Comenius University, Bratislava
Faculty of Mathematics, Physics and Informatics

Porovnanie ohodnocovacích funkcií pri simulovanom žíhaní na ťažkých problémoch

Bakalárska práca

2014

Rudolf Krumpál

Comenius University, Bratislava
Faculty of Mathematics, Physics and Informatics

Porovnanie ohodnocovacích funkcií pri simulovanom žíhaní na ťažkých problémoch

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 2508 Informatika
Školiace pracovisko: Katedra informatiky
Školiteľ: RNDr. Tomáš Kulich, PhD.

Bratislava, 2014
Rudolf Krumpál

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Rudolf Krumpál
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: slovenský

Názov: Porovnanie ohodnocovacích funkcií pri simulovanom žíhaní na ťažkých problémoch

Cieľ: Vytvorte prípadovú štúdiu, ako riešiť rôzne problémy pomocou simulovaného žíhania. Zamerajte sa na problémy, ktoré sú pre túto techniku náročné a zároveň, ktoré nemajú striktne zadanú hodnotiacu funkciu. Vyšetrite vplyv voľby ohodnocovacej funkcie na efektivitu riešenia problému.

Vedúci: RNDr. Tomáš Kulich, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 28.10.2013

Dátum schválenia: 28.10.2013 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie

Rád by som veľmi pekne poďakoval môjmu školiteľovi RNDr. Tomášovi Kulichovi PhD. za jeho nezištnú pomoc a cenné rady. Ďalšia vďaka patrí mojej rodine, od ktorej som cítil podporu v každej ťažkej chvíli.

Abstract

This thesis summarizes the research, which dealt with the use of the Simulated annealing algorithm on a problem, which was not customized for it. More specifically, we endeavored to resolve this problem with minimum customizations of most of the factors entering this algorithm. We were trying to focus mainly on the usage of adequate evaluation functions and proper establishment of adjacent configurations. The problem we attempted to solve is pentominoes. The quality of individual methods of generating configurations and valuation functions was assessed with comparative graphs.

Keywords: simulated annealing, evaluation functions, generating of neighbouring configurations, pentomino

Abstrakt

V tejto práci je zhrnutý výskum, ktorý sa zaoberal použitím algoritmu Simulované žíhanie na probléme, ktorý mu nebol šitý na mieru. Konkrétne sme sa snažili vyriešiť problém len minimálnymi úpravami faktorov vstupujúcich do tohto algoritmu. Snažili sme sa zamerať hlavne na použitie vhodných ohodnocovacích funkcií a vhodného vytvárania susedných konfigurácií. Problém, ktorý sme sa takto pokúsili vyriešiť sú pentomína. Kvalitu jednotlivých spôsobov generovania konfigurácií a ohodnocovacích funkcií sme zhodnotili porovnávacími grafmi.

Kľúčové slová: simulované žíhanie, ohodnocovacie funkcie, generovanie susedných konfigurácií , pentomíno

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Simulované žíhanie je aproximačný algoritmus, ktorý sa úspešne používa ako heursitika na riešenie viacerých náročných problémov. Algoritmus je tiež veľmi rozšírený v informatickom odbore umelej inteligencie. Tento algoritmus sa ukazuje ako veľmi efektívny v riešení problémov ako je problém obchodného cestujúceho. Algoritmus je aproximačný, čo znamená, že si môžeme pri jeho implementácii zvoliť, či sa chceme k úplnému riešeniu problému len priblížiť, alebo či sa chceme pokúšať špecifikovaný problém riešiť úplne.

Hra pentomíno, bola popísaná v päťdesitych rokoch ako hra dvoch pre dvoch hráčov. Hra pozostávala z hracej plochy rozmerov 8x8 a 12 herných modulov, z ktorých každý zaberal práve 5 políčok. Do plochy sa snažíme hracie moduly ukladať. Pri vkladaní do hracej plochy sa každý z hracích modulov mohol ľubovoľne otáčať a preklápať. Najpopulárnejšie sa stali však modifikácie pre jedného hráča. Jednými z prvých boli modifikácie, kde sa znížil počet políčok hracej plochy o 4, teda počet políčok, ktoré zakrývajú hracie moduly je rovnaký ako počet políčok hracej plochy. Ďalej existujú aj modifikácie, ktoré okrem hracej plochy, menili aj počty modulov, ich tvar, prípadne možnosti otáčania a preklápania.

Jednou z takýchto modifikácií, sú pentomína na hracej ploche rozmerov 5x12. Pre túto modifikáciu sa používajú klasické moduly z pôvodnej hry. Táto modifikácia má 1010 správnych konfigurácií, ktoré vedú k úspešnému zloženiu. Cieľom tejto práce bolo preskúmať, či simulované žíhanie zvládne túto ťažkú úlohu vyriešiť a poskladať pentomína. Navyše sme si stanovili, že cieľ tejto práce sa budeme snažiť dosiahnuť len správnymi ohodnocovacími funkciami.

V prvej kapitole predstavíme simulované žíhanie a bližšie si povieme o všetkých faktoroch, ktoré majú vplyv na jeho beh. V druhej kapitole si bližšie predstavíme problém pentomín. Následne popíšeme, ako sme si vytvárali stabilne pracovné prostredie pre náš výskum. Vo štvrtej kapitole si ukážeme ohodnocovacie funkcie, ktorými sme sa pokúšali pentomína riešiť. Ukážeme porovnanie kvality jednotlivých ohodnocovacích funkcií. Nakoniec výsledky zhrnieme do záveru.

Kapitola 1

Simulované žíhanie

1.1 Základné informácie

Simulované žíhanie je algoritmus, ktorý bol popísaný v roku 1983, viacerými autormi[1]. Ide o aproximačný algoritmus, ktorý je inšpirovaný žíhaním kovov v hutníctve. Takéto žíhanie funguje na prudkom ohrievaní, resp. ochladzovaní kovu, čím dosahujeme vyššiu pevnosť, prípadne pružnosť daného kovu. Podobne funguje aj simulované žíhanie. Keďže ide o aproximačný algoritmus, jeho cieľom nie je vždy získať správny výsledok, ale v priemernom prípade sa k tomuto výsledku priblížiť. Ako blízko sa chceme k výsledku približovať stanovuje konštanta zadefinovaná na začiatku algoritmu, ktorá určuje chybovosť výsledkov. Tento algoritmus sa správa podobne ako pažravé algoritmy, ale má niekoľko rozdielov[2].

1.2 Beh algoritmu

Veľká časť simulovaného žíhania beží vo while cykle. Najprv ale rozoberieme čo sa deje pred tým. Pred samotným cyklom musíme ešte ponastavovať rôzne premenné a konštanty aby žíhanie bežalo správne. Niektoré premenné a ich nastavenia sa najlepšie vysvetlia na príklade. Navyše súvisia s inými funkciami, ktoré vysvetlíme v neskorších odsekoch. Ako príklad pre tieto, sme zvolili problém ruksaku, ktorý sa dá týmto algoritmom riešiť.

Počiatočné nastavenia Prvé čo by sme chceli nastaviť, je nejaký vstupný stav. V prípade spomínaného problému ruksaku, by to mohlo byť niekoľko položiek zo zoznamu, nahádzaných v našom ruksaku. Ďalej musíme zistiť ako dobrý tento počiatočný stav. Na zistenie kvality stavov slúžia ohodnocovacie (evaluačné) funkcie. Zatiaľ nám stačí vedieť, že funkcia dostane ako vstup nejaký stav, v tomto prípade počiatočný, a vráti nám nejakú hodnotu, podľa ktorej budeme vedieť rozhodnúť ako dobrý daný stav je. Vo väčšine prípadov, nízka návratová hodnota znamená lepší stav ako vysoká návratová hodnota. Keď už vieme aký je

náš počiatočný stav, a poznáme aj jeho kvalitu, priradíme ich aj k premenným, ktoré nám budú určovať najlepšie získané riešenie. Tieto dve premenné nám pomáhajú zastaviť while cyklus, ak už máme dostatočne dobré riešenie.

Aby sme vedeli odlíšiť, čo to znamená "dostatočne dobré riešenie"môžeme si zadeklarovať premennú, ktorá stanovuje chybovosť výsledku, ktorá je prijateľná. Pri našom riešení sme zvolili chybovosť 0, lebo sme chceli zistiť, či simulované žíhanie dokáže úplne vyriešiť problém, ktorý sme si stanovili. Keď už máme nastavené, ako dobré chceme riešenia, potrebujeme ešte stanoviť počiatočnú teplotu a jej pokles. Tieto dve premenné jednak ovplyvňujú správanie žíhania počas behu, a zároveň nám určujú časové ohraničenie behu algoritmu. Keď už máme stanovené aj tieto premenné, dostávame sa k najväčšej časti algoritmu, ktorou je spomínaný while cyklus.

While cyklus Ako sme už spomínali, sú dve možnosti kedy sa tento cyklus zastaví. Prvé je časové ohraničenie, a druhé ohraničenie je výsledkové. Teda ak sa počas behu dostaneme do stavu, o ktorom ohodnocovacia funkcia povie, že je dosť dobrý aby sa zmestil do intervalu medzi ideálnym výsledkom a chybovosťou, vrátime tento dosiahnutý stav.

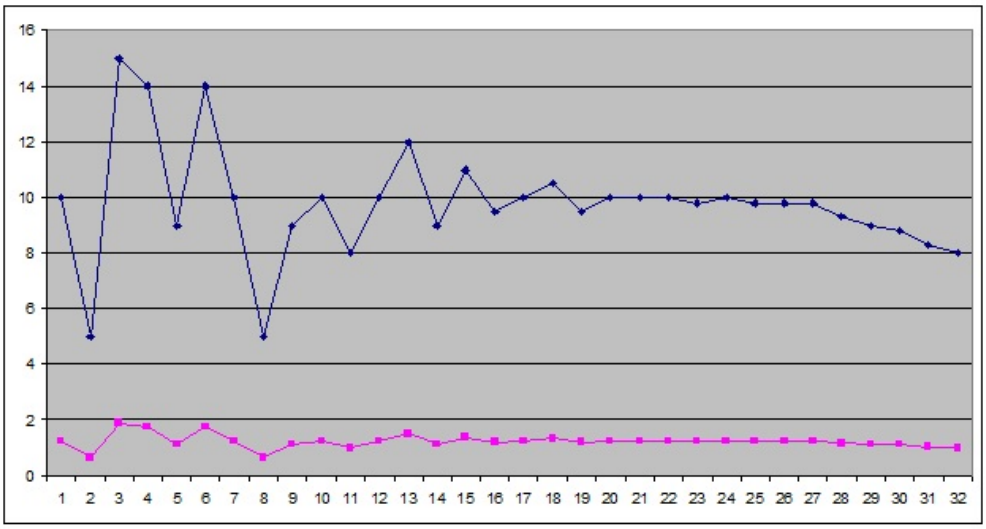
Keď už vieme ako while cyklus zastavíme, pozrime sa čo sa v ňom deje. V prvom rade by sme chceli vygenerovať susedný stav nášho súčasného stavu. Na to slúži funkcia neighbour. Jej vstupom je náš súčasný stav, a jej úlohou je v tomto stave niečo zmeniť. Zároveň však musí obsahovať nejakú náhodnosť. V prípade prblému ruksaku, by si mohla napríklad hodiť mincou, ktorá by rozhodla či ide do ruksaku objekt pridávať, odobrať, alebo ho ide vymeniť. A podľa toho by náhodnú položku do ruksaku pridala, respektíve náhodnú položku z ruksaku odobrala, alebo náhodnú položku z ruksaku nahradila náhodnou položkou zo zoznamu ešte nepoužitých. Určite ste postrehli, že môže nastať prípad kedy neighbour pridá aj odoberie tú istú položku. To nám však neprekáža, algoritmus by mal mať dostatočne veľa času na to, aby takéto stavy boli prijateľné. To však nie je jediná, zdanlivo nepríjemná vec čo sa môže pri generovaní suseda stať. Neighbour sa môže rozhodnúť, že objekt z ruksaku iba odoberie. Toto je však len zdanlivo nepríjemné. Po prvé, nás tento ťah môže dostať zo "slepej uličky"kedy už máme prekročenú maximálnu nosnosť ruksaku. Ešte nám môže pomôcť v prípade, kedy máme v ruksaku objekt, ktorý je lacný, ale zároveň veľmi ťažký. Takýto objekt v ruksaku zjavne nechceme, takže opäť nám jeho odobratie môže pomôcť. Čo sa teda deje, je že si zhoršíme lokálne optimum riešenia za cenu globálneho zlepšenia. Často krát sa ale stane, že zvolený susedný stav nebude lepší ako náš súčasný a ani nepovedie k zlepšovaniu. To nám ale neprekáža, lebo žíhanie musí mať dostatok času, aby sa s týmto problémom vysporiadalo.

Keď sme už získali nový susedný stav, opäť použijeme ohodnocovaciu funkciu, ktorá nám zodpovie otázku, ako dobrý tento stav je. Pre ďalšie účely zadefinujeme, pojmy e_{cur} a e_{new}, sú návratové hodnoty súčasného a nového stavu. Ďalej potrebujeme funkciu, ktorá nam zodpovie otázku, či sa chceme do nového stavu presunúť. Sú len dve pravidlá obmedzujúce voľnosť tejto pravdepodobnostnej funkcie P, je aby jej návratová hodnota bola z intervalu <0, 1> a aby návratová hodnota závisela od súčasnej teploty žíhania. Toto nám dava dostatočnú voľnosť vo výbere P funkcie. Tradične sa pri simulovanom žíhaní využíva táto funkcia $P = e^{(-\frac{(e_{cur}-e_{new})}{Temperature})}$ ale keďže obmedzenia pre P funkciu sú relatívne malé, funkcií pre určovanie pravdepodobnosti presunu zo súčasného stavu do nového, je v podstate nekonečne veľa. Výstupná hodnota z pravdepodobnostnej funkcie sa následne porovná s náhodne vygenerovaným číslom z intervalu <0, 1>. Ak je výstupná hodnota väčšia, presúvame sa do nového stavu. Nastavíme teda nový stav ako náš súčasný, a aj do e_{cur} priradíme hodnotu z e_{new}. Inak zachovávame aj stav aj jeho energiu na rovnakých hodnotách. Ďalej ešte skontrolujeme, či nie je náš nový stav lepší, ako najlepší stav čo sme doteraz dosiahli. Ak áno, opäť nastavíme príslušné premenné, aby zodpovedali zisteniu. Následne znížime teplotu podľa stanoveného poklesu, a pokračujeme od začiatku.

1.3 Vlastnosti simulovaného žíhania pri rôznych nastaveniach

Počiatočná teplota Simulolvané žíhanie je veľmi citlivé na nastavenia rôznych parametrov, ktoré doňho vstupujú. Ako prvým parametrom sa budeme zaoberať počiatočnou teplotou. Zdanlivo nie príliš dôležitá konštanta, môže mať veľmi jednoducho nepriaznivý vplyv na získavanie výsledkov.

