Local Search and Optimization

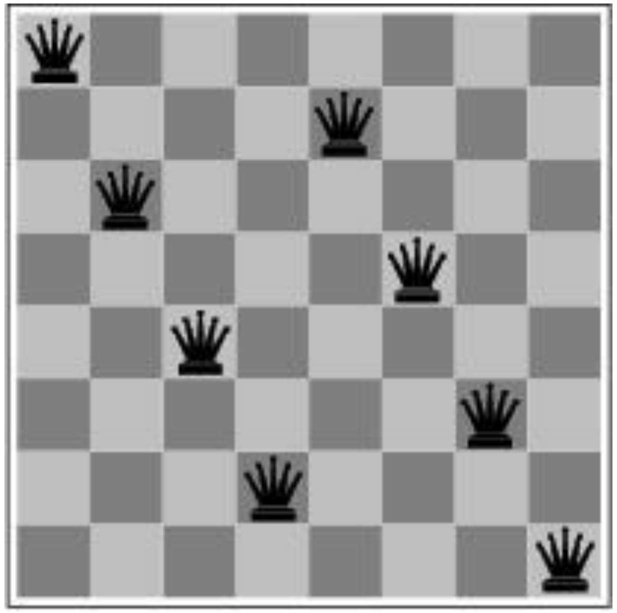
Chapter 4

Outline

- Local search techniques and optimization
  - Hill-climbing
  - Gradient methods
  - Simulated annealing
  - Genetic algorithms
  - Issues with local search

Local search and optimization

- This lecture: a state is solution to problem
  - for some problems path is irrelevant.
  - E.g., 8-queens
- Different algorithms can be used
  - Depth First Branch and Bound
  - Local search



Goal Satisfaction

reach the goal node
Constraint satisfaction

Optimization

optimize(objective fn)
Constraint Optimization

You can go back and forth between the two problems
Typically in the same complexity class

Local search and optimization

- Local search
  - Keep track of single current state
  - Move only to neighboring states
  - Ignore paths

- Advantages:
  - Use very little memory
  - Can often find reasonable solutions in large or infinite (continuous) state spaces.

- "Pure optimization" problems
  - All states have an objective function
  - Goal is to find state with max (or min) objective value
  - Local search can do quite well on these problems.

Trivial Algorithms

- Random Sampling
  - Generate a state randomly
- Random Walk
  - Randomly pick a neighbor of the current state

