Explaining and Improving Formula-Represented Heuristic Functions in Grid Pathfinding

by

Shuwei Wang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

© Shuwei Wang, 2024

Abstract

Heuristic functions substantially influence heuristic search performance. Recent work used program synthesis to produce high-performance formula-based heuristics, offering a promise of human explanability. In this thesis we investigate the promise and present a tool to improve a given heuristic function while visualizing how each subformula computes heuristic values on video-game-style maps. The visualization algorithm decomposes a heuristic formula into subformulae and associates each with a region of the map, aiding a human expert to modify the heuristic formula to improve search performance.

We introduce our approach by applying it in three simple problem instances. To expand upon them, we study the approach in two adversarial settings. First, a video-game map is fixed but once the heuristic formula is improved by a human user for a given pathfinding problem, another problem is selected so that the just improved heuristic provides poor guidance on it. The cycle repeats, improving a heuristic for various pathfinding problems on the given map. The second setting generalizes the cycle across video-game maps by selecting a new map on which the user-improved heuristic provides poor guidance. The cycle repeats, improving a heuristic for several maps. We demonstrate from the adversarial case studies that this approach can improve existing heuristics with respect to both guiding performance and explanability for actual video-game maps. Furthermore, we demonstrate that our approach is able to generate an explainable heuristic formula that works well in general across several actual video-game maps in the adversarial setting.

Note that our visualization is more effective when subformulae are short, since the user is more likely to readily understand behavior of formulae with shorter subformulae. Thus we make another contribution by modifying an existing algorithm for heuristic synthesis to generate formulae with short subformulae. We empirically show that imposing an upper bound on subformula size does not degrade search performance of synthesized heuristics.

To wrap up, we propose a visualization tool to explain synthesized heuristic formulae on video-game maps. We show examples of applying the tool to improve existing heuristic formulae. Furthermore, we show empirical results of synthesizing heuristic formulae with constrained subformula sizes and show that the constraints impose less effect on average for problems with varying goal locations than for problems with a shared goal location, and in general such constraints do not hurt heuristic's guiding performance while improving their explanability.

Preface

Some of the research conducted for this thesis forms part of an international research collaboration, led by Professor Vadim Bulitko at the University of Alberta, with Professor William Yeoh being the collaborator at Washington University in St. Louis. The genetic programming method referred to in chapter 6 was designed by myself.

Parts of chapters 4, 5 and 6 of this thesis has been published as S. Wang, V. Bulitko and W. Yeoh, "Explaining Synthesized Pathfinding Heuristics via Iterative Visualization and Modification," in Proceedings of the annual IEEE Conference on Games (CoG), 2024 pp. 1-4. I was responsible for the case studies and the computational experiments as well as the manuscript composition. Vadim Bulitko was the supervisory author and was involved with concept formation and manuscript composition.

“If you would thoroughly know anything, teach it to others.”

—— Tryon Edwards

Acknowledgements

Throughout my degree I have received tremendous support and advice from my supervisor Professor Vadim Bulitko. He has time and time again proven to be very knowledgable, patient and generous. I could not have reached where I am now without Professor Bulitko's help. He has patiently observed my research interests and introduced me to numerous connections for the betterment of my knowledge and research collaboration opportunities. The research shown in this thesis is the result of collaboration between Professor Bulitko and myself. The term we is used to reflect this fact.

I would like to thank my mother Shuqiao Wang and my father Dequan Wang. They have dedicated tremendous efforts supporting my pursue of the degree and celebrated my every step along the way. They have been the pillars supporting my life over the decades and where I stand now owes completely to them.

I appreciate funding from the Natural Sciences and Engineering Research Council of Canada and the computational resources from the Digital Research Alliance of Canada.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Video-game pathfinding is a common testbed for heuristic search (Bulitko 2020, Wang *et al.* 2024, Saunders *et al.* 2024, Bulitko *et al.* 2011). Performance of heuristic search substantially depends on quality of the heuristic function used. Recent work attempted to combine the simplicity, cross-map portability and human-readability of formula-represented heuristics such as Manhattan distance and the high performance of memory-based heuristics (Sturtevant *et al.* 2009). To do so researchers used program synthesis in a space of formulae defined by a context-free grammar (Bulitko *et al.* 2022, Wang *et al.* 2023, Bulitko 2020, Bulitko *et al.* 2021, Hernandez and Bulitko 2021). The resulting formulae were better than weighted Manhattan distance and still compact. Their readability gave a promise of explainability which was not, however, studied beyond a single example (the wall-hugging heuristic). In this thesis we explore explainability of automatically synthesized formula-represented heuristics.

