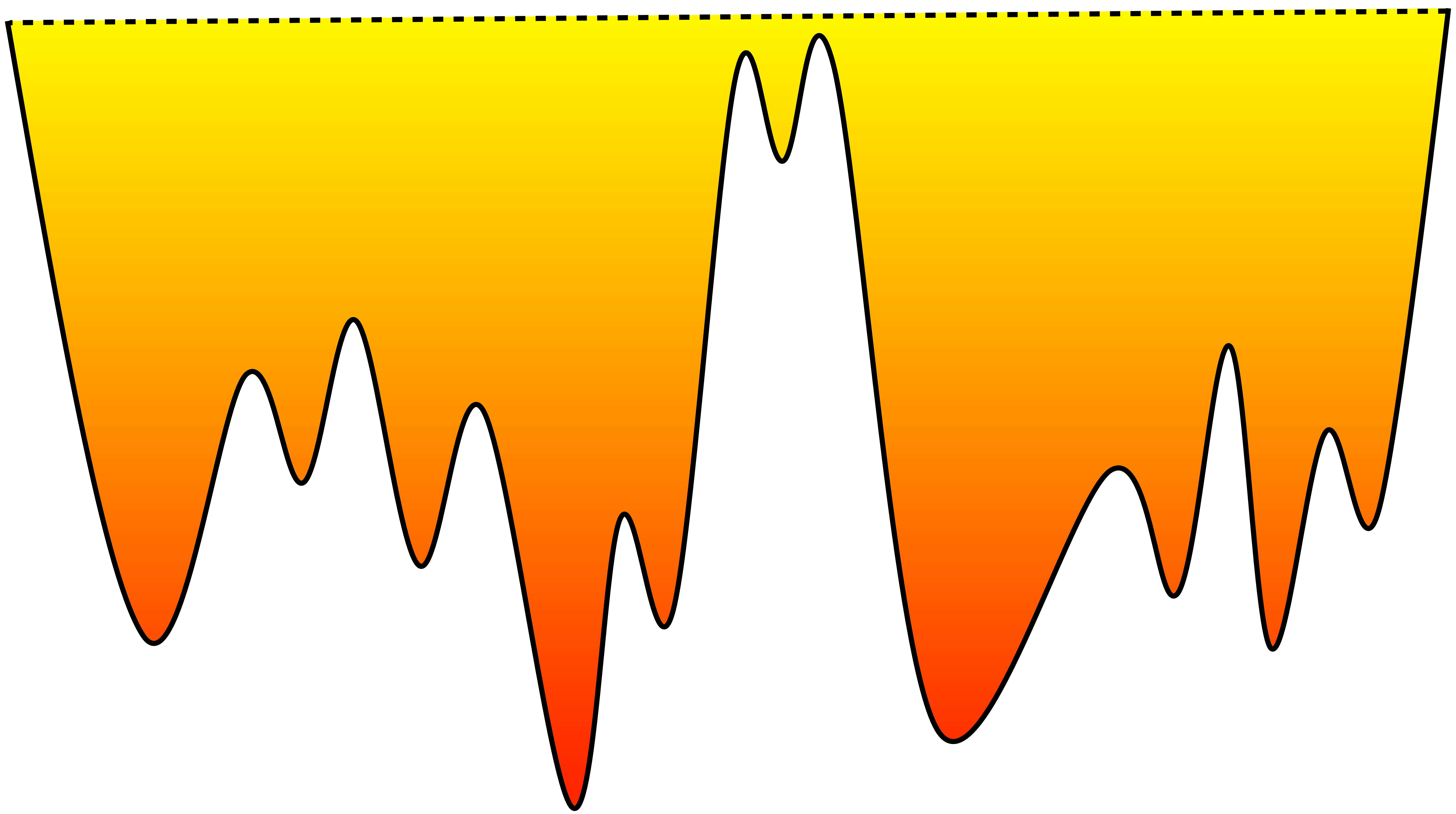
Discrete Optimization

Local Search: Part IX

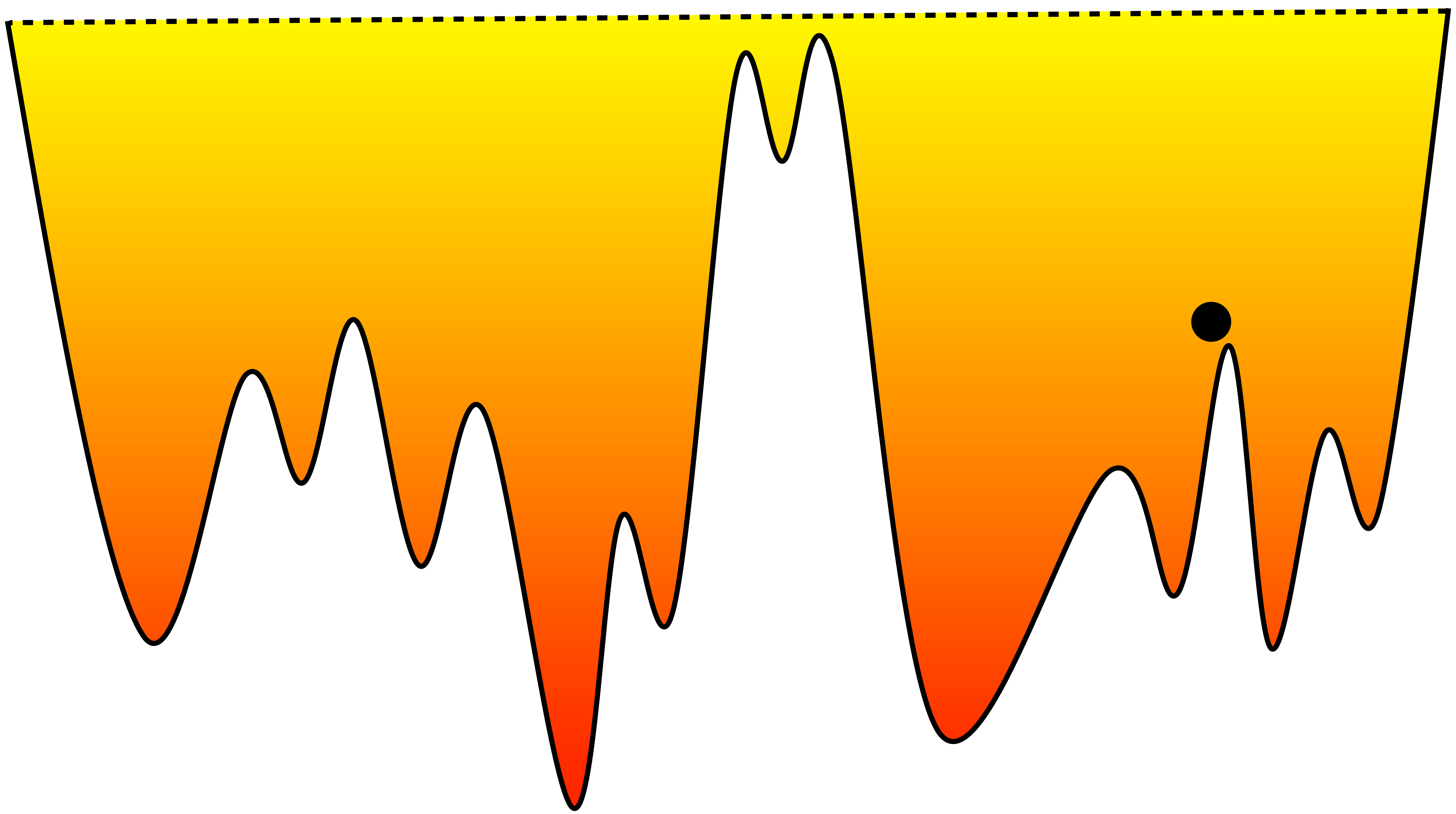
Goal of the Lecture

