8 QUEENS DENGAN LOCAL SEARCH

Kelompok 8

Anggota Kelompok :

Abdurrahman Farimza	5025201125
Kartika Diva Asmara Gita	5025211039
Hana Maheswari	5025211182

LOCAL SEARCH

Algoritma Local search merupakan metode pencarian solusi berdasarkan neighborhood dari solusi awal (Arya, dkk 2009). Algoritma ini mencari solusi disekitar solusi awal untuk memperbaiki solusi berikutnya. Jika ditemukan solusi yang lebih baik maka solusi ini yang akan menggantikan solusi sebelumnya dan pencarian lokal akan diteruskan hingga tidak ada kemungkinan untuk memperbaiki solusi lagi atau telah mencapai local optimum.

HILL CLIMBING

Algoritma hill-climbing adalah salah satu algoritma local search. Algoritma ini memiliki node yang terdiri dari dua bagian: status dan nilai. Ini dimulai dengan keadaan yang tidak optimal dan meningkat hingga prasyarat tertentu terpenuhi. Fungsi heuristik digunakan sebagai dasar untuk prasyarat ini.

```python
import random
import copy
import math
import time

def C2n(n):
    return n * (n-1) / 2


class CheckeredPageState:
    def __init__(self, checkeredPage):
        self.checkeredPage = checkeredPage
        self.dimension = len(self.checkeredPage)
        self.setDic()
        self.setHeuristic()

    def setDic(self):
        dicRows = {}
        dicDiagonal1 = {}
        dicDiagonal2 = {}
        for i in range(self.dimension):
            dicRows[i] = 0
            for j in range(self.dimension):
                dicDiagonal1[i-j] = 0
                dicDiagonal2[i+j] = 0
        for i in range(self.dimension):
            for j in range(self.dimension):
                if self.checkeredPage[i][j]:
                    dicRows[i] += 1
                    dicDiagonal1[i-j] += 1
                    dicDiagonal2[j+i] += 1
        self.dicRows = dicRows
        self.dicDiagonal1 = dicDiagonal1
        self.dicDiagonal2 = dicDiagonal2
```

```python
    def setHeuristic(self):
        h = 0
        for key in self.dicRows:
            if self.dicRows[key] > 1:
                h += C2n(self.dicRows[key])
        for key in self.dicDiagonal1:
            if self.dicDiagonal1[key] > 1:
                h += C2n(self.dicDiagonal1[key])
        for key in self.dicDiagonal2:
            if self.dicDiagonal2[key] > 1:
                h += C2n(self.dicDiagonal2[key])
        self.h = h

    def getRandomSteepestAscent(self):
        neighbors = []
        huristic = float("inf")
        for j in range(self.dimension):
            for i in range(self.dimension):
                if self.checkeredPage[i][j] == 1:
                    ikeep = i
                    break
            for i in range(self.dimension):
                if self.checkeredPage[i][j] == 0:
                    newCheck = copy.deepcopy(self.checkeredPage)
                    newCheck[i][j] = 1
                    newCheck[ikeep][j] = 0
                    neighbor = CheckeredPageState(newCheck)
                    if neighbor.h < huristic:
                        neighbors[:] = []
                        huristic = neighbor.h
                    if neighbor.h == huristic:
                        neighbors.append(neighbor)
        return(random.choice(neighbors))
```

```
70      def getFirstChoice(self):
71          test = [[False for i in range(self.dimension)] for j in range(self.dimension)]
72          while 1:
73              i = random.randrange(0, self.dimension)
74              j = random.randrange(0, self.dimension)
75              test[i][j] = True
76              newCheck = copy.deepcopy(self.checkeredPage)
77              newCheck[i][j] = 1
78              for k in range(self.dimension):
79                  if self.checkeredPage[k][j]:
80                      ikeep = k
81                      break
82              newCheck[ikeep][j] = 0
83              newCheck[i][j] = 1
84              neighbor = CheckeredPageState(newCheck)
85              if neighbor.h < self.h:
86                  return neighbor
87              flag = True
88              for x in test:
89                  for y in x:
90                      if y is False:
91                          flag = False
92                          break
93                  if flag is False:
94                      break
95              if flag is True:
96                  return None
97
98      def printPage(self):
99          for xs in self.checkeredPage:
100             print(" ".join(map(str, xs)))
```

```
102     def getMove(self, neighbor):
103         test = False
104         for j in range(self.dimension):
105             for i in range(self.dimension):
106                 if self.checkeredPage[i][j] != neighbor.checkeredPage[i][j]:
107                     if self.checkeredPage[i][j] == 1:
108                         istart = i
109                     else:
110                         iend = i
111                     if test is False:
112                         test = True
113                     else:
114                         print("move in column "+ str(j+1) + " from row " + str(istart+1) + " to " + str(iend+1))
115                         break
116 
117     def randomSuccessor(self):
118         j = random.randrange(0, self.dimension)
119         while 1:
120             i = random.randrange(0, self.dimension)
121             if self.checkeredPage[i][j] != 1:
122                 break
123         for k in range(self.dimension):
124             if self.checkeredPage[k][j]:
125                 break
126         newCheckeredPage = copy.deepcopy(self.checkeredPage)
127         newCheckeredPage[i][j] = 1
128         newCheckeredPage[k][j] = 0
129         return CheckeredPageState(newCheckeredPage)
```

```python
def HillClimbingSteepestAscent(checkeredPageInitial):
    current = CheckeredPageState(checkeredPageInitial)
    print("start of hill climbing algorithm steepest ascent")
    while 1:
        print("current state checkered page:")
        current.printPage()
        print("current state h:", current.h)
        neighbor = current.getRandomSteepestAscent()
        if neighbor.h >= current.h:
            if current.h == 0:
                print("the hill climbing algorithm steepest ascent variant found a solution")
                return True, current
            else:
                print("the hill climbing algorithm steepest ascent variant got stuck in local minimum")
