k+1} ∉ NS{ · } then
      NS{ · }←gene_{k+1};
      ES{ · }←E_i(gene_k, gene_{k+1});
      C(gene_k)←C(gene_k) + 1 and C(gene_{k+1})←C(gene_{k+1}) + 1;
      i←i + 1;
    else
      Find cycle;  //E(gene_k, gene_{k+1}) results in a cycle.
      Compare cost between edges in cycle and Tie is broken randomly;
      Select an edge, E(p, q), which C(p) and C(q) > 1 and has the highest cost;
      NS{ · }←gene_{k+1};
      E(p, q)←E(gene_k, gene_{k+1});
      C(p)←C(p) − 1 and C(q)←C(q) − 1;
      C(gene_k)←C(gene_k) + 1 and C(gene_{k+1})←C(gene_{k+1}) + 1;
      i←i + 1;
    k←k + 1;
  end
end
```

Figure 6. Pseudocode of CB-TCR.

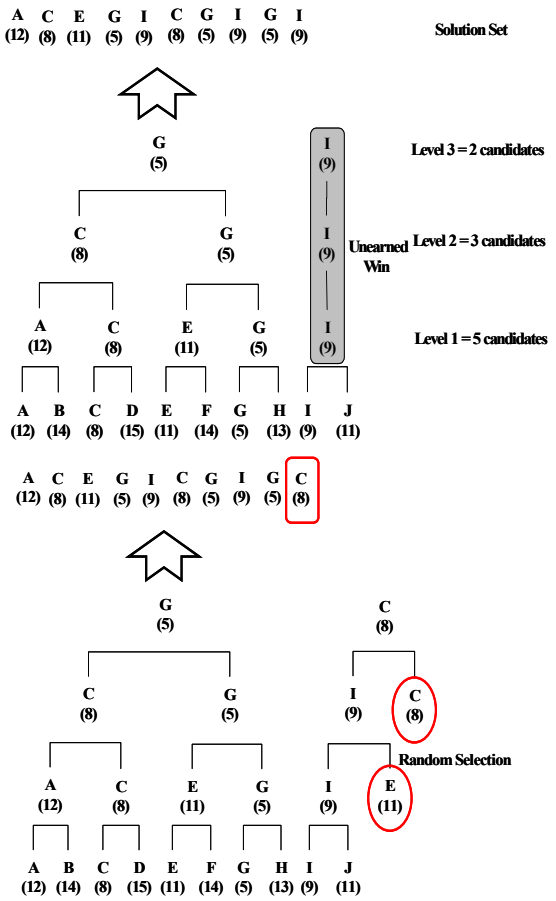


Figure 7. RWTS vs. IRWTS.

4.2 교차연산자(Crossover Operator)

본 논문은 새로운 트리 표현법과 함께 새로운 교차연산자를 제안한다. 제안하는 연산자는 부모가 지니고 있는 공통된 정보를 이용한다는 의미에서 Common Gene Preservation Crossover(CGPX)라 부른다. 일반적으로 부모의 정보를 이용하는 교차연산자들이 단순한 치환에 기반한 연산자들에 비해서 훨씬 더 우수한 결과를 준다고 알려져 있다. 따라서 제안하는 CGPX 역시 부모의 유전 정보의 상속성(inheritability)에 초점을 맞춘다. 그리고 또한 부모 해의 공통 정보뿐만 아니라 각각의 에지가 가지고 있는 에지 정보도 이용함으로써 보다 우수한해를 용이하게 찾아갈 수 가 있다. <Figure 8>은 CGPX의 pseudo-code를 나타내는데 여기서, P_i^1 와 P_i^2는 각각 부모 1과 부모 2의 i번째 gene을 나타내고, $f[min\{cost(a,b), cost(a,c)\}]$는 에지 (a,b)와 (a,c)의 비용을 비교해서 노드 b, c 둘 중에서 작은 비용을 형성하는 노드를 반환하는 함수이다.

