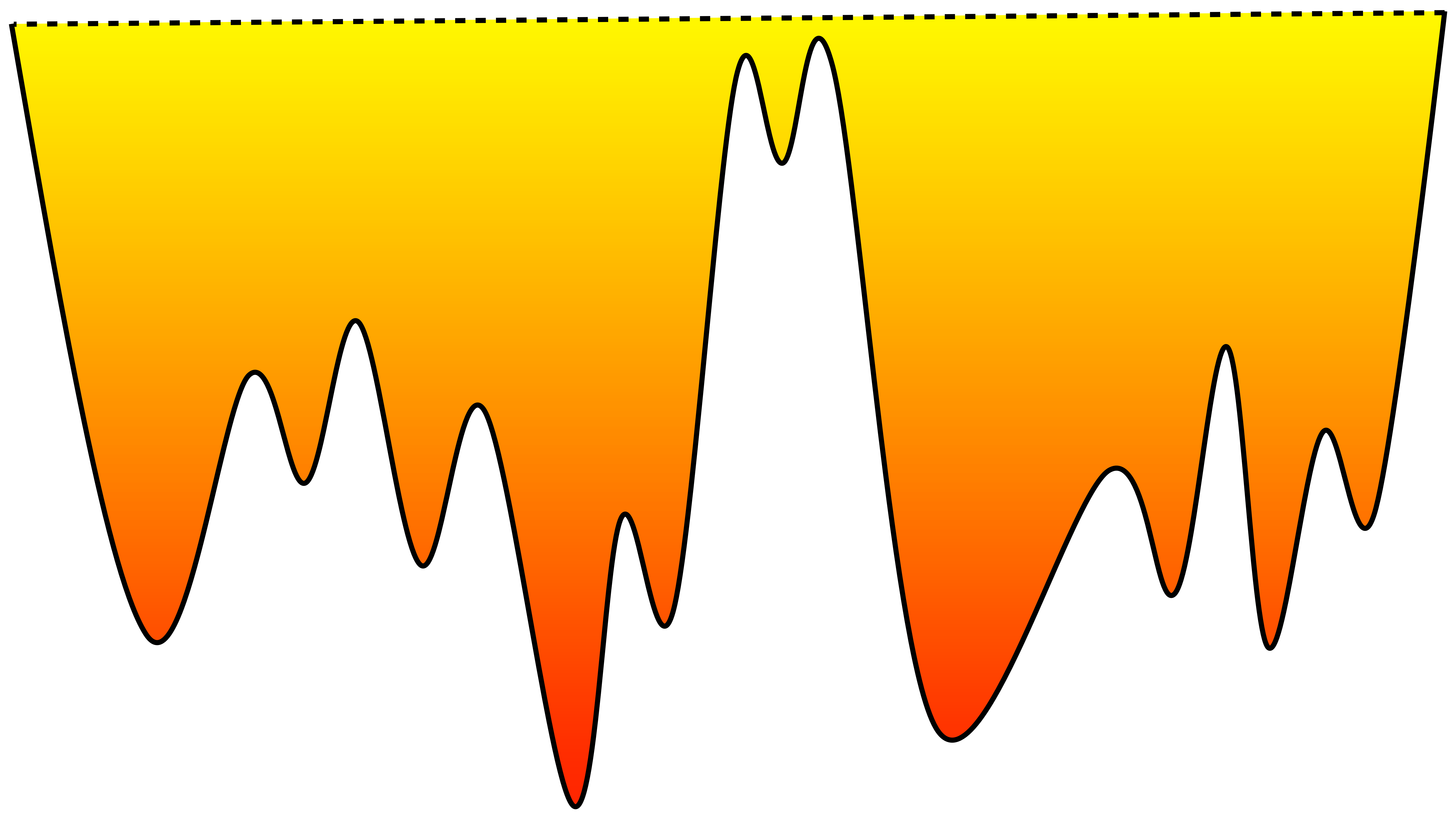
Discrete Optimization

Local Search: Part VIII

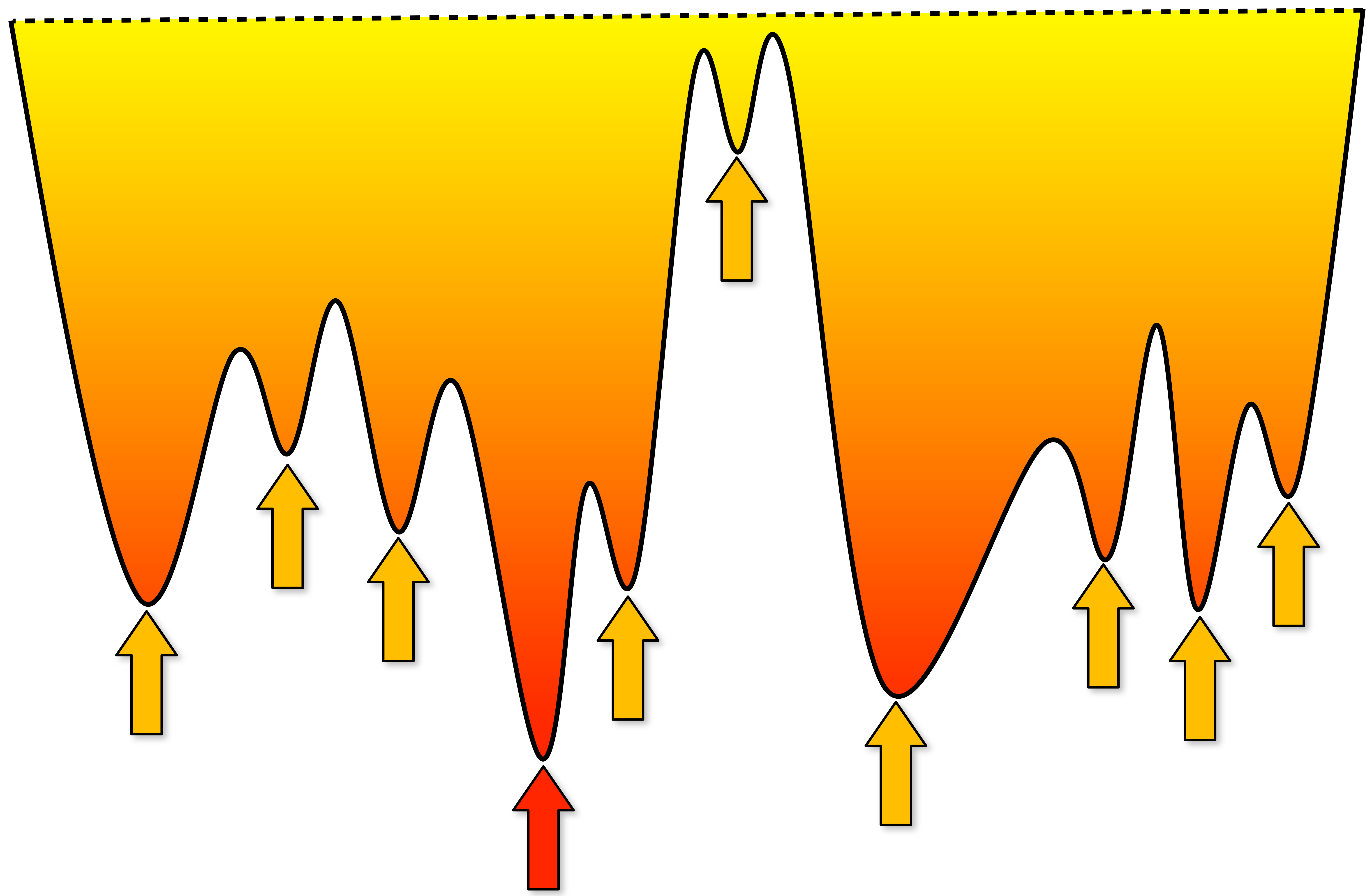
Goal of the Lecture

- Local search
  - meta-heuristics
  - multi-start search
  - simulated annealing
  - tabu search