- Local search
  - tabu-search

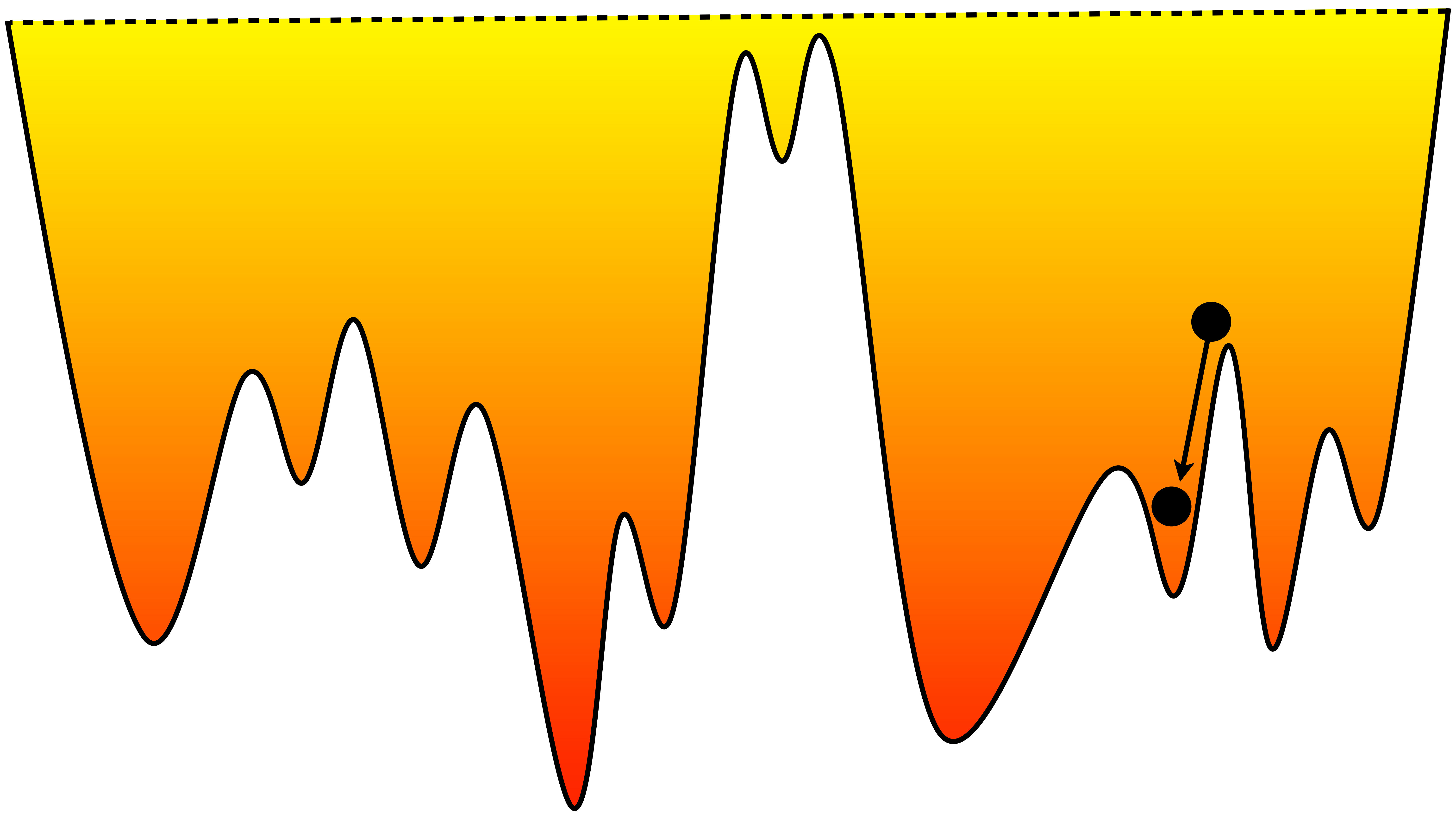
Tabu Search



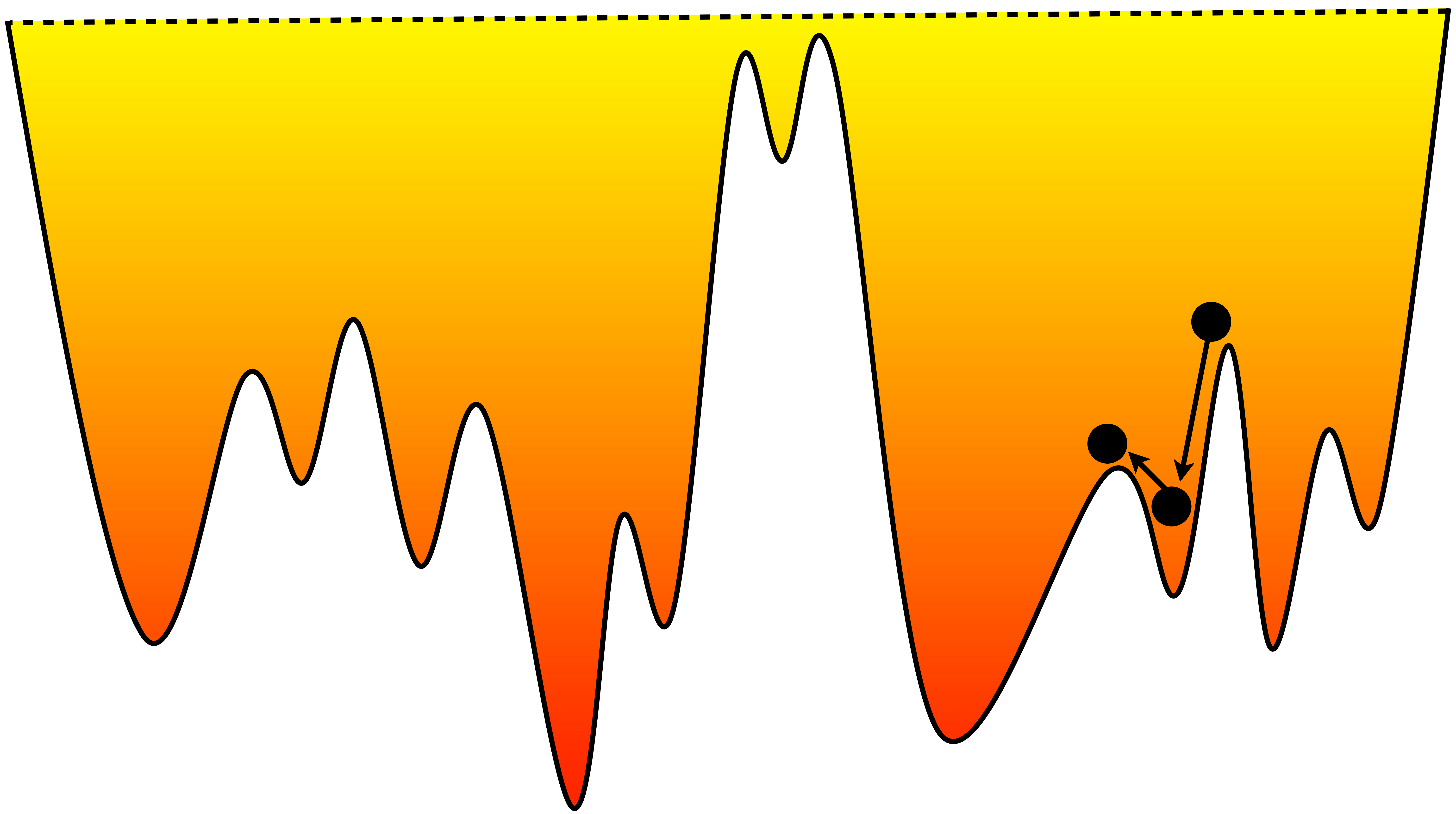
Tabu Search