CGPX를 간단하게 설명하면 다음과 같다. 자손2를 형성하는 과정은 자손1을 형성하는 과정과 첫 번째 gene을 형성하는 과정을 제외하고 모든 것이 동일하기 때문에 여기서는 자손1을 형성하는 과정을 중심으로 설명한다. P_i^k 와 O_i^k 는 각각 k 번째 chromosome에 있는 i 번째 gene을 의미한다. 그리고 자손 1을 생성하기 위해 O_1^1에 P_1^1의 값을 넣는다. 만약 P_i^1과 P_i^2가 같고 $C(P_i^1) < d-1$이면 O_i^1에 P_i^1을 넣고 그렇지 않을 경우에는 $cost(O_{i-1}^1, P_i^1)$과 $cost(O_{i-1}^1, P_i^2)$의 비용값을 비교하고 작은 값을 형성하는 부모의 gene을 반환하는 함수

```
Begin
  Set i←1;
  O_i^1 ← P_i^1 and C(O_i^1)←C(O_i^1)+1;
  while i < L do  // L is the length of chromosome.
    i←i+1;
    if (P_i^1 == P_i^2) and (C(P_i^1) < d−1) then
      O_i^1←P_i^1 and C(O_i^1)←C(O_i^1)+1;
    else
      P_i^1 or P_i^2 ← f[min{cost(O_{i−1}^1, P_i^1), cost(O_{i−1}^1, P_i^2)}];
      if C(P_i^1) < d−1 then //assume that P_i^1 is returned.
        O_i^1←P_i^1, C(O_i^1)←C(O_i^1)+1;
      else
        if C(P_i^2) < d−1 then
          O_i^1←P_i^2, C(O_i^1)←C(O_i^1)+1;
        else
          Select a node n among C(n) < d−1 randomly;
          O_i^1←n, C(O_i^1)←C(O_i^1)+1;
  end
end
```

Figure 8. Pseudocode of CGPX.

$f[min\{A, B\}]$를 이용해서 해당 gene을 찾는다. 여기서는 부모 1의 gene 값, 즉 P_i^1가 반환되었다고 가정한다. 다음 반환된 P_i^1가 $C(P_i^1) < d-1$이면 이 P_i^1을 자손 O_i^1에 넣고 그렇지 않을 경우에는 P_i^2 값을 고려한다. 만약 $C(P_i^2) < d-1$이면 P_i^2을 자손 O_i^1에 넣는다. 만약 이 또한 만족하지 않을 경우에는 $C(n) < d-1$을 만족하는 n값을 임으로 선택해서 O_i^1에 넣는다. 이러한 과정을 chromosome 내에 있는 모든 gene들이 다 고려되었을 때까지 수행한다.

제안하는 연산자를 어떤 해에 적용한 이후 모든 새로운 해들이 유효한 트리를 만들어 낼 수 있다고 보증을 하지 못한다. 예를 들어, 아래 <Figure 9>는 차수 제약이 3인 경우인데, 이 경우 자손1은 CGPX 적용 후 유효한 해를 만들어 내었지만 자손2의 경우는 가장 마지막 gene에 임의로 선택되는 노드에 따라 유효한 트리를 만들어 낼 수도 있고 그렇지 않을 수 도 있다. 따라서 Repair 과정이 필요하게 되는데, 본 논문에서는 단순한 repair 과정을 이용하였다.

항상 교차연산자를 적용하는 과정에서 각 노드가 선택된 횟수를 확인하기 때문에 만약 교차연산자 적용 후 한 번도 선택되지 않은 노드가 존재할 경우 임의로 한 지점에 있는gene을 선택하고 그 gene에 있는 노드가 선택된 횟수가 1보다 크면 $(C(n) > 1)$ 이 노드를 한 번도 선택되지 않은 노드로 대체한다. 이러한 방법을 이용해서 항상 유효한 트리를 만들어 낼 수 있도록 chromosome을 수정한다.

4.3 변이연산자(Mutation Operator)

본 논문에서는 다양한 연산자들을 이용한 선행 실험에서 우수한 결과를 보여 준 상호 교환 변이연산자(reciprocal exchange)를 이용하였다. 상호 교환 변이연산자는 우선 임의로 두 지점을 선택한 후 두 지점에 있는 노드를 상호 교환하는 방법이다.

