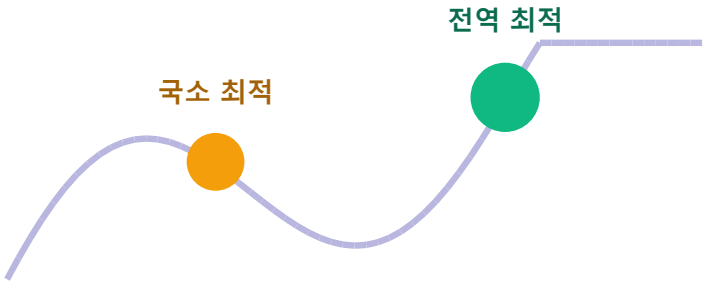
그림으로 쉽게 설명하는 파이썬 알고리즘

CHAPTER 11

휴리스틱 알고리즘

Heuristic Algorithms

완벽한 지도가 없다면, 나침반과 직관을 믿고 나아가라



언덕을 오르되, 갇히지 않는 법

보장은 없다, 대신 빠르다

CONTENTS

이번 장에서 배울 내용

11.1
왜 다시 직관인가
근사의 한계와 휴리스틱

11.2
언덕 오르기
반복적 개선의 기본

11.3
TSP 언덕 오르기
swap으로 경로 다듬기

11.4
A* 알고리즘
f(n) = g(n) + h(n)

11.5
메타 휴리스틱
탐색과 활용의 균형

11.6
모의 담금질
온도로 무작위성을 조절한다

11.7
TSP 모의 담금질
2-opt 구간 뒤집기

11.8
유전 알고리즘
적자생존 · 교배 · 돌연변이

11.9
TSP 유전 알고리즘
순서 교배(OX)가 핵심

11.10
자연 영감 알고리즘
PSO와 ACO

11.11
정리 및 비교
선택 가이드

10장에서 내려놓은
수학적 보장 대신,
경험과 직관으로
실전을 돌파한다.

11.1

왜 다시 직관인가?

11.1

이론이 멈추는 곳에서 산업 현장은 계속 돌아간다.
증명을 기다릴 시간이 없을 때 무엇을 쓸 것인가.

근사 알고리즘의 한계 — 이론이 멈추는 곳

증명 자체가 불가능하다

현실의 복잡한 공학적 최적화 문제는 수학적으로 모델링하기조차 벅차다. 근사 비율 증명은 최고의 수학자들에게도 난제다.

보장된 비율이 너무 나쁘다

"최악의 경우 정답의 100배 이내를 보장한다"는 수준이라면, 비용 절감이 절실한 기업에게는 사실상 아무 의미가 없다.

산업 현장에는 수년간 증명에 매달릴 시간이 없다

당장 돌아가는 솔루션이 필요하다. 10장에서 붙였던 수학적 보증서를 떼어 내는 대신, 실행 가능성과 속도를 얻는 선택이 남아 있다.

그래서 다시 직관으로 돌아온다

보장이 없다는 것은 약점이지만, 어떤 문제에도 붙일 수 있다는 것은 강점이다.

휴리스틱의 정의 — 경험과 직관의 힘

Heuristics — 그리스어 heuriskein(발견하다)에서 왔다. 최적해를 보장하지는 않지만, 현실적인 시간 안에 "꽤 괜찮은" 해를 찾아내는 경험적이고 직관적인 방법이다.

근사 알고리즘

"반드시 이 오차 범위 안에 든다"는 계약서를 쓰는 것

휴리스틱

"써 보니까 대체로 잘 맞더라"는 입소문을 믿는 것

VS

계약서가 없다고 해서 쓸모없는 것은 아니다 — 계약서를 쓸 수 없는 문제가 훨씬 많다.

휴리스틱의 예 — 열쇠 찾기

완전 탐색 (brute-force)

집 안의 모든 바닥을 1cm² 단위로 돋보기를 대고 확인한다. 시간은 엄청 걸리지만 반드시 찾아낸다.

100% 보장 · 지수 시간

휴리스틱 (heuristic)

현관 신발장, 거실 탁자, 침실 옷걸이처럼 유력한 곳만 빠르게 훑는다. 몇 분이면 끝난다.

보장 없음 · 매우 빠름

"공짜 점심은 없다" (No Free Lunch) 정리

모든 종류의 문제에 대해 평균적으로 가장 좋은 성능을 내는 만능 알고리즘은 존재하지 않는다. 어떤 문제에서 뛰어난 방법은 다른 문제에서 반드시 그만큼 뒤처진다.

그래서 "문제의 특성에 맞는 휴리스틱을 고르는 일" 자체가 실력이 된다

이 장에서 배우는 것은 만능 해법이 아니라, 고를 수 있는 도구의 목록과 각각의 성격이다.

11.2

지역 탐색과 언덕 오르기

11.2

이미 만들어진 해를 조금씩 고쳐 나간다.
단, 더 올라갈 곳이 없어 보일 때가 함정이다.

§11.2 언덕 오르기

욕심쟁이의 직관 (greedy heuristic)

대표 예시 — TSP의 "가장 가까운 이웃(nearest neighbor)". 5장에서 이미 만난 방법이다.

