Chapter 9
해 탐색 알고리즘

차례

9.1 백트래킹(Backtracking) 기법

- 해를 찾는 도중에 '막히면'(즉, 해가 아니면) 되돌아가서 다시 해를 찾아 가는 기법이다.
- 백트래킹 기법은 최적화(optimization) 문제와 결정(decision) 문제를 해결한다.

- 결정 문제
  - 문제의 조건을 만족하는 해가 존재하는 지의 여부를 'yes' 또는 'no'로 답하는 문제

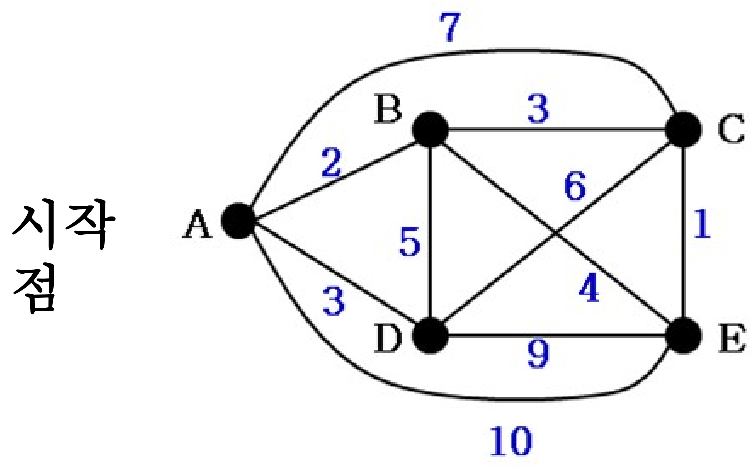
TSP를 위한 백트래킹 알고리즘

- tour = [시작점]
  - tour는 점의 순서 (sequence)
- bestSolution = (tour, ∞)
  - bestSolution은 현재까지 찾은 가장 거리가 짧은 해
  - 2개의 성분 (tour, tour의 거리)으로 표시
    - tour는 점의 순서
    - tour의 거리는 'bestSolution의 거리'로 표현
  - 초기에 tour는 시작점만 가지므로 그 거리는 가장 큰 상수로 초기화

알고리즘

```
tour = [시작점]  // tour는 점의 순서
bestSolution = (tour, ∞)
BacktrackTSP(tour)
1.  if tour가 완전한 해이면
2.      if tour의 거리 < bestSolution의 거리
3.          bestSolution = (tour, tour의 거리)
4.  else
5.      for tour를 확장 가능한 각 점 v에 대해서
6.          newTour = tour + v    // 기존 tour의 뒤에 v 를 추가
7.          if newTour의 거리 < bestSolution의 거리
8.              BacktrackTSP(newTour)
```

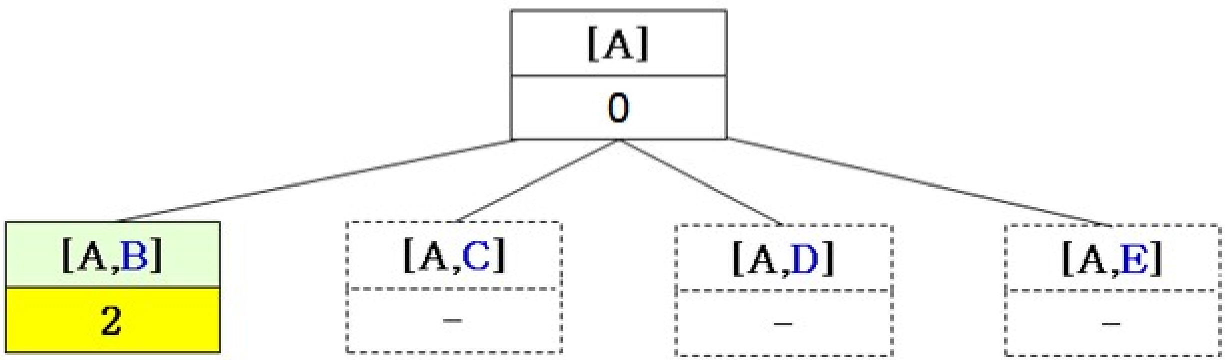
BacktrackTSP 알고리즘의 수행 과정



- 시작점 A, tour=[A]이고, bestSolution=([A],∞)
- BacktrackTSP(tour) 호출

newTour = [A, B]

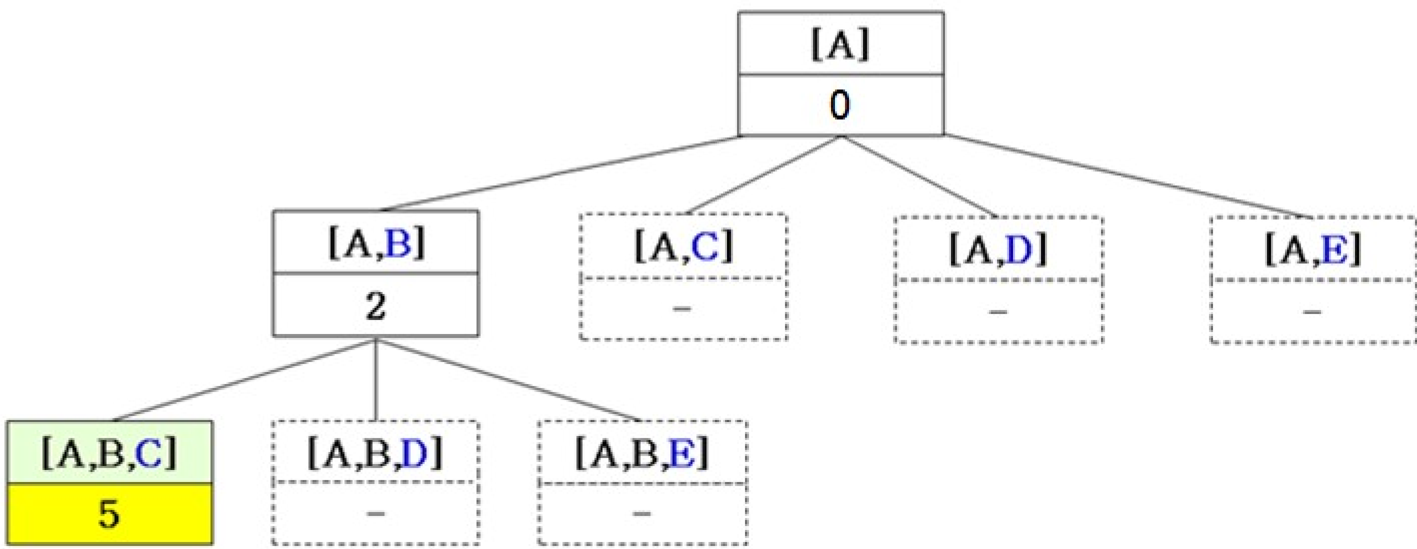
- tour [A]를 확장할 수 있는 점은 B, C, D, E, 따라서 각 점에 대해 루프 수행
- 먼저 점 B에 대해서
- newTour = [A, B], newTour의 거리 = 2, 왜냐하면 간선 (A, B)의 가중치가 2이므로



- BacktrackTSP([A, B]) 순환 호출

newTour = [A, B, C]

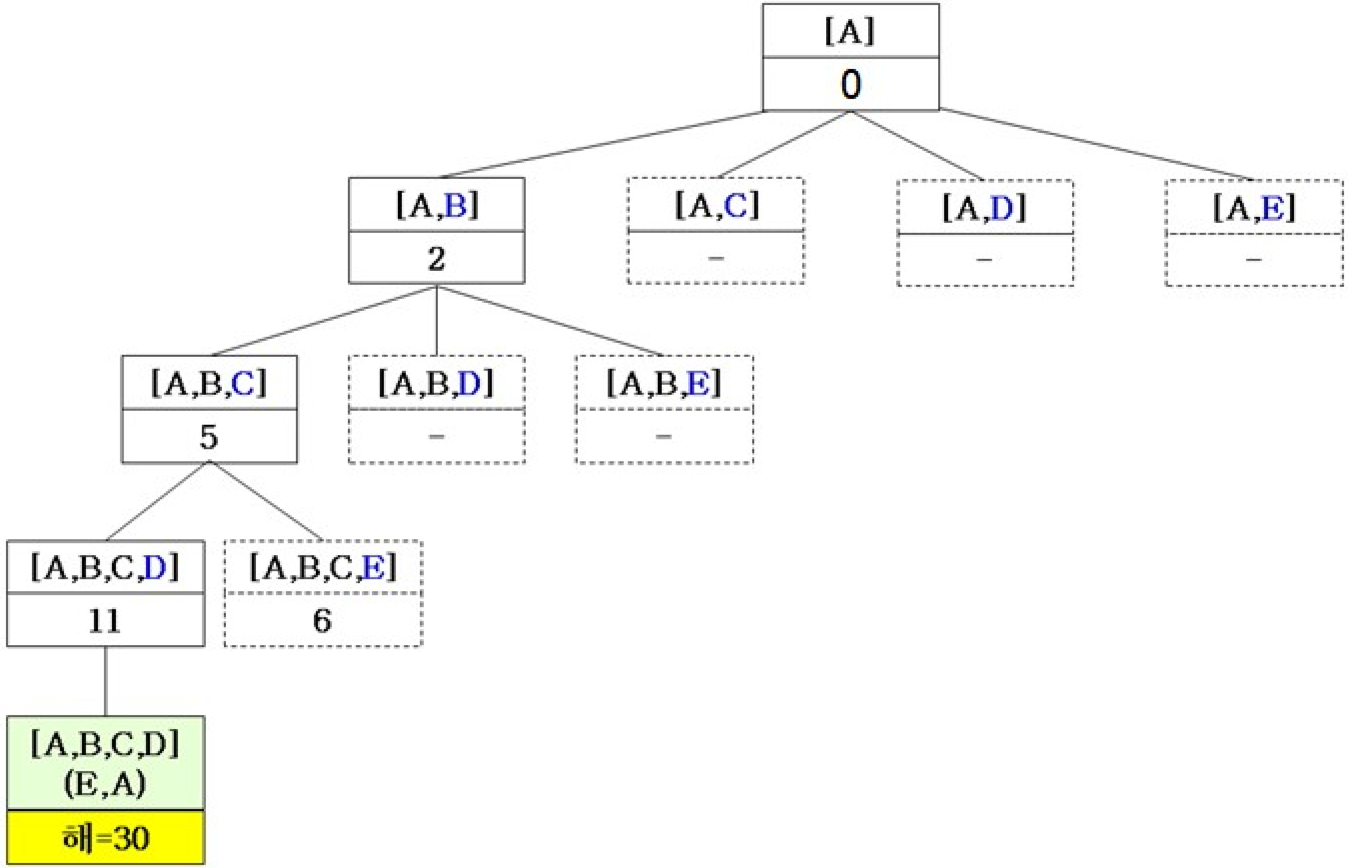
– tour [A, B]를 확장할 수 있는 점은 C, D, E, 따라서 각 점에 대해 루프 수행, 먼저 점 C에 대해서
– newTour=[A, B, C], newTour의 거리 = 5, 왜냐하면 간선 (B,C)의 가중치가 3이므로



– BacktrackTSP([A, B, C]) 순환 호출

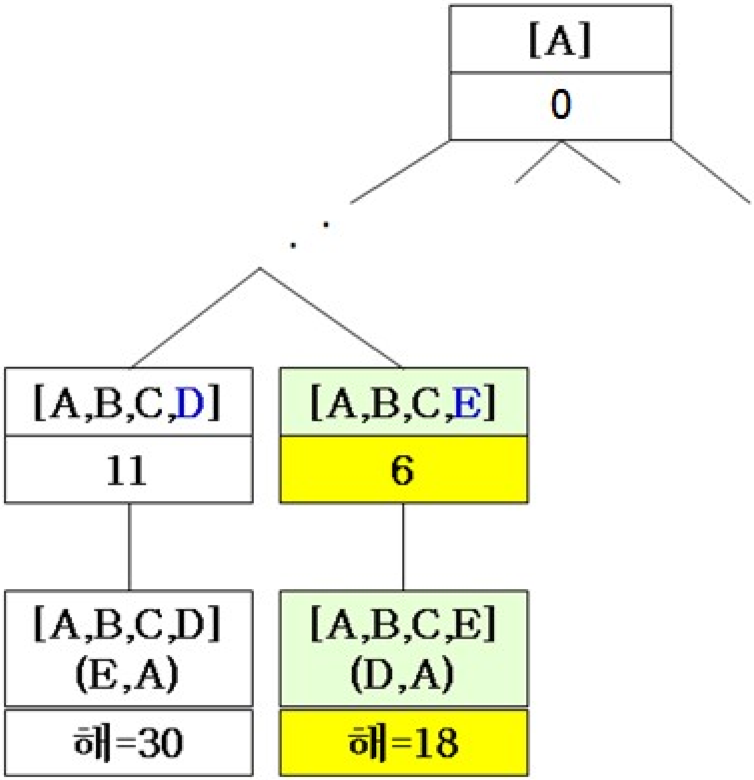
bestSolution=([A, B, C, D, E, A], 30)

- 이와 같이 계속 탐색을 진행하면 첫 번째 완전한 해 bestSolution=([A, B, C, D, E, A], 30)



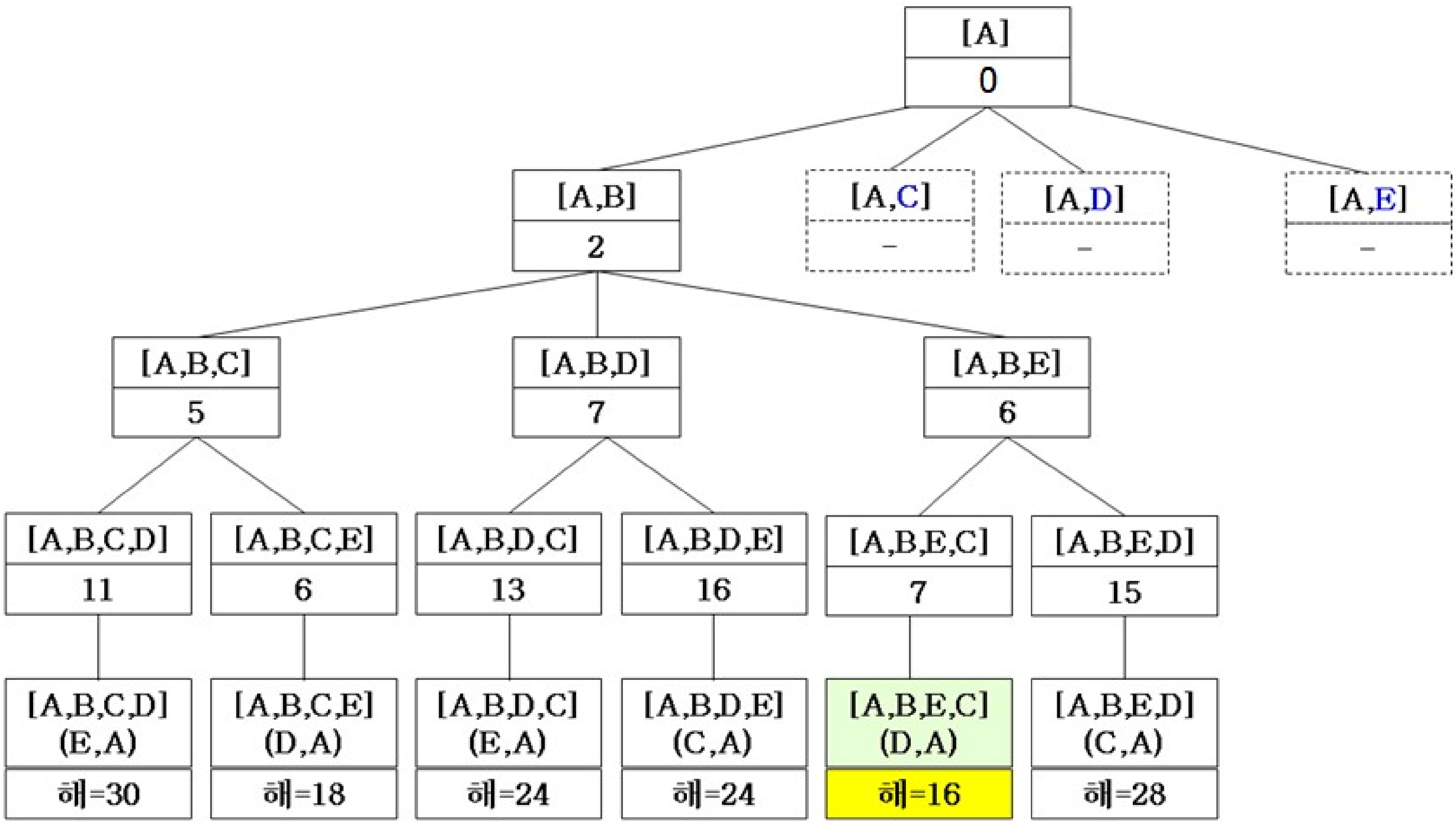
bestSolution=([A, B, C, E, D, A], 18)

- 첫 번째 완전한 해를 찾은 후, 더 짧은 해인 bestSolution=([A, B, C, E, D, A], 18)을 찾는다.