5. 실 험

제5장에서는 기존의 알고리듬들과의 비교 실험을 수행한 결과를 보인다. 실험을 위해 2가지 실험 데이터를 이용하였다.

첫 번째 방법은 Zhou and Gen(1998)이 제안한 방법으로, 이 방법은 [10, 100] 사이의 값으로 각 에지에 임의의 거리값을 할당하여 모든 노드들이 서로 연결된 완전한 네트워크를 구성한다. 그리고 두 번째 방법은 첫 번째 데이터들 보다 더 복잡한 구조를 가지는 문제들로 Krishnamoorthy *et al.*(2001)이 제안한 structured hard problem (SHRD)을 만드는 방법을 이용해서 만들어 내었다. 이 방법은 우선 첫 번째 노드에서 모든 노드로 l +rand[1, 18]의 값을 가지는 에지들을 형성하고 두 번째 노드는 첫 번째 노드를 제외한 모든 노드를 $2l$ + rand[1, 18]의 길이를 가지는 에지들로 연결한다. 이렇게 모든 에지에 값을 할당하여 완전히 연결된 네트워크(complete network)를 구성한다. 여기서 rand[1, 18]는 1부터 18 사이의 임의의 값을 만들어 내는 랜덤 함수를 의미한다. 첫 번째 실험 데이터의 경우 Soak *et al.*(2004)이 사용한 실험 데이터와 동일한 데이터를 이용해서 실험을 하였다. 그리고 논문에 실린 결과를 인용해서 기존의 알고리듬과 비교함으로써 제안하는 알고리듬의 우수성을 보인다. 그리고 두 번째 실험 데이터를 이용해서 기존에DCMST를 해결하는 가장 우수한 알고리듬이라고 알려진Edge Set 표현법과의 비교 실험을 수행하였다.

각 노드의 차수(degree) 제약은 Zhou and Gen(1998)의 방법의 경우 Soak *et al.*(2004)과 같이 논문에서처럼 3으로 제한하였고, Krishnamoorthy의 방법으로 구성된 문제들의 경우는3에서 5까지로 제한을 하였다. 또한 각 알고리듬은 Visual C++를 이용해서 프로그램되었고 Pentium IV 2.2 Ghz 컴퓨터 상에서 실험을 하였다.

유전 알고리듬에 사용된 파라미터로 population 크기는 100을 사용하였고, 교차확률은 제안하는 방법의 경우 0.6을 Edge Set 표현법의 경우 Raidl and Julstorm(2003)가 제안하는 0.8을 사용하였다. 그리고 돌연변이확률의 경우는 제안하는 방법이 0.6을 Edge Set 표현법의 경우 논문에서 제안하는0.8로 설정하였고, 종료 조건은 두 방법 모두 generation이 10,000이 될 때까지로 설정하였다. 그리고 선택 전략으로는 제안하는 방법의 경우 Improved real world 토너먼트 선택 전략을, Edge Set 표현법의 경우(Raidl and Julstorm, 2003)에서 처럼 binary 토너먼트 선택 전략(토너먼트 크기 2)을 사용하였다. 각 실험은 10번씩 수행하였다.

<Table 1>은 Zhou and Gen(1998)의 방법에 의해 만들어진 실

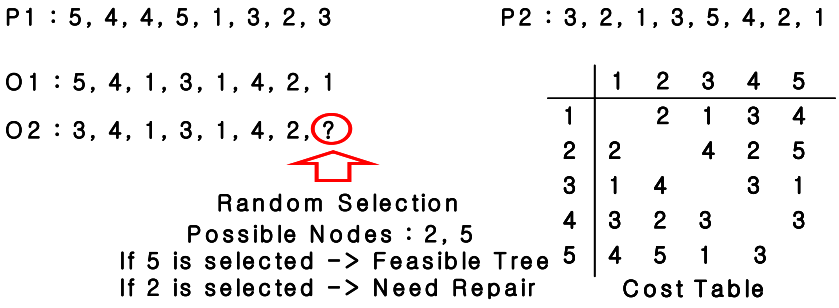


Figure 9. The example requiring repair process.