Uvedený obr. 1.1 znázorňuje, predpokladané správanie simulovaného žíhania, pri postupnom chladnutí, na dvoch rôznych ohodnocovacích funkciách. Na osi x vidíme iterácie while cyklu. Teplota pri týchto iteráciách klesá. Na osi y vidíme konfigurácie, do ktrých sa žíhanie presunulo. Všimnime si funkciu znázornenú modrým grafom. Pri príliš veľkých teplotách sa od tohto algoritmu očakáva, že sa bude pohybovať v celku náhodne, teda že bude akceptovať aj stavy, ktoré jeho súčasný stav poriadne zhoršujú. Postupom času sa odchýlky zmenšujú, až sa zhoršovania úplne eliminujú, a nastane fáza kde sa pripúšťajú už len zlepšenia nášho súčasného stavu. V grafe si môžeme všimnúť nasledovné, ak zvolíme počiatočnú teplotu príliš veľkú, spôsobí to, že zbytočne strácame čas v miestach, kde sa žíhanie správa až príliš náhodne. Naopak keď zvolíme príliš nízku teplotu, naše pozorovanie sa môže zamerať už len na časť algoritmu, kde nebudeme pripúšťať zhoršovanie. Čo tiež nie je to čo sme chceli dosiahnuť. Takže počiatočnú teplotu chceme nastaviť niekde medzi tieto dva extrémy, čo



Obr. 1.1: Beh simulovaného žíhania

nemusí byť triviálne. Navyše ak sa teraz pozrieme na ružovú funkciu, tam sú rozdiely medzi zlepšovaním a zhoršovaním minimálne už pri teplotách, ktoré sú pre modrú funkciu privysoké. Z toho vyplýva, že pre ružovú funkciu by sme chceli voliť ešte vyššiu teplotu, ako pre modrú funkciu. Na záver dodáme, že treba teda pre každú ohodnocovaciu funkciu a pre každú funkciu unikátnu P zisťovať počiatočnú teplotu osobitne.

Pravdepodobnostná funkcia P Ako sme už spomínali, P funkcia má dve obmedzenia a to, že jej návratová hodnota je z intervalu <0, 1>, navyše je veľmi odporúčané, aby P funkcia závisela aj od Teploty. Čím je teplota nižšia, tým by mala byť nižšia pravdepodobnosť, že prejdeme do stavu ktorý je horší ako náš súčasný stav. Naopak, do stavu, ktorý je lepší ako náš, chceme prejsť vždy. Tieto obmdedzenia nám ale dávaju dostatočnú voľnosť pri výbere funkcie P. Zlá voľba funkcie P, môže funkcionalitu algoritmu, úplne zruinovať. Predstavme si napríklad funkciu, ktorá by vracala 1 ak je nový stav lepší ako náš súčasný, a $result = 1 - min(\frac{1}{Temperature}, 1)$ inak. Zjavne čím je teplota nižšia, tým sa znižuje pravdepodobnosť, že akceptujeme horší stav. Takže táto funkcia spĺňa predpoklady aby sa stala pravdepodobnostnou funkcie P. Ale táto funkcia sa nespráva úplne dobre. Očakávali by sme, že P funkcia sa nebude pri všetkých nízkych teplotách správať rovnako. Pri žíhaní sa často stáva, že sa dostaneme na takéto teploty, a vtedy by sme chceli mať, aj keď malú, ale aspoň nejakú šancu na zhoršenie stavu. To nám táto funkcia neposkytuje, takže nie je vhodná pre žíhanie. Tradične sa používa Boltzmannova pravdepodobnostná funkcia $P = e^{(-\frac{(e_{cur}-e_{new})}{Temperature})}$, ktorá sa správa tak, ako by sme očakávali. Občas sa používajú aj iné, zložitejšie pravdepodobnostné funkcie, ale pre náš výskum neboli potrebné.

Pokles teploty je dôležitý hlavne kvôli kontrolovaniu správania algoritmu počas jeho behu. Ako sme spomínali v predchádzajúcom odseku, P funkcia by mala redukovať možnosť presunu do horšieho stavu pri nižších teplotách, naopak pri vyšších by mala pripúšťať aj zhoršenia. Preto je dôležité akým spôsobom teplota klesá. Ak zvolíme lineárny pokles, kupujeme tým dobre vyvážené správanie počas behu. Algoritmus dostáva rovnaký čas pri takmer náhodnom skákaní z jedného stavu do druhého. Voľbou inej funkcie dostávame diametrálne odlišné správanie. Keby sme napríklad správne upravili funkciu prirodzeného logaritmu, vieme ju donútiť aby v konečnom čase išla k nule, algoritmus by teda v konečnom čase vychladol. Keď by sme sa teraz pozreli na správanie samotného žíhania, zistili by sme, že zo začiatku chladne oveľa rýchlejšie ako v neskorších častiach behu. Tým sme nášmu algoritmu získali viacej času na zlepšovanie, za cenu straty času pre náhodné skákanie. Voľbou inej funkcie vieme zase časový zisk vymeniť. Pre rôzne problémy sa ukazuje výhodné používať rôzne poklesy na ich riešenie.

Ohodnocovacie funkcie majú však zo všetkých aspektov žíhania asi najväčší vplyv na ich riešenie. Správnou voľbou ohodnocovacej funkcie môžeme získať lepšie výsledky ako úpravou vyššie popísaných aspektov, ale na druhej strane netreba ich zanedbať. V mnohých štúdiách riešení problémov pomocou tejto heuristiky, sa dbalo na súhru všetkých aspektov. Tým sa pomaly dostávame k ďalšej kapitole, v ktorej sa pokúsime vysvetliť aké nástrahy číhajú za touto heuristikou, a prečo nie je až natoľko využívaná.

1.4 Problémy simulovaného žíhania

Ako sme naznačili v predchádzajúcej kapitole, simulované žíhanie nie je príliš využívané. Je to tak preto, že obsahuje príliš veľa aspektov, ktoré majú vplyv na jeho správanie. S týmito aspektami treba stráviť veľa času, aby boli dobre vyladené. Často krát sa preto uvažujú iné heuristiky, ktoré nie sú až tak náročné na vyváženie. Navyše žíhanie vie byť dosť časovo zložité. Je to hlavne z dôvodu ohodnocovacích funkcií a zadania problému. Ak je problém časovo zložitý, bude si pravdepodobne vyžadovať evaluačné funkcie s podobnou časovou zložitosťou. Tým začneme veľmi rýchlo naberať čas, počas každej iterácie sa raz volá ohodnocovacia funkcia. Pri žíhaní býva spravidla niekoľko sto tisíc iterácií vo while cykle, čím sa čas, ktorý potrebujeme na získanie aproximovaného výsledku značne zväčší.

Jednou častí žíhania, ktoré sa ťažko vyvažujú, je pokles teploty. Zistenie správnej funkcie, ktorú volíme pre pokles teploty je často nedeterministické. Vyžaduje znova testovanie výsledkov, ktoré zaberie množstvo času. V podstate, každý parameter, ktorý sa dá nastavovať treba testovať. Teda voľba teploty, poklesu, P funkcia a aj ohodnocovacie funkcie treba vždy

doladʼovatʼ aby spolupracovali správne. Navyše toto testovanie treba robitʼ pri každej zmene. Teda ak napríklad zmeníme ohodnocovaciu funkciu, určite chceme zmenitʼ počiatočnú teplotu, od ktorej začíname testovatʼ, dʼalej možno chceme zmenitʼ funkciu pre pokles teploty a P funkciu. Keďʼ zmeníme P funkciu, tiež možno chceme menitʼ ostatné parametre a podobne pre ostatné aspekty ovplyvňujúce beh simulovaného žíhania.

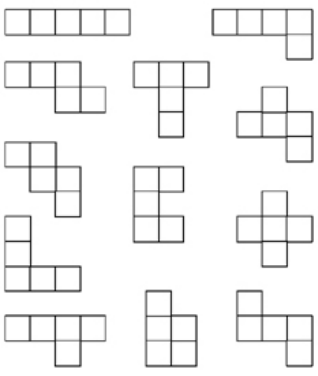
Kapitola 2

Špecifikácia problému

2.1 História

Problém ktorým sme sa zaoberali je spoločenská hra pentomína. Pravídlá tejto hry spísal v roku 1954 Solomon W. Golomb[3]. Pôvodne bola hra určená dvom hráčom. Hráči sa striedali v ťahoch a ich úlohou bolo rozmiestniť 12 hracích modulov, z ktorých každý zaberal práve päť herných políčok, na hraciu plochu veľkosti 8x8 hracích políčok. Hráč prehral v momente, keď bol na ťahu, a musel umiestniť hrací modul tak, aby prekrýval políčko zakryté iným hracím modulom, alebo by pri umiestňovaní modul musel vytŕčať z hracej plochy von.

2.2 Hracie moduly



Obr. 2.1: Moduly pentomína

Pre objasnenie, o ktorých moduloch sa bavíme, zavedieme ich pomenovanie. Použijeme jedno z dvoch zaužívaných, zhora nadol a zľava doprava sú názvy kociek nasledovné I, N,

W, V, Y, T, U, P, L, F, X a Z. Modul X sa nedá nijako rotovať ani prevracať. Pod pojmom prevracať mám na mysli zrkadlový obraz hracieho modulu. Modul I má dve pozície, vodorovnú a zvislú. Modul Z môže nadobúdať 4 rôzne podoby a to 2 rotáciou, a ďalšie dve prevrátením modulu a jeho následnou rotáciou. Moduly V, W, U a T nadobúdajú 4 podoby použitím rotácie. Zvyšné moduly teda L, N, P, F a Y môžu získať až 8 pozícií rotáciou a prevrátením. Spolu tieto moduly zaberajú 60 políčok hracej plochy. Pri rôznych modifikáciach hry pentomín, sa často modifikujú aj hracie moduly. My sme však používali také pravidlá, kde sa tieto moduly nemodifikujú. Spravidla, hry ktoré modifikujú hracie moduly, majú dosť málo riešení. K počtom riešení pri modifikáciách pentomína pre dvoch hráčov sa dostanem.

2.3 Pravidlá pre jedného hráča

Postupom času sa ľudia začali zaoberať tým kto má výhodu pri hre pre dvoch hráčov, a koľko je takých riešení, kde hra skončí remízov. Teda všetky moduly budú úspešne uložené do hracieho poľa. Ormanová v roku 1975 dokázala, vyskúšaním všetkých možností, že hráč, ktorý je prvý na ťahu, po správnom umiestnení prvého modulu má zaručené víťazstvo[4]. Po tomto dôkaze sa ľudia začali viacej zaoberať pravidlami pre jedného hráča. Vznikli 4 nové rozmery pre hracie plochy. Ako som už spomínal, hracia plocha musela obsahovať 60 herných políčok. Nové rozmery hracej plochy pre jedného hráča, 6x10, 5x12, 4x15 a 3x20. Vznikli aj ďalšie hracie plochy, ale tie majú aj modifikované hracie moduly. Samozrejme každá z týchto hracích plôch má rôzny počet riešení. 3x20 má len 3 riešenia, je to hlavne preto, že väčšina hracích modulov nemôže byť, na tejto hracej ploche, orientovaná zhora-nadol, ale len vodorovne. 4x15 má už viacej riešení, a to 368, zvýšený nárast pozorujeme preto, že existuje len jediný hrací modul ktorý nemôže byť orientovaný zhora-nadol a to je modul I. 5x12 má už 1010 riešení a 6x10 má až 2339 riešení. Tieto dve hracie plochy už nijako neobmedzujú orientácie modulov, ktoré na plochu ukladáme, preto majú aj najviac riešení.

Kapitola 3

Pracovné prostredie

Ako programovací jazyk sme zvolili python, kvôli jeho veľmi dobrým vlastnostiam čo sa týka pracovania s funkciami. Python ponúka možnosť posielať funkcie ako vstupné parametre do iných funkcií, čo je veľmi výhodný nástroj použiteľný pri našom výskume. Keďže sme chceli pracovať s ohodnocovacími funkciami, bolo veľmi výhodné mať takúto vlastnosť programovacieho jazyka k dispozícii. V podstate celá funkcia samotného simulovaného žíhania, pracuje s premennou, ktorá volá funkciu, ktorá sa do nej posiela zo vstupu.

Pravdepodobnostnú funkciu P sme zvolili $P = e^{(-\frac{(e_{cur}-e_{new})}{Temperature})}$, táto funkcia sa správa veľmi dobre v rámci sledovania nášho výskumu. Akonáhle dostane na vstup dvojicu riešení, z ktorých nové je lepšie, vracia P funkcia hodnotu 1, čo je výborné. Keďže sa návratová hodnota porovnáva s náhodne vygenerovaným číslom z intervalu <0, 1>, takáto návratová hodnota zaručuje akceptáciu lepšieho stavu. Naopak, keď dostane na vstup dvojicu, kde nový stav je horší, pravdepodbnosť akceptácie závisí hlavne na teplote, ktorá do pravdepodobnostnej funkcie vstupuje. Čím je teplota nižšia, tým sa znižuje pravdepodbnosť akceptácie takéhoto stavu.

3.1 Pracovná plocha

Za pracovnú plochu sme si pri našom výskume vybrali plochu 5x12, učinili sme tak preto, lebo v počte riešení je zhruba v strede. Navyše ide o klasické pentomína, takže pravidlá pre túto hru nijako neobmedzujú hracie moduly. Tieto nie sú nijako modifikované a teda pracujeme s rovnakou sadou aká je popísaná v predchádzajúcej kapitole.

Špecifikácia hracej plochy Pri našom riešení problému uloženia modulov do poľa sme sa zaoberal tým, či sa pomocou simulovaného žíhania dá docieliť úplné riešenie problému. Teda či sa vieme pomocou tohto algoritmu dostať do stavu kedy sú zakryté všetky políčka

hracej plochy. Moduly sme reprezentovali ako polia 5x5 kde sme mali označené, ktoré políčka modul zakrýva jednotkami, a ktoré nezakrýva, nulami. Toto riešenie sa nám zdalo v celku jednoduché a zároveň postačovalo na správne využitie v algoritme. Keďže sme sa nezaoberali pamäťovou respektíve časovou zložitosťou programu. Hraciu plochu sme reprezentovali ako pole 10x17 inicializované na -1, pričom sme moduly umiestňovali od pozície $[0, 0]$ po pozíciu $[4, 11]$. Ideálne pokrytie poľa teda predstavovalo keď časť $[0, 0]$ až $[4, 11]$ bola pokrytá nulami. Keďže nepokryté políčko má hodnotu -1, je v absolútnej hodnote táto hodnota rovnako zlá ako keď je niektoré iné políčko na hracej ploche pokryté dvoma modulmi, prípadne rovnako zlé ako keď niektorý z modulov vytŕča z hracej plochy von. To presne zodpovedá správaniu, keď sa snažíme fyzicky pentomína skladať. Počas skladania sa prioritou stáva eliminácia zle pokrytých plôch. Popri tom sa nám často stane, že niektorú plochu pokryjeme nesprávne, teda viacerými modulmi, prípadne že niektoré moduly budú kúsok vytŕčať von. Keď už rozložíme všetky moduly, potom sa ich snažíme poposúvať tak, aby sme získali dobre pokrytú hraciu plochu.