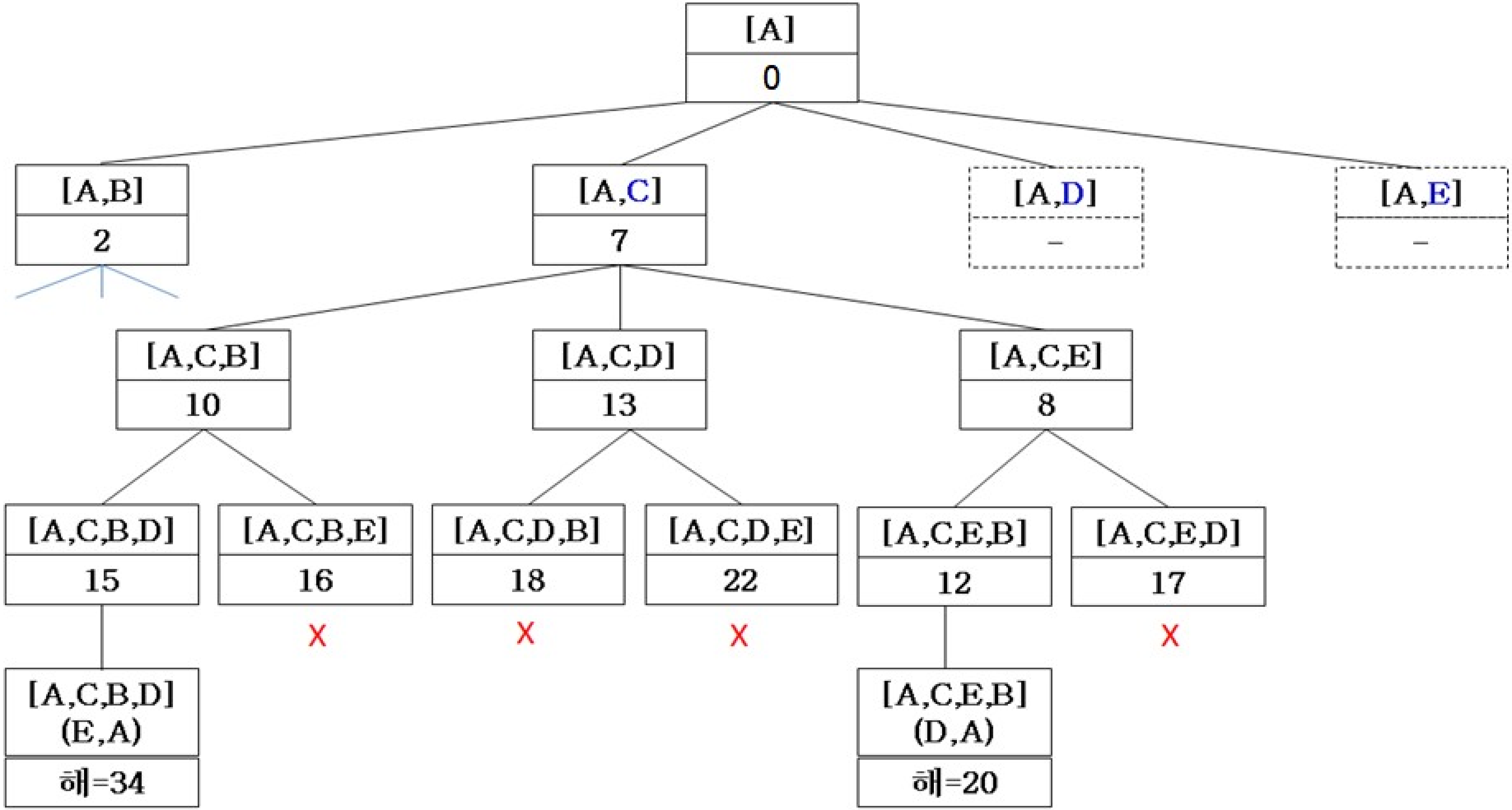


bestSolution=([A, B, E, C, D, A], 16)

- tour=[A,B]에 대해 모든 수행을 마친 결과
  - bestSolution=([A, B, E, C, D, A], 16)

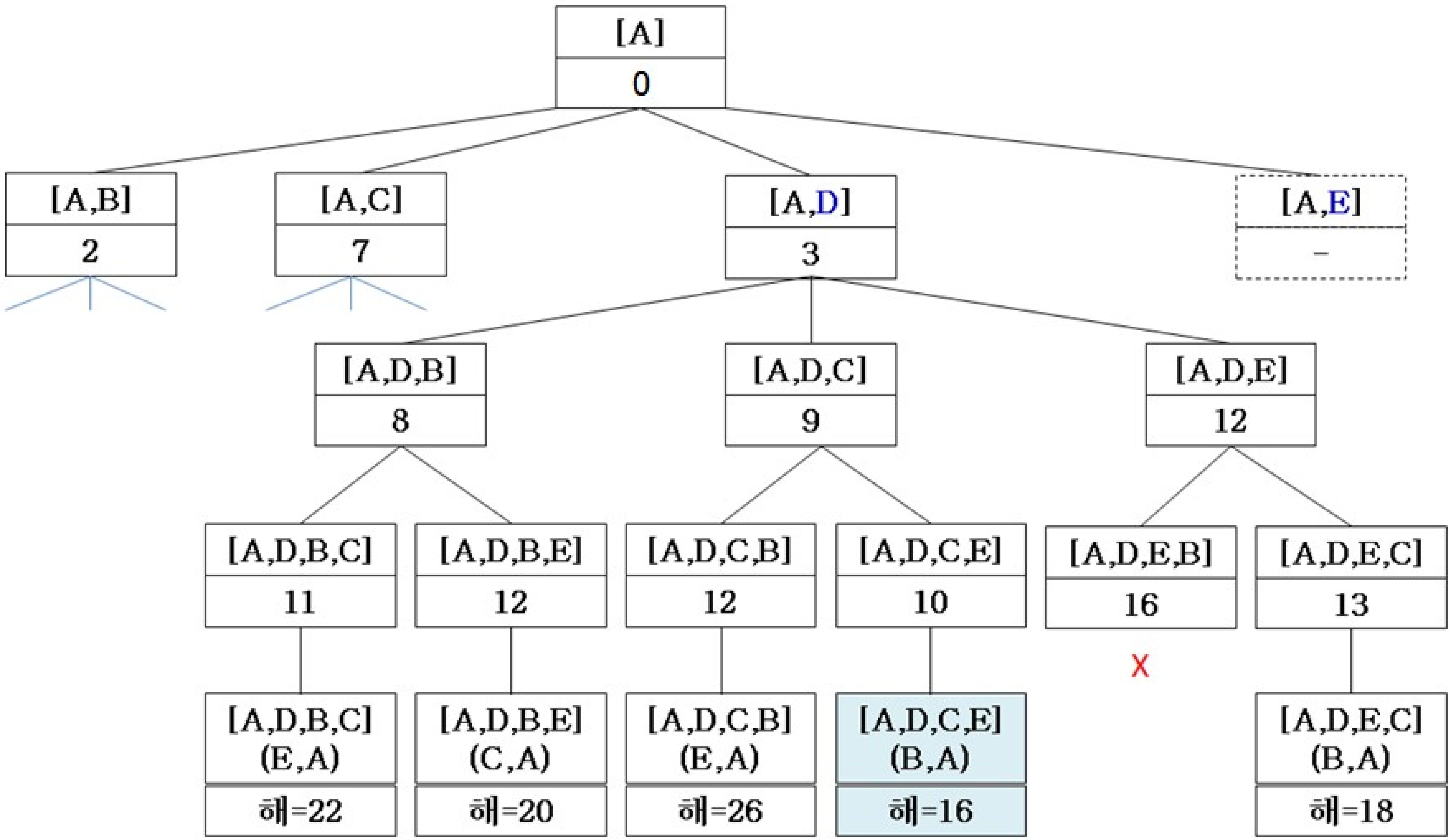


tour = [A, C]에 대한 결과



x로 표시된 4개의 상태 각각은 bestSolution의 거리보다 짧지 않으므로 가지치기(pruning) 됨

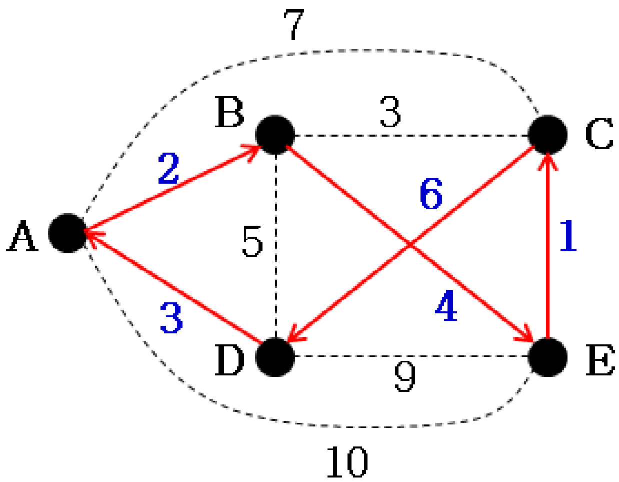
tour = [A, D]에 대한 결과



BacktrackTSP 알고리즘의 수행 결과

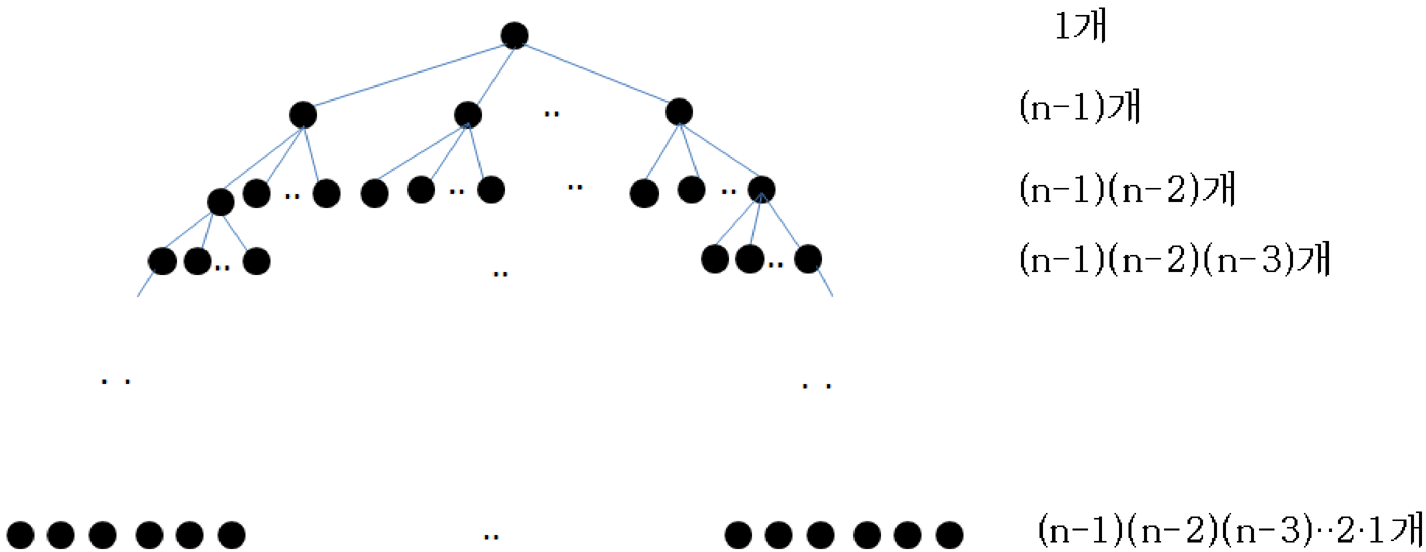
- 마지막으로 tour=[A, E]에 대해서 탐색을 수행하여도 bestSolution보다 더 우수한 해는 없다.

- 최종해 = [A, B, E, C, D, A], 거리 = 16



시간 복잡도

- Backtracking 알고리즘의 시간 복잡도는 상태 공간 트리의 노드 수에 비례
- n개의 점이 있는 입력 그래프에 대해서 BacktrackTSP 알고리즘이 탐색하는 최대 크기의 상태 공간 트리



  – 위의 트리의 이파리 노드 수만 계산해도 (n-1)!

시간 복잡도

- 문제에 따라서 이진 트리 형태의 상태 공간 트리가 형성되기도 하는데 이때에도 최악의 경우에 2^n개의 노드를 대부분 탐색해야 하므로 지수 시간이 걸림
- 이는 모든 경우를 다 검사하여 해를 찾는 완전 탐색 (Exhaustive Search)의 시간 복잡도와 같음
- 그러나 일반적으로 백트래킹 기법은 '가지치기'를 하므로 완전 탐색보다 훨씬 효율적임

9.2 분기 한정 (Branch-and-Bound) 기법

- 백트래킹 기법은 깊이 우선 탐색수행
- 최적화 문제에 대해서는 최적해가 상태 공간 트리의 어디에 있는지 알 수 없으므로, 트리에서 대부분의 노드를 탐색하여야 함
- 입력의 크기가 커지면 해를 찾는 것은 거의 불가능
- 분기 한정(Branch-and-bound) 기법은 이러한 단점을 보완하는 탐색 기법

분기 한정 (Branch-and-Bound) 기법

- 분기 한정 기법은 상태 공간 트리의 각 노드(상태)에 특정한 값 (한정값)을 부여
- 노드의 한정값을 활용하여 가지치기를 함으로써 백트래킹 기법보다 빠르게 해를 찾는다.
- 분기 한정 기법에서는 가장 우수한 한정값을 가진 노드를 먼저 탐색하는 최선 우선 탐색(Best First Search)으로 해를 찾는다.

