FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Řešič Morpion Solitaire
Student:	Kryštof Zindulka
Vedoucí:	Ing. Radomír Polách
Studijní program:	Informatika
Obor / specializace:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Nastudujte princip fungování Morpion Solitaire a seznamte se s nejlepšími dosaženými řešeními a metodami, kterými bylo řešení dosaženo.
Analyzujte a navhněte nástroj, který by Morpion Solitaire řešil, při implementaci myslete na využití paralelizace, běch v terminálovém prostředí, ukládání nejlepších nalezených řešení a možnost startu ze zadané konfigurace. Řešení Morpion Solitaire je NP-těžný problém a proto se zaměřte na návrh vhodné heuristiky k prožezávání stavového prostoru.
Implementované řešení důkladně testujte a zkuste dosáhnout co nejlepšího možného výsledku.

Elektronicky schválil/a doc. Ing. Jan Janoušek, Ph.D. dne 26. února 2021 v Praze.

Bakalářská práce

ŘEŠIČ MORPION SOLITAIRE

Kryštof Zindulka

Fakulta informačních technologií ČVUT v Praze
Katedra teoretické informatiky
Vedoucí: Ing. Radomír Polách
13. května 2021

Obsah

Seznam obrázků

Seznam tabulek

Chtěl bych poděkovat především vedoucímu této práce Ing. Radomíru Poláchovi za veškeré rady a také výběr tématu. Dále bych chtěl poděkovat své rodině za podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 11. května 2020 .....

Abstrakt

Morpion solitaire je hra s tužkou a papírem pro jednoho hráče. Cílem hry je zahrát co nejvíce tahů. Tato hra je zajímavá v tom, že překonat řešení nalezené člověkem trvalo pomocí počítače 34 let. V této práci jsou ukázána nejlepší řešení této hry nalezená jak člověkem, tak počítačem. Dále je představen nový algoritmus na hledání řešení využívající heuristiky, který je následně paralelizován. Na závěr jsou představena řešení nalezená pomocí tohoto algoritmu.

Klíčová slova morpion solitaire, prohledávání stavového prostoru, NP-težký problém, paralelizace

Abstract

Morpion solitaire is a paper and pencil game for one player. The main objective of the game is to play as many moves as you can. The interesting fact about this game is that it took 34 years to beat the human record using computers. In this thesis, both human and computer records are displayed. In the following sections, a new algorithm is presented for finding solutions of morpion solitaire. The presented sequential algorithm is paralyzed. Lastly, the solutions found using this algorithm are presented.

Keywords morpion solitaire, state space search, NP-hard problem, parallelization

Seznam zkratek

DFS	Depth first search

Úvod

Morpion soliatire je hra s tužkou a papírem pro jednoho hráče. Hraje se na čtverečkovaném papíře. Začíná se s obrazcem bodů ve tvaru symbolu plus a cílem je zahrát co nejvíce tahů. Tyto tahy nelze hrát do nekonečna. Nalézt optimální řešení nebo nějakou jeho aproximaci je NP-těžký problém.

Morpion solitaire pochází z Francie. Přesné počátky jsou neznámé. K prvnímu velkému proslavení hry došlo roku 1974, když byla zveřejněna ve francouzském magazínu *Science & Vie*. S tímto proslavením hry se lidé začali předhánět v nalezení co lepšího řešení. Později se tento problém rozšířil i mezi komunitou programátorů, kteří chtěli najít co nejlepší řešení pomocí počítače. Překonat počítačem lidský rekord trvalo 34 let.

Cílem této práce je nastudovat zatím nejlepší dosažená řešení hry morpion solitaire a seznámit se s metodami, kterými jich bylo dosaženo, dále navrhnout nástroj, který bude morpion solitaire řešit a dosáhnout pomocí něj co nejlepšího výsledku.

V této práci v kapitole 1 vysvětlím přesná pravidla a představím základní informace o hře.

V kapitole 2 ukážu zatím nejlepší nalezená řešení hry, kterých bylo dosaženo jak pouhým hraním této hry na papír, tak pomocí počítače.

V kapitole 3 představím návrh vlastního algoritmu na hledání řešení hry morpion solitaire. První představím návrh sekvenčního algoritmu a následně jeho paralelizaci. Na závěr této kapitoly ukážu, jakých řešení se mi pomocí mého algoritmu podařilo dosáhnout.

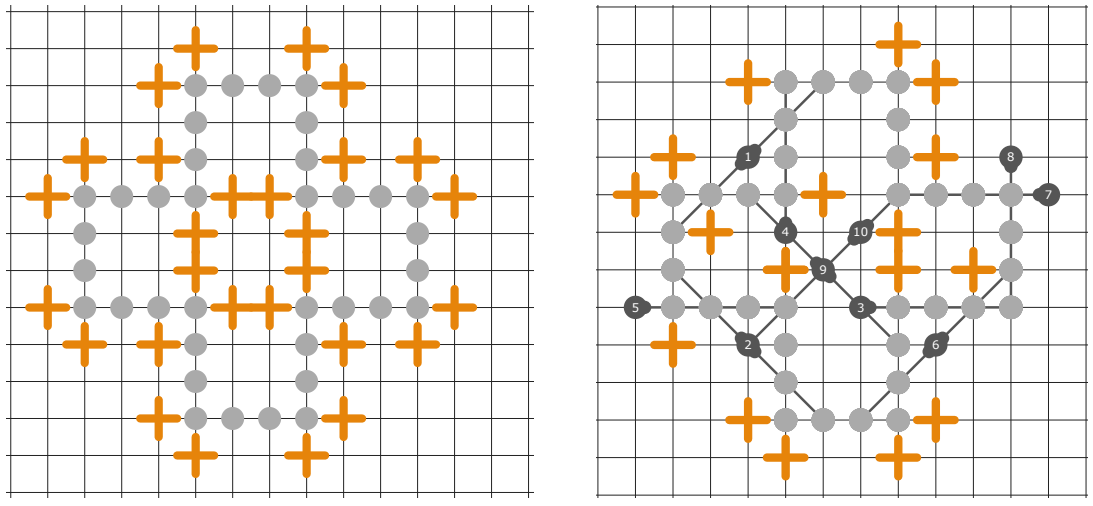
Kapitola 1

Morpion solitaire

1.1 Pravidla

Morpion solitaire je hra pro jednoho hráče, která se hraje s papírem a tužkou. Hrací plochou je čtvercová mřížka u které se předpokládá, že je nekonečná. Hrát se začíná ze zadané konfigurace 36 bodů ležících na mřížce ukázané na obrázku 1.1a. Cílem této hry je zahrát co nejvíce tahů. Tah spočívá v přidání nového bodu tak, že vznikne horizontální, vertikální nebo diagonální řada pěti bodů. Čára spojující tyto body je také přidána do hrací plochy. Jednotlivé čáry se nesmějí překrývat. Na obrázku 1.1b vidíme hrací plochu po zahrání deseti tahů. Místa, kam je možné zahrát tah, jsou na obrázcích zobrazeny oranžově. Pro zobrazování všech mřížek je použit nástroj `https://morsol.shy.cz` [1].

Hra končí, když není možné zahrát další tah. Počet zahraných tahů nebo počet čar na herní ploše je skóre hry.

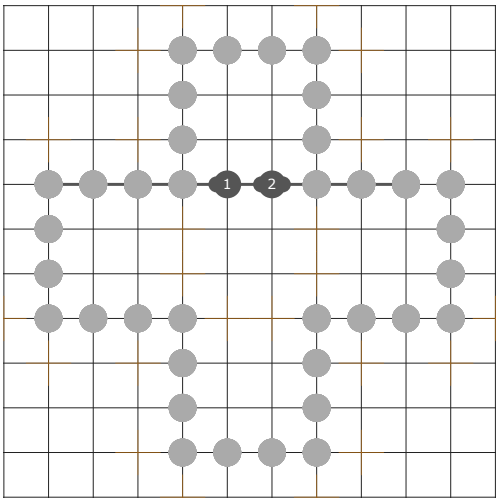


(a) Počáteční hrací plocha. **(b)** Hrací plocha po zahrání deseti tahů.

Obrázek 1.1 Hrací plocha.

1.2 Varianty hry

Existuje několik variant této hry. Mezi 2 nejznámější patří *Touching* a *Disjoint*. V *Touching* variantě mohou 2 stejnosměrné čáry sdílet koncový bod. V *Disjoint* variantě naopak nesmějí sdílet koncový bod. *Disjoint* varianta je více omezující. Všechny tahy možné v *Disjoint* variantě jsou možné i v *Touching* variantě, ale naopak to neplatí. Na obrázku 1.2 lze vidět tah číslo 2, který lze zahrát pouze v *Touching* variantě. V této práci se zaměřím na *Touching* variantu.



Obrázek 1.2 Druhý tah je možné zahrát pouze v *Touching* variantě.

1.3 Historie

Hra pochází z Francie. Přesné počátky jsou neznámé. Hru jako první velmi proslavil Pierre Berloquin, když o ní napsal článek v měsíčním magazínu *Science & Vie* roku 1974. Název je pravděpodobně odvozen ze dvou her, morpion a solitaire, které již existovaly dříve. [2]

1.4 Definice pojmů

Zde definuji několik pojmů, které budu v dalším textu používat.

▶ **Definice 1.1.** *Rozehraná hra se nazývá* Mřížka. Počáteční mřížka *je mřížka neobsahující žádné tahy.* Dokončená mřížka *je mřížka, ve které nelze již zahrát žádný další tah.*

▶ **Definice 1.2.** Velikost *nebo* skóre *mřížky je počet zahraných tahů v mřížce.*

▶ **Definice 1.3.** Posloupnost tahů *mřížky je posloupnost tahů, která vedla k vytvoření mřížky.*

▶ **Definice 1.4.** Možné tahy *mřížky jsou tahy, které je v mřížce možné zahrát.*

▶ **Definice 1.5.** *Dva možné tahy* kolidují *právě tehdy, když zahrání jednoho vyloučí zahrání druhého.*

▶ **Definice 1.6.** *Dvě mřížky jsou* podobné *právě tehdy, když jedna vznikne otočením nebo překlopením té druhé.*

▶ **Definice 1.7.** Čára *vznikne přidáním nového tahu. Každá* čára *spojuje 5 bodů.*

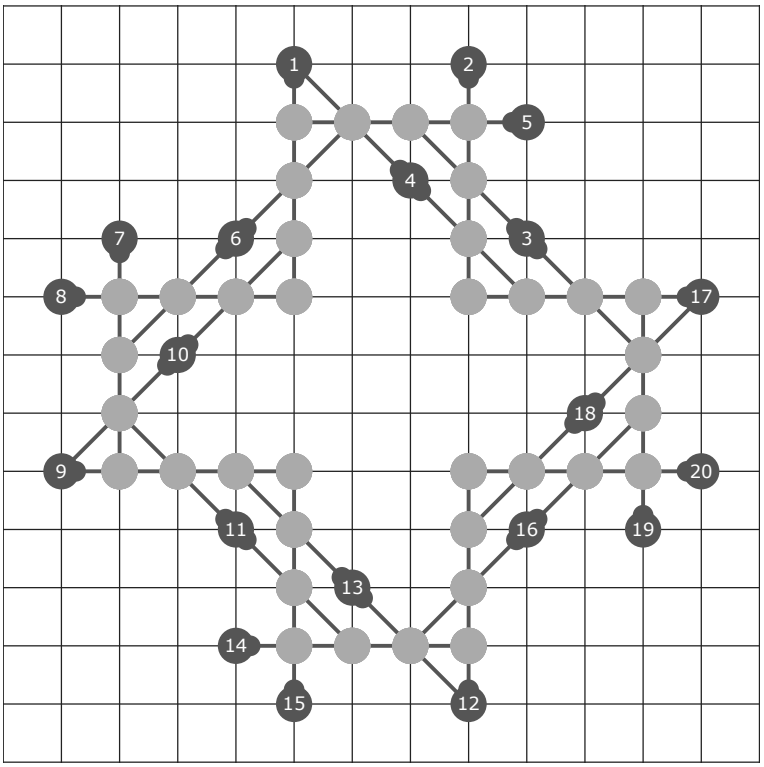
1.5 Dokázaná tvrzení

▶ **Věta 1.8.** *Nechť n je velikost největší možné mřížky. Nalézt mřížku velikosti n nebo velikosti n^{ϵ} pro nějaké $\epsilon > 0$ je také NP-těžké.* [3]

▶ **Věta 1.9.** *Horní hranice pro velikost mřížky je 485.* [4]

Matematicky bylo dokázáno, že neexistuje mřížka větší než 485. To však neznamená, že mřížka velikosti 485 existuje, jedná se pouze o horní hranici pro velikost mřížky.

▶ **Věta 1.10.** *Nejmenší dokončená mřížka má velikost 20.* [5]



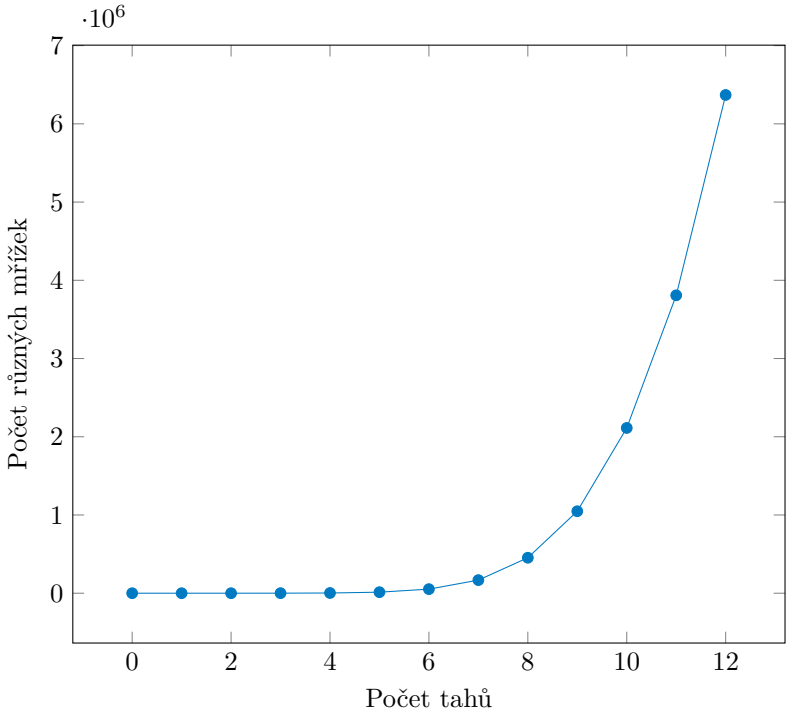
Obrázek 1.3 20 tahů. Nejmenší možná dokončená mřížka.

1.5.1 Počet možných mřížek

Pro prvních několik tahů bylo spočítáno, kolik existuje různých mřížek. V tabulce 1.1 je vypsáno prvních 22 tahů a prvních 12 je zobrazeno v grafu 1.4. z grafu je vidět exponenciální růst počtu různých mřížek vzhledem k počtu tahů. [6]

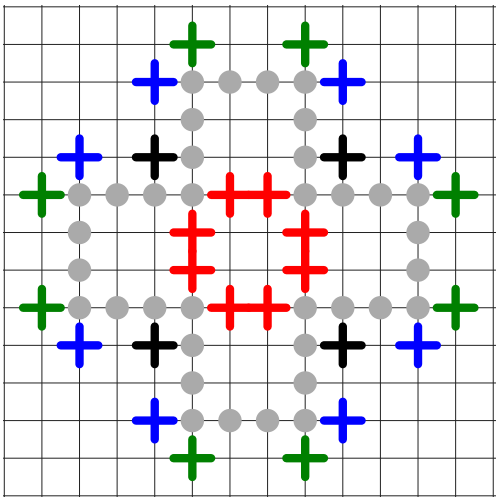
Počet tahů	Počet mřížek
0	1
1	4
2	56
3	428
4	2741
5	13247
6	52059
7	167502
8	453377
9	1047750
10	2112634
11	3808004
12	6369041
13	10436597
14	18107008
15	36365073
16	90462493
17	282733629
18	1074838721
19	4716194782
20	22663033612
21	114269420056
22	586835167740

Tabulka 1.1 Počet různých mřížek pro prvních 22 tahů.



Obrázek 1.4 Počet různých mřížek pro prvních 12 tahů.

V tomto vyčíslení se nepočítají podobné mřížky. Z toho důvodu je v tabulce 1.1 uvedeno, že po zahrání prvního tahu existují pouze 4 různé mřížky. Na obrázku 1.1a je vidět, že na zahrání prvního tahu je 28 možností. Těchto 28 možností se dá rozdělit do čtyř skupin, kde v každé skupině jsou tahy, které vedou na podobnou mřížku. Tyto 4 skupiny tahů jsou zobrazené na obrázku 1.5.



Obrázek 1.5 4 skupiny počátečních tahů.

Kapitola 2
Rekordy

Po zveřejnění hry morpion solitaire v magazínu *Science & Vie* roku 1974 se lidé začali snažit nalézt co největší mřížku. V této kapitole zmíním jednotlivé rekordy, které v průběhu času vznikaly. V druhé části ukážu rekordy nalezené pomocí počítačů.