3.2 Generovanie počiatočných a susedných konfigurácií

Generovanie s možnosťou vytŕčania modulov: Keď už sme mali zastabilizované jednotlivé moduly a aj hraciu plochu, mohli sme sa pustiť do ďalšej dôležitej časti simulovaného žíhania a tou bolo generovanie počiatočnej konfigurácie a generovanie susedných konfigurácií. Počiatočná konfigurácia bola vygenerovaná náhodne. Pre každý modul sme vybrali jeho náhodne otočenie a položili sme ho na náhodnú pozíciu z intervalu $< [0, 0], [4, 11] >$. Túto konfiguráciu nastavíme ako naše súčasné riešenie a aj ako zatiaľ, najlepšie dosiahnuté riešenie. Následne by sme chceli z tohto stavu generovať susedné stavy. To sme robili tak, že sme vybrali náhodný hrací modul, a ten sme náhodne pootáčali a presunuli na náhodné miesto. Pri odoberaní modulu z hracej plochy si však musíme pamätať jeho otočenie a pozíciu na ktorej bol, aby sme mohli správnu časť plochy dekrementovať. preto sme zaviedli ešte jedno pole kde si pre každý hrací modul pamätáme jeho súčasnú polohu a otočenie. Pri každom zapisovaní modulu, si teda navyše poznačíme, je ho polohu a otočenie, aby sme ho vedeli presúvať. Keď nejaký modul presúvame, je presun značne náhodný, preto sa môže stať, že ho odoberieme a vrátime na to isté miesto v tom istom stave. To ale pri žíhaní neprekáža, lebo máme dostatok času na to, aby počet rovnakých presunov bol zanedbateľný v porovnaní s rôznymi presunmi.

Generovanie bez možnosti vytŕčania Kvôli ďalšiemu výskumu, sme špecifikovali aj ďalšie generovanie počiatočnej konfigurácie a susedných konfigurácií. Obmedzili sme zapisovanie modulov tak, aby z hracej plochy nevytŕčali von. Dôvodom tejto úpravy boli neúspechy

evaluačných funkcií pri predchádzajúcej generácii. Pri vyhodnocovaní evaluačných funkcií, sme taktiež sledovali ako výstupná hracia plocha v priemere vyzerá. Z výsledkov predchádzajúcej generácie zdanlivo vyplývalo, že problémom na získanie optimálneho riešenia boli práve vytŕčajúce moduly. Ak zapisujeme niektorý hrací modul, inkrementujeme úsek 5x5 na hracej ploche, ale pri špeciálne vybratých súradniciach sa môže stať, že veľká časť úseku trčí mimo hraciu plochu. Preto vždy keď nejaké miesto v poli inkrementujeme o 1, kontrolujeme, či náhodou neinkrementujeme políčko mimo rozmerov stanovených pre hraciu plochu. Tým zabraňujeme tomu aby hracie moduly, vytŕčali z hracieho poľa von, ale dovoľujeme ich správny zápis v rámci pravidiel. Na začiatku behu žíhania teda hracie moduly náhodne rozhádžeme na hraciu plochu. Keď generujeme susednú konfiguráciu, vyberieme náhodný hrací modul, vygenerujeme mu otočenie a polohu kam by sme ho chceli potencionálne zapísať. Keď sme pri zapisovaní zistili, že niektorá časť modulu by vytŕčala z hracej plochy von, tento pokus zvrátime, a nanovo vygenerujeme pozíciu a otočenie. Toto sa opakuje, kým sa nám nepodarí zapísať modul správne.

Princíp hracej ruky je tretím generovaním susedov, ktoré sme skúšali pri našich testovaniach ohodnocovacích funkcií. Pri tomto generovaní sme sa snažili implementovať správanie človeka pri skladaní pentomín. Niekedy sa totiž pri skladaní dostaneme do situácie, kedy si potrebujeme niektoré hracie moduly na chvíľku odložiť na bok, aby sme mohli ostatné moduly preusporiadať. A potom tam odložené moduly vrátiť späť. Vytvorili sme pre to špeciálne miesto, kam sa môžu hracie moduly odkladať. Rovnako ako pri predchádzajúcom generovaní, aj tu sme zakázali vytŕčanie modulov z hracej plochy von. Problémom však bolo, ako nastaviť pravdepodobnosť odloženia hracieho modulu. Keďže na uloženie do hracieho poľa máme veľa, potrebovali sme otestovať, výsledky niektorej z ohodnocovacích funkcií, pri rôzne veľkých šanciach na odloženie modulu z hracej plochy na ruku.

Pravdepodobnosť odobratia modulu	efektivita
75 %	4.7535
50 %	4.0475
10 %	4.135
5 %	4.225
2 %	4.3675
1 %	4.4525

Tabuľka 3.1: Rôzne pravdepodobnosti pre ruku

V tabuľke Tabuľka 3.1 sa porovnávajú výsledky jednej z ohodnocovacích funkcií, pri rôznych pravdepodobnostiach odobratia modulu z hracej plochy a ponechanie na ruke. Z

tabuľky zjavne vyplýva, že najlepšie výsledky sú dosiahnuté pri pomerne vysokých pravdepodobnostiach. Ale opäť pravdepodobnosť ponechania modulov na ruke, netreba prehnať. Ako môžeme vidieť na riadku tabuľky pre test so 75% pravdepodobnosťou, tu je výsledok značne zhoršený oproti ostatným.

Veľkosť intervalu Simulované žíhanie niekedy využíva možnosť zmenšovania intervalu z ktorého sa susedné stavy vyberajú. My sme sa rozhodli použiť jednoduchšie riešenie, a to také, že sme stanovili dostatočne malý interval za účelom zachovania jeho veľkosti. Spočiatku sme rozmýšľali, že by sme mohli napríklad zvyšovať počet presunov jednotlivých modulov, podľa súčasnej teploty, ale to sa ukázalo ako v celku nevýhodné, lebo pri nižších teplotách, by sme mali menšiu možnosť výberu, čo sme nechceli.

3.3 Stanovovanie počiatočnej teploty

Prvé pokusy Spočiatku sme si neuvedomili aké dramatické následky môže nesprávne nastavená počiatočná teplota priniesť. A nastavovali sme ju ako konštantu pre každú z testovaných ohodnocovacích funkcií. Ako sme už ukázali v kapitole 1, dobre nastavená teplota, prináša lepší prehľad o sile ohodnocovacích funkcií. Ako sme už spomínali, teplotu treba nastaviť optimálne, aby sme preskočili časť algoritmu, kedy je žíhanie príliš náhodné. Teplota tiež nemôže byť prinízke, aby sme sa nedostali rovno od začiatku do časti kde sa simulované žíhanie správa pažravo. Keď sme prišli na to, že konštantná teplota nie je postačujúca, začali sme sa venovať tomu, ako správnu teplotu zisťovať.

Metóda rozžíhania Na použitie tejto metódy sme rozšírili výstup zo simulovaného žíhania, aby sme vedeli zistiť aj odchýlku dvoch stavov, ktorá sa posiela do P funkcie. Na základe priemeru z týchto odchýlok vieme dobre určiť veľkosť počiatočnej teploty vstupujúcej do algoritmu. Metóda rozžíhania spočíva v niekoľkých zbehnutiach žíhania kde odchytávame spomínanú odchýlku. Z týchto získaných dát spravíme priemer, ktorý potom nastavíme ako počiatočnú teplotu pre simulované žíhanie. Vystupujú však tri otázky. Prvá, ako nastavíme teplotu pri prvom behu, druhá, koľko behov sa oplatí spraviť. K tretej ešte pripomíname, že pre získavanie výsledkov, musí žíhanie niekoľko krát zbehnúť a pre každé jedno zbehnutie algoritmu, musíme samotné žíhanie pustiť niekoľko krát. Je to hlavne kvôli náhodnosti algoritmu, aby sme predišli možným extrémnym riešeniam. Treťou otázkou teda je, či rozžíhavať pred každým zbehnutím žíhania, alebo len pred každým zbehnutím samotného testu. Ľahko vidieť, že ak by sme rozžíhavali pred každým zbehnutím žíhania, znova by sme predĺžili celkový beh programu. Naopak, keď by sme rozžíhavali iba pred testom, nestratíme toľko času, ale je možné, že teplota sa nenastaví až natoľko dobre.

Odpoveďou na prvú otázku je, že pri prvom rozžíhaní nastavíme počiatočnú teplotu na nejakú konštantu, ktorá je spoločná pre všetky testy. Túto konštantu sme nastavili pozorovaním počiatočných teplôt pri rôznych testovacích behoch. Z tohto behu získame odchýlky stavov ktoré sa posielali do P funkcie. Spravíme z týchto odchýlok priemer, a ten nastavíme ako počiatočnú teplotu pre ďalšie kolo rozžíhavania. Tento priemer nepočítame do priemerov, z ktorých vypočítame finálnu počiatočnú teplotu. Pre každé ďalšie rozžíhanie bude počiatočná teplota posledný získaný priemer odchýlok stavov v žíhaní.

Teraz odpovieme naraz odpovieme na otázky dva a tri. Na získanie týchto odpovedí sme spravili test, v ktorom sme sledovali správanie rovnakej ohodnocovacej funkcie. V teste bola použitá objektívna ohodnocovacia funkcia, ktorú popíšeme nižšie. Na získanie výsledkov sme nastavili 100 behov žíhania, a pre každý beh maximálne 100000 iterácií while cyklu. Iterácií môže byť aj menej za predpokladu, že počas niektorého z behov úspešne pokryjeme hraciu plochu. V teste sme sledovali úspešnosť ohodnocovacej funkcie, pri rôznych typoch rozžíhania, ktoré sme už spomínali. Pre jeden z dvoch typov, sme spravili ešte jeden test navyše kde sme použili nižší počet rozžíhaní, aby sme vedeli porovnať aj časový vplyv na efektivitu tejto metódy.

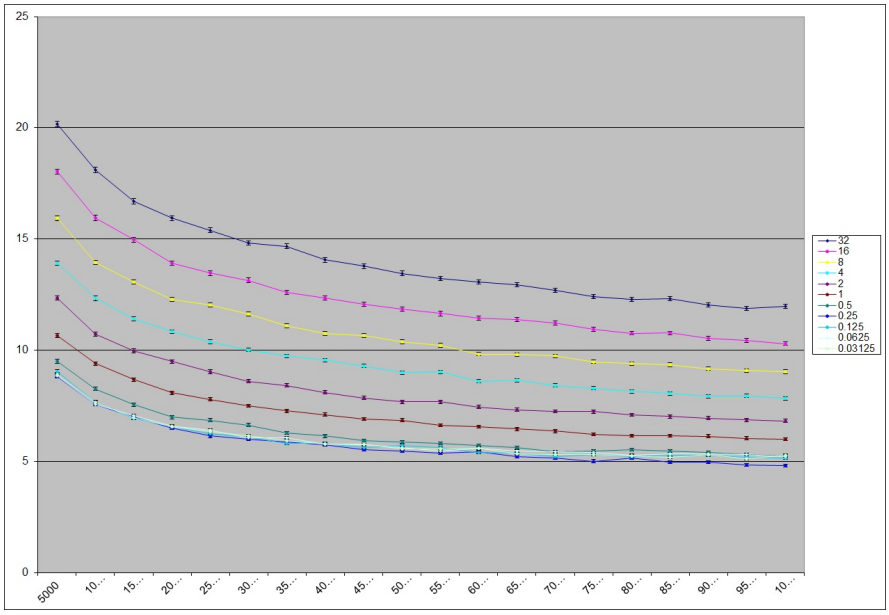
Použité rozžíhanie	efektivita	sigma	čas(min)
Vonkajšie rozžíhanie 6 iterácií	6,968	0,07298	9
Vonkajšie rozžíhanie 101 iterácií	6,818	0,07685	15
Vnútorné rozžíhanie 6 iterácií	6,724	0,07205	31

Tabuľka 3.2: Porovnanie efektivity rozžíhania

Vnútorné rozžíhanie znamená že pre každý zo 100 behov, rozžíhavame osobitne. Vonkajšie rozžíhanie, naopak znamená, že rozžíhavame pre všetkých 100 behov raz. Číslo za názvom, udáva počet iterácií rozžíhania. Pripomíname, že prvý beh rozžíhania nenastavuje počiatočnú teplotu. Z tabuľky 3.1 môžeme vidieť, že rozdiel medzi vonkajším a vnútorným rozžíhaním nie je až tak veľký. Efektivita označuje, koľko políčok bolo v priemere zle pokrytých. Stĺpec sigma nám ukazuje, smerodajnú odchýlku, ktorá sa používa v štatistike. V poslednom stĺpci máme časový údaj, ktorňy nám hovorí o tom, koľko každý z testov trval. Z časového hľadiska je vidieť, že vnútorné rozžíhavanie je ďaleko náročnejšie ako vonkajšie, rovnako ako sme predpokladali. Ale keď by sme porovnali zlepšenie vonkajšieho rozžíhania 6, a vnútorného rozžíhania 6, zistíme, že za cenu zhruba 3,5 krát dlhšieho behu, získame len niečo málo cez dve desatiny zlepšenia vo výsledkoch. Čo nie je postačujúce zlepšenie na to aby sme si o tolko predĺžili beh. Keď teraz porovnáme výsledky vonkajšieho rozžíhania 101 a 6, vidíme že sme sa zlepšili zhruba o 0,15 v priemernom výsledku. Na základe týchto

výsledkov, sme sa rozhodli, že budeme používať vonkajšie rozžíhanie 51. To nám zaručí postačujúce informácie ohľadom dobrej počiatočnej teploty.

Násobenie výstupov z rozžíhania Keď už sme zhruba vedeli ako zisťovať počiatočnú teplotu, spravili sme ďalšie testy, aby sme mohli výsledok z rozžíhania ešte trošku zlepšiť.



Obr. 3.1: Test násobkov počiatočnej teploty

Na obrázku obr. 3.1: Test násobkovv počiatočnej teploty, vidíme graf, kde na osi x, sa postupne zvyšuje čas, ktorý má simulované žíhanie k dispozícii počas výpočtu. Postupne sa zvyšuje od maximálne 5000, po maximálne 100000 iterácií while cyklu. Na osi y vidíme počet zle pokrytých hracích políčok. V legende môžeme vidieť priradenia farieb jednotlivých testov k násobkom počiatočnej teploty. Vo všetkých testoch bola použitá rovnaká ohodnocovacia funkcia. Pre každý test bolo použité vonkajšie rozžíhanie s 51 iteráciami, a jeho výsledok bol vynásobený číslom uvedeným v legende.