In doing so we make the following three **contributions**. First, we propose a new approach for explaining a heuristic based on an automatic decomposition of a heuristic formula and associating each component with a region of a video-game map. We present it as a tool that allows a game-AI developer or a heuristic researcher to iteratively modify formula-based heuristics and visualize the effects of the modifications. Such a heuristic modification playground allows a human to gain understanding and therefore trust of computer-synthesized heuristics which can be important for heuristic deployment in a game to ship. Second, we integrate the visualization tool into a human-in-the-loop procedure to improve a synthesized heuristic formula for search problems on a video-game map. The procedure selects and visualizes the problem that causes the current heuristic to guide A* search the poorest compared to the weighted Manhattan Distance and the human user iteratively makes improvements to the formula to make it more suitable for this adversarial problem. We demonstrate several case studies with this iterative procedure. As an extension, we allow the user the option to call a light-weight genetic algorithm to improve the current heuristic formula in the process thus forming a collaboration between the human user and the machine. We identify reasons of failed iterative improvement cases and propose the mitigation of applying a constraint on subformulae size of synthesized formulae. Third, we constrain heuristic synthesis to produce heuristic formulae which are more susceptible to explain via our approach and empirically show that such a constrain in general does not decrease search performance of synthesized formulae. We show through another case study that formulae with shorter subformulae are more conducive to the iterative improvement.

1.1 An Illustrative Example

We begin with an example to illustrate how our approach works. Consider the following heuristic synthesized for a 2D grid-based video-game map (Bulitko *et al.* 2022):

$$h = \Delta y + 44.9 \cdot \max(\Delta x, \Delta y) \tag{1.1}$$

where (x, y) is the map cell whose distance to the goal $(x_{\text{goal}}, y_{\text{goal}})$ is estimated by the heuristic $h(x, y, x_{\text{goal}}, y_{\text{goal}})$. Here $\Delta x = |x - x_{\text{goal}}|$ and $\Delta y = |y - y_{\text{goal}}|$.

Our approach first converts h into a conditional formula h' in style of the C language's ternary operator:

$$h' = \Delta y + 44.9 \cdot (\Delta x > \Delta y \; ? \; \Delta x \; : \; \Delta y) \tag{1.2}$$

Then the converted formula h' is decomposed into two subformulae $h_1 = \Delta y + 44.9 \cdot \Delta x$ and $h_2 = \Delta y + 44.9 \cdot \Delta y$ whose application is controlled by the condition $c = \Delta x > \Delta y$ such that for any cell (x, y) on the map, when c is evaluated as true, h_1 is used to compute the heuristic value for (x, y), otherwise h_2 is used.

The condition c effectively partitions the map at which h is applied into two regions where the first region's heuristic values are solely computed by h_1 and the second by h_2. Figure 1.1 shows an example of the two regions for a small grid map when h is applied to the search problem shown on it. In Figure 1.1, white squares are impassable grid cells (i.e., the walls) and other grid cells are colored distinctly according to their regions. The cell where the star symbol lies is the start state and the one where the diamond symbol lies is the goal state.