2.1 Lidské rekordy

2.1.1 149 tahů

První zveřejněný rekord Pierrem Berloquinem v magazínu *Science & Vie* v roce 1974 byla mřížka Angličana Charlese Williama Millingtona obsahující 149 tahů. [7]

2.1.2 152 tahů

Ještě v roce 1974 Charles William Millington překonal svůj vlastní rekord. Jeho nová mřížka obsahovala 152 tahů. Tato mřížka vznikla upravováním jeho prvního rekordu. [7]

2.1.3 162 tahů

Stále ve stejném roce 1974 Francouz Rémy Daubié našel mřížku složenou z 162 tahů, čímž překonal stávající rekord o celých 10 tahů. Tato mřížka byla úplně nová, nevycházela z předchozího rekordu. [7]

2.1.4 163 tahů

Roku 1975 se do vedení vrátil Charles William Millington. Pozměnil mřížku stávajícího rekordu a dokázal do ní přidat jeden tah. [7]

2.1.5 164 tahů

Francouz Joseph Martin v roce 1975 stávající rekord ještě poupravil a podařilo se mu přidat další tah. [7]

2.1.6 170 tahů

Velké zlepšení přinesl roku 1976 Francouz Charles-Henri Bruneau. Ještě dříve než byla v magazínu *Science & Vie* zveřejněna mřížka velikosti 164 tahů, tak Charles-Henri Bruneau objevil mřížku velikosti 166 tahů. Tuto mřížku nezveřejňoval, dokud se mu dařila zlepšovat a v roce 1976 zveřejnil obdivuhodný rekord 170 tahů. Tato mřížka je na obrázku A.1. [8]

Tento rekord nebyl dodnes člověkem překonán. Roku 2010 ho po 34 letech překonal počítač.

2.2 Počítačové rekordy

2.2.1 117 tahů

První zveřejněnou mřížku nalezenou počítačem publikoval roku 1995 Američan Hugues Juillié jako součást svojí závěrečné práce na vysoké škole. Mřížka se skládala ze 117 tahů. Tohoto skóre dosáhl za použití evolučního algoritmu. [9]

2.2.2 143 tahů

V roce 2003 dosáhl algoritmus pana Pascala Zimmra 143 tahů. Použil algoritmus, který publikoval Hugues Juillié a upravil ho.

2.2.3 146 tahů

V roce 2010 překonal počítačový rekord japonský student Haruhiko Akiyama. Nalezl mřížku velikosti 146 zobrazenou na obrázku A.2. Použil upravený algoritmus *Nested Monte Carlo search*. Tento algoritmus na morpion solitaire použil první Tristan Cazenave na *Disjoint* verzi, našel rekord této verze a to 80 tahů.

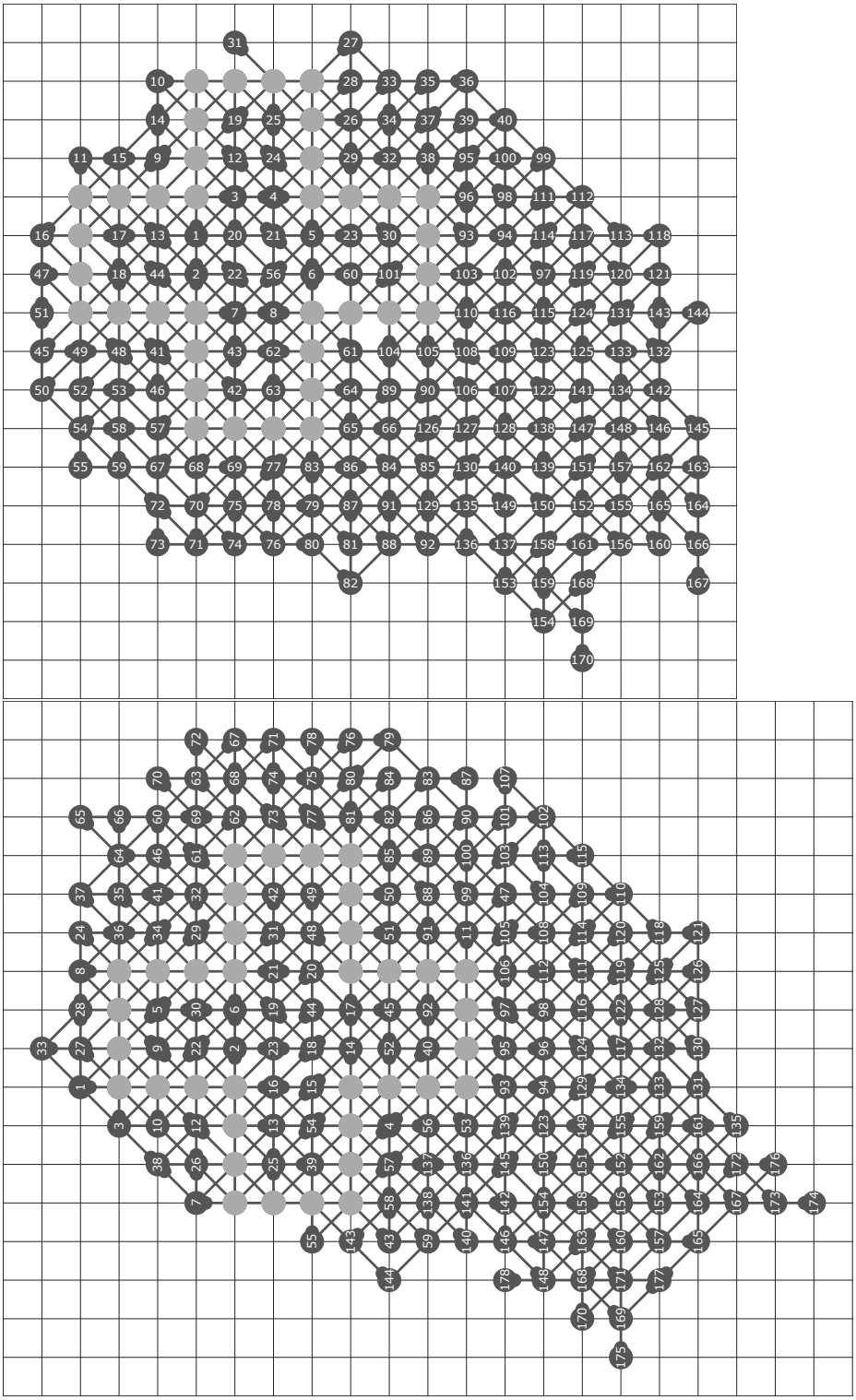
Tento algoritmus prohledává stavový prostor pomocí velkého množství náhodných her. V každém kroku vyzkouší všechny možné tahy a pokaždé hru několikrát náhodně dohraje. Zahraje tah, který našel nejlepší řešení. Tento postup se opakuje, dokud existují možné tahy. [10], [11]

2.2.4 178 tahů

Američan Christopher D. Rosin v roce 2010 našel mřížku velikosti 171 tahů pomocí algoritmu *Nested Rollout Policy Adaptation for Monte Carlo Tree Search*. Ještě roku 2010 překonal svůj rekord novým skóre 172. V roce 2011 našel ještě 2 nové mřížky velikosti 177 a 178. Jedná se o první prekonání lidského rekordu pomocí počítače. Lidský rekord si držel první místo 34 let. [11]

Algoritmus *Nested Rollout Policy Adaptation for Monte Carlo Tree Search* je upravený algoritmus *Nested Monte Carlo search*. Funguje velice podobně. Také používá k procházení stavového prostoru velké množství náhodných her. V tomto případě ale hry nejsou úplně náhodné. Pokaždé, když se nalezne nové nejlepší řešení, tak se zvětší pravděpodobnost jednotlivým jeho tahům. To znamená, že se hledání více zaměřuje na okolí dosud nalezeného nejlepšího výsledku. Tato poměrně malá změna přinesla výrazně lepší výsledky hledání. [12]

Při pootočení této mřížky ve vidět určitá podoba s lidským rekordem. Tato podoba je zobrazená na obrázku 2.1. Na obrázku A.3 je vidět nepootočená mřížka s rekordem.



Obrázek 2.1 Porovnání lidského a počítačového rekordu. Lidský rekord nahoře (170 tahů), počítačový rekord dole (178 tahů)

2.2.4.1 Pokus o překonání rekordu

Roku 2019 Polák Andrzej Nagórko upravil a paralelizoval Rosinův algoritmus. Upravený algoritmus spouštěl na 768 jádrovém počítači a dosáhl 547 násobného zrychlení. Z deseti běhů programu každý našel aktuální rekord 178 tahů. Všechny nalezené mřížky byly až na pořadí tahů stejné jako rekordní mřížka. Překonat rekord se však nepodařilo. [13]

Kapitola 3

Návrh nové heuristiky

V této kapitole předvedu návrh vlastní heuristiky na prořezávání stavového prostoru. V minulé kapitole jsem popsal algoritmy, které vedly k dosud nejlepším řešením. Navrhnu zcela odlišný algoritmus, který na těchto algoritmech nebude založený. Použít stejný algoritmus, jako vedl k doposud nejlepšímu řešení a nějak ho upravit by nejspíše vedlo k dobrým výsledkům, ale nepřinášel bych tím nic nového. Jak jsem již zmínil v 2.2.4.1, tak tento algoritmus byl spuštěn na velice výkonném počítači a i tak nenašel lepší řešení než 178. Popíšu jednotlivé algoritmy, které jsem navrhl, a jakého výsledku dosáhly.

3.1 Datová reprezentace

Zde popíšu, jak si jednotlivé herní entity v paměti reprezentuji, což pomůže pochopení následných postupů.

3.1.1 Programovací jazyk

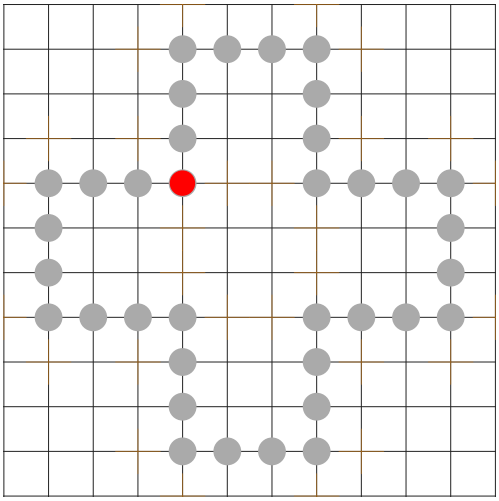
Jako programovací jazyk jsem zvolil C++ z důvodu vysoké efektivity.

3.1.2 Formát ukládání řešení

Na ukládání jednotlivých řešení v textovém formátu se mezi hráči morpion solitaire používá formát, kde jsou na řádcích napsány tahy v pořadí, v jakém byly hrány. Zapsání jednoho tahu je následující:

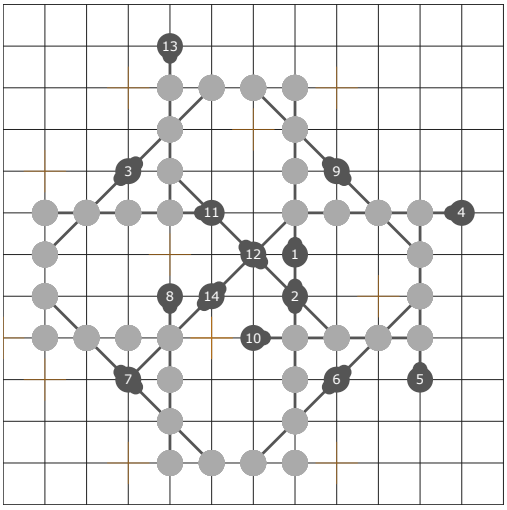
`({souřadnice x}, {souřadnice y}) {směr čáry} {posunutí čáry}`.

Počátek soustavy souřadnic je vlevo nahoře. Na prvním řádku jsou souřadnice bodu označeného na obrázku 3.1. Vetšinou se pro tento bod používají souřadnice $(28, 28)$. Směr čáry je určen znaménkem `(|, /, -, \)`. Posunutí čáry je 0, když je nový bod středem čáry, jinak je to číslo mezi 2 a -2, podle posunutí vzhledem k počátku souřadnic.



Obrázek 3.1 Podle označeného bodu se nastavují souřadnice.

(28,28)
(31,29) | -2
(31,30) | 1
(27,27) / 0
(35,28) – -2
(34,32) | -2
(32,32) / 0
(27,32) \ 0
(28,30) | 2
(32,27) \ 0
(30,31) – 2
(29,28) – -2
(30,29) \ 0
(28,24) | 2
(29,30) / 0



Obrázek 3.2 Ukázka formátu na ukládání řešení.

3.1.3 Hrací plocha

Hrací plocha je nekonečná. Nekonečnou hrací plochu nelze mít v paměti uloženou. Mohl bych udělat dynamicky se zvětšující hrací plochu, která by se zvětšila, pokud by se nové tahy blížily ke kraji uložené plochy, ale radši jsem si zvolil plochu pevné velikosti.

Hrací plocha je sice teoreticky nekonečná, ale ani nejlepší řešení neobsahuje tah, který by se nacházel dále, než 20 bodů od středu počátečního kříže [14]. Hrací plocha pevné velikosti je snadnější na implementaci a ušetří se nějaké instrukce tím, že se nemusí neustále kontrolovat, zda se nemá zvětšit. Zvolil jsem plochu velikosti 60×60, aby byl kříž reprezentovaný souřadnicemi podle obrázku 3.1 přesně uprostřed.

Hrací plochu reprezentuji jako 2D pole. Každý bod v tomto 2D poli drží informaci o tom, zda byl přidán do hry neboli zahrán. Pokud ano, tak si také drží informace pro všechny 4 směry o tom, zda do něj vchází a zda z něj vychází čára.

3.1.4 Tahy

Nyní je otázkou, jak reprezentovat a hledat možné tahy. Zde je znovu více možností.

Mohl bych možné tahy spočítat vždy pouze na základě hrací plochy. Výhodou tohoto řešení je, že by stačilo ručně v programu uložit jednotlivé počáteční body a možné tahy by si pak našel program sám. Takové řešení by ale znamenalo procházení celé hrací plochy po každém tahu a to by vyžadovalo mnoho operací.

Druhá možnost je tahy vždy přepočítat pouze na základě posledně zahraného tahu. Nevýhodou je, že tahy v počáteční konfiguraci je nutné do programu ručně vypsat. Výhodou je, že po každém tahu stačí projít jeho blízké okolí a přidat nebo odebrat tahy. Na to bude stačit výrazně méně operací, než na procházení celé hrací plochy.

Z důvodu efektivity jsem se rozhodl pro druhou možnost. Jednotlivé možné tahy reprezentuji podobně, jako je to v používaném formátu na ukládání řešení 3.1.2. Každý tah si drží informaci směru čáry, posunutí čáry, a o souřadnicích bodu, který přidá. V některých případech se stane, že různé možné tahy přidávají stejný bod, ale jinou čáru. V takovém případě je zřejmé, že bude možné zahrát pouze jeden z těchto tahů.

Tyto tahy si ukládám v datové struktuře list implementované spojovým seznamem. Z listu mohu na rozdíl od pole odebírat tahy v konstantním čase.

3.1.5 Formát pseudokódů

Ke každému algoritmu napíšu pseudokód. Aby bylo jasné, jak algoritmus pracuje s rozhraním hry, tak zde vysvětlím jednotlivé funkce, které budu používat.

Struktura *Plocha* reprezentuje herní plochu. Struktura *Tahy* reprezentuje spojový seznam možných tahů a *Tah* je jeden tah z tohoto spojového seznamu.

***Plocha.zahraj(Tah)*:** Zaznamená v ploše nový tah.

***Plocha.smaz()*:** Smaže z plochy poslední zahraný tah.

***Tahy.zahraj(Tah)*:** Odebere z možných tahů tahy, které s daným tahem kolidují a přidá tahy, které vzniknou zahráním daného tahu.

***Tahy.pocet()*:** Vrátí počet prvků v listu *Tahy*.

***Plocha.skore*:** Vrátí kolik tahů je zrovna zahráno.

***Tahy(Tah, konec)*:** Vytvoří nový list, který je kopií listu *Tahy* od prvku *Tah* do konce.

3.2 Sekvenční řešení

3.2.1 Náhodné tahy

Jako první jsem napsal algoritmus, který hraje tahy pouze náhodně. Algoritmus začne v počáteční konfiguraci a hraje náhodné tahy tak dlouho, dokud je to možné. Tento algoritmus jsem spustil milionkrát a největší mřížka měla velikost 83 tahů. Tato mřížka je na obrázku B.1. Oproti rekordům, které jsem ukázal v kapitole 2, 83 tahů je velice málo, ale zahrát tolik tahů nemusí být pro začátečníka jednoduché.

Algoritmus 1: Náhodné tahy

Vstup : Počáteční deska *Plocha*, List možných počátečních tahů *Tahy*
Výstup: Hrací deska *Plocha* ve stavu náhodně zahrané hry

```
1 Dokud Tahy.pocet() > 0:
2     Vyber náhodný tah Tah z Tahy.
3     Plocha.zahraj(Tah)
4     Tahy.zahraj(Tah)
```

3.2.2 Depth first search

Minulý algoritmus při každém běhu našel jedno řešení a tím skočil. Nyní je na čase navrhnout nějakou heuristiku, která stavový prostor prořezává a nezastaví se s nalezením prvního řešení. Zvolil jsem si rekurzivní algoritmus depth first search (DFS) neboli hledání do hloubky.

Algoritmus prochází strom, ve kterém jsou jednotlivé vrcholy různé stavy hry a hrany jsou jednotlivé tahy. V listech jsou hry ve stavu, že není možné zahrát další tah. Kořen je počáteční konfigurace hry a hrany vycházející z kořenu jsou tahy možné při počáteční konfiguraci.

Běh algoritmu je popsán následujícím pseudokódem.