Všimnime si, že keď te počiatočná teplota privysoká, výsledky sú oveľa horšie ako výsledky pri nižších počiatočných teplotách. Tiež si všimnime, že naopak pri vysokých počiatočných teplotách sa zvyšovaním času na výpočet, zlepšujú dosiahnuté výsledky viac, ako pri nízkych. Dôvodom tohto správania, je použitá P funkcia. Jej správaniu sme sa už venovali v predchádzajúcich odsekoch, preto len spomenieme, že naša P funkcia sa pri nízkych teplotách správa pažravo, naopak pri vyšších teplotách dosť náhodne. Tým že zvyšujeme

výpočtový čas, zvyšujeme aj časový úsek v pre obe tieto časti a aj pre časti medzi nimi. Ale keď sa už pozrieme na zlepšovanie okolo 50000 až 100000 iterácií, môžeme vidieť, že prudké zlepšovanie v začiatkoch, sa značne zmierňuje. A teda ak by sme chceli naplno využiť potenciál vysokých teplôt, potrebovali by sme značne viac výpočtovej sily. Navyše predpokladáme, že by sa zlepšovanie nespomalilo natoľko, až by ostalo stagnovať.

Teraz sa bližšie pozrime na nižšie teploty. Všímať si budeme najmä tmavomodrú krivku, ktorá symbolizuje jednu štvrtinu z nameranej hodnoty, ďalej zeleno-modrú krivku symbolizujúcu polovicu z výstupu rozžíhania. Najprv sa budeme venovať, druhej spomínanej krivke. Ako si môžeme všimnúť, pri malých počtoch iterácií, je o čosi horšia ako menšie počiatočné teploty, ale postupom času sa im vyrovnáva, až napokon prekoná jednu z kriviek pre menšie teploty. Tým sa preukazuje, že simulované žíhanie môže byť, pri dostatočnom čase na jeho beh, lepšie ako pažravé algoritmy. Toto potvrdzuje najmä tmavomodrá krivka symbolizujúca, štvrtinu rozžíhavacej hodnoty. Táto hodnota je pri kratšom čase na výpočet takmer rovnako efektívna ako nižšie násobky. Navyše, pri dlhšom čase tento násobok nameranej hodnoty, dosahuje lepšie výsledky ako nižšie počiatočné teploty.

My sme napokon za počiatočnú teplotu pre naše testy zvolili polovicu nameranej hodnoty z rozžíhania. Hlavne kvôli tomu, aby sme nemali príliš veľa času v častiach kde je simulované žíhanie pažravé. Keď zvolíme počiatočnú teplotu prinízku, dávame v podstate pôvodnému stavu šancu iba sa zlepšovať, čo nie je cieľom simulovaného žíhania.

Posledná vec, čo sme ešte nenastavili je maximálny počet iterácií while cyklu, ktoré obmedzujú beh algoritmu časovo. Počet iterácií sa obmedzuje rýchlosťou chladnutia, teda rýchlosť chladnutia má navyše aj priamy vplyv na to, ako sa simulované žíhanie počas behu správa. My sme zvolili lineárne chladnutie, lebo sme chceli zabezpečiť, aby každý typ správania algoritmu mal rovnaký čas na preukázanie svojich predností. Ďalej sme chladnutie nastavovali tak, aby sme za rozumne dlhý výpočtový čas dostali výsledky.

Týmto nastavením, sme už mali zastabilizované takmer všetky vstupné parametre, ktoré samotný algoritmus ovplyvňujú. A tak sme sa dostali ku generovaniu ohodnocovacích, resp. evaluačných funkcií. Týmto sa však budeme venovať až v ďalšej kapitole, keďže táto skupina parametrov, je pre náš výskum kľúčová.

Kapitola 4

Výsledky

V tejto kapitole sa budeme zaoberať hlavným cieľom tejto práce. A to skúmaním sily jednotlivých ohodnocovacích funkcií. Ako sme už spomínali v predchádzajúcej kapitole, používali sme tri rôzne generovania susedných konfigurácií. Generovanie s možnosťou vytŕčania, bez nej a generovanie s použitím hracej ruky. Pri každom teste sme použili vonkajšie rozžíhanie s 51 rozžíhavacími iteráciami. Po rozžíhaní sme žíhali ďalších 400 krát, s nameranou počiatočnou teplotou. Počet iterácií while cyklu v samotnomžíhaní sme nastavili na 250000.

4.1 Použité ohodnocovacie funkcie

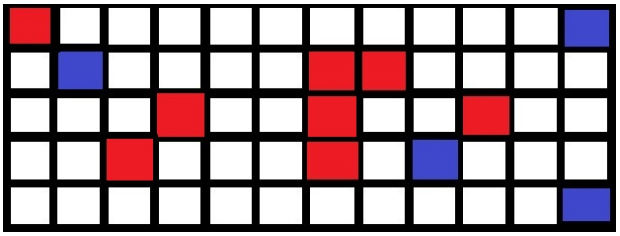
V tejto podkapitole popíšeme použité ohodnocovacie funkcie. Pri každej funkcii sa bude nachádzať aj obrázok, reprezentujúci štandartné správanie danej ohodnocovacej funkcie, pri generovaní susedných konfigurácií s možnosťou vytŕčania modulov z hracej plochy.

Ohodnocovacia funkcia, dostáva na vstup stav, v ktorom sa možno chceme nachádzať. Jej úlohou je tento stav ohodnotiť podľa ľubovoľných pravidiel. Návratová hodnota z tejto funkcie sa následne v žíhaní porovnáva, a rozhoduje o tom, či sa do daného stavu žíhanie posunie, alebo zotrvá vo svojom súčasnom stave.

Najprv sme vytvorili niekoľko ohodnocovacích a pomocných, boli to - objektívna ohodnocovacia funkcia, funkcia minimalizujúca obvody ostrovov, funkcia penalizujúca zle pokryté políčka, funkcia minimalizujúca počty ostrovov, ktoré sú navyše ďaleko od stredu hracej plochy a funkcia penalizujúca políčka, ktoré sa na ploche prekrývajú. Tieto funkcie sa však ťažko porovnávajú keďže každá sa zaoberá inou vlastnosťou daného stavu. Preto sme sa rozhodli, že vytvorime rozšírenú sadu testovacích funkcií a to tak, že budeme vždy vybrané dve funkcie kombinovať. Na ich vhodné kombinovanie, sme použili konštanty, ktorými sme vyvažovali výstupy z ohodnocovacích funkcií. Takto sme získali jednak nástroj na porovná-

vanie sily jednotlivých funkcií, a navyše sme získali možnosť kombinovať funkcie a tým pádom zlepšovať konečné výstupy.

Objektívna ohodnocovacia funkcia Táto funkcia kontroluje počet zle pokrytých hracích políčok. Postupne prechádza celé hracie pole, a spočítava absolútne hodnoty, ktoré sa na jednotlivých políčkach nachádzajú. Túto funkciu sme nazvali objektívna, lebo sa naozaj zameriava na celkový stav hracej hracej plochy, a nie len na niektorú vlastnosť stavu. Tradične, stav, ktorý žíhanie vracia ako výstupný vyzerá ako na uvedenom obr. 4.1.



Obr. 4.1: Hracia plocha s použitím objektívnej ohodnocovacej funkcie

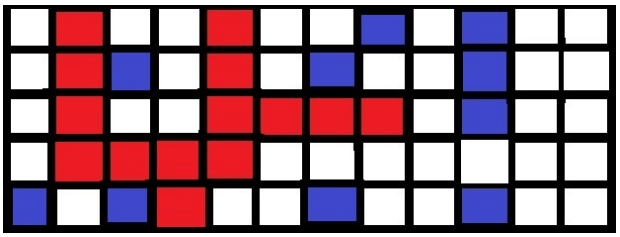
Pripomíname, že na obrázku je hracia plocha, ktorú dostaneme pri nižšom počte iterácií while cyklu, konkrétne 300. Je hlavne z dôvodu, aby bolo viacero políčok nesprávne pokrytých, aby vynikli vlastnosti jednotlivých ohodnocovacích funkcií. Červené políčka na obrázku znázorňujú úseky, ktoré nie sú pokryté žiadnym hracím modulom. Modré naopak, sú pokryté dvoma alebo viacerými hracími modulmi. Všimnime si, že červených políčok je na ploche viacej ako modrých. Je to jednak kvôli spôsobu akým generujeme jednotlivé stavy a aj kvôli tomu, že na modrých políčkach môže byť niekoľko hracích modulov naraz. Čo sa týka generovania stavov, pri vzniku obrázku nebolo obmedzenie vytŕčania modulov mimo plochu, preto okrem modrých políčok na obrázku hracej plochy, existujú ešte modré políčka mimo nej. Jednoducho niektoré moduly z hracej plochy vytŕčajú von.

Problémom tejto ohodnocovacej funkcie, je to, že zle pokryté políčka sú na hracej ploche rozhádzané príliš náhodne. Čo nie je dobre, lebo keď by sme chceli niektoré zle pokryté políčko zakryť, musíme presunúť hrací modul, a prípadne ho aj správne pootáčať. Tým nám ale vzniknú ďalšie zle pokryté plochy, ktoré budú opäť náhodne rozhádzané.

Naopak jej výhodou je, že dostávame funkciu, ktorou môžeme dobre ohodnotiť aj úspešnosť iných ohodnocovacích funkcií. A to tak, že jednoducho vrátený stav ohodnotíme touto ohodnocovacou funkciou. Bez ohľadu na to, akú funkciu sme použili.

K tejto funkcii, sme vytvorili ešte jednu, ktorá jej je v celku podobná. Táto funkcia sa pozerá iba na nezakryté hracie políčka, a vracia dvojnásobok ich celkového pokrytia. Tým pádom získavame prehľad aj o vytŕčajúcich moduloch. Túto funkciu sme neskôr používali pri porovnávaní ohodnocovacích funkcií, kedy sme dané funkcie aj kombinovali.

Funkcia minimalizujúca počty nepokrytých ostrovov bola prvou z pomocných funkcií, ktoré sme implementovali. Predtým ako sa budeme venovať tejto funkcii, vysvetlíme pojem ostrov. Ostrov je skupina nezakrytých hracích políčok, susediacich hranami, ktoré sú zo všetkých strán ohraničené zakrytými políčkami, alebo okrajom hracej plochy. Teraz zodpovieme otázku, prečo sme sa rozhodli skúšať, ako si môžete všimnúť na obrázku Obr. 4.1, červené políčka znázorňujúce nezakryté plochy, sú rozhádzané v celku náhodne po celej hracej ploche. Pre získanie správneho riešenia, je lepšie, keď nezakryté políčka tvoria nejakú súvislú plochu. Hlavným dôvodom je, že do takejto súvislej plochy sa ľahšie umiestňuje hrací modul, tak aby zakryl naraz viacej políčok, než do nesúvislej plochy.

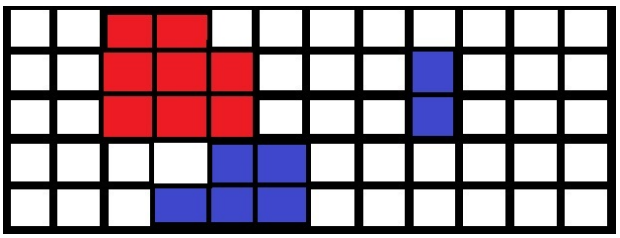


Obr. 4.2: Hracia plocha s použitím ohodnocovacej funkcie minimalizujúcej počty nepokrytých ostrovov

Prvá vec, ktorú si môžeme na obrázku Obr. 4.2 všimnúť, je, že zle pokrytých políčok je oveľa viacej ako na predchádzajúcom Obr. 4.1, je to tak kvôli tomu, že táto ohodnocujúca funkcia, sa vôbec nestará o to, koľko políčok je na ploche zle zakrytých. Čo je v celku smutné, lebo aj táto ohodnocovacia funkcia má šancu na úspech. Keby sa jej podarilo minimalizovať počet nepokrytých ostrovov na 0, znamenalo by to, že hracia plocha je dobre pokrytá. Dôležitejšia je však druhá vec znázornená na obrázku 4.2 a to, že ohodnocovacia funkcia sa správa tak ako sme očakávali. Naozaj, nepokryté hracie políčka znázornené červenou farbou, sú zoskupené do jedného súvislého ostrova. Políčka, ktoré sú pokryté dvoma alebo viacerými hracími modulmi, nás až tak netrápia, lebo minimalizáciou nepokrytých ostrovov, minimalizujeme aj počet takýchto políčok. Pripomíname, že táto ohodnocovacia funkcia bola len pomocná, takže sa vždy kombinovala s niektorou majoritnou funkciou, ktorá mala preukázateľne lepšie výsledky

Čo je však horšie, je tvar tohto ostrova. Ostrov v podstate tvorí akéhosi dlhého kľukatého hada, čo je problém, lebo len málo hracích modulov má priamy tvar. Keď sa teda snažíme takýto ostrov zakrývať, máme len malú pravdepodobnosť, na to, že ostrov nerozbijeme na dva menšie ostrovy, ktoré táto ohodnocovacia funkcia odsúdi k neakceptovaniu. Ak by sa nám aj podarilo ostrov nerozbiť, presúvaním hracieho modulu, si s veľkou pravdepodobnosťou vytvoríme niekoľko ostrovov, na pozícii odkiaľ sme tento modul zobrali. Takže sa nám opäť zvyšuje počet, ostrovov a takýto stav je tiež odsúdený k malej šanci na akceptovanie.

Funkcia minimalizujúca obvody ostrovov Aby sme predišli problému predchádzajúcej funkcie, zamerali sme sa na inú vlastnosť ostrovov, a to je ich celkový obvod. Vytvorili sme ohodnocovaciu funkciu, ktorá sa snažila tento obvod minimalizovať. Podľa našich predpokladov, by sa teda had z predchádzajúcej funkcie mal zmeniť na plochu príjemnejšieho tvaru. Ďalšou výhodou tejto vlastnosti je to, že má vplyv aj na veľkosti ostrovov. Pochopiteľne čím má ostrov menší obvod, tým sa zmenšuje aj plocha ktorú pokrýva. Aby sme túto funkciu ešte vylepšili, pridali sme jej dve varianty. Prvá minimalizovala obvody nepokrytých ostrovov a druhá obvody ostrovov kde sa moduly prekrývali. Toto by malo spôsobiť, že niekde na hracej ploche sa vytvorí ostrov kde sa políčka prekrývajú a inde kde sa sústredia nepokryté políčka.