방식 현재 위치에서 가장 가까운, 아직 방문하지 않은 도시로 이동하기를 반복한다.

속도 $O(N^2)$로 매우 빠르게 경로를 완성한다. $O(N^2)$

쓰임새 구현이 간단하고, "그럭저럭 쓸 만한" 첫 번째 해를 얻는 데 유용하다.

근시안적 결정의 위험성

당장 가까운 도시로만 이동하다 보면, 마지막에 남은 도시들이 서로 멀리 떨어져 있어 최종 경로가 엄청나게 길어질 수 있다.

그래서 "만들고 끝"이 아니라 "만든 뒤에 고치는" 전략이 필요해진다.

언덕 오르기 (hill climbing)

이미 만들어진 해를 조금씩 수정하여 더 좋은 해로 나아가는 "반복적 개선" 기법이다.

안개 낀 산맥의 비유

"일단 내 발밑 주변을 더듬어 보고, 지금 위치보다 조금이라도 경사가 높은 쪽으로 한 발자국 내딛는다. 더 이상 올라갈 곳이 없을 때까지 반복한다."

1

초기 해 생성

무작위 또는 그리디로 시작한다.

2

이웃 해 탐색

현재 해를 조금 수정해 후보를 만든다.

3

개선 여부 판단

더 좋으면 채택하고, 아니면 그대로 둔다.

④ 종료 조건 — 더 이상 개선이 없으면 멈춘다.

언덕 오르기 유사 코드

hill_climbing.pseudo

```
Algorithm DISCRETE_HILL_CLIMBING(start)

  current <- start

  while true:
    bestNeighbor <- NIL
    bestValue    <- -무한대

    for each neighbor in NEIGHBORS(current):
      value <- EVAL(neighbor)
      if value > bestValue:
        bestValue    <- value
        bestNeighbor <- neighbor

    if bestValue <= EVAL(current):
      return current            // 개선이 없으면 종료

    current <- bestNeighbor
```

NEIGHBORS(current)
현재 해를 조금 바꿔 만든 후보들. 이 정의가 알고리즘의 성격을 결정한다.

EVAL(x)
해의 좋음을 점수로 매기는 함수. TSP라면 경로 길이의 부호를 뒤집은 값이다.

최선의 이웃만 채택
이웃 전부를 보고 가장 좋은 하나로 이동한다. 이것을 steepest-ascent라 한다.

종료 조건
어떤 이웃도 현재보다 낫지 않으면 멈춘다 — 바로 이 지점이 국소 최적해다.

"더 좋은 이웃이 없다"는 것과 "여기가 전역 최적이다"는 전혀 다른 말이다.

§11.2 언덕 오르기

치명적 약점 — 국소 최적해의 함정

열심히 언덕을 올라 정상에 도착했다. 사방이 내리막길이다. "여기가 가장 높은 곳이구나!" 하지만 안개가 걷히고 보니 그저 동네 뒷산이었다. 진짜 정상은 골짜기를 지나야 한다.

국소 최적해

주변보다는 높지만 전체로 보면 낮은 봉우리에 갇힌다.

평원 (plateau)

주위가 모두 평평해 어느 쪽으로 가야 할지 방향을 잡을 수 없다.

능선 (ridge)

좁고 가파른 길에서는 한 발만 헛디뎌도 급격히 낮은 곳으로 떨어진다.

언덕 오르기와 딥러닝의 그라디언트 디센트

구분	언덕 오르기	그라디언트 디센트
상태 공간	이산적 (discrete)	연속적 수학 공간
이동 방법	이웃 상태를 하나씩 비교	기울기(gradient)로 방향 계산
문제 유형	최대화 — 높은 곳 탐색	최소화 — 손실 함수 최소화
갇히는 곳	국소 최적해	국소 최솟값 · 안장점

공통 철학 — 현재 위치에서 얻을 수 있는 정보만으로 점진적으로 더 나은 상태로 이동한다

이름과 분야는 달라도 구조가 같다. 딥러닝의 학습률 조절이나 모멘텀도, 결국 국소 최적을 벗어나기 위한 장치라는 점에서 이 장의 메타 휴리스틱과 문제의식을 공유한다.

그래서 딥러닝을 배운 사람에게 이 장은 낯설지 않다

언덕 오르기는 이산 공간판 경사 하강법이라고 생각하면 된다.

11.3

TSP 언덕 오르기

11.3

무작위 경로에서 출발해
두 도시를 바꿔 가며 조금씩 줄인다.

TSP 언덕 오르기 — 알고리즘 흐름

① 초기 해
무작위 순서로 도시를 배열해 초기 경로를 만든다.

② 이웃 생성
임의의 두 도시 위치를 swap해 새 경로를 만든다.

③ 평가
새 경로의 총 거리가 현재보다 짧은지 비교한다.

④ 선택
더 좋으면 채택하고, 아니면 원상복구한다 (swap-back).

```
RESULT

Initial   : 522.03
Iter    4 : 489.93
Iter   25 : 428.15
Iter   32 : 376.57
Iter   47 : 326.91
Iter  277 : 313.56
```

3000회 동안 개선 없음 → 종료
522에서 시작해 313까지 내려왔다. 그 뒤로는 어떤 swap도 더 나은 결과를 주지 못한다 — 국소 최적해에 도달한 것이다.