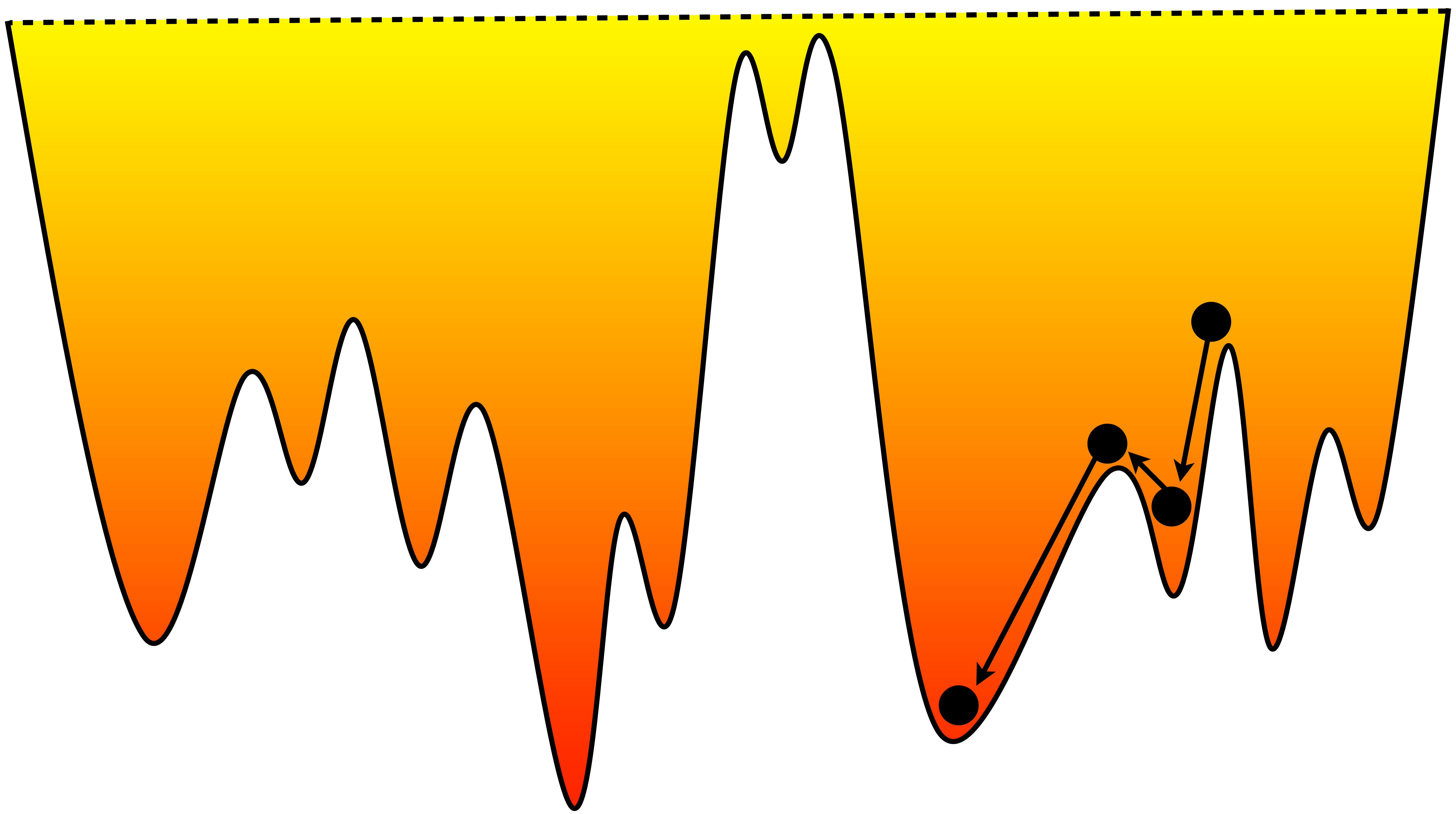
Tabu Search



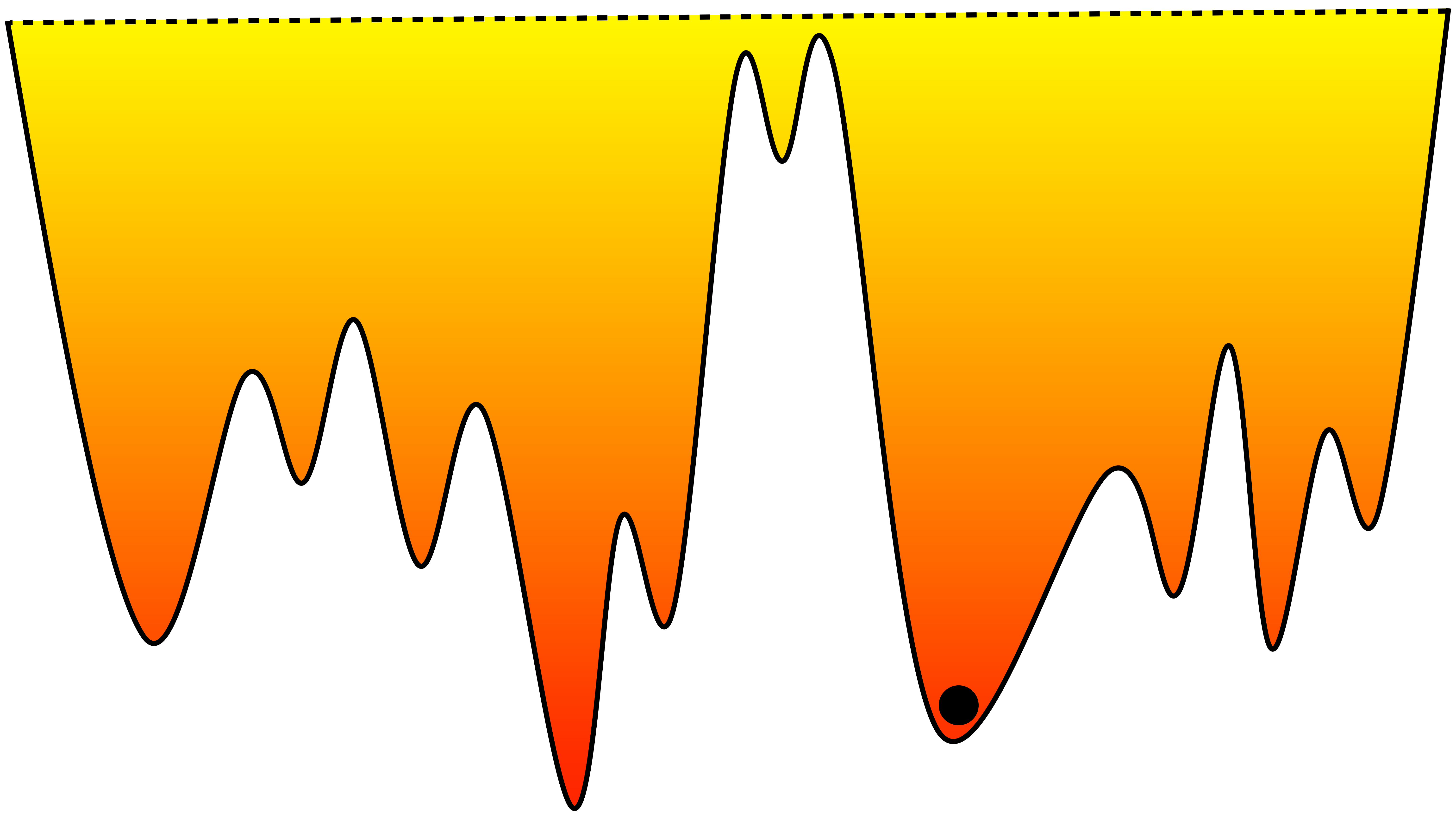
Tabu Search



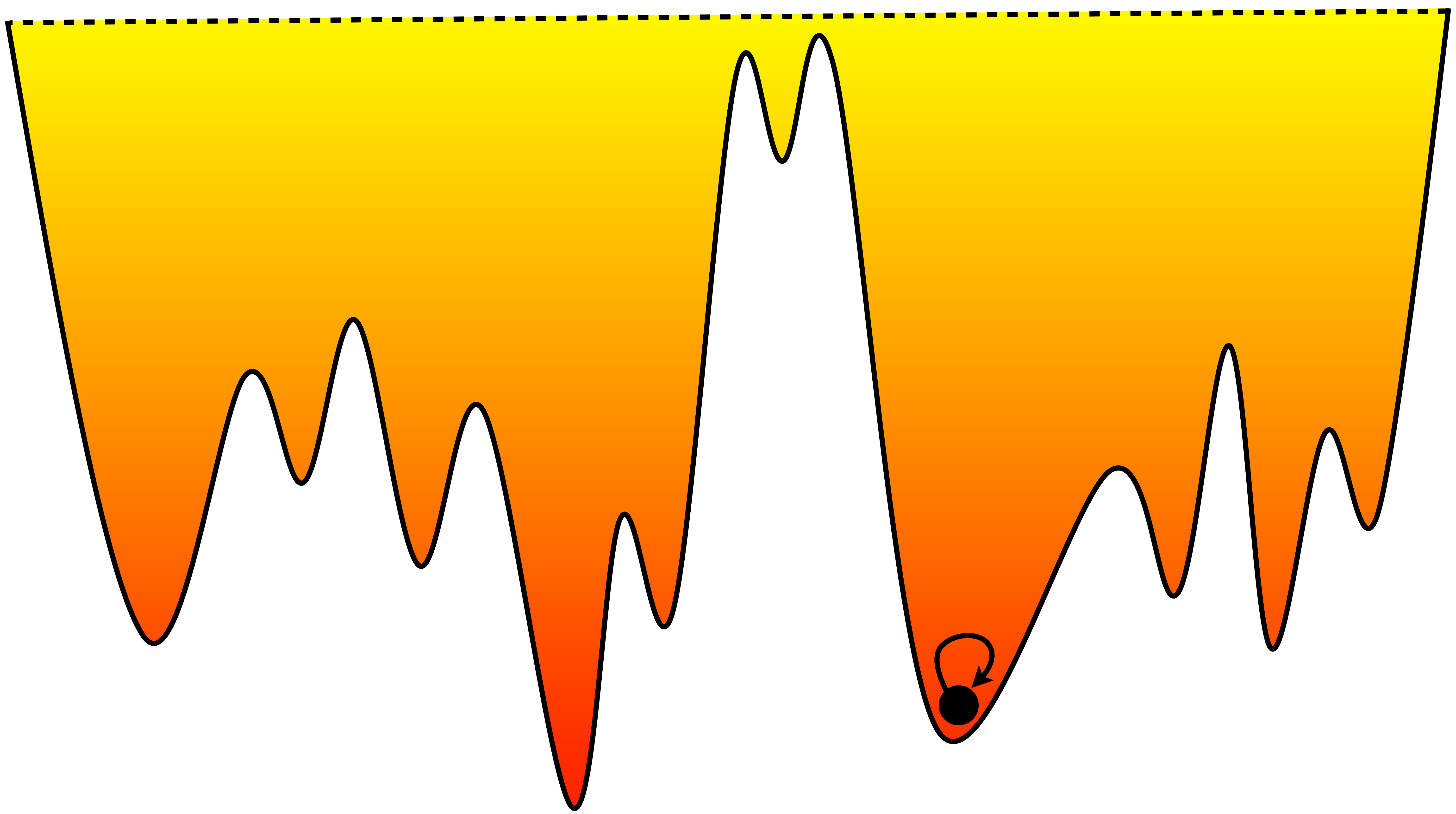
Tabu Search