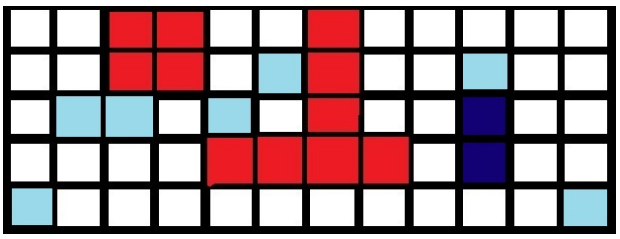


Obr. 4.3: Hracia plocha s použitím ohodnocovacej funkcie minimalizujúcej obvody ostrovov

Ako si môžeme všimnúť na obrázku Obr. 4.3, naše predpoklady na správanie boli správne. Čo nás milo prekvapilo bol fakt, že táto funkcia dokonca aj minimalizuje počty ostrovov. Ako si môžeme všimnúť, nepokryté políčka sú umiestnené do v celku pekne vyzerajúceho ostrova. Do takéhoto tvaru vieme umiestniť oveľa viacej modulov, ako do ostrova, ktorý vytvárala predošlá funkcia. Prekrývajúce sa časti hracej plochy sú sústredené len do dvoch ostrovov, čo je určite lepšie ako keď sú moduly rozhádzané po hracej ploche. Do problémov sa táto funkcia však dostáva v momente, keď má žíhanie dosť času na rozmiestňovanie modulov. Pripomíname, že obrázky sú vybrané zo žíhania pri 300 iteráciách while cyklu, zatiaľ čo testy bežali pri 250000 iteráciách. Pri testovacích hodnotách táto funkcia často narážala na problém, kedy ostali na ploche nepokryté dve až tri hracie políčka, ktoré spolu nijako

nesusedili. Pri nižších teplotách sme sa z tohto stavu už nevedeli pohnúť, lebo akýkoľvek presun hracieho modulu spôsoboval zhoršenie súčasného stavu.

Funkcia penalizujúca políčka, kde sa hracie moduly prekrývajú Táto ohodnocovacia funkcia, vznikla pomerne neskoro a jej zameranie je podobné ako u objektívnej ohodnocovacej funkcie. Rozdielom však je, že tvrdo penalizuje políčka, kde sa moduly prekrývajú. Jej vznik bol podmienený vznikom generovania bez možnosti vytŕčania modulov. Pri tomto generovaní sa objektívnej ohodnocovacej funkcii stávalo, hlavne pri vyšších teplotách, že sa prekrývalo veľa modulov na jednom mieste. Čo je dosť nepríjemná záležitosť, lebo ak takýto stav akceptujeme, máme problém sa takéhoto zhluku modulov zbaviť. Preto sme sa rozhodli takéto zhluky poriadne penalizovať.



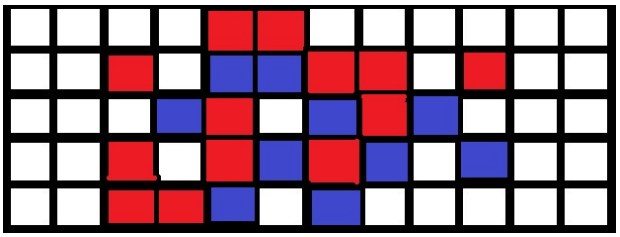
Obr. 4.4: Hracia plocha s použitím ohodnocovacej funkcie penalizujúcej prekryvy modulov

Za povšimnutie na obrázku Obr. 4.4 stojí, že môžeme vidieť dva odtiene modrej. Tmavší odtieň znamená, že na daných políčkach sa prekrývajú tri hracie moduly. Na svetlejších dva, biele políčka sú pokryté práve jedným hracím modulom, a červené ostali nepokryté. Všimnime si, že oproti Obr. 4.2 je modrých políčok oveľa menej, a navyše keď porovnáme počty modrých políčok, na Obr 4.4 sú počty vyrovnanejšie, takže táto funkcia pracuje dobre. Pri dlhšom čase na chladnutie, sa táto funkcia správa ešte o čosi lepšie, ale zlepšenie žiaľ nie je dostatočne rýchle. Zdanlivo máme možnosť ešte viac zvýšiť pokutu, ktorú musíme zaplatiť za prejdenie do stavu s prekrytými modulmi. Toto sprísnenie požiadavky by však spôsobilo, že budeme mať obrovské rozdiely medzi jednotlivými stavmi, čo nám v konečnom dôsledku rapídne zvýši počiatočnú teplotu. Tým pádom budeme mať menší počet iterácií while cyklu pri dostatočne nízkych teplotách, kde už algoritmus pôsobí pažravo.

Napriek týmto nevýhodám, sa funkcia celkom úspešne snažila eliminovať prekryvy. Na obrázku Obr. 4.4 môžeme vidieť, že po 300 iteráciách, sú dvomi modlmi pokryté len dve políčka hracej plochy. Preto sme túto funkciu neskôr ešte použili ako pomocnú, v kombinácii s inou ohodnocovacou funkciou. Ku kombináciam ohodnocovacích funkcií sa ešte dostaneme v neskorších podkapitolách.

Častým problémom pri testovacích riešeniach predchádzajúcich ohodnocovacích funkcií bolo to, že ostrovy boli na ploche rozmiestnené náhodne. Pravidelne sa stávalo, že zle pokrytý ostrov bol na okraji, v najhoršom prípade v rohu hracej plochy.

Funkcia penalizujúca vzdialenosť od stredu Pridali sme teda novú ohodnocovaciu funkciu, ktorej úlohou nebolo nič iné, len penalizovať zle pokryté políčka, za ich vzdialenosť od pomyselného stredu hracej plochy. Predpokladali sme, že jej výsledky nebudú príliš oslňujúce, nakoľko sa vôbec nestaráme o veľkosť zle pokrytej hracej plochy. Táto funkcia bola vytvorená trošku neskôr ako ostatné, aby sme vyskúšali, ako vie výsledky zlepšiť, keď dopomáha iným ohodnocovacím funkciám.



Obr. 4.5: Hracia plocha s použitím ohodnocovacej funkcie penalizujúcej vzdialenosť od stredu

Naša hracia plocha má dva stredy, táto ohodnocovacia funkcia najprv pre každé zle pokryté políčko zistí, na ktorej polovičke hracej plochy sa nachádza. Hraciu plochu sme rozdelili na dve časti vertikálne. Podľa toho určíme, ktoré políčko zoberieme ako náš súčasný stred. Následne pomocou pytagorovej vety, kde strany a a b v trojuholníku sú vzdialenosti políčka os x a os y od nášho stredu, vypočítame skutočnú vzdialenosť políčka od stredu. Túto hodnotu pošleme ako návratovú z ohodnocovacej funkcie.

Na Obr. 4.5 si môžeme všimnúť, že znova nám narástol počet zle pokrytých políčok na hracej ploche. To sme však predpokladali. Pozrime sa bližšie na vlastnosť, ktorú má táto pomocná funkcia zohľadňovať. Ako môžeme vidieť, zle pokryté políčka sú natlačené čo najviac do stredu. Pričom okrajové časti hracej plochy ostávajú nepokryté. Tento fakt je veľmi dôležitý pre ukladanie modulov do poľa. Mnohé z nich majú totiž časti, ktoré vytŕčajú do rôznych smerov. Tieto moduly by sme teda nechceli mať na kraji, lebo nám často zabránia úspešne zložiť pentomína. Prázdne alebo viacnásobne prekryté políčko sa z okrajových častí ťažko odstraňuje, preto je lepšie tieto políčka tlačiť do stredu. Podobne postupujeme aj keď sa snažíme pentomína fyzicky skladať. Najprv sa snažíme kockami obstavať okraje hracej plochy a následne sa pokúšame vtesnať ostatné hracie moduly do vnútra. Naskladaním modulov na okraj sa nám v strede vytvára množstvo zle pokrytých políčok, čo je presne to, o čo

sa pokúša táto ohodnocovacia funkcia

Táto pomocná funkcia je poslednou, ktorú sme vytvorili, ďalej sme sa zaoberali tým, ako sa dajú jednotlivé funkcie porovnávať.

V predchádzajúcej kapitole sme si popísali funkcionalitu ohodnocovacích funkcií. Pri otázke porovnávania ohodnocovacích funkcií, sme dospeli k záveru, že jednotlivé vlastnosti by sa dali skombinovať do nových funkcií. Preto sme pridali do funkcií parameter, ktorý nám určoval akú váhu majú mať jednotlivé ohodnocovacie funkcie vo veľkej skombinovanej funkcii. Tým sme jednak získali dobrý nástroj na porovnávanie sily ohodnocovacích funkcií a zároveň sme v podstate zadarmo dostali možnosť využívať silu viacerých funkcií naraz. Čo sme však stratili, bol čas, keďže sme nevytvárali nové funkcie, ale používali sme dve existujúce funkcie, dostali sme konštantné zhoršenie času, pri behu simulovaného žíhania.

Začali teda vznikať nové ohodnocovacie funkcie, ako napríklad funkcia, ktorá zároveň minimalizovala počty ostrovov na hracej ploche a súčasne minimalizovala obvody týchto ostrovov. Alebo ohodnocovaciu funkciu ktorá minimalizovala počty ostrovov a snažila sa tieto vtesnať čo najviac do stredu hracej plochy.

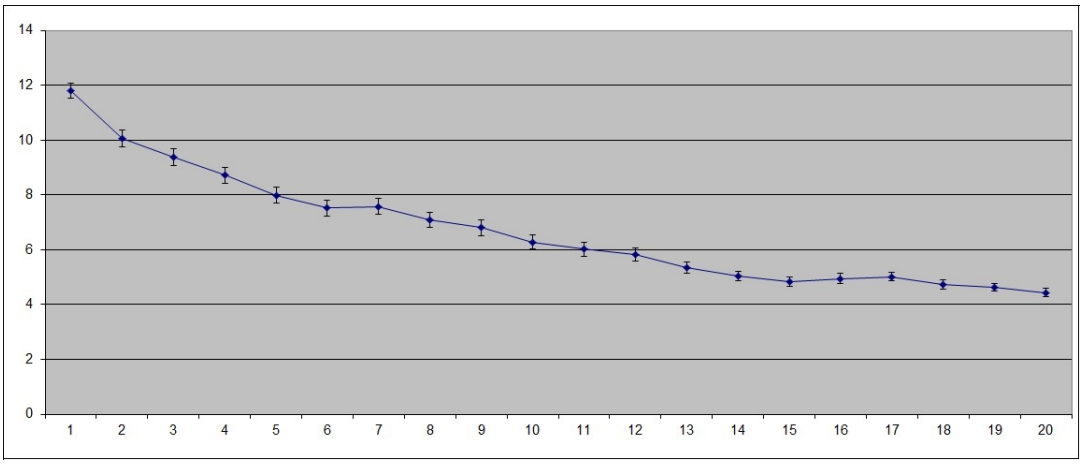
Pri testovaní sme zároveň zistili aké je optimálne vyváženie dvoch funkcií, ktoré boli skombinované, a moli sme vytvoriť ešte zložitejšiu funkciu, ktorá sa pozerala na viacej vlastností naraz. Testovanie ohodnocovacích funkcií teda prebiehalo nasledovne.

Zobrali sme dve ohodnocovacie funkcie, o ktorých sme predpokladali, že by mohli dobre spolupracovať. Nastavili sme počet iterácií v jednom žíhaní 250000 a počet behov celého žíhania na 400. Postupne sme menili 21 váh v kombinovanej funkcii a sledovali sme správanie na základe priemerných výsledkov, ktoré sme získali zo 400 behov. Ďalej sme ešte sledovali aj ako sa mení priemerná odchýlka výsledkov z jednotlivých behov. Na získavanie hodnôt pre výsledky, sme používali objektívnu ohodnocovaciu funkciu, keďže táto nám povie presne koľko políčok na hracej ploche je zle pokrytých. Pri váhe 0, bola maximálna váha na prvej z dvoch vstupujúcich funkcií, pri váhe 20 na druhej z nich. Teda porovnaním extrémnych váh, získavame prehľad o sile vstupujúcich funkcií. Sledovaná odchýlka nám zase prezrádzala nakoľko sú výsledky presné, a zhruba okolo akých počtov zle pokrytých políčok sa pohybujeme.

4.2 Generovanie s možnosťou vytŕčania modulov

Generovanie s možnosťou vytŕčania modulov z hracej plochy von je najjednoduchšie generovanie, ktoré sme testovali. Pri inicializácii, pre každý hrací modul vyberieme náhodné otočenie a preklopenie, následne náhodnú pozíciu a tam modul zapíšeme. Často sa stáva že jednotlivé moduly vytŕčajú z hracej plochy, lebo im v tom nijako nebránime. Takéto generovanie má určite svoje nevýhody, ale najlepšie dodržiava náhodnosť algoritmu. Hlavnou nevýhodou je, že nevieme kontrolovať pretŕčajúce moduly, keďže do ohodnocovacích funkcií posielame len hraciu plochu, bez jej okolia.

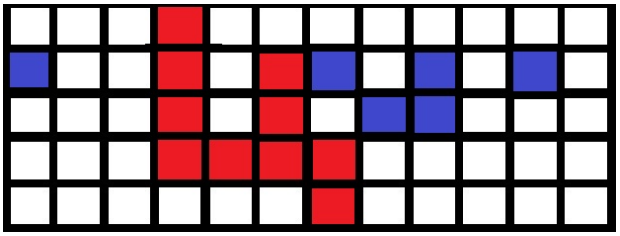
Pri tomto spôsobe generovania susedov, sme vyskúšali dve kombinované funkcie. Prvou z nich bola funkcia, minimalizujúca počty ostrovov a zároveň minimalizujúca veľkosť zle pokrytej plochy. Druhú sme použili funkciu minimalizujúcu obvody ostrovov a veľkosť zle pokrytej plochy.



Obr. 4.6: Porovnanie funkcie minimalizujúcej počty nepokrytých ostrovov a objektívnej ohodnocovacej funkcie

Na Obr. 4.6 môžeme vidieť graf, kde na osi x, sa postupne presúva váha z výsledku ohodnocovacej funkcie, ktorá minimalizuje počty ostrovov, na výsledok objektívnej ohodnocovacej funkcie. Na osi y, môžeme vidieť priemerný výsledok, ktorý sme dosiahli. Na krivke sú znázornené jednotlivé namerané výsledky, aj s chybovosťou, ktorá má veľkosť dva krát štandartnú odchýlku. Ako môžeme vidieť, čím viac sa presúva váha na objektívnu ohodnocovaciu funkciu, tým sa odchýlky zmenšujú. Je to dané hlavne tým, že prvá z dvoch použitých funkcií, nijako neminimalizuje počty ostrovov. Jej k spokojnosti stačí, keď sú nepokryté políčka vtesnané do jedného ostrova, ktorý môže mať ľubovoľnú veľkosť. Rovnako zľava doprava klesá aj počet zle pokrytých hracích políčok. To sme aj predpokladali, ale mysleli sme, že graf porovnávaných funkcií bude mať konvexný tvar. Teda že pri dobrom vyvážení

budú tieto dve funkcie dobre spolupracovať, a budú mať lepšie výsledky ako v ekxtrémnych prípadoch na okrajových bodoch grafu. Test preukázal, že kombinácia týchto dvoch ohodnocovacích funkcií nie je príliš dobrá, lebo pomocná funkcia minimalizujúca počty ostrovov, je príliš slabá.

