02_heuristic_search

April 11, 2019

1 Testat Heuristic Search

1.1 Info

- All Questions answerd in the Document itself
- Code can be found on https://github.com/Inux/aiso

2 Exercise Informed Search Algorithms

In the last session, we implemented different systematic search strategies. If we want to find paths between different cities in a map, we can use additional information to guide our search. We don't rely on the 'blind' search and can implement more efficient algorithms, that consider the coordinates of the cities for example.

Implement the following algorithms and answer the same questions as you did for the systematic search algorithms.

1. Greedy Search
2. A* Algorithm
3. IDA* Search

In [1]:
```
import heapq

class PriorityQueue:
    """A Queue in which the minimum (or maximum) element (as determined by f and
    order) is returned first.
    If order is 'min', the item with minimum f(x) is
    returned first; if order is 'max', then it is the item with maximum f(x).
    Also supports dict-like lookup."""

    def __init__(self, order='min', f=lambda x: x):
        self.heap = []

        if order == 'min':
            self.f = f
        elif order == 'max':  # now item with max f(x)
            self.f = lambda x: -f(x)  # will be popped first
        else:
```

```
                raise ValueError("order must be either 'min' or max'.")

        def append(self, item):
            """Insert item at its correct position."""
            heapq.heappush(self.heap, (self.f(item), item))

        def extend(self, items):
            """Insert each item in items at its correct position."""
            for item in items:
                self.append(item)

        def pop(self):
            """Pop and return the item (with min or max f(x) value
            depending on the order."""
            if self.heap:
                return heapq.heappop(self.heap)[1]
            else:
                raise Exception('Trying to pop from empty PriorityQueue.')

        def __len__(self):
            """Return current capacity of PriorityQueue."""
            return len(self.heap)

        def __contains__(self, item):
            """Return True if item in PriorityQueue."""
            return (self.f(item), item) in self.heap

        def __getitem__(self, key):
            for _, item in self.heap:
                if item == key:
                    return item

        def __delitem__(self, key):
            """Delete the first occurrence of key."""
            self.heap.remove((self.f(key), key))
            heapq.heapify(self.heap)
```

In [2]:
```
from sbb import SBB
from search import *

sbb = SBB()
sbb.importData('linie-mit-betriebspunkten.json')

start = 'Rotkreuz'
goal = 'Thalwil'
sbb_map = UndirectedGraph(sbb.createMap())
problem = GraphProblem(start, goal, sbb_map)
```

```
heuristic = lambda node: sbb.get_distance_between(node.state, problem.goal)

def search_greedy(problem, heuristic):
    return search_heuristic(problem, heuristic)

def search_astar(problem, heuristic):
    return search_heuristic(problem, lambda node: heuristic(node) + node.path_cost)

def print_result(state, cost, explored, stored):
    print("node:", str(state))
    print("cost:", str(cost))
    print("nodes visited:", str(explored))
    print("nodes stored:", str(stored))

def search_heuristic(problem, heuristic):
    explored = set()

    node = Node(problem.initial)
    last_node = node

    iteration = 0
    max_iterations = 1000

    while iteration < max_iterations:
        iteration = iteration + 1
        print("Iteration:", iteration)

        childs = {}
        if problem.goal_test(node.state):
            print_result(node.state, node.path_cost, len(explored), len(explored))
            return node

        explored.add(node.state)

        childs = [(child, heuristic(child)) for child in node.expand(problem) if child n
        if not childs:
            print("search_heuristic: no valid childrens found!")
            return None

        #sort by best heuristic value
        childs.sort(key=lambda c: c[1])
        node = childs[0][0] #select first node of tuple(child, heuristic(child))

        if node == last_node:
            print("search_heuristic: stucked at: " + str(node))
            return None
        else:
            last_node = node
```

```
        return None

def search_astar_iterative(problem, heuristic):
    def priority(n):
        return heuristic(n) + n.path_cost

    node = Node(problem.initial)
    frontier = PriorityQueue('min', priority)
    frontier.append(node)
    explored = set()

    iteration = 0

    while frontier:
        iteration = iteration + 1
        print("Iteration:", iteration)

        node = frontier.pop()
        if problem.goal_test(node.state):
            print_result(node.state, node.path_cost, len(explored), len(explored)+len(fr
            return node

        explored.add(node.state)
        for child in node.expand(problem):
            in_frontier = len([n for _, n in frontier.heap if n.state == child.state]) >

            if not in_frontier and child.state not in explored:
                frontier.append(child)

    return None
```

```
successfully imported 2787 hubs
successfully imported 401 train lines
```