분기 한정 기법의 효율적인 탐색 원리

- 최적해를 찾은 후에 나머지 노드의 한정값이 최적해의 값과 같거나 나쁘면 더 이상 탐색하지 않는다.
- 상태 공간 트리의 대부분의 노드가 문제의 조건에 맞지 않아서 해가 되지 못한다.
- 최적해가 있을 만한 영역을 먼저 탐색한다.

알고리즘

Branch-and-Bound(S) // S는 문제의 초기 상태

1. 상태 S의 한정값을 계산한다.
2. activeNodes = { S } // 탐색되어야 하는 상태의 집합
3. bestValue = ∞ // 현재까지 탐색된 해 중의 최솟값
4. while (activeNodes ≠ Ø) {
5. S_{min}= activeNodes의 상태 중에서 한정값이

   가장 작은 상태
6. S_{min}을 activeNodes에서 제거
7. S_{min}의 자식(확장 가능한) 노드 S'_1, S'_2, …, S'_k를 생성하고, 각각의 한정값을 계산한다.

```
for i=1 to k          // 확장한 각 자식 S'_i에 대해서
    if S'_i의 한정값 ≥ bestValue
        S'_i를 가지치기한다.  // 더 우수한 해가 없으므로
    else if  S'_i가 완전한 해이고 S'_i의 값 < bestValue
        bestValue = S'_i의 값
        bestSolution = S'_i
    else
        S'_i를 activeNodes에 추가
```

TSP

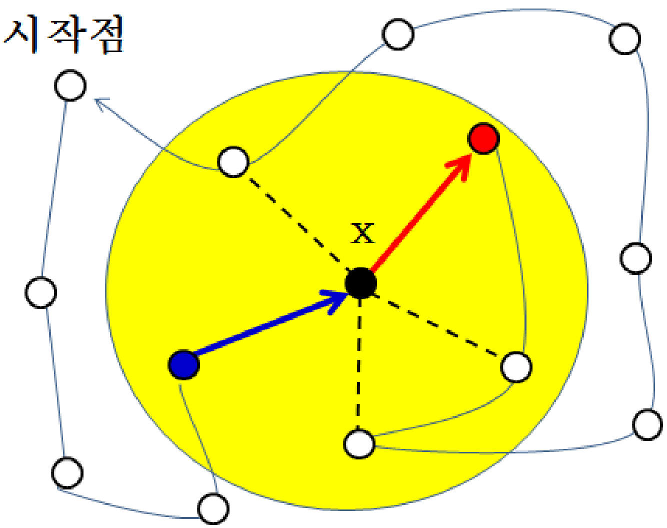
- 분기 한정 기법으로 문제의 최적해를 찾으려면, 먼저 각 상태에서의 한정값을 계산하여야

- 한정값 계산을 위한 여행자 문제의 조건
  - 문제의 해는 주어진 시작점에서 출발하여 모든 다른 점을 1번씩만 방문하고 시작점으로 돌아와야 한다.

경로 상의 1개의 점 x를 살펴보면, 다른 점에서 점 x로 들어온 후에 점 x를 떠나 또 다른 점으로 나간다. 이를 점 x의 한정값 계산에 활용

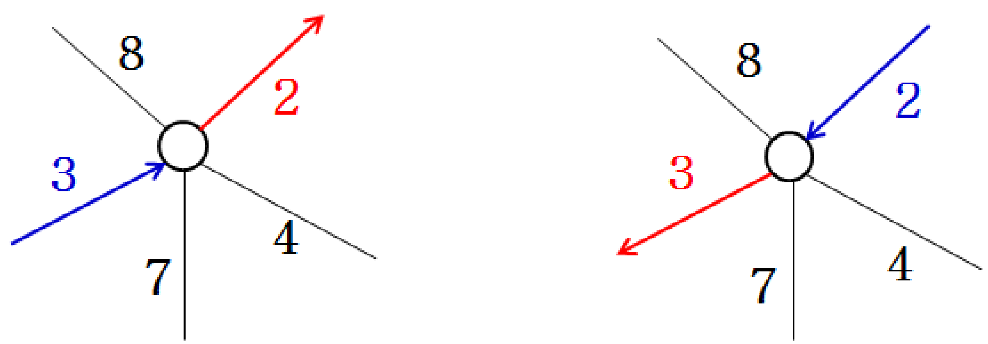
TSP의 한정값 계산 방법

- TSP에서 임의의 점 x에서의 한정값 = 시작점부터 점 x까지의 경로 길이 + 점 x를 떠나서 남은 다른 점들을 1번씩만 방문하고 시작점으로 돌아오는 경로의 '예측' 길이
- 여행자 문제는 최단 경로를 찾는 문제이므로 앞으로 방문해야 할 각 점 x에 연결된 간선 중에서 가장 짧은 두 간선의 가중치의 평균의 합을 예측 길이를 계산하는데 사용
  - 가중치의 합을 1/2로 곱하는 (평균을 내는) 이유는 한 점에서 나가는 간선은 인접한(다른) 점에서 들어오는 간선과 동일하므로
  - 단, 소수점 이하의 숫자는 올림



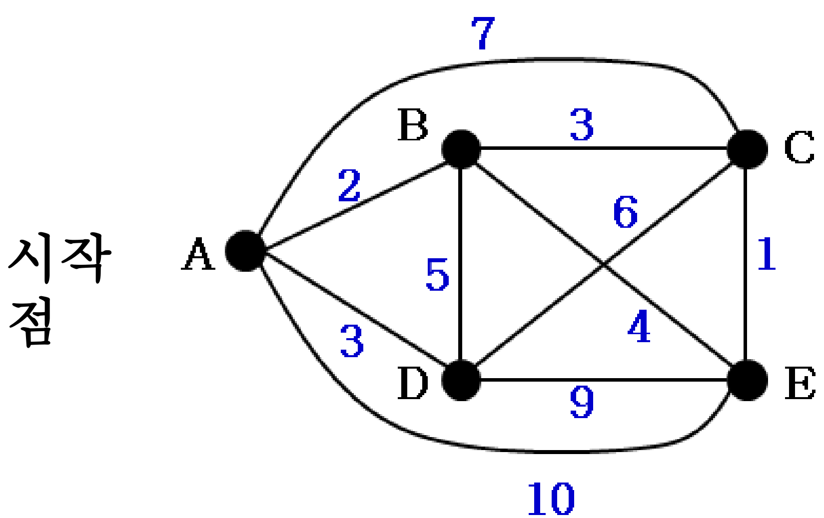
점에 인접한 간선 중에서 2개의 가장 작은 가중치

- 가중치 3인 간선으로 들어와서 가중치 2인 간선으로 나가든지 (왼쪽 그림)
- 반대로 가중치 2인 간선으로 들어와서 가중치 3인 간선으로 나가든지 (오른쪽 그림)
- 두 경우 모두 최소의 비용으로 점을 방문한다.



Branch-and-Bound 알고리즘 수행 과정

- 5개의 점(A, B, C, D, E)으로 된 그래프
- 초기 상태= [A]
- Branch-and-Bound([A]) 호출



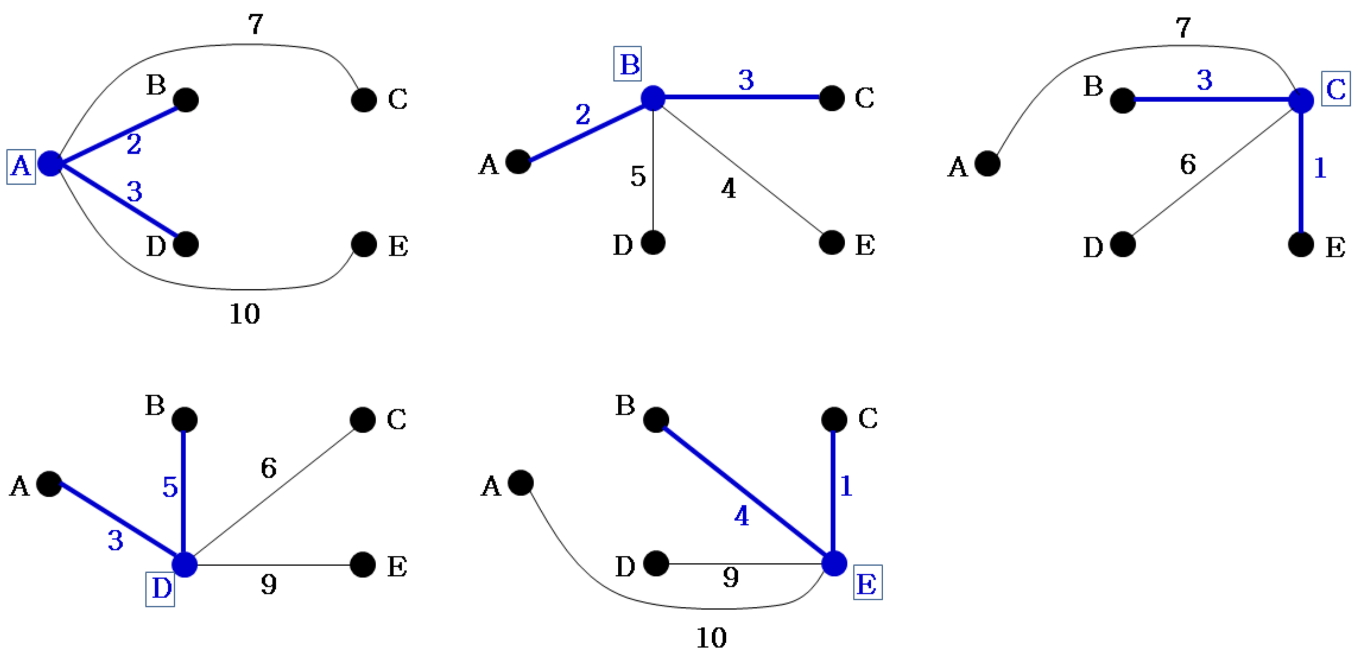
초기 상태 [A]의 한정값

- 초기 상태는 경로를 시작하기 전이므로, 각 점에 인접한 간선의 가중치 중에서 가장 작은 2개의 가중치의 합을 구한 다음에, 모든 점의 합의 1/2을 한정값으로

A B C D E

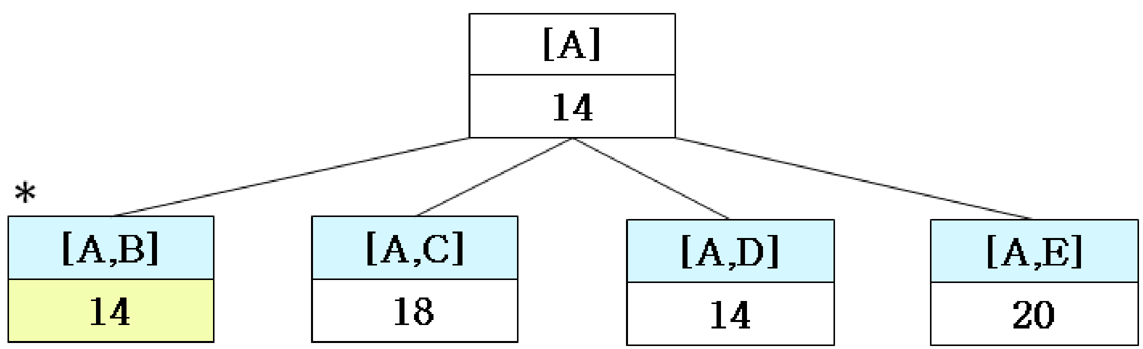
$$[(2+3) + (2+3) + (1+3) + (3+5) + (1+4)] \times 1/2 = 27/2 = 14$$

[A]
14



S_{min}= [A]

- activeNodes={S}, bestValue=∞로 각각 초기화
- activeNodes 집합에 초기 상태 [A]만 있으므로, S_{min}= [A]
- S_{min}(즉, 상태 [A])의 자식 노드 생성 및 한정값 계산
- 자식 노드는 점이 B인 상태 [A,B], C인 상태 [A,C], D인 상태 [A,D], E인 상태 [A,E]

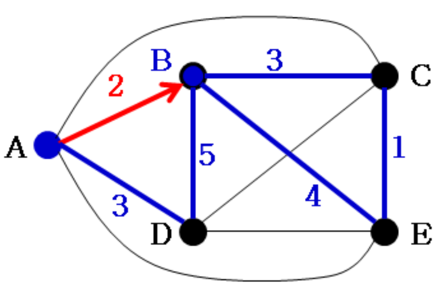


상태 [A,B], [A,C], [A,D], [A,E]의 한정값

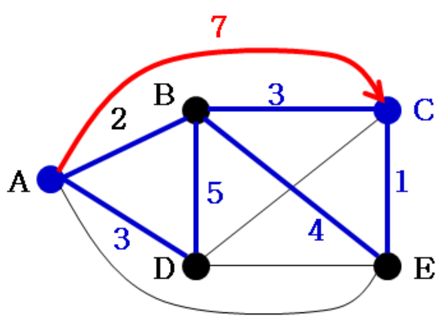
– [A,B]의 한정값 =([2+3]+[2+3]+[1+3]+[3+5]+[1+4])/2 = 27/2 = 14

– [A,C]의 한정값 = ([2+7]+[2+3]+[1+7]+[3+5]+[1+4])/2 = 36/2 = 18

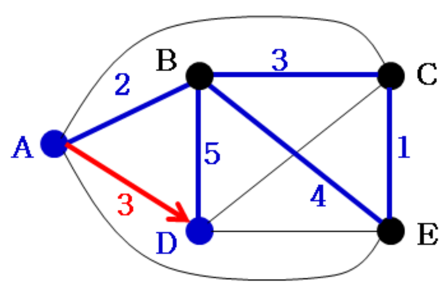
– [A,D]의 한정값 = ([2+3]+[2+3]+[1+3]+[3+5]+[1+4])/2 = 27/2 = 14



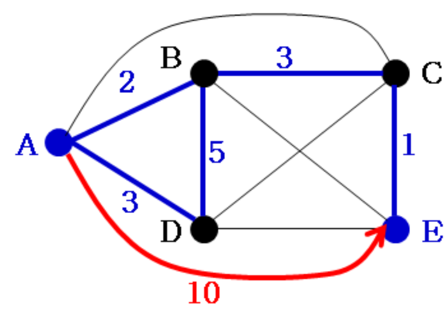
상태 [A, B]



상태 [A, C]



상태 [A, D]



상태 [A, E]