Hill-climbing (Greedy Local Search) max version

```
function HILL-CLIMBING( problem) return a state that is a local maximum
    input: problem, a problem
    local variables: current, a node.
                     neighbor, a node.

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest valued successor of current
        if VALUE [neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

min version will reverse inequalities and look for lowest valued successor

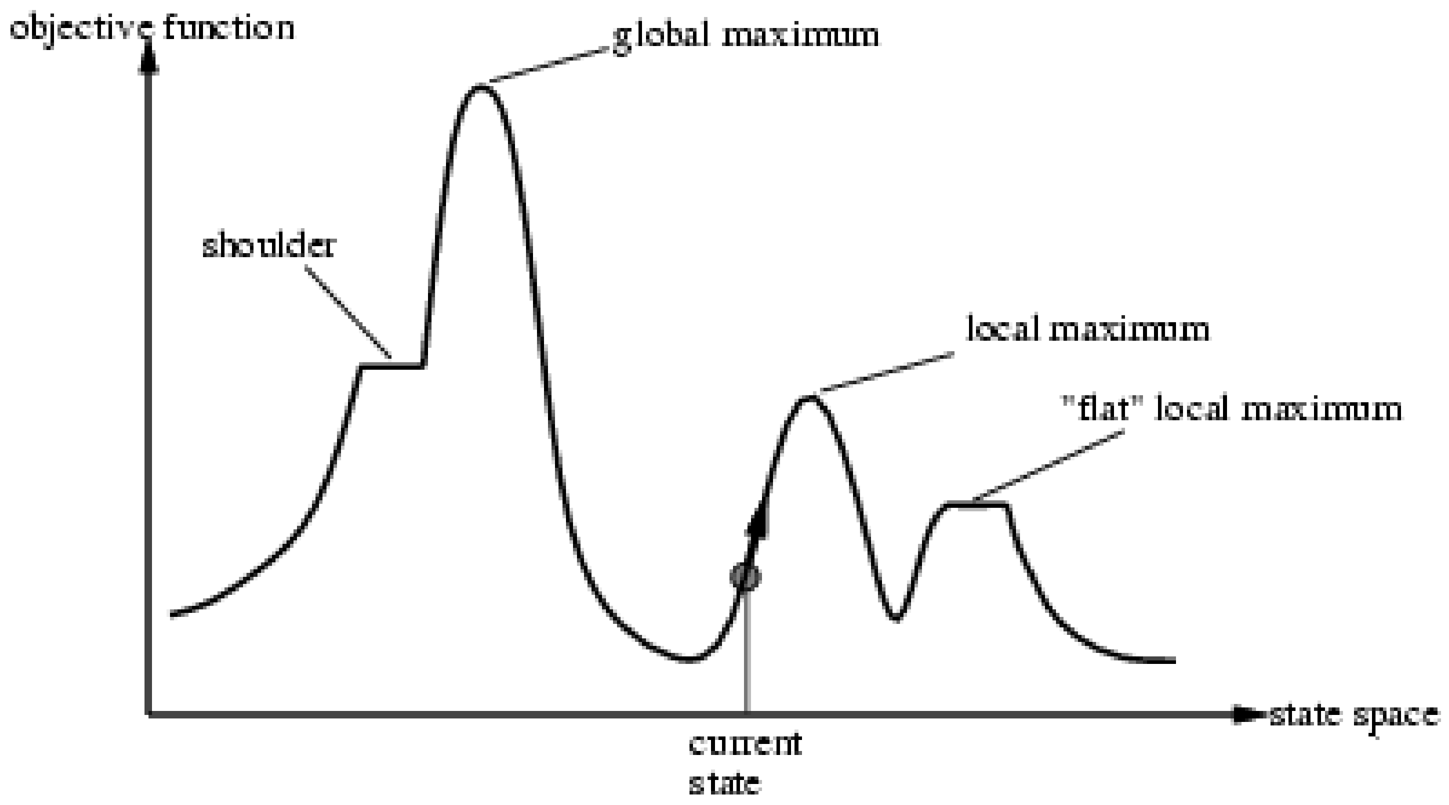
Hill-climbing search

- “a loop that continuously moves towards increasing value”
  - terminates when a peak is reached
- Value can be either
  - Objective function value
  - Heuristic function value (minimized)

- Hill climbing does not look ahead of the immediate neighbors
- Can randomly choose among the set of best successors
  - if multiple have the best value

- “climbing Mount Everest in a thick fog with amnesia”

“Landscape” of search

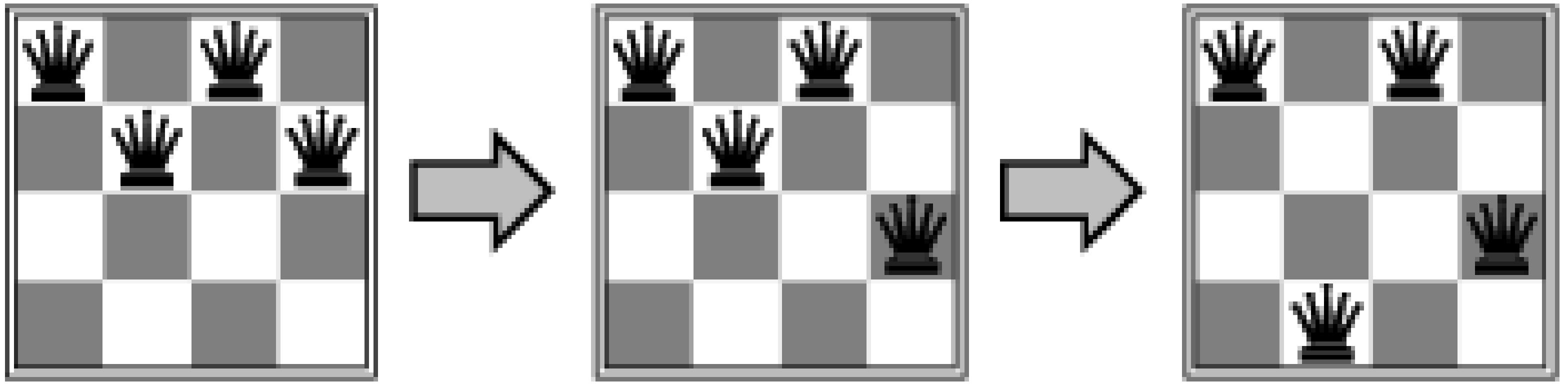


Hill Climbing gets stuck in local minima

depending on?

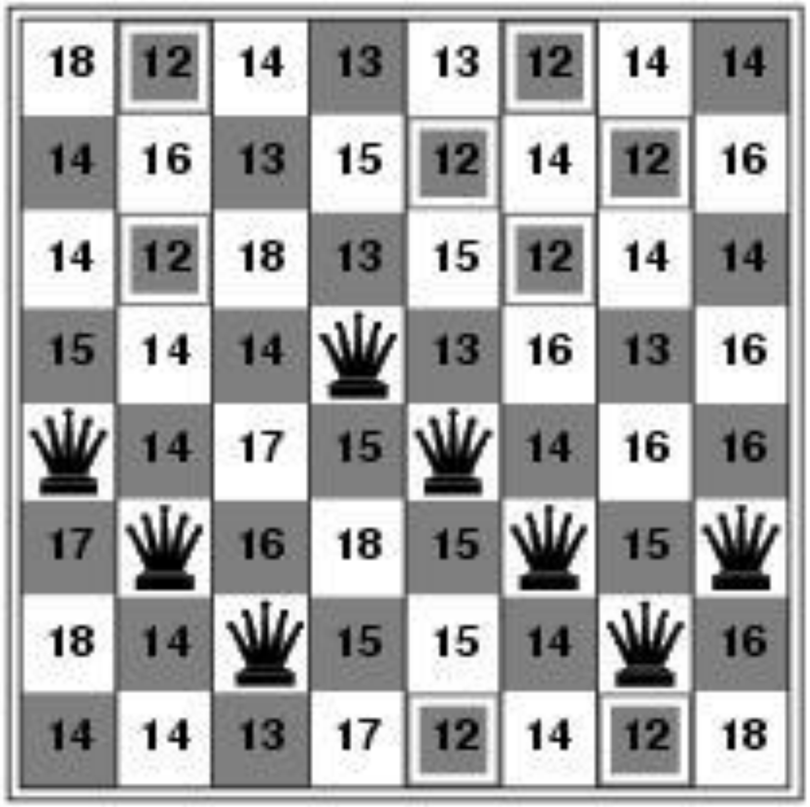
Example: *n*-queens

- Put *n* queens on an *n* x *n* board with no two queens on the same row, column, or diagonal



- Is it a satisfaction problem or optimization?

Hill-climbing search: 8-queens problem

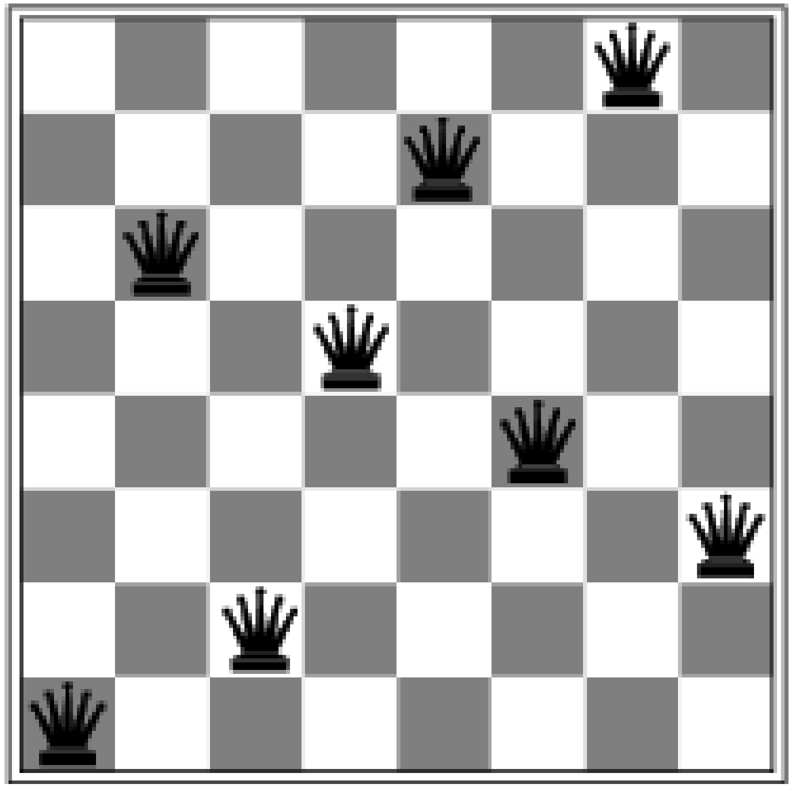


- Need to convert to an optimization problem
- *h* = number of pairs of queens that are attacking each other
- *h = 17* for the above state

Search Space

- State
  - All 8 queens on the board in some configuration
- Successor function
  - move a single queen to another square in the same column.
- Example of a heuristic function *h(n)*:
  - the number of pairs of queens that are attacking each other
  - (so we want to minimize this)

Hill-climbing search: 8-queens problem



- Is this a solution?
- What is h?

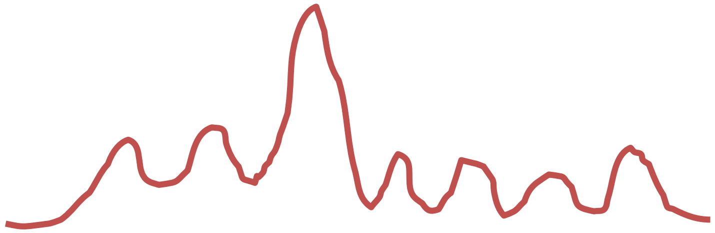
Hill-climbing on 8-queens

- Randomly generated 8-queens starting states…
- 14% the time it solves the problem
- 86% of the time it get stuck at a local minimum

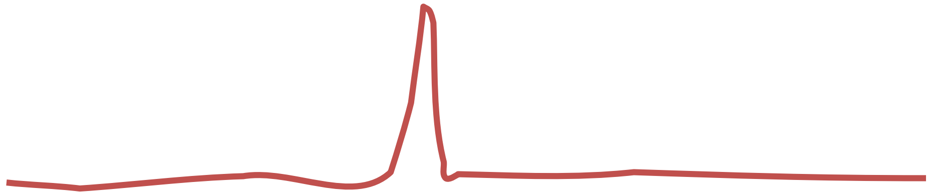
- However…
  - Takes only 4 steps on average when it succeeds
  - And 3 on average when it gets stuck
  - (for a state space with 8^8 =~17 million states)

Hill Climbing Drawbacks

- Local maxima



- Plateaus



Systematic heuristics (1)

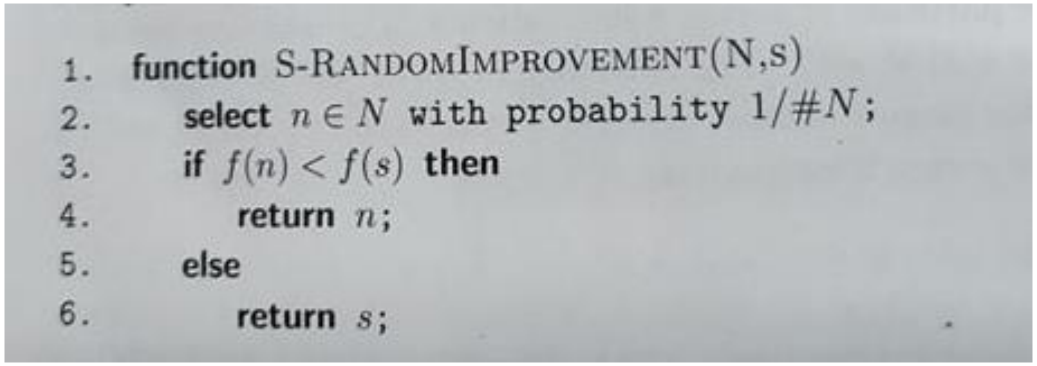
- Best neighbor
  - Choosing the neighbor with the best evaluation
  - Scan al next states (neighors)
- First neighbor
  - Select the first-found neighbor which is better than current solution
  - Might increase the number of interation to reach local minima

Systematic heuristics (2)

- Mutistage heuristics
  - Balance between the greedy aspect and the computation time
  - Choice of a neighbor can be considered as several decision that are performed together in the best-improvement heuristics
  - Example: Choose a pair of queens to swap to each other in N-queen problem, in two-stage heuristic: it first select the queen Q1 with largest violation, then chooses the best queen Q2 to swap with Q1.

Random walks (1)

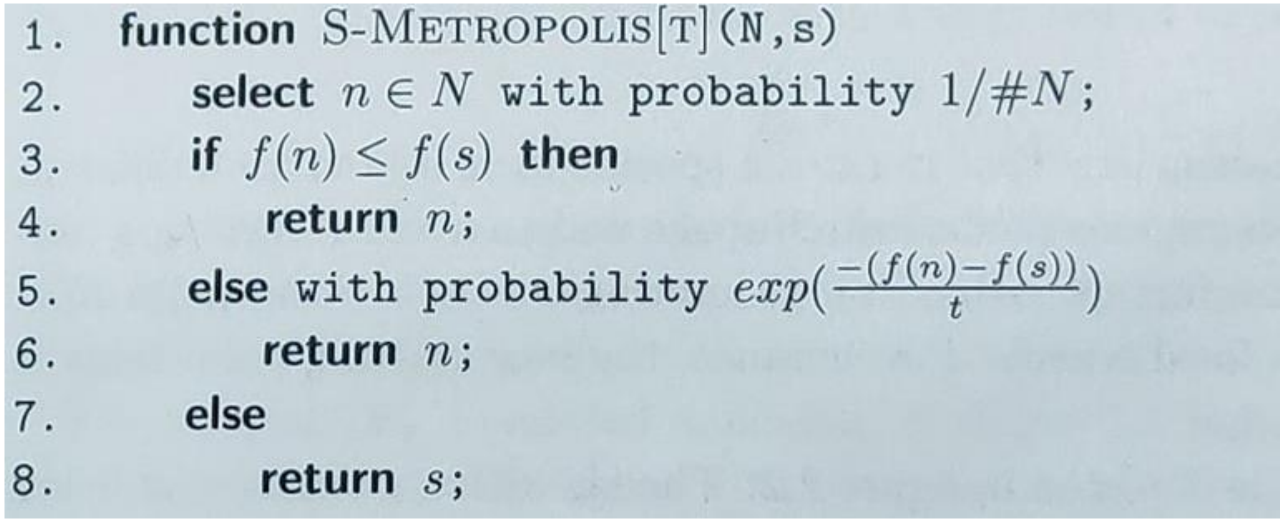
- Random-walk heuristics are quit different to systematic heuristics: They select a neighbor randomly and decide whether to accept it as the next solution
- Random improvement
  - Simplest example of random-walk heuristics
  - Accept a neighbor if it improves the current solution