Hints: you can use the heap library heapq for your frontier:
`from heapq import heappush, heappop`
The following line will add the node `f` to the frontier with priority `f`:
`heappush(frontier, (f, node))`
To get the first node, use: `node = heappop(frontier)[1]`
The aerial distance between a node and the goal can be computed with the following function:
`sbb.get_distance_between(node.state, problem.goal)`

```
In [3]: greedy = search_greedy(problem, heuristic)
        if greedy is None:
            print("Greedy Search: No Solution!")
```

```
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Iteration: 10
Iteration: 11
Iteration: 12
Iteration: 13
Iteration: 14
Iteration: 15
Iteration: 16
Iteration: 17
node: Thalwil
cost: 37.272000000000006
nodes visited: 16
nodes stored: 16
```

In [4]:
```python
astar = search_astar(problem, heuristic)
if astar is None:
    print("Astar Search: No Solution!")
```

```
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Iteration: 10
Iteration: 11
Iteration: 12
Iteration: 13
Iteration: 14
Iteration: 15
Iteration: 16
Iteration: 17
Iteration: 18
Iteration: 19
Iteration: 20
Iteration: 21
```

```
Iteration: 22
Iteration: 23
Iteration: 24
Iteration: 25
Iteration: 26
Iteration: 27
Iteration: 28
Iteration: 29
Iteration: 30
Iteration: 31
Iteration: 32
Iteration: 33
Iteration: 34
Iteration: 35
Iteration: 36
Iteration: 37
Iteration: 38
Iteration: 39
Iteration: 40
Iteration: 41
Iteration: 42
Iteration: 43
Iteration: 44
Iteration: 45
Iteration: 46
Iteration: 47
Iteration: 48
Iteration: 49
Iteration: 50
Iteration: 51
Iteration: 52
Iteration: 53
Iteration: 54
Iteration: 55
Iteration: 56
Iteration: 57
Iteration: 58
Iteration: 59
Iteration: 60
Iteration: 61
Iteration: 62
Iteration: 63
Iteration: 64
Iteration: 65
Iteration: 66
Iteration: 67
Iteration: 68
Iteration: 69
```

```
Iteration: 70
Iteration: 71
Iteration: 72
Iteration: 73
Iteration: 74
Iteration: 75
Iteration: 76
Iteration: 77
Iteration: 78
Iteration: 79
Iteration: 80
Iteration: 81
Iteration: 82
Iteration: 83
Iteration: 84
Iteration: 85
Iteration: 86
Iteration: 87
Iteration: 88
Iteration: 89
Iteration: 90
Iteration: 91
Iteration: 92
Iteration: 93
Iteration: 94
Iteration: 95
Iteration: 96
Iteration: 97
Iteration: 98
Iteration: 99
Iteration: 100
Iteration: 101
Iteration: 102
Iteration: 103
Iteration: 104
Iteration: 105
Iteration: 106
Iteration: 107
Iteration: 108
Iteration: 109
Iteration: 110
Iteration: 111
Iteration: 112
Iteration: 113
Iteration: 114
Iteration: 115
Iteration: 116
Iteration: 117
```

```
Iteration: 118
Iteration: 119
Iteration: 120
Iteration: 121
Iteration: 122
Iteration: 123
Iteration: 124
Iteration: 125
Iteration: 126
Iteration: 127
Iteration: 128
Iteration: 129
Iteration: 130
Iteration: 131
Iteration: 132
Iteration: 133
Iteration: 134
Iteration: 135
Iteration: 136
Iteration: 137
Iteration: 138
Iteration: 139
Iteration: 140
Iteration: 141
Iteration: 142
Iteration: 143
Iteration: 144
Iteration: 145
Iteration: 146
Iteration: 147
Iteration: 148
Iteration: 149
Iteration: 150
Iteration: 151
Iteration: 152
Iteration: 153
Iteration: 154
Iteration: 155
Iteration: 156
Iteration: 157
Iteration: 158
Iteration: 159
Iteration: 160
Iteration: 161
Iteration: 162
Iteration: 163
Iteration: 164
Iteration: 165
```

```
Iteration: 166
Iteration: 167
Iteration: 168
Iteration: 169
Iteration: 170
Iteration: 171
Iteration: 172
Iteration: 173
Iteration: 174
Iteration: 175