[A, B], [A, C], [A, D], [A, E] activeNodes에 추가

- activeNodes = {[A,B], [A,C], [A,D], [A,E]}

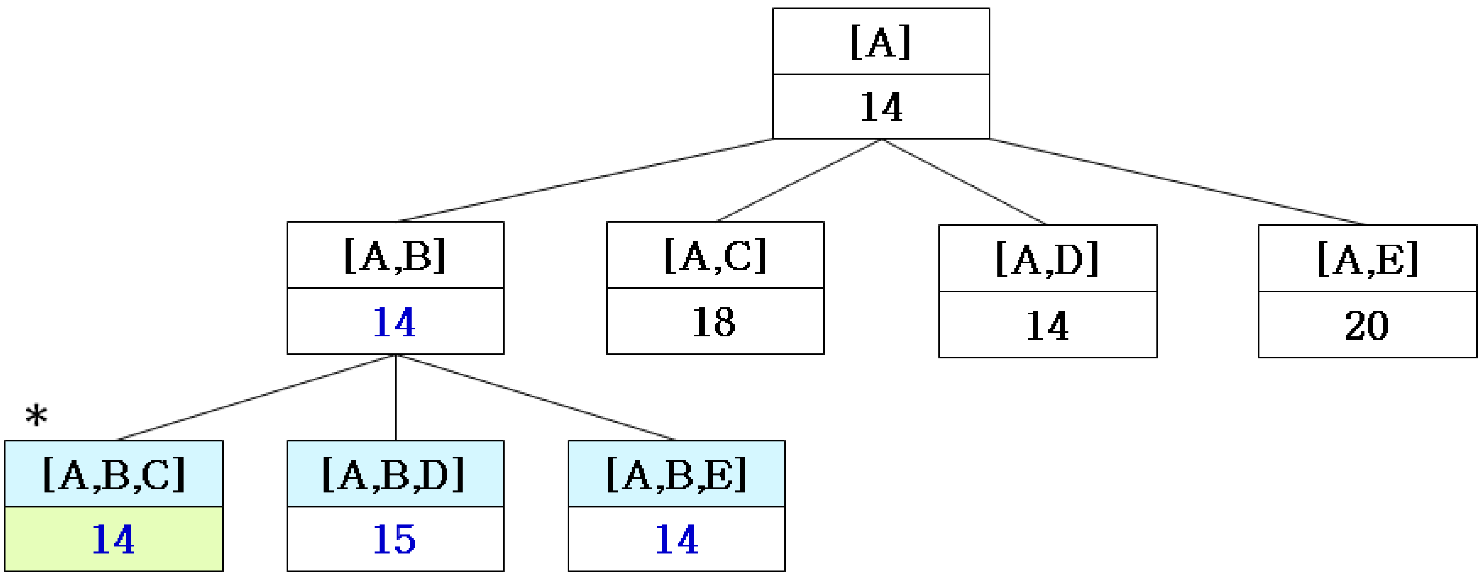
  14 18 14 20

- 상태 [A, B]와 [A, D]가 동일한 최소의 한정값을 가지므로 임의로 S_{min} = [A, B]

- activeNodes에서 [A, B]를 제거
  - activeNodes = { [A, C], [A, D], [A, E]}

[A,B]의 자식 상태 생성

– 자식 노드는 세 번째 방문하는 점이 C인 상태 [A, B, C], D인 상태 [A, B, D], E인 상태 [A, B, E]

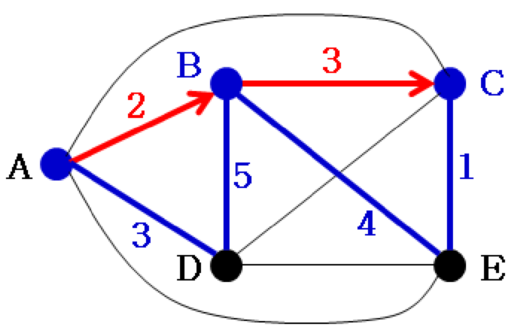


[A, B, C], [A, B, D], [A, B, E] 한정값 계산

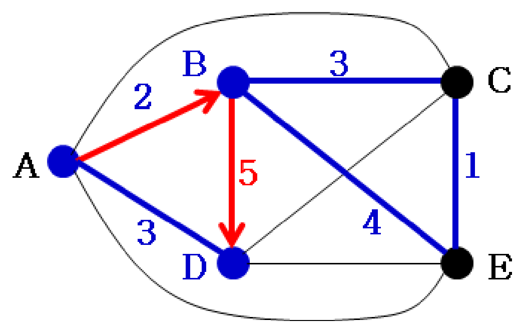
– [A, B, C]의 한정값
  - ([2+3]+[2+3]+[1+3]+[3+5]+[1+4])/2 = 27/2 = 14

– [A, B, D]의 한정값
  - ([2+3]+[2+5]+[1+3]+[3+5]+[1+4])/2 = 29/2 = 15

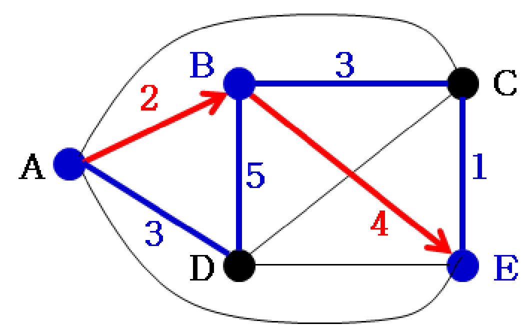
– [A, B, E]의 한정값



상태 [A, B, C]



상태 [A, B, D]



상태 [A, B, E]

[A, B, C], [A, B, D], [A, B, E] activeNodes에 추가

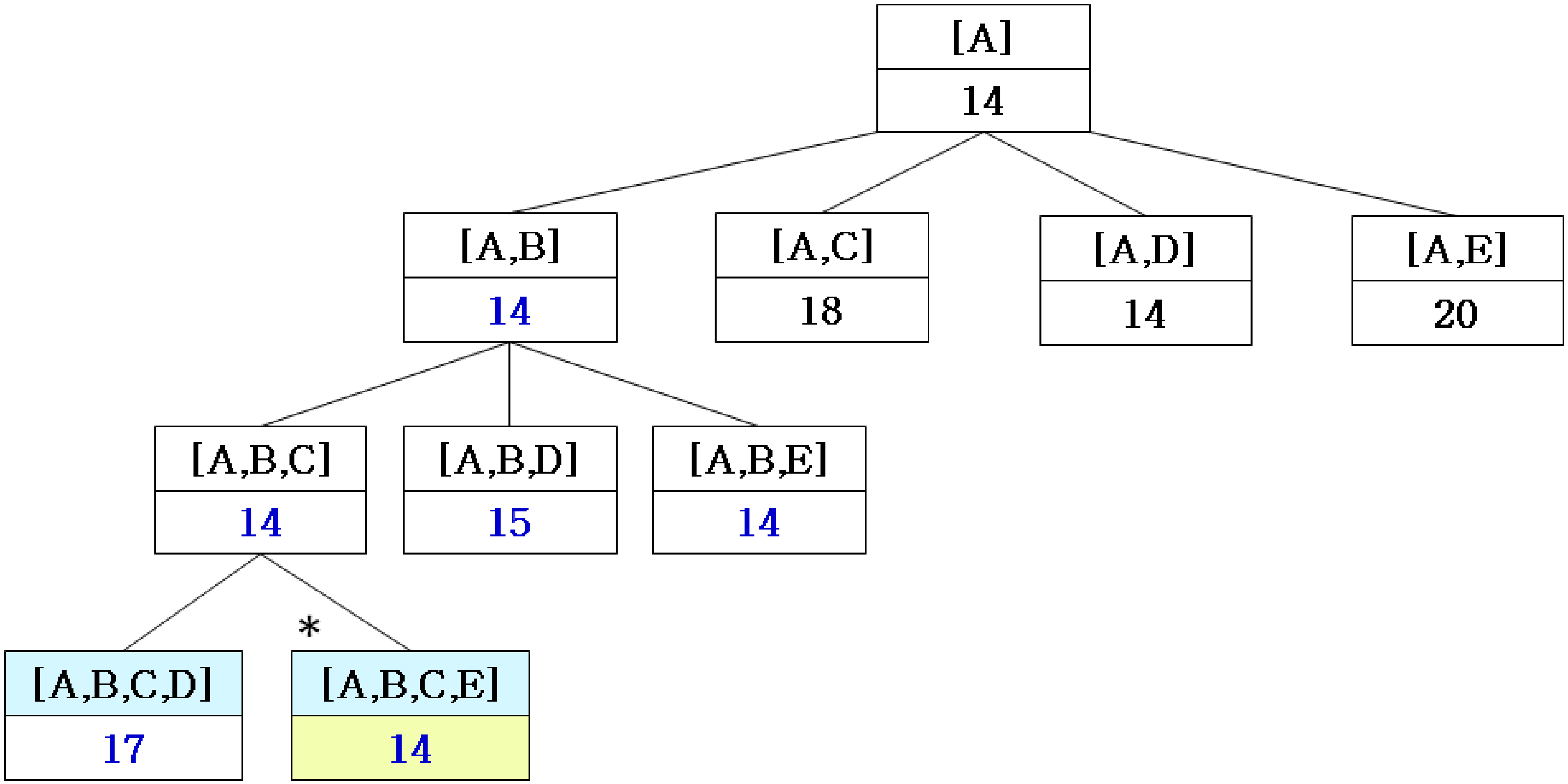
- activeNodes = {[A, C], [A, D], [A, E], [A, B, C], [A, B, D], [A, B, E]}

S_{min} = [A, B, C]

- 상태 [A, B, C], [A, B, E], [A, D]가 동일한 최소의 한정값을 가지므로 임의로 S_{min} = [A,B,C]
- activeNodes에서 [A, B, C] 제거
  - activeNodes = {[A, C], [A, D], [A, E], [A, B, D], [A, B, E]}

[A,B,C]의 자식 상태 생성

- 자식 노드들은 네 번째 방문하는 점이 D인 상태 [A, B, C, D]와 E인 상태 [A, B, C, E]



[A, B, C, D], [A, B, C, E]의 한정값

– [A, B, C, D]의 한정값:
  ([2+ 3]+ [2+ 3]+ [6+ 3]+ [3+ 6]+ [1+ 4])/2 = 33/2
  = 17
– [A, B, C, E]의 한정값:
  ([2+ 3]+ [2+ 3]+ [1+ 3]+ [3+ 5]+ [1+ 4])/2 = 27/2
  = 14

[A, B, C, D], [A, B, C, E] activeNodes에 추가

- activeNodes = {[A, C], [A, D], [A, E], [A, B, D], [A, B, E], [A, B, C, D], [A, B, C, E]}

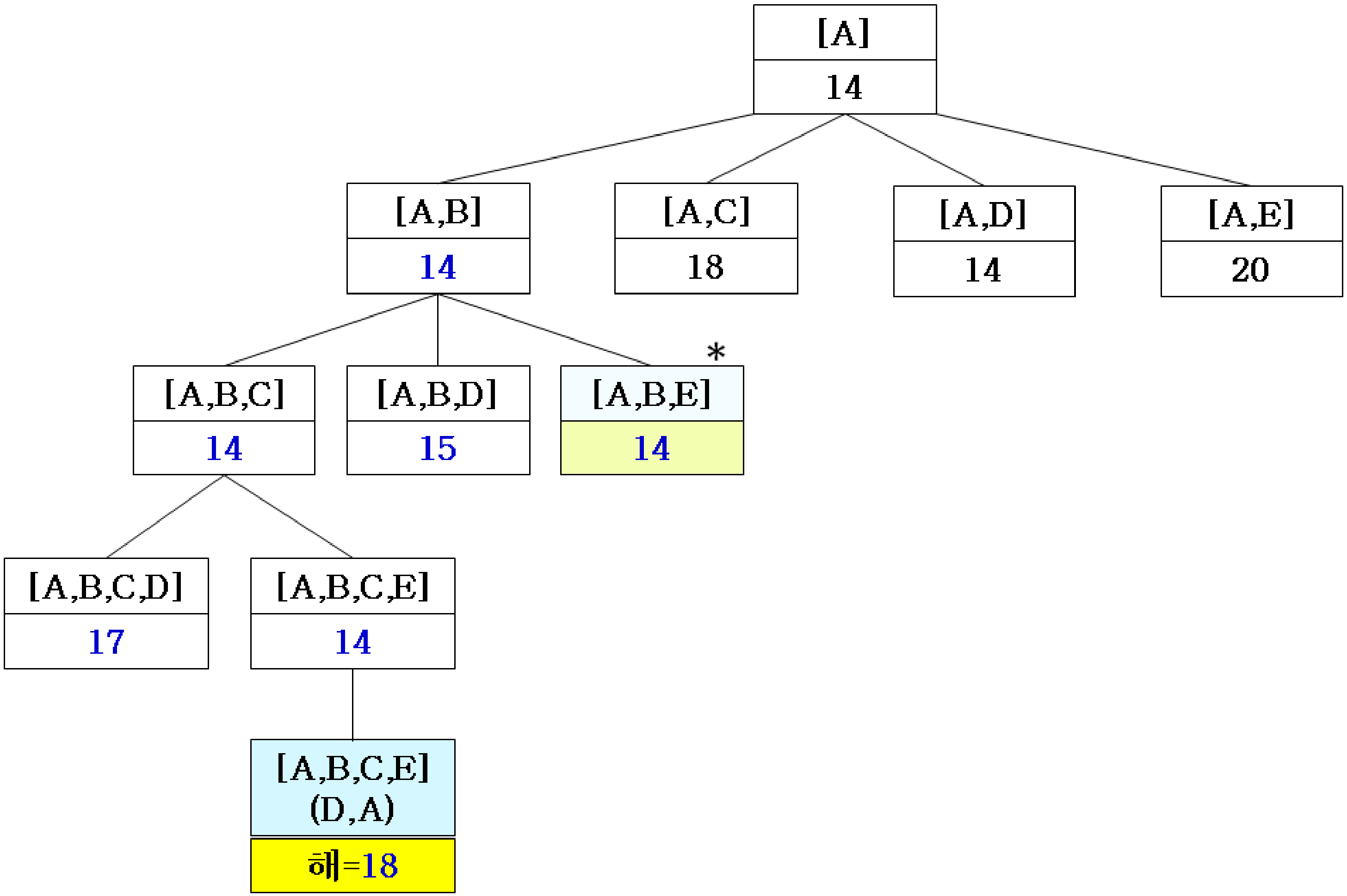
S_{min} = [A, B, C, E]

- 상태 [A, B, C, E], [A, B, E], [A, D]가 동일한 최소 한정값을 가지므로 임의로 S_{min} = [A, B, C, E]
- activeNodes에서 [A, B, C, E] 제거
  - activeNodes = {[A, C], [A, D], [A, E], [A, B, D], [A, B, E], [A, B, C, D]}