초반에는 빠르게 좋아지다가 점점 개선 폭이 줄어드는 것이 전형적인 패턴이다.

가장 가까운 이웃 vs 언덕 오르기

구분	가장 가까운 이웃 (greedy)	언덕 오르기 (hill climbing)
전략	구성 (construction)	**개선 (improvement)**
시작점	빈 경로에서 시작	완성된 경로에서 시작
방식	가장 가까운 미방문 도시를 연결	도시 순서를 swap하며 개선
수정 가능성	한 번 결정하면 변경 불가	계속 수정 가능
속도	매우 빠름	상대적으로 느림 (반복)
품질	보통	초기 해보다 확실히 나음

둘은 경쟁 관계가 아니라 이어 붙이는 관계다

가장 가까운 이웃으로 빠르게 초기 해를 만들고, 그 해를 언덕 오르기로 다듬는 조합이 실무에서 가장 흔하다.

다만 언덕 오르기만으로는 국소 최적해를 벗어날 수 없다 — 다음 절의 주제다.

11.4

A* 알고리즘

지금까지 쓴 비용과 앞으로 들 비용을 더한다.
나침반 하나가 탐색량을 통째로 줄인다.

11.4

A* 알고리즘 — 핵심 아이디어

$$f(n) = g(n) + h(n)$$

f(n) — 총 예상 비용

시작점에서 목표점까지의 총 예상 비용.
이 값이 작은 정점부터 꺼낸다.

g(n) — 실제 비용

시작점에서 현재 정점 n까지 실제로 소요된 비용. 이미 확정된 값이다.

h(n) — 휴리스틱 비용

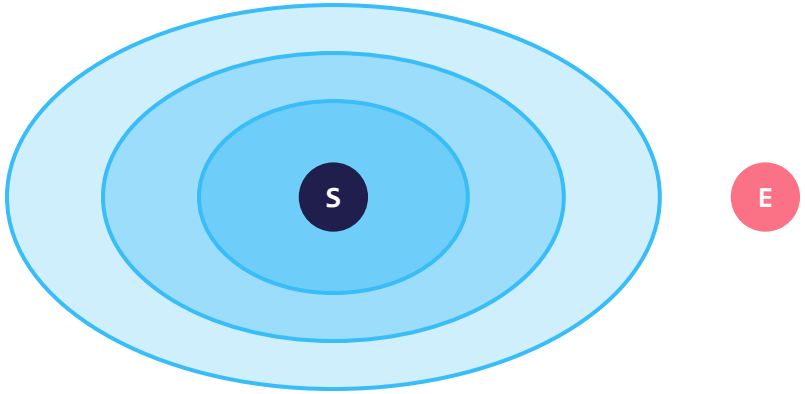
현재 정점 n에서 목표까지의 예상 비용.
추정치이며 정확할 필요는 없다.

A*는 f(n)이 가장 작은 정점을 우선 탐색한다

5장 다익스트라가 g(n)만 보고 골랐다면, A*는 여기에 "앞으로 얼마나 남았을까"라는 추정을 더한다. 그만큼 목적지 방향으로 탐색이 쏠린다.

다익스트라 vs A* — 탐색 방식 비교

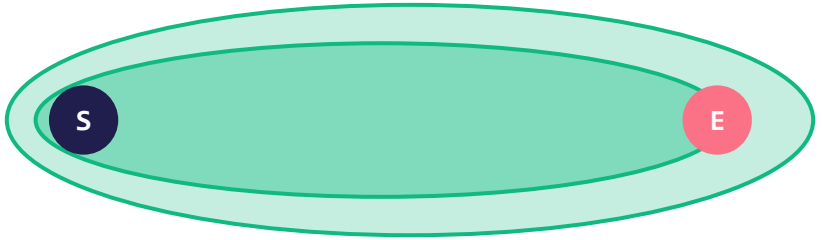
다익스트라 — 맹목적 탐색



시작점을 중심으로 동그랗게 모든 방향으로 퍼져 나간다

비효율적

A* — 휴리스틱 탐색



시작점에서 목적지 방향으로 타원형으로 길게 뻗어 나간다

효율적

h(n)이 나침반 역할을 한다 — 불필요한 탐색을 줄이고 목표를 향해 직진한다.

나침반의 역할 — 휴리스틱 함수 h(n)

유클리드 거리

대각선 이동이 가능한 경우 — 직선거리를 추정치로 쓴다.

맨해튼 거리

직각으로만 이동하는 경우 — $|x_1-x_2| + |y_1-y_2|$ 를 쓴다.