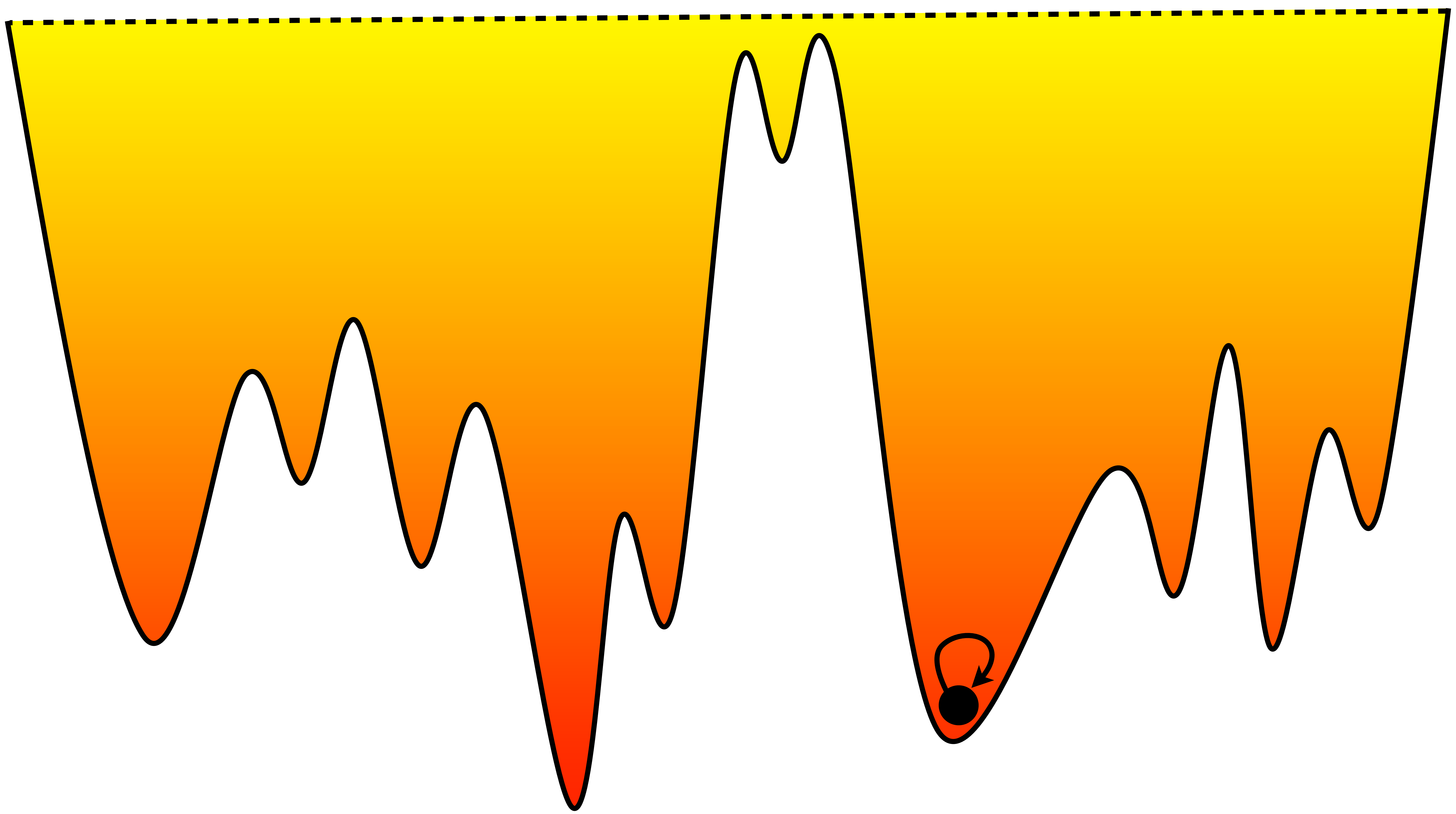
Tabu Search



Tabu Search

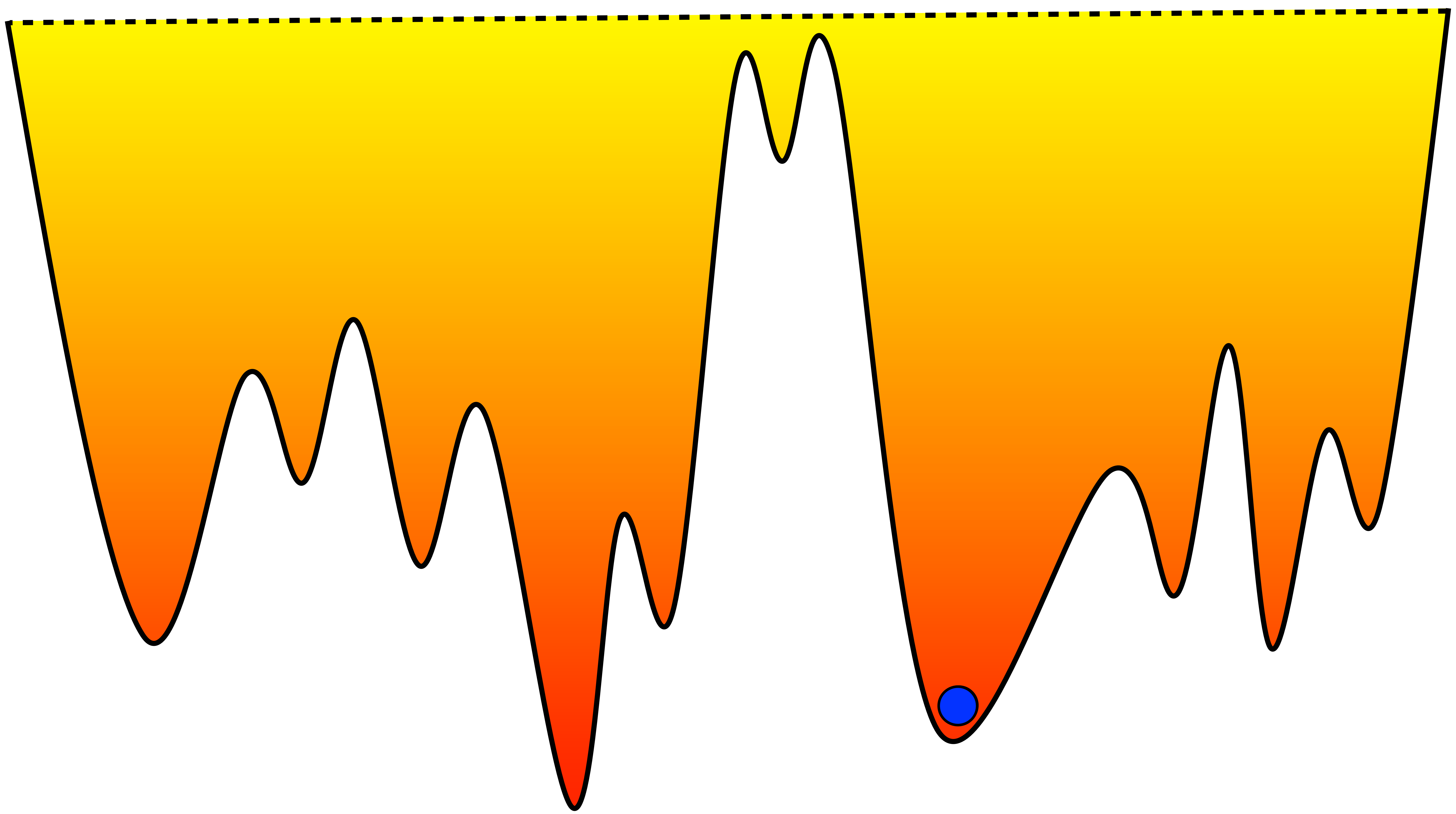


Tabu Search



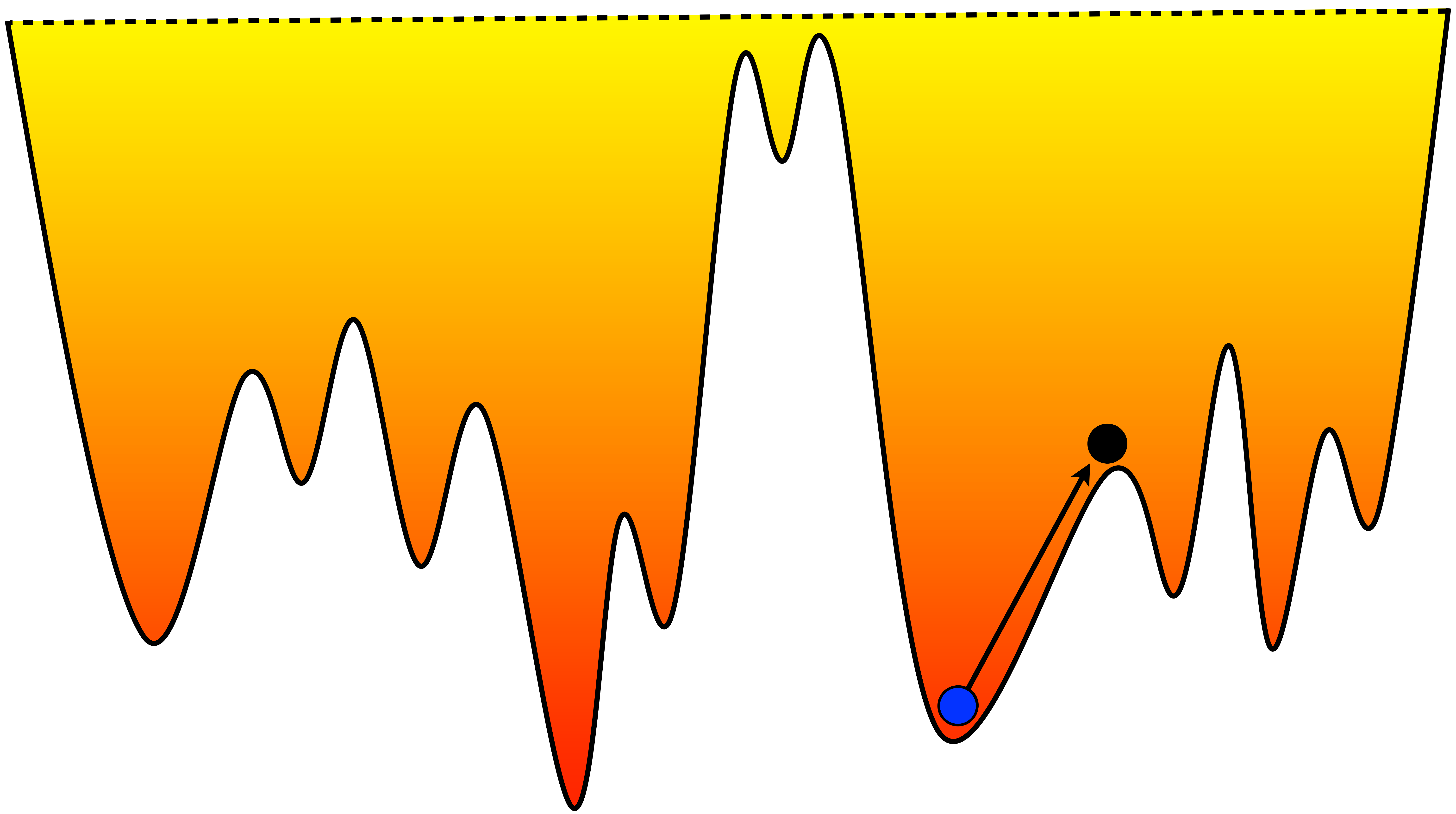