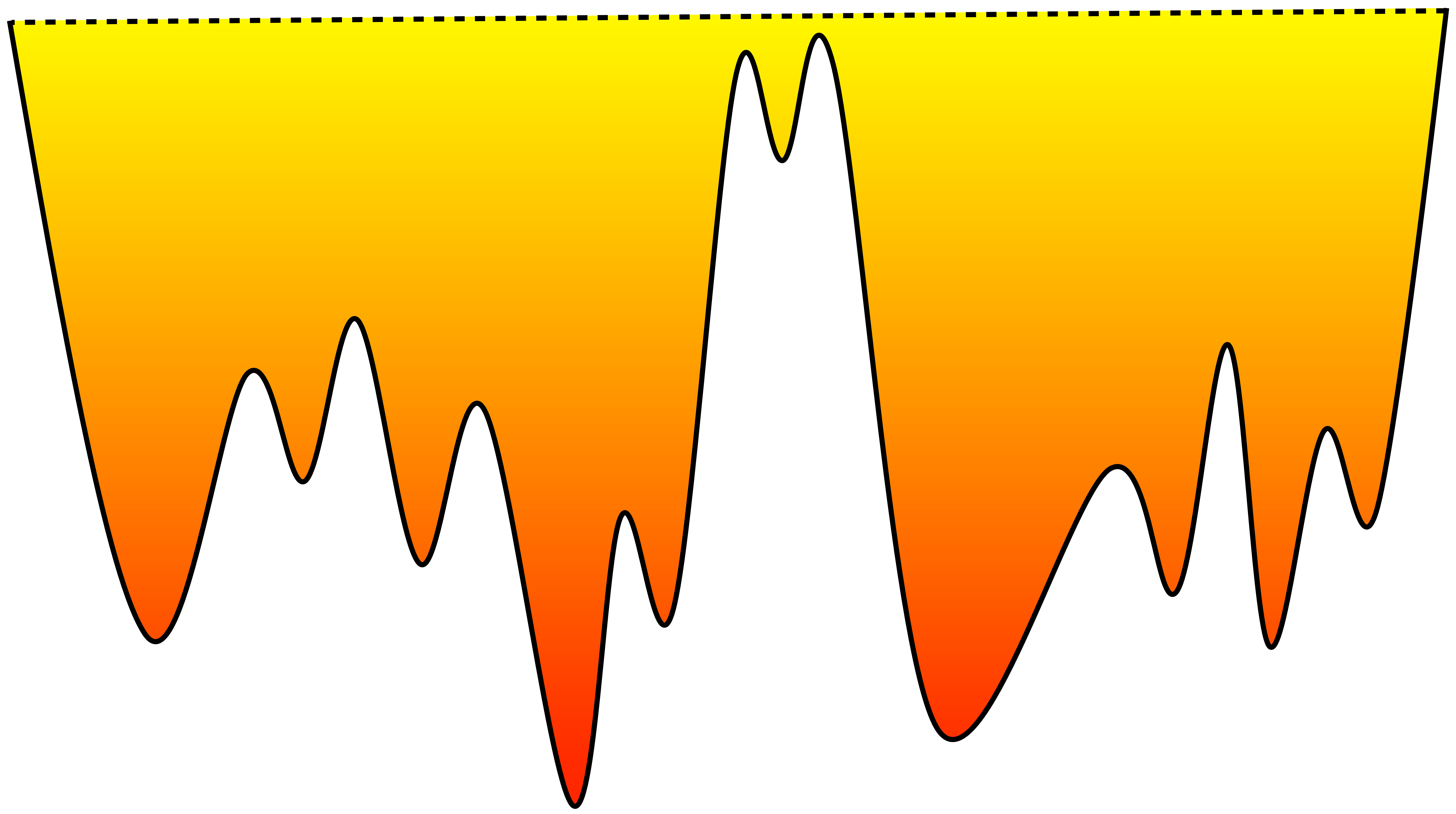
Escaping Local Minima



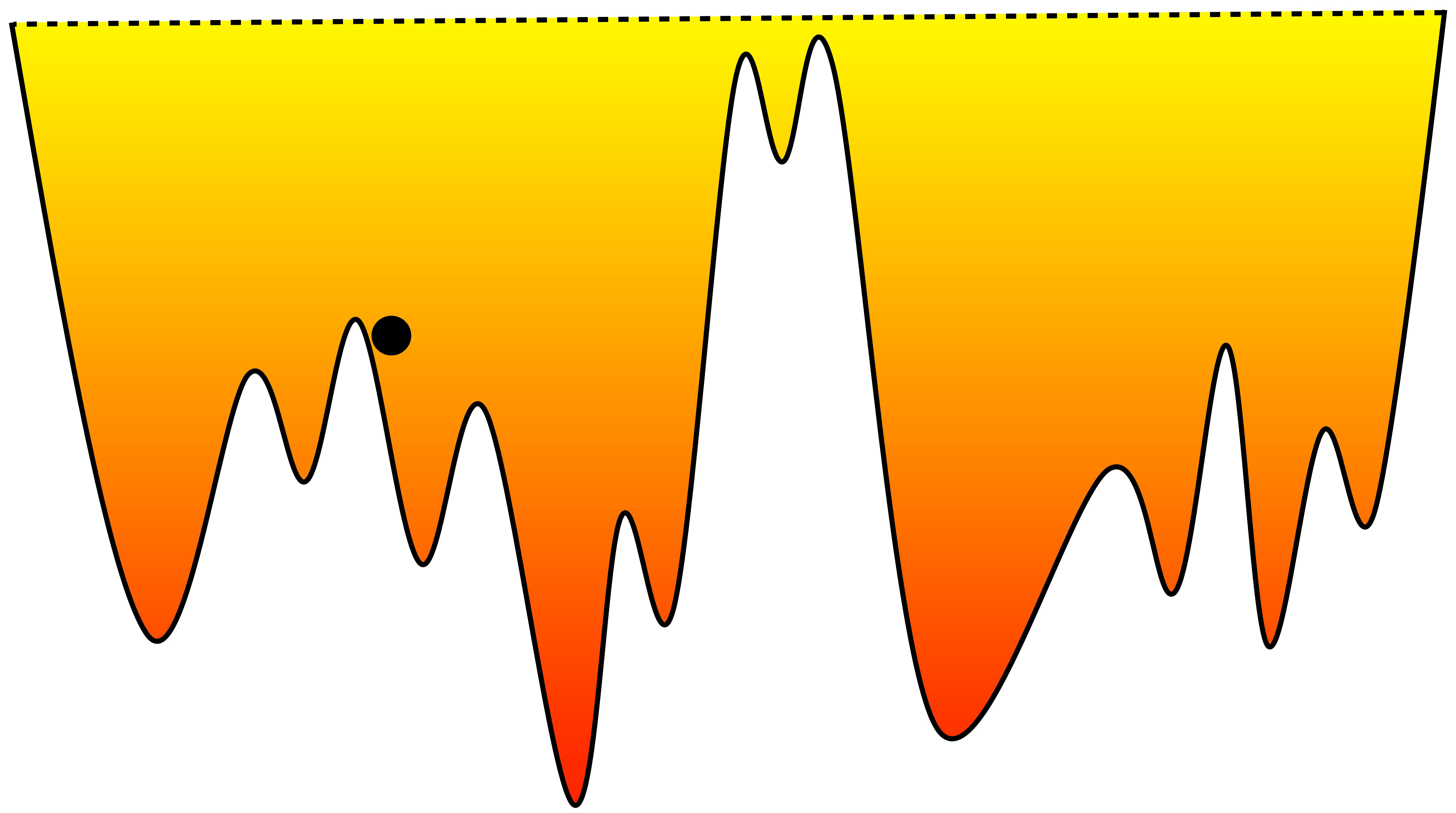
Escaping Local Minima



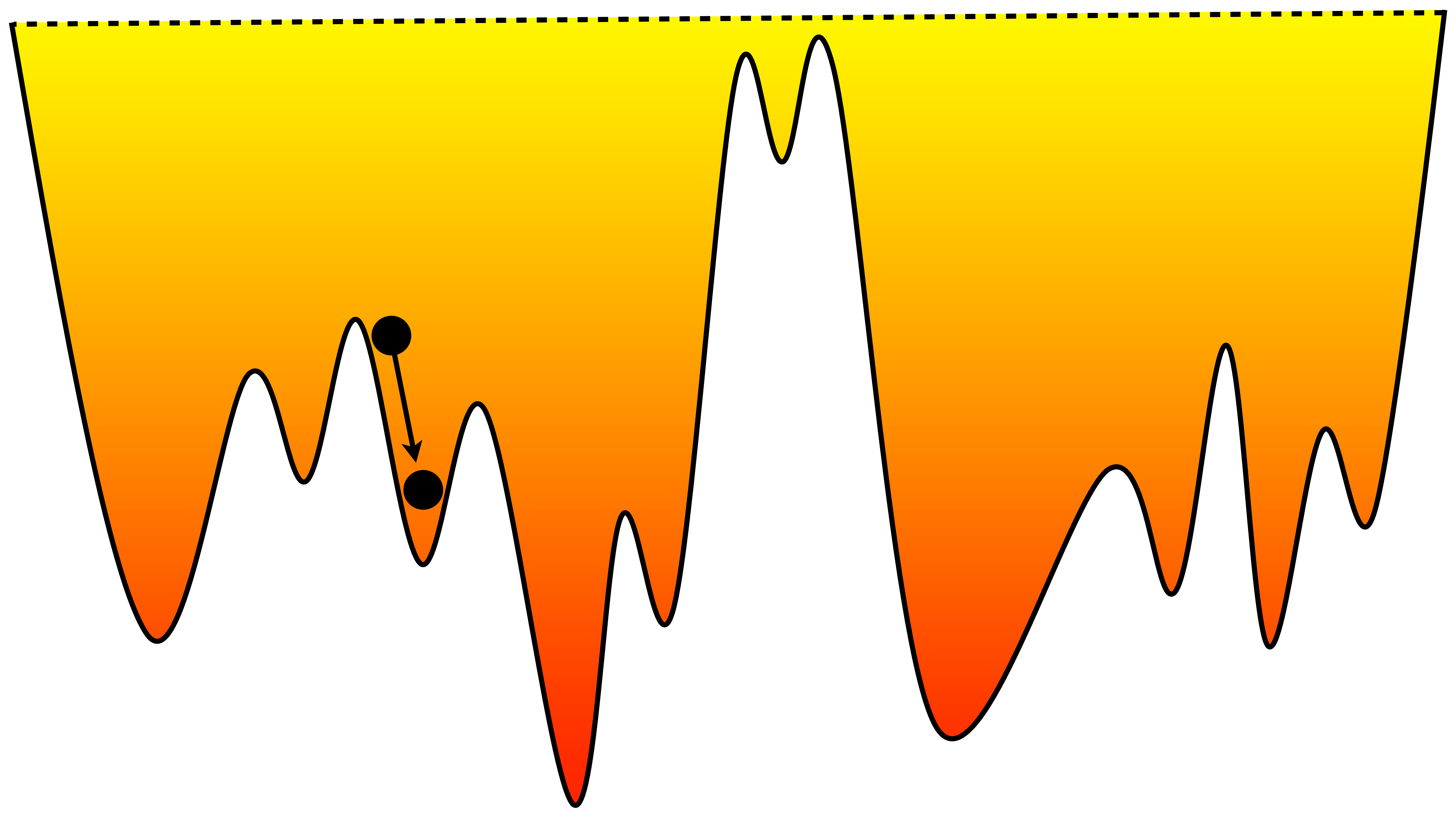
Iterated Local Search



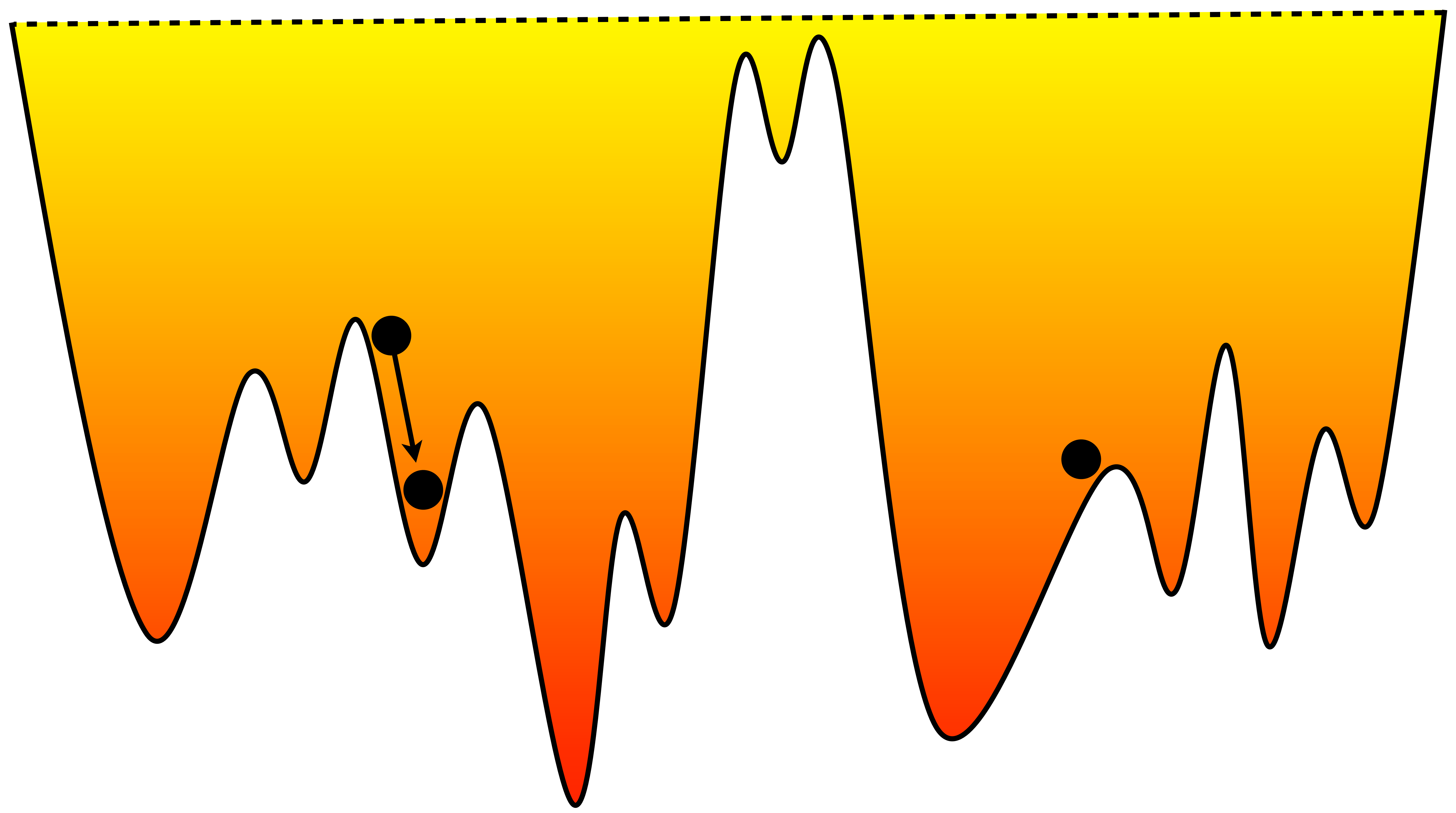
Iterated Local Search



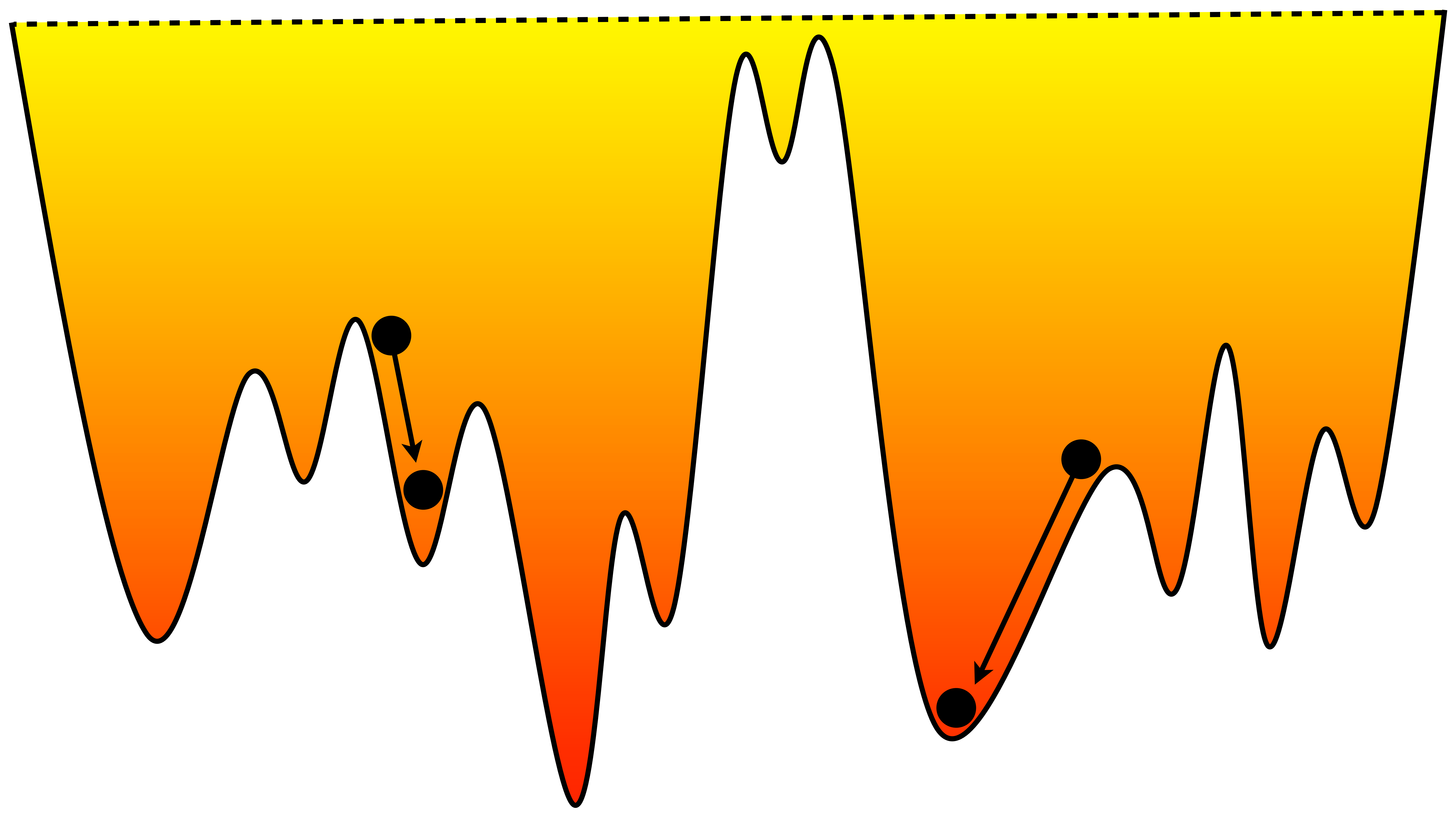
Iterated Local Search



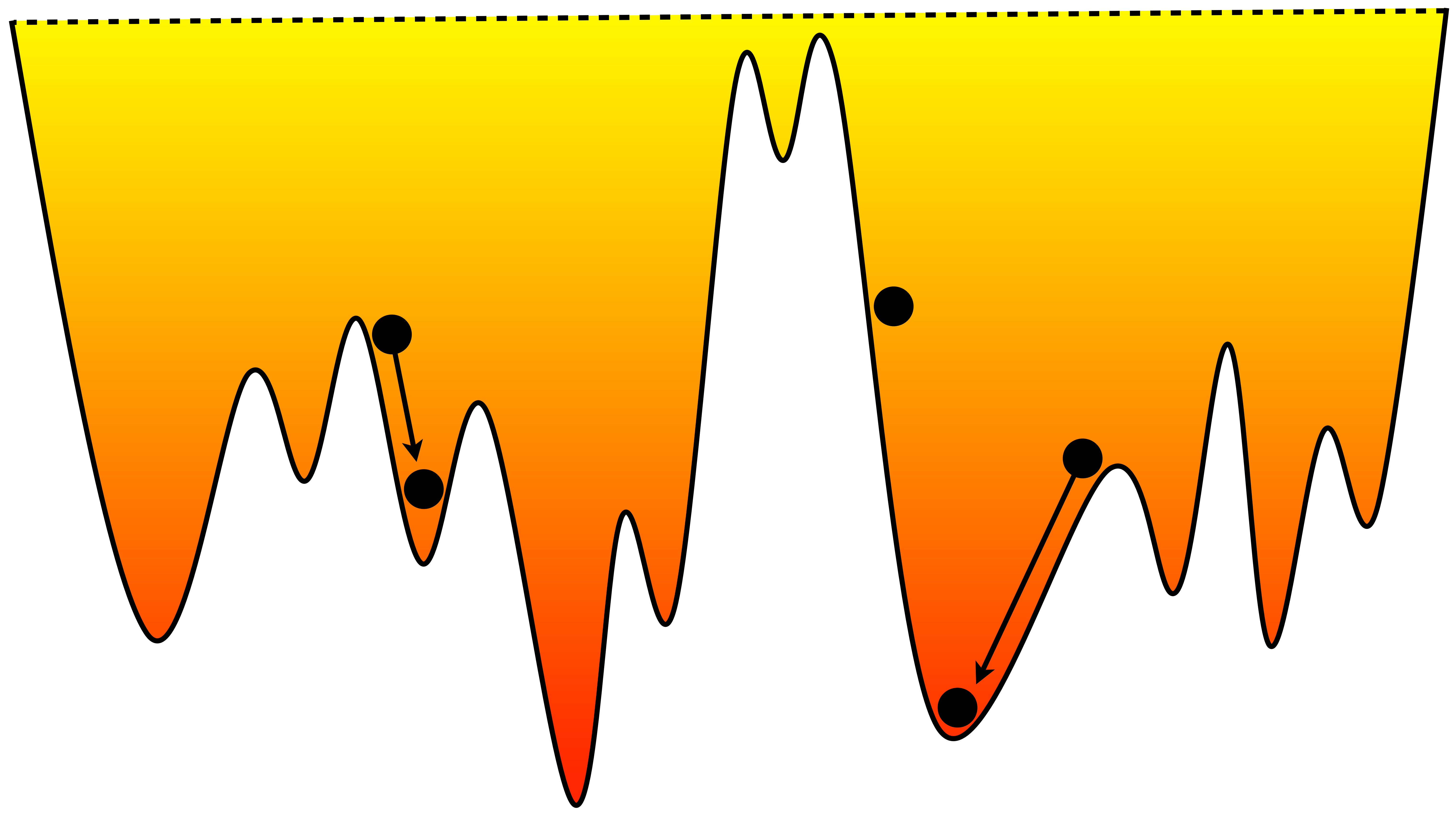
Iterated Local Search



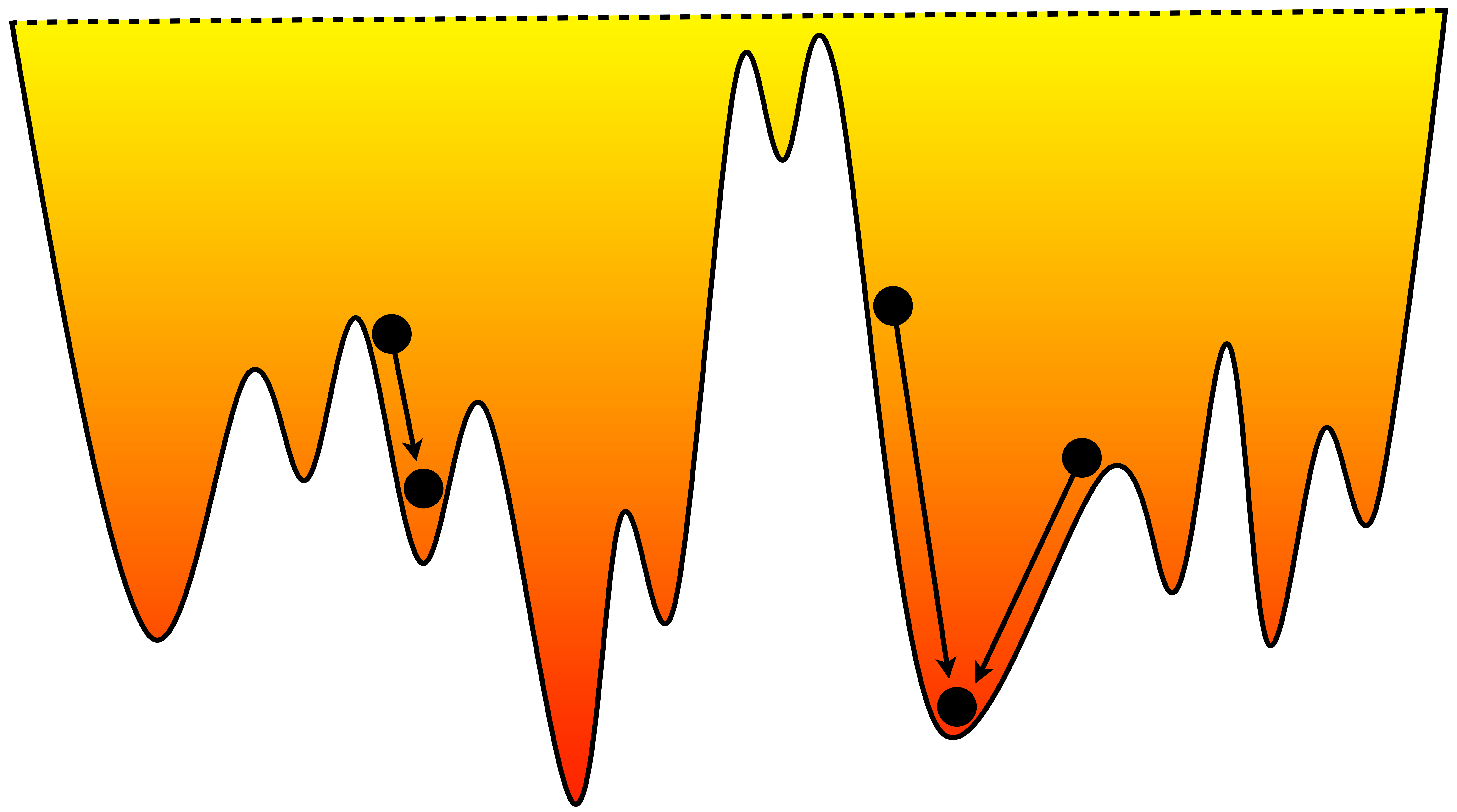
Iterated Local Search