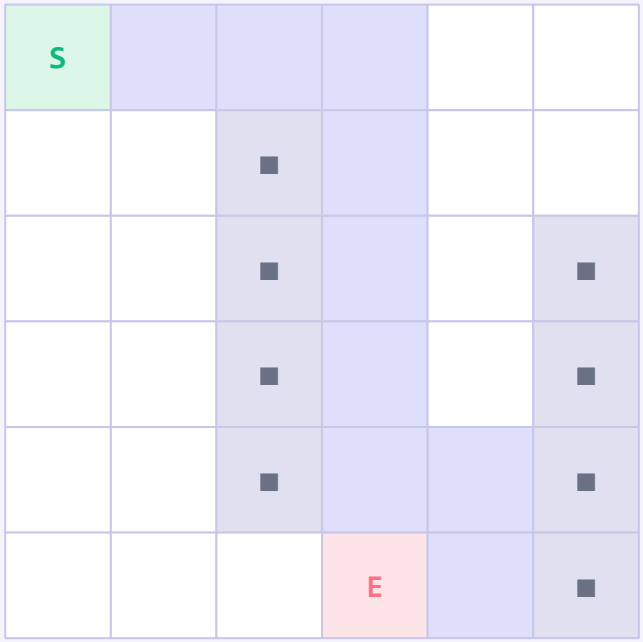
h(n) ≤ 실제 남은 거리

허용성(admissibility) 조건 — 언제나 낙관적으로 추정해야 한다

조건 만족 이 조건을 만족하면 A*는 반드시 최단 경로를 찾아낸다는 것이 보장된다.

조건 위반 h(n)이 실제보다 크면, 좋은 경로를 탐색 대상에서 빼 버릴 위험이 생긴다.

예제 — 미로 찾기 (grid pathfinding)



S 출발 · E 도착 · ■ 벽 · 연보라 = A*가 찾은 경로

최단 경로 발견

벽을 피해 돌아가면서도 길이 8칸의 최단 경로를 찾아낸다.

h(n)이 방향을 안내

맨해튼 거리가 나침반 역할을 해 목적지 쪽 칸부터 먼저 열어 본다.

다익스트라보다 효율적

같은 답을 찾되, 열어 보는 칸의 수가 훨씬 적다.

A*는 두 알고리즘의 일반화다

h(n) = 0 이면 다익스트라와 같고, g(n) = 0 이면 순수 그리디 탐색과 같아진다.

A* 알고리즘의 위상과 활용

내비게이션

실시간 길 찾기 시스템의 핵심 두뇌다. 도로망 규모에서도 실용적으로 동작한다.

게임 AI

실시간 전략 게임 유닛의 경로 계획에 표준처럼 쓰인다.

로봇 경로

자율 주행 로봇의 이동 경로 최적화에 사용된다.

A* = 다익스트라의 정확성 + 그리디의 신속성

현대 컴퓨터 과학에서 가장 널리 쓰이는 길 찾기 도구

휴리스틱을 쓰면서도 조건만 지키면 최적성을 잃지 않는다는 점에서, 이 장의 다른 기법들과는 성격이 조금 다르다.

11.5

메타 휴리스틱

11.5

가끔은 나쁜 쪽으로도 움직여야
더 높은 봉우리에 닿을 수 있다.

메타 휴리스틱이란?

다양한 문제에 범용적으로 적용할 수 있는 상위 레벨의 탐색 전략 프레임워크

하위 레벨 — 휴리스틱

실제로 해를 수정하고 이동하는 구체적 행동

예: "두 도시의 순서를 바꿔 본다"

상위 레벨 — 메타 휴리스틱

전체적인 탐색 방향과 규칙을 지휘

예: "국소 최적해 탈출을 위해 나쁜 해로의 이동을 허락한다"

등반가와 베이스캠프 대장의 관계

등반가(하위)가 발밑을 보며 한 발씩 옮길 때, 무전기로 전체 작전을 지시하는 베이스캠프 대장이 메타 휴리스틱이다. 대장은 지형을 직접 밟지 않지만 어디로 갈지를 정한다.

메타 휴리스틱은 문제에 독립적이다 — TSP든 스케줄링이든 같은 틀을 얹을 수 있다.

핵심 메커니즘 — 탐색과 활용의 균형

탐색 (exploration)

· 미지의 새로운 영역을 넓게 훑어본다

· 나쁜 해로의 이동을 허용한다

· 국소 최적해를 탈출할 기회를 준다

과도하면 → 무작위 탐색과 다를 바 없다

활용 (exploitation)

· 현재 좋은 해의 주변을 집중적으로 판다

· 더 좋은 이웃 해로만 이동한다

· 빠르게 국소 최적해로 수렴한다

과도하면 → 조기 수렴 문제가 생긴다

초반 : 넓은 탐색 → 후반 : 깊은 활용

§11.5 메타 휴리스틱

자연에서 배우다 — 물리와 생물의 지혜

수십억 년 동안 최적화 과정을 거쳐 온 자연의 섭리야말로 가장 잘 검증된 메타 휴리스틱이다.

물리 현상

모의 담금질 (SA)

금속의 담금질 과정을 모방한다. 높은 온도에서는 탐색, 낮은 온도에서는 활용.

생물학적 진화

유전 알고리즘 (GA)

자연선택과 교배, 돌연변이로 해의 집단을 진화시킨다.

곤충 행동

개미 집단 (ACO)

페로몬을 기반으로 경로를 최적화한다. 짧은 경로에 페로몬이 더 쌓인다.

새 떼 비행

입자 군집 (PSO)

개인의 경험과 집단의 정보를 합쳐 최적 위치를 향해 이동한다.