Tabu node : node I have already visited

Tabu Search



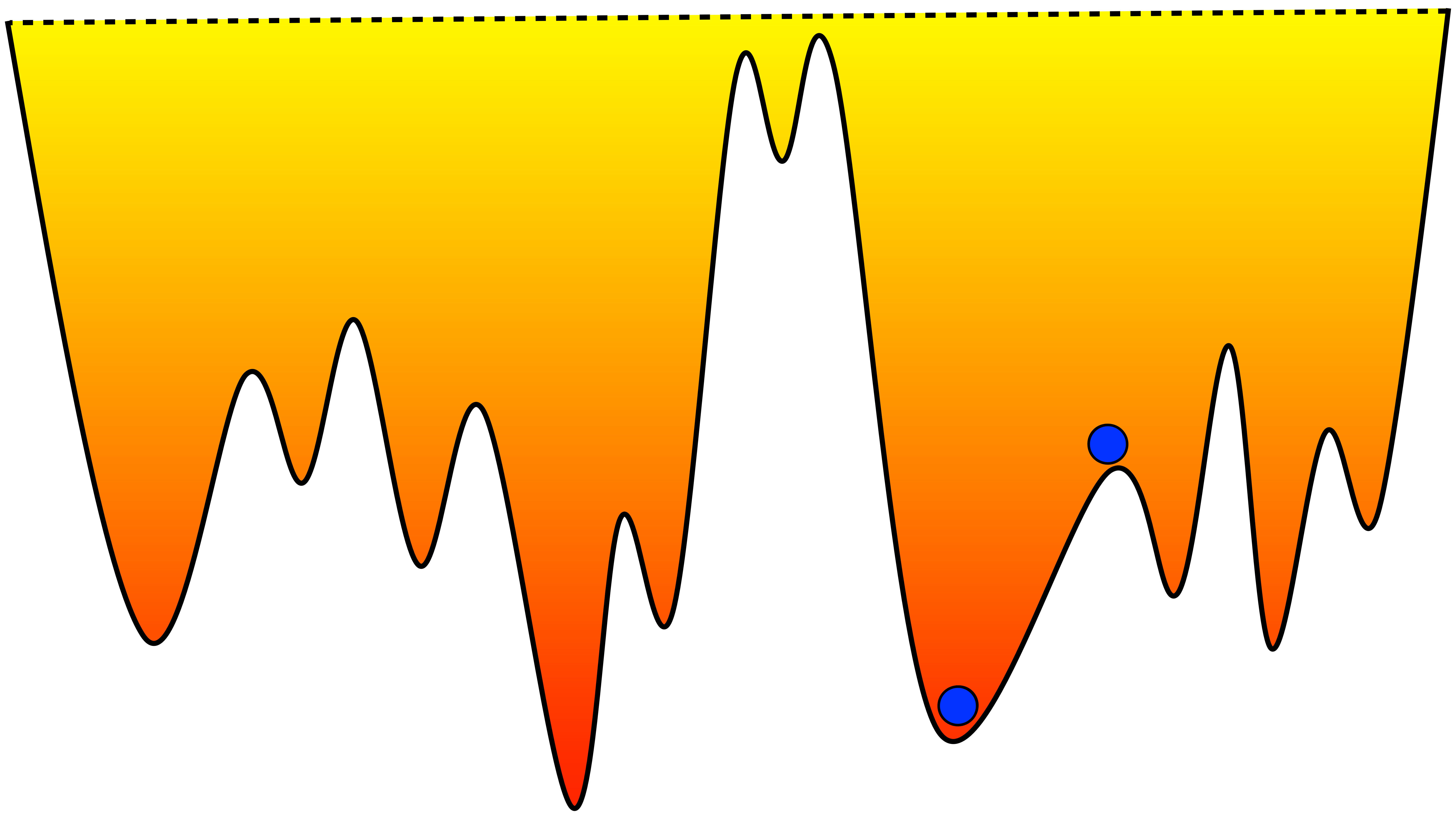
Tabu node ● : node I have already visited