Algoritmus 2: DFS

Vstup: Počáteční deska *Plocha*, List možných počátečních tahů *Tahy*

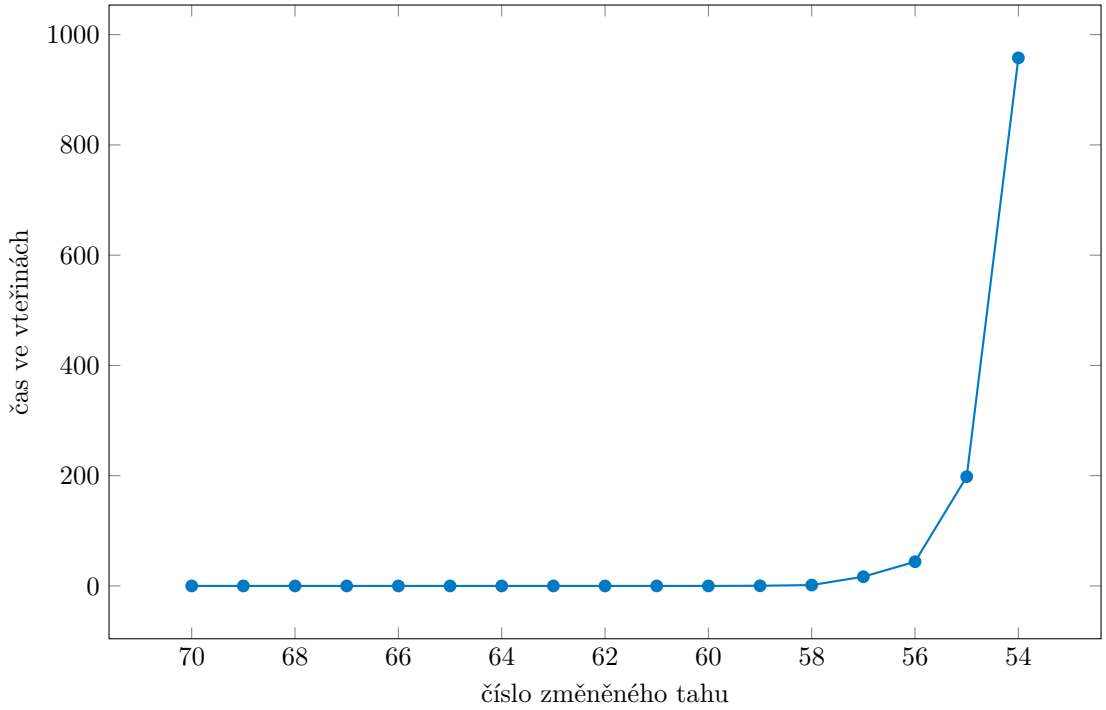
```
1  Pokud Tahy.pocet() == 0:
2      Pokud Plocha.skore > maxSkore:
3          ulož Plocha.
4          maxScore = Plocha.skore
5  Jinak:
6      Pro každý Tah z Tahy:
7          noveTahy = Tahy
8          Plocha.zahraj(Tah)
9          noveTahy.zahraj(Tah)
10         DFS(Plocha, noveTahy)
11         Plocha.smaz()
```

Existuje pouze jedna instance hrací plochy, z toho důvodu se při návratu z rekurze musí z plochy smazat poslední tah. List tahů se vytváří nový při každém rekurzivním volání.

Tento algoritmus by časem prošel celý stavový prostor a našel optimální řešení, protože zkouší všechny kombinace tahů. Ale vzhledem k velikosti prostoru by se nikdo z nás nedožil výsledku. V rozumně dlouhém čase stihne tento algoritmus projít pouze zanedbatelně malou část stavového prostoru.

Tento algoritmus našel během několika vteřin řešení složené ze 71 tahů viditelné na obrázku B.2. Dále ani po 1000 vteřinách nenašel lepší řešení.

Abych zjistil, jak velkou část stavového prostoru tento algoritmus prochází, tak jsem si vypisoval po jaké době se algoritmus rekurzivně poprvé vrátí na jaký tah. V grafu 3.3 lze vidět, jak dlouho to trvalo, než se algoritmus vrátil k určitému tahu. Z obrázku lze jasně vidět, jak doba změny tahu na nižší úrovni exponenciálně stoupá. Za 1000 vteřin algoritmus nezměnil ani jeden z prvních 53 tahů.



Obrázek 3.3 Doba potřebná k návratu k tahu na určité úrovni u algoritmu DFS.

3.2.3 Vyřazení opakujících se stavů

Jeden z problémů použití dříve zmíněného DFS je, že se stejné stavy hry procházejí vícekrát. Pokud mám tahy A a B, které nekolidují, a zahraji první A a pak B nebo první B a pak A, tak se dostanu do stejného stavu.

▶ **Věta 3.1.** *Je dána validní posloupnost tahů nějakého řešení. Při sestavování mřížky lze v každém kroku vybrat jakýkoliv tah posloupnosti, který pravidla umožní do aktuální mřížky přidat.*

Důkaz. Opravdu není nutné mřížku sestavit přesně podle pořadí tahů. Přidání tahu dříve, než je v pořadí, nemůže zabránit přidání žádného jiného tahu z posloupnosti. Pokud by se tak stalo, tak by tyto 2 tahy kolidovaly a nebylo by možné, aby byly ve stejné validní posloupnosti. ◀

▶ **Důsledek 3.2.** *Když se algoritmus DFS vrátí z podstromu do kterého vešel tahem A, tak již prošel všechny možné dokončení aktuální hry obsahující tah A.*

Pokud algoritmus DFS vstupuje do nějakého podstromu, tak mu nemusí předávat všechny možné tahy v aktuální mřížce, ale stačí mu předat pouze tahy, kterými na aktuální úrovni do žádného podstromu nevstoupil.

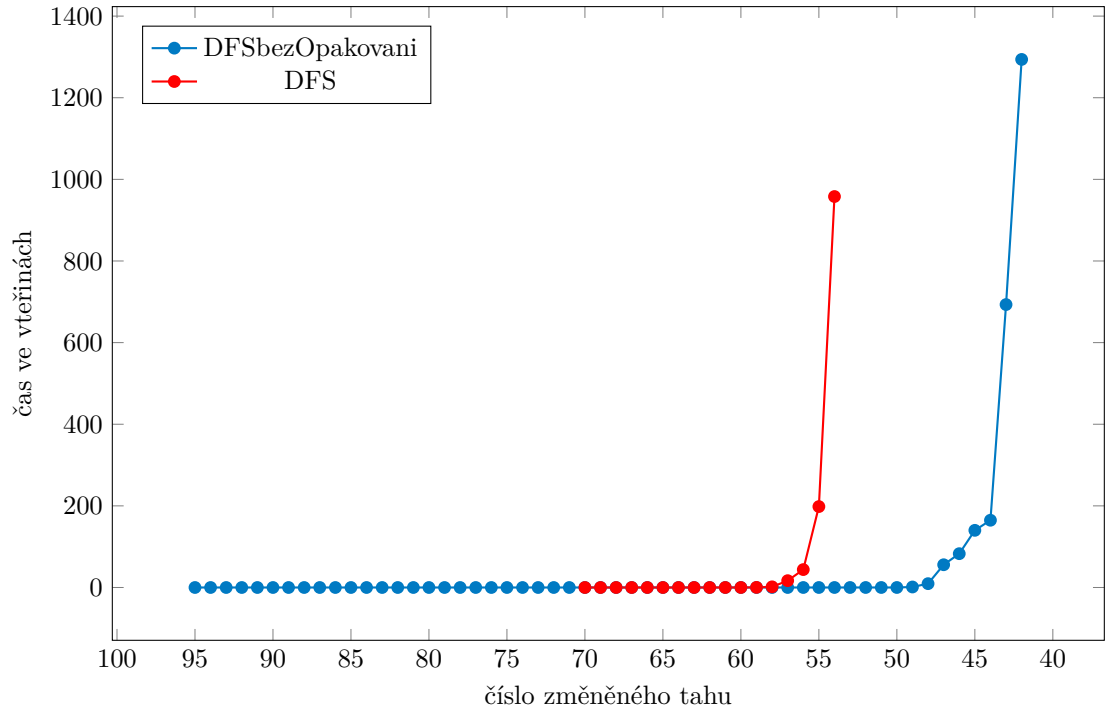
Pro tuto úpravu stačí pozměnit předchozí algoritmus na řádku 7.

Algoritmus 3: DFSbezOpakovani

Vstup: Počáteční deska *Plocha*, List možných počátečních tahů *Tahy*

```
1  Pokud Tahy.pocet() == 0:
2      Pokud Plocha.skore > maxSkore:
3          ulož Plocha.
4          maxScore = Plocha.skore
5  Jinak:
6      Pro každý Tah z Tahy:
7          noveTahy = Tahy(Tah, konec) // změna oproti přechozímu algoritmu
8          Plocha.zahraj(Tah)
9          noveTahy.zahraj(Tah)
10         DFS(Plocha, noveTahy)
11         Plocha.smaz()
```

Tato změna přinesla výrazné zlepšení výsledků. Během minuty našel algoritmus mřížku skládající se z 96 tahů. Tato mřížka je zobrazena na obrázku B.3. Také je z grafu 3.4 vidět, že tento algoritmus se vrací zpět k dřívějším tahům výrazně rychleji než pouhé DFS 3.3. Oba algoritmy jsem nechal běžet 1500 vteřin. DFS se vrátilo k tahu 54 s tím, že našlo řešení pouze 71 tahů. Vrátilo se tedy o 17 tahů zpět. DFSbezOpakovani našlo řešení 96 tahů a vrátilo se až k tahu 42, což znamená vrácení o 54 tahů.



Obrázek 3.4 Porovnání doby návratu k tahu na určité úrovni u algoritmu DFS a algortimu DFSbezOpakovani.

3.2.4 Odstranění zbytečných tahů

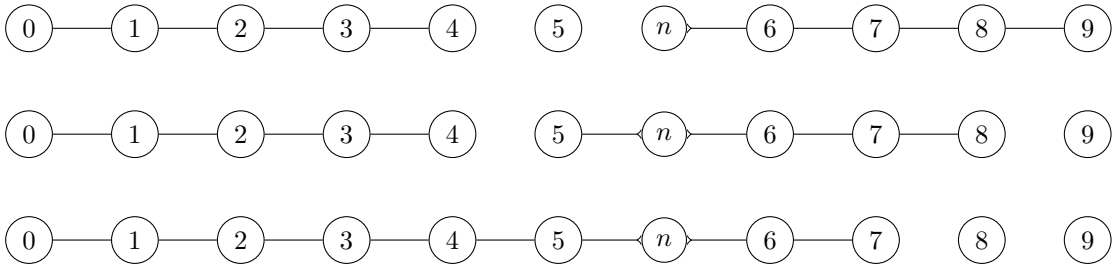
Pokud více možných tahů přidává bod na stejné místo, tak je zřejmé, že zahrání jednoho znemožní zahrání ostatních. Pokud chceme vybrat jeden z nich, tak vzhledem k tomu, že všechny přidají

bod na stejné místo, tak vybíráme pouze to, jak umístíme čáru.

Pokud více možných tahů přidává bod na stejné místo a čáru stejným směrem, tak je někdy jeden z nich jednoznačně lepší než ostatní. Tato situace tedy může nastat, když při přidání nějakého bodu existuje více možností, jak přidat čáru v jednom směru.

Na obrázku 3.5 vidíme část herní plochy. Na volné místo lze přidat bod a jsou tři různé způsoby, jak přidat čáru. Tyto tři způsoby jsou zobrazené na obrázku 3.6.

Obrázek 3.5 Pokud se na volné místo přidá bod, tak jsou tři různé způsoby jak přidat čáru.



Obrázek 3.6 Tři způsoby jak přidat čáru u přidání bodu n.

Je zřejmé, že ze tří možností, jak nakreslit čáru, zobrazených na obrázku 3.6, je nejlepší ta třetí. Ve třetím případě může stejnosměrná čára vedoucí doprava začínat již v bodě 7, ale také v bodech 8 a 9. Ve druhém případě může tato čára začínat v bodech 8 a 9. V prvním případě může čára začínat pouze v bodě 9. Třetí možnost nakreslení čáry tedy nejméně omezuje nové tahy. Hrát jednu z prvních dvou možností je méně výhodné.

Přidal jsem do algoritmu odstraňování těchto zbytečných tahů. Tato úprava nepomohla nalézt lepší řešení, ale algoritmus se zrychlil.

3.2.5 Seřazení tahů

Zatím algoritmus procházel tahy bez nějakého definovaného pořadí. Pořadí bylo určeno pořadím počátečních tahů ručně zadaných do programu a dále funkcemi na přidávání tahů. Vzhledem k tomu, že celý stavový prostor se projít v rozumném čase nestihne, tak je pořadí procházení tahů velmi důležité. Bohužel vyhodnotit, které tahy jsou lepší než ostatní, není vůbec jednoduché, protože se jedná o NP-těžký problém [3]. Vyzkoušel jsem několik různých přístupů na řazení tahů.

3.2.5.1 Řazení podle souřadnic

Pozorováním nejlepšího řešení A.3 jsem si všiml, že se tato mřížka táhne směrem dolů a trochu směrem doleva. Pro napodobení šíření mřížky jedním směrem se v tomto přístupu rozhodl seřadit tahy podle souřadnic. Dříve bude hrán tah, který má menší souřadnici x. Pokud mají 2 tahy stejnou souřadnici x, tak bude dříve hrán tah s menší souřadnicí y. Znamená to, že se budou upřednostňovat tahy vlevo a dále tahy nahoře.

3.2.5.2 Řazení podle vah tahů

Přístup který se přirozeně nabízí, je seřadit tahy podle toho, kolik bude možných tahů po jejich zahrání. Tahy, po jejichž zahrání bude nejvíce možných tahů, se zahrají dříve, než ostatní tahy.

Pro vysvětlení tohoto způsobu řazení budu používat pojem *váha* tahu. Váha tahu je rozdíl počtu nových možných tahů po zahrání tahu a počtu možných tahů, které s tahem kolidují. Z toho důvodu se tento způsob jmenuje řazení podle vah tahů.

Existují 2 způsoby, jak interpretovat počet tahů. První způsob je počítat všechny tahy. Tedy více tahů, které přidávají bod na stejné místo a liší se pouze v čáře se počítá jako více tahů. Tento způsob je v mém návrhu jednodušší na implementaci. Tento způsob bude dále označován jako řazení podle *celkových vah*.

Druhý způsob je počítat různé místa, kam jednotlivé možné tahy můžou přidat bod. Více tahů přidávající bod na stejné místo se počítá jako jeden. Tento způsob bude dále označován jako řazení podle *bodových vah*.

3.2.5.3 Řazení podle čar

Na tento přístup jsem přišel znovu pozorováním nejlepšího řešení A.3. Všiml jsem si, že počet čar vedoucích jednotlivými čtyřmi směry je podobný.

Směr čáry	Počet čar
\	42
/	44
-	45
|	47

Tabulka 3.1 Počet čar v nejlepším řešení A.3 vedoucích všemi směry.

V tabulce 3.1 je ukázáno, kolik čar vede každým směrem v nejlepším řešení A.3 obsahujícím 178 tahů. Nejvíce používaná čára je použita 47×, a nejméně používaná čára je použita 42×. To znamená, že se v tomto řešení nevyskytuje nějaká čára, která by bylo použita výrazně vícekrát než ostatní čáry nebo naopak čára, která by byla málo využívaná. Pro zachování rovnoměrného používání směrů čar, jsem se rozhodl řadit tahy podle jejich čar. První budou hrané tahy přidávající čáru, která byla zatím nejméně využívaná.

3.2.5.4 Spojení předchozích dvou přístupů

U řazení podle čar a podle počtu přidaných tahů se stává, že se při řazení více tahů rovná. Z toho důvodu je také možné tyto 2 přístupy zkombinovat. To znamená například seřadit tahy podle toho, kolik přidávají tahů a když přidávají stejně tahů, tak je seřadit podle nejméně používané čáry.

3.2.5.5 Žádné řazení

Nepoužít žádné řazení jako doposud je také možnost. Sice to znamená, že se stavový prostor bude procházet bez jakéhokoliv definovaného pořádku, ale ušetří se čas na řazení. Za stejný čas se prozkoumá vetší část stavového prostoru.

3.2.6 Návrat k dřívějším tahům

Jak bylo ukázáno na obrázku 3.4, tak se zatím používaný algoritmus vrací k dřívějším tahům pomalu. V prohledávaném stromu se dostane do určité hloubky a doba návratu k dřívějším tahům exponenciálně stoupá. Znamená to tedy, že algoritmus u prvních přibližně 40 tahů vyzkouší pouze jednu možnost. Pokud tyto tahy rozehraje nějak nevhodně, tak už není šance nalézt dobré řešení.

Abych zajistil změnu dřívějších tahů, nežli těch několika posledních, tak jsem přidal skoky zpět. Pokud se po určitém počtu iterací nenalezne nové nejlepší řešení, tak se program vrátí k nějakému nastavenému dřívějšímu tahu. Jedna iterace je každé nové rekurzivní volání algoritmu.

Po testování různých možností skoků zpět jsem zvolil 2 možnosti. Jedna skáče zpět častěji a druhá méně často. Obě varianty obsahují 3 různé druhy skoků zpět. Pro každý skok se počítá, kolik iterací proběhlo od posledního stejně velkého skoku. Toto počítadlo se vynuluje pří nalezení nového nejlepšího skóre, nebo při větším skoku. $MaxSkore$ je velikost nejlepší mřížky, kterou program za svůj běh nalezl. Milion iterací trvá přibližně třetinu vteřiny.

Těmito skoky zpět se samozřejmě velké množství stavů hry přeskočí. Neprochází již tak k procházení celého stavového prostoru, ale pouze k jeho prořezávání.

3.2.6.1 Časté skoky zpět

Malý skok: Po milionu iterací se vrátí na tah $(MaxSkore/2)$.

Střední skok: Po pěti milionech iterací se vrátí na tah $(MaxSkore/4)$.

Velký skok: Po dvaceti milionech iterací se vrátí na tah 10.

3.2.6.2 Méně časté skoky zpět

Malý skok: Po třiceti milionech iterací se vrátí na tah $(MaxSkore/2)$.

Střední skok: Po sto milionech iterací se vrátí na tah $(MaxSkore/4)$.

Velký skok: Po pět seti milionech iterací se vrátí na tah 5.