왜 자연을 흉내 내는가

자연의 최적화 과정은 모두 "가끔 손해를 감수하면서 다양성을 유지한다"는 공통점을 가진다. 그것이 곧 국소 최적해를 벗어나는 방법이다.

11.6

모의 담금질

11.6

금속을 천천히 식히면 가장 안정된 결정이 된다.
알고리즘도 천천히 식히면 더 좋은 해에 닿는다.

§11.6 모의 담금질

핵심 아이디어 — 담금질 과정

1

가열

금속을 높은 온도로 가열한다. 원자들이 무질서하게 움직인다.

2

급랭 (나쁜 예)

찬물에 확 식히면 불안정한 상태 그대로 굳어 버린다.

3

서냉 (좋은 예)

천천히 온도를 낮추면 가장 안정적인 결정 구조로 배열된다.

물리적 개념	알고리즘 대응
금속의 상태	문제의 해 (solution)
내부 에너지	비용 함수의 값 — 최소화 대상
온도 (T)	탐색의 무작위성을 제어하는 파라미터

온도 하나로 "탐색 ↔ 활용"의 비중을 연속적으로 조절한다는 것이 이 기법의 핵심이다.

확률적 수용 규칙 — 볼츠만 분포

$$P(\text{accept}) = e^{(-\Delta E / T)}$$

이웃이 더 좋으면 (ΔE < 0)

무조건 이동한다 (P = 1). 좋은 쪽으로 가는 것을 망설일 이유가 없다.

이웃이 더 나쁘면 (ΔE > 0)

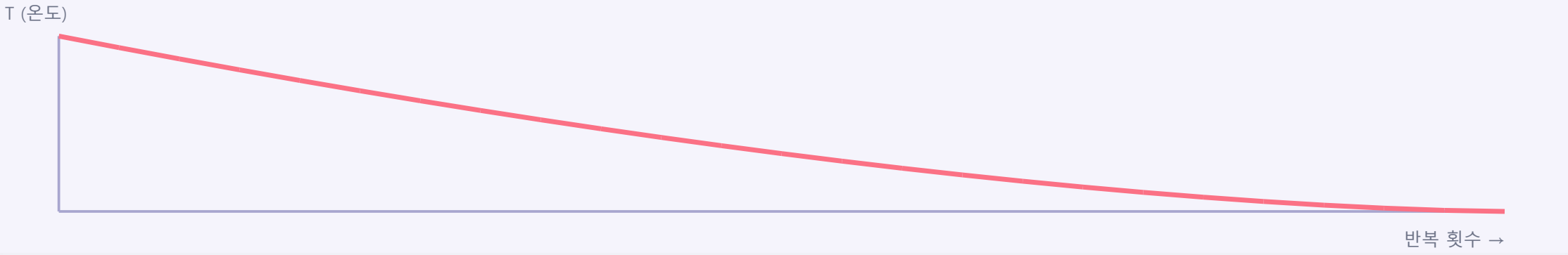
확률 P로 이동을 허용한다. 이 한 줄이 국소 최적해를 탈출하는 유일한 통로다.

T가 높으면 초반 — 나쁜 해도 높은 확률로 수용한다. 곧 탐색(exploration)이다.

T가 낮으면 후반 — 나쁜 해는 거의 거부한다. 곧 활용(exploitation)이다.

§11.6 모의 담금질

"초반엔 과감하게, 후반엔 신중하게"



초반 — 고온 상태 (탐색)

해가 좋아졌다 나빠졌다를 반복한다.
탐색 공간을 활발히 돌아다닌다.
깊은 국소 최적해도 탈출할 수 있다.

중반 — 서냉 중 (균형)

좋은 쪽으로 움직이려 하지만,
국소 최적해에 빠지더라도
탈출할 가능성이 아직 남아 있다.

후반 — 저온 상태 (수렴)

나쁜 해로는 거의 이동하지 않는다.
가장 좋은 지역에서
미세 조정하며 수렴한다.

"처음에는 술 취한 사람처럼 비틀거리며 헤매다가, 술이 깰수록 목적지를 향해 똑바로 걸어가는 전략"

모의 담금질 유사 코드

simulated_annealing.pseudo

```
Algorithm SIMULATED_ANNEALING(s0, kmax, T0):

  s <- s0

  for k <- 0 to kmax - 1 do
    T <- T0 * (1 - (k+1) / kmax)          // 온도 감소
    s_new <- RANDOM_CHOICE(NEIGHBORS(s))  // 이웃 하나 선택

    if E(s_new) <= E(s) then
      s <- s_new                          // 더 좋으면 무조건 수용
    else
      P <- exp( -(E(s_new) - E(s)) / T )  // 수용 확률 계산
      if RANDOM(0,1) < P then
        s <- s_new                        // 확률적 수용!

  return s
```

온도 스케줄

T를 어떻게 낮추느냐가 성능을 좌우한다. 선형·지수 감소가 흔히 쓰인다.

이웃 하나만 뽑는다

언덕 오르기와 달리 모든 이웃을 비교하지 않는다. 무작위로 하나만 본다.

exp(-ΔE / T)

ΔE가 클수록, T가 작을수록 수용 확률이 급격히 떨어진다.

고정 횟수 종료

kmax번 반복하면 끝난다. 개선 여부와 무관하게 예산이 정해진다.