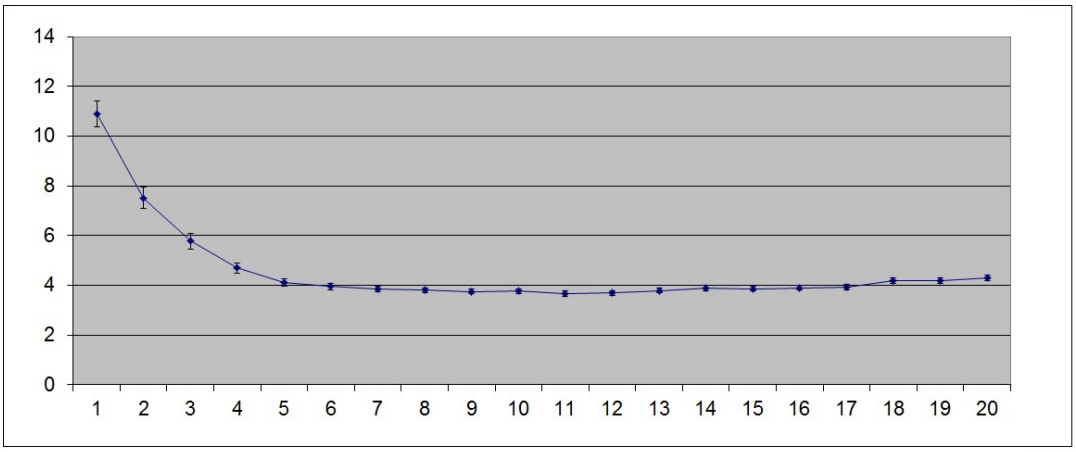


Obr. 4.7: Príklad rozloženia pre funkcie minimalizujúcej počty nepokrytých ostrovov a objektívnej ohodnocovacej funkcie

Na Obr. 4.7 je príklad rozloženia hracej plochy, pri vyvážení 50 na 50 týchto dvoch funkcií. Všimnime si, že na hracej ploche je opäť dlhý kľukatý had, nepokrytých políčok. Viacnásobné prekrytia, sú naopak rozhádzané po hracej ploche náhodne a navyše ich je tam menej ako nepokrytých políčok hracej plochy. Je to kvôli spôsobu akým generujeme susedné konfigurácie. Niektoré moduly vytŕčajú z hracej plochy von. Ohodnocovacie funkcie s týmto faktom nemôžu nič spraviť, lebo sa pozerajú len na hraciu plochu a nie do jej okolia. Rozhádzanie prekrývajúcich sa políčok spôsobuje to, že prvá zo spomínaných funkcií, minimalizuje len počty ostrovov pre nepokryté políčka. Tento obrázok bol vytvorený podobne ako v predchádzajúcej podkapitole.

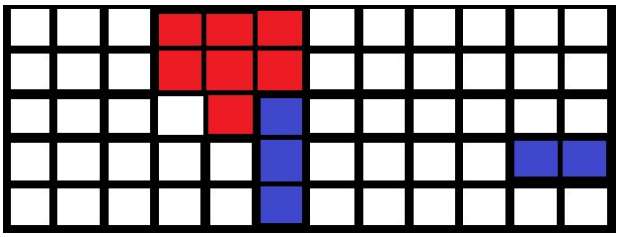
Pozrime sa teraz na druhú kombinovanú funkciu, ktorú sme pri tomto generovaní testovali. Opäť do nej vstupovali dve ohodnocovacie funkcie, jedna minimalizovala obvody funkcií a druhá bola objektívna ohodnocovacia funkcia. Od tejto kombinácie sme očakávali lepšie výsledky ako od predchádzajúcej.

Na obrázku Obr. 4.8 vidíme priebeh funkcie, kde na osi x je najprv najvyššia váha na funkcii minimalizujúcej obvody ostrovov, postupne sa presúva v prospech objektívnej ohodnocovacej funkcie. Na osi y je naznašené, v priemere koľko políčok zostávalo nepokrytých na hracej ploche, po 400 zbehnutiach simulovaného žíhania. Ako môžeme vidieť, opäť je objektívna ohodnocovacia funkcia lepšia ako druhá vstupujúca do kombinovanej funkcie. Na tomto grafe je veľmi pekne vidieť, ako sa postupne znižuje priemerná odchýlka, až je napokon takmer nepatrná.



Obr. 4.8: Porovnanie funkcie minimalizujúcej obvody ostrovov a objektívnej ohodnocovacej funkcie

Toto všetko sme zatiaľ očakávali. Čo nás však pozitívne prekvapilo, je konvexný tvar krivky grafu. Ktorý dokazuje, že okolo vyváženia 50 na 50 táto zložená ohodnocovacia funkcia pracuje lepšie ako pri extrémnych vyváženiach. Tento graf nám zároveň ukazuje, že funkcia minimalizujúca obvody je silnejšia ako funkcia minimalizujúca počty nepokrytých ostrovov. Ako sme naznačili v predchádzajúcej podkapitole, táto funkcia sa zameriava na podobnú vlastnosť problému, ale trošku z iného uhla pohľadu. Navyše sa snaží minimalizovať aj obvod prekrývajúcich sa políčok, čo ešte zvyšuje jej schopnosti. Ďalej si analyzujeme, ako vyzerá hracia plocha pri rýchlejšom chladnutí.



Obr. 4.9: Príklad rozloženia pre funkcie minimalizujúcej obvody ostrovov a objektívnej ohodnocovacej funkcie

Ako môžeme vidieť na Obr. 4.9 počet nepokrytých hracích plošiniek sa znížil a znížil sa aj počet miest kde sa prekrývajú dva alebo viacej hracích modulov. Prvé zníženie má na svedomí objektívna ohodnocovacia funkcia, ktorá sa snaží znižovať počty zle pokrytých políčok. Keby však pracovala sama, boli by tieto políčka porozhadzované na hracej ploche náhodne. To, že sú v sústredené v rozumných útvaroch, spôsobuje zase druhá funkcia. Opäť sa však stáva, že niektoré moduly vytŕčajú z hracej plochy von. V tomto štádiu nemáme riešenie pre daný problém.

Táto funkcia sa však dá, napriek jej problémom, považovať za doteraz najväčší úspech. Vôbec prvý krát sa nám podarilo mať výsledky v priemere lepšie ako 4 zle pokryté hracie políčka. A to aj pri viacerých vyváženiach spomínaných funkcií. Tento výsledok pripisujeme aj spôsobu, ktorým generujeme jednotlivé konfigurácie. Môže totiž nastať prípad, kedy budeme mať na hracej ploche dve nepokryté políčka ale iba jedno políčko také, kde sa prekrývajú hracie moduly. Nezakazujeme totiž vytŕčanie z hracej plochy von, čo môže viesť k miernemu skresľovaniu výsledkov. Kvôli tomuto skresleniu, sme sa rozhodli použiť generovanie bez možnosti vytŕčania.

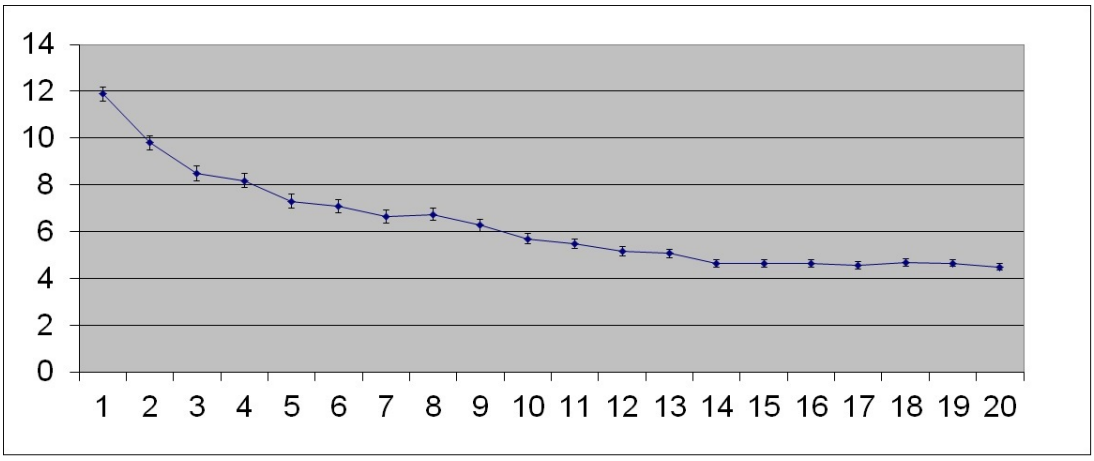
4.3 Generovanie bez možnosti vytŕčania modulov mimo hraciu plochu

Počiatočné inicializovanie a generovanie susedných konfigurácií pracuje podobne ako predchádzajúci spôsob. Rozdielom je, že ak sme pri generovaní konfigurácie chceli zapísať hrací modul tak, aby niektorá jeho časť vytŕčala z hracej plochy von, neurobíme to, ale znova vyberie modul, pre ktorý vygeneruje otočenie, pozíciu kam sa modul opäť pokúsime zapísať. Toto generovanie sa opakuje dovtedy, kým sa mu nepodarí zapísať modul.

Pri tomto spôsobe generovania konfigurácií sme testovali rovnaké funkcie ako pri predchádzajúcom generovaní, ale pridali sme ešte jednu, ktorá kombinuje len dve pomocné funkcie. Funkcia, ktorá penalizuje políčka kde sa moduly prekrývajú, skombinovaná s funkciou, ktorá minimalizuje počty nepokrytých ostrovov. Túto funkciu sme pri predchádzajúcom generovaní neskúšali, lebo prvá z dvoch kombinovaných funkcií spôsobovala, že veľa modulov vytŕčalo z hracej plochy a teda nemala vôbec dobré výsledky. Keď ale zabránime modulom vytŕčať, sila tejto funkcie pochopiteľne narastá.

Poďme sa najprv ale pozrieť, ako na tom boli funkcie z predchádzajúcej podkapitoly. Prvá funkcia, ktorú sme testovali bola opäť zložená tak, aby minimalizovala počty nepokrytých ostrovov a celkovú zle pokrytú plochu.

Na obrázku Obr. 4.10 vidíme podobný graf ako v predchádzajúcej podkapitole. Opäť na osi y, môžeme vidieť priemerný počet zle pokrytých políčok. Na osi x vidíme zmenu váhy, najľavejší vrchol grafu, znázorňuje maximálnu váhu pre počty nepokrytých ostrovov. Najpravejší, váhu na celkovú zle pokrytú plochu. Ako si môžete všimnúť, sila funkcie minimalizujúcej počty ostrovov, sa v podstate nezmenila. Pre porovnanie uvádzame namerané hodnoty z predchádzajúceho generovania. Tam bol priemerný výsledok 11,81 zle pokrytých políčok. Pri súčasnom generovaní bol výsledok 11,875. Teda pozorujeme zhoršenie na

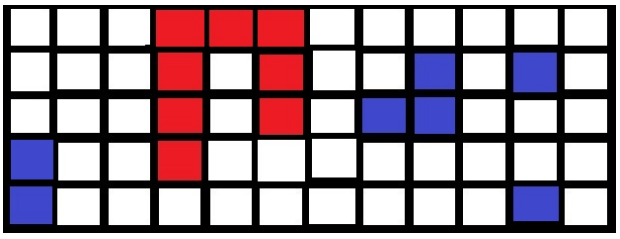


Obr. 4.10: Porovnanie funkcie minimalizujúcej počty nepokrytých ostrovov a objektívnej ohodnocovacej funkcie, pri generovaní bez možnosti vytŕčania

hranici stotín. Keď zoberieme do úvahy, že v predchádzajúcom generovaní sme nevedeli posudzovať koľko modulov vytŕča z hracej plochy, toto zhoršenie je naozaj minimálne.

Porovnajme ďalej s predchádzajúcim generovaním aj ďalej aj druhý extrém v grafe a to je najrpravejší bod. Pri generovaní s možnosťou vytŕčania, sme namerali priemernú hodnotu 4,43 zatiaľ čo na Obr 4.10 je priemerná hodnota 4,49 čo tiež nie je veľké zhoršenie.

Celkovo však funkcia klesá vyrovnanejšie a nie až natoľko lineárne ako pri predchádzajúcom meraní (Obr. 4.6). Výsledky sú síce o niečo horšie ako pri predchádzajúcom generovaní, ale to pripisujeme faktu, že teraz naozaj musia byť všetky časti hracích modulov nejako poukladané v hracej ploche.

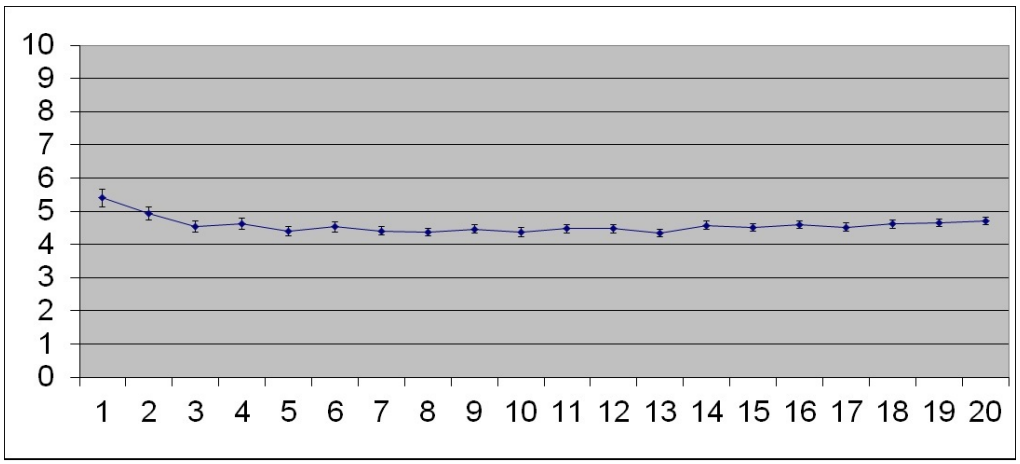


Obr. 4.11: Príklad rozloženia pre funkcie minimalizujúcej počty ostrovov a objektívnej ohodnocovacej funkcie pri generovaní bez možnosti vytŕčania

Pri kratšom čase na výpočet sme znova vytvorili obrázok Obr. 4.11 ilustrujúci skombinovanú ohodnocovaciu funkciu, pri rovnakých váhach na obe ohodnocovacie funkcie, ktoré do nej vstupujú. Môžeme si všimnúť, že oproti obrázku Obr. 4.7, sa nám zvýšil počet prekrývajúcich sa políčok. Aj tak ich je menej ako nepokrytých políčok, ale to je preto, že táto

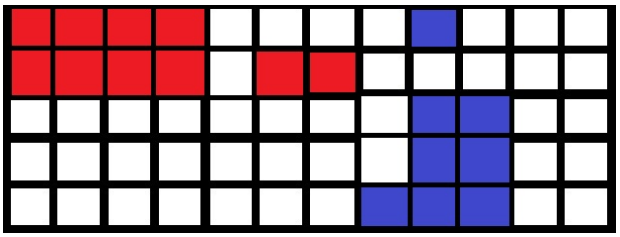
ohodnocovacia funkcia nepenalizuje prekryvy dostatočne na to, aby jednotlivé konfigurácie prinútili znižovať počet modulov zakrývajúcich jedno hracie políčko.