3.2.7 Algoritmus s řazením tahů a skoky zpět

Pro ukázání algoritmu, který řadí tahy a skáče zpět, si nadefinuji nové funkce.

Tahy.serad(): Seřadí tahy podle zvolené řadící funkce.

SkocZpet(): Zkontroluje, jestli již uběhlo dostatečně iterací na nějaký ze skoků a pokud ano, tak se rekurzivně vrátí k tahu definovanému skokem.

Algoritmus 4: DFSRazeniASkoky

Vstup: Počáteční deska *Plocha*, List možných počátečních tahů *Tahy*

```
1  Pokud Tahy.pocet() == 0:
2      Pokud Plocha.skore > maxSkore:
3          ulož Plocha.
4          maxScore = Plocha.skore
5  Jinak:
6      Tahy.serad() // řazení tahů
7      Pro každý Tah z Tahy:
8          noveTahy = Tahy(Tah, konec)
9          Plocha.zahraj(Tah)
10         noveTahy.zahraj(Tah)
11         DFSRazeniASkoky(Plocha, noveTahy)
12         Plocha.smaz()
13         SkocZpet() // skok zpět
```

3.2.8 Testování řazení tahů a skoků zpět

Jednotlivé možnosti řazení tahů a skoků zpět jsem otestoval. Pro každý způsob řazení jsem pustil 3 algoritmy. Jeden bez skoků, jeden s častými skoky a jeden s méně častými skoky.

Dosažené skóre každého algoritmu po 24 hodinách běhu je zobrazeno v grafu 3.7. Zde je testováno celkem 8 způsobů řazení tahů.

1: Žádné řazení.

2: Řazení podle souřadnic.

3: Řazení podle celkových vah a při shodě podle čar.

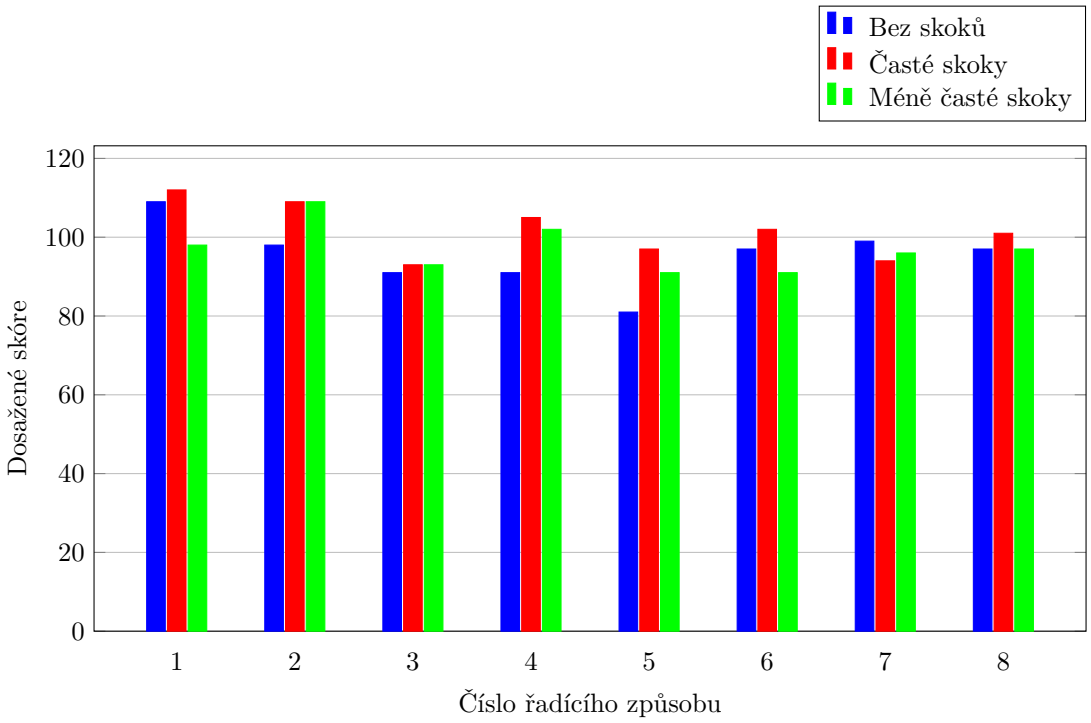
4: Řazení podle celkových vah, hrají se pouze tahy s maximální váhou.

5: Řazení podle čar a při shodě podle celkových vah.

6: Řazení podle bodových vah a při shodě podle čar.

7: Řazení podle bodových vah, hrají se pouze tahy s maximální váhou.

8: Řazení podle čar a při shodě podle bodových vah.



Obrázek 3.7 Porovnání jednotlivých způsobů řazení a skoků zpět.

Časté skoky vedly v sedmi případech z osmi na lepší výsledky, než žádné skoky. Méně časté skoky si vedly podobně, jako žádné skoky. Na základě tohoto testování považuji časté skoky zpět jako poměrně dobrou metodu na prohledávání stavového prostoru.

Nejlepší výsledek našel kupodivu algoritmus, který tahy neřadil vůbec. Našel mřížku velikosti 112 zobrazenou na obrázku B.4. Vysokého výsledku 109 také dosáhlo řazení podle souřadnic.

3.3 Paralelní řešení

Zatím jsem v této kapitole představil návrh a testování algoritmu na hledání co nejlepšího řešení hry morpion solitaire. Tento algoritmus je sekvenční, používá tedy jenom jeden procesor. Já mám k dispozici 64 jádrový počítač. Na plné využití tohoto počítače je potřeba zatím používaný algoritmus paralelizovat.

Algoritmus prohledává strom, ve kterém jsou vrcholy různé stavy hry a hrany jsou jednotlivé tahy. Kořen je počáteční konfigurace bodů a hrany z něj vycházející jsou možné tahy v počáteční konfiguraci.

Algoritmus prohledává tento strom do hloubky a někdy skáče zpět. Hlubší místa podstromů, do kterých vstoupil, prohledává velmi důkladně a díky skokům zpět se nestává, že by se zasekl v nějakém podstromu a už se nikdy nevynořil.

Pro paralelizaci toho algoritmu jsem se rozhodl tento algoritmus spouštět vícekrát tak, aby pokaždé prohledával jinou část tohoto stromu.

Toho docílím tak, že si určím nějakou hloubku h. Jedno hlavní vlákno zahraje prvních h tahů a předá rozehranou hru jinému vláknu. Takto hlavní vlákno pokračuje, dokud jsou volná vlákna. Pokud nejsou k dispozici volná vlákna, tak čeká na uvolnění nějakého vlákna. Vlákna, která nějakou delší dobu nenaleznou dobrý výsledek, budou ukončena.

Zmíněná hloubka h je tedy hloubka zanoření hlavního vlákna. V dalším textu bude označována jako *hloubka hledání hlavního vlákna* nebo pouze *hloubka*.

3.3.1 Ukončování vláken

Vlákna, která po určitém počtu iterací nenaleznou určitý výsledek, budou končena, aby se uvolnilo místo na nová vlákna. Ukončování vláken jsem nastavil podle tabulky 3.2. Neznamená to však, že vlákna, která naleznou skóre 140 nebo vyšší poběží do nekonečna. Díky skokům zpět budou i tato vlákna časem ukončena.

Skóre	Počet milionů iterací
80	10
90	50
110	500
120	1000
130	5000
140	100000

Tabulka 3.2 Počet iterací po kterém, když nebude dosaženo dané skóre, bude vlákno ukončeno.

3.3.2 Paralelní algoritmus

Paralelní algoritmus se skládá ze dvou částí. Parametrem se nastaví počet vláken *vlaknaG* a hloubka zanoření hlavního vlákna *hloubka*. Hlavní vlákno postupně prochází všechny kombinace prvních *hloubka* tahů a tyto rozehrané hry předává vedlejším vláknům. Ostatních vláken může být až $vlaknaG - 1$. Proměnné *vlaknaG* a *maxScoreG* jsou atomické globální proměnné, které sdílejí všechny procesy. Přidal jsem jednu novou funkci, která ukončuje dlouho běžící vlákna.

ZkontrolujVlakno(): Podle tabulky 3.2 rozhodne, zda vlákno má být ukončeno. Pokud ano, tak ho ukončí a zvýší *vlaknaG* o 1.

Procedura VedlejsiVlakno

Vstup: Herní deska *Plocha* ve stavu rozehrané hry, List možných tahů *Tahy*

```
1  ZkontrolujVlakno()
2  Pokud Tahy.pocet() == 0:
3      Pokud Plocha.skore > maxScoreG:
4          ulož Plocha.
5          maxScoreG = Plocha.skore
6  Jinak:
7      Tahy.serad()
8      Pro každý Tah z Tahy:
9          noveTahy = Tahy(Tah, konec)
10         Plocha.zahraj(Tah)
11         noveTahy.zahraj(Tah)
12         VedlejsiVlakno(Plocha, noveTahy)
13         Plocha.smaz()
14         SkocZpet()
```

Algoritmus 5: HlavniVlakno

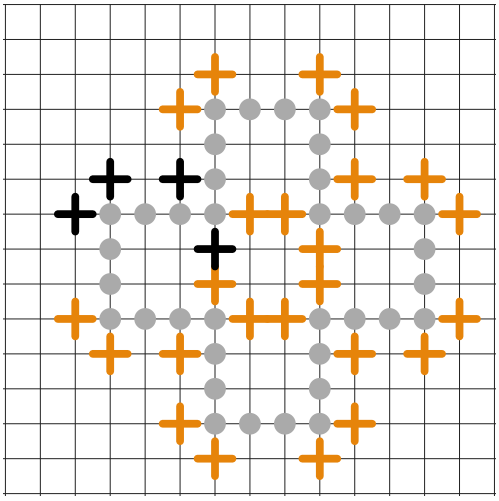
Vstup: Počáteční deska *Plocha*, List možných počátečních tahů *Tahy*

```
1  Pokud Tahy.pocet() == 0:
2      return
3  Pokud Plocha.skore == hloubka:
4      Pokud vlaknaG == 0, tak čekej
5      vlaknaG = vlaknaG − 1
6      VedlejsiVlakno(Plocha, Tahy)
7  Jinak:
8      Tahy.serad()
9      Pro každý Tah z Tahy:
10         noveTahy = Tahy(Tah, konec)
11         Plocha.zahraj(Tah)
12         noveTahy.zahraj(Tah)
13         HlavniVlakno(Plocha, noveTahy)
14         Plocha.smaz()
```

Procedura VedlejsiVlakno je téměř identická jako algoritmus DFSRazeniASkoky, změna je pouze v nové funkci na ukončení vlákna.

3.3.3 Omezení opakovaní podobných mřížek

V kapitole 1.5.1 je ukázáno, že všechny možné počáteční tahy se dají rozdělit do čtyř skupin, kde v každé skupině všechny tahy vedou na podobnou mřížku. Abych omezil opakování podobných mřížek, tak jsem nastavil, že jako první tah se bude zkoušet maximálně jeden tah z každé ze čtyř skupin. Vybrané tahy jsou zobrazené na obrázku 3.8. Tato skutečnost není zaznamenána v pseudokódu, aby nebyl komplikovaný.



Obrázek 3.8 Černě jsou označeny 4 tahy, které nevedou na podobné mřížky.

3.3.4 Testování paralelního algoritmu

Pro první otestování jsem spustil 2 instance tohoto algoritmu, každé jsem přiřadil 20 vláken. K řazení tahů jsem vybral dvě zatím nejúspěšnější metody. První metoda je bez řazení tahů a druhá je řazení tahů podle souřadnic. Hloubku prohledávání hlavního vlákna jsem nastavil na 5.

Algoritmus bez řazení tahů našel mřížku velikosti 123. Mřížka je zobrazena na obrázku B.5.

Algoritmus s řazením tahů pomocí souřadnic našel mřížku velikosti 130. Mřížka je zobrazena na obrázku B.6.

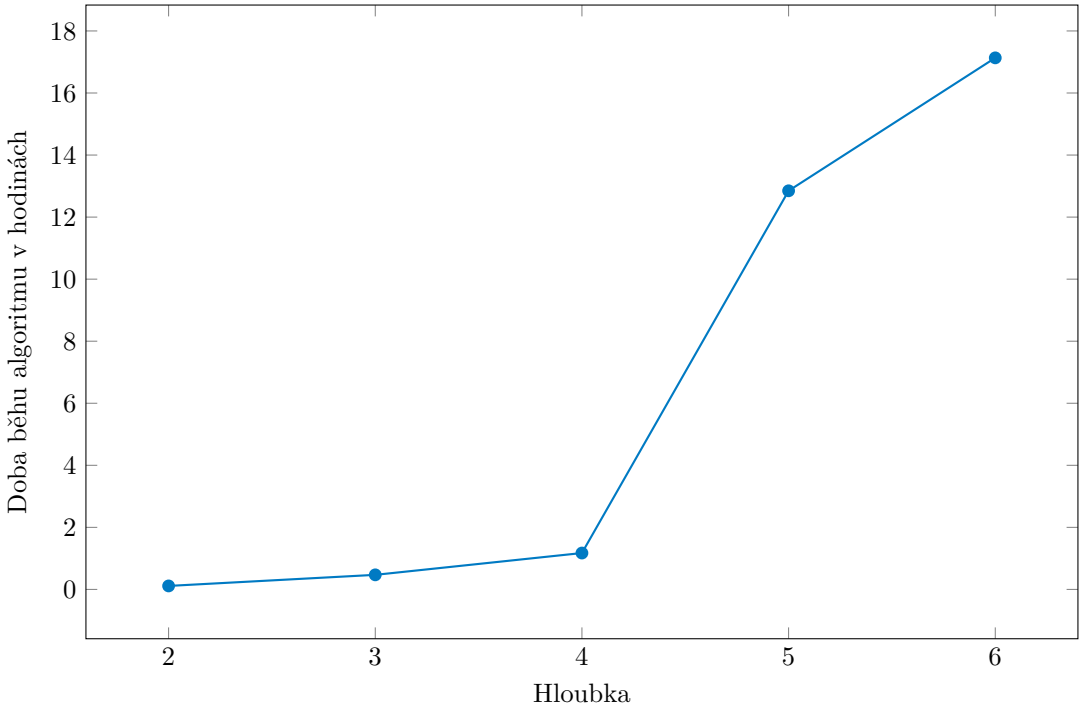
I na poměrně malé hloubce prohledávání hlavního vlákna přinesla paralelizace výrazné zlepšení výsledků.

3.3.4.1 Testování parametru hloubka

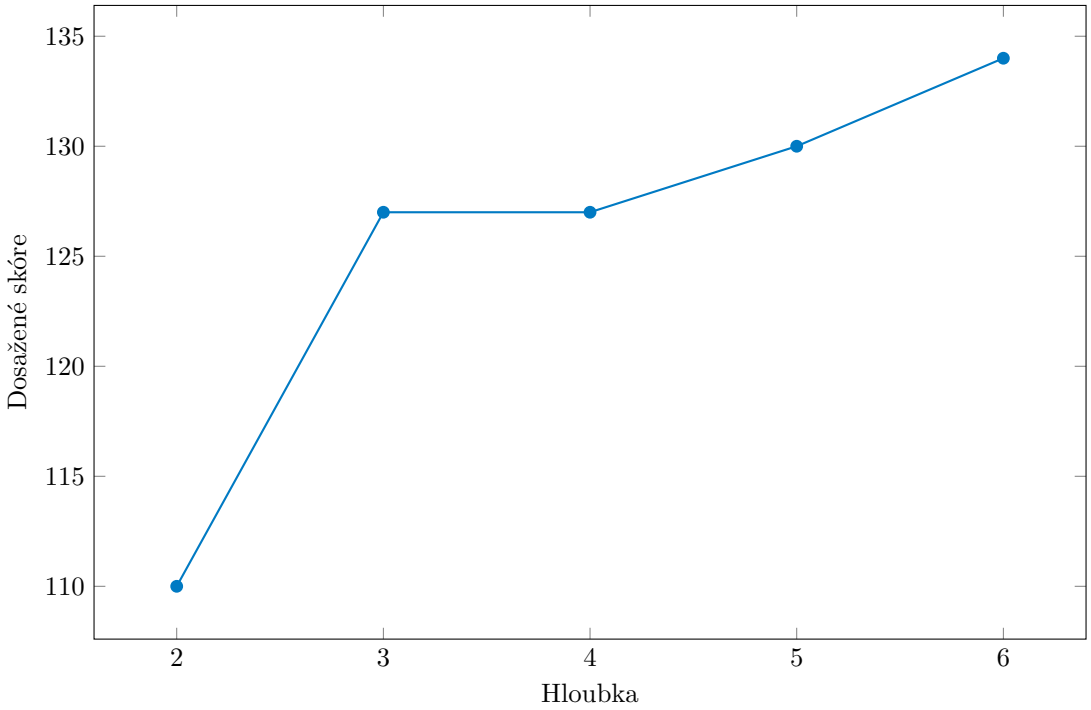
Pro otestování parametru hloubka jsem spustil paralelní algoritmus s řazením tahů podle souřadnic na 64 vláknech. Spustil jsem ho pro hloubky v rozmezí 2 až 6 a zaznamenal jsem, jak dlouho každý běh trval a jaké našel nejlepší skóre. Graf 3.9 zobrazuje vliv parametru hloubka na dobu běhu a graf 3.10 zobrazuje vliv parametru hloubka na nejlepší nalezené skóre.

Doba běhu roste s hloubkou poměrně rychle. Doba běhu také závisí na nejlepším nalezeném skóre. Velký skok mezi hloubkami 4 a 5 je způsoben tím, že při hloubce 5 našel algoritmus skóre 130 a vlákno, které toto skóre našlo běželo velmi dlouho, protože podle tabulky 3.2 bylo ukončeno výrazně později, než vlákna které našla skóre menší než 130.