                return False, current
        current.getMove(neighbor)
        current = neighbor

def HillClimbingFirstChoice(checkeredPageInitial):
    current = CheckeredPageState(checkeredPageInitial)
    print("start of hill climbing algorithm first choice variant")
    while 1:
        print("current state checkered page:")
        current.printPage()
        print("current state h:", current.h)
        neighbor = current.getFirstChoice()
        if neighbor is None:
            if current.h == 0:
                print("the hill climbing algorithm first choice variant found a solution")
                return True, current
            else:
                print("the hill climbing algorithm first choice variant got stuck in local minimum")
                return False, current
        current.getMove(neighbor)
        current = neighbor
```

```
169  def getRandomCheckeredPage(dimension):
170      checkeredPage = [[0 for i in range(dimension)] for j in range(dimension)]
171      randNumbers = random.sample(range(0, dimension), dimension)
172      for j in range(dimension):
173          checkeredPage[randNumbers[j]][j] = 1
174      return checkeredPage
175
176  def HillClimbingRandomRestart(dimension):
177      print("start of hill climbing algorithm with random restart")
178      while 1:
179          print("-----------------------------------")
180          print("new start of hill climbing algorithm with random restart")
181          checkeredPage = getRandomCheckeredPage(dimension)
182          boolean, state = HillCLimbingSteepestAscent(checkeredPage)
183          if boolean:
184              print("the hill climbing algorithm with random restart ended")
185              return state
```

```
187  def SimulatedAnnealing(checkeredPageInitial, T=4000, tChange=0.8):
188      current = CheckeredPageState(checkeredPageInitial)
189      print("start of simulated annealing algorithm")
190      while 1:
191          print("current state checkered page:")
192          current.printPage()
193          print("current state h:", current.h)
194          T *= tChange
195          if T < 1:
196              print("final state checkered page:")
197              current.printPage()
198              print("final state h:", current.h)
199              if current.h == 0:
200                  print("the simulated annealing found a solution")
201                  return True, current
202              else:
203                  print("the simulated annealing could not find the solution")
204                  return False, current
205          next = current.randomSuccessor()
206          deltaE = current.h - next.h
207          if deltaE > 0:
208              current.getMove(next)
209              current = next
210          else:
211              rand = random.uniform(0, 1)
212              probability = math.exp(deltaE / T)
213              if rand <= probability:
214                  current.getMove(next)
215                  current = next
```

```python
for i in range(1):
    print("------------------")
    randomCheck = getRandomCheckeredPage(8)
    print("new random check generated")
    startHillFirst = time.time()
    HillCLimbingFirstChoice(randomCheck)
    endHillFirst = time.time()
    print("------------------")
    HillCLimbingSteepestAscent(randomCheck)
    endHillSteep = time.time()
    print("------------------")
    HillClimbingRandomRestart(8)
    endHillRandom = time.time()
    print("------------------")
    SimulatedAnnealing(randomCheck)
    endSim = time.time()
    print("run time of hill climbing first choice", endHillFirst - startHillFirst)
    print("run time of hill climbing steepest ascent", endHillSteep - endHillFirst)
    print("run time of hill climbing random restart", endHillRandom - endHillSteep)
    print("run time of simulated annealing", endSim - endHillSteep)
```

```
new random check generated
start of hill climbing algorithm first choice variant
current state checkered page:
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
current state h: 9.0
move in column 5 from row 7 to 3
current state checkered page:
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 1 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
current state h: 8.0
move in column 7 from row 3 to 8
current state checkered page:
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 0
current state h: 6.0
```

```
move in column 3 from row 5 to 8
current state checkered page:
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0
current state h: 5.0
move in column 7 from row 8 to 5
current state checkered page:
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0
current state h: 4.0
move in column 6 from row 8 to 7
current state checkered page:
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
current state h: 2.0
the hill climbing algorithm first choice variant got stuck in local minimum
-----------------
```

```
-----------------
start of hill climbing algorithm steepest ascent
current state checkered page:
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
current state h: 9.0
move in column 4 from row 6 to 2
current state checkered page:
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 1
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
current state h: 5.0
move in column 5 from row 7 to 6
current state checkered page:
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 1
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
current state h: 3.0
move in column 8 from row 2 to 1
current state checkered page:
0 1 0 0 0 0 0 1
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
```

```
move in column 6 from row 1 to 6
current state checkered page:
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1
1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
current state h: 3.0
final state checkered page:
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1
1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
final state h: 3.0
the simulated annealing could not find the solution
run time of hill climbing first choice 0.024975776672363281
run time of hill climbing steepest ascent 0.021001100540161133
run time of hill climbing random restart 0.044002771377563477
run time of simulated annealing 0.13201236724853516
```

TERIMA KASIH