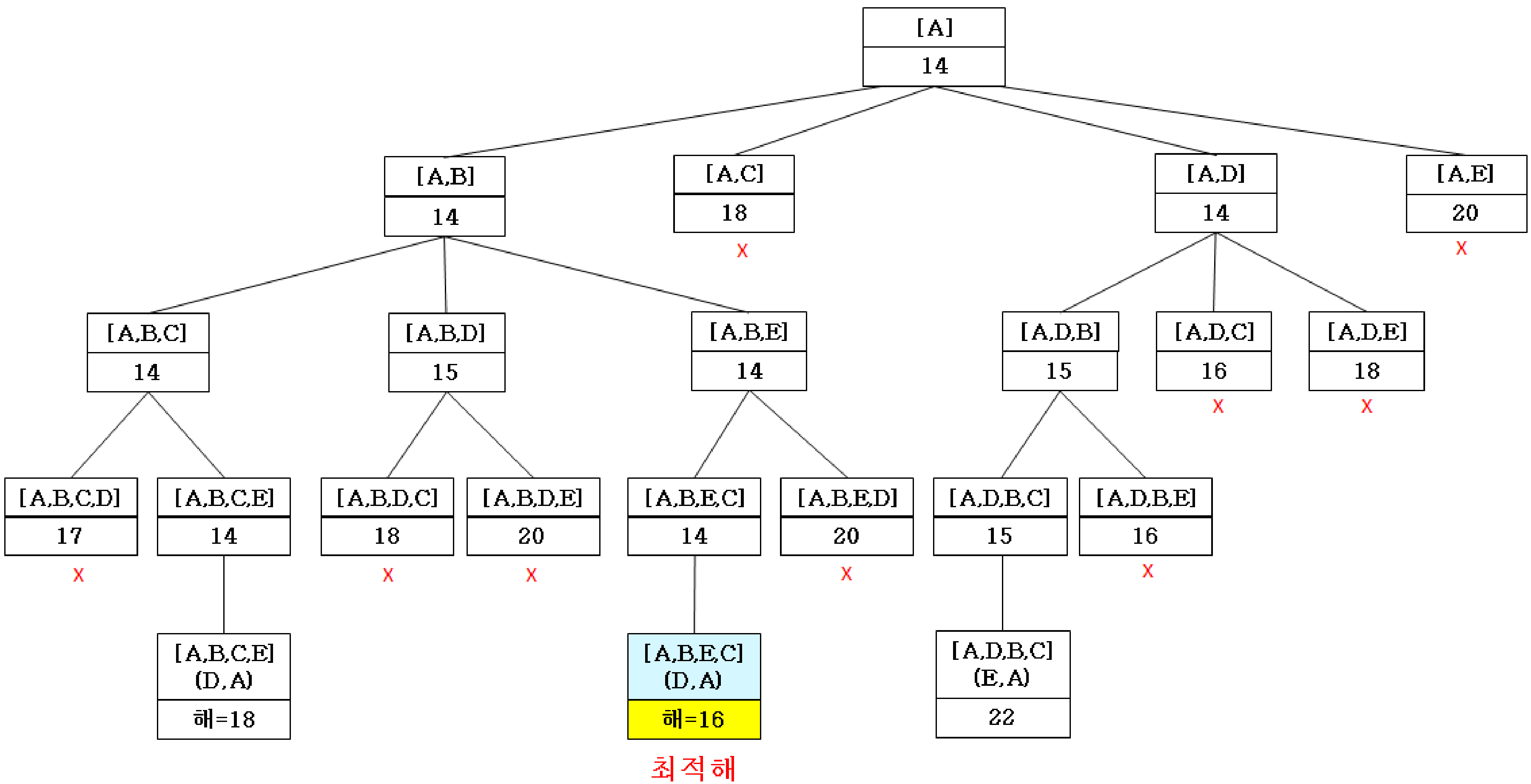
[A,B,C,E]의 자식 상태 생성

- [A, B, C, E]의 자식 상태는 D를 방문하는 상태 [A, B, C, E, D]
  - D에서 시작점 A로 돌아가야 하므로 하나의 해가 완성
  - 경로 A-B-C-E-D-A의 거리는 2+3+1+9+3 = 18
- Line 11
  - bestValue=18, bestSolution=[A, B, C, E, D, A]

bestValue=18, bestSolution=[A, B, C, E, D, A]

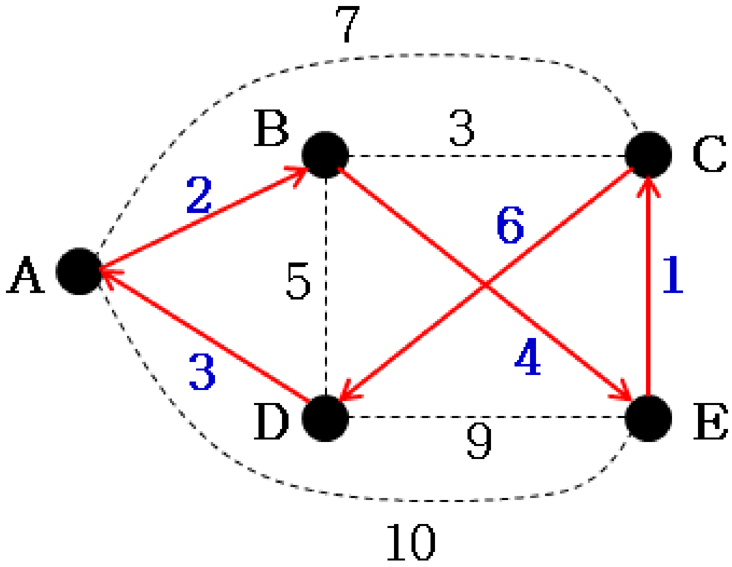


상태 [A,B,E]로부터 탐색 결과



최적해

- [A, B, E, C, D, A]가 최적해이고, 경로의 길이는 16



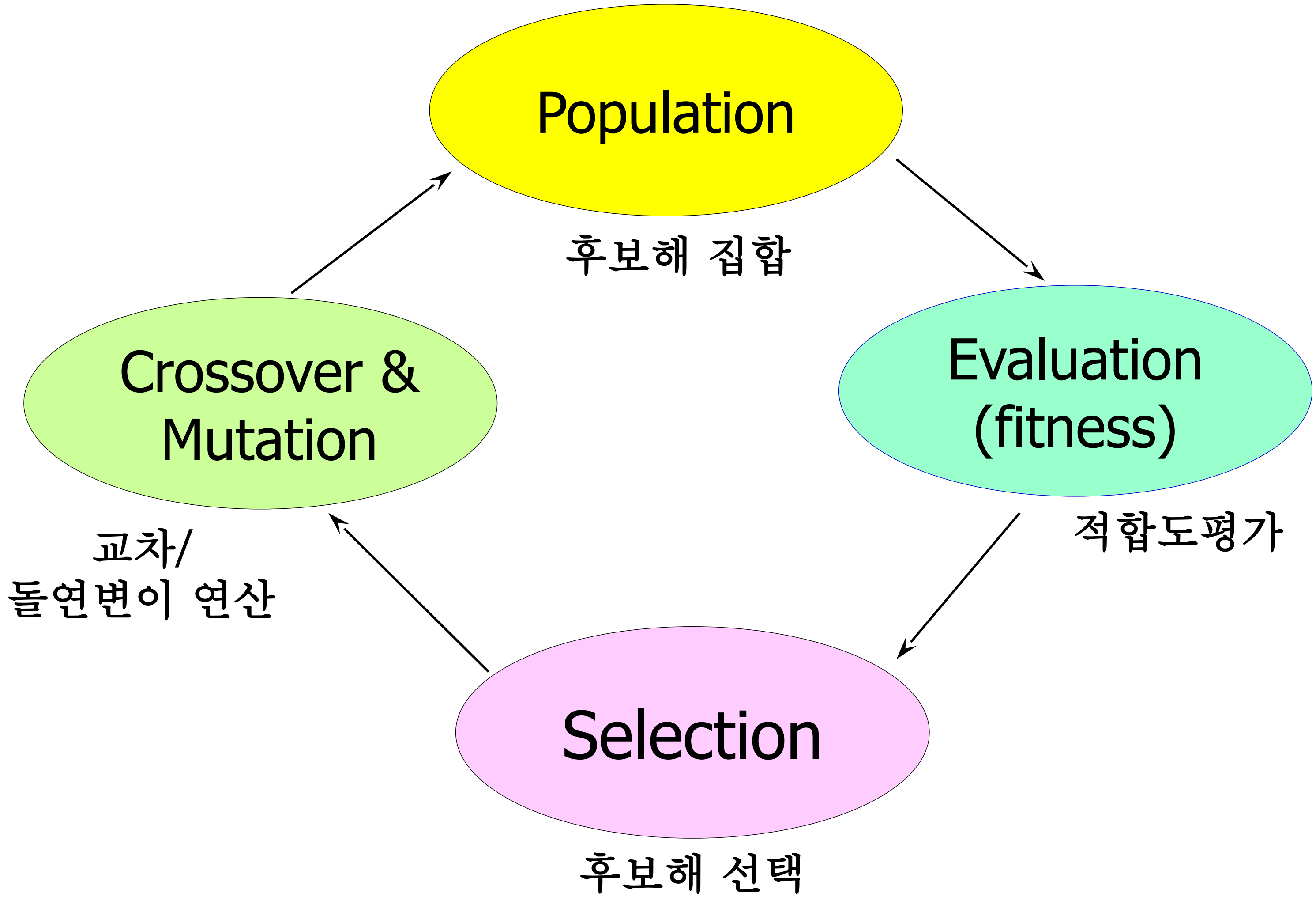
분기 한정 vs 백트래킹

- TSP를 위한 백트래킹 알고리즘이 방문한 상태 공간 트리의 노드 수는 54개이나 분기 한정 알고리즘은 22개
- 최적화 문제의 해를 탐색하는 데는 분기 한정 기법이 백트래킹 기법보다 훨씬 우수한 성능을 보인다.
- 분기 한정 알고리즘은 한정값을 사용하여 최적해가 없다고 판단되는 부분은 탐색을 하지 않고 최선 우선 탐색

9.3 유전자 알고리즘

- 유전자 알고리즘 (Genetic Algorithm, GA)
  - 다윈의 진화론으로부터 창안된 해 탐색 알고리즘
  - '적자생존'의 개념을 최적화 문제를 해결하는데 적용

GA 사이클


Population
후보해 집합
Evaluation (fitness)
적합도평가
Selection
후보해 선택
Crossover & Mutation
교차/
돌연변이 연산

알고리즘

GeneticAlgorithm

1. 초기 후보해 집합 G_0을 생성
2. G_0의 각 후보해를 평가
3. t = 0
4. repeat
5. G_t로부터 G_{t+1}을 생성
6. G_{t+1}의 각 후보해를 평가
7. t = t + 1
8. until 종료 조건이 만족될 때까지
9. return G_t의 후보해 중에서 가장 우수한 해

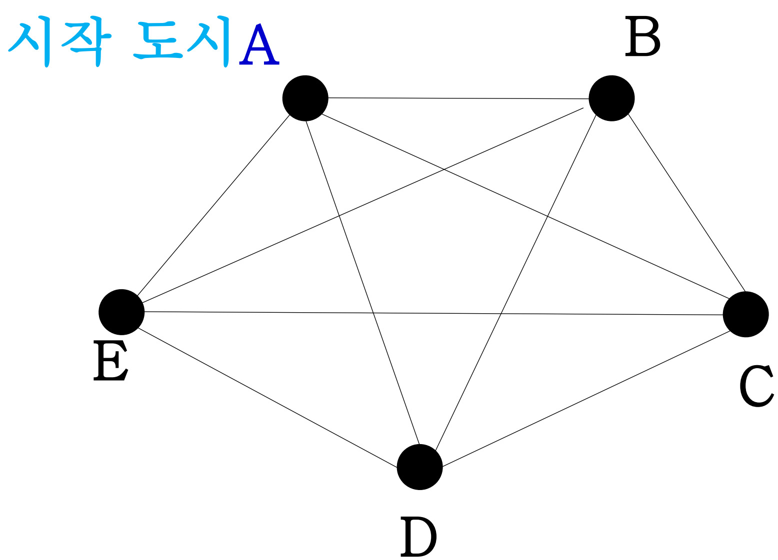
GeneticAlgorithm

- 여러 개의 해를 임의로 생성하여 이들을 초기 세대 (generation) G_0로 놓고
- repeat-루프에서 현재 세대의 해로부터 다음 세대의 해를 생성해가며,
- 루프가 끝났을 때의 마지막 세대에서 가장 우수한 해를 반환

- 이 해들은 repeat-루프의 반복적인 수행을 통해서 최적해 또는 최적해에 근접한 해가 될 수 있으므로 후보해 (candidate solution)라고 부른다.

후보해

- TSP: 5개의 도시 (A, B, C, D, E), 시작 도시 = A
- TSP는 시작 도시에서 출발하여 모든 다른 도시를 1번씩만 방문하고 시작 도시로 돌아와야 하므로, ABCDEA, ACDEBA, AECDBA 등이 후보해



후보해의 수

- 시작 도시를 제외한 4개의 도시를 일렬로 나열하는 방법의 수: (5-1)! = 4! = 24
  - n개의 도시의 후보해 수 = (n-1)!

후보해의 평가

- ABCDEA의 값 =

  (A와 B 사이의 거리)

  \+ (B와 C 사이의 거리)

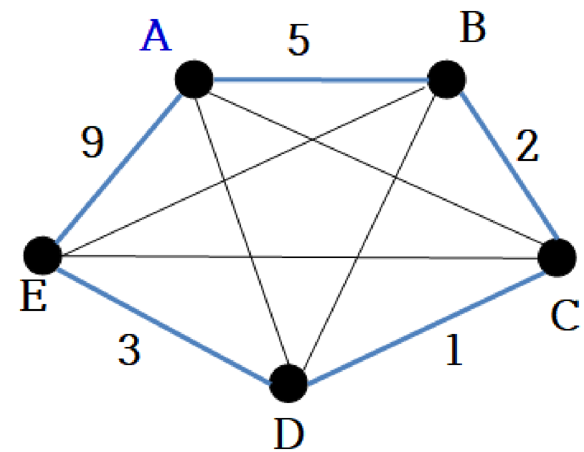
  \+ (C와 D 사이의 거리)

  \+ (D와 E 사이의 거리)

  \+ (E와 A 사이의 거리)

  = 5 + 2 + 1 + 3 + 9

  = 20



적합도

- 후보해의 값 = 후보해의 적합도(Fitness value)
- 후보해 중에서 최적해의 값에 근접한 적합도를 가진 후보해를 '우수한' 해라고 부른다.

GA 연산

- 선택 (selection) 연산
- 교차 (crossover) 연산
- 돌연변이 (mutation) 연산

1. 선택 연산

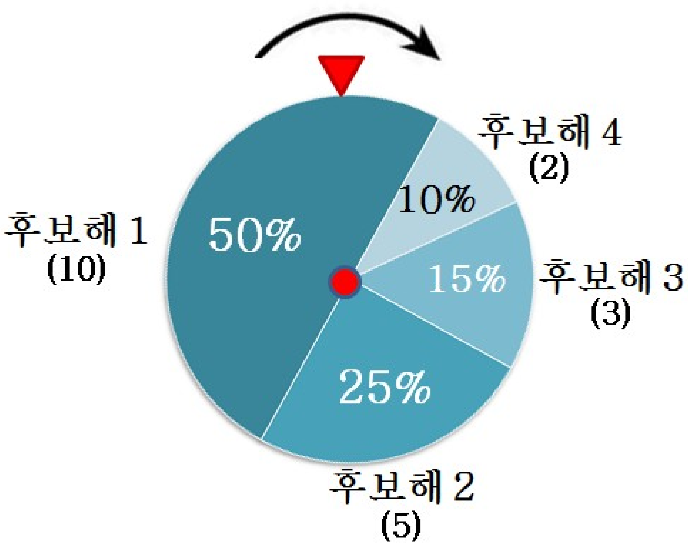
- 현재 세대의 후보해 중에서 우수한 후보해를 선택하는 연산

- 현재 세대에 n개의 후보해가 있으면
  - 이들 중에서 우수한 후보해는 중복되어 선택될 수 있고, 적합도가 상대적으로 낮은 후보해들은 선택되지 않을 수도 있다.