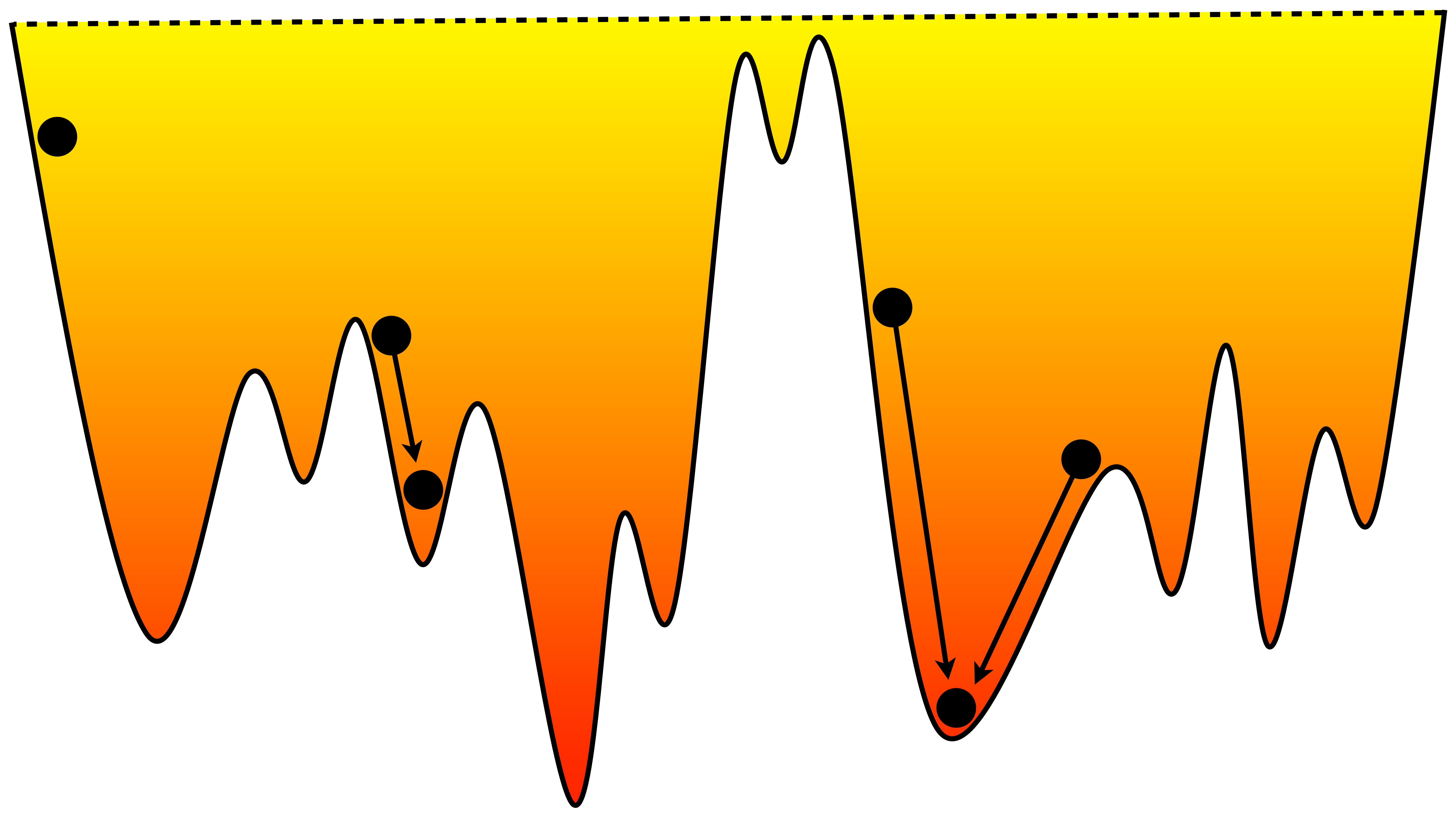
Iterated Local Search



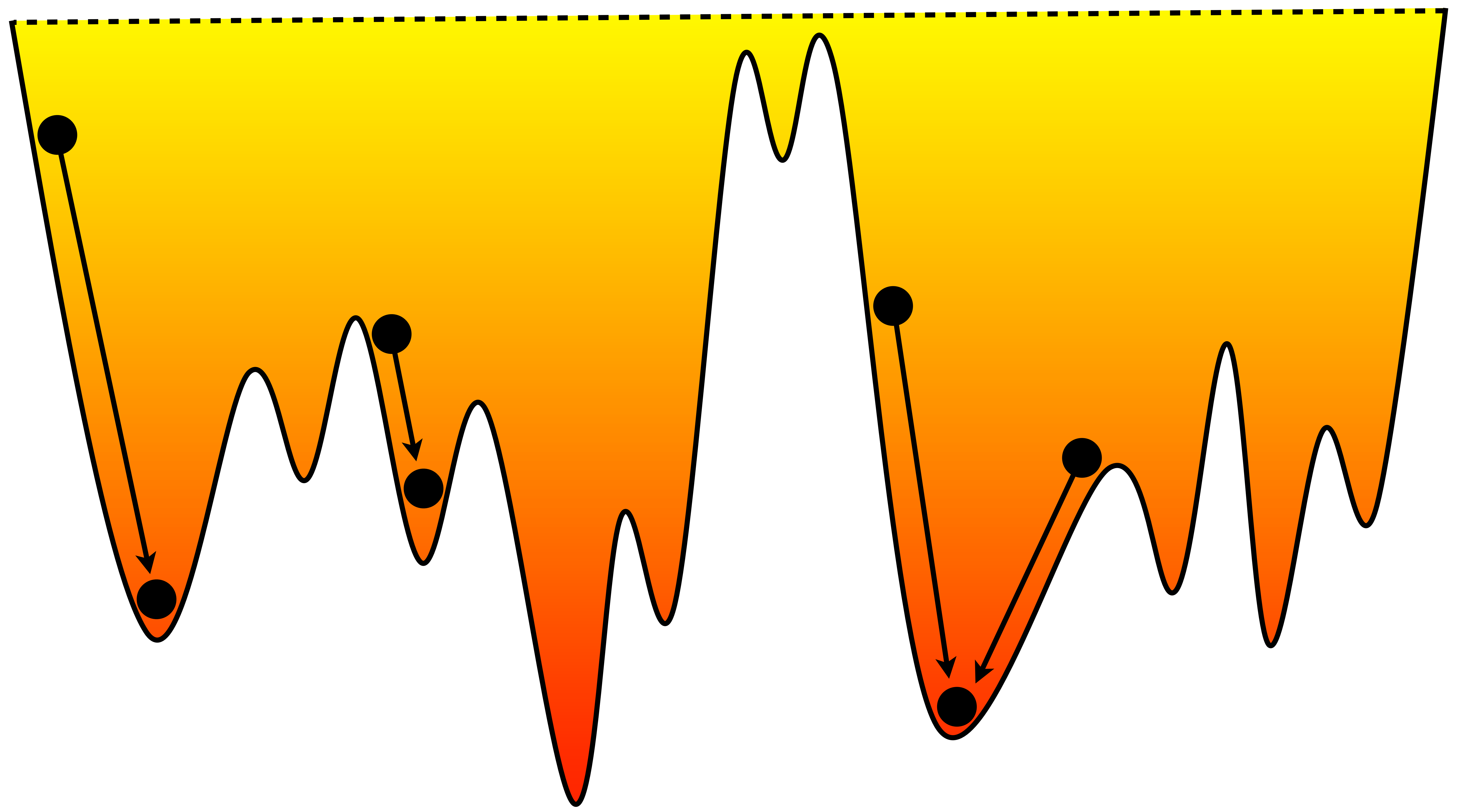
Iterated Local Search



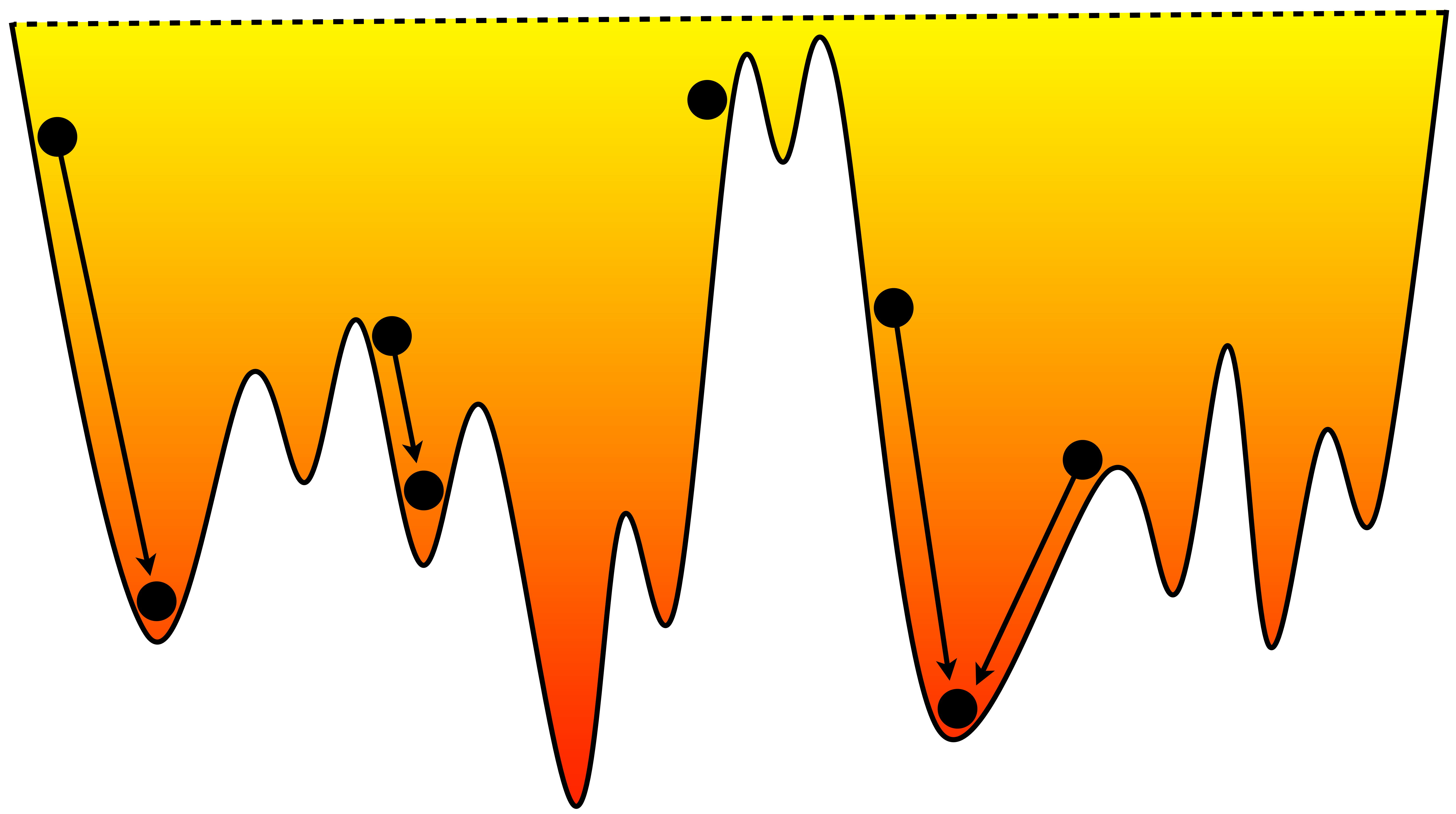
Iterated Local Search



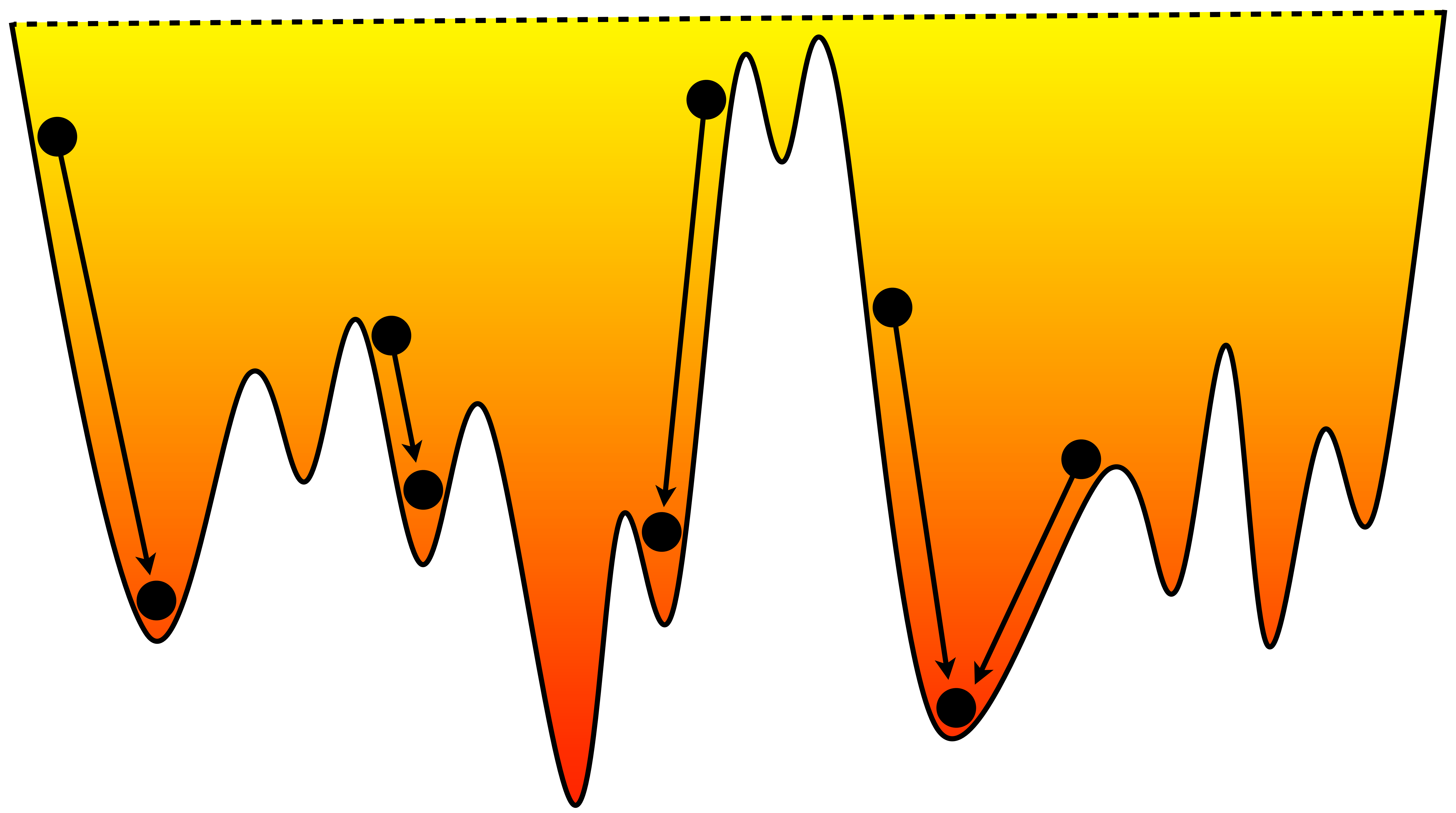
Iterated Local Search



Iterated Local Search



Iterated Local Search



Iterated Local Search

- Execute multiple local search from different starting configuration
  - generic
    - can be combined with other metaheuristics
    - multistarts or restarts