Skóre roste s hloubkou pomaleji. Mezi hloubkami 2 a 3 je velký skok, ale pak už jsou skoky menší a někdy i žádné.



Obrázek 3.9 Závislost doby běhu na hloubce.



Obrázek 3.10 Závislost nalezeného skóre na hloubce.

3.4 Použití mého programu

3.4.1 Kompilace

Pro kompilaci stačí spustit příkaz `make compile`. Soubor main.cpp se zkompiluje do spustitelného souboru main.

3.4.2 Spuštění

Při spouštění je možné nastavit několik parametrů. Spuštění s parametry může vypadat následovně.

```
./main --threads 10 --main-depth 3 --directory vystup --sort weights_lines
```

3.4.3 Parametry

Parametry je možné zadat v libovolném pořadí.

3.4.3.1 threads

Parametr `threads` nastavuje počet používaných vláken. Jedno z těchto vláken je hlavní vlákno a zbytek jsou vedlejší vlákna, Z toho důvodu je minimální možný počet vláken 2. Při vynechání tohoto parametru se počet vláken automaticky nastaví na 64. Takto vypadá spuštění programu s 20 vlákny.

```
./main --threads 20
```

3.4.3.2 main-depth

Parametr `main-depth` nastavuje hloubku hledání hlavního vlákna. Minimální možná nastavitelná hloubka je 2. Při vynechání tohoto parametru se hloubka hledání nastaví na 5. Takto vypadá spuštění programu s hloubkou hledání 8.

```
./main --main-depth 8
```

3.4.3.3 init-file

Při použití parametru `init-file` program nezačíná hledat ze základní konfigurace hry, ale začíná z nějaké rozehrané hry. Tomuto parametru se ještě musí předat název souboru, ve kterém je uložené nějaké řešení a také počet tahů, kolik se jich má z tohoto řešení převzít. Po zavolání následujícího příkazu začne prohledávání mřížkou, která vznikne zahráním prvních 20 tahů ze souboru `record.txt`.

```
./main --init-file record.txt 20
```

3.4.3.4 directory

Parametrem `directory` nebo `dir` se určí adresář, kam se budou ukládat nalezené řešení. Tento adresář musí existovat. Při vynechání tohoto parametru se adresář automaticky nastaví na `output`. Takto vypadá spuštění programu s přesměrováním výstupu do adresáře `vystup`.

```
./main --directory vystup
```

3.4.3.5 sort

Parametr `sort` nastavuje způsob řazení tahů. Způsoby jsou představené v kapitole 3.2.5. Pomocí tohoto parametru se dá zvolit jeden z následujících čtyř způsobů řazení.

`no`	Tahy nejsou nijak seřazeny.
`weights_lines`	Tahy jsou seřazeny podle bodových vah a dále podle počtu čar.
`lines_weights`	Tahy jsou seřazeny podle počtu čar a dále podle bodových vah.
`coords`	Tahy jsou seřazeny podle souřadnic.

Při vynechání tohoto argumentu se použije řazení podle souřadnic. Takto vypadá spuštění programu bez řazení tahů.

```
./main --sort no
```

3.4.3.6 jumps-back

Parametrem `jumps-back` se zapínají skoky zpět vysvětlené v kapitole 3.2.6. Argument `frequent` nastaví časté skoky zpět, `nonfrequent` nastaví méně časté skoky zpět a `no` vypne skoky zpět. Při vynechání tohoto argumentu jsou automaticky nastavené časté skoky zpět. Takto vypadá spuštění programu s méně častými skoky zpět.

```
./main --jumps-back nonfrequent
```

3.4.4 Výstup

Všechny soubory se ukládají do adresáře nastaveného pomocí parametru `directory`.

3.4.4.1 Řešení

Jednotlivá řešení se ukládají ve formátu ukázaném v kapitole 3.1.2. Pokaždé, když program nalezne nové největší skóre, tak ho uloží do souboru `<početTahů>.txt`. Pokud se tedy jedná o skóre 130 tahů, tak bude uložené v souboru `130.txt`. kromě jednotlivých tahů je také v souboru uložen čas, jak dlouho hledání tohoto řešení ve vteřinách trvalo a počet iterací, kolik vlákno, které ho našlo, před tím udělalo.

3.4.4.2 Postup hlavního vlákna

Do souboru `main_progress.txt` se ukládá postup hlavního vlákna. Formát souboru vypadá takto.
číslo tahu: počet tahů k vyzkoušení / celkový počet tahů
Při hloubce hledání 5 může soubor vypadat následovně.
0: 4/4
1: 27/27
2: 26/26
3: 20/25
4: 14/19
Pro každou úroveň zanoření hlavního vlákna se vypisuje kolik tahů se ještě bude procházet a kolik tahů se procházelo dohromady. To slouží k odhadnutí času běhu programu.

3.4.4.3 Bežící vlákna

Do souboru `running_threads.txt` se zaznamenává, jaká vlákna běží, kolik provedla iterací a jaké našla maximální skóre. Hlavní vlákno se do tohoto výpisu nezahrnuje. Do tohoto souboru se také zaznamenává, jak dlouho již program běží.

K aktualizaci souborů `running_threads.txt` a `main_progress.txt` dochází pouze, když jsou využívána všechna vlákna a hlavní vlákno čeká.

3.4.4.4 Standardní výstup

Na standardní výstup se vypisuje vytvoření každého nového vlákna a ukončení vlákna. Při nalezení nového nejlepšího skóre se mřížka vykreslí na standardní výstup. Takto vypadá mřížka lidského rekordu zobrazená na obrázku A.1 vypsaná na standardní výstup.

```
                  O     O
                   \   /|\
                O-O-O-O-O-O-O-O-O
                |X|X|X|X|X|/|X|/|\
                O O-O-O-O-O-O-O-O-O
               /|X|X|X|X|X|X|X|X|\
             O-O-O-O-O-O-O-O-O-O-O-O-O
             |X|X|/|X|X|X|X|X|X|X|X|\
             O-O-O-O-O-O-O-O-O O-O-O-O-O
            /|X|X|X|X|X|X|X X|X|X|X|X|\
          O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O
          |\|\|X|X|X|\|X|X X|X|X|/|X|X|X|/|
          O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O
          |\|X|X|X|X|X|X|  |X|X|X|X|X|X|/|
          O O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O
          |/|X|X|X|X|X|X|  |X|X|X|X|X|X|\|/
          O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O
          |/|/|X|X|X|/|X|X|\|\|X|X|X|X|\|X|
          O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O O
           \|X|\|X|X|X|X|X|X|X|X|X|X|X|X|\|
            O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O
            |\|\|X|X|X|X|X|X|X|X|X|X|X|X|/|
            O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O
               \|\|X|X|X|X|X|X|X|\|X|X|X|X|
                O-O-O-O-O-O-O-O-O-O-O-O-O O O
                |\|X|X|X|X|X|X|X|\|/|/|X|X|\|
                O-O-O-O-O O-O-O-O-O-O-O-O-O O
                         \|/     \|X|/|/
                          O       O O O
                                   \|X|
                                    O O
                                      |
                                      O
```

3.5 Nejlepší nalezená řešení

Při hledání co nejlepšího řešení jsem opakovaně spouštěl svůj program na 64 vláknech s různými parametry `jumps-back`, `main-depth` a `sort`. Když jsem nalezl řešení větší než 135, tak jsem pomocí argumentu `init-file` spustil program, který z toho řešení převzal prvních několik tahů, abych více zaměřil prohledávání stavového prostoru na okolí nalezeného řešení.

Třemi různými způsoby se mi podařilo nalézt tři různé mřížky velikosti 140.

I přes fakt, že podle testu v kapitole 3.2.8 řazení podle bodových vah nalezlo oproti ostatním řazením podprůměrné skóre, tak se mi pomocí tohoto řazení podařilo dosáhnout dobrých výsledků.

Pomocí řazení tahů podle bodových vah a dále podle čar při hloubce hledání 8 našel algoritmus mřížku velikosti 136 tahů. Tuto mřížku nalezl po 82 hodinách běhu. Dále se stejnými parametry a startu z konfigurace prvních 30 tahů nalezeného řešení, našel algoritmus mřížku velikosti 140 zobrazenou na obrázku B.7. Toto zlepšení o 4 tahy ukazuje, že může dojít ke zlepšení nějakého řešení pomocí prohledávání jeho okolí ve stavovém prostoru. Při použití jiného řazení tahů a startu ze stejné konfigurace se však lepší řešení než 136 nenašlo.

Dále jsem experimentoval s větší hloubkou hledání. Větší hloubka hledání však znamená výrazně delší dobu běhu. Z tohoto důvodu jsem nastavil 10krát rychlejší ukončování vláken, než je v tabulce 3.2. Tato změna vedla ve většině případů na horší výsledky, ale jednou se mi podařilo nalézt mřížku velikosti 140. Toho bylo docíleno stejným řazením tahů jako v prvním případě, pomocí bodových vah a dále čar. Hloubka hledání byla nastavena na 12. Nalezení této mřížky trvalo 152 hodin. Tato mřížka je zobrazena na obrázku B.8. V tomto případě nové hledání z konfigurace několika prvních tahů lepší řešení nenašlo.

Stejně velkou mřížku se mi podařilo nalézt ještě jednou a to algoritmem s řazením tahů podle souřadnic a hloubkou hledání 8. Tato mřížka byla nalezena již po 10 hodinách. Ani v tomto případě hledání z konfigurace prvních několika tahů této mřížky lepší řešení nenašlo.

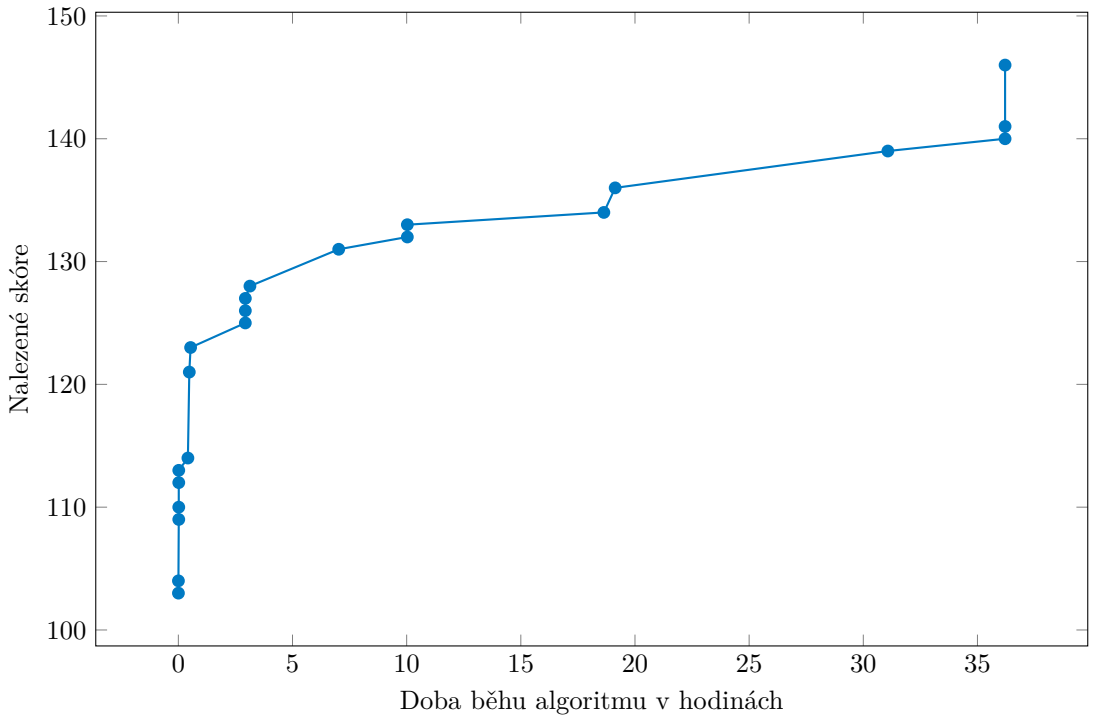
Nalezení mřížky velikosti 140 s řazením tahů pomocí souřadnic bylo nejrychlejší, tudíž jsem spustil algoritmus znovu, tentokrát s hloubkou 9. Tento algoritmus po 36 hodinách našel mřížku velikosti 146 zobrazenou na obrázku B.10.

Na grafu 3.11 je zobrazeno, po jak dlouhé době tento algoritmus nalezl jaké skóre. Zobrazena jsou data pouze pro skóre větší než 100. Kromě skoků na místech, kde program našel několik nových nejlepších řešení hned po sobě, připomíná křivka logaritmickou křivku což naznačuje exponenciální složitost hledání větších řešení.

Spuštěním programu z konfigurace prvních 28 tahů zmíněného řešení, hloubkou hledání 3 a řazením tahů také podle souřadnic se po necelé půl hodině podařilo nalézt o 2 větší mřížku. Zmíněné mřížka velikosti 148 je zobrazena na obrázku B.11. Mřížky B.10 a B.11 jsou velmi podobné. Prvních 29 tahů je úplně stejných.

Řazení podle souřadnic preferuje tahy vlevo a při více možnostech tahy nahoře. Zajímavé je, že žádná ze 4 čtyř mřížek nalezených s řazením tahů pomocí souřadnic ukázaných v této práci nesměřuje směrem doleva. Může to být způsobeno tím, že brzké hraní tahů vlevo napomáhá k rozvíjení mřížky jiným směrem v pozdější fázi hry.

Mřížka velikosti 148 je největší, kterou se mi podařilo nalézt. Pro vyzkoušení co největšího množství různých parametrů programu jsem nechával jednotlivé programy pouze několik dnů. Věřím, že při delším běhu a větším parametru hloubka bych nalezl větší mřížky.



Obrázek 3.11 Závislost nalezeného skóre na době běhu.

Závěr

V této práci jsem se zabýval hrou morpion solitaire. Vysvětlil jsem pravidla a popsal historii této hry. Ukázal jsem historický vývoj nejlepších řešení, kterých bylo dosaženo jak hraním hry na papír, tak pomocí počítače. U počítačových řešení jsem popsal, jakými algoritmy jich bylo dosaženo.

Dále jsem navrhl sekvenční algoritmus, který prořezává stavový prostor hry morpion solitaire a hledá její řešení. U návrhu algoritmu jsem se rozhodl vydat jiným směrem, než moji předchůdci. Algoritmy, kterými bylo dosaženo zatím nejlepších řešení, byly otestovány na výrazně výkonnějších výpočetních systémech, než mám k dispozici. Usoudil jsem, že bych modifikací zmíněných algoritmů nepřinesl nic nového. Z toho důvodu jsem navrhl vlastní program, který se na těch dřívějších nijak nezakládá. Tento algoritmus jsem následně paralelizoval.

Program se spouští v terminálovém prostředí. Může startovat ze základní konfigurace bodů, nebo z nějaké konfigurace již rozehrané hry. Všechna nová nalezená nejlepší řešení ukládá do souborů.

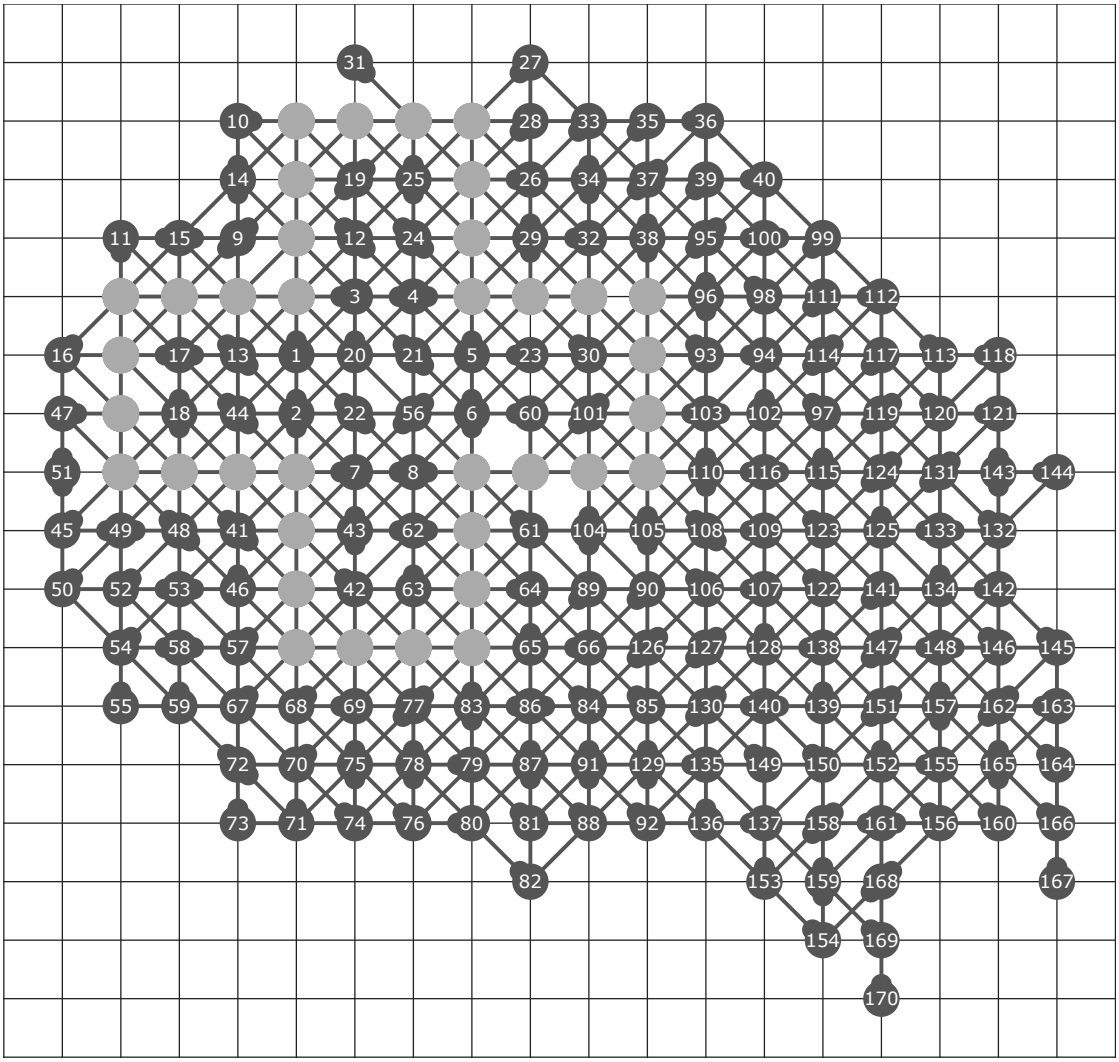
Pomocí testování mého programu se mi podařilo nalézt řešení velikosti 148. Tento výsledek nejlepšího řešení 178 zdaleka nedosahuje. Podařilo se mi však překonat řešení, které našel pomocí svého programu japonský student Haruhiko Akiyama. Jeho řešení se skládá ze 146 tahů. Dříve, než Christopher D. Rosin objevil pomocí svého programu aktuální rekord, se jednalo o nejlepší řešení nalezené počítačem.