- 이렇게 선택된 후보해의 수는 n개로 유지
  - 이러한 선택은 '적자생존' 개념을 모방한 것

룰렛 휠 선택

- 룰렛 휠 (roulette wheel) 방법
  - 각 후보해의 적합도에 비례하여 원반의 면적을 할당하고, 원반을 회전시켜서 원반이 멈추었을 때 핀이 가리키는 후보해를 선택
  - 면적이 넓은 후보해가 선택될 확률이 높다.
- 각 후보해의 적합도
  - 후보해 1의 적합도: 10
  - 후보해 2의 적합도: 5
  - 후보해 3의 적합도: 3
  - 후보해 4의 적합도: 2



룰렛 휠

- 각 후보해의 원반 면적
  - (후보해의 적합도 / 모든 후보해의 적합도의 합)에 비례

- 예제에서 모든 적합도의 합이 20 = (10 + 5 + 3 + 2) 이므로,
  - 후보해 1의 면적은 10/20 = 50%
  - 후보해 2의 면적은 5/20 = 25%
  - 후보해 3의 면적은 3/20 = 15%
  - 후보해 4의 면적은 2/20 = 10%

- 현재 4개의 후보해가 있으므로, 원반을 4번 돌리고 회전이 멈추었을 때 핀이 가리키는 후보해를 각각 선택

토너먼트 선택 (Tournament Selection)

1. 후보해 집합(population)에서 k개의 후보해를 랜덤하게 선택한다.

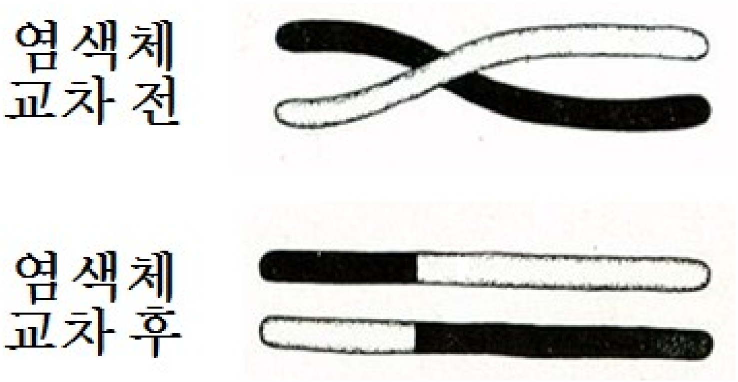
2. 선택된 k개 중에서 가장 적합도가 우수한 해를 선택한다.

3. 선택 후 k개를 모두 후보해 집단에 넣는다.

- 이진 토너먼트 선택(Binary Tournament Selection)
  - k = 2, 가장 많이 쓰이는 선택 연산

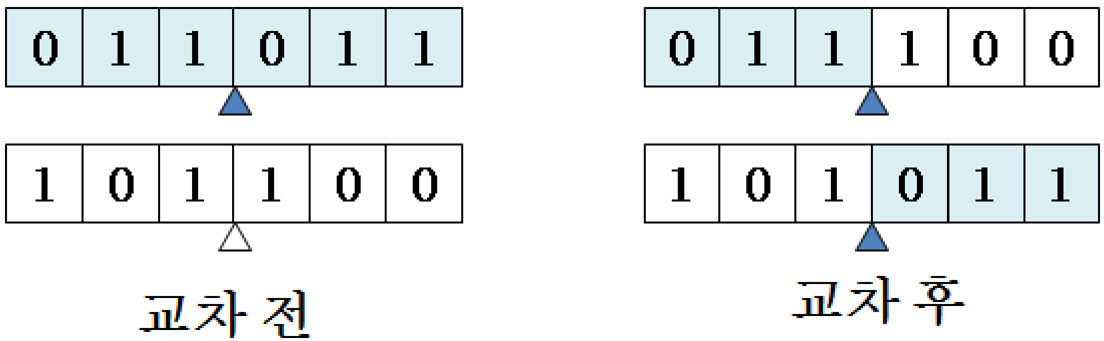
2. 교차 연산

- 선택 연산을 수행한 후의 후보해 사이에 수행되는데, 이는 염색체가 교차하는 것을 모방



1-점 (point) 교차 연산

- 랜덤하게 교차할 점을 선택한 후, 두 개의 후보해를 교차점을 기준으로 뒷부분을 서로 교환



- 후보해가 길면, 여러 개의 교차점을 랜덤하게 정하여 교차 연산을 할 수도

교차 연산

- 교차 연산의 목적
  - 선택 연산을 통해서 얻은 우수한 후보해보다 우수한 후보해를 생성하기 위해

- 교차율 (Crossover Rate)
  - 문제에 따라 교차 연산을 수행할 후보해의 수를 조절하는데, 이를 교차율이라고 한다.
  - 일반적으로 교차율은 0.2 ~ 1.0 범위에서 정한다.

3. 돌연변이 연산

- 교차 연산 수행 후에 돌연변이 연산 수행

- 돌연변이 연산
  - 아주 작은 확률로 후보해의 일부분을 임의로 변형시킨다.
  - 이 확률을 돌연변이율 (Mutation Rate)이라고 하며, 일반적으로 (1/PopSize) ~ (1/Length)의 범위에서 사용
    - PopSize란 모집단 크기 (Population Size)로서 한 세대의 후보해의 수
    - Length란 후보해를 이진 표현으로 했을 경우의 bit 수
  - 돌연변이가 수행된 후에 후보해의 적합도가 오히려 나빠질 수도

돌연변이 연산 예제

- 두 번째 bit가 0에서 1로 돌연 변이된 것을 보여준다.

1	0	1	0	1	1

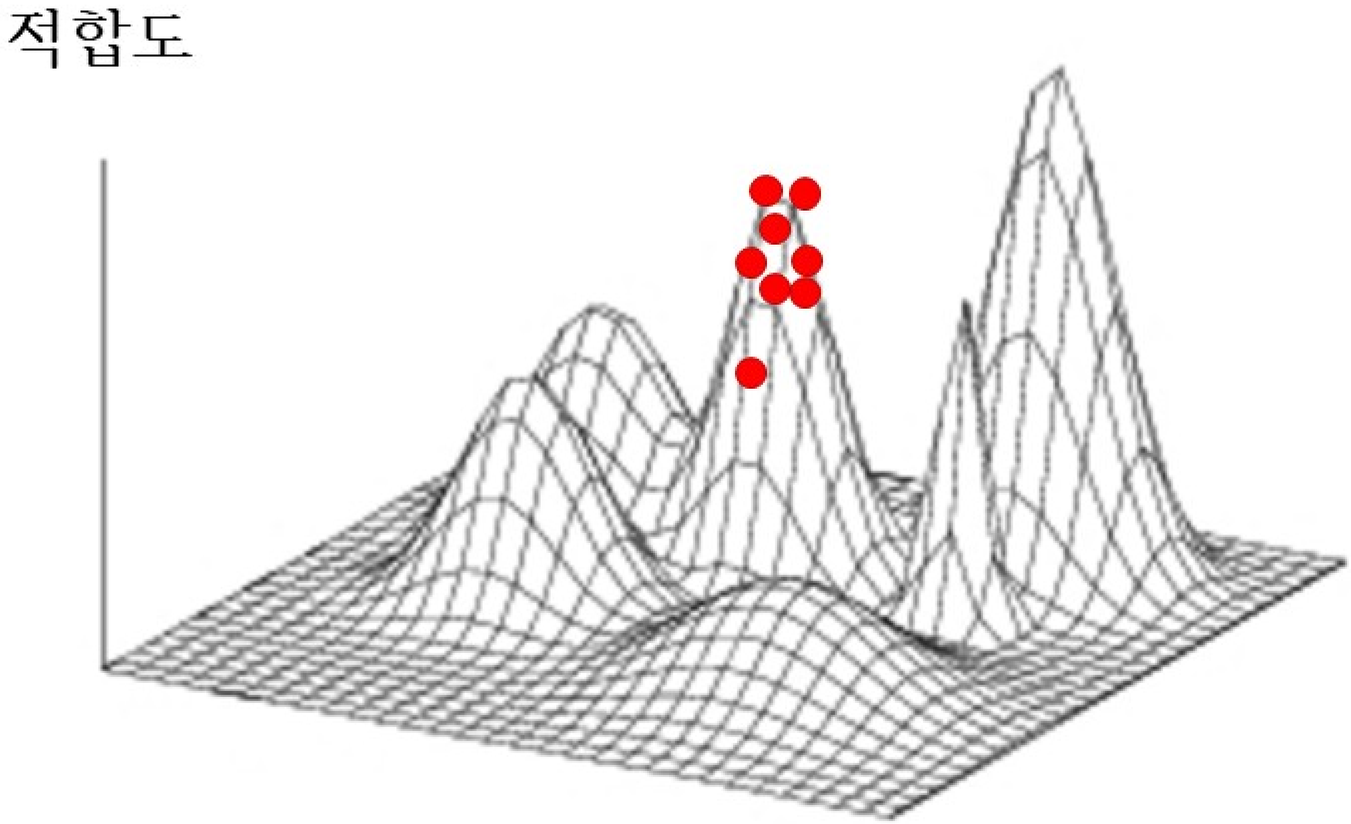
돌연변이 전

1	1	1	0	1	1

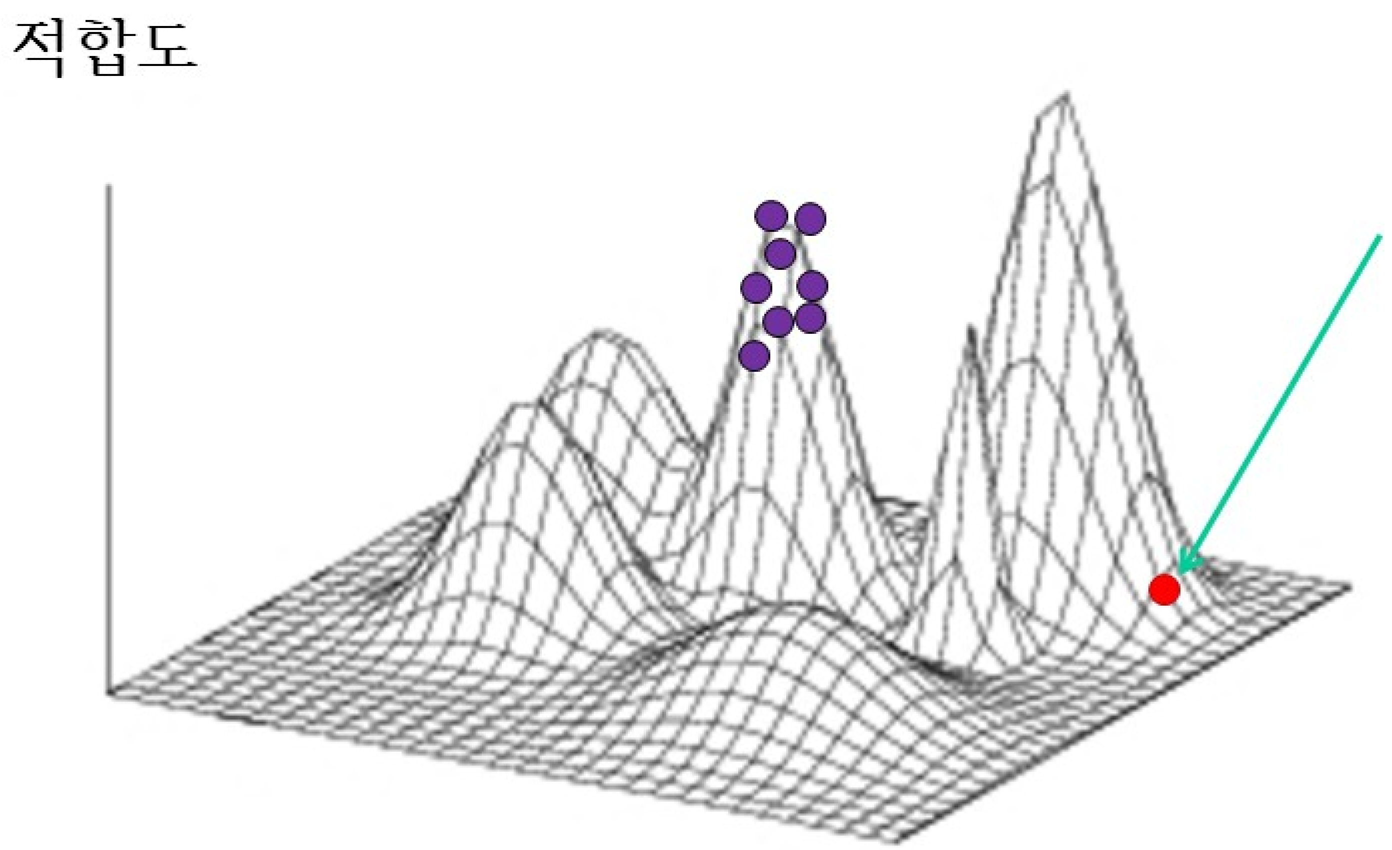
돌연변이 후

- 돌연변이 연산의 목적
  - 다음 세대에 돌연변이가 이루어진 후보해와 다른 후보해를 교차 연산함으로써 이후 세대에서 매우 우수한 후보해를 생성하기 위해

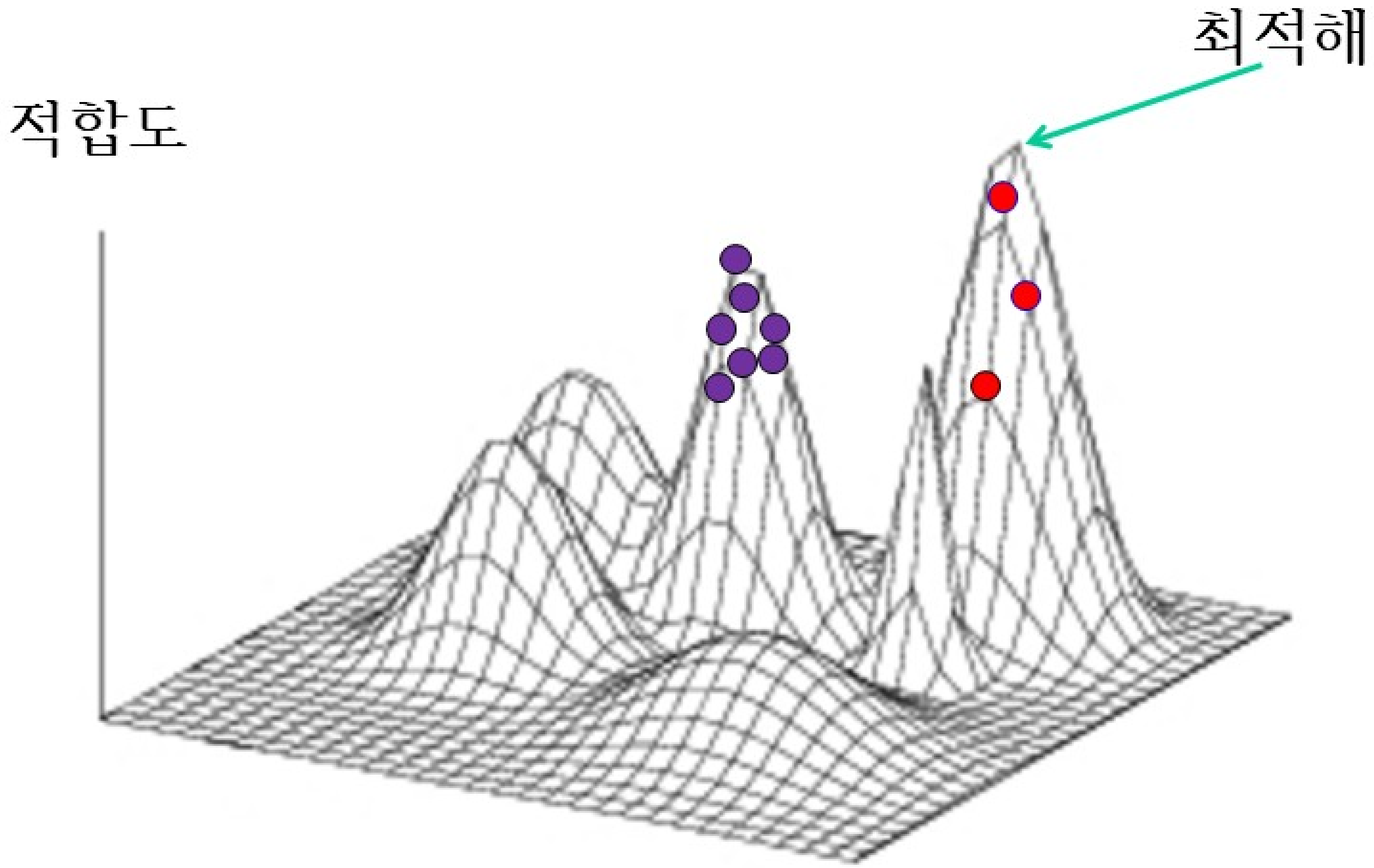
돌연변이 연산 역할



돌연변이 연산 후



여러 세대가 지난 후



종료 조건

- 유전자 알고리즘이 항상 최적해를 찾는다는 보장이 없기 때문에 종료 조건은 일정하지 않다.
  - 일반적으로 알고리즘을 수행시키면서 더 이상 우수한 해가 출현하지 않으면 알고리즘을 종료