Iteration: 176
Iteration: 177
Iteration: 178
Iteration: 179
Iteration: 180
Iteration: 181
Iteration: 182
Iteration: 183
Iteration: 184
Iteration: 185
Iteration: 186
Iteration: 187
Iteration: 188
Iteration: 189
Iteration: 190
Iteration: 191
Iteration: 192
Iteration: 193
Iteration: 194
Iteration: 195
Iteration: 196
Iteration: 197
Iteration: 198
Iteration: 199
Iteration: 200
Iteration: 201
Iteration: 202
Iteration: 203
Iteration: 204
Iteration: 205
Iteration: 206
Iteration: 207
Iteration: 208
Iteration: 209
Iteration: 210
Iteration: 211
Iteration: 212
Iteration: 213
```

```
Iteration: 214
Iteration: 215
Iteration: 216
Iteration: 217
Iteration: 218
Iteration: 219
Iteration: 220
Iteration: 221
Iteration: 222
Iteration: 223
Iteration: 224
Iteration: 225
Iteration: 226
Iteration: 227
Iteration: 228
Iteration: 229
Iteration: 230
Iteration: 231
Iteration: 232
Iteration: 233
Iteration: 234
Iteration: 235
Iteration: 236
Iteration: 237
Iteration: 238
Iteration: 239
Iteration: 240
Iteration: 241
Iteration: 242
Iteration: 243
Iteration: 244
Iteration: 245
Iteration: 246
Iteration: 247
Iteration: 248
Iteration: 249
Iteration: 250
Iteration: 251
Iteration: 252
Iteration: 253
Iteration: 254
Iteration: 255
Iteration: 256
Iteration: 257
Iteration: 258
Iteration: 259
Iteration: 260
Iteration: 261
```

```
Iteration: 262
Iteration: 263
Iteration: 264
Iteration: 265
Iteration: 266
Iteration: 267
Iteration: 268
Iteration: 269
Iteration: 270
Iteration: 271
Iteration: 272
Iteration: 273
Iteration: 274
Iteration: 275
Iteration: 276
Iteration: 277
Iteration: 278
Iteration: 279
Iteration: 280
Iteration: 281
Iteration: 282
Iteration: 283
Iteration: 284
Iteration: 285
Iteration: 286
Iteration: 287
Iteration: 288
Iteration: 289
Iteration: 290
Iteration: 291
Iteration: 292
Iteration: 293
Iteration: 294
Iteration: 295
Iteration: 296
Iteration: 297
Iteration: 298
Iteration: 299
Iteration: 300
Iteration: 301
Iteration: 302
Iteration: 303
Iteration: 304
Iteration: 305
Iteration: 306
Iteration: 307
Iteration: 308
Iteration: 309
```

```
Iteration: 310
Iteration: 311
Iteration: 312
Iteration: 313
Iteration: 314
Iteration: 315
Iteration: 316
Iteration: 317
Iteration: 318
Iteration: 319
Iteration: 320
Iteration: 321
Iteration: 322
Iteration: 323
Iteration: 324
Iteration: 325
Iteration: 326
Iteration: 327
Iteration: 328
Iteration: 329
Iteration: 330
Iteration: 331
Iteration: 332
Iteration: 333
Iteration: 334
Iteration: 335
Iteration: 336
Iteration: 337
Iteration: 338
Iteration: 339
Iteration: 340
Iteration: 341
Iteration: 342
Iteration: 343
Iteration: 344
Iteration: 345
Iteration: 346
Iteration: 347
Iteration: 348
Iteration: 349
Iteration: 350
Iteration: 351
Iteration: 352
Iteration: 353
Iteration: 354
Iteration: 355
Iteration: 356
Iteration: 357
```

```
Iteration: 358
Iteration: 359
Iteration: 360
Iteration: 361
Iteration: 362
Iteration: 363
Iteration: 364
Iteration: 365
Iteration: 366
Iteration: 367
Iteration: 368
Iteration: 369
Iteration: 370
Iteration: 371
Iteration: 372
Iteration: 373
Iteration: 374
Iteration: 375
Iteration: 376
Iteration: 377
Iteration: 378
Iteration: 379
Iteration: 380
Iteration: 381
Iteration: 382
Iteration: 383
Iteration: 384
Iteration: 385
Iteration: 386
Iteration: 387
Iteration: 388
Iteration: 389
Iteration: 390
Iteration: 391
Iteration: 392
Iteration: 393
Iteration: 394
Iteration: 395
Iteration: 396
Iteration: 397
Iteration: 398
Iteration: 399
Iteration: 400