Iterated Local Search

- Execute multiple local search from different starting configuration
  - generic
    - can be combined with other metaheuristics
    - multistarts or restarts

```
1.  function IteratedLocalSearch($f, N, L, S$) {
2.      $s :=$ generateInitialSolution();
3.      $s^* := s$;
4.      for $k := 1$ to $MaxSearches$ do
5.          $s :=$ LocalSearch($f, N, L, S, s$);
6.          if $f(s) < f(s^*)$ then
7.              $s^* := s$;
8.          $s :=$ generateNewSolution($s$);
9.      return $s^*$;
10. }
```

Metropolis Heuristic

- Basic idea
  - accept a move if it improves the objective value or, in case it does not, with some probability
  - the probability depends on how “bad” the move is
  - inspired by statistical physics

Metropolis Heuristic

- Basic idea
  - accept a move if it improves the objective value or, in case it does not, with some probability
  - the probability depends on how “bad” the move is
  - inspired by statistical physics
- How is the probability chosen?
  - t is a temperature
  - Δ is the difference f(n) - f(s)
  - a degrading move is accepted with probability,

$$\exp(\frac{-\Delta}{t})$$

Metropolis Heuristic

```
1.  function S-METROPOLIS[t](N,s)
2.      select $n \in N$ with probability $1/\#N$;
3.      if $f(n) \leq f(s)$ then
4.          return $n$;
5.      else with probability $exp(\frac{-(f(n)-f(s))}{t})$
6.          return $n$;
7.      else
8.          return $s$;
```

Metropolis Heuristic

```
1.  function S-METROPOLIS[t](N,s)
2.      select $n \in N$ with probability $1/\#N$;
3.      if $f(n) \leq f(s)$ then
4.          return $n$;
5.      else with probability $exp(\frac{-(f(n)-f(s))}{t})$
6.          return $n$;
7.      else
8.          return $s$;
```

- What happens for a large $\Delta = f(n) - f(s)$?

Metropolis Heuristic

```
1. function S-METROPOLIS[t](N,s)
2.     select $n \in N$ with probability $1/\#N$;
3.     if $f(n) \leq f(s)$ then
4.         return $n$;
5.     else with probability $exp(\frac{-(f(n)-f(s))}{t})$
6.         return $n$;
7.     else
8.         return $s$;
```

- What happens for a large $\Delta = f(n) - f(s)$?
  - the probability of accepting the move becomes very small

Metropolis Heuristic

```
1. function S-METROPOLIS[t](N,s)
2.     select $n \in N$ with probability $1/\#N$;
3.     if $f(n) \leq f(s)$ then
4.         return $n$;
5.     else with probability $exp(\frac{-(f(n)-f(s))}{t})$
6.         return $n$;
7.     else
8.         return $s$;
```

- What happens for a large t ?

Metropolis Heuristic

```
1. function S-METROPOLIS[t](N,s)
2.     select $n \in N$ with probability $1/\#N$;
3.     if $f(n) \leq f(s)$ then
4.         return $n$;
5.     else with probability $exp(\frac{-(f(n)-f(s))}{t})$
6.         return $n$;
7.     else
8.         return $s$;
```

- What happens for a large t ?
  - the probability of accepting a degrading move is large

Metropolis Heuristic

```
1. function S-METROPOLIS[t](N,s)
2.     select $n \in N$ with probability $1/\#N$;
3.     if $f(n) \leq f(s)$ then
4.         return $n$;
5.     else with probability $exp(\frac{-(f(n)-f(s))}{t})$
6.         return $n$;
7.     else
8.         return $s$;
```

- What happens for a small t ?

Metropolis Heuristic

```
1. function S-METROPOLIS[t](N,s)
2.     select $n \in N$ with probability $1/\#N$;
3.     if $f(n) \leq f(s)$ then
4.         return $n$;
5.     else with probability $exp(\frac{-(f(n)-f(s))}{t})$
6.         return $n$;
7.     else
8.         return $s$;
```

- What happens for a small t ?
  - the probability of accepting a degrading move is small

Simulated Annealing

- Based on statistical physics
  - heating and cooling schedules to produce crystals with few defects

Simulated Annealing

- Based on statistical physics
  - heating and cooling schedules to produce crystals with few defects
- Key idea

Simulated Annealing

- Based on statistical physics
  - heating and cooling schedules to produce crystals with few defects
- Key idea
  - Use Metropolis algorithm but adjust the temperature dynamically

Simulated Annealing

- Based on statistical physics
  - heating and cooling schedules to produce crystals with few defects
- Key idea
  - Use Metropolis algorithm but adjust the temperature dynamically
  - start with a high temperature
    - essentially a random walk

Simulated Annealing

- Based on statistical physics
  - heating and cooling schedules to produce crystals with few defects
- Key idea
  - Use Metropolis algorithm but adjust the temperature dynamically
  - start with a high temperature
    - essentially a random walk
  - decrease the temperature progressively

Simulated Annealing

- Based on statistical physics
  - heating and cooling schedules to produce crystals with few defects
- Key idea
  - Use Metropolis algorithm but adjust the temperature dynamically
  - start with a high temperature
    - essentially a random walk
  - decrease the temperature progressively
  - when the temperature is low
    - essentially a local improvement search

Simulated Annealing

```
1.  function SimulatedAnnealing($f$, $N$) {
2.      $s$ := generateInitialSolution();
3.      $t_1$ := initTemperature(s);
4.      $s^*$ := $s$;
5.      for $k$ := 1 to $MaxSearches$ do
6.          $s$ := LocalSearch($f$,$N$,L-All,S-Metropolis[$t_k$],$s$);
7.          if $f(s) < f(s^*)$ then
8.              $s^*$ := $s$;
9.          $t_{k+1}$ := updateTemperature($s$,$t_k$);
10.     return $s^*$;
11. }
```

Simulated Annealing

- guaranteed to converge to a global optimum
  - connected neighborhood
  - slow schedule
    - slower than exhaustive search

Simulated Annealing

- guaranteed to converge to a global optimum
  - connected neighborhood
  - slow schedule
    - slower than exhaustive search
- In practice
  - some excellent results on some hard benchmarks
    - e.g., TTP, minimizing tardiness in scheduling
  - reasonably fast schedule

$$t_{k+1} = \alpha \; t_k$$

Simulated Annealing

- Various additional techniques
  - restarts
  - reheats
  - see also tabu search later

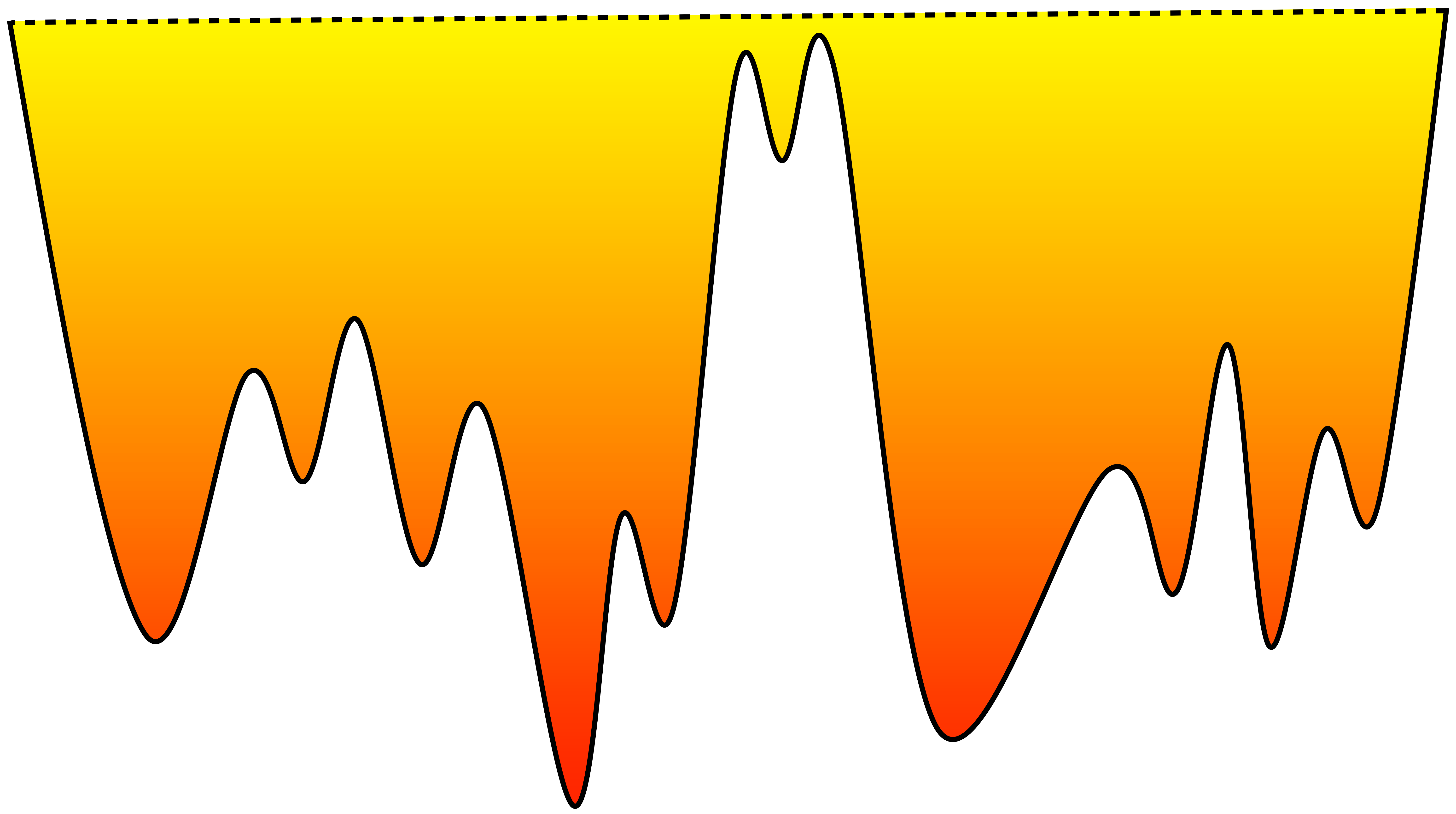
Simulated Annealing

- Various additional techniques
  - restarts
  - reheats
  - see also tabu search later
- Restarts
  - like in multi-start procedure

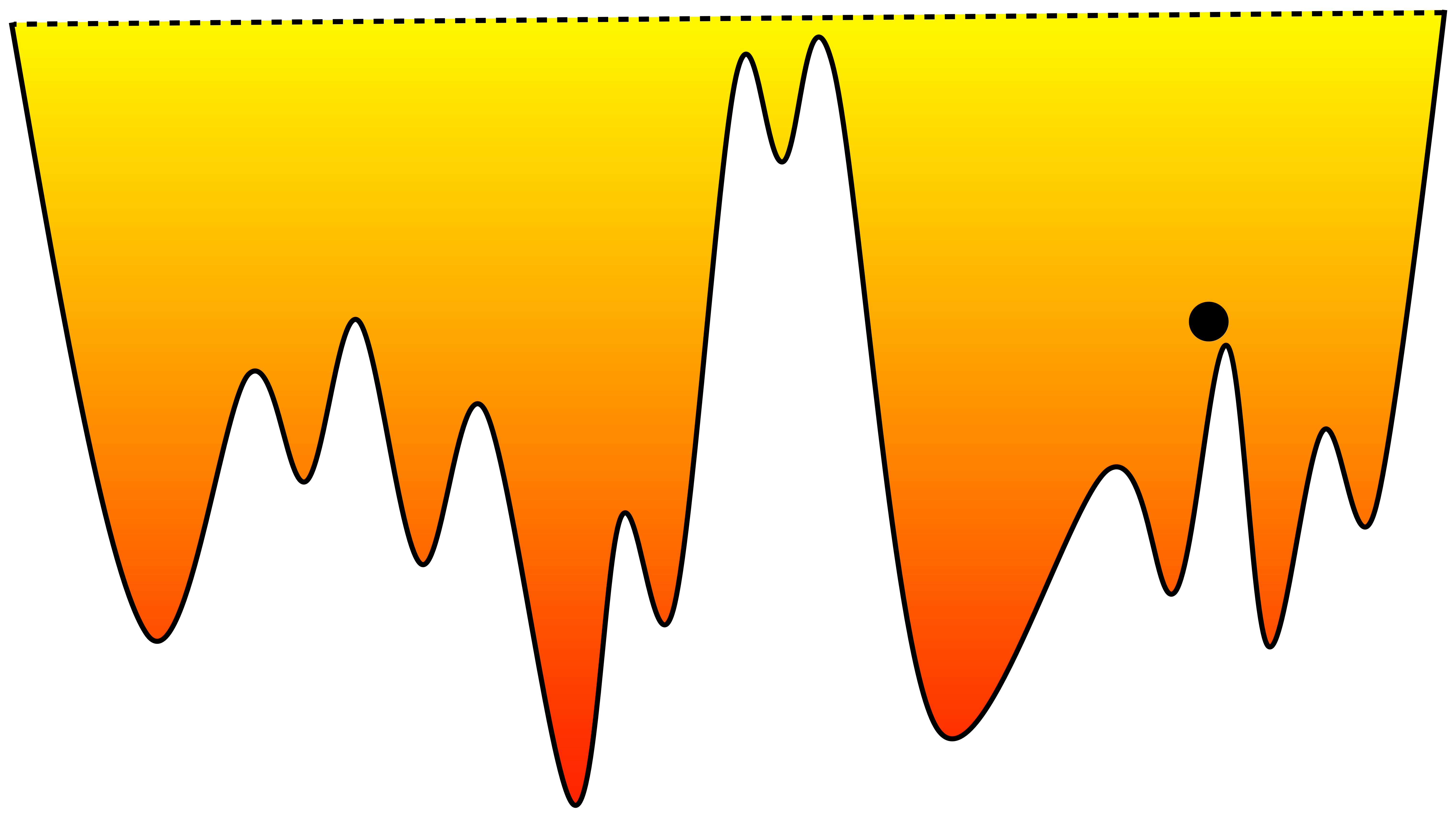
Simulated Annealing

- Various additional techniques
  - restarts
  - reheats
  - see also tabu search later
- Restarts
  - like in multi-start procedure
- Reheat
  - increase the temperature

