Seminarska naloga 2

Povezovanje točk v ravnini

Danijel Maraž

Fakulteta za Računalništvo in Informatiko UL dm9929@student.uni-lj.si

Abstract. The article covers the work done in the scope of the second seminar assignment as part of the subject Algorithms.

Keywords: Graham Scan Convex Hull Akl-Toussaint Heuristic

1 Uvod

Pri seminarski nalogi nam je bil dan cilj implementirati algoritem Graham scan z namenom iskanja konveksne ovojnice med točkami v ravnini. S pomočjo te smo sestavili svoj algoritem, ki ustvari disjunktne povezave med pari točk pod pogojem, da se te med seboj ne sekajo.

2 Struktura programa

```
def BFS_move(brd, c):
    visited = set()
    nextq = [(0, c, id(brd), brd)]
    heapq.heapify(nextq)
    while nextq:

            #Pop best rated board
        current = heapq.heappop(nextq)
        current[3].turn = moveside

            #Add board to visited
        zh = zobrist_hash(current[3])
        visited.add(zh - current[1])

        # Check if the given position is a checkmate
        if current[1] == 0:
            current[3].turn = enemyside
            if current[3].is_checkmate():
                sstr = ""
                moveslist = [str(m) for m in current[3].move_stack]
                for move in moveslist:
```

```
                sstr += move[:2] + "-" + move[2:] + ";"
            print(sstr[:-1])
            break
        current[3].turn = moveside
        continue
```

3 Graham Scan

4 Akl-Toussaint

5 Ovrednotev

Rezultati merjenja izboljšav med posameznimi hevristikami		
Brez KingSquares	Brez Manhattan	Vse hevristike
9.788 s	8.104 s	7.208 s
42/60	44/60	46/60

Tabela prikazuje rezultate pri testiranju končne programske rešitve z vsemi hevristikami, rešitev brez upoštevanja Manhattanske razdalje in brez hevristike za ocenjevanje razdalje do kraljevega kvadrata. Za vsako kategorijo sem pet krat pognal testne primere, vsakič računal povprečen čas iskanja rešitve in končnih pet povprečji povprečil v končno povprečje. Opazimo lahko, da je prav hevristika KingSquares najbolj pripomogla k hitrosti in posledično kakovosti rešitve. Teste sem izvajal iz druge Python datoteke brez posebnega testnega okolja pri naslednjih specifikacijah računalnika:

- Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90GHz
- 12.0 GB RAM
- 64-bit Operating System, x64-based processor

6 Programska koda

```
import chess
import sys
import random
import heapq
import time
from chess.polyglot import zobrist_hash

start = time.time()

file = open(sys.argv[1], "r")
fen0 = [line for line in file][0]
fen0s = fen0.split(" ")
avmoves = int(fen0.split(" ")[-1])
startfen = fen0s[-3] + " " + fen0s[-2] + " " + "- - 0 0"
board = chess.Board(startfen)

# Check who plays black or white
color = fen0s[1]
moveside = False
if color == "w":
    moveside = True
enemyside = not moveside


def KSCoverage(board, kingpos):
    return 100 * sum([len(board.attackers(moveside, a)) for a in
                      board.attacks(kingpos)
                      if board.is_attacked_by(moveside, a)])


def Man_dist(sq1, sq2):
    return abs(chess.square_file(sq1) - chess.square_file(sq2)) + abs(
        chess.square_rank(sq1) - chess.square_rank(sq2))


def ManhattanLight(move, kingpos):
    return 100 * (Man_dist(move.to_square, kingpos) - 1)


def BFS_move(brd, c):
    visited = set()
    nextq = [(0, c, id(brd), brd)]
    heapq.heapify(nextq)
```

```
while nextq:

    current = heapq.heappop(nextq)
    current[3].turn = moveside

    zh = zobrist_hash(current[3])
    visited.add(zh - current[1])
    # Check if the given position is a checkmate
    if current[1] == 0:
        current[3].turn = enemyside
        if current[3].is_checkmate():
            sstr = ""
            moveslist = [str(m) for m in current[3].move_stack]
            for move in moveslist:
                sstr += move[:2] + "-" + move[2:] + ";"
            print(sstr[:-1])
            break
        current[3].turn = moveside
        continue

    # The main loop
    for move in current[3].legal_moves:

        sc = current[1]

        current[3].push(move)
        current[3].turn = moveside

        # Eliminate premature attacks on the king + eliminate last moves tha
        wic = current[3].was_into_check()
        if (sc > 1 and wic) or (sc == 1 and not wic):
            current[3].pop()
            continue

        # Check if we already saw this board before
        zh = zobrist_hash(current[3])
        if zh - (sc - 1) in visited:
            current[3].pop()
            continue

        r = 0
        # Reward promotions
        prom = move.promotion
        if prom is not None:
            r -= -200
```

```
            if prom == chess.Piece(5, moveside):
                r -= 300
            if prom == chess.Piece(4, moveside):
                r -= 200

        # Desperate measures
        if (time.time() - start) > 17:
            rand = random.randint(1, 20)
            r += rand

        # Reward last turn attacks on the king
        if sc == 1:
            r -= 1000

        # Apply various heuristics

        # Coverage of squares around the King
        r -= KSCoverage(current[3], board.king(enemyside))
        # Manhattan distance from all non pawn figures to the King
        r += ManhattanLight(move, board.king(enemyside))

        # Reward depth
        r -= (sc - 1) * 100

        ccopy = current[3].copy()
        current[3].pop()

        heapq.heappush(nextq, (r, sc - 1, id(ccopy), ccopy))


BFS_move(board, avmoves)
```

References

Janko, V. (2015). *Razvoj programa za igranje 1-2-3 šaha* (Unpublished doctoral dissertation). Univerza v Ljubljani.