험 데이터에서 수행된 기존 실험 결과와의 비교 결과를 보여준다. CF-TCR과 CB-TCR의 경우 제안하는 해 표현법 상에서 수행된 결과이며, *d*-based 표현법(CF-TCR 사용)과 NetKey 표현법의 결과는 Soak *et al.*(2004)에서 인용하였다.

실험 결과를 통해 알 수 있듯이 모든 경우에서 CB-TCR을 이용하는 방법이 기존의 *d*-based 표현법이나 NetKey 표현법보다 우수한 결과를 보여주었다. 비록 제안하는 해 표현법에 CF-TCR을 이용하는 방법의 경우가 기존의 *d*-based 표현법에 CF-TCR을 이용하는 경우보다는 좋지 않은 결과를 보였지만 NetKey 표현법보다는 우수한 결과를 보여주었다. 그리고 CB-TCR을 이용하는 방법이 기존의 방법들을 개선하는 정도는 *d*-based 표현법의 경우 0~6.30%였고, NetKey 표현법의 경우 0~25.97%였다. <Figure 10>은 각각의 네트워크 크기에 따른 알고리듬의 개선 정도를 보여주는 그래프이다. 네트워크의 크기가 커질수록 개선의 정도(%)가 커짐을 확인할 수 있다. 본 실험에서는 제안하는 방법과 기존의 방법과의 계산 시간 비교는 수행하지 못하였는데, 이는 인용한 논문에서 사용한 시스템과 본 논문에서 사용한 시스템의 차이 때문이었다. 하지만 간단하게 동일한 조건에서 실험된 제안하는 해 표현법에 CF-TCR을 사용한 결과와 *d*-based 표현법에 CF-TCR을 사용한 결과를 비교해 본다면 제3장에서 언급한 것처럼 제안하는 해 표현법의 해의 길이가 $(N-1)\times 2$이고 *d*-based 표현법의 해의 길이가 $N\times(d-1)$이기 때문에 제안하는 방법이 d가 2인 경우를 제외하고 항상 더 적은 계산 시간을 필요로 할 것이라는 것은 분명하다.

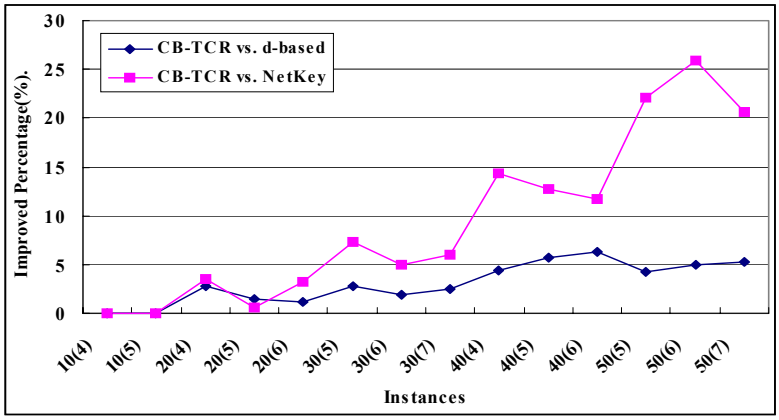


Figure 10. The improved percentage at all instances.