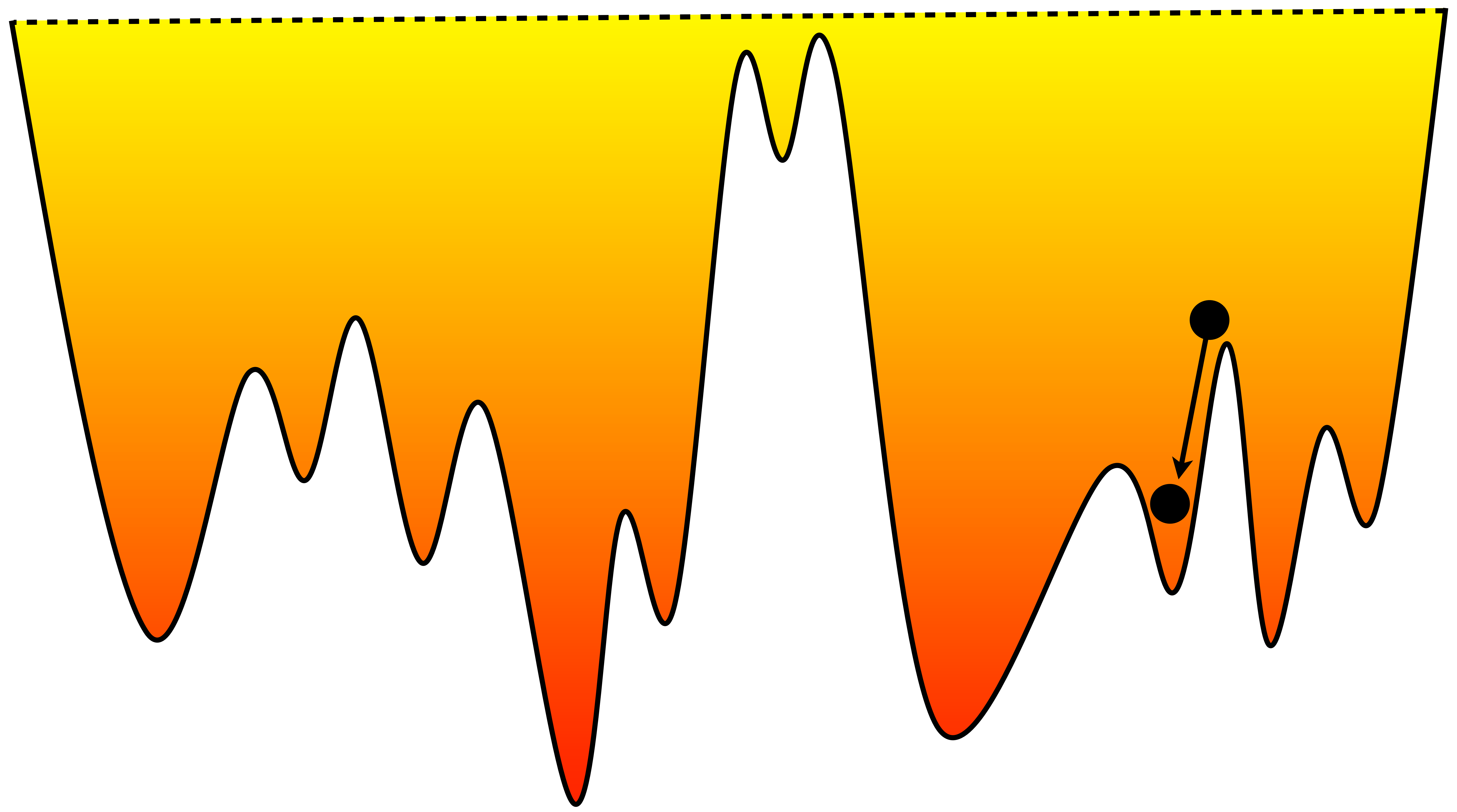
Tabu Search



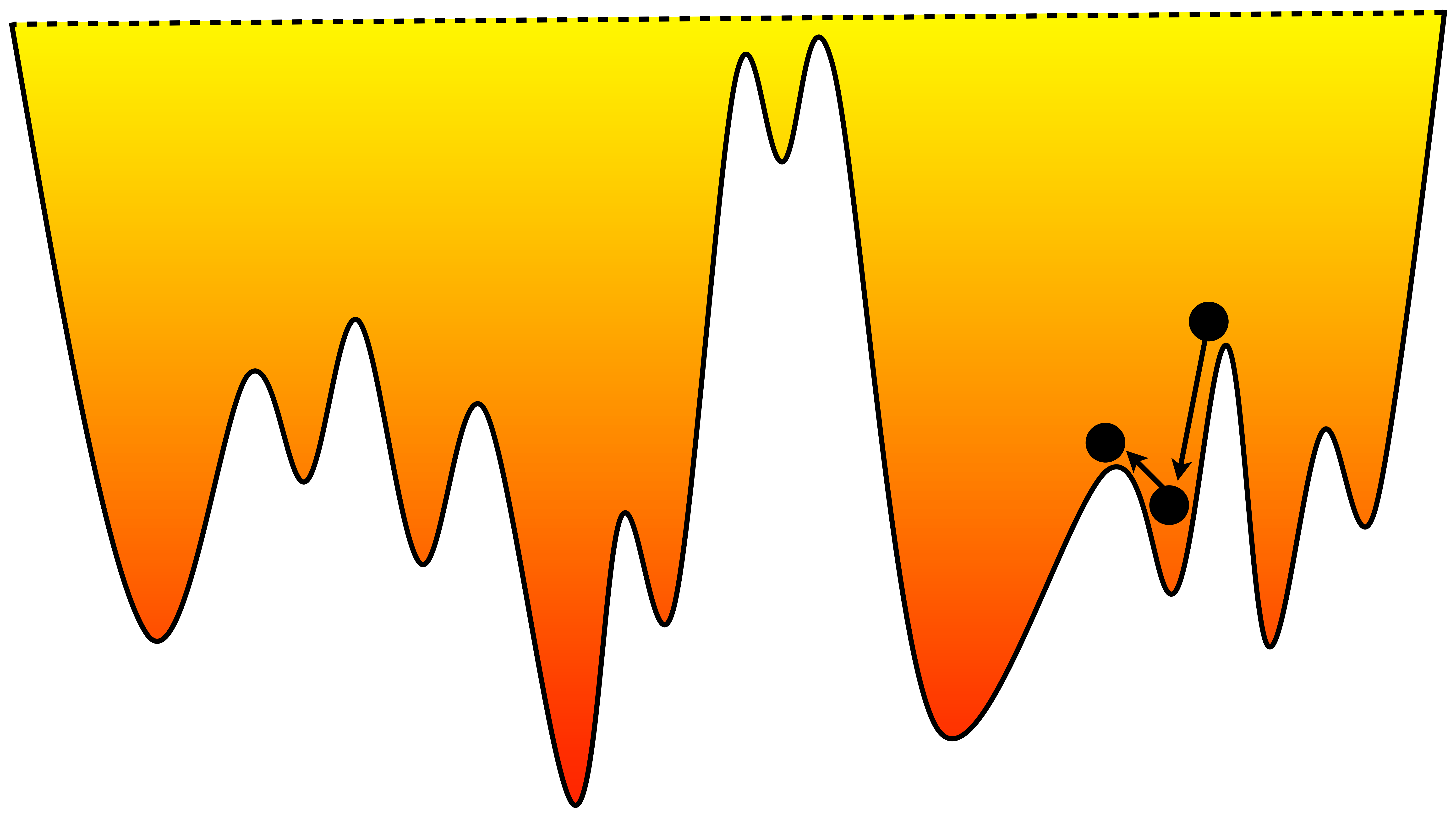
Tabu Search



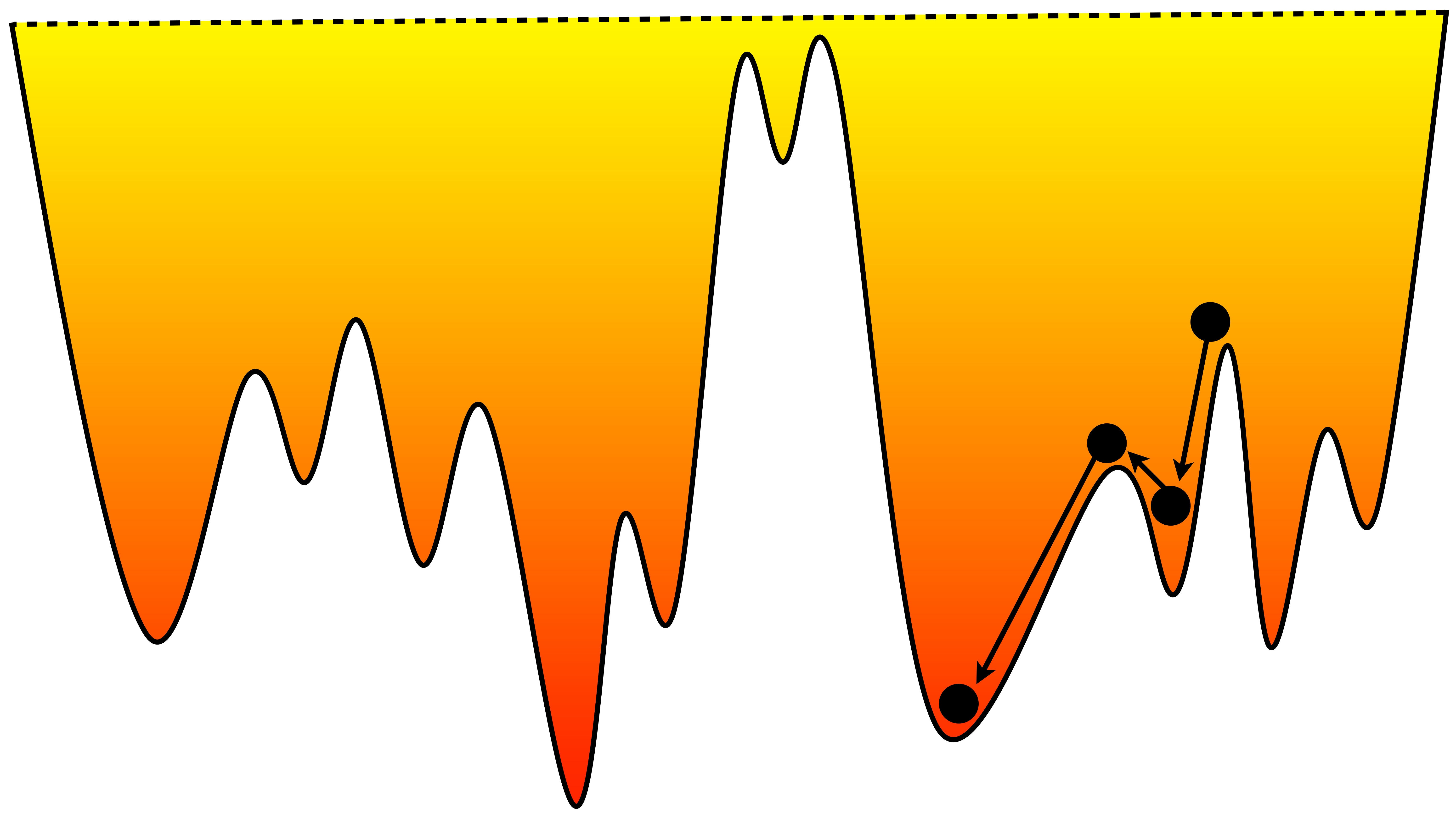
Tabu Search



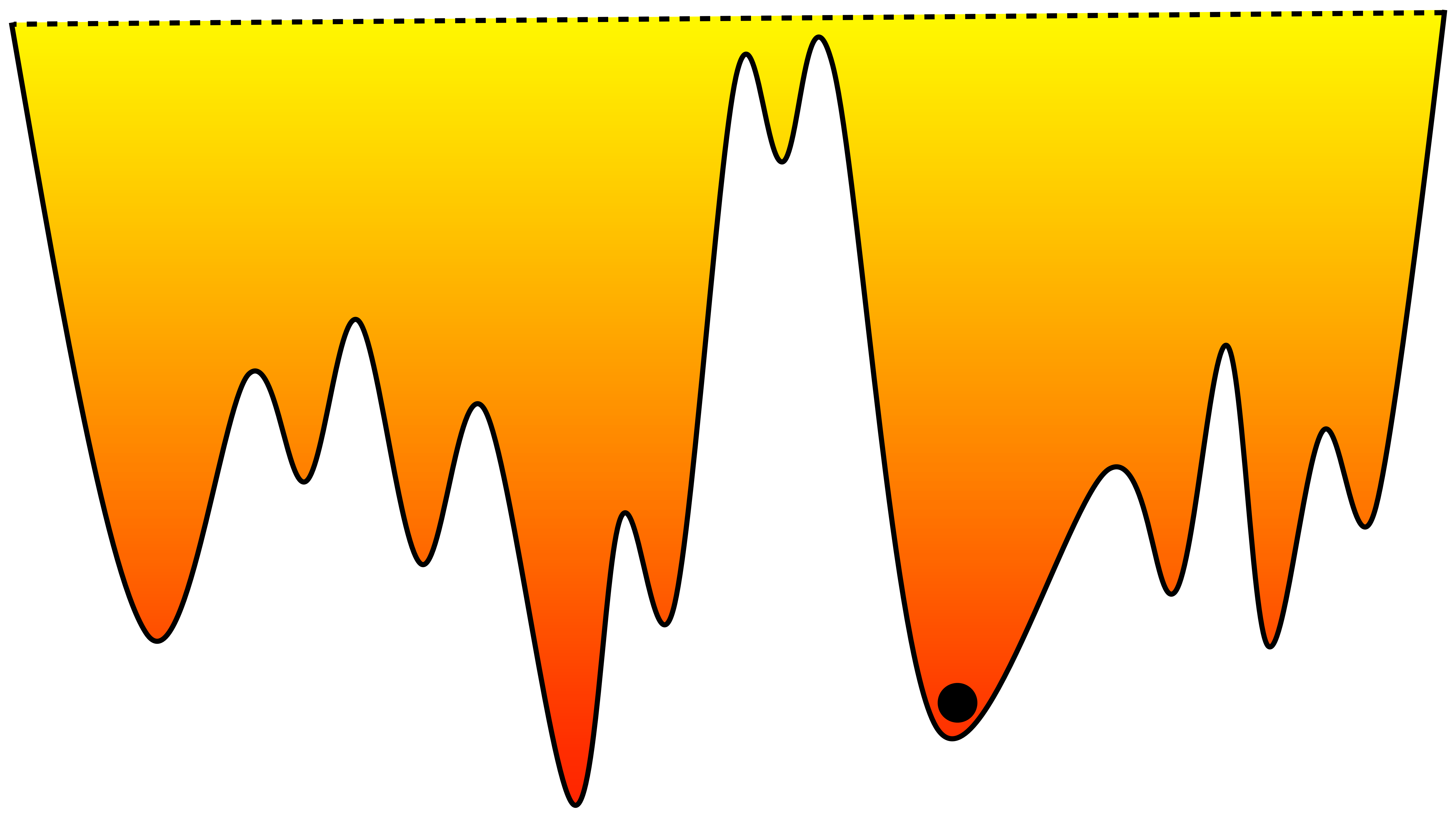
Tabu Search



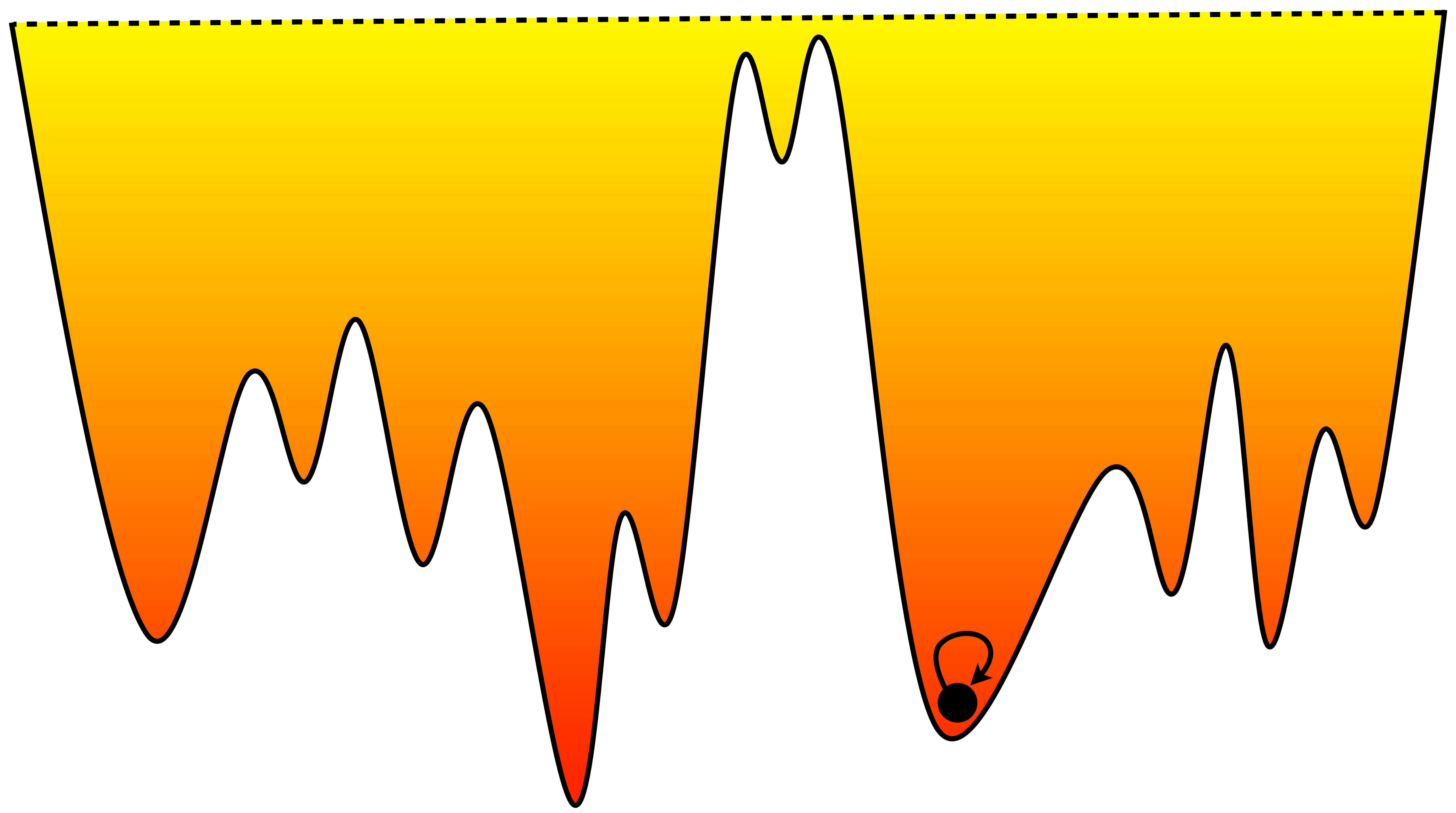
Tabu Search