```
function S-RandomImprovement(N,s)
    select $n \in N$ with probability $1/\#N$;
    if $f(n) < f(s)$ then
        return $n$;
    else
        return $s$;
```

Random walks (2)

- The Metropolis heuristics
  - Extension of random improvement
  - Moves degrading the objective value is allowed with a small probability



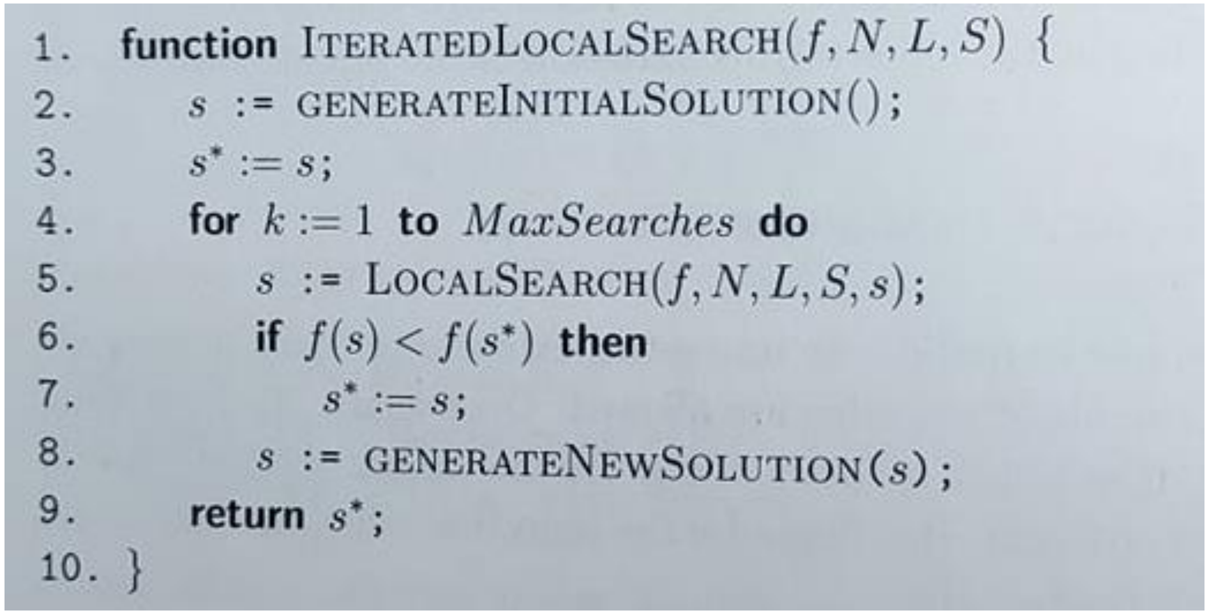
```
function S-METROPOLIS[T](N,s)
    select $n \in N$ with probability $1/\#N$;
    if $f(n) \leq f(s)$ then
        return $n$;
    else with probability $exp(\frac{-(f(n)-f(s))}{t})$
        return $n$;
    else
        return $s$;
```

Metaheuristics

- Goal of heuristics is to reach high-quality local minima quickly.
- Metaheuristics aim at escaping these ocal minima and a directing the search toward global optimality.
  - Iterated Local Search
  - Simulated Annealing
  - Guided Local Search
  - Variable Neighborhood Search
  - Tabu Search

Iterated Local Search

- It is a ubiquitous metaheuristics that iterates a specific local search from different starting points in order to sample various regions of the search space and to avoid returning a low-quality local minimum.