최종 해가 아니라 "지금까지 본 최선"을 따로 기억해 두는 것이 실무의 요령이다

온도가 남아 있는 동안에는 마지막 s가 최선이 아닐 수 있기 때문이다.

모의 담금질 적용 분야

반도체 회로 배치

VLSI layout — 수백만 개의 트랜지스터를 배치하며 배선 길이를 최소화한다.

공정 스케줄링

job-shop 문제 — 공장의 작업 순서를 정해 전체 완료 시간을 줄인다.

TSP

꼬인 경로를 천천히 풀어내며 깔끔한 순회 경로로 수렴시킨다.

냉각 속도(cooling rate) 조절이 성패를 가른다

너무 빠르면(예: 0.9) 꼬인 선을 다 풀지 못한 채 멈추고, 너무 느리면(예: 0.9999) 시간이 감당할 수 없이 늘어난다.

파라미터 하나에 결과가 크게 흔들린다는 점이 메타 휴리스틱의 공통된 약점이다

이 때문에 실무에서는 여러 설정으로 여러 번 돌려 보고 가장 좋은 것을 고르는 방식을 쓴다.

11.7

TSP와 모의 담금질

11.7

엉킨 실타래 같은 경로가
서서히 풀려 깔끔한 다각형이 된다.

TSP + 모의 담금질 — 핵심 기법

구간 뒤집기 (2-opt)

임의의 두 지점 i, j를 잡아 path[i:j+1]을 통째로 뒤집는다. 꼬인 선을 한 번에 풀어 주므로 단순 swap보다 훨씬 효과적이다.

확률적 점프

초반에는 T가 높아 P ≈ 1 로 거의 무조건 수용하고, 후반에는 T가 낮아 P ≈ 0 이 되어 좋은 것만 남는다.

tsp_sa.py

```python
# 2-opt: 구간을 통째로 뒤집는다
i, j = sorted(random.sample(range(n), 2))
new_path = path[:i] + path[i:j+1][::-1] + path[j+1:]

# 확률적 수용
diff = length(new_path) - length(path)
if diff < 0 or random.random() < math.exp(-diff / temp):
    path = new_path
```

왜 뒤집기인가

두 도시를 바꾸면 네 개의 간선이 흔들리지만, 구간을 뒤집으면 딱 두 개만 바뀐다.

수렴하는 모습

뒤죽박죽 엉킨 실타래 같던 경로가 서서히 풀려, 외곽을 도는 깔끔한 다각형이 된다.

이웃(neighbor)을 어떻게 정의하느냐가 메타 휴리스틱 성능의 절반을 결정한다.

11.8

유전 알고리즘

11.8

하나의 해를 고치는 대신,
해의 집단을 세대에 걸쳐 진화시킨다.

다윈의 진화론과 적자생존

적자생존

더 우월한 개체가 다음 세대의 부모가 될 확률이 높다.

교배

우수한 부모의 유전자를 섞으면 더 뛰어난 자식이 나올 수 있다.

돌연변이

유전자에 무작위 변화를 준다. 다양성을 유지하고 새로운 가능성을 연다.

교배 = 활용(exploitation) · 돌연변이 = 탐색(exploration)

앞 절의 "탐색과 활용의 균형"이 여기서는 두 연산으로 분리되어 나타난다

모의 담금질이 온도 하나로 균형을 잡았다면, 유전 알고리즘은 교배율과 돌연변이율이라는 두 손잡이로 조절한다.

주요 구성 요소 — 유전자와 적합도

유전자 & 염색체

하나의 해(solution)가 곧 염색체이고,
그 안의 각 비트나 값이 유전자다.

`[ 1, 0, 1, 1, 0 ]`

배낭 문제의 한 해 — 1번·3번·4번 물건을 담는다

적합도 함수 (fitness)

해가 "얼마나 좋은지"를 점수로 평가한다.
배낭 문제라면 담은 물건 가치의 총합이다.

점수 ↑ → 생존 확률 ↑

TSP라면 경로 길이의 역수를 쓴다 — 짧을수록 적합하다

해를 어떤 모양으로 표현할 것인가가 첫 번째 설계 결정이다

표현이 정해져야 교배와 돌연변이를 어떻게 할지가 따라 나온다. TSP처럼 순열이 해가 되는 경우에는 표현 방식 자체가 문제가 되는데, 그것이 §11.9의 주제다.

진화의 과정 — 알고리즘 사이클

① 초기 집단 생성
무작위 해를 수십에서 수백 개 만든다.

② 적합도 평가 & 선택
점수를 매기고 룰렛 휠 등으로 부모를 고른다.

③ 교배 (crossover)
부모의 유전자를 조합해 자식을 만든다.

④ 돌연변이 (mutation)
극히 일부 유전자를 무작위로 바꾼다.

⑤ 세대교체
자식들이 새로운 부모가 되어 ②부터 반복한다.

crossover.txt

```
# 교배 예시 (단일 지점)
부모1: [1,1,1 | 0,0]
부모2: [0,0,0 | 1,1]
--------------------
자식 : [1,1,1 | 1,1]   <- 슈퍼 개체!
```

mutation.txt

```
# 돌연변이 예시
[1,1,1,1,1]  ->  [1,1,0,1,1]

3번째 비트가 무작위로 변경됨
```

왜 돌연변이가 필요한가
집단이 비슷해지면 교배만으로는 새로운 해가 나오지 않는다. 무작위 변화가 정체를 깬다.