Tabu Search



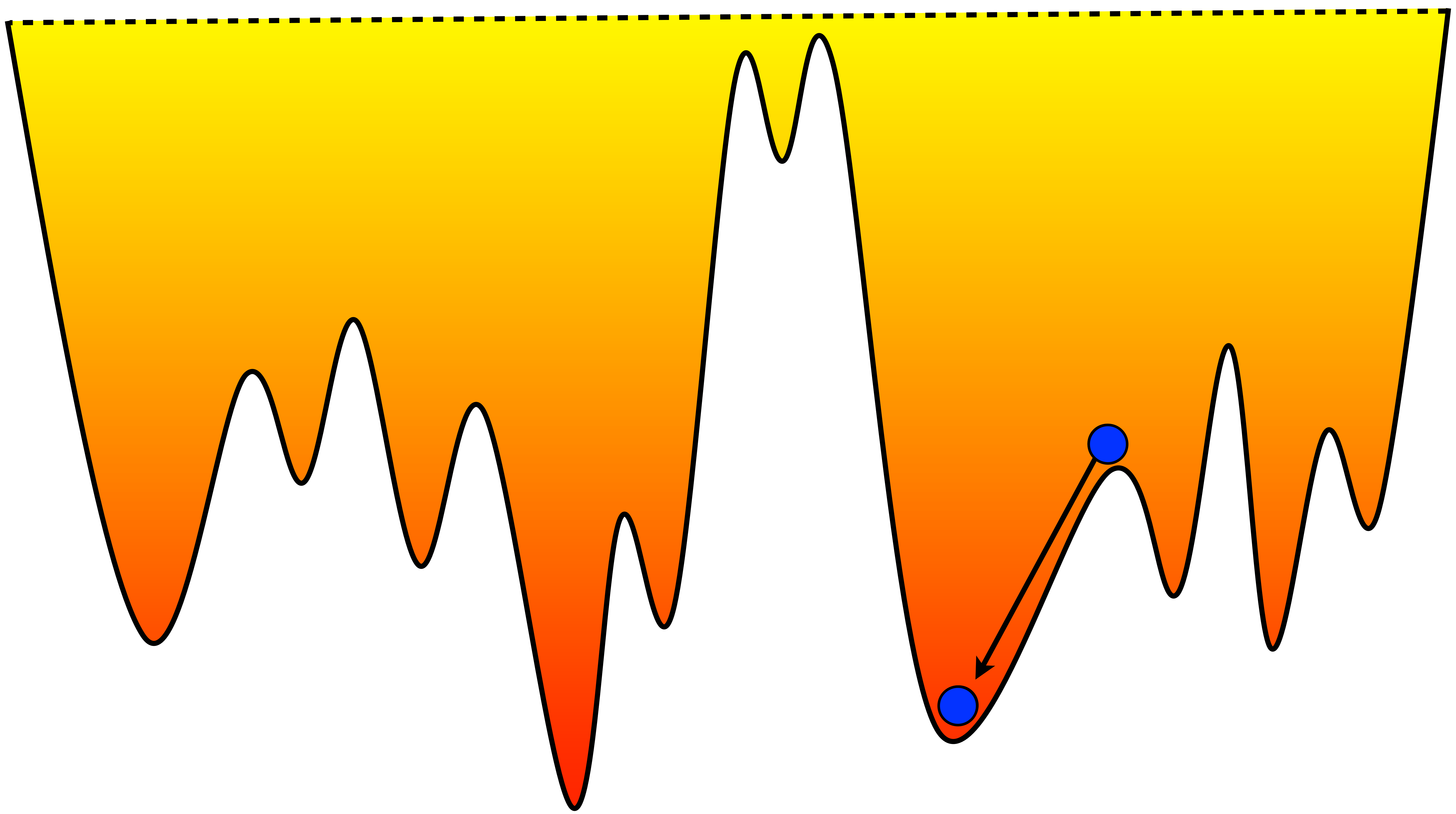
Tabu node ● : node I have already visited

Tabu Search



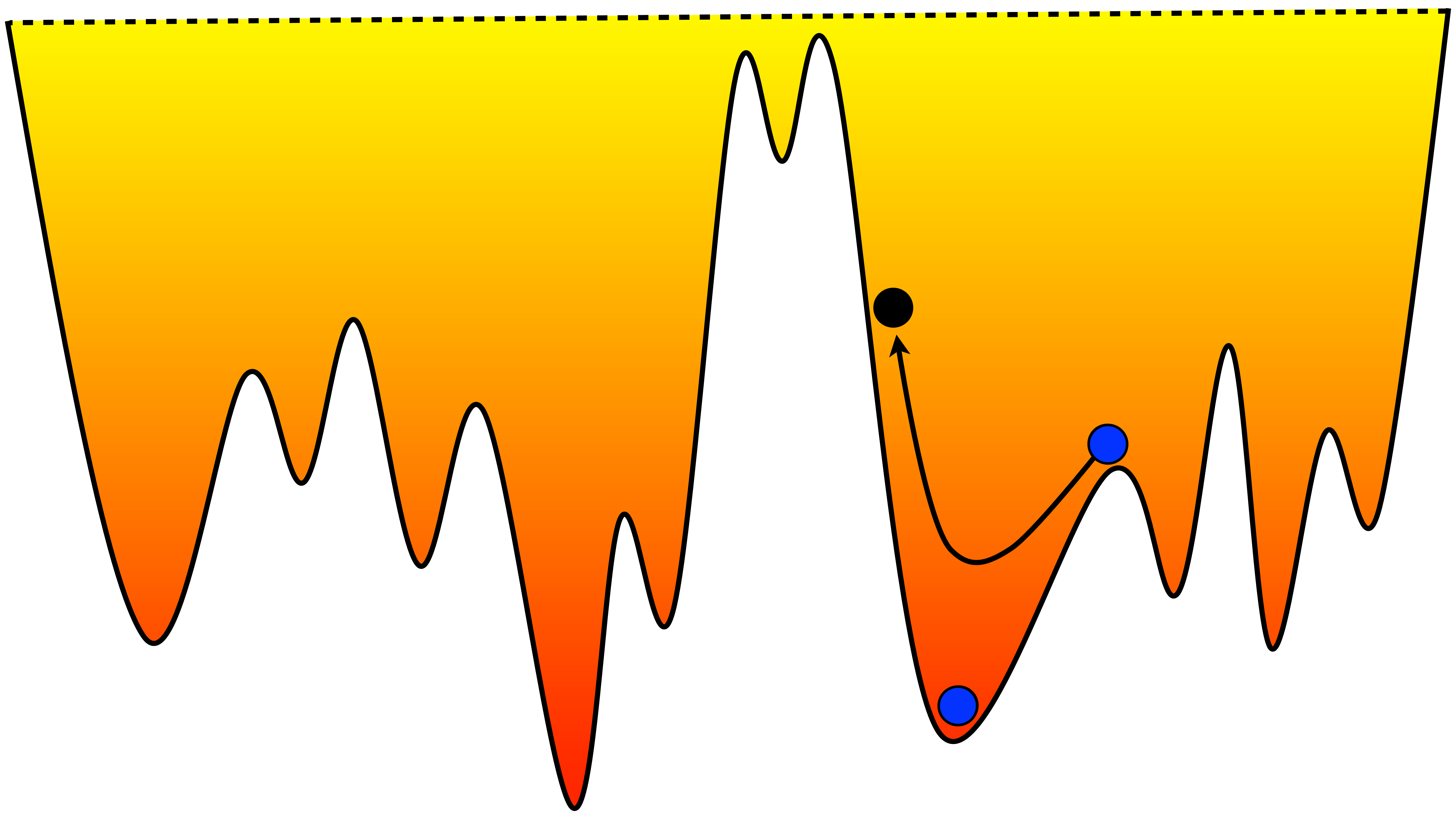
Tabu node ● : node I have already visited

Tabu Search



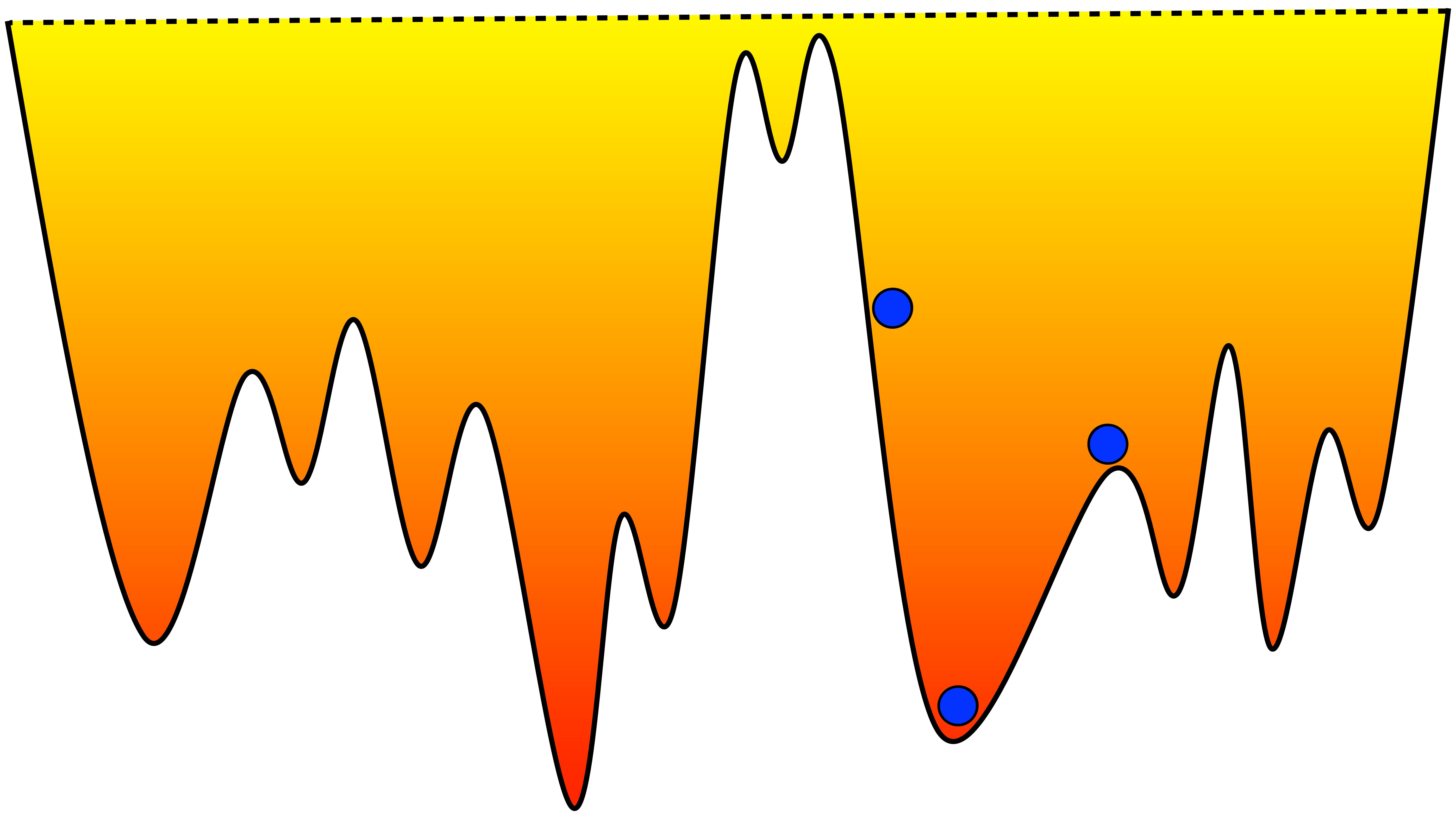
Tabu node ● : node I have already visited