Table 1. The experimental results of the test instances generated through Zhou and Gen' method

N	Max-*d*	New Encoding		*d*-based (With CF-TCR)	NetKey
		CF-TCR	CB-TCR		
10	4	**249**	**249**	**249**(0)*	**249**(0)**
	5	**186**	**186**	**186**(0)*	**186**(0)**
20	4	351.5	**342.9**	352.6(2.83)*	354.9(3.50)**
	5	322.1	**316.4**	321.2(1.52)*	318.2(0.57)**
	6	315.1	**305.3**	308.9(1.18)*	315.2(3.24)**
30	5	392.8	**375.2**	385.5(2.75)*	402.9(7.38)**
	6	412.5	**399.6**	407(1.85)*	419.5(4.98)**
	7	415.2	**397.3**	407.1(2.47)*	421.3(6.04)**
40	4	562.8	**528**	551.2(4.39)*	603.8(14.36)**
	5	544.3	**513.2**	542.6(5.73)*	578.3(12.69)**
	6	545	**511.5**	543.7(6.30)*	571.7(11.77)**
50	5	650.3	**609.7**	635.5(4.23)*	744.7(22.14)**
	6	612.1	**576**	604.6(4.97)*	725.6(25.97)**
	7	624.4	**591.5**	622.3(5.21)*	713.9(20.69)**

Max-*d* : Maximum degree. In case of *d*-based and NetKey, the results of(Soak et. al) were referred to.
* and ** indicate the degree of which CB-TCR improves d-based and NetKey respectively.

<Table 2>는 SHRD 실험 데이터를 이용하여 제안하는 방법과 기존에 DCMST를 위한 가장 우수한 알고리듬으로 알려져 있는 Edge Set 표현법과의 비교 실험한 결과를 보여준다. 여기서, *d*는 차수(degree) 제약을 나타내고, Min.과 Max 그리고 Avg는 각 알고리듬을 이용해서 찾아낸 각 실험 데이터에서의 최소값, 최대값 그리고 평균값을 나타내며, CPU는 CPU time(초)을 나타낸다. 이 실험에서 비록 CB-TCR이 더 많은 계산 시간을 필요로 하였지만, 실험 결과를 비교해 보면 N=70(d=3)의 경우를 제외하고 모든 경우에서 더 우수한 결과를 산출하였다. 이는 제안하는 해 표현법의 해의 길이와Edge Set 표현법의 해의 길이가 동일하지만 제안하는 방법의 경우 트리를 형성하기 위해서 사용한 CB-TCR이 사이클(cycle)을 형성할 경우 해당 사이클을 찾는 과정에서 많은 계산 시간이 소요되기 때문이다. 하지만 CB-TCR의 경우 Edge Set 표현법과는 달리 각 chromosome에 있는 gene들을 쌍(pair)을 지어서 고려하는 것이 아니라 제안하는 해 표현법 내에 있는 모든 가능한 정보를 순차적으로 전부 고려하기 때문에 실험 결과 측면에서는 Edge Set 표현법보다 우수한 결과를 보여주었다. 그리고 CB-TCR과 Edge Set 표현법의 Min. 값 또한 비교해 보면 N=60(d=3)과 N=70(d=3)을 제외하고 모든 실험 데이터에서 더 우수한 값을 찾아냈으며, Max 값의 경우에는 모든 실험 데이터에서 더 우수한 값을 찾아내었다. 제안하는 방법이 기존에 DCMST를 위한 가장 우수한 알고리듬인 Edge Set 표현법보다 우수한 결과를 보여준다는 점은 아주 고무적인 사실이다.

그리고 <Table 3>은 각 방법들 간의 해의 개선 정도와 계산 시간의 비율을 나타내는데, Eedg Set 표현법이 CF-TCR을 개선하는 정도가 최대 0.43%인데 비해서 CB-TCR이 Eedg Set를 개선하는 정도는 최대 1.69%로 더 크다는 것을 알 수 있으며, 계산 시간 비율을 비교해 보면, CB-TCR이 EedgSet에 비해 최대 6.18배 더 많은 계산 시간을 필요로 하는 반면에 EedgSet는 CF-TCR에 비해 최대 13.63배 더 많은 계산 시간을 필요로 한다. 따라서 소요되는 계산 시간과 찾은 해의 우수성을 고려해 볼 때 제안하는 방법들이 Edge Set 표현법보다 우수한 결과값을 준다는 것을 알 수 있다.