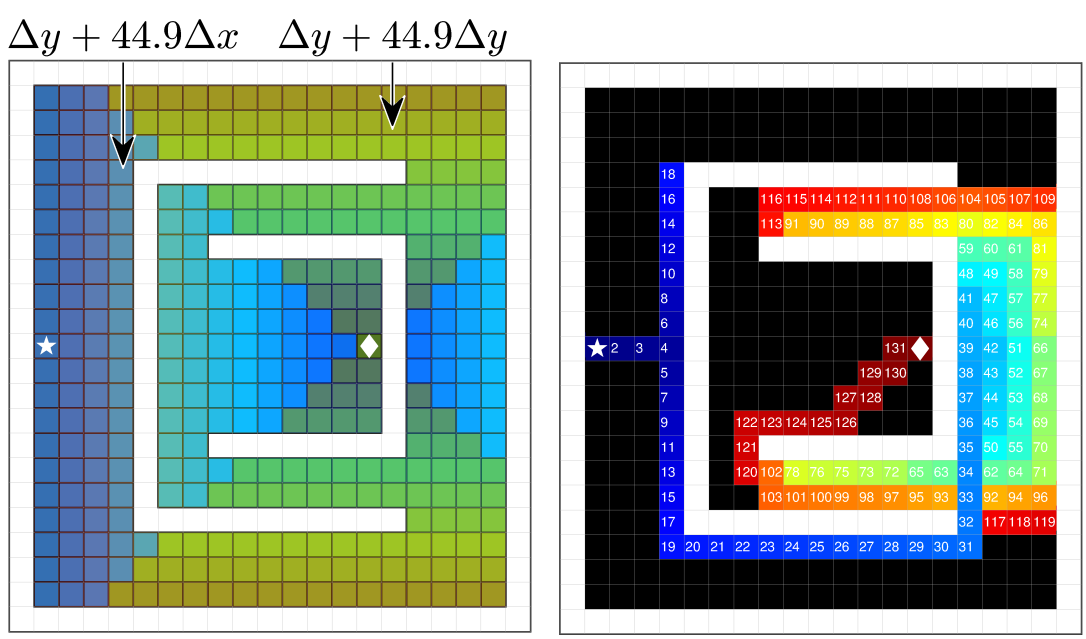

Figure 1.1: The left plot shows the two subformulae, their regions and h gradient. The right plot shows the closed list with the number of expansion order marked on top of each state.

Each grid cell within a region is shown with a color gradient indicating the direction of search guided by the heuristic values. In order to trace the search one needs to compute $f = g + h$ which requires the per-problem g table. Thus by picking a start state one can visualize how A* will expand its open list for that problem.

With the visualization, the user is informed of how the heuristic formula guides the search and can thus modify it to improve its *speedup* on the given search problem. For instance, observe from the figure on the right that many nodes of the closed list are clustered on the right of the smaller backward C-shape wall. From the figure on the left, one can observe that the center of the cluster of the nodes are on top of the blue region governed by the sub-formula $\Delta y + 44.9\Delta x$. One way to improve the heuristic formula so that it causes less nodes expansions is by increasing the heuristic values in that region to make it less attractive to search: modify h' to become $\Delta y + 44.9 \cdot (\Delta x > \Delta y \; ? \; 2\Delta x \; : \; \Delta y)$.

1.2 Thesis Organization

In Chapter 2 we set up the problem of explaining synthesized heuristic formulae. We first introduce the single-agent pathfinding problem and introduce A* search. Second we describe the problem of heuristic formula synthesis problem as an optimization problem with respect to how much better the synthesized heuristic is compared to the weighted Manhattan Distance. The aforementioned points were addressed by Bulitko *et al.* 2022. Finally as a new contribution of this thesis, we go over the measure of heuristic improvement and our notion of heuristic explanability.

In Chapter 3 we review the related works. We go over works related to automatic search algorithm selection and formula synthesis for search and their attempt to explain the automatically composed formulae. We then survey a work that utilizes pre-trained large language models to explain written programs and explain why such method does not fit our requirement for heuristic formula explanation. Finally we move on to automatic symbolic decomposition for formula explanation and elaborate on how our method differ from the previous work.

In Chapter 4 we describe our approach in detail. We start with how to simplify the formulae and describe in pseudocode how to decompose a given formula based on the branching operators. The approach is outlined by Wang *et al.* 2024, but this thesis describes the steps in more details, showing a table of simplification rules we used and providing pseudocode for the main steps. We then describe our visualizations and set up the notion of explanation-conducive formulae. In the end, we describe the idea of extending the current formula space (a grammar) with the conditional operator as a background for the experiment in Chapter 6. This is also presented by Wang *et al.* 2024 but we detail the two grammars formally in the thesis.