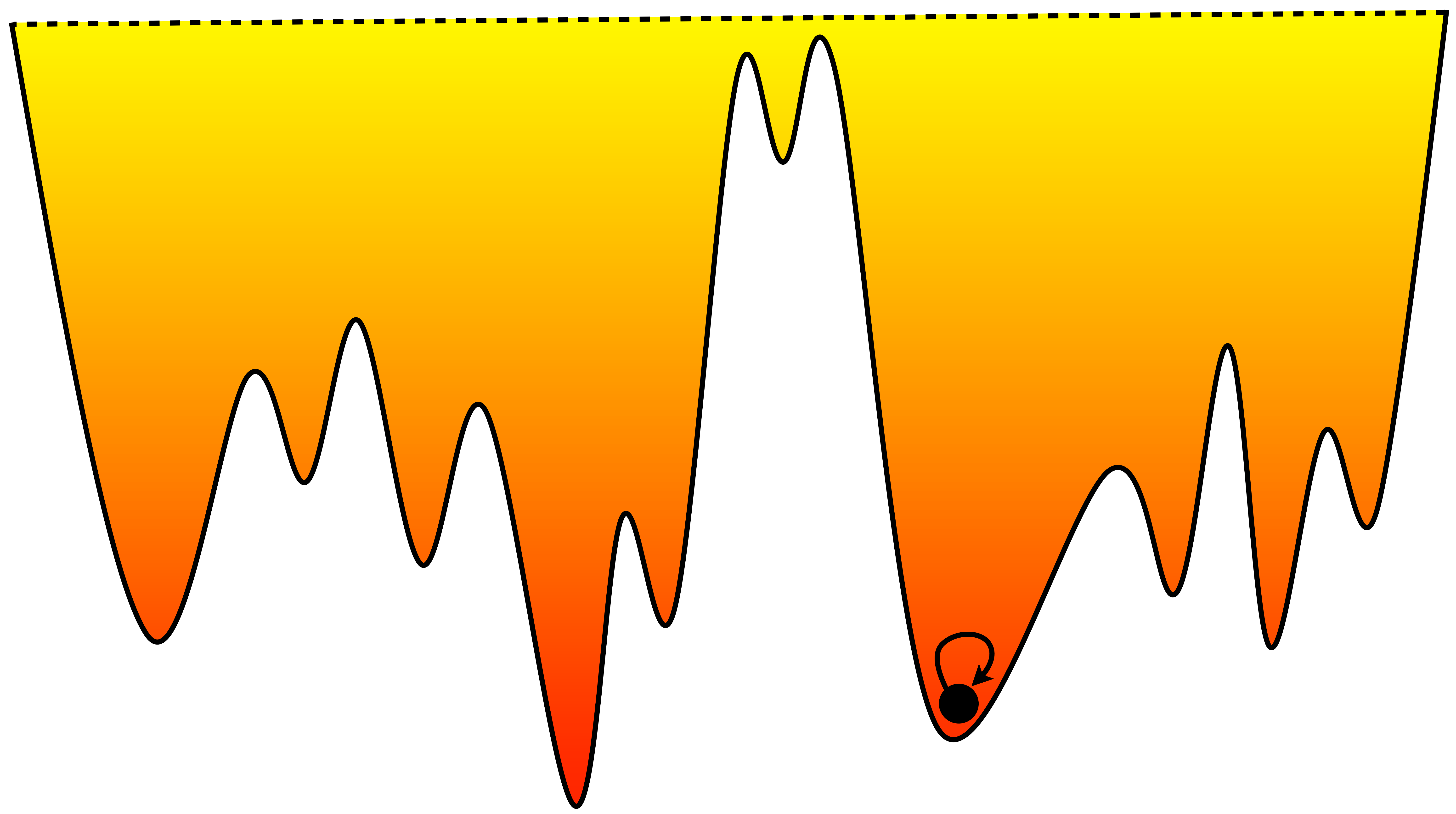
Tabu Search



Tabu Search

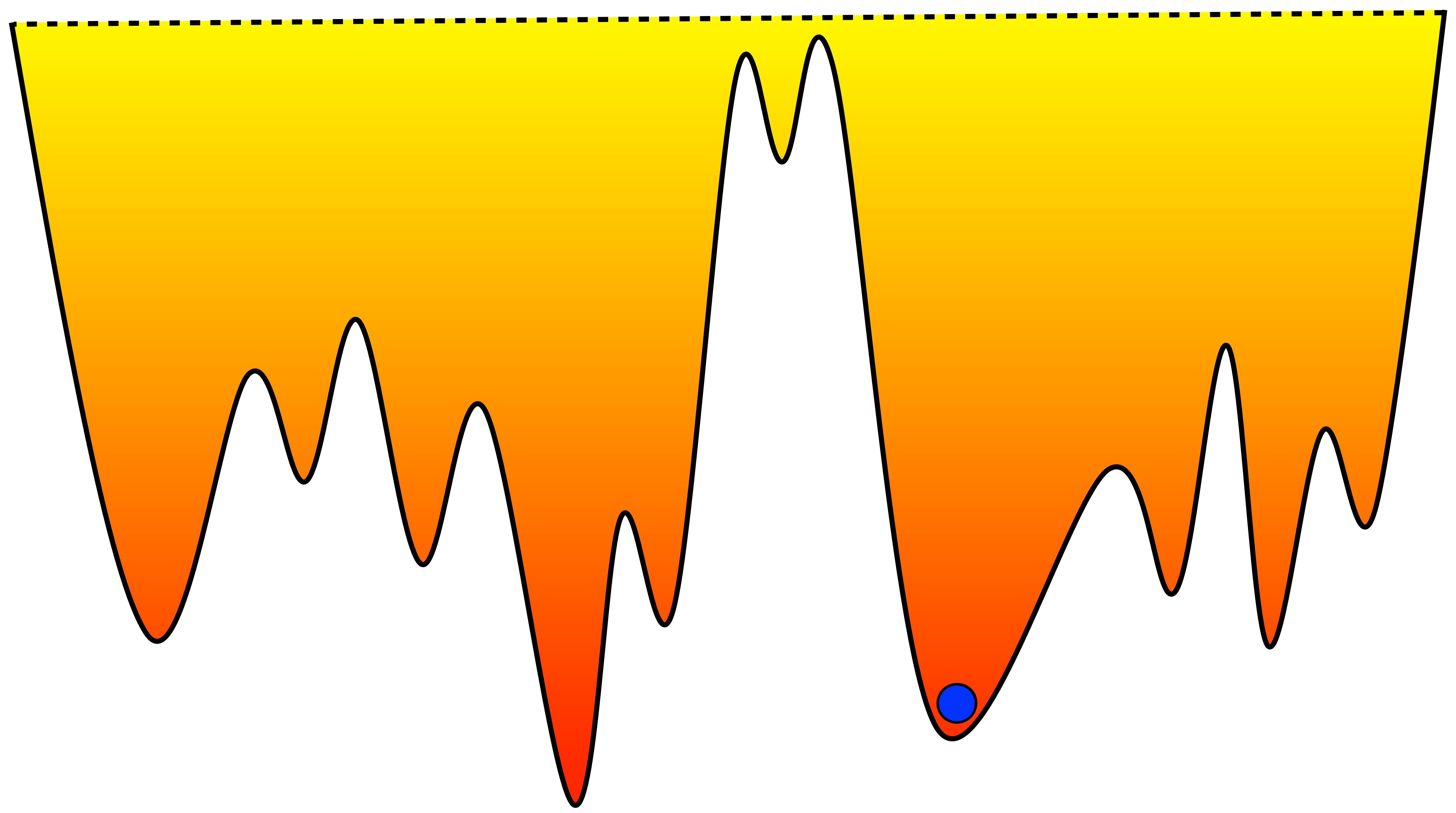


Tabu Search



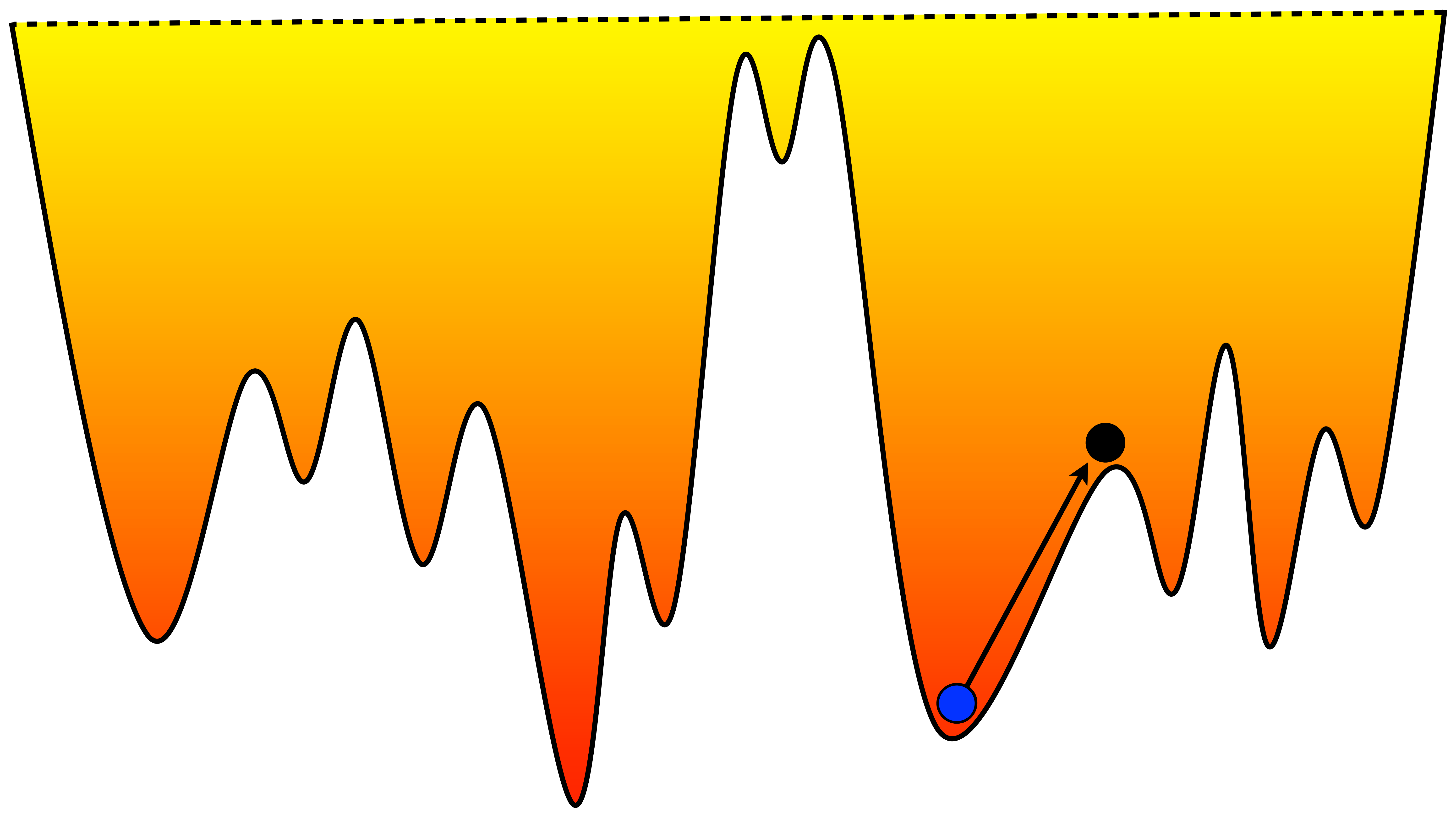
Tabu node ● : node I have already visited

Tabu Search



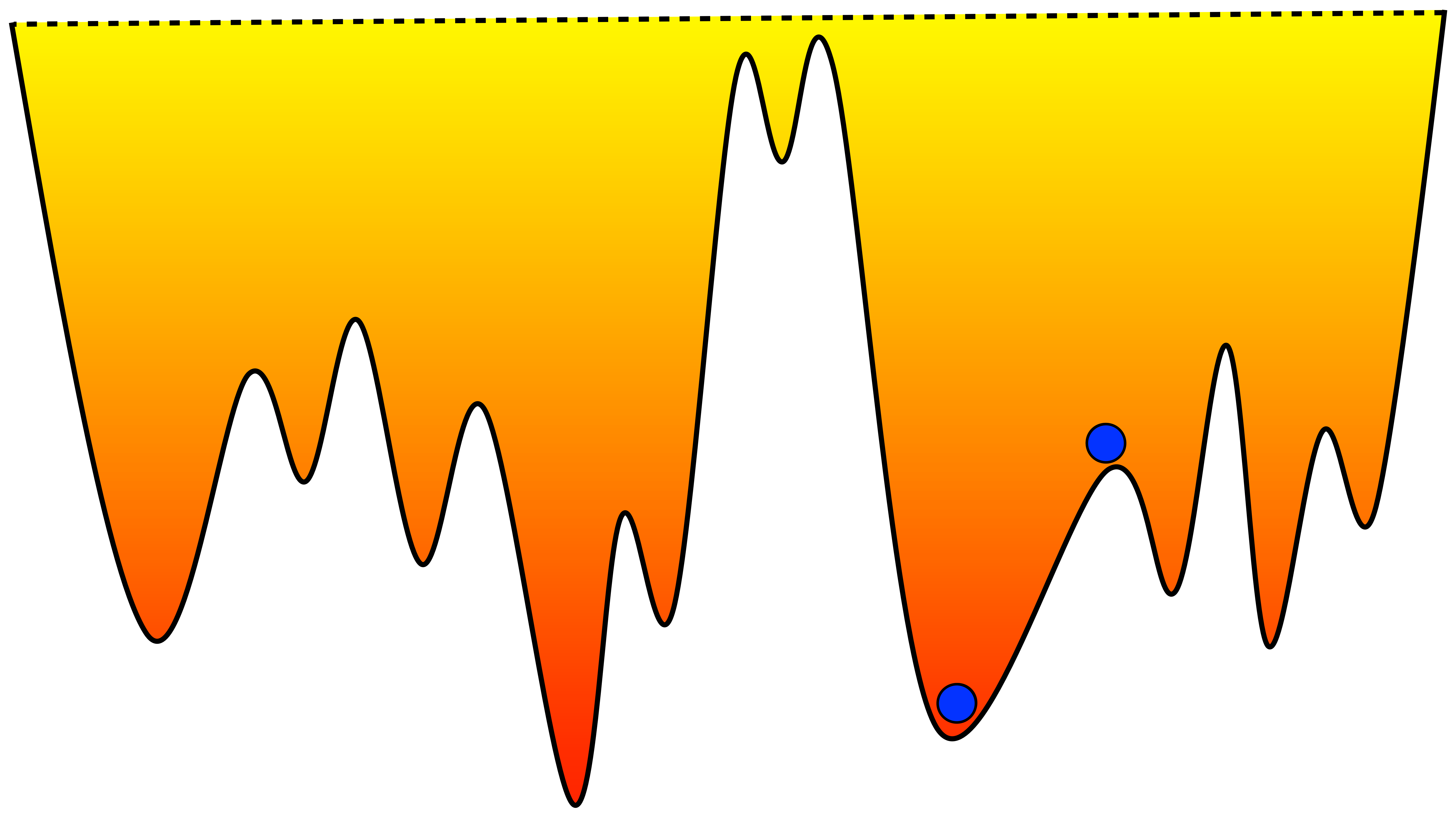
Tabu node ● : node I have already visited