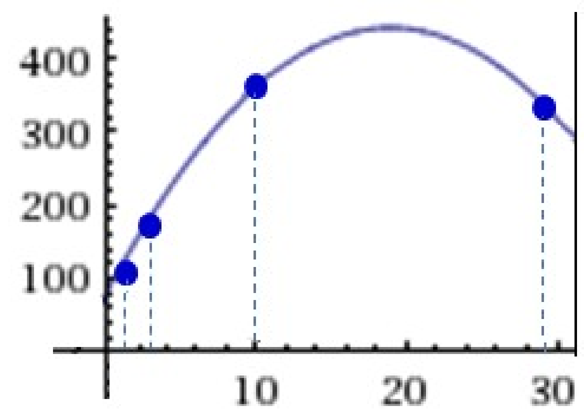
GeneticAlgorithm 수행 과정

- 다음의 2차 함수에 대해 유전자 알고리즘으로 $0 \le x \le 31$ 구간에서 최대값을 찾아보자.

$$f(x) = -x^2 + 38x + 80$$

- 초기 세대를 구성하는 후보해들을 결정한다.
  - 먼저 한 세대의 후보해 수를 4로 정하고, 0~31에서 랜덤하게 4개의 후보해인 1, 29, 3, 10을 선택하였다고 가정

- 각 후보해의 적합도
  - $f(1) = -(1)^2 + 38(1) + 80 = 117$
  - $f(29) = 341$
  - $f(3) = 185$
  - $f(10) = 360$



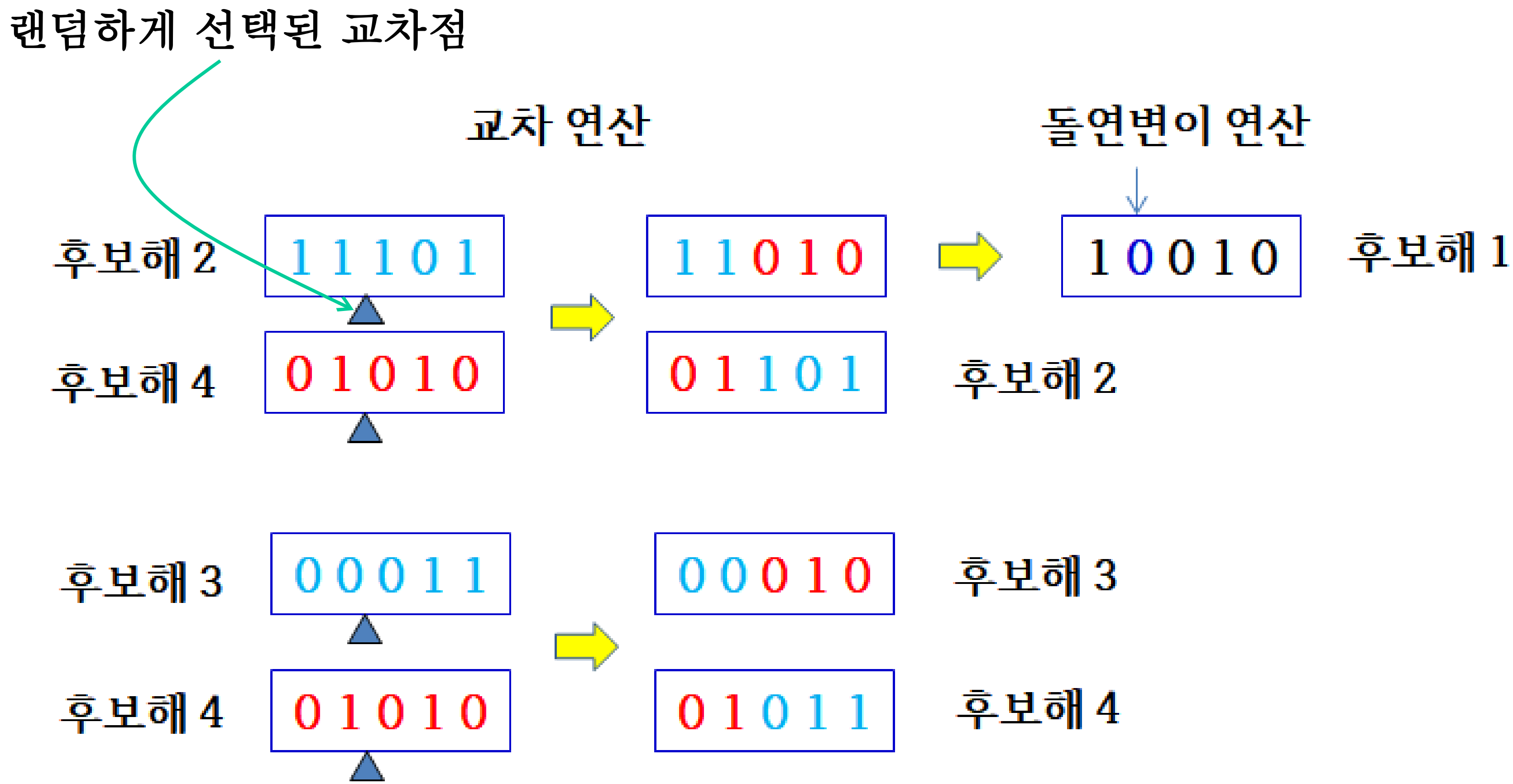
GeneticAlgorithm 수행 과정

후보해	2진 표현	x	적합도 f(x)	원반 면적 (%)
1	0 0 0 0 1	1	117	12
2	1 1 1 0 1	29	341	34
3	0 0 0 1 1	3	185	18
4	0 1 0 1 0	10	360	36
계			1,003	100
평균			250.75	

- 초기 세대의 평균 적합도는 250.75

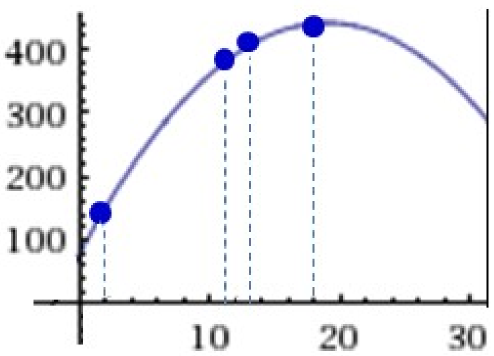
GeneticAlgorithm 수행 과정

- 선택 연산
  - 룰렛 휠 선택 방법으로 후보해 4는 2번 선택, 후보해 2와 3은 각각 1번 선택, 후보해 1은 선택이 안되었다고 가정
- 교차 연산
  - 후보해 4가 2개이므로, 후보해 2와 4를 짝짓고, 후보해 3과 4를 짝지어 아래와 같이 교차 연산을 수행
  - 단, 1점-교차 연산을 위해 아래와 같이 임의의 교차점이 선택되었다고 가정
- 돌연변이 연산
  - 교차 연산 후에 후보해 1의 왼쪽에서 두 번째 bit가 돌연변이가 되어 '1'에서 '0'으로 바뀌었다고 가정
  - 다른 후보해는 교차 연산 후와 동일


랜덤하게 선택된 교차점
교차 연산
돌연변이 연산
후보해 2
1 1 1 0 1
후보해 4
0 1 0 1 0
1 1 0 1 0
0 1 1 0 1
1 0 0 1 0
후보해 1
후보해 2
후보해 3
0 0 0 1 1
후보해 4
0 1 0 1 0
0 0 0 1 0
0 1 0 1 1
후보해 3
후보해 4

두 번째 세대의 후보해에 대한 적합도

- 평균 적합도가 343.5로 첫 세대의 250.75보다 많이 향상됨



후보해	2진 표현	x	적합도 f(x)	원반 면적 (%)
1	1 0 0 1 0	18	440	32
2	0 1 1 0 1	13	405	29
3	0 0 0 1 0	2	152	11
4	0 1 0 1 1	11	377	27
계			1,374	100
평균			343.5	

알고리즘 종료

- 충분한 세대를 거쳐 repeat-루프를 더 수행하여 후보해의 적합도가 변하지 않으면 알고리즘을 종료
- 후보해 중에서 가장 적합도가 높은 후보해를 리턴

TSP를 위한 GeneticAlgorithm

- 여행자 문제를 해결할 때 GeneticAlgorithm을 적용하기 위해 사용되는 2가지의 교차 연산
  - 2점 교차 연산
  - 사이클 교차 연산

- 여행자 문제의 후보해
  - 시작 도시부터 각 도시를 중복없이 나열하여 만들어진다.

2점-교차 연산

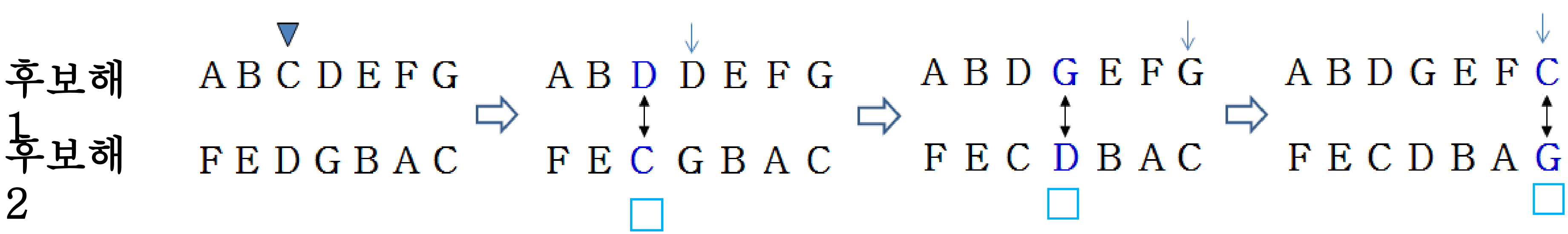
– 임의의 2점을 정한 후, 가운데 부분을 서로 교환
– 이후 중복되는 도시(점선 박스 내의 도시)를 현재 후보해에 없는 도시로 차례로 바꾼다.

후보해 1	A B | C D E | F G H	A B | H B A | F G H	E D | H B A | F G C
후보해 2	G D | H B A | F E C	G D | C D E | F E C	G B | C D E | F A H

– 후보해 1에 대해 가운데 부분을 제외한 부분에 있는 H, B, A를 각각 C, D, E로 바꾸고
– 후보해 2에 대해 가운데 부분을 제외한 부분에 있는 C, D, E 를 각각 H, B, A 로 바꾼다.

사이클 교차 연산

- 후보해 1에서 임의의 도시 C를 선택한 후, C와 같은 위치에 있는 후보해 2의 도시 D와 바꾼다.
- 바꾼 후에는 후보해 1에는 C가 없고, D가 2개 존재한다.
- 이를 해결하기 위해 후보해 1에 원래부터 있었던 D를 후보해 2에 D와 같은 위치에 있는 G와 바꾼다.
- 이렇게 반복하여 C가 후보해 2로부터 후보해 1로 바뀌게 되면 교차 연산을 마친다.



다양한 실험 필요

- 유전자 알고리즘은 대부분의 경우 실제로 적지 않은 실험이 필요
- 주어진 문제에 대해서 모집단의 크기, 교차율, 돌연변이율 등과 같은 파라미터가 다양한 실험을 통해서 조절되어야
- repeat-루프의 종료 조건도 실험을 통해서 결정할 수밖에 없다.
- 또한 다양한 선택 연산과 교차 연산 중에서 어떤 연산이 주어진 문제에 적절한지도 많은 실험을 통해서 결정해야

유전자 알고리즘 특징

- 문제의 최적해를 알 수 없고, 기존의 어느 알고리즘으로도 해결하기 어려운 경우에, 최적해에 가까운 해를 찾는데 매우 적절한 알고리즘

- 유전자 알고리즘이 최적해를 반드시 찾는다는 보장은 없으나 대부분의 경우 매우 우수한 해를 찾는다.