```
function IteratedLocalSearch($f, N, L, S$) {
    $s$ := GenerateInitialSolution();
    $s^* := s$;
    for $k := 1$ to $MaxSearches$ do
        $s$ := LocalSearch($f, N, L, S, s$);
        if $f(s) < f(s^*)$ then
            $s^* := s$;
        $s$ := GenerateNewSolution($s$);
    return $s^*$;
}
```

Simulated Annealing (1)

- Simulated annealing is a popular metaheuristic based on the Metropolis heuristic.
- Metropolis heuristic accepts a degrading move with probability

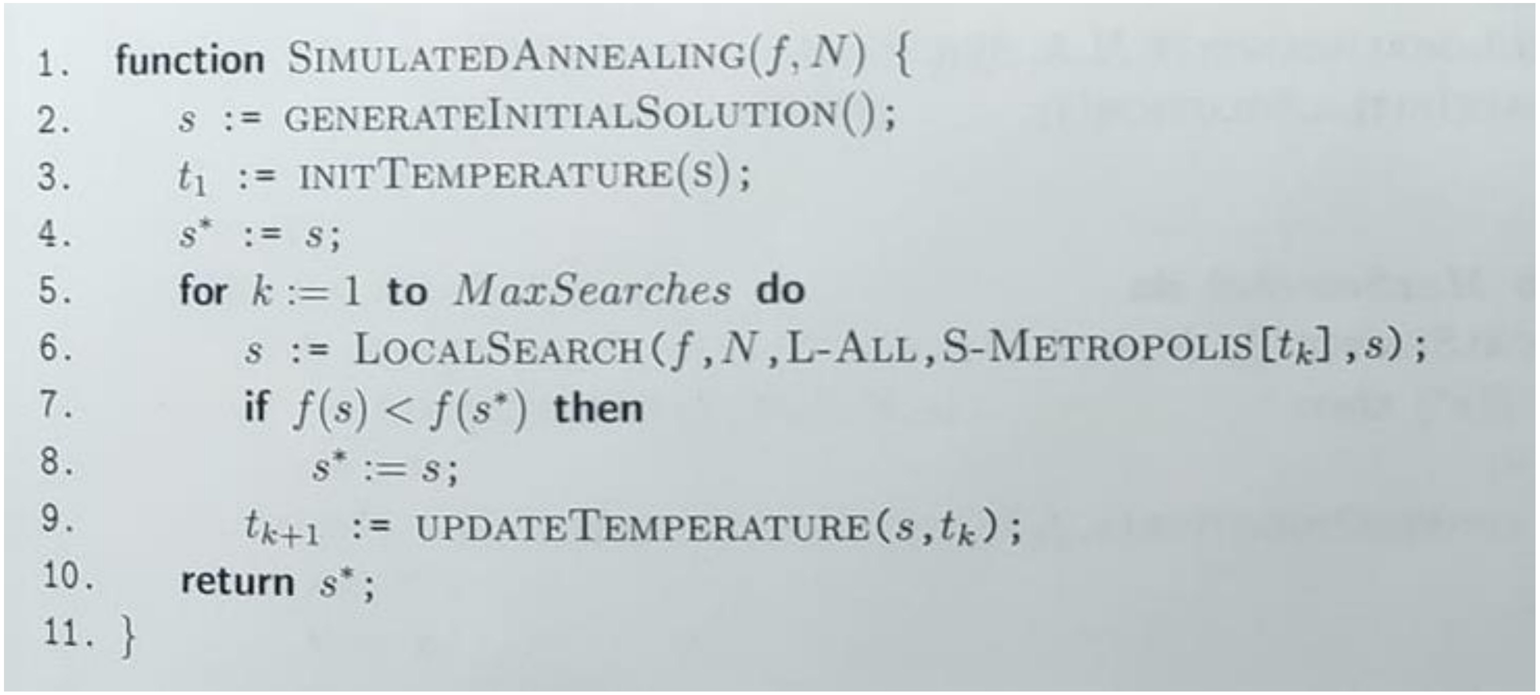
$$\exp(\frac{-(f(n) - f(s))}{t})$$

where t is a parameter of the heuristic. Different values of t produces different trade-offs between the quality of the solutions and the execution time.

Simulated Annealing (2)

- The key idea underlying simulated annealing is to iterate the Metropolis algorithms with a sequence of decreasing temperatures
$$t_0, t_1, \ldots, t_i, \ldots (t_{k+1} \leq t_k)$$
- The goal is to accept many moves initially in order to sample the search space widely (large values of t_k) vaf to move progressively toward small values of t_k, thus coverging toward random improvement and we hope a high-quality local minimum when $t_i \to 0$.

Simulated Annealing (3)

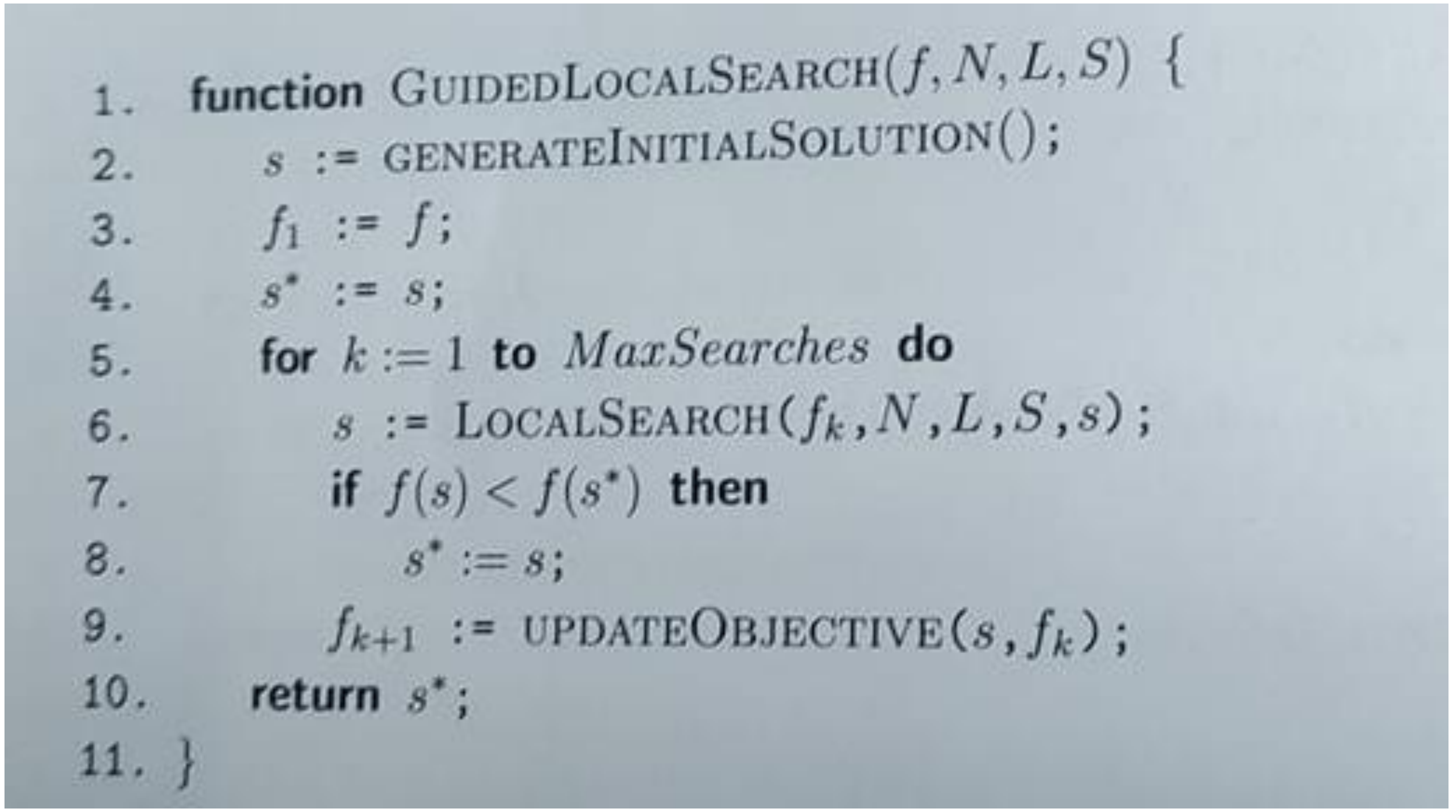