Tabu Search



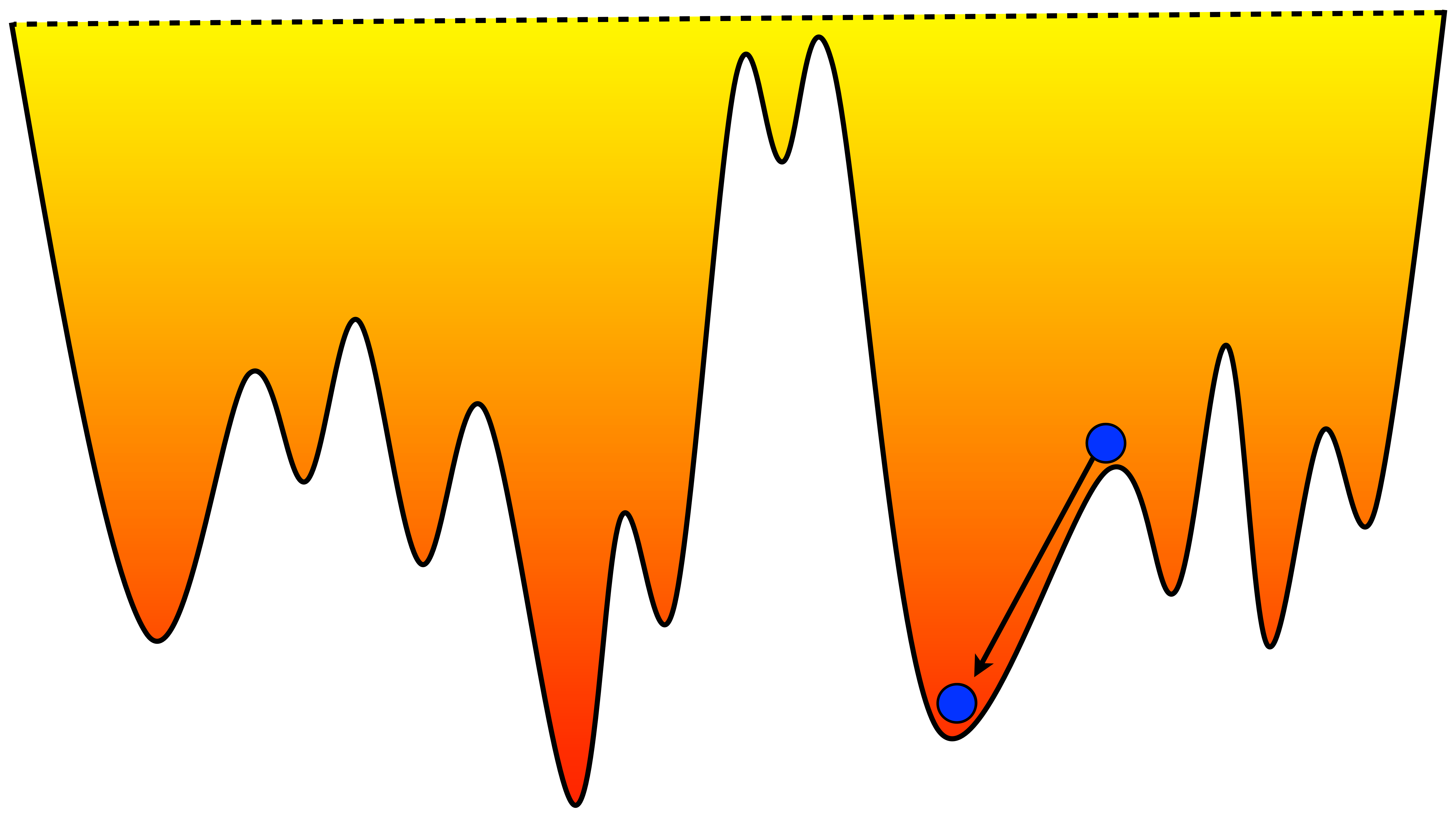
Tabu node ● : node I have already visited

Tabu Search



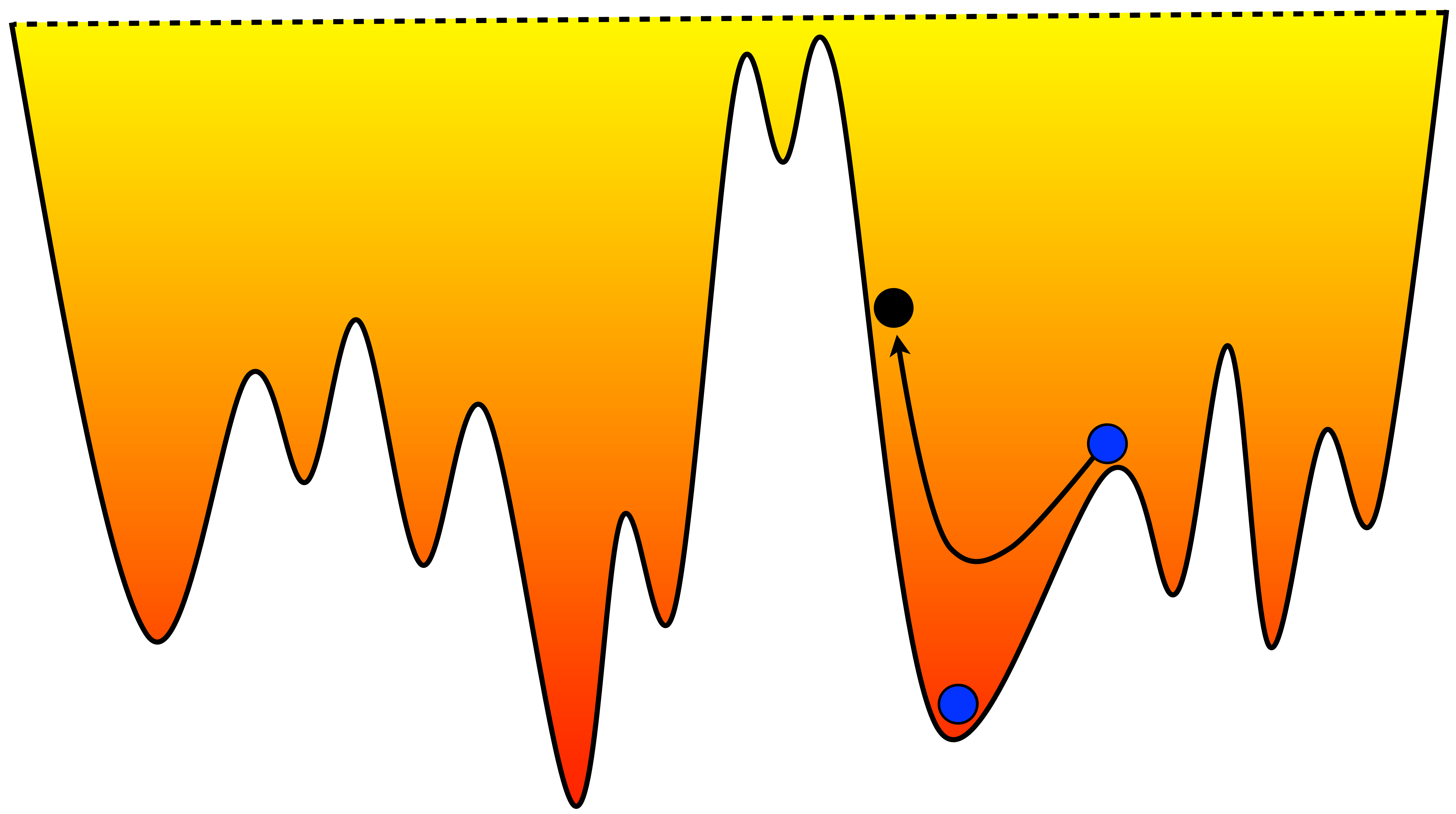
Tabu node ● : node I have already visited

Tabu Search



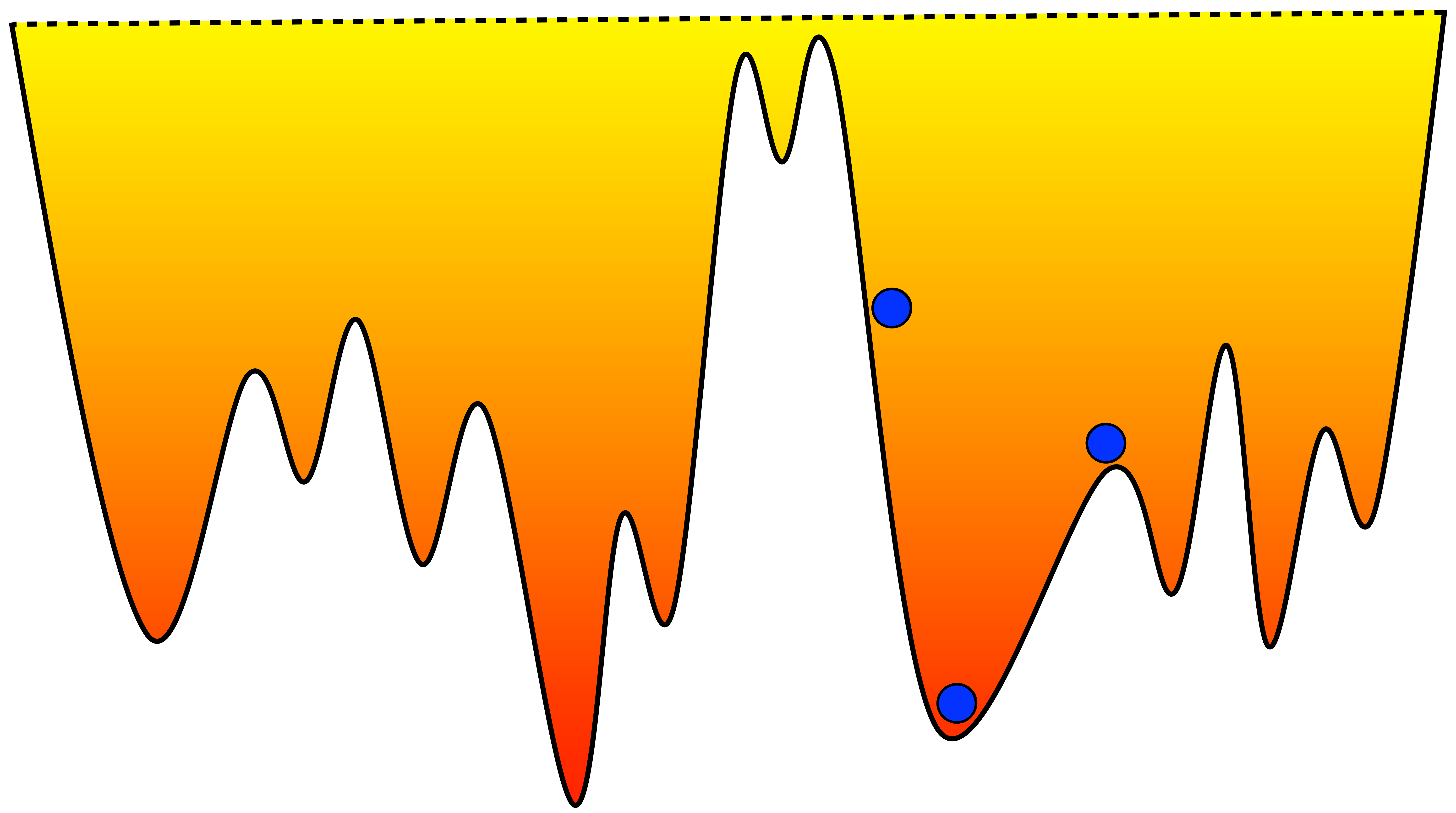
Tabu node ● : node I have already visited

Tabu Search



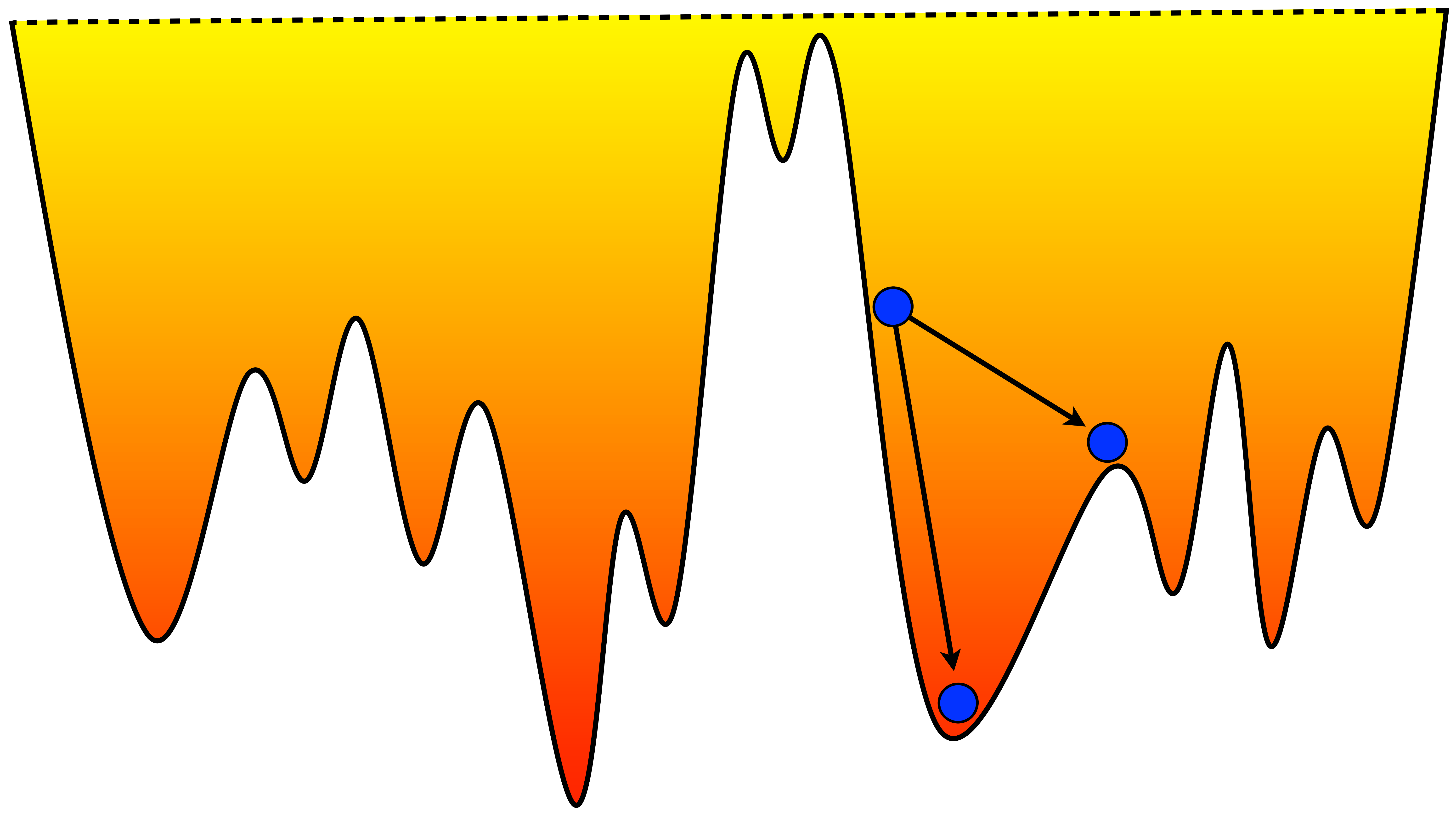
Tabu node ● : node I have already visited

