

8 QUEENS DENGAN LOCAL SEARCH

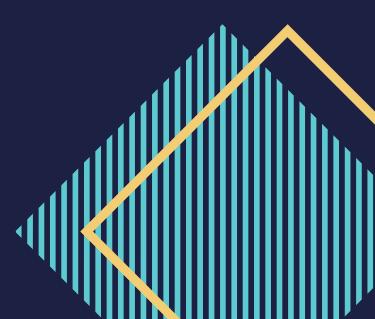
Kelompok 8

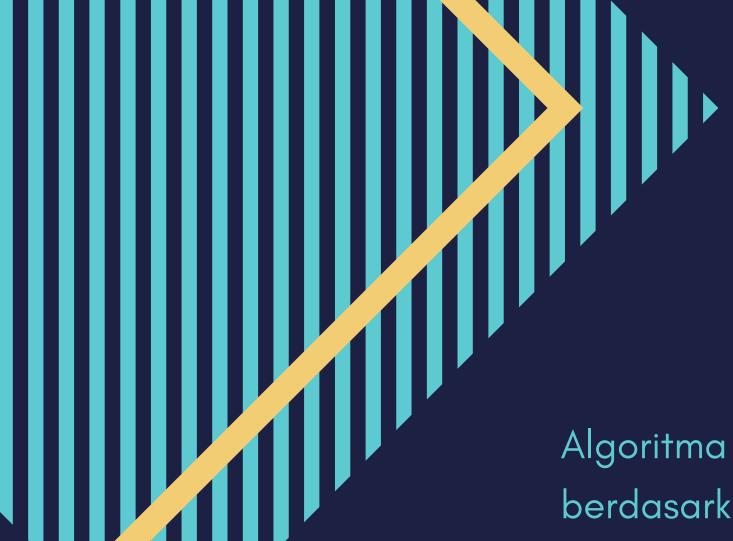
Anggota Kelompok:

Abdurrahman Farimza 5025201125

Kartika Diva Asmara Gita 5025211039

Hana Maheswari 5025211182





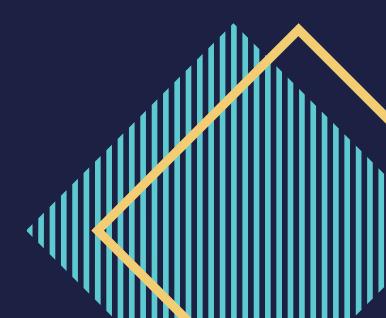
LOCAL SEARCH

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.



```
1 ∨ import random
     import copy
     import math
     import time
 6 \vee def C2n(n):
         return n * (n-1) / 2
10 ∨ class CheckeredPageState:
         def init (self, checkeredPage):
11 🗸
             self.checkeredPage = checkeredPage
12
             self.dimension = len(self.checkeredPage)
13
             self.setDic()
14
             self.setHeuristic()
15
16
         def setDic(self):
17 🗸
             dicRows = {}
18
             dicDiagonal1 = {}
19
             dicDiagonal2 = {}
20
             for i in range(self.dimension):
21 🗸
                 dicRows[i] = 0
22
                 for j in range(self.dimension):
23 🗸
                     dicDiagonal1[i-j] = 0
24
                     dicDiagonal2[i+j] = 0
25
             for i in range(self.dimension):
26 🗸
                 for j in range(self.dimension):
27 🗸
                     if self.checkeredPage[i][j]:
28 🗸
                         dicRows[i] += 1
29
                         dicDiagonal1[i-j] += 1
30
                         dicDiagonal2[j+i] += 1
31
             self.dicRows = dicRows
32
             self.dicDiagonal1 = dicDiagonal1
33
             self.dicDiagonal2 = dicDiagonal2
34
```

```
def setHeuristic(self):
36
37
              h = 0
             for key in self.dicRows:
38
39 🗸
                  if self.dicRows[key] > 1:
                      h += C2n(self.dicRows[key])
40
             for key in self.dicDiagonal1:
41 ∨
                  if self.dicDiagonal1[key] > 1:
42 🗸
                      h += C2n(self.dicDiagonal1[key])
43
44
             for key in self.dicDiagonal2:
                  if self.dicDiagonal2[key] > 1:
45
                      h += C2n(self.dicDiagonal2[key])
46
47
             self.h = h
48
         def getRandomSteepestAscent(self):
49
             neighbors = []
50
             huristic = float("inf")
51
             for j in range(self.dimension):
52
53
                  for i in range(self.dimension):
                      if self.checkeredPage[i][j] == 1:
54
55
                          ikeep = i
56
                          break
                  for i in range(self.dimension):
57
58
                      if self.checkeredPage[i][j] == 0:
                          newCheck = copy.deepcopy(self.checkeredPage)
59
                          newCheck[i][j] = 1
60
                          newCheck[ikeep][j] = 0
61
62
                          neighbor = CheckeredPageState(newCheck)
                          if neighbor.h < huristic:</pre>
63
64
                              neighbors[:] = []
                              huristic = neighbor.h
65
                          if neighbor.h == huristic:
66
                              neighbors.append(neighbor)
67
68
             return(random.choice(neighbors))
```