Pro otestování co nejvíce různých konfigurací programu jsem jednotlivé programy nechával běžet pouze několik hodin nebo dnů. Vzhledem k tomu, že se při zvětšování hloubky hledání, se zvětšuje nalezená mřížka, si myslím, že při delším běhu programu se mi podaří nalézt větší mřížku.

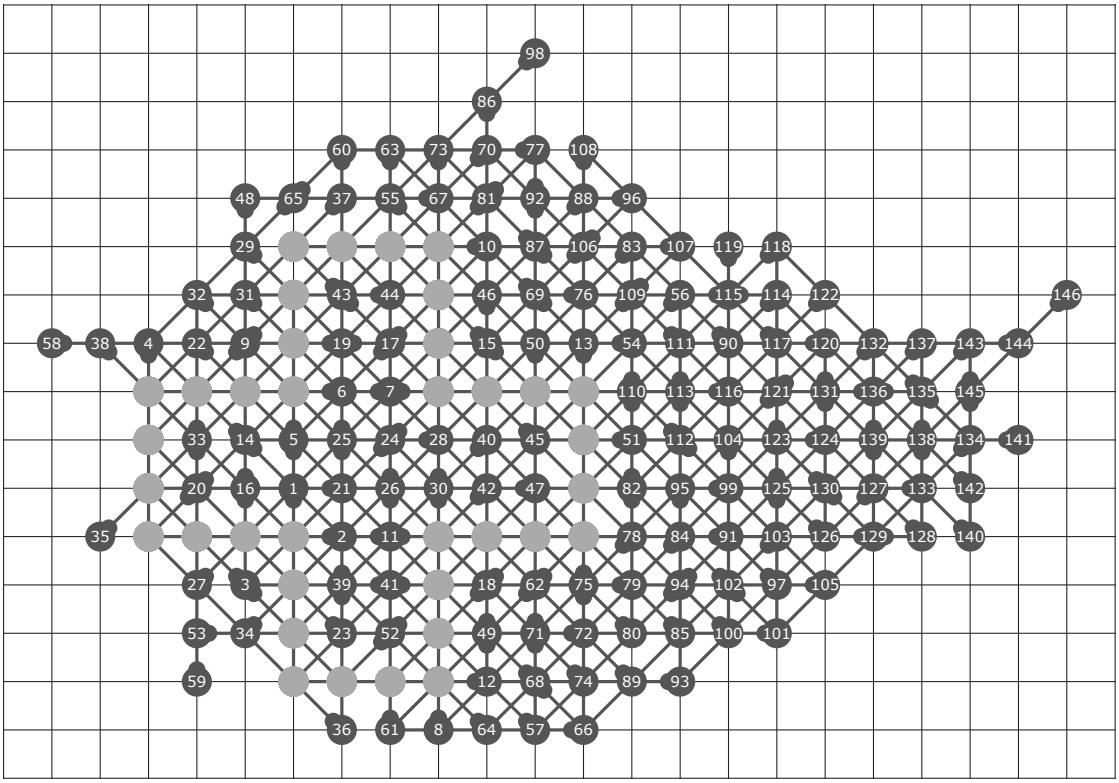
Mimo morpion solitaire existuje mnoho jiných problémů, kde dochází k prohledávání velkého, velice nepravidelného stavového prostoru. Navržený algoritmus by mohl mít využití i na jiné problémy.

Příloha A

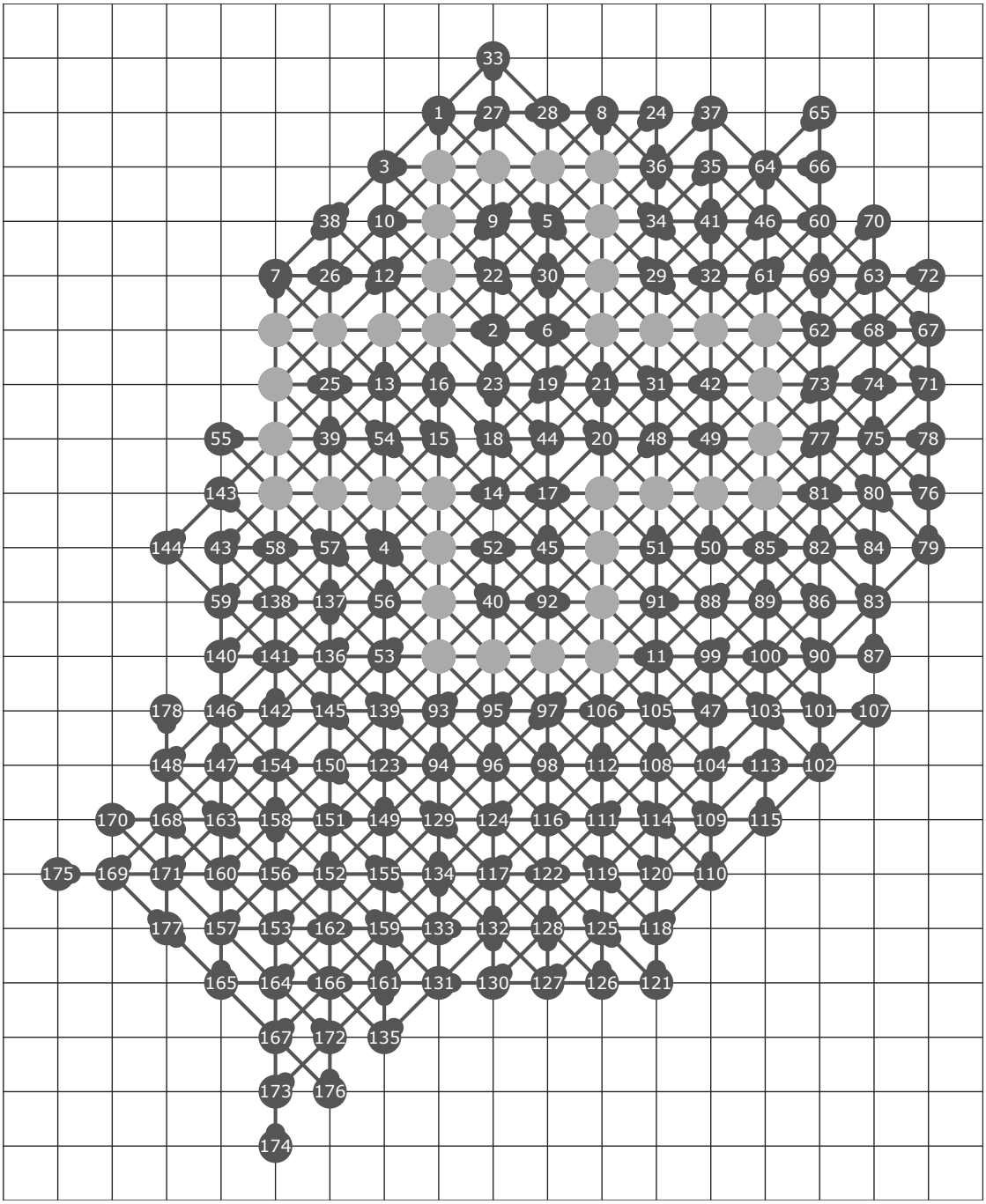
Nejlepší řešení



Obrázek A.1 170 tahů. Lidský rekord, který objevil roku 1976 Charles-Henri Bruneau.



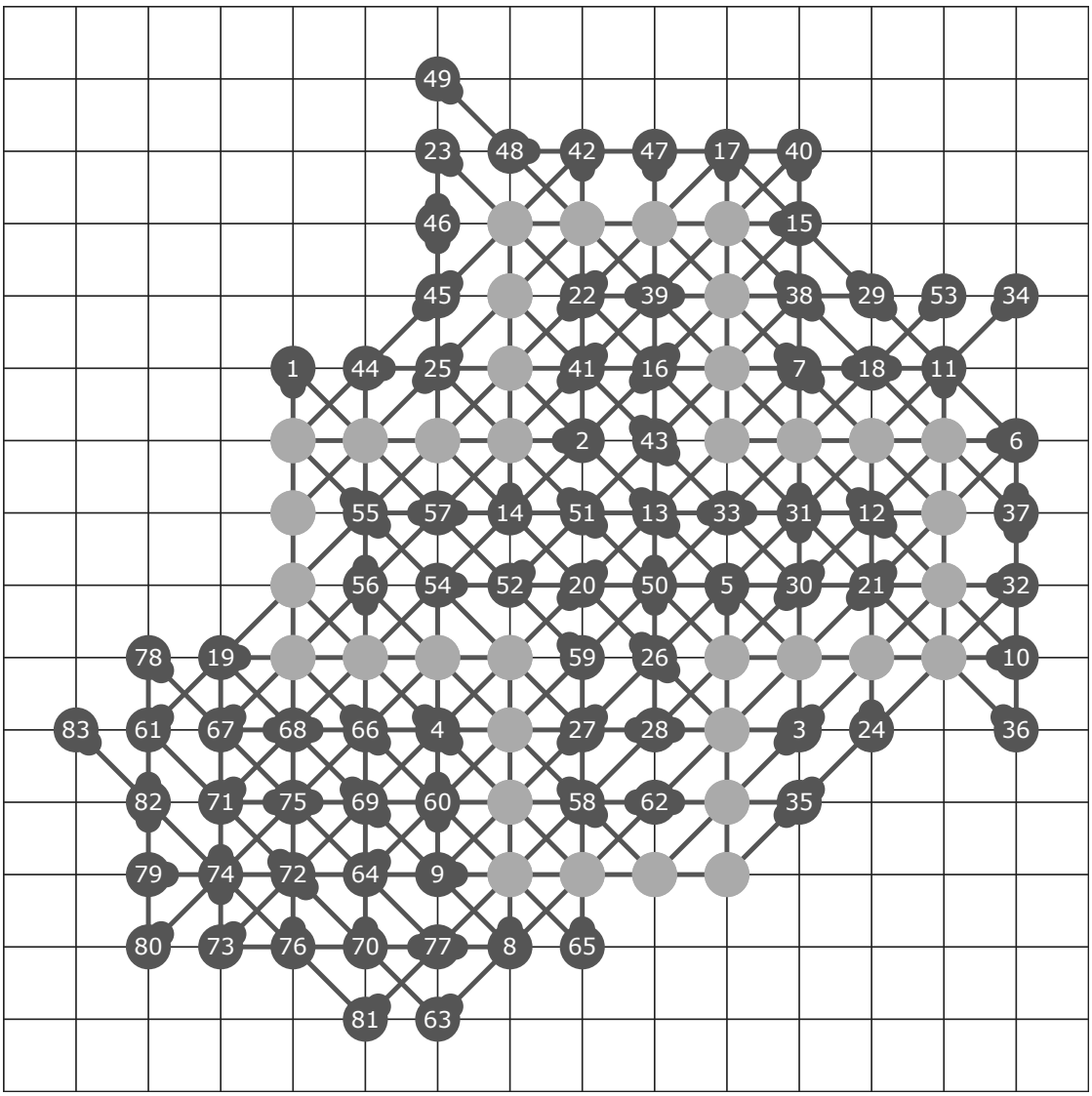
Obrázek A.2 146 tahů. Tuto mřížku nalezl pomocí počítače japonský student Haruhiko Akiyama.



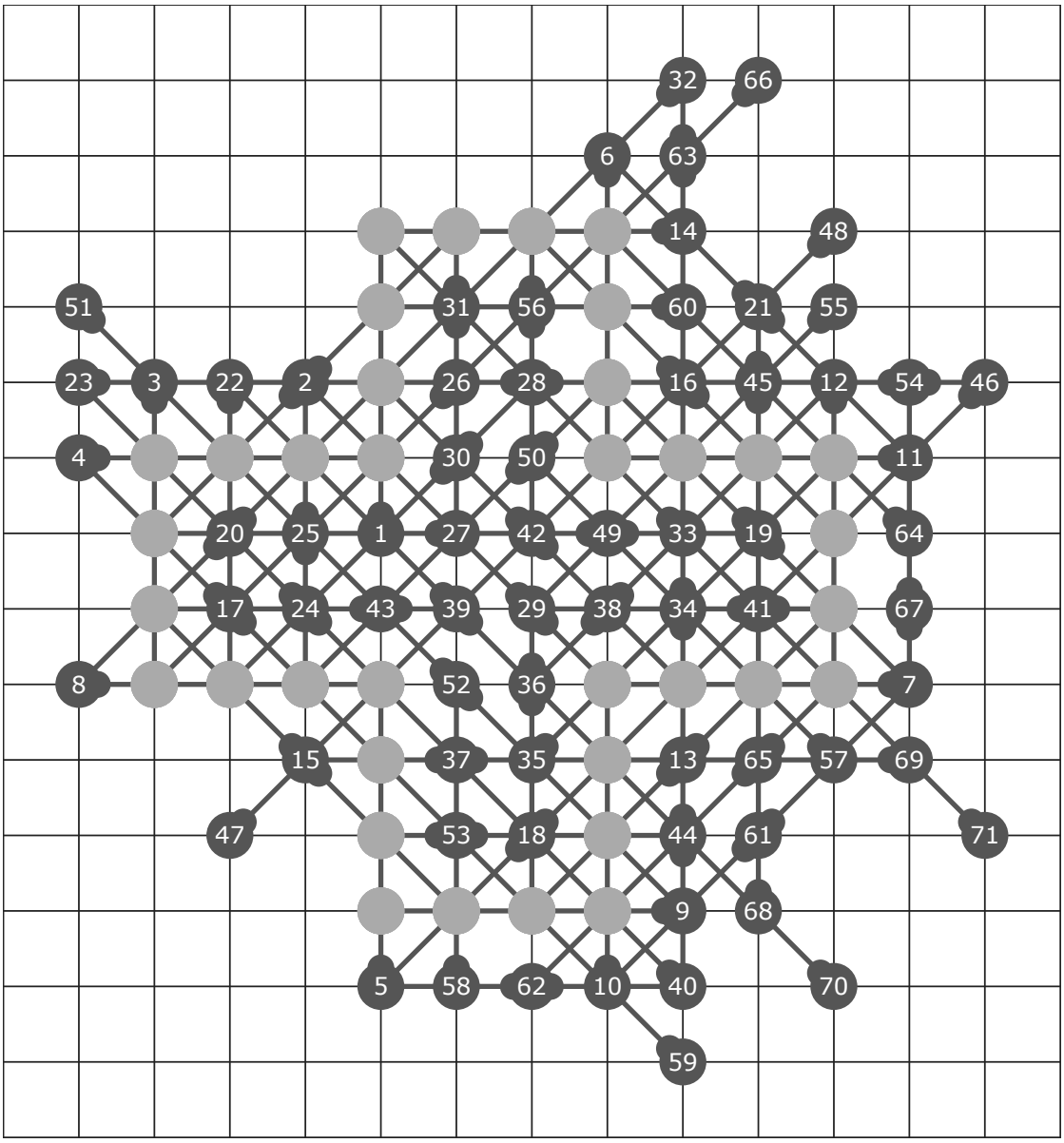
Obrázek A.3 178 tahů. Zatím nepřekonaný rekord, který našel pomocí počítače Christopher D. Rosin.