Table 3. Improved percentages and CPU time ratios

N	*d*	$\frac{(ES-CBTCR)}{CBTCR}\times 100\,(\%)$	$\frac{(CFTCR-ES)}{ES}\times 100\,(\%)$
60	3	0.07(3.97)	0.32(12.04)
	4	1.22(4.27)	0.06(13.63)
	5	1.69(4.18)	0.22(13.41)
70	3	-0.06(4.14)	0.43(12.37)
	4	1.22(4.95)	0.43(13.46)
	5	1.02(4.76)	0.20(13.44)
80	3	0.28(5.29)	0.07(11.71)
	4	1.29(6.18)	0.19(13.00)
	5	1.68(5.97)	0.17(12.81)

( ) : the ratio of computation time between different methods.
ES : Edge Set.

Table 4. Statistical effect of proposed method

N	*d*	CB-TCR vs. EdgeSet	
		t value	P
60	3	-1.0885	0.1453
	4	-8.2774	< 0.0001
	5	-8.5944	< 0.0001
70	3	1.3243	0.1009
	4	-6.3892	< 0.0001
	5	-8.5122	< 0.0001
80	3	-3.6555	< 0.0001
	4	-7.8469	< 0.0001
	5	-9.9225	< 0.0001

<Table 4>는 보다 객관적인 결과 비교를 위해 t-Test를 수행한 결과를 보여 준다. t-Test 결과 N=60(d=3)과 N=70(d=3)을 제

Table 2. The experimental results of test instances used Krishnamoorthy's method

N	*d*	CF-TCR				CB-TCR				EdgeSet			
		Min	Avg.	Max	CPU	Min	Avg.	Max	CPU	Min	Avg.	Max	CPU
60	3	12335	12373.4	12449	12.9	12315	**12324.4**	12337	616.8	12314	12333.6	12399	155.3
	4	9421	9449.9	9506	11.6	9264	**9329.8**	9371	675.2	9412	9443.8	9496	158.1
	5	7683	7707.5	7754	11.6	7501	**7562.8**	7675	651	7678	7690.8	7704	155.6
70	3	16702	16754.9	16837	14.6	16678	16692.2	16709	748.2	16660	**16682.4**	16715	180.6
	4	12739	12788.9	12834	13.2	12500	**12579.7**	12722	879.1	12726	12733.8	12750	177.7
	5	10371	10393.5	10429	13.2	10205	**10268**	10336	844.6	10348	10373	10414	177.4
80	3	21800	21835.3	21900	16.6	21750	**21759.8**	21767	1028.1	21771	21819.9	21939	194.4
	4	16588	16646.3	16718	14.9	16281	**16402.9**	16546	1196.8	16572	16615.2	16681	193.7
	5	13471	13534.6	13623	14.9	13201	**13288.1**	13432	1139.2	13440	13511.1	13559	190.8

외한 모든 경우에서 유의수준0.001이하의 유의한 결과를 보여줌을 확인할 수 있었다. 따라서 CB-TCR이 Eedg Set보다 항상 우수한 결과를 산출할 것임을 알 수 있다. 특히 네트워크의 크기가 커질수록 그리고 차수제약이 높을수록 우수한 결과를 산출할 것임을 알 수 있다.

6. 결 론

본 논문에서는 진화 알고리듬을 위한 새로운 트리 표현 방법과 유전연산자를 제안하였다. 그리고 제안하는 방법을 이용해서 DCMST 문제를 다른 기존 알고리듬과 비교 실험을 수행하였다.

새로운 방법은 비록 유효한 트리를 만들어 내기 위해 repair 과정을 필요로 한다는 단점을 지니고 있지만 기존에 DCMST 문제를 위한 가장 우수한 알고리듬이라 알려진Edge Set 표현법보다 우수한 결과를 보여준다는 것을 다양한 실험 및 통계적인 분석을 통해서 확인할 수 있었다. 특히, CB-TCR의 경우 Edge Set 표현법에 비해 최대 5배의 계산시간을 더 필요로 하지만 성능은 거의 모든 경우에서Edge Set 표현법보다 1% 이상 좋은 결과를 산출하였다. 또한 CF-TCR의 경우 비록 성능 면에서는 Edge Set 표현법보다 나쁜 결과를 산출하였지만, 그 차가 모든 경우에서 0.5% 미만이었던 반면, 계산 시간은 최대 13배까지 줄일 수 있었다. 따라서 두 방법 모두 상황에 따라서 DCMST 문제를 다룰 경우 우선적으로 고려되어야 할 것이다

앞으로 제안하는 트리 표현법들을 다양한 다른 트리 네트워크 문제에 적용하여 제안하는 방법의 유연성을 확인해 볼 계획이며, 동시에 대상 문제에 합당한 유전연산자를 개발할 계획이다.