```
function SimulatedAnnealing($f, N$) {
    $s$ := generateInitialSolution();
    $t_1$ := initTemperature(s);
    $s^*$ := $s$;
    for $k := 1$ to $MaxSearches$ do
        $s$ := LocalSearch($f$, $N$, L-All, S-Metropolis[$t_k$], $s$);
        if $f(s) < f(s^*)$ then
            $s^* := s$;
        $t_{k+1}$ := updateTemperature($s$, $t_k$);
    return $s^*$;
}
```

Guided Local Search (1)

- Guided Local Search is based on the recognition that a local optima s for the objective function f may not be locally optimal with respect to another function f'. Using f' instead of f would thus drive the search away from s.
- The key idead behind guided local search is to use a sequence of the objective functions $f_0, f_1, \ldots, f_i$ to drive the search away from local optima and to explore the search space more extensively.

Guided Local Search (2)

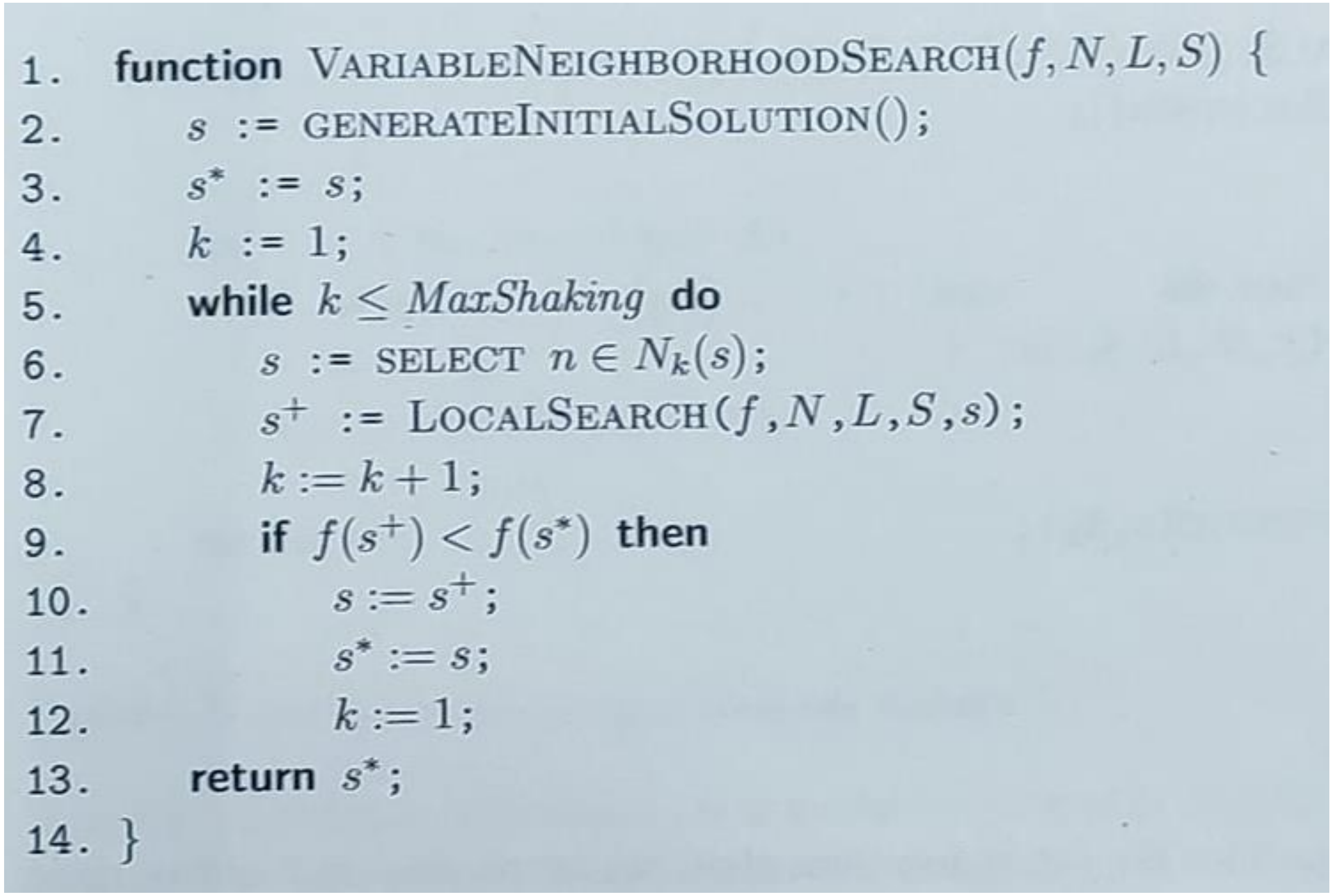