Applications

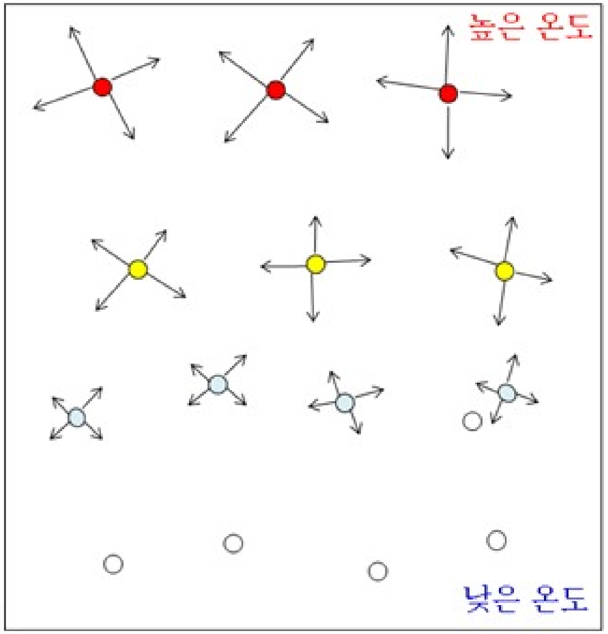
- 유전자 알고리즘은 통 채우기, 작업 스케줄링, 차량 경로, 배낭 문제 등과 같은 NP-완전 문제를 해결하는 데 활용
- 로봇 공학
- 기계 학습 (Machine Learning)
- 신호 처리 (Signal Processing)
- 반도체 설계
- 항공기 디자인
- 통신 네트워크
- 패턴 인식
- 그 외에도 경제, 경영, 환경, 의학, 음악, 군사 등과 같은 다양한 분야에서 최적화 문제를 해결하는데 활용

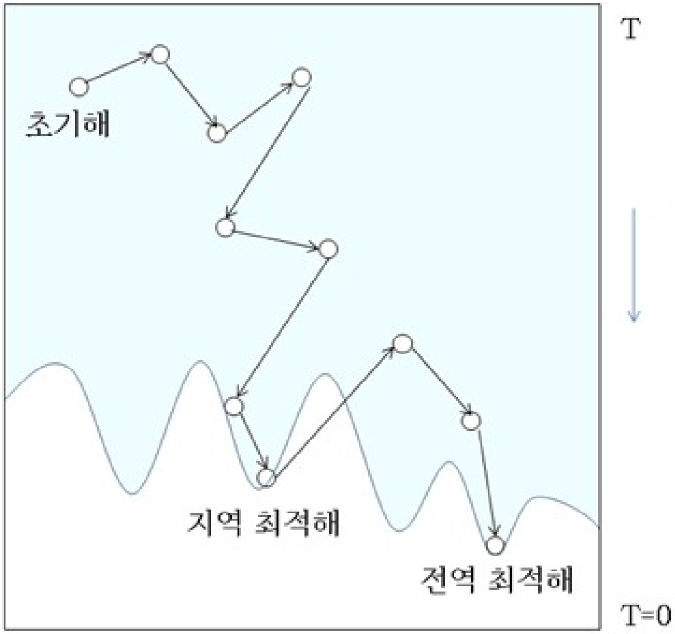
9.4 모의 담금질 기법

- 모의 담금질(Simulated Annealing) 기법은 높은 온도에서 액체 상태인 물질이 온도가 점차 낮아지면서 결정체로 변하는 과정을 모방한 해 탐색 알고리즘
- 용융 상태에서는 물질의 분자가 자유로이 움직이는데 이를 모방하여, 해를 탐색하는 과정도 특정한 패턴 없이 이루어진다.
- 온도가 점점 낮아지면 분자의 움직임이 점점 줄어들어 결정체가 되는데, 해 탐색 과정도 이와 유사하게 점점 더 규칙적인 방식으로 이루어진다.

이웃해

- 이러한 방식으로 해를 탐색하려면, 후보해에 대해 이웃하는 해 (이웃해)를 정의하여야
- 아래의 오른쪽 그림에서 각 점은 후보해이고 아래쪽에 위치한 해가 위쪽에 있는 해보다 우수한 해이다. 또한 2개의 후보해 사이의 화살표는 이 후보해들이 서로 이웃하는 관계임을 나타낸다.





탐색 과정

- 높은 T에서의 초기 탐색은 최솟값을 찾는데도 불구하고 확률 개념을 도입하여 현재 해의 이웃해 중에서 현재 해보다 '나쁜' 해로 (위 방향으로) 이동하는 자유로움을 보일 수도 있다.
- T가 낮아지면서 점차 탐색은 아래 방향으로 향한다.
  - T가 낮아질수록 위 방향으로 이동하는 확률이 점차 작아진다.
- 그림에서 처음 도착한 골짜기 (지역 최적해, local optimum)에서 더 이상 아래로 탐색할 수 없는 상태에 이르렀을 때 '운 좋게' 위 방향으로 탐색하다가 전역 최적해 (global optimum)를 찾은 것을 보여준다.

모의 담금질 기법의 특성

- 유전자 알고리즘과 마찬가지로 모의 담금질 기법도 항상 전역 최적해를 찾아준다는 보장은 없다.
- 모의 담금질 기법의 또 하나의 특징은 하나의 초기해로부터 탐색이 진행된다는 것이다. 반면에 유전자 알고리즘은 여러 개의 후보해를 한 세대로 하여 탐색을 수행

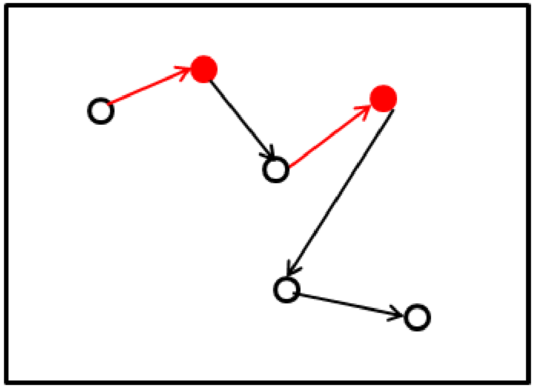
알고리즘

SimulatedAnnealing

```
1. 임의의 후보해 s를 선택
2. 초기 T를 정한다.
3. repeat
4.    for i = 1 to k_T      // k_T는 T에서의 for-루프 반복 횟수
5.          s의 이웃해 중에서 랜덤하게 하나의 해 s'를 선택
6.          d = (s'의 값) - (s의 값)
7.          if  d  <  0       // 이웃해인 s'가 더 우수한 경우
8.               s ← s'
9.           else             // s'가 s보다 우수하지 않은 경우
10.              q ← (0,1) 사이에서 랜덤하게 선택한 수
11.              if ( q < p )   s ← s'    // p는 자유롭게 탐색할 확률
12.    T ← αT   // 1보다 작은 상수 □를 T에 곱하여 새로운 T를 계산
13. until  종료 조건이 만족될 때까지
14. return  s
```

자유롭게 탐색할 확률 p

- Line 9~11: s'가 s보다 우수하지 않더라도 0~1 사이에서 랜덤하게 선택한 수 q가 확률 p보다 작으면, s'가 현재 해인 s가 될 기회를 준다.
  - 이 기회가 그림에서 최소값을 찾는데도 불구하고 위쪽에 위치한 이웃해로 탐색을 진행한다.
  - p는 자유롭게 탐색할 확률



냉각율

- Line 12: T를 일정 비율 □로 감소시킨다. 실제로 0.8 ≤ α ≤ 0.99 범위에서 미리 정한 냉각율 □(cooling ratio)를 T에 곱하여 새로운 T를 계산
  - 일반적으로 0.99에 가까운 수로 선택

확률 p 조절

- 모의 담금질 기법은 T가 높을 때부터 점점 낮아지는 것을 확률 p에 반영시켜서 초기에는 탐색이 자유롭다가 점점 규칙적이 되도록 한다.
- 확률 p는 T에 따라서 변해야
  - T가 높을 땐, p를 크게 하고,
  - T가 0이 되면, p를 0으로 만들어서 나쁜 이웃해 s'가 s가 되지 못하도록 한다.
- s'와 s의 값의 차이 d에 따라 p 조절
  - d 값이 크면, p를 작게 하고,
  - d 값이 작으면, p를 크게 한다.
- 이렇게 하는 이유는 값의 차이가 큼에도 불구하고 p를 크게 하면 그 동안 탐색한 결과가 무시되어 랜덤하게 탐색하는 결과를 낳기 때문

확률 p

- 두 가지 요소를 종합한 확률 p

$$p = 1 / e^{d/T} = e^{-d/T}$$

- T는 큰 값에서 0까지 변하고, d는 s'와 s의 값의 차이

이웃해 정의

- TSP의 이웃 해 정의 3가지 예
  1. 삽입 (Insertion)
  2. 교환 (Switching)
  3. 반전 (Inversion)

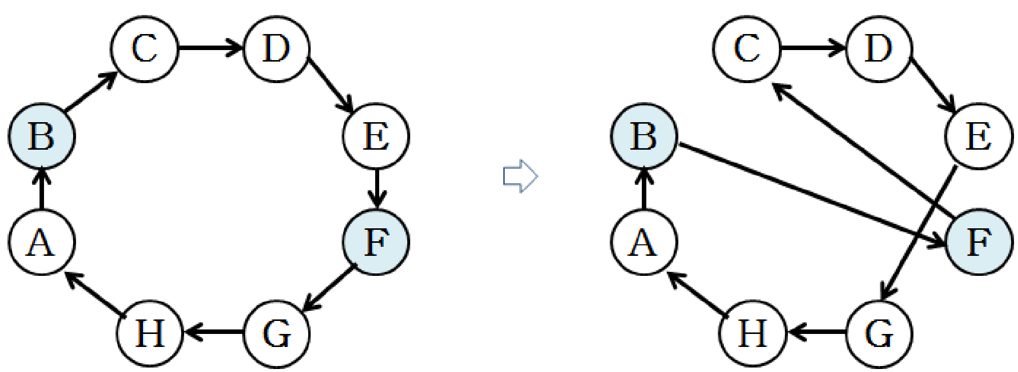
삽입 (Insertion)

- 2개의 도시를 랜덤하게 선택한 후에, 두 번째 도시를 첫 번째 도시 옆으로 옮기고, 두 도시 사이의 도시들은 오른쪽으로 1칸씩 이동
- 도시 B와 F가 랜덤하게 선택되었다면, F가 B의 바로 오른쪽으로 이동한 후, B와 F 사이의 C, D, E를 각각 오른쪽으로 1칸씩 이동

A	B	C	D	E	F	G	H

⇨

A	B	F	C	D	E	G	H



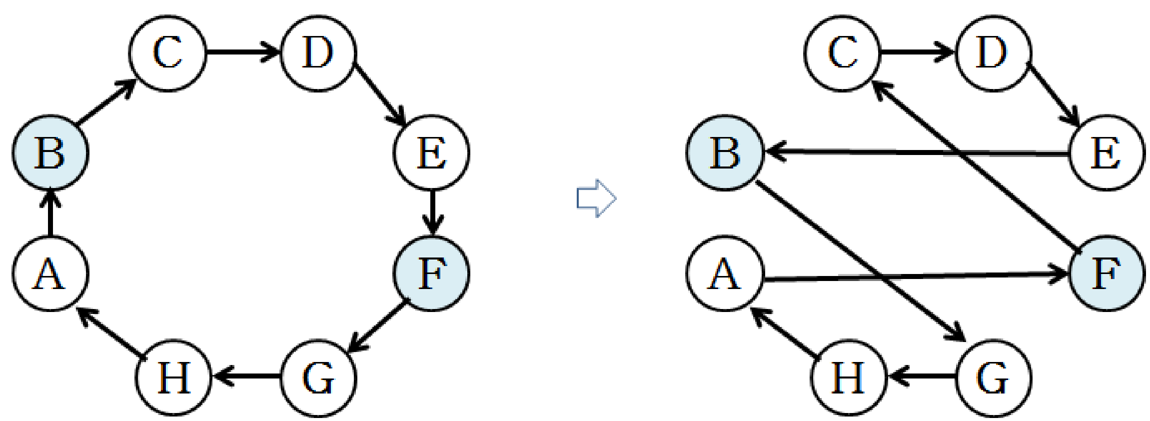
교환 (Switching)

- 2개의 도시를 랜덤하게 선택한 후에, 그 도시들의 위치를 서로 바꾼다.

- 도시 B와 F가 랜덤하게 선택되었다면, B와 F의 자리를 서로 바꾼다.

A	B	C	D	E	F	G	H

A	F	C	D	E	B	G	H



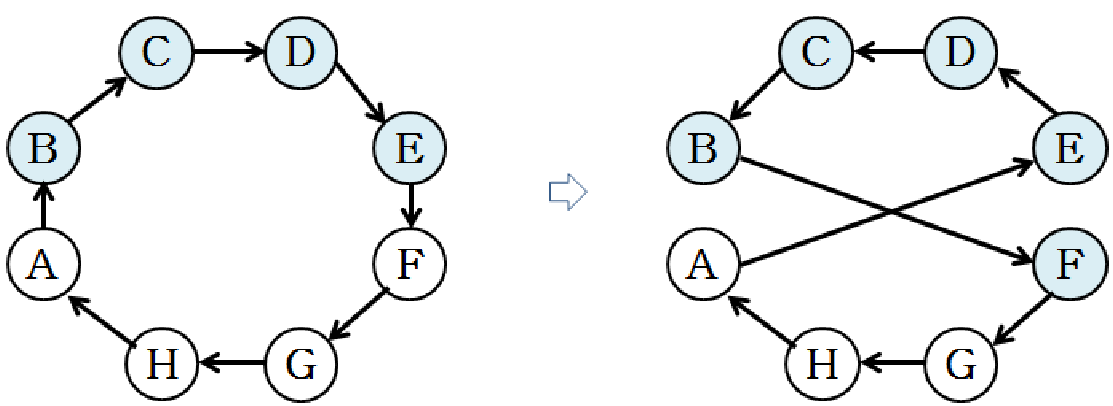
반전 (Inversion)

- 2개의 도시를 랜덤하게 선택한 후에, 그 두 도시 사이의 도시를 역순으로 만든다. 단, 선택된 두 도시도 반전에 포함시킨다.
- 도시 B와 E가 랜덤하게 선택되었다면, [B C D E]가 역순으로 [E D C B]로 바뀐다.

A	B	C	D	E	F	G	H

⇨

A	E	D	C	B	F	G	H



Applications

- 반도체 회로 설계
- 유전자 배열
- 단백질 구조 연구
- 경영 분야의 재고 계획
- 원자재 조달
- 상품의 생산 및 유통
- 운송 분야의 스케줄링
- 건축 분야의 빌딩 구획 및 배치 (Building Layout)
- 항공기 디자인
- 복합 물질 모델링
- 금융 분야의 은행의 재무 분석 등 매우 광범위하게 활용

요약

- 백트래킹 (Backtracking) 기법은 해를 찾는 도중에 '막히면' 되돌아가서 다시 해를 찾아 가는 기법으로 상태 공간 트리에서 깊이 우선 탐색 (Depth First Search)으로 해를 찾는 알고리즘
- 백트래킹 기법의 시간 복잡도는 상태 공간 트리의 노드 수에 비례하고, 이는 모든 경우를 다 검사하여 해를 찾는 완전 탐색의 시간 복잡도와 같다. 그러나 일반적으로 백트래킹 기법은 '가지치기'하므로 완전 탐색보다 훨씬 효율적이다.
- 분기 한정 기법은 상태 공간 트리의 각 노드(상태)에 특정한 값(한정값)을 부여하고, 노드의 한정값을 활용하여 가지치기를 함으로서 백트래킹 기법보다 빠르게 해를 찾는다.

요약

- 분기 한정 기법에서는 가장 우수한 한정값을 가진 노드를 먼저 탐색하는 최선 우선 탐색 (Best First Search)으로 해를 찾는다.

- 유전자 알고리즘은 다윈의 진화론으로부터 고안된 해 탐색 알고리즘이다. '적자생존' 개념을 최적화 문제를 해결하는데 적용한 것이다.

- 유전자 알고리즘은 여러 개의 해를 임의로 생성하여 이들에 대해 선택, 교차, 돌연변이 연산을 반복 수행하여 마지막에 가장 우수한 해를 리턴

요약

- 유전자 알고리즘은 문제의 최적해를 알 수 없고, 기존의 어느 알고리즘으로도 해결하기 어려운 경우에, 최적해에 가까운 해를 찾는데 매우 적절한 알고리즘

- 모의 담금질 (Simulated Annealing) 알고리즘은 높은 온도에서 액체 상태인 물질이 온도가 점차 낮아지면서 결정체로 변하는 과정을 모방한 해 탐색 알고리즘

- 유전자 알고리즘과 마찬가지로 모의 담금질 기법도 항상 전역 최적해를 찾아준다는 보장은 없다.