참고문헌

Abuali, F.N., Wainwright, R.L and Schoenefeld, D.A.(1995), Determinant Factorization: A New Encoding Scheme for Spanning Trees Applied to the Probabilistic Minimum Spanning Tree Problem, in *Proceedings of The Sixth International Conference on Genetic Algorithms*, 470-477.

Bean, J. C.(1994), Genetic algorithms and random keys for sequencing and optimization, *ORSA Journal on Computing*, **6**(2), 154-160.

Dengiz, B., Altiparmak, F., and Smith, A.E.(1997), Local Search Genetic Algorithm for Optimal Design of Reliable Networks, *IEEE Transactions on Evolutionary Computation*, **1**(3), 179-188.

Garey, M.R. and Johnson, D.S.(1979), Computers and Intractability, A Guide to the Theory of NP-Completeness, *San Francisco, Freeman*.

Gen, M. and Cheng, R.(1997), Genetic Algorithms and Engineering Design, *JOHN WILEY & SONS*.

Gottlieb, J., Julstrom, G.A., Raidl, G.R. and Rothlauf, F.(2001), Prufer number: A Poor Representation of Spanning Trees for Evolutionary Search, *IlliGAL Report No.2001001, Illinois Genetic Algorithms Lab., Univ. of Illinois*.

Krishnamoorthy, M., Ernst, A.T. and Sharaiha, Y.M.(2001), Comparision of Algorithms for the Degree Constrained Minimum Spanning Tree, *Journal of Heuristics*, **7**, 587-611.

Yu Li,(2001), An Effective Implementation of a Direct Spanning Tree Representation in GAs, *EvoWorkshop 2001, LNCS* 2037, 11-19.

Michalewicz, Z.(1992), *Genetic Algorithms+Data Structures= Evolution Programs*.

Narula, S.C. and Ho, C.A.(1980), Degree-constrained minimum spanning tree, *Computer and Operations Research*, 7, 239-249.

Piggott, P. and Suraweera, F.(1995), Encoding graphs for genetic algorithms: An investigation using the minimum spanning tree problem. Progress in Evolutionary Computation, *LNAI* 956, 305-314.

Raidl, G.R. and Julstrom, B.A.(2003), Edge-Sets: An Effective Evolutionary Coding of Spanning Trees, *IEEE Transactions on Evolutionary Computation,* **7**(3), 225-239.

Raild, G.R. and Drexel, C.(2000) A Predecessor Coding in an Evolutionary Algorithm for the Capacitated Minimum Spanning Tree Problem, *Late-Breaking-Papera, Proc. of the 2000 Genetic and Evolutionary computation Conference*, 309-316.

Rothlauf, F., Goldberg, D.E. and Heinzl, A.(2002), Network Random Keys-A Tree Network Representation Scheme for Genetic and Evolutionary Algorithms, *Evolutionary Computation*, **10**(1), 75-97.

Schindler, B., Rothlauf, F. and Pesch, H.J.(2002), Evolution Strategies, Network Random Keys, and the One-Max Tree Problem, *LNCS* 2279, 143-152.

Soak, S.M., David Corne and Ahn, B.H.(2004) A New Encoding for the Degree Constrained Minimum Spanning Tree Problem, *KES2004, LNAI* 3213, 952-958.

Zhou, G. and Gen, M.(1998), An Effective Genetic Algorithm Approach to The Quadratic Minimum Spanning Tree Problem, *Computers & Operations Researches*, **25**(3), 229-237.