Iteration: 401
Iteration: 402
Iteration: 403
Iteration: 404
Iteration: 405
```

```
Iteration: 406
Iteration: 407
Iteration: 408
Iteration: 409
Iteration: 410
Iteration: 411
Iteration: 412
Iteration: 413
Iteration: 414
Iteration: 415
Iteration: 416
Iteration: 417
Iteration: 418
Iteration: 419
Iteration: 420
Iteration: 421
Iteration: 422
Iteration: 423
Iteration: 424
Iteration: 425
Iteration: 426
Iteration: 427
Iteration: 428
Iteration: 429
Iteration: 430
Iteration: 431
Iteration: 432
Iteration: 433
Iteration: 434
Iteration: 435
Iteration: 436
Iteration: 437
Iteration: 438
Iteration: 439
Iteration: 440
Iteration: 441
Iteration: 442
Iteration: 443
Iteration: 444
Iteration: 445
Iteration: 446
Iteration: 447
Iteration: 448
Iteration: 449
Iteration: 450
Iteration: 451
Iteration: 452
Iteration: 453
```

```
Iteration: 454
Iteration: 455
Iteration: 456
Iteration: 457
Iteration: 458
Iteration: 459
Iteration: 460
Iteration: 461
Iteration: 462
Iteration: 463
Iteration: 464
Iteration: 465
Iteration: 466
Iteration: 467
Iteration: 468
Iteration: 469
Iteration: 470
Iteration: 471
Iteration: 472
Iteration: 473
Iteration: 474
Iteration: 475
Iteration: 476
Iteration: 477
Iteration: 478
Iteration: 479
Iteration: 480
Iteration: 481
Iteration: 482
Iteration: 483
Iteration: 484
Iteration: 485
Iteration: 486
Iteration: 487
Iteration: 488
Iteration: 489
Iteration: 490
Iteration: 491
Iteration: 492
Iteration: 493
Iteration: 494
Iteration: 495
Iteration: 496
Iteration: 497
Iteration: 498
Iteration: 499
Iteration: 500
Iteration: 501
```

```
Iteration: 502
Iteration: 503
Iteration: 504
Iteration: 505
Iteration: 506
Iteration: 507
Iteration: 508
Iteration: 509
Iteration: 510
Iteration: 511
Iteration: 512
Iteration: 513
Iteration: 514
Iteration: 515
Iteration: 516
Iteration: 517
Iteration: 518
Iteration: 519
Iteration: 520
Iteration: 521
Iteration: 522
Iteration: 523
Iteration: 524
Iteration: 525
Iteration: 526
Iteration: 527
Iteration: 528
Iteration: 529
Iteration: 530
Iteration: 531
Iteration: 532
Iteration: 533
Iteration: 534
Iteration: 535
Iteration: 536
Iteration: 537
Iteration: 538
Iteration: 539
Iteration: 540
Iteration: 541
Iteration: 542
Iteration: 543
Iteration: 544
Iteration: 545
Iteration: 546
Iteration: 547
Iteration: 548
Iteration: 549
```

```
Iteration: 550
Iteration: 551
Iteration: 552
Iteration: 553
Iteration: 554
Iteration: 555
Iteration: 556
Iteration: 557
Iteration: 558
Iteration: 559
Iteration: 560
Iteration: 561
Iteration: 562
Iteration: 563
Iteration: 564
Iteration: 565
Iteration: 566
Iteration: 567
Iteration: 568
Iteration: 569
Iteration: 570
Iteration: 571
Iteration: 572
Iteration: 573
Iteration: 574
Iteration: 575
Iteration: 576
Iteration: 577
Iteration: 578
Iteration: 579
Iteration: 580
Iteration: 581
Iteration: 582
Iteration: 583
Iteration: 584
Iteration: 585
Iteration: 586
Iteration: 587
Iteration: 588
Iteration: 589
Iteration: 590
Iteration: 591
Iteration: 592
Iteration: 593
Iteration: 594
Iteration: 595
Iteration: 596
Iteration: 597
```

```
Iteration: 598
Iteration: 599
Iteration: 600
Iteration: 601
Iteration: 602
Iteration: 603
Iteration: 604
Iteration: 605
Iteration: 606
Iteration: 607
Iteration: 608
Iteration: 609
Iteration: 610
Iteration: 611
Iteration: 612
Iteration: 613
Iteration: 614
Iteration: 615
Iteration: 616
Iteration: 617
Iteration: 618
Iteration: 619
Iteration: 620
Iteration: 621
Iteration: 622
Iteration: 623
Iteration: 624
Iteration: 625