Ďalšiu ohodnocovaciu funkciu, ktorú sme testovali, je funkcia minimalizujúca obvody ostrovou, skombinovaná s objektívnou ohodnocovacou funkciou.



Obr. 4.12: Porovnanie funkcie minimalizujúcej obvody ostrovov a objektívnej ohodnocovacej funkcie, pri generovaní bez možnosti vytŕčania

Na ľavej strane obrázku Obr. 4.12 môžeme vidieť maximálnu váhu pre funkciu minimalizujúcu obvody nepokrytých ostrovov, ktorá sa postupne presúva v prospech objektívnej funkcie. Porovnajme teraz tento obrázok s jeho variantom pre generovanie s možnosťou vytŕčania Obr. 4.8. Ako si môžeme všimnúť, funkcia minimalizujúca obvody ostrovov, je takmer dvojnásobne silnejšia pri tomto generovaní. Ako si môžeme ďalej všimnúť, tieto dve funkcie sú veľmi vyrovnané pri všetkých váhach.



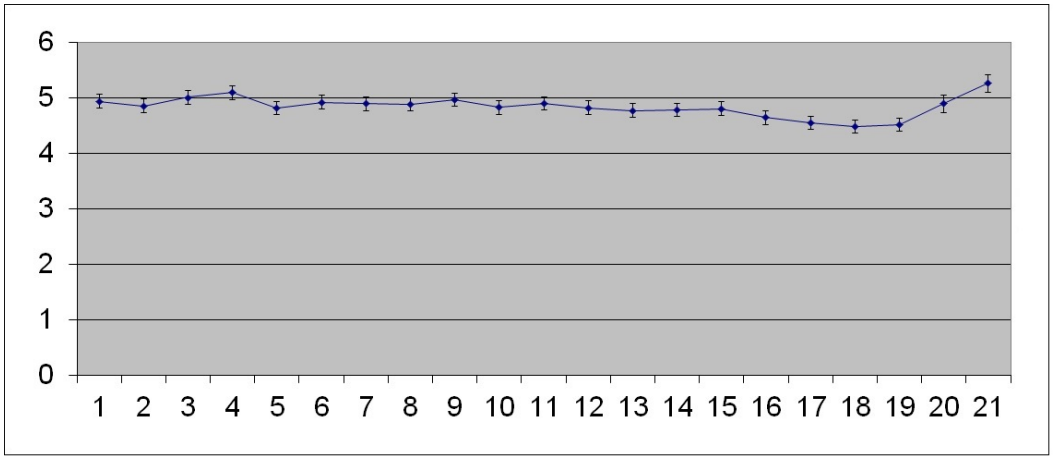
Obr. 4.13: Príklad rozloženia pre funkcie obvody ostrovov a objektívnej ohodnocovacej funkcie pri generovaní bez možnosti vytŕčania

Na príklade hracej plochy si všimnime, že zle pokryté políčka sú usporiadané v peknych útvaroch. Nevieme síce povedať presnú polohu každého z hracích modulov, ale môžeme si všimnúť, že na ploche je jeden veľký modrý ostrov. V ňom sa môže nachádzať dokonca

niekoľko celých hracích modulov. Keby sme chceli pentomína z tohto momentu doriešiť, môžeme mať šťastie a presunutím jedného modulu, získame veľké zlepšenie pokrytia. Stále sa však na ploche nachádzajú zle pokryté políčka, ktoré sú sústredené do menších ostrovov. Tieto sa nám budú pokrývať ťažšie. Ako sme už spomínali v predchádzajúcej podkapitole, funkcia minimalizujúca obvody sa prirodzene snaží minimalizovať aj počet ostrovov. Ale nie je to až tak drastické ako funkcia, ktorá sa na túto vlastnosť hracej plochy sústredí.

Aby sme zosumarizovali tento test, pri tomto generovaní susedných konfigurácií, sa ukázala ozajstná sila obvodovej funkcie. Celkové výsledky síce boli horšie ako pri predchádzajúcom generovaní, ale pozitívny je fakt, že teraz naozaj máme všetky moduly niekde na hracej ploche.

Ďalšia funkcia, ktorú sme testovali vznikla spojením dvoch pomocných funkcií. Prvou z nich bola funkcia penalizujúca prekrývajúce sa políčka a druhou bola upravená funkcia minimalizujúca počty ostrovov. Týmto dvom funkciám sa postupne menil vplyv pomocou váhovania ich výstupov. Navyše sme k nim ešte pridali objektívnu ohodnocovaciu funkciu, aby sme mali pokrytú aj túto vlastnosť. Dôvodom prečo sme neskúšali prvú spomínanú funkciu pri predchádzajúcom generovaní bolo, že vlastnosti, na ktoré sa táto sústredí spôsobovala príliš veľa vytŕčajúcich modulov z hracej plochy, čo sme nechceli. Druhú spomínanú funkciu sme upravili tak, aby penalizovala podobne ako prvá funkcia. Tieto penalty sme museli vyvážiť, aby sme mohli dobre vypočítať graf, s ktorým sa v tejto kapitole často stretávame.



Obr. 4.14: Porovnanie funkcie penalizujúcej prekryvy a funkcie minimalizujúcej počty ostrovov, s doplkom objektívnej ohodnocovacej funkcie pri generovaní bez možnosti vytŕčania

Na obrázku Obr. 4.14 vidíme postupnú zmenu vplyvu vstupných funkcií na výsledky. Najľavejší bod grafu predstavuje maximálny vplyv funkcie minimalizujúcej prekryvy, ktorej vplyv sa postupne znižuje v prospech funkcie minimalizujúcej počty ostrovov. Na osi y

vidíme počet zle pokrytých políčok na hracej ploche. Ako si môžeme všimnúť, tieto dve funkcie spolupracujú veľmi dobre. Dokonca, keď sa dostane dostatočná váha na druhú vstupnú funkciu, môžeme vidieť, že počet zle pokrytých políčok sa zníži až na hranicu 4,54 zle pokrytého políčka v priemere. Čo je najlepší výsledok pre toto generovanie pre všetky použité ohodnocovacie funkcie.

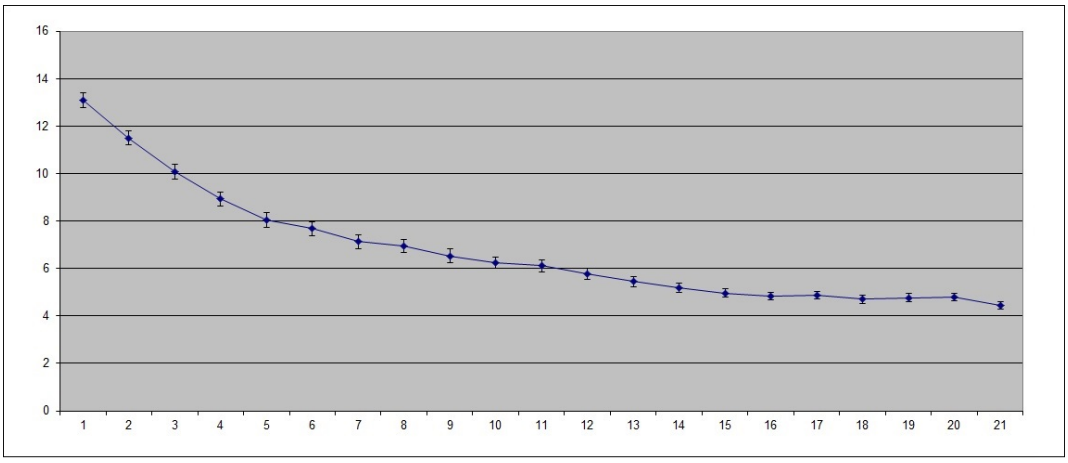
Príklad hracej plochy pre túto hraciu funkciu uvedieme až v ďalšej podkapitole, pretože táto ohodnocovacia funkcia bola mierená najmä na generovanie pomocou princípu hracej ruky.

4.4 Generovanie pomocou hracej ruky

Princíp hracej ruky je bližšie popísaný v kapitole 3. Tu len spomenieme, že okrem toho, že moduly nemôžu vytŕčať z hracej plochy von, máme k dispozícii akési odkladisko. Kam si s určitou pravdepodobnosťou, môžeme moduly odkladať. Tento spôsob generovania veľmi dobre simuluje správanie človeka, ktorý sa snaží pentomína skladať. Často sa stane, že si všimneme nejakú neriešiteľnú konfiguráciu a preto si odložíme niekoľko modulov na ruku a skúsime ich povkladať inak.

V tejto časti nebudeme uvádzať príklady rozložení, pre už známe funkcie, lebo sú veľmi podobné príkladom uvedeným v predchádzajúcej podkapitole.

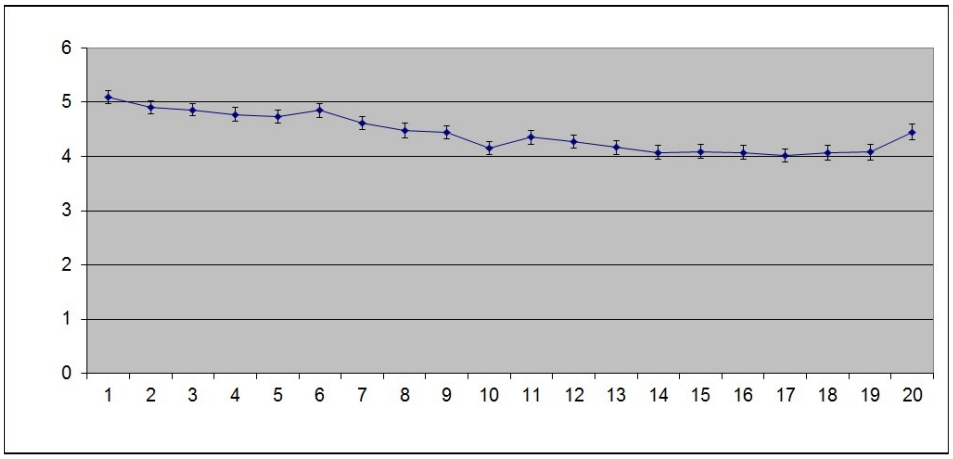
Aj pri tomto generovaní sme najprv testovali kombinovanú funkciu zloženú z funkcie minimalizujúcej počty nepokrytých ostrovov a objektívnej ohodnocovacej funkcie



Obr. 4.15: Porovnanie funkcie minimalizujúcej počty ostrovov a objektívnej ohodnocovacej funkcie, pri použití hracej ruky

Na obrázku Obr. 4.15 máme znázornený priebeh spomínanej ohodnocovacej funkcie. Tradične na osi y vidíme počet zle pokrytých políčok a na osi x vidíme postupne sa meniace váhy. Najľavejší bod grafu znázorňuje maximálnu váhu na ohodnocovacej funkcii, ktorá sa zameriava na minimalizovanie počtu ostrovov. Znova môžeme sledovať, že počty zle pokrytých polí sa znižujú, čím je väčšia váha na funkciu minimalizujúcu zle pokryté políčka na hracej ploche. Najľavejší sak zajtra ak sa neozve tak napisem mail bod grafu má o čosi vyšší priemerný výsledok. Je to spôsobené generovaním. Vo výsledných konfiguráciách sa stávalo, že niekoľko modulov ostalo na ruke, lebo ich ľubovoľným položením do hracej plochy sa zvýšil počet nepokrytých ostrovov. Pre algoritmus teda nastala patová situácia, kedy pri pažravej časti neakceptoval stav s viacerými modulmi na hracej ploche.

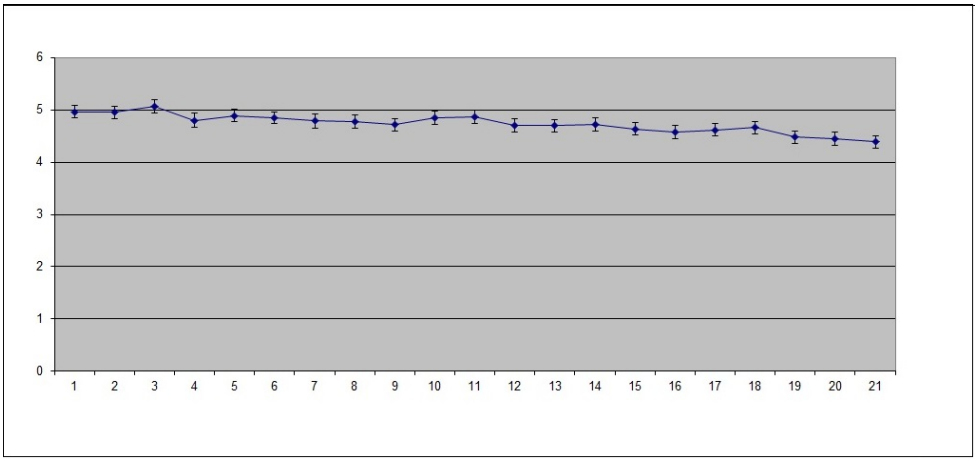
Pri maximálnej váhe na objektívnej ohodnocovacej funkcii, sme dostali veľmi podobné výsledky. Pre toto generovanie to bolo 4,445 a pre generovanie bez možnosti vytŕčania to bolo 4,9 zle pokrytých políčok v priemere. Z toho sme usúdili, že sila objektívnej ohodnocovacej funkcie sa príliš nezmenila, zmenou spôsobu generovania.