```
function GUIDEDLOCALSEARCH($f, N, L, S$) {
    $s$ := GENERATEINITIALSOLUTION();
    $f_1$ := $f$;
    $s^*$ := $s$;
    for $k := 1$ to $MaxSearches$ do
        $s$ := LOCALSEARCH($f_k, N, L, S, s$);
        if $f(s) < f(s^*)$ then
            $s^* := s$;
        $f_{k+1}$ := UPDATEOBJECTIVE($s, f_k$);
    return $s^*$;
}
```

Variable Neighborhood Search (1)

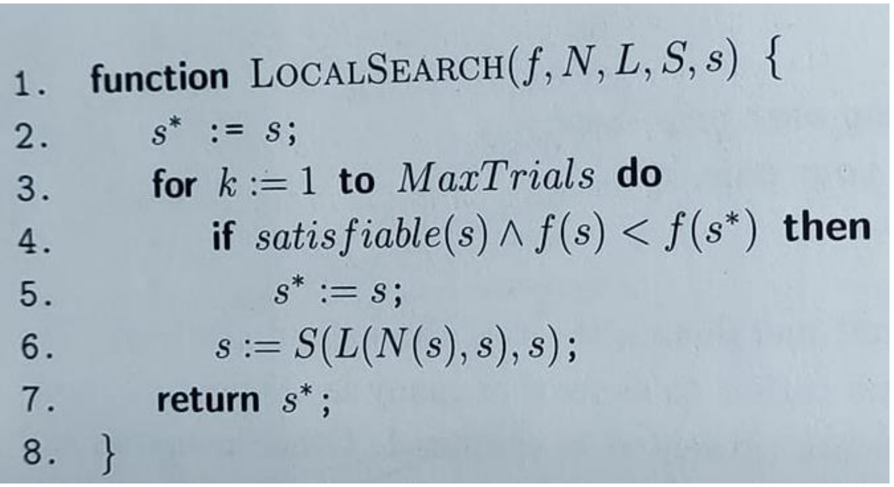
- It works with a collection of neighborhoods $N_1, N_2, \dots, N_i$ to diversify the current solution.
- The intuition is that the neighborhoods $N_1, N_2, \dots, N_i$ are increasing in size, providing more opportunies for significant diversifications over time.

Variable Neighborhood Search (2)