Tabu Search



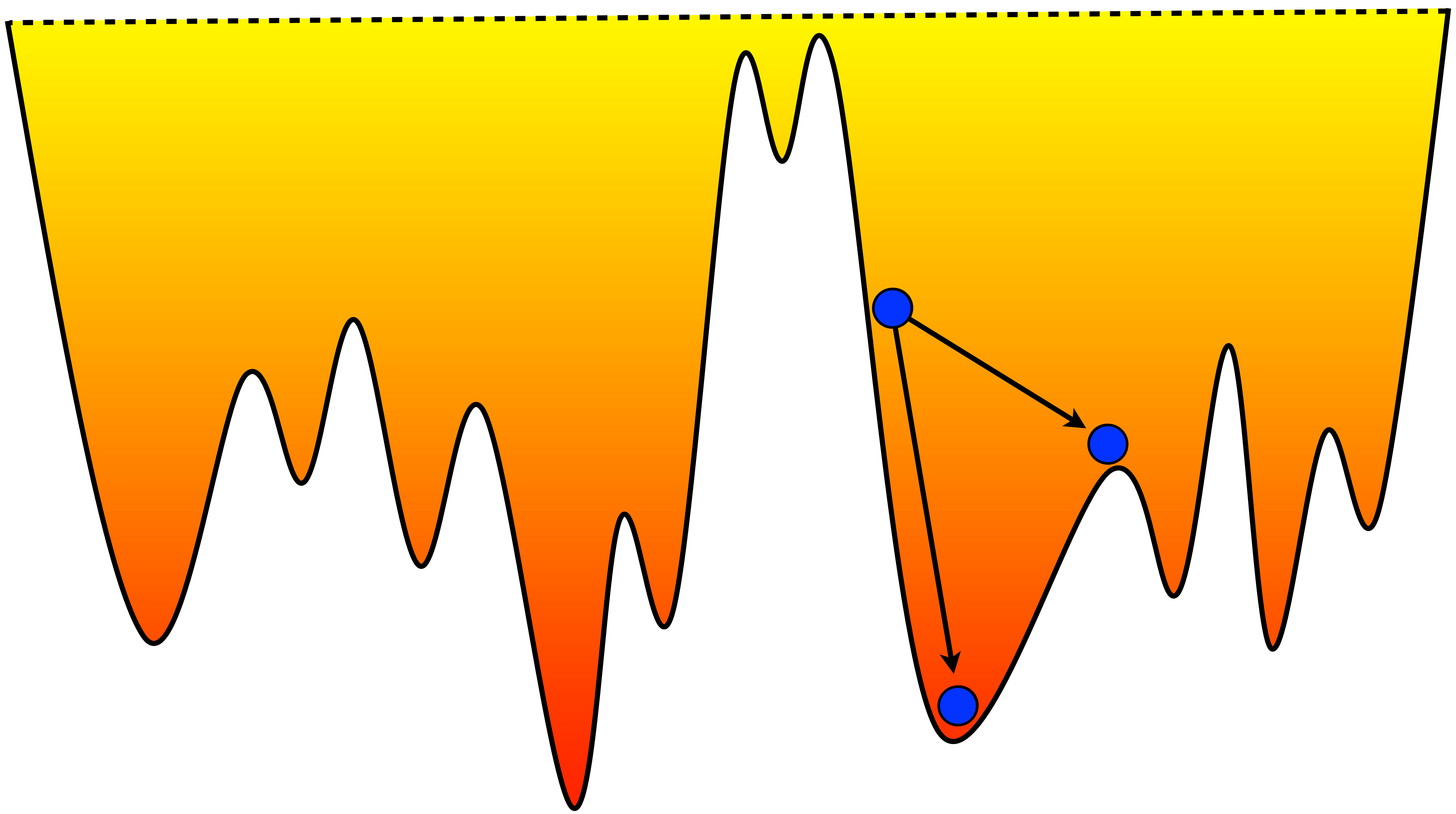
Tabu node ● : node I have already visited

Tabu Search



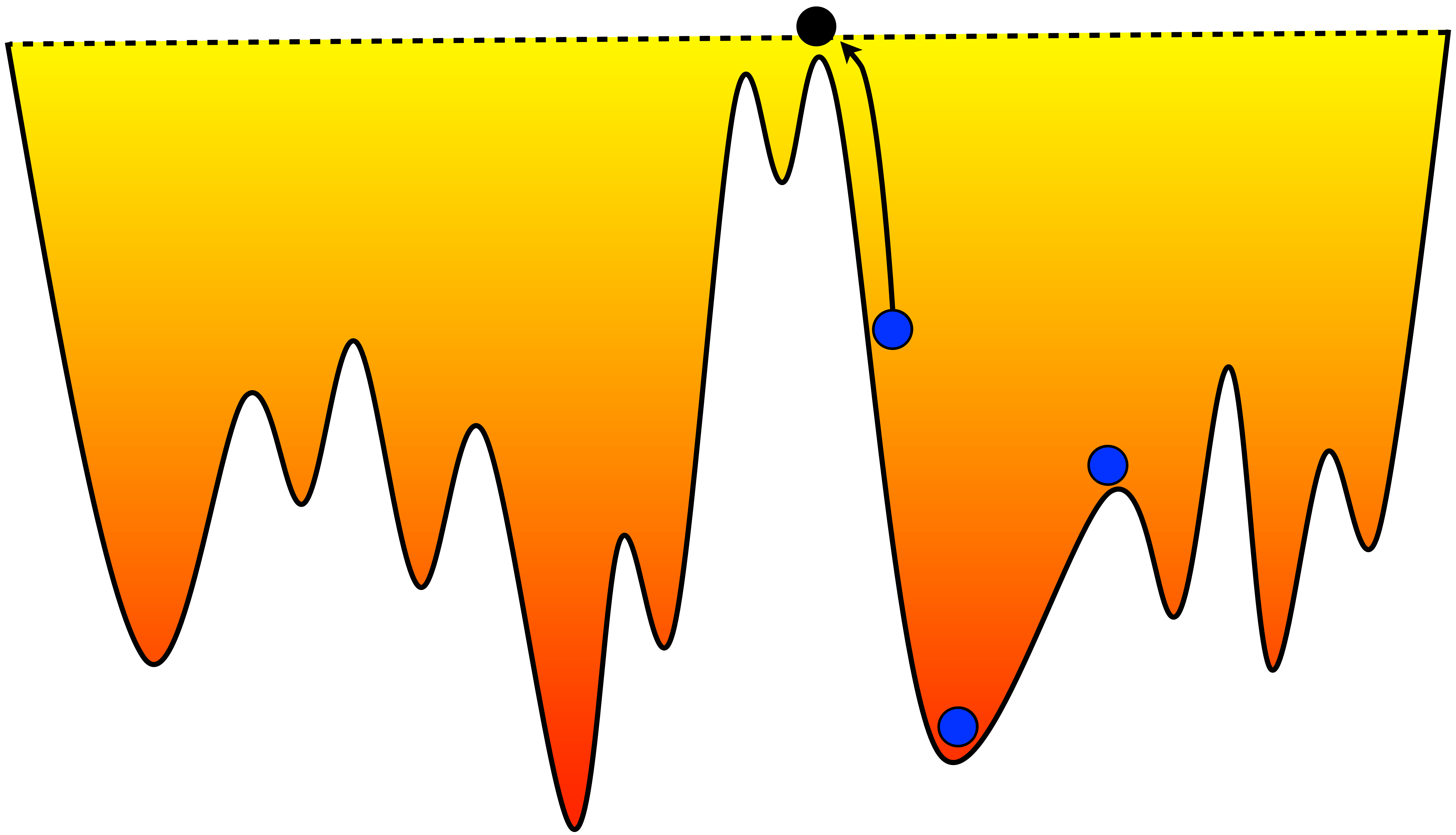
Tabu node ● : node I have already visited