Obr. 4.16: Porovnanie funkcie minimalizujúcej obvody zle pokrytých ostrovov a objektívnej ohodnocovacej funkcie, pri použití hracej ruky

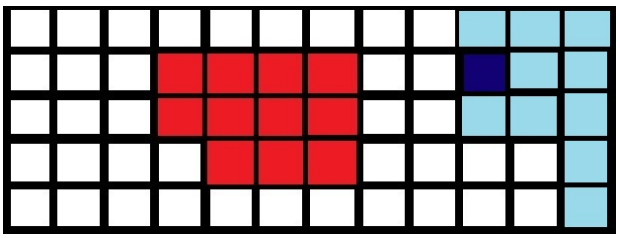
Ďalšia kombinovaná evaluačná funkcia bola zložená z funkcie minimalizujúcej obvody ostrovov a objektívnej ohodnocovacej funkcie. Na Obr. 4.16 môžeme sledovať jej priebeh. Najľavejší bod grafu znova znázorňuje maximálnu váhu na obvodovej funkcii. Znova sa ukazuje, že sila oboch funkcií, je takmer rovnaká. Pri maximálnej váhe na obvody ostrovov bola nameraná priemerná hodnota 5,095 a pri maximalnej váhe na objektívnu ohodnocovaciu funkciu bolo nameraných priemerne 4,575 zle pokrytých políčok. Teda rozdiel medzi týmito dvoma funkciami tvorí niečo málo cez 0,5 pokrytého políčka v priemere.

Keďže sme vedeli akú silu má obvodová funkcia, rozhodli sme sa vyskúšať ju skombinovať s pomocnou funkciou, ktorá tvrdo penalizovala konfiguráciu, ak sa na nej prekrývali dva alebo viacej modulov na jednom mieste. Táto funkcia sa svojim spôsobom podobá objektívnej ohodnocovacej, ale pozerá sa na problém z trocha iného uhla. Preto sme chceli zistiť, či tento uhol pohľadu zlepší výsledky.



Obr. 4.17: Porovnanie funkcie penalizujúcej prekryvy a funkcie minimalizujúcej obvody zle pokrytých ostrovov, pri použití hracej ruky

Na obrázku Obr. 4.17 môžeme sledovať priebeh testu tejto kombinovanej funkcie. V grafe naľavo vidíme maximálnu váhu pre obvodovú funkciu a napravo maximálnu váhu pre penalizujúcu funkciu. Ako môžeme z nameraných hodnôt vidieť, skombinovaním týchto dvoch funkcií, sme sa skoro počas všetkých váh držali nad hranicou 4,5 zle pokrytého políčka v priemere. Čo sú horšie hodnoty ako boli pri predchádzajúcej ohodnocovacej funkcii. Čo je však zaujímavejšie, je výzor plochy pri kratšom čase na žíhanie.

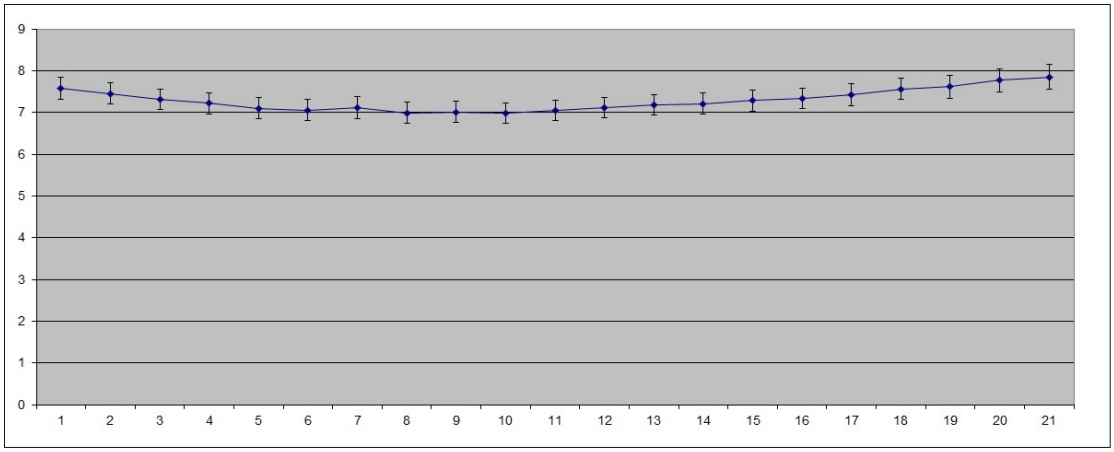


Obr. 4.18: Príklad rozloženia pre funkciu penalizujúcej prekryvy a funkciu minimalizujúcej obvody zle pokrytých ostrovov, pri použití hracej ruky

Na obrázku Obr. 4.18 môžeme sledovať výsledok takejto konfigurácie. Môžeme vidieť, že zle pokryté políčka sú pekne sústredené do dvoch ostrovov. Modrý ostrov, znázorňujúci políčka, kde sa hracie moduly prekrývajú obsahuje jedno tmavšie políčko, ktoré znázorňuje

miesto kde sa naraz prekrývajú dva moduly. Z toho vyplýva, že penalizujúca funkcia dobre funguje už aj pri rýchlejšom chladnutí. V modrom ostrove sa s veľkou pravdepodobnosťou nachádza jeden celý modul, ktorý by sa pri riešení dal presunúť do červeného ostrova, a tým by sme určite vytvorili lepšiu konfiguráciu.

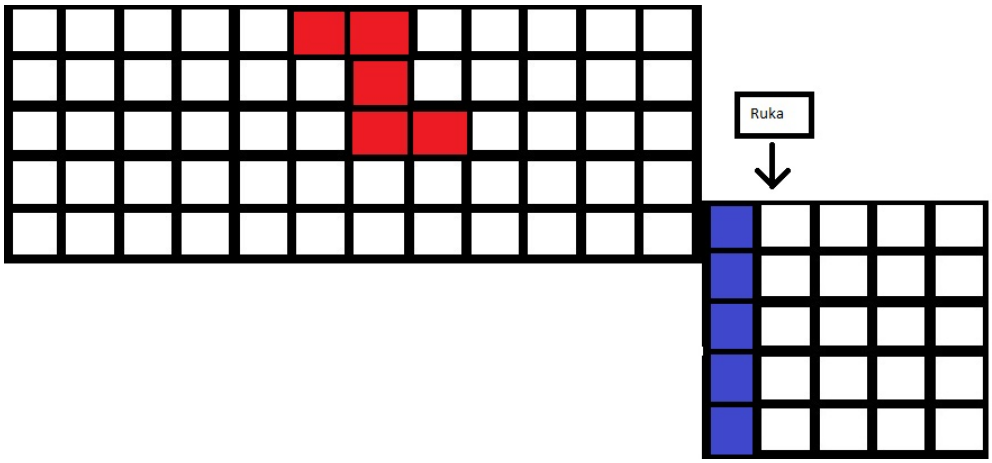
S poslednou funkciou, ktorú sme skúšali, sme sa už stretli v predchádzajúcej kapitole. Je to kombinácia evaluačnej funkcie penalizujúcej prekryvy a funkcie penalizujúcej počty ostrovov, s nemenným vplyvom objektívnej ohodnocovacej funkcie. Pri tomto teste sme ešte trošku upravili funkciu minimalizujúcu počty ostrovov, aby až natoľko nepenalizovala konfiguráciu, kedy má na hracej ploche len jeden ostrov nepokrytých políčok, ktorý má veľkosť 5. Pri tejto funkcii sme chceli dosiahnuť to, aby sme aspoň niekedy mali hraciu plochu dobre pokrytú až na jeden takýto ostrov. Na ruke nám ostane hrací modul, ktorý sa ale nedá vložiť do takéhoto ostrova.



Obr. 4.19: Porovnanie funkcie penalizujúcej prekryvy a funkcie minimalizujúcej počty ostrovov, s doplkom objektívnej ohodnocovacej funkcie pri generovaní s použitím hracej ruky

Na Obr. 4.19 sa zľava doprava postupne présúva váha z funkcie minimalizujúcu políčka kde sa prekrývajú dva, alebo viacej hracích modulov, na funkciu minimalizujúcu počty ostrovov. Tretiu funkciu sme do tejto kombinovanej funkcie zakomponovali kvôli tomu, že keď sme robili testy, kde boli len váhové funkcie, spôsobovalo to, že nám ostávalo množstvo hracích modulov na ruke, čím sme dostávali zlé výsledky. Ako vidno na Obr. 4.19, intervaly chybovosti sa natiahli. Je to tak preto, lebo sa často stáva, že buď na ruke ostane jeden alebo dva hracie moduly. Ďalej za povšimnutie stojí, že obe ohodnocovacie funkcie fungujú lepšie keď spolupracujú, ako keď sú v extrémnej vyváženosti.

Nikdy sa nestalo, že by vo výslednom riešení boli rozložené všetky moduly, za cenu toho, že by sa moduly prekrývali na niektorom políčku. Obrázok Obr. 4.20 znázorňuje výstupnú



Obr. 4.20: Príklad rozloženia funkcie penalizujúcej prekryvy a funkcie minimalizujúcej počty ostrovov, s doplkom objektívnej ohodnocovacej funkcie pri generovaní s použitím hracej ruky

konfiguráciu pre klasické testovacie parametre. Teda počet iterácií while cyklu pre tento obrázok bol 250000 a nie 300 ako v predchádzajúcich podkapitolách. Dôvodom, prečo je tento obrázok iný, je to, že sme chceli poukázať na náš cieľ pre túto ohodnocovaciu funkciu. Ako sme spomínali, chceli sme dosiahnuť konfiguráciu, kedy na ploche máme nepokrytý ostrov tvaru niektorého z hracích modulov a na ruke iný hrací modul. Toto je jeden z troch typov výstupov, ktoré sme namerali. Druhým typom bola konfigurácia, kedy sme tiež síce mali nepokrytých políčok len 5, ale boli rozbité na dva ostrovy a na ruke sme mali jeden hrací modul. Tretí typ bol, keď nám na ruke ostali dva hracie moduly a na hracej ploche sme mali jeden väčší nepokrytý ostrov.

Táto ohodnocovacia funkcia síce nedokázala pentomína vyriešiť, ale k riešeniu sa priblížila asi najbližšie zo všetkých testovaných evaluačných funkcií, napriek tomu, že graf na Obr. 4.19 tomu spočiatku nenasvedčoval. Táto funkcia je náš najväčší úspech pri pokuse riešiť pentomína v tejto práci.

Kapitola 5

Diskusia

Rovnakým problémom ako sme si stanovili my, sa nezaoberal predtým nikto. Poďme teda rozobrať jednotlivé časti tejto práce. Prvou časťou bolo nastavovanie faktorov v simulovanom žíhaní.

Autori [1] , uvádzajú, že používajú Boltzmannov vzorec na výpočet tejto funkcie. Tento vzorec sme použili aj my, lebo sa veľmi dobre správa počas priebehu simulovaného žíhania.

V rovnakom zdroji sa uvádza, že chladnutie musí byť veľmi pomalé. V zdroji je uvedených viacero možností chladnutia. My sme zvolili lineárne chladnutie, lebo sme chceli dosiahnuť, aby mal algoritmus rovnako dlhý čas pre každý typ správania.

Kirkpatrick, Gelatt a Vecchi ďalej píšu, že na zistenie počiatočnej teploty, je potrebné pozorvať správanie algoritmu pri špecifickom probléme. A na základe tohto pozorovania nastavovať počiatočnú teplotu. Na základe výsledkov v kapitole 3.3 Stanovovanie počiatočnej teploty, sme sa rozhodli používať vlastnú funkciu rozžíhavania. Túto funkciu sme použili hlavne kvôli tomu, že sme sa chystali testovať veľa funkcií a generovaní. Ak by sme pre každú z týchto ešte špecificky sledovali správanie behu kvôli získaniu presnejšej hodnoty, stratili by sme až príliš veľa času. Preto sme sa zhodli, že metóda rozžíhavanie síce nezisťuje úplne ideálnu počiatočnú teplotu, ale zaručuje nám aspoň postačujúcu teplotu. Vzhľadom na to, že stanovovanie počiatočnej teploty nebolo cieľom tejto práce, dospeli sme k rozhodnutiu, že metóda rozžíhania bude postačujúca.

Čo sa týka pentomín, zvolili sme klasické pentomína, bez modifikovaných hracích modulov. Hraciu plochu sme zvolili s rozmermi 5x12. Obe tieto rozhodnutia nám poskytujú 1010 správnych konfigurácií. Medzi všetkými modifikáciami pentomín to nie je najviac, ale tento počet je dostatočne veľký. Pre porovnanie, pre hraciu plochu s rozmerimi 6x10 existuje 2339 správnych konfigurácií a pre rozmery 4x15 len 368.

Kapitola 6

Záver

V tejto práci sme museli najprv zastabilizovať niektoré faktory simulovaného žíhania, aby sme si vytvorili stabilné pracovné prostredie. Na výpočet pravdepodobnostnej funkcie sme použili Boltzmannov vzorec, ktorý sa tradične využíva pri simulovanom žíhaní. Ďalej sme vytvorili metódu rozžíhavania, ktorou vieme aproximovať správnu počiatočnú teplotu, z ktorej algoritmus začína. Chladnutie algoritmu sme zvolili lineárne lebo je postačujúce na náš výskum. Vytvorili sme tri spôsoby generovania susedných konfigurácií. Vytvorili sme aj päť evaluačných funkcií, ktoré sme testovali. Ďalej sme vymysleli spôsob ako jednoducho kombinovať jednotlivé evaluačné funkcie a tým zároveň získavať prehľad o ich spolupráci. Kombinovaním týchto funkcií sa nám rozšíril pracovný priestor. Napriek viacerým testovaniam, sa nám ani raz nepodarilo úspešne zložiť kompletné pentomína. Pri jednom z posledných testov sa však podarilo usporiadať hracie kocky tak, že nám ostala nepokrytá súvislá plocha susediacich políčok, a na ruke sme držali pentomíno, ktoré sa však do tejto plochy nedalo vložiť. V podstate celá hracia plocha bola úspešne pokrytá až na 5 hracích políčok, ktoré sme nevedeli pokryť.

6.1 Do budúcnosti

Tento problém by sa podľa nás dal simulovaným žíhaním vyriešiť. Bolo by treba viacej skúmať faktory ako pravdepodobnostná funkcia, zisťovanie počiatočnej teploty a pokles teploty. Tieto tri faktory môžu mať na žíhanie nesmierny vplyv, ale zatiaľ sme sa nimi nezaoberali. Ďalej by sa dali možno vymyslieť aj iné ohodnocovacie funkcie, prípadne spôsob ako kombinovať viacero funkcií naraz. Kombinovanie nám však spôsobuje zvýšenie časovej náročnosti behu algoritmu. Aj to bolo dôvodom prečo sme kombinovali vždy len dve ohodnocovacie funkcie.

Literatúra

[1] S. Kirkpatrick; C. D. Gelatt; M. P. Vecchi. *Optimization by Simulated Annealing*. Science, Volume 220, Issue 4598, pp. 671-680, 1983.

[2] Vlado Černý. *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*. Journal of Optimization Theory and Applications, Volume 45, pp. 41-51, 1985.

[3] Solomon W. Golomb. *Checker Boards and Polynominoes*. The American Mathematical Monthly Vol. 61, No. 10 (Dec., 1954), pp. 675-682, 1954.

[4] Hilarie K. Orman. *Pentominoes: A First Player Win*. Games of No Chance MSRI Publications, Vol. 29, 1996, 1996.