Příloha B

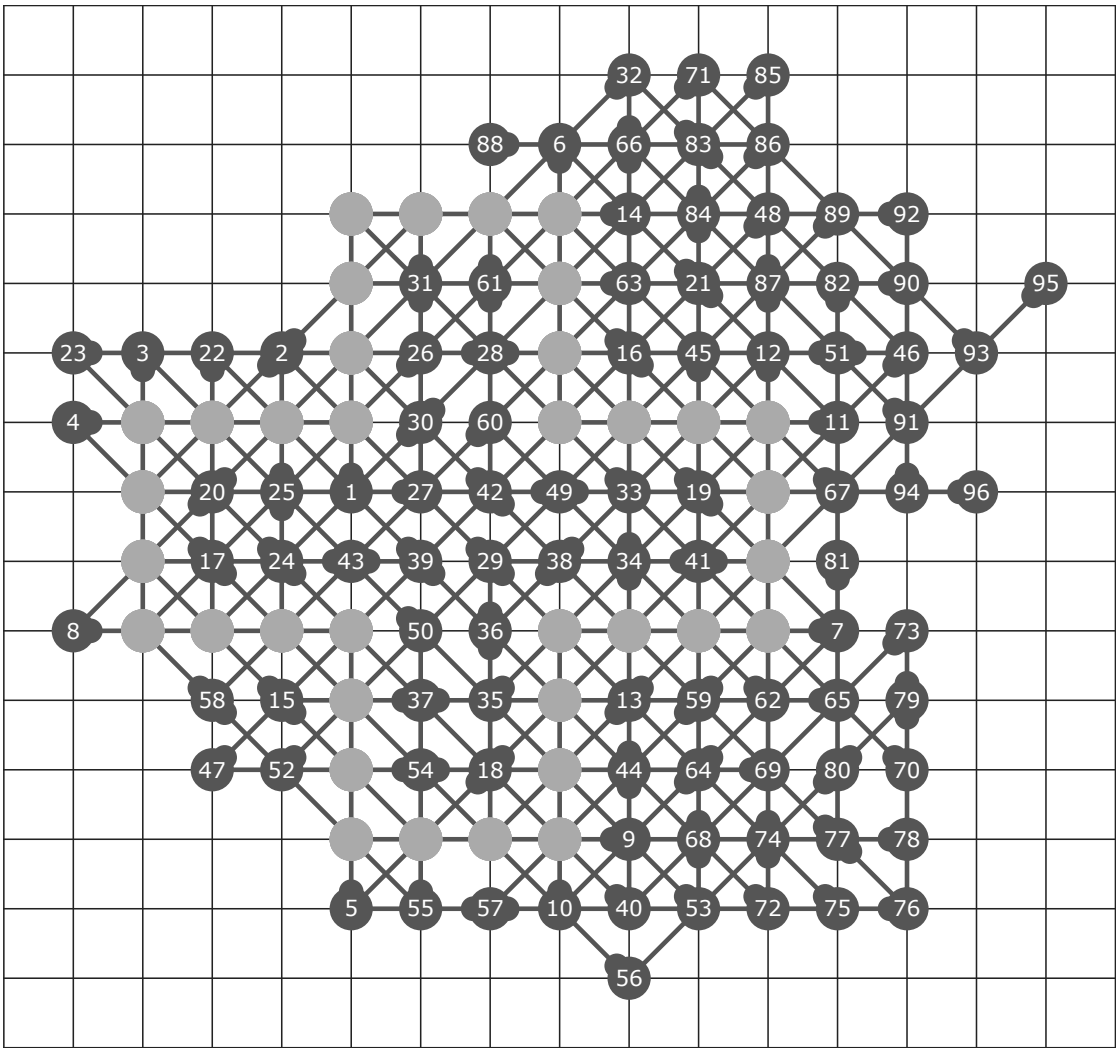
Moje řešení



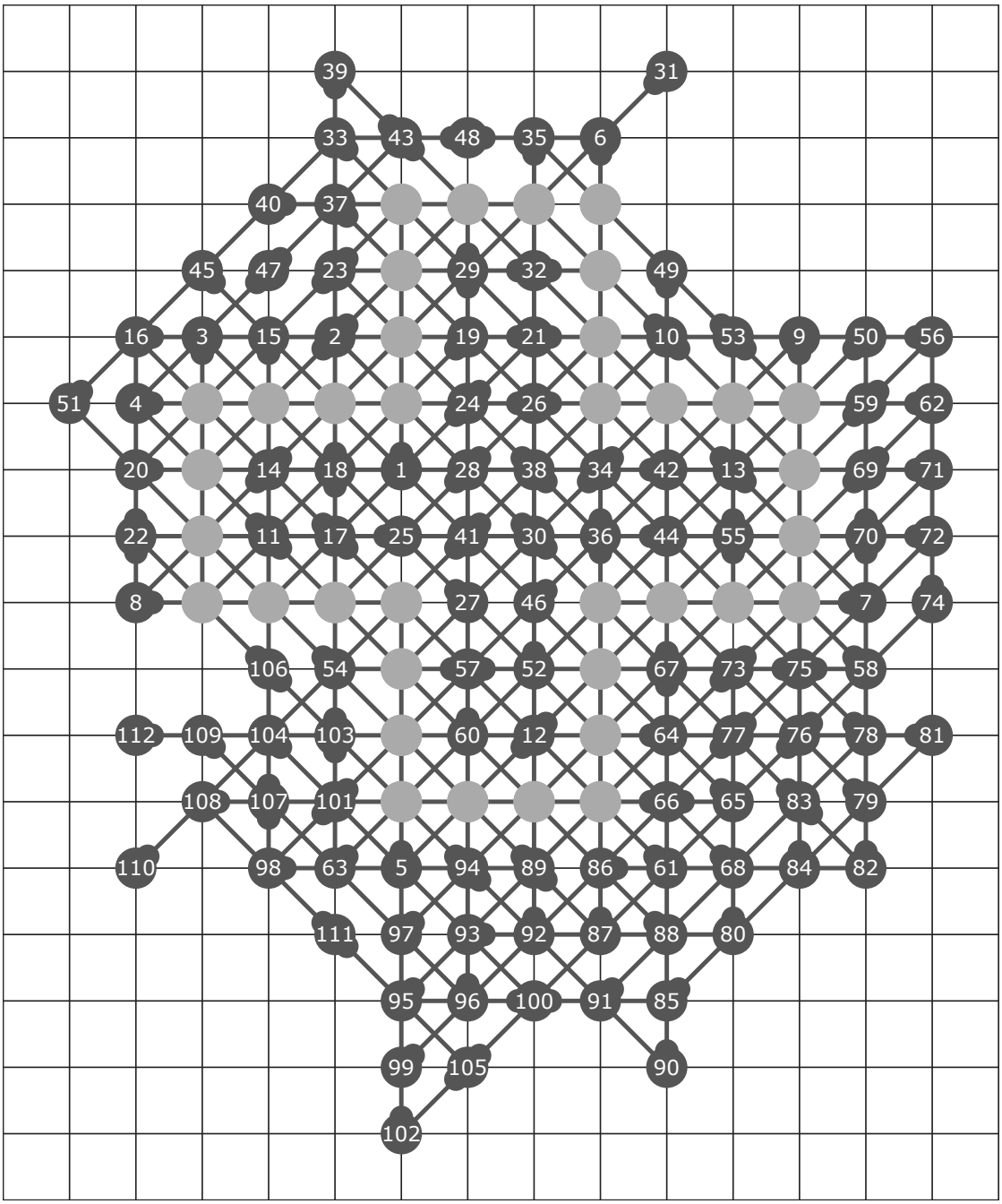
Obrázek B.1 83 tahů nalezených algoritmem náhodných tahů.



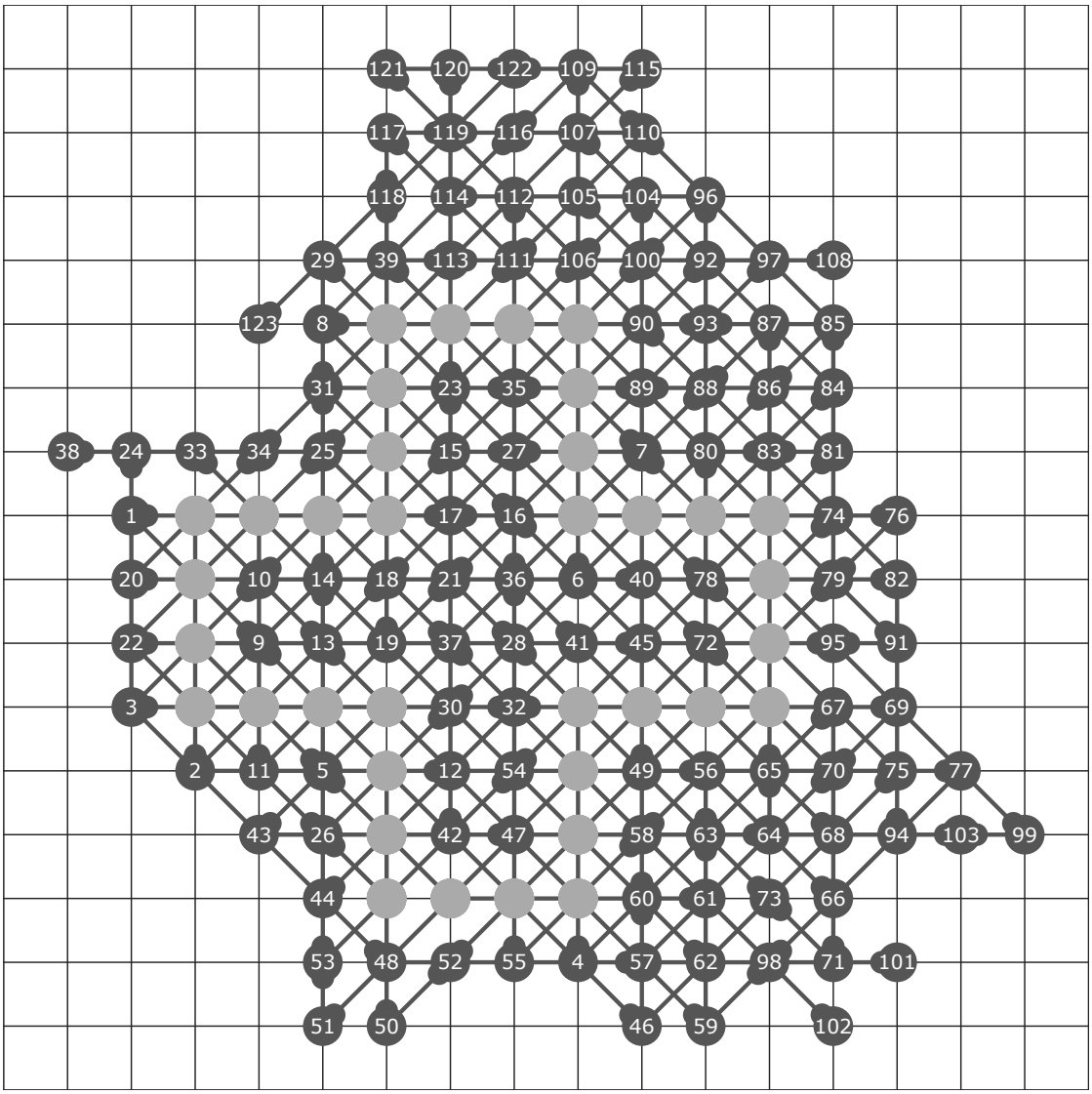
Obrázek B.2 71 tahů nalezených algoritmem DFS.



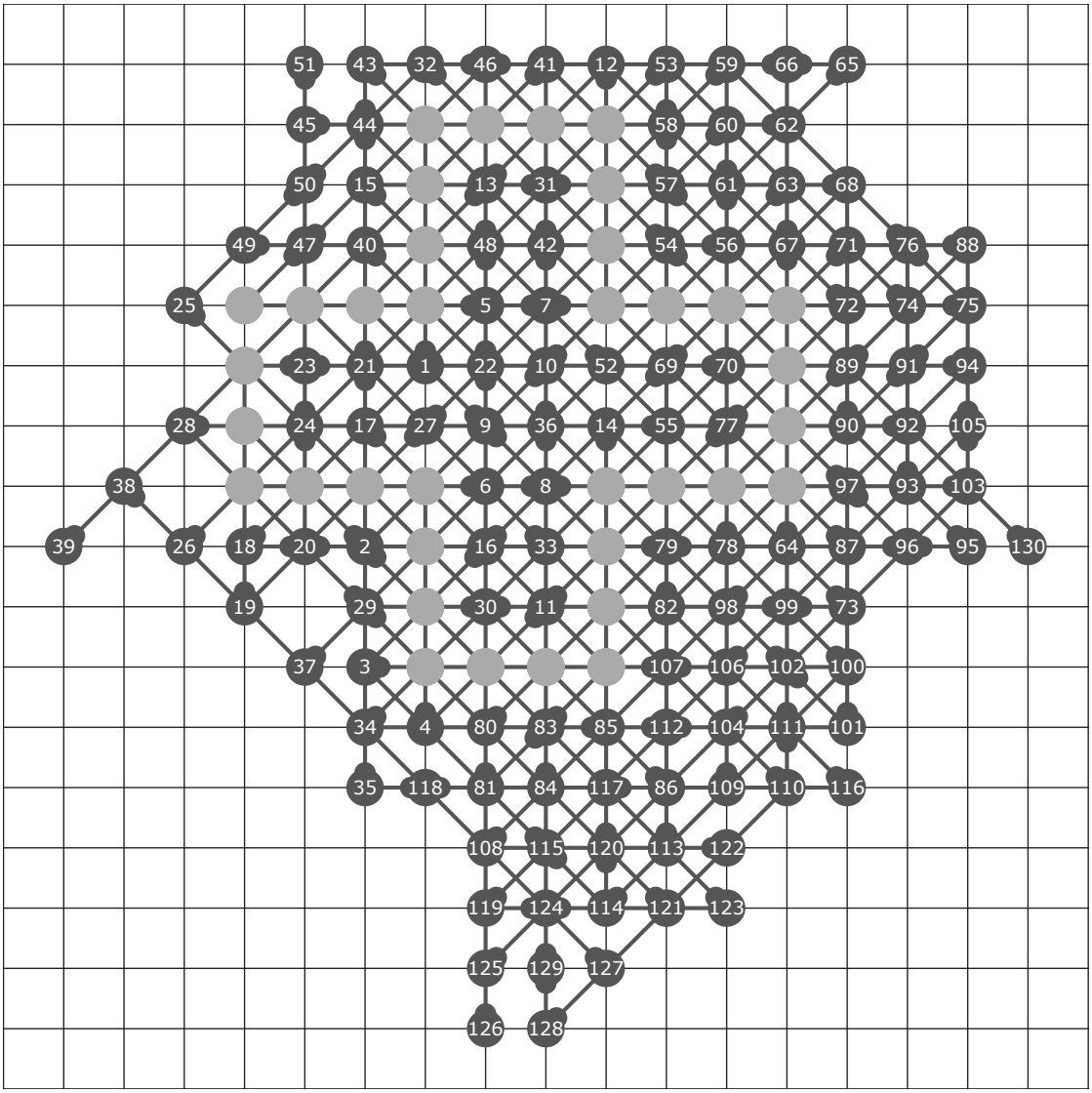
Obrázek B.3 96 tahů nalezených algoritmem DFSbezOpakovani.



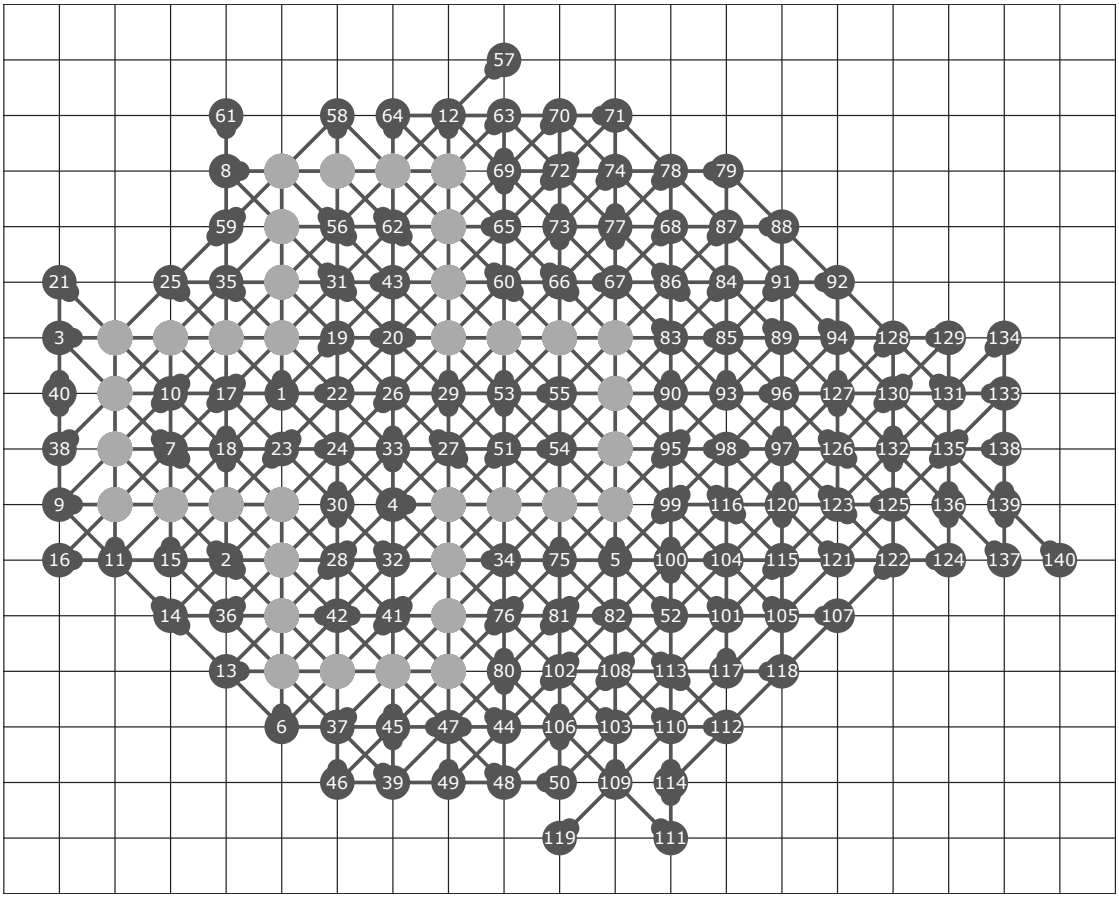
Obrázek B.4 112 tahů nalezených algoritmem bez řazení tahů a s častými skoky zpět