Iteration: 626
Iteration: 627
Iteration: 628
Iteration: 629
Iteration: 630
Iteration: 631
Iteration: 632
Iteration: 633
Iteration: 634
Iteration: 635
Iteration: 636
Iteration: 637
Iteration: 638
Iteration: 639
Iteration: 640
Iteration: 641
Iteration: 642
Iteration: 643
Iteration: 644
Iteration: 645
```

```
Iteration: 646
Iteration: 647
Iteration: 648
Iteration: 649
Iteration: 650
Iteration: 651
Iteration: 652
Iteration: 653
Iteration: 654
Iteration: 655
Iteration: 656
Iteration: 657
Iteration: 658
Iteration: 659
Iteration: 660
Iteration: 661
Iteration: 662
Iteration: 663
Iteration: 664
Iteration: 665
Iteration: 666
Iteration: 667
Iteration: 668
Iteration: 669
Iteration: 670
Iteration: 671
Iteration: 672
Iteration: 673
Iteration: 674
Iteration: 675
Iteration: 676
Iteration: 677
Iteration: 678
Iteration: 679
Iteration: 680
Iteration: 681
Iteration: 682
Iteration: 683
Iteration: 684
Iteration: 685
Iteration: 686
Iteration: 687
Iteration: 688
Iteration: 689
Iteration: 690
Iteration: 691
Iteration: 692
Iteration: 693
```

```
Iteration: 694
Iteration: 695
Iteration: 696
Iteration: 697
Iteration: 698
Iteration: 699
Iteration: 700
Iteration: 701
Iteration: 702
Iteration: 703
Iteration: 704
Iteration: 705
Iteration: 706
Iteration: 707
Iteration: 708
Iteration: 709
Iteration: 710
Iteration: 711
Iteration: 712
Iteration: 713
Iteration: 714
Iteration: 715
Iteration: 716
Iteration: 717
Iteration: 718
Iteration: 719
Iteration: 720
Iteration: 721
Iteration: 722
Iteration: 723
Iteration: 724
Iteration: 725
Iteration: 726
Iteration: 727
Iteration: 728
Iteration: 729
Iteration: 730
Iteration: 731
Iteration: 732
Iteration: 733
Iteration: 734
Iteration: 735
Iteration: 736
Iteration: 737
Iteration: 738
Iteration: 739
Iteration: 740
Iteration: 741
```

```
Iteration: 742
Iteration: 743
Iteration: 744
Iteration: 745
Iteration: 746
Iteration: 747
Iteration: 748
Iteration: 749
Iteration: 750
Iteration: 751
Iteration: 752
Iteration: 753
Iteration: 754
Iteration: 755
Iteration: 756
Iteration: 757
Iteration: 758
Iteration: 759
Iteration: 760
Iteration: 761
Iteration: 762
Iteration: 763
Iteration: 764
Iteration: 765
Iteration: 766
Iteration: 767
Iteration: 768
Iteration: 769
Iteration: 770
Iteration: 771
Iteration: 772
Iteration: 773
Iteration: 774
Iteration: 775
Iteration: 776
Iteration: 777
Iteration: 778
Iteration: 779
Iteration: 780
Iteration: 781
Iteration: 782
Iteration: 783
Iteration: 784
Iteration: 785
Iteration: 786
Iteration: 787
Iteration: 788
Iteration: 789
```

```
Iteration: 790
Iteration: 791
Iteration: 792
Iteration: 793
Iteration: 794
Iteration: 795
Iteration: 796
Iteration: 797
Iteration: 798
Iteration: 799
Iteration: 800
Iteration: 801
Iteration: 802
Iteration: 803
Iteration: 804
Iteration: 805
Iteration: 806
Iteration: 807
Iteration: 808
Iteration: 809
Iteration: 810
Iteration: 811
Iteration: 812
Iteration: 813
Iteration: 814
Iteration: 815
Iteration: 816
Iteration: 817
Iteration: 818
Iteration: 819
Iteration: 820
Iteration: 821
Iteration: 822
Iteration: 823
Iteration: 824
Iteration: 825
Iteration: 826
Iteration: 827
Iteration: 828
Iteration: 829
Iteration: 830
Iteration: 831
Iteration: 832
Iteration: 833
Iteration: 834
Iteration: 835
Iteration: 836
Iteration: 837
```

```
Iteration: 838
Iteration: 839
Iteration: 840
Iteration: 841
Iteration: 842
Iteration: 843
Iteration: 844
Iteration: 845
Iteration: 846
Iteration: 847
Iteration: 848
Iteration: 849