한 세대가 아니라 수백 세대를 돌린다 — 집단 전체가 조금씩 좋아진다.

유전 알고리즘 유사 코드

genetic_algorithm.pseudo

```
Algorithm GENETIC_ALGORITHM():

  P    <- INITIALIZE_POPULATION(pop_size)
  best <- BEST_IN(P)

  for gen <- 1 to max_generations do
    FITNESS_EVALUATE(P)                // 적합도 평가
    P_new <- TOP_K(P, elite_count)     // 엘리트 보존

    while SIZE(P_new) < pop_size do
      p1, p2 <- SELECT(P)              // 부모 선택
      child  <- CROSSOVER(p1, p2)      // 교배
      child  <- MUTATE(child, rate)    // 돌연변이
      P_new  <- P_new + {child}

    P <- P_new                         // 세대 교체

  return best
```

pop_size
집단의 크기. 크면 다양성이 늘지만 세대당 계산량도 늘어난다.

elite_count
무조건 살아남는 상위 개체 수. 퇴보를 막는 안전장치다.

SELECT
적합도에 비례해 확률적으로 부모를 뽑는다 (룰렛 휠).

rate
돌연변이 확률. 너무 크면 무작위 탐색이 되고, 너무 작으면 정체된다.

네 개의 파라미터가 서로 얽혀 있다 — 튜닝이 곧 실력이 되는 이유다.

유전 알고리즘 실전 응용

게임 AI 행동 진화

다양한 AI를 서로 싸우게 하고 승리한 AI의 행동을 진화시켜, 사람이 생각하지 못한 전략을 얻는다.

NASA 안테나 설계

GA로 설계를 진화시킨 결과물은 기괴하게 생겼지만 성능이 가장 좋았고, 실제 우주선에 탑재되었다.

복잡한 함수 최적화

자동차 공기역학 설계나 금융 모델 파라미터 튜닝처럼 변수가 수없이 많은 문제에 쓰인다.

GA는 인간의 고정관념을 뛰어넘는 창의적인 해를 찾아낼 수 있다

사람은 "이렇게 생겨야 안테나답다"는 선입견에서 벗어나기 어렵지만, 알고리즘에는 그런 편견이 없다. 평가 함수만 정확하면 형태는 아무래도 좋다.

뒤집어 말하면, 평가 함수가 잘못되면 엉뚱한 방향으로 진화한다

무엇을 좋다고 정의했는지가 결과의 전부를 결정한다.

11.9

TSP와 유전 알고리즘

11.9

순열을 유전자로 쓰면 교배가 까다로워진다.
도시가 중복되거나 빠지기 때문이다.

TSP 유전 알고리즘 — 순서 교배 (OX)

일반적인 단일 지점 교배를 쓰면 도시가 중복되거나 누락된다 — 순서 교배(ordered crossover)가 필요하다.

ordered_crossover.txt

```
① 부모1에서 구간 (start=2, end=5) 을 고른다
   Parent1: [0, 1, 2, 3, 4, 5, 6, 7, 8]
   Parent2: [8, 4, 2, 6, 3, 1, 5, 0, 7]

② 부모1의 그 구간을 자식에 그대로 복사한다
   Child  : [-1,-1, 2, 3, 4, 5,-1,-1,-1]

③ 부모2의 순서대로 남은 칸을 채운다 (중복은 건너뛴다)
   8 -> 빈칸0    4 -> 이미 있음    2 -> 이미 있음
   6 -> 빈칸1    1 -> 빈칸6        0 -> 빈칸7 ...
   Child  : [ 8, 6, 2, 3, 4, 5, 1, 0, 7]
```

중복도 누락도 없다

모든 도시가 정확히 한 번씩만 등장한다.

순서 정보 보존

부모1의 구간 순서와 부모2의 상대 순서를 함께 물려받는다.

순열 문제의 표준

스케줄링처럼 순서가 해가 되는 문제 전반에 그대로 쓰인다.

해의 표현이 순열이면 교배 연산자부터 새로 설계해야 한다.

TSP GA — 엘리트 보존과 돌연변이

엘리트 보존 (elitism)

· 상위 20% 개체는 교배 없이 다음 세대로 넘긴다

· 우연히 찾은 좋은 경로가 파괴되는 것을 막는다

· 세대가 지날수록 최선이 나빠지지 않도록 하는 안전장치

돌연변이 (mutation)

· 도시 순서를 강제로 swap한다

· 비슷한 경로끼리만 교배하는 정체 상태를 깬다

· 엉뚱한 곳에서 획기적인 지름길이 발견되기도 한다

"끼리끼리만 결혼하면 유전병이 생기듯, 알고리즘도 비슷한 경로끼리만 교배하면 발전이 멈춘다"

엘리트 보존은 활용(exploitation)을, 돌연변이는 탐색(exploration)을 담당한다. 둘 중 하나만 세게 밀면 반드시 실패한다.