Tabu Search



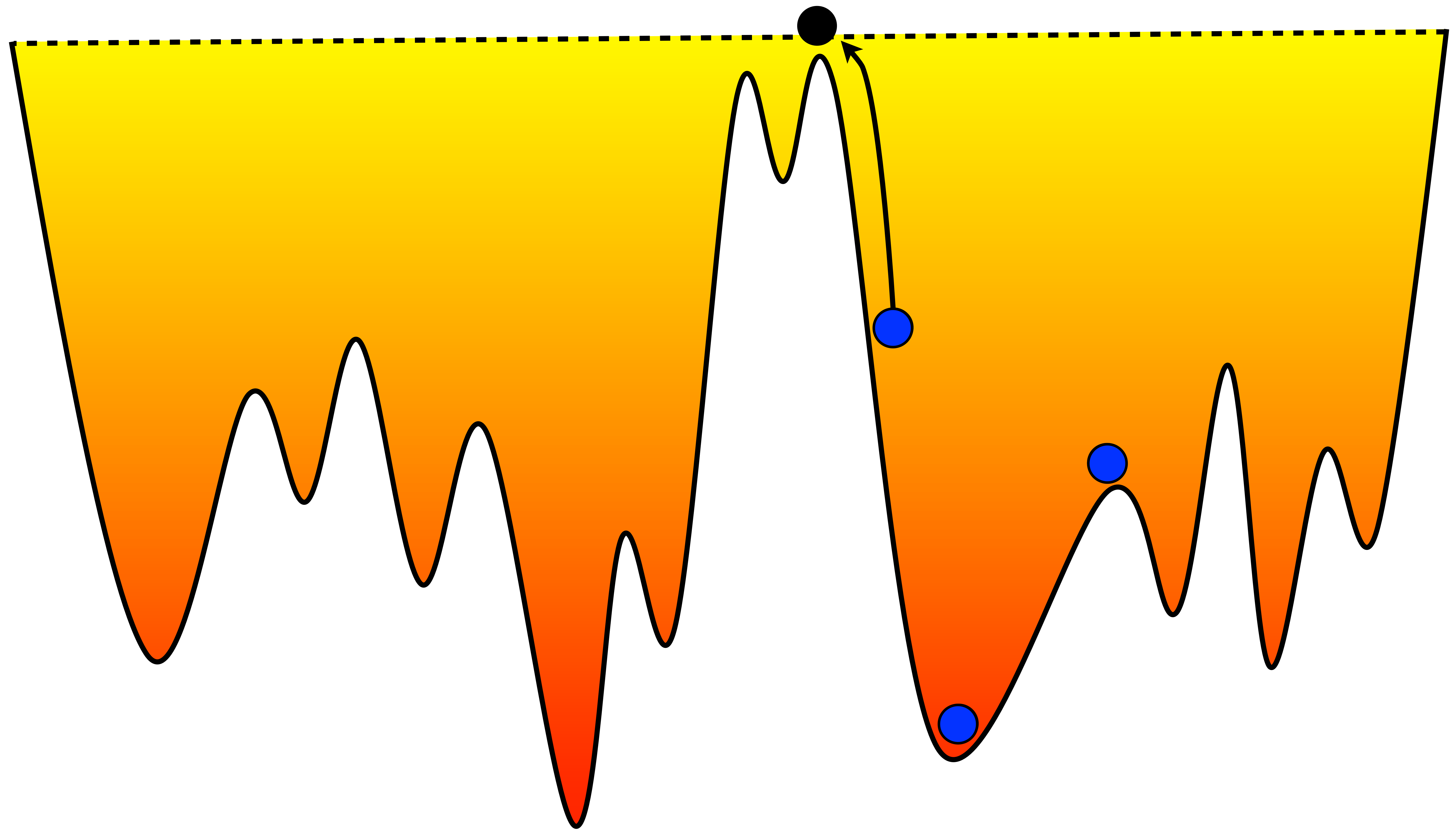
Tabu node ● : node I have already visited

Tabu Search



Tabu node ● : node I have already visited

Tabu Search



Tabu node ● : node I have already visited

Tabu Search

- Key abstract idea
  - maintain the sequence of nodes already visited
    - tabu list and tabu nodes

```
1.  function LocalSearch($f, N, L, S, s_1$) {
2.      $s^* := s_1$;
3.      $\tau := \langle s_1 \rangle$;
4.      for $k := 1$ to $MaxTrials$ do
5.          if $satisfiable(s) \wedge f(s_k) < f(s^*)$ then
6.              $s^* := s_k$;
7.          $s_{k+1} := S(L(N(s_k), \tau), \tau)$;
8.          $\tau := \tau :: s_{k+1}$;
9.      return $s^*$;
10. }
```

Tabu Search

- Basic abstract tabu-search
  - select the best configurations that is not tabu, i.e., has not been visited before

Tabu Search

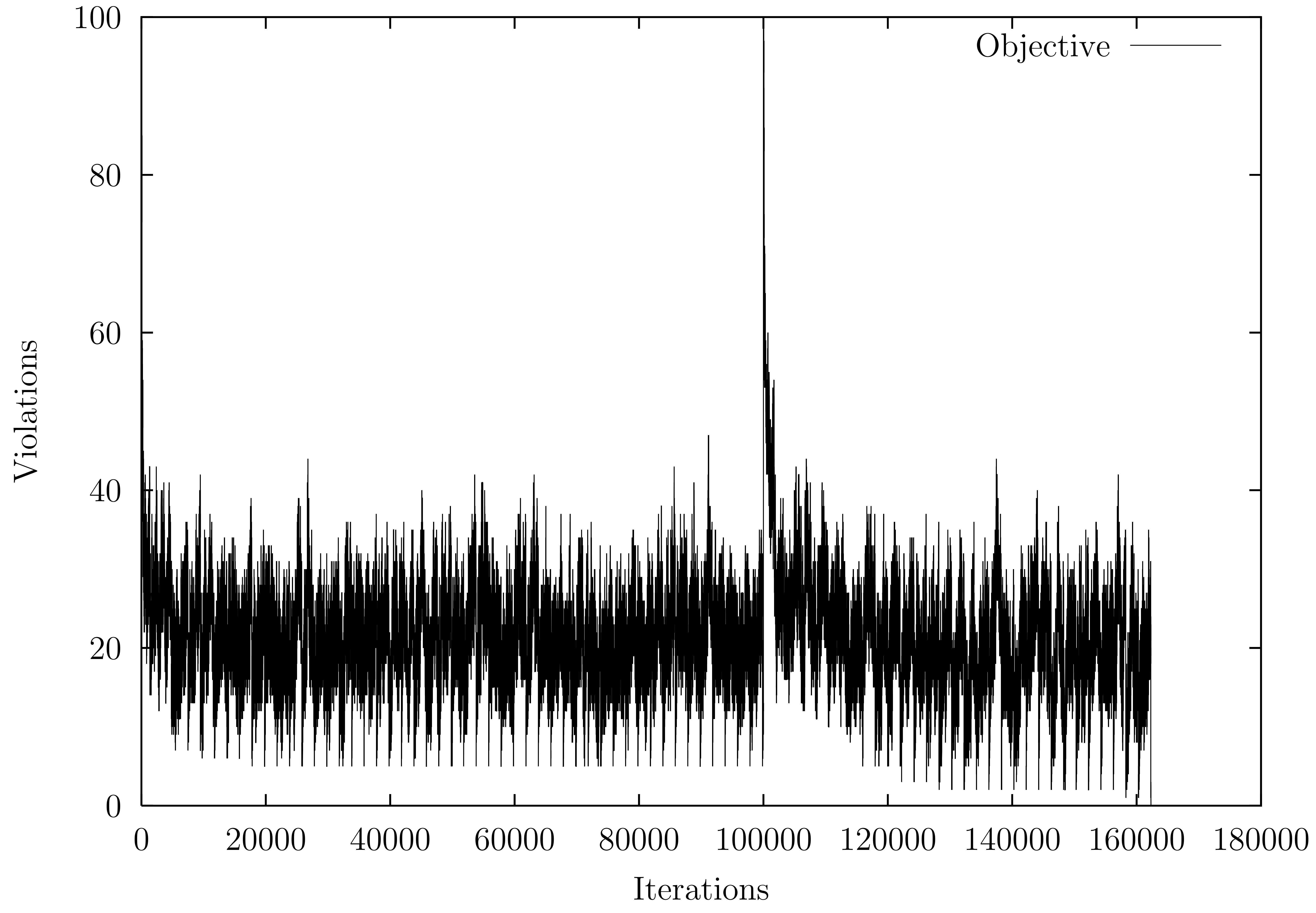
- Basic abstract tabu-search
  - select the best configurations that is not tabu, i.e., has not been visited before

```
1.   **function** TabuSearch($f$,$N$,$s$)
2.       **return** LocalSearch($f$,$N$,L-NotTabu,S-Best);
```

where

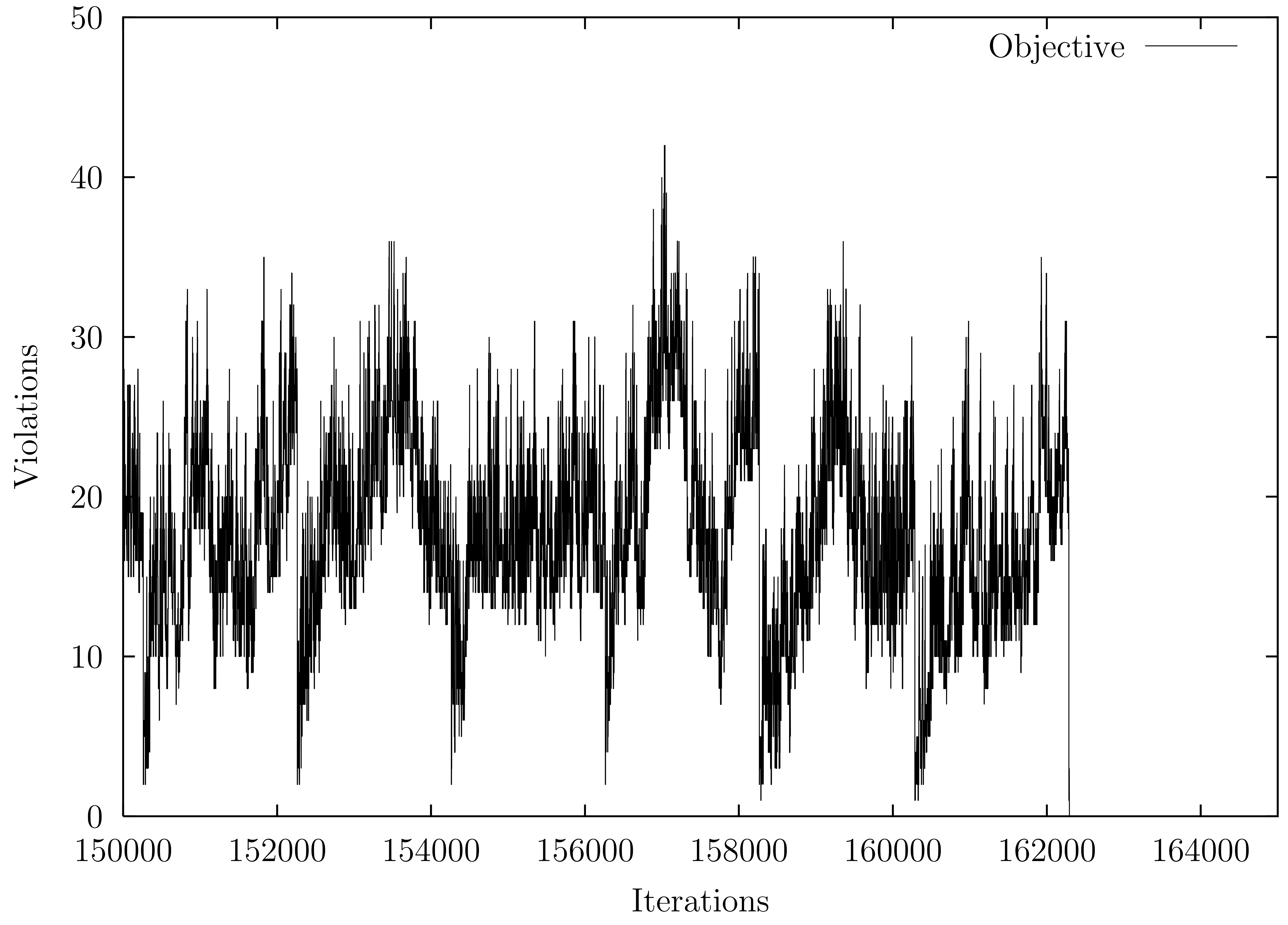
```
1.   **function** L-NotTabu($N$,$\tau$)
2.       **return** $\{ n \in N \mid n \notin \tau \}$;
```

Tabu Search



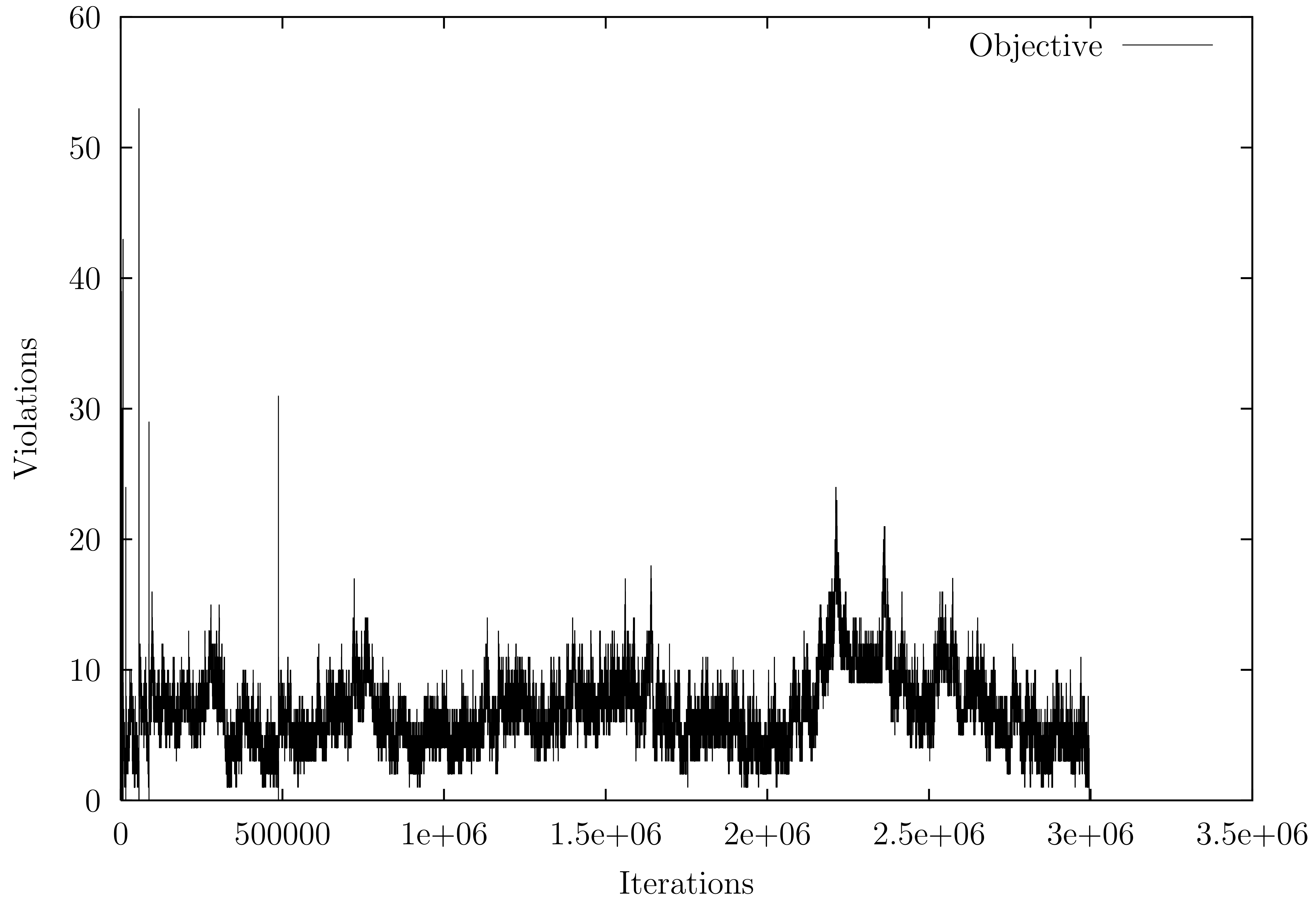
Objective
Violations
Iterations
0
20
40
60
80
100
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

Tabu Search



Objective
Violations
Iterations
0
10
20
30
40
50
150000
152000
154000
156000
158000
160000
162000
164000

Tabu Search



Objective
Violations
Iterations
0
10
20
30
40
50
60
0
500000
1e+06
1.5e+06
2e+06
2.5e+06
3e+06
3.5e+06

Metaheuristics

- Many others
  - variable neighborhood search
  - guided local search
  - ant-colony optimization
  - hybrid evolutionary algorithms
  - scatter search
  - ...

Until Next Time