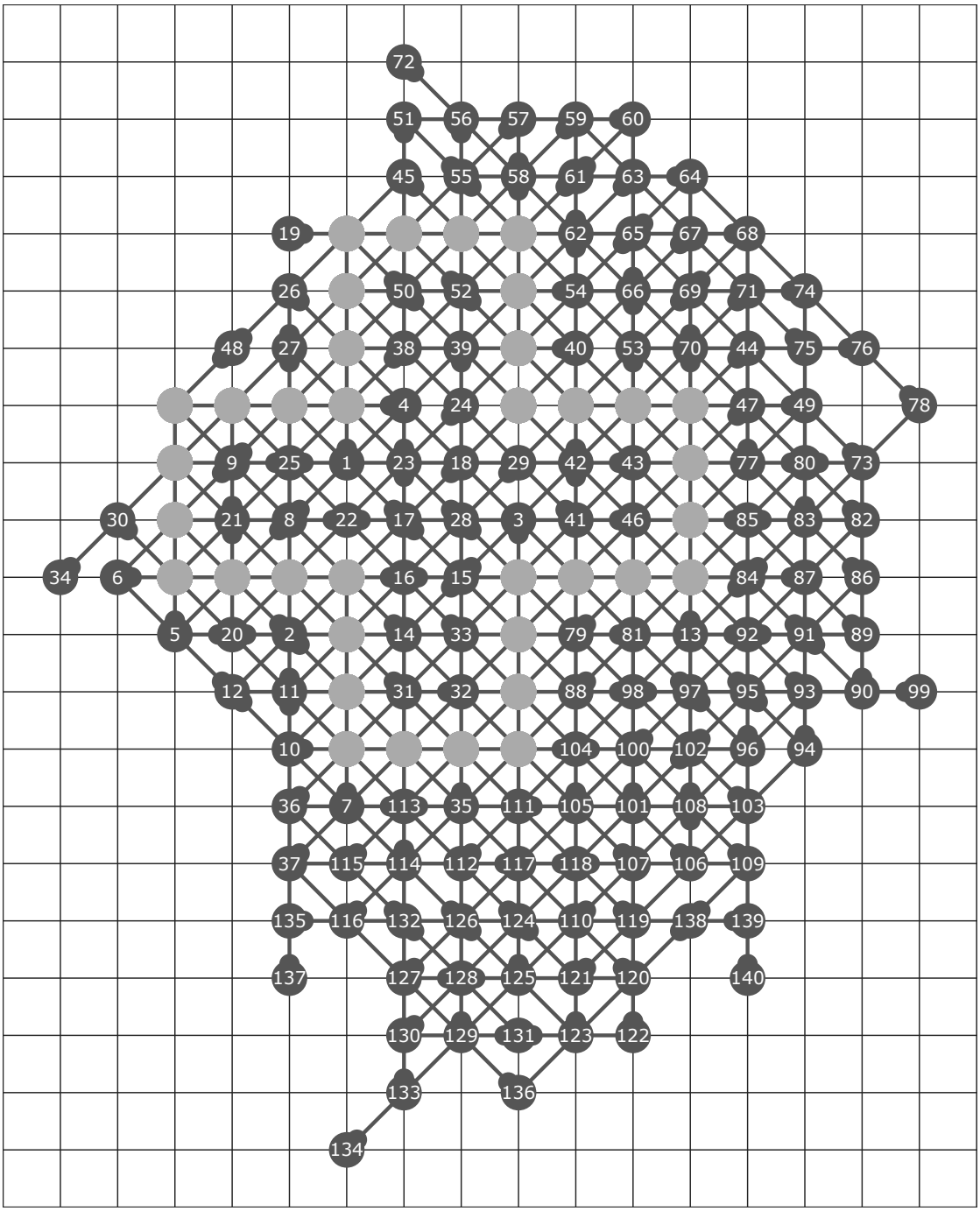
Obrázek B.5 123 tahů nalezených paralelním algoritmem s hloubkou hledání 5, 20 vlákny a bez řazení tahů.



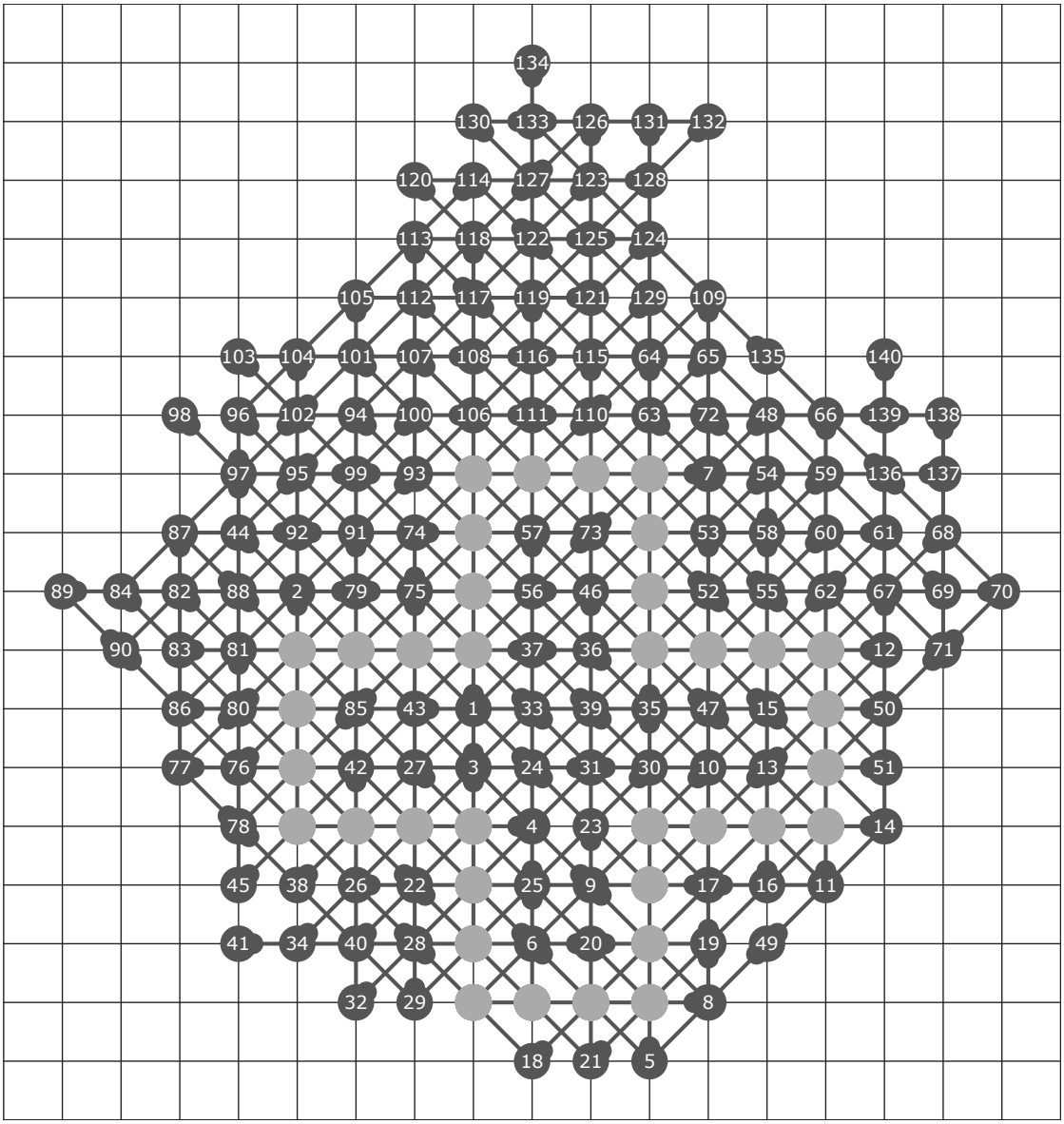
Obrázek B.6 130 tahů nalezených paralelním algoritmem s hloubkou hledání 5, 20 vlákny a řazením tahů podle souřadnic.



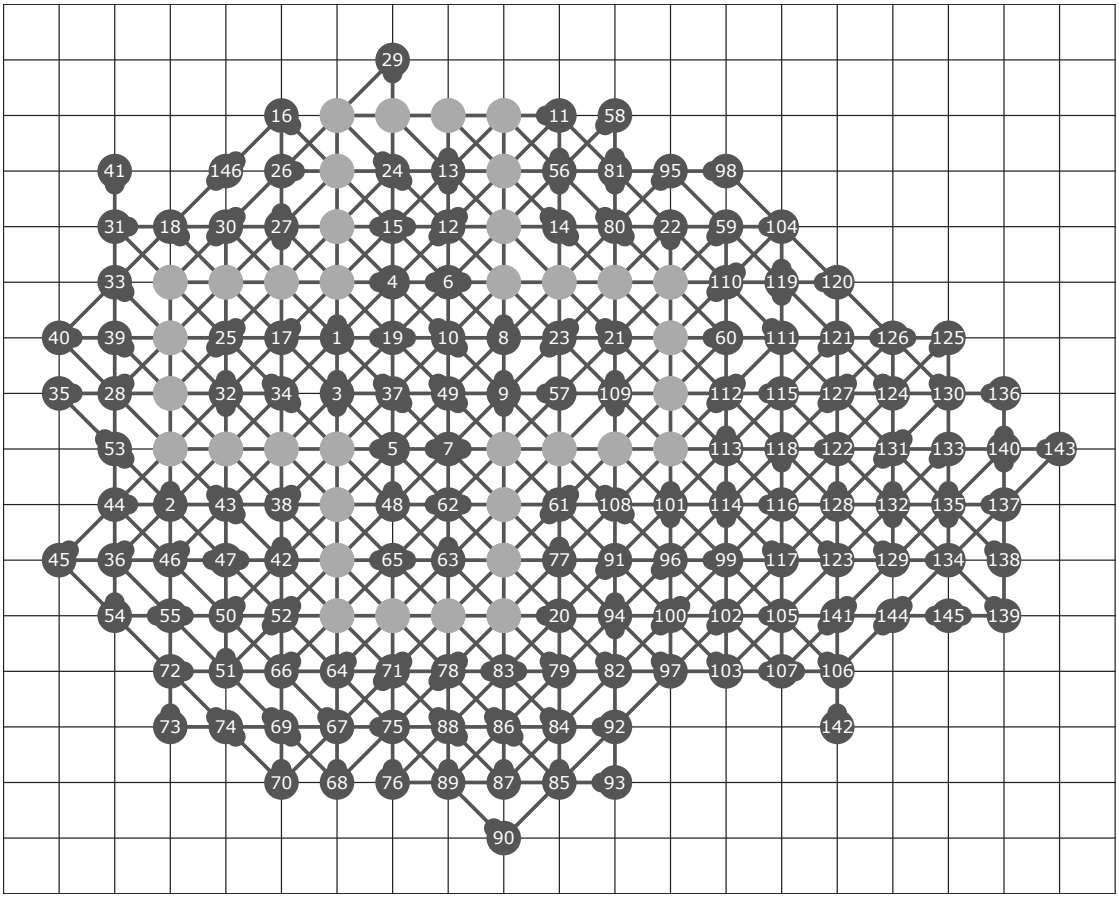
Obrázek B.7 140 tahů nalezených paralelním algoritmem s hloubkou hledání 9, 64 vlákny a řazením tahů podle bodových vah.



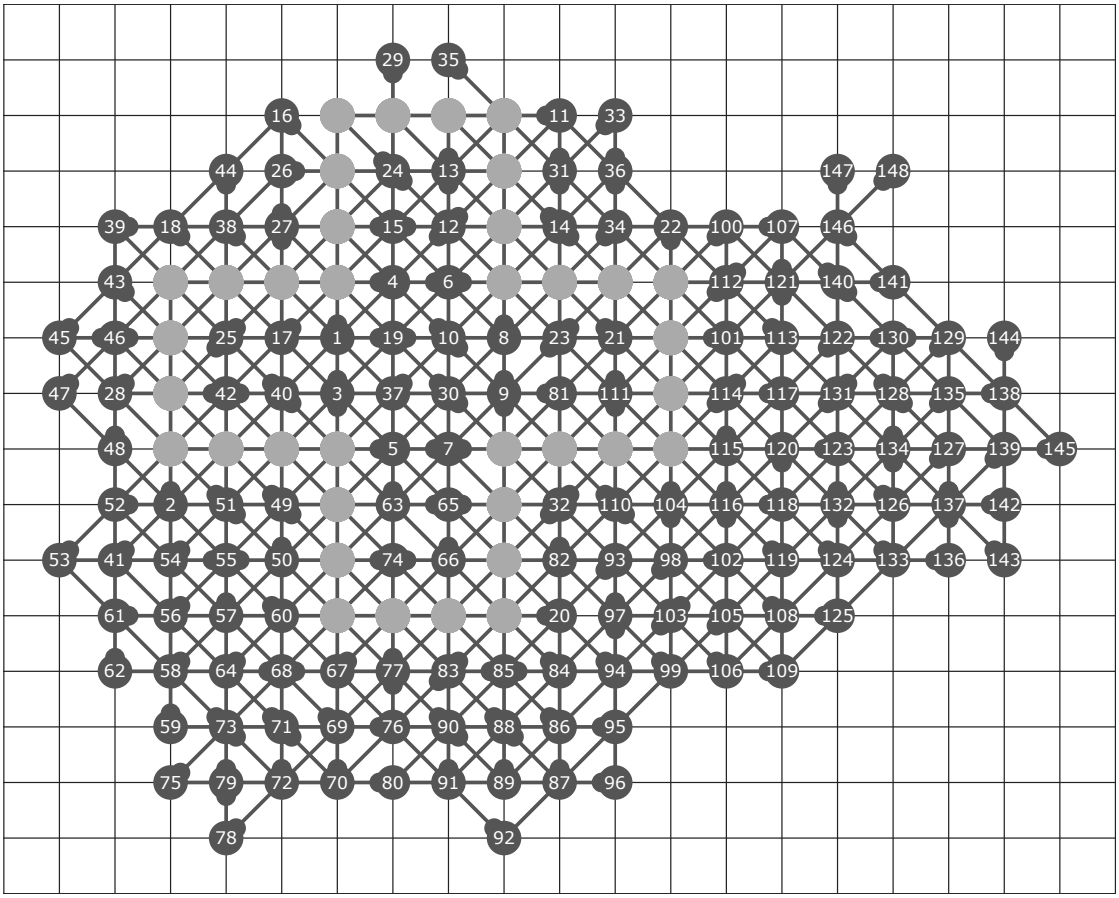
Obrázek B.8 140 tahů nalezených paralelním algoritmem s hloubkou hledání 12, 64 vlákny, řazením tahů podle bodových vah a rychlejším ukončování vláken.



Obrázek B.9 140 tahů nalezených paralelním algoritmem s hloubkou hledání 8, 64 vlákny a řazením tahů podle souřadnic.



Obrázek B.10 146 tahů nalezených paralelním algoritmem s hloubkou hledání 9, 64 vlákny a řazením tahů podle souřadnic.



Obrázek B.11 148 startem z konfigurace prvních 28 tahů řešení zobrazeného na obrázku B.10, hloubkou hledáni 3 a řazením tahů podle souřadnic.

Bibliografie

1. DUDOVÁ, Kristýna. *Morsol* [online] [cit. 2021-04-15]. Dostupné z: `https://morsol.shy.cz`.
2. BOYER, Christian. *Morpion Solitaire - Origin* [online]. 2012 [cit. 2021-04-15]. Dostupné z: `http://www.morpionsolitaire.com/English/Origin.htm`.
3. DEMAINE, Erik D.; DEMAINE, Martin L.; LANGERMAN, Arthur; LANGERMAN, Stefan. *Parallel Nested Rollout Policy Adaptation* [online]. 2004 [cit. 2021-04-21]. Dostupné z: `http://erikdemaine.org/papers/Morpion_TheoryComputSys/paper.pdf`.
4. MICHALEWSKI, Henryk; NAGÓRKO, Andrzej; PAWLEWICZ, Jakub. 485 – A New Upper Bound for Morpion Solitaire. In: 2016, s. 44–59. ISBN 978-3-319-39401-5. Dostupné z DOI: `10.1007/978-3-319-39402-2_4`.
5. BOYER, Christian. *Morpion Solitaire - Score Limits* [online]. 2012 [cit. 2021-04-21]. Dostupné z: `http://www.morpionsolitaire.com/English/Limits.htm`.
6. BOYER, Christian. *Morpion Solitaire - Enumeration* [online]. 2012 [cit. 2021-04-22]. Dostupné z: `http://www.morpionsolitaire.com/English/Enumeration.htm`.
7. BOYER, Christian. *Morpion solitaire - first record grids* [online]. 2012 [cit. 2021-04-20]. Dostupné z: `http://www.morpionsolitaire.com/English/FirstRecords5T.htm`.
8. BOYER, Christian. *Morpion Solitaire - Bruneau's Grid (5T game), the world record from 1976 to 2010* [online]. 2012 [cit. 2021-04-20]. Dostupné z: `http://www.morpionsolitaire.com/English/BruneauRecord5T.htm`.
9. HUGUES, Juillé. *Incremental Co-evolution of Organisms: A New Approach and Discovery for Optimization of Strategies* [online]. 1995 [cit. 2021-04-20]. Dostupné z: `http://demo.cs.brandeis.edu/papers/ecal95.pdf`.
10. CAZENAVE, Tristan. *Nested Monte-Carlo Search* [online]. 2012 [cit. 2021-04-20]. Dostupné z: `https://www.ijcai.org/Proceedings/09/Papers/083.pdf`.
11. BOYER, Christian. *Morpion Solitaire - Record Grids* [online]. 2012 [cit. 2021-04-20]. Dostupné z: `http://www.morpionsolitaire.com/English/RecordsGrids5T.htm`.
12. ROSIN, Christopher D. *Nested Monte-Carlo Search* [online]. 2011 [cit. 2021-04-20]. Dostupné z: `http://www.chrisrosin.com/rosin-ijcai11.pdf`.
13. NAGÓRKO, Andrzej. *Parallel Nested Rollout Policy Adaptation* [online]. 2019 [cit. 2021-04-20]. Dostupné z: `https://ieee-cog.org/2019/papers/paper_77.pdf`.
14. ROSIN, Christopher D. *Nested Rollout Policy Adaptation for Monte Carlo Tree Search* [online]. 2011 [cit. 2021-04-15]. Dostupné z: `http://www.chrisrosin.com/rosin-ijcai11.pdf`.

Obsah přiloženého média