엘리트 비율과 돌연변이율은 서로 반대 방향으로 작용한다 — 함께 조절해야 한다.

11.10

자연 영감 알고리즘들

11.10

지휘자 없는 무리가 최적을 찾아낸다.
PSO와 ACO.

§11.10 자연 영감 알고리즘

입자 군집 최적화 (PSO)

영감 — 새 떼나 물고기 떼의 움직임. 지휘자가 없는데도 무리 전체가 하나처럼 움직인다.

pBest — 개인 경험

"내가 지금까지 날아다닌 곳 중 가장 좋았던 위치" 쪽으로 끌린다. 인지적 요소라고 부른다.

gBest — 집단 정보

"무리 전체가 발견한 곳 중 가장 좋은 위치" 쪽으로 끌린다. 사회적 요소라고 부른다.

특징 교배나 돌연변이 없이 간단한 속도 벡터 연산만으로 구현된다. 계산 효율이 매우 높다.

주요 응용 연속적인 파라미터 최적화, 신경망 학습 등에 널리 쓰인다.

개미 집단 알고리즘 (ACO)

영감 — 개미들의 최단 경로 탐색. 페로몬(pheromone)을 통한 간접적인 정보 공유가 핵심이다.

① 초기 탐색 아무 정보가 없으므로 개미들이 무작위로 경로를 돌아다닌다.

② 페로몬 축적 짧은 경로일수록 왕복이 빨라 같은 시간에 페로몬이 더 많이 쌓인다.

③ 경로 강화 페로몬이 진한 길을 고를 확률이 올라간다. 양성 피드백이 일어난다.

④ 페로몬 증발 오래된 정보는 서서히 사라져, 새로운 경로를 탐색할 여지를 남긴다.

강점 — TSP, 네트워크 라우팅, 물류 배차처럼 그래프 기반 경로 최적화에서 뛰어나다

11.11

11장을 마치며

11.11

세 가지 접근을 나란히 놓고,
언제 무엇을 쓸지 정리한다.

§11.11 정리 및 비교

근사 알고리즘 vs 휴리스틱 vs 메타휴리스틱

구분	근사 알고리즘	휴리스틱	메타휴리스틱
핵심 목적	이론적 보장 + 빠른 해	빠른 시간에 괜찮은 해	범용 탐색 프레임워크
최적해 보장	근사 비율만 보장	보장 없음	보장 없음
수학적 분석	가능 (증명)	거의 불가능	거의 불가능
문제 의존성	비교적 낮음	매우 높음	낮음 (범용)
결과 안정성	안정적	실행마다 다름	실행마다 다름
대표 기법	2-근사, Greedy SC	최근접 이웃, 언덕 오르기	SA, GA, ACO, PSO
장점	"얼마나 나쁜지"를 안다	빠르고 구현이 단순	복잡한 문제에도 적용 가능
단점	품질이 제한적일 수 있음	국소 최적해 위험	파라미터 튜닝이 필요

세 접근은 대체재가 아니라, 문제의 성격에 따라 고르는 선택지다.

§11.11 정리 및 비교

언제 무엇을 써야 하는가?

근사 알고리즘 — Safety First

· 최악의 상황에 대한 보장이 필수인 경우

· 금융 리스크, 구조물 안전, 의료 분야

· 문제 구조가 수학적으로 명확할 때

메타 휴리스틱 — Performance First

· 최악보다 평균적 성능이 중요할 때

· 문제가 너무 복잡하거나 제약 조건이 많을 때

· 내부 구조를 모르는 블랙박스 최적화

파라미터 튜닝 = 과학 + 예술

온도 스케줄, 돌연변이율, 집단 크기에는 정답 공식이 없다. 훌륭한 엔지니어는 파라미터들이 서로 얽혀 춤추는 리듬을 이해하고 지휘할 수 있는 사람이다.

무엇을 고르든, 결과를 실측으로 검증하는 절차만큼은 빼놓을 수 없다.

SUMMARY

11장 정리

휴리스틱
최적해 보장 없이 빠르게 괜찮은 해를 찾는 경험적 방법. 보장을 내려놓는 대신 적용 범위를 얻는다.

언덕 오르기
이웃 해 중 더 좋은 것으로만 이동한다. 빠르지만 국소 최적해에 갇힐 위험이 있다.

A* 알고리즘
f(n) = g(n) + h(n) — 과거 비용과 미래 예측을 결합한다. h가 낙관적이면 최적성도 지킨다.

메타 휴리스틱
탐색(exploration)과 활용(exploitation)의 균형을 지휘하는 상위 전략이다.

모의 담금질
온도 기반 확률적 수용. 초반에는 과감하게, 후반에는 신중하게 움직인다.

유전 알고리즘
적자생존 · 교배 · 돌연변이로 해의 집단을 세대에 걸쳐 진화시킨다.

PSO / ACO
무리 지능 기반. 개체가 정보를 공유하며 함께 최적해를 찾아 나간다.

NFL 정리
만능 알고리즘은 없다. 문제 특성에 맞는 선택이 결국 실력이 된다.

CHAPTER 11

Q & A

정답이 아니어도 괜찮습니다.
지금보다 조금 더 나은 쪽으로 한 걸음이면 충분합니다.