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

Tabu Search

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

$$
\begin{array}{l}
\textbf{function}\ \textsc{LocalSearch}(f, N, L, S, s_1)\ \{ \\
\quad s^* := s_1; \\
\quad \tau := \langle s \rangle; \\
\quad \textbf{for}\ k := 1\ \textbf{to}\ MaxTrials\ \textbf{do} \\
\quad\quad \textbf{if}\ \mathit{satisfiable}(s) \wedge f(s_k) < f(s^*)\ \textbf{then} \\
\quad\quad\quad s^* := s_k; \\
\quad\quad s_{k+1} := S(L(N(s_k), \tau), \tau); \\
\quad\quad \tau := \tau :: s_{k+1}; \\
\quad \textbf{return}\ s^*; \\
\}
\end{array}
$$

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,τ)
2. **return** $\{ n \in N \mid n \notin \tau \}$;

Short-Term Memory

- Key issue with tabu search
  - expensive to maintain all the visited nodes

Short-Term Memory

- Key issue with tabu search
  - expensive to maintain all the visited nodes
- Short-term memory
  - only keep a small suffix of visited nodes
    - typically called the tabu list
  - may increase or decrease the size of the list dynamically
    - decrease when the selected node degrades the objective
    - increase when the selected node improves the objective

Short-Term Memory

- Still too big?
  - may still be too costly
    - requires to store and compare entire solutions
  - keep an abstraction of the suffix
    - many possibilities

Illustration: Car Sequencing

- Greedy local improvement

```
s := some initial configuration;

while (violations(s) > 0) {
  N := { <i,j> | violations(i) > 0 & i != j };
  <i,j> := argmin(<i,j> in N) f(swap(s,i,j));
  n = swap(s,i,j);
  if f(n) < f(s)
    s := n;
  else
    break;
}
return s;
```

Transition Abstractions

- Key idea
  - store the transitions, not the states

Transition Abstractions

- Key idea
  - store the transitions, not the states
- Consider car-sequencing
  - a move swaps two slots a_i and b_i
  - the tabu list then keep pairs (a_i,b_i) to denote that slots a_i and b_i have been swapped in step i
  - a configuration n in N(s) is tabu if n can be obtained by swapping a pair (a_i,b_i) in the tabu-list

Transition Abstractions

- How to implement this
  - keep an iteration counter it
  - keep a data structure tabu[i,j] which stores the next iteration when pair (i,j) can be swapped
    - an iteration number
    - not legal to swap this pair before
  - assume that the tabu list of size L

Transition Abstractions

- How to implement this
  - keep an iteration counter it
  - keep a data structure tabu[i,j] which stores the next iteration when pair (i,j) can be swapped
    - an iteration number
    - not legal to swap this pair before
  - assume that the tabu list of size L
- When is a move (i,j) tabu?
  - when tabu[i,j] > it

Transition Abstractions

- How to implement this
  - keep an iteration counter it
  - keep a data structure tabu[i,j] which stores the next iteration when pair (i,j) can be swapped
    - an iteration number
    - not legal to swap this pair before
  - assume that the tabu list of size L
- When is a move (i,j) tabu?
  - when tabu[i,j] > it
- What happens when applying a move?
  - tabu[i,j] becomes it + L

Illustration: Car Sequencing

- Tabu search: quadratic neighborhood

```
s := some initial configuration;
s* := s;
it := 0;
for(int i = 1; i <= n; i++)
  for(int j = 1; j <= n; j++)
     tabu[i,j] := 0;
while (violations(s) > 0) {
  N := { <i,j> | violations(i) > 0 & i != j };
  NotTabu := { <i,j> in N | tabu[i,j] <= it};
  if (|NotTabu| > 0)
    <i,j> := argmin(<i,j> in NotTabu) f(swap(s,i,j));
    s = swap(s,i,j);
    tabu[i,j] := it + L;
    tabu[j,i] := it + L;
  if (f(s) < f(s*))
    s* := s;
  it++;
}
return s*;
```

Transition Abstractions

- What happens if all moves are tabu
  - the counter is still incremented and hence at some point, some move will not be tabu any more.

Transition Abstractions

- What happens if all moves are tabu
  - the counter is still incremented and hence at some point, some move will not be tabu any more.
- In practice
  - the implementation does not perform the swaps to find the best ones;
  - the effect of the potential moves is computed incrementally (differentiation)
  - important since tabu-search is (almost always) greedy

Transition Abstractions

- Too strong or too weak?
  - a bit of both

Transition Abstractions

- Too strong or too weak?
  - a bit of both
- Too weak
  - they cannot prevent cycling since they only consider a suffix

Transition Abstractions

- Too strong or too weak?
  - a bit of both
- Too weak
  - they cannot prevent cycling since they only consider a suffix
- Too strong
  - they may prevent us from going to configurations that have not yet been visited.
  - the swaps would produce different configurations but they are forbidden by the tabu list

Transition Abstractions

- Strengthening the abstraction
  - store the transitions and the objective values

Transition Abstractions

- Strengthening the abstraction
  - store the transitions and the objective values
- Consider car-sequencing
  - a move swaps two slots a_i and b_i
  - the tabu list then keep pairs $(a_i,b_i\ f_i,f_{i+1})$ to denote that slots a_i and b_i have been swapped in step i moving from an objective value f_i to f_{i+1}
  - a configuration n in N(s) is tabu if n can be obtained by swapping a pair (a_i,b_i) in the tabu-list and $f(s)= f_i$ and $f(n)=f_{i+1}$

Illustration: Car Sequencing

- Tabu search: linear neighborhood

```
s := some initial configuration;
s* := s;
it := 0;
for(int i = 1; i <= n; i++)
  for(int j = 1; j <= n; j++)
     tabu[i,j] := 0;
while (violations(s) > 0) {
  select i such that violations(i) > 0;
  N := { <i,j> | i != j };
  NotTabu := { <i,j> in N | tabu[i,j] <= it};
  if (|NotTabu| > 0)
   <i,j> := argmin(<i,j> in NotTabu) f(swap(s,i,j));
   s = swap(s,i,j);
   tabu[i,j] := it + L;
 if (f(s) < f(s*))
   s* := s;
  it++;
}
return s*;
```

Queens Problem

Queens Problem

- Move
  - assign the value of a variable

Queens Problem

- Move
  - assign the value of a variable
- What is the transition abstraction?
  - the variable cannot be assigned its old value

Queens Problem

- Move
  - assign the value of a variable
- What is the transition abstraction?
  - the variable cannot be assigned its old value
- The tabu list should be viewed as
  - storing pairs (x,v)

Illustration: The Queens

- Tabu search: linear neighborhood

```
s := some initial configuration;
s* := s;
it := 0;
for(int c = 1; c <= n; c++)
  for(int r = 1; r <= n; r++)
     tabu[c,r] := 0;
while (violations(s) > 0) {
  select c such that violations(queens[c]) > 0;
  N := { <c,r> | queens[c] != r};
  NotTabu := { <c,r> in N | tabu[c,r] <= it};
  if (|NotTabu| > 0)
    <c,r> := argmin(<c,r> in NotTabu) f(s[queens[c]:=r]);
    tabu[i,queens[c]] := it + L;
    s = s[queens[c]:=r];
 if (f(s) < f(s*))
    s* := s;
  it++;
}
return s*;
```

Queens Problem

Queens Problem

- Transition abstraction may be even coarser
  - more diversification

Queens Problem

- Transition abstraction may be even coarser
  - more diversification
- Move
  - assign the value of a variable

Queens Problem

- Transition abstraction may be even coarser
  - more diversification
- Move
  - assign the value of a variable
- What is the transition abstraction?
  - make the variable tabu!
  - no move with this variable for a number of iterations

Aspiration

- What if the move is tabu but really good?
  - e.g., $f(n) < f(s^*)$
  - this is possible since the tabu list is too strong

Aspiration

- What if the move is tabu but really good?
  - e.g., f(n) < f(s*)
  - this is possible since the tabu list is too strong
- Aspiration criterion
  - override the tabu status
  -
```
NotTabu := { <i,j> in N | tabu[i,j] <= it or
                          f(swap(s,i,j)) < f(s*)};
```

Long-Term Memory

- Intensification
  - store high-quality solutions and return to them periodically

Long-Term Memory

- Intensification
  - store high-quality solutions and return to them periodically
- Diversification
  - when the search is not producing improvement, diversify the current state
    - e.g., randomly change the values of some variables

Long-Term Memory

- Intensification
  - store high-quality solutions and return to them periodically
- Diversification
  - when the search is not producing improvement, diversify the current state
    - e.g., randomly change the values of some variables
- Strategic oscillation
  - change the percentage of time spent in the feasible and infeasible regions

Final Remark

- Techniques such as
  - diversification
  - intensification
  - strategic oscillation

Final Remark

- Techniques such as
  - diversification
  - intensification
  - strategic oscillation

have also been used in simulated annealing

Until Next Time