```
function VariableNeighborhoodSearch($f, N, L, S$) {
    $s$ := GenerateInitialSolution();
    $s^*$ := $s$;
    $k$ := 1;
    while $k \leq MaxShaking$ do
        $s$ := SELECT $n \in N_k(s)$;
        $s^+$ := LocalSearch($f, N, L, S, s$);
        $k := k + 1$;
        if $f(s^+) < f(s^*)$ then
            $s := s^+$;
            $s^* := s$;
            $k := 1$;
    return $s^*$;
}
```

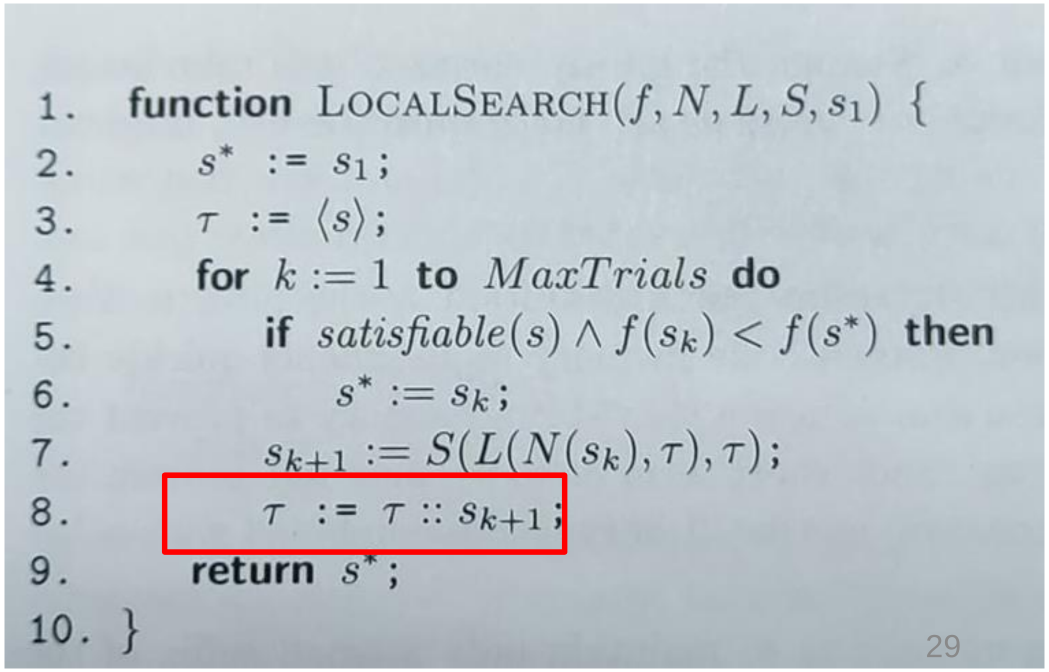
Tabu Search (1)



```
function LOCALSEARCH($f, N, L, S, s$) {
    $s^*$ := $s$;
    for $k := 1$ to $MaxTrials$ do
        if $satisfiable(s) \wedge f(s) < f(s^*)$ then
            $s^* := s$;
        $s := S(L(N(s), s), s)$;
    return $s^*$;
}
```

$$\tau = \langle s_0, s_1, \dots, s_k \rangle$$

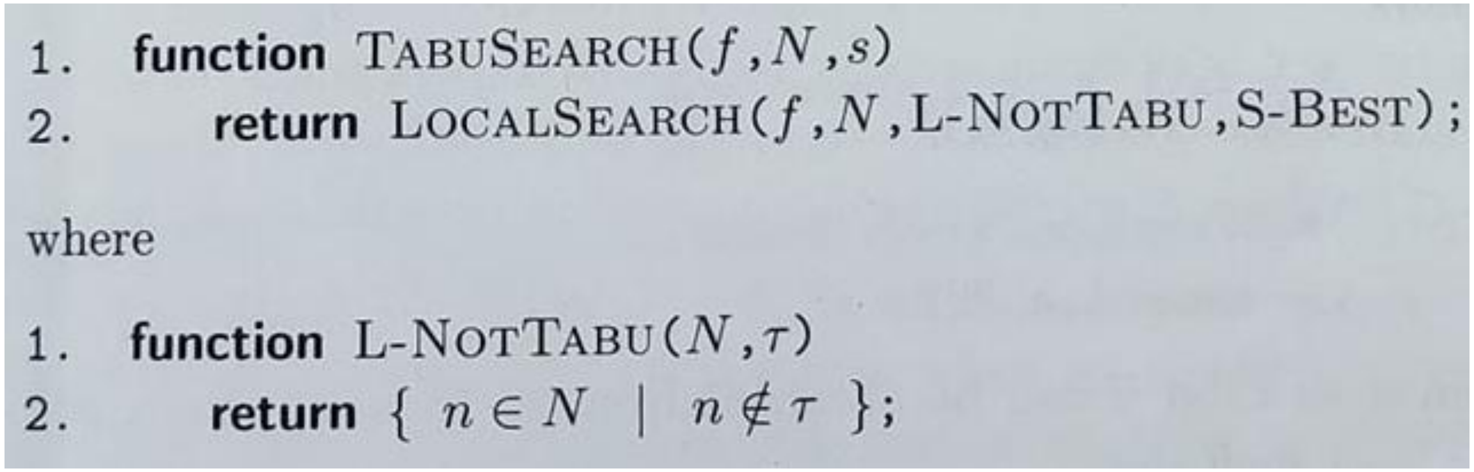
Tabu search: Given a sequence $\langle s_0, s_1, \dots, s_k \rangle$, select s_{k+1} to be the best neighbor in $N(s_k)$ that has not yet been visited.