Iteration: 850
Iteration: 851
Iteration: 852
Iteration: 853
Iteration: 854
Iteration: 855
Iteration: 856
Iteration: 857
Iteration: 858
Iteration: 859
Iteration: 860
Iteration: 861
Iteration: 862
Iteration: 863
Iteration: 864
Iteration: 865
Iteration: 866
Iteration: 867
Iteration: 868
Iteration: 869
Iteration: 870
Iteration: 871
Iteration: 872
Iteration: 873
Iteration: 874
Iteration: 875
Iteration: 876
Iteration: 877
Iteration: 878
Iteration: 879
Iteration: 880
Iteration: 881
Iteration: 882
Iteration: 883
Iteration: 884
Iteration: 885
```

```
Iteration: 886
Iteration: 887
Iteration: 888
Iteration: 889
Iteration: 890
Iteration: 891
Iteration: 892
Iteration: 893
Iteration: 894
Iteration: 895
Iteration: 896
Iteration: 897
Iteration: 898
Iteration: 899
Iteration: 900
Iteration: 901
Iteration: 902
Iteration: 903
Iteration: 904
Iteration: 905
Iteration: 906
Iteration: 907
Iteration: 908
Iteration: 909
Iteration: 910
Iteration: 911
Iteration: 912
Iteration: 913
Iteration: 914
Iteration: 915
Iteration: 916
Iteration: 917
Iteration: 918
Iteration: 919
Iteration: 920
Iteration: 921
Iteration: 922
Iteration: 923
Iteration: 924
Iteration: 925
Iteration: 926
Iteration: 927
Iteration: 928
Iteration: 929
Iteration: 930
Iteration: 931
Iteration: 932
Iteration: 933
```

```
Iteration: 934
Iteration: 935
Iteration: 936
Iteration: 937
Iteration: 938
Iteration: 939
Iteration: 940
Iteration: 941
Iteration: 942
Iteration: 943
Iteration: 944
Iteration: 945
Iteration: 946
Iteration: 947
Iteration: 948
Iteration: 949
Iteration: 950
Iteration: 951
Iteration: 952
Iteration: 953
Iteration: 954
Iteration: 955
Iteration: 956
Iteration: 957
Iteration: 958
Iteration: 959
Iteration: 960
Iteration: 961
Iteration: 962
Iteration: 963
Iteration: 964
Iteration: 965
Iteration: 966
Iteration: 967
Iteration: 968
Iteration: 969
Iteration: 970
Iteration: 971
Iteration: 972
Iteration: 973
Iteration: 974
Iteration: 975
Iteration: 976
Iteration: 977
Iteration: 978
Iteration: 979
Iteration: 980
Iteration: 981
```

```
Iteration: 982
Iteration: 983
Iteration: 984
Iteration: 985
Iteration: 986
Iteration: 987
Iteration: 988
Iteration: 989
Iteration: 990
Iteration: 991
Iteration: 992
Iteration: 993
Iteration: 994
Iteration: 995
Iteration: 996
Iteration: 997
Iteration: 998
Iteration: 999
Iteration: 1000
Astar Search: No Solution!
```

```
In [5]: astar_it = search_astar_iterative(problem, heuristic)
        if astar_it is None:
            print("Astar Iterative Search: No Solution!")
```

```
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Iteration: 10
Iteration: 11
Iteration: 12
Iteration: 13
Iteration: 14
Iteration: 15
Iteration: 16
Iteration: 17
Iteration: 18
Iteration: 19
Iteration: 20
Iteration: 21
Iteration: 22
```

```
Iteration: 23
Iteration: 24
Iteration: 25
Iteration: 26
Iteration: 27
Iteration: 28
Iteration: 29
Iteration: 30
Iteration: 31
Iteration: 32
Iteration: 33
Iteration: 34
Iteration: 35
Iteration: 36
Iteration: 37
Iteration: 38
Iteration: 39
Iteration: 40
node: Thalwil
cost: 36.906
nodes visited: 39
nodes stored: 48
```

How do theses informed search algorithms perform on our problem? Create the following overview table for the example problem.

Algorithm	start	goal	cost	number of nodes visited	maximal stored nodes	complete	optimal
Greedy Search	Rotkreuz	Thalwil	37.27	16	16	no	no
A*	Rotkreuz	Thalwil	no solution	no solution	no solution	yes	yes
IDA*	Rotkreuz	Thalwil	36.9	39	48	yes	yes