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

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



```
def HillCLimbingSteepestAscent(checkeredPageInitial):
          current = CheckeredPageState(checkeredPageInitial)
133
          print("start of hill climbing algorithm steepest ascent")
134
          while 1:
135
              print("current state checkered page:")
136
              current.printPage()
137
              print("current state h:", current.h)
138
              neighbor = current.getRandomSteepestAscent()
139
              if neighbor.h >= current.h:
140
                  if current.h == 0:
141
142
                      print("the hill climbing algorithm steepest ascent variant found a solution")
143
                      return True, current
144
                  else:
                      print("the hill climbing algorithm steepest ascent variant got stuck in local minimum")
145
                      return False, current
146
              current.getMove(neighbor)
147
              current = neighbor
148
149
      def HillCLimbingFirstChoice(checkeredPageInitial):
150
          current = CheckeredPageState(checkeredPageInitial)
151
          print("start of hill climbing algorithm first choice variant")
152
          while 1:
153
              print("current state checkered page:")
154
              current.printPage()
155
              print("current state h:", current.h)
156
157
              neighbor = current.getFirstChoice()
              if neighbor is None:
158
159
                  if current.h == 0:
160
                      print("the hill climbing algorithm first choice variant found a solution")
                      return True, current
161
162
                  else:
                      print("the hill climbing algorithm first choice variant got stuck in local minimum")
163
                      return False, current
164
              current.getMove(neighbor)
165
              current = neighbor
166
```

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



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

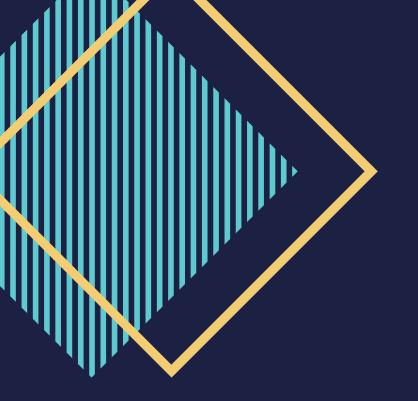
```
217
      for i in range(1):
         print("----")
218
         randomCheck = getRandomCheckeredPage(8)
219
         print("new random check generated")
220
         startHillFirst = time.time()
221
         HillCLimbingFirstChoice(randomCheck)
222
         endHillFirst = time.time()
223
         print("----")
224
         HillCLimbingSteepestAscent(randomCheck)
225
         endHillSteep = time.time()
226
         print("----")
227
         HillClimbingRandomRestart(8)
228
         endHillRandom = time.time()
229
         print("----")
230
         SimulatedAnnealing(randomCheck)
231
         endSim = time.time()
232
         print("run time of hill climbing first choice", endHillFirst - startHillFirst)
233
         print("run time of hill climbing steepest ascent", endHillSteep - endHillFirst)
234
         print("run time of hill climbing random restart", endHillRandom - endHillSteep)
235
         print("run time of simulated annealing", endSim - endHillSteep)
236
```

```
new random check generated
start of hill climbing algorithm first choice variant
current state checkered page:
01000000
00000001
00000010
10000000
00100000
00010000
00001000
00000100
current state h: 9.0
move in column 5 from row 7 to 3
current state checkered page:
01000000
00000001
00001010
10000000
00100000
00010000
00000000
00000100
current state h: 8.0
move in column 7 from row 3 to 8
current state checkered page:
01000000
00000001
00001000
10000000
00100000
00010000
00000000
00000110
current state h: 6.0
```

```
move in column 3 from row 5 to 8
current state checkered page:
01000000
00000001
00001000
10000000
00000000
00010000
00000000
00100110
current state h: 5.0
move in column 7 from row 8 to 5
current state checkered page:
01000000
00000001
00001000
10000000
00000010
00010000
00000000
00100100
current state h: 4.0
move in column 6 from row 8 to 7
current state checkered page:
01000000
00000001
00001000
10000000
00000010
00010000
00000100
00100000
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:
01000000
00000001
00000010
10000000
00100000
00010000
00001000
00000100
current state h: 9.0
move in column 4 from row 6 to 2
current state checkered page:
01000000
00010001
00000010
10000000
00100000
00000000
00001000
00000100
current state h: 5.0
move in column 5 from row 7 to 6
current state checkered page:
01000000
00010001
00000010
10000000
00100000
00001000
00000000
00000100
current state h: 3.0
move in column 8 from row 2 to 1
current state checkered page:
01000001
00010000
00000010
```

```
move in column 6 from row 1 to 6
current state checkered page:
00000000
00010000
00000001
10001000
00000000
01000100
00100000
00000010
current state h: 3.0
final state checkered page:
00000000
00010000
00000001
10001000
00000000
01000100
00100000
00000010
final state h: 3.0
the simulated annealing could not find the solution
run time of hill climbing first choice 0.02497577667236328
run time of hill climbing steepest ascent 0.021001100540161133
run time of hill climbing random restart 0.04400277137756348
run time of simulated annealing 0.13201236724853516
```



TERIMA KASIH