```
function LOCALSEARCH($f, N, L, S, s_1$) {
    $s^*$ := $s_1$;
    $\tau$ := $\langle s \rangle$;
    for $k := 1$ to $MaxTrials$ do
        if $satisfiable(s) \wedge f(s_k) < f(s^*)$ then
            $s^* := s_k$;
        $s_{k+1} := S(L(N(s_k), \tau), \tau)$;
        $\tau$ := $\tau :: s_{k+1}$;
    return $s^*$;
}
```

Tabu Search (2)

- Tabu search can be viewed as the combination of a greedy strategy with a definition of legal moves ensuring that a solution is never visited twice.
  - Allow the LS to select moves degrading the quality of the current solution
  - Greedy nature of the Tabu search ensures that the objective function does not degrade too much.



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

where

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

Tabu Search (3)

- Short-Term Memory
  - Difficultiy to kee track of all visited solutions: computation time, memory
  - Short-term memory may not prevent the search from revisiting solution entirely, so that it is typically combined with other techniques
  - Transition abstraction that stores the transitions, not the states, since moves only involve a few variables in general.

Tabu Search (4)

- Aspiration:
  - Since Tabu search stores sequence of abstractions and not sequences of solutions, it may forbid transitions $s_1 \rightarrow s_2$, in which s_2 has not been visited before.
  - Some of these transition may be quite desirable, for instance when s_2 would be the best solution found so far $(f(s_2) \leq f(s^*))$
  - Aspiration feature can help the search overcome this limitation when the tabu status may be overridden. The simplest and most widely used aspiration criterion overrides the tabu status of those moves that improve the best solution found so far

Tabu Search (5)

- Long-Term Memory
  - Tabu list abstractions a small suffix of the solution sequence and cannot capture long-term information. As a consequence, it cannot prevent the search from taking long walks whose solutions have low-quality objective values or spending too much time in the same region, leaving otherparts of the search space unexplored.
  - Many tabu search maintain a long-term memory structures to intensify and diversify the search.

Exercise

- Problem 1: Propose and implement a Tabu Search or other Metaheuristics Search for N-Queen problem.
- Problem 2: Propose and implement a Tabu Search or other Metaheuristics Search for Travelling Salesman Problem.

Bài tập: Xây dựng thuật toán Tabu search cho bài toán Phân công giảng dạy

Bài toán phân công giảng dạy

- Có *N* lớp 1,2,…, *N* đã được sắp xếp thời khóa biểu, cần được phân cho *M* giáo viên 1,2,…, *M*.
- Mỗi lớp *i* có *T*(*i*) là danh sách giáo viên có thể thực hiện giảng dạy (*i* = 1,…, *N*) và *c*(*i*) là số tín chỉ của môn học của lớp đó.
- Do đã được xếp thời khóa biểu từ trước nên giữa *N* lớp này có tập *Q* các cặp 2 lớp (*i*,*j*) bị xếp trùng giờ (2 lớp này không thể phân cho cùng 1 giáo viên).
- Hãy tìm phương án phân công giảng dạy sao cho tổng số tín chỉ lớn nhất của các lớp phân cho 1 giáo viên là nhỏ nhất.

Bài toán phân công giảng dạy

- Ví dụ

Lớp	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Số tiết	3	3	4	3	4	3	3	3	4	3	3	4	4

Giáo viên	**Danh sách lớp học có thể dạy**
0	0, 2, 3, 4, 8, 10
1	0, 1, 3, 5, 6, 7, 8
2	1, 2, 3, 7, 9, 11, 12

Cặp lớp trùng tiết

0	2
0	4
0	8
1	4
1	10
3	7
3	9
5	11
5	12
6	8
6	12

Bài toán phân công giảng dạy

Lớp	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Số tiết	3	3	4	3	4	3	3	3	4	3	3	4	4

Giáo viên	**Danh sách lớp học có thể dạy**
0	0, 2, 3, 4, 8, 10
1	0, 1, 3, 5, 6, 7, 8
2	1, 2, 3, 7, 9, 11, 12

Cặp lớp trùng tiết

0	2
0	4
0	8
1	4
1	10
3	7
3	9
5	11
5	12
6	8
6	12

Phương án phân công

Giáo viên	**Danh sách môn học được phân công**	**Số tiết**
0	2, 4, 8, 10	15
1	0, 1, 3, 5, 6	15
2	7, 9, 11, 12	14

Tìm kiếm Tabu

- Biến: x[c] là giáo viên dạy lớp c.
- Láng giềng: N(x) là tập hợp tất cả các lời giải láng giếng của lời x bằng cách thay đổi giáo viên của một lớp học nào đó.
- Tìm kiếm:
  - Leo đồi tham lam
  - Tabu
  - Intensificatio
  - Restarting component