In Chapter 5 we first go through some simple examples explaining how the iterative heuristic formula improvement works (Wang *et al.* 2024), then move on to apply the method to improve previously synthesized heuristics for their corresponding video-game maps (a new contribution of this thesis). Last but not least, we extend the method to improve a heuristic to make it work well across several video-game maps (another new contribution of this thesis) and show how the human user can collaborate with the synthesizer as an assistant. We identify possible reasons that causes certain cases to success and others to fail and propose a mitigation.

In Chapter 6 we answer two research questions: "If a constraint is imposed on subformulae size, will guiding performance decrease?" and "Under what circumstance will the extended grammar produce heuristics with better quality than ones produced by the original grammar?" by comparing synthesized heuristics' guiding performance of varied subformula size constraints combined with the two formula spaces (defined by grammars). The formula synthesis algorithm was adopted from the work by Wang *et al.* 2023. The results are from the published work (Wang *et al.* 2024). We show through another case study that formulae synthesized with constraint subformula sizes are more conducive to iterative improvement.

In Chapter 7 we propose future research directions. In Chapter 8 we conclude the thesis by summarizing the contributions and results.

Chapter 2

Problem Formulation

In this chapter we formulate the pathfinding problem and describe the A* search algorithm. We then formulate the heuristic formula synthesis problem as an optimization problem and then move on to introduce heuristic improvement and the notion of heuristic explanability.

2.1 Background: Pathfinding

We adapt the problem setting by Bulitko *et al.* 2022 and reproduce it here for the reader's convenience.

A *pathfinding problem* p is defined by a tuple $(G, s_{\text{start}}, s_{\text{goal}})$ where $G = (S, E, c)$ is the search graph composed of the set of states S, the set of edges E and a cost function $c : E \to \mathbb{R}$. The states s_{start} and s_{goal} are the start and the goal states respectively. A state $s_i \in S$ is connected to its neighbor $s_j \in S$ by the edge $(s_i, s_j) \in E$ with a cost $c(s_i, s_j) > 0$. The *neighborhood* of state s is $N(s) = \{s' \mid (s, s') \in E\}$. A state is *expanded* by a heuristic search algorithm when its $N(s)$ is computed. A solution to a graph search problem is a path $(s_{\text{start}}, s_1, \ldots, s_{\text{goal}})$ such that every pair $(s_i, s_{i+1}) \in E$. The sum of all edge costs on a solution path is the *solution cost*. We define the solution cost function $C(h, p)$ that computes the solution cost of running A* on problem p using heuristic function h. We denote the optimal solution cost (length of the shortest path from the start state to the goal state) of a given problem p as $C^*(p)$.

The *solution optimality* is the ratio of the solution cost to the lowest possible solution cost for a given pathfinding problem p:

$$\alpha(h,p) = \frac{C(h,p)}{C^*(p)} \tag{2.1}$$

with h a given heuristic function. For instance, $\alpha(h,p) = 1$ indicates that an optimal solution was found on problem p with heuristic function h and $\alpha(h,p) = 2$ indicates that a solution twice as costly as the optimal was found.

Indeed for any admissible heuristic h^{adm}, we have $C(h^{\text{adm}},p) = C^*(p)$. However, due to Chen and Sturtevant 2021, we focus on optimizing for search speed instead of solution cost. Solution optimality is introduced so that we can make a fair comparison between synthesized heuristics and the weighted Manhattan Distance with appropriate weights as elaborated in the next section.

2.2 Background: Heuristic Synthesis

A *heuristic synthesis* problem is to find a heuristic formula h from a space of heuristic formulae H in order to maximize the expansion speedup of a given search algorithm on a set of problems. Formally, we intend to approximate the optimal heuristic h_{max} satisfying the expression below:

$$h_{\text{max}} = \arg\max_{h \in H} \sigma(h,P). \tag{2.2}$$

Here H is a space of heuristic functions, P is a set of pathfinding problems, and σ is the speedup function (Bulitko and Lawrence 2023). *Speedup* over the baseline was used in previous works as the performance measure for synthesized heuristics. The definition of *speedup* is reproduced below for the reader's convenience:

$$\sigma(h,P) = \frac{1}{|P|}\sum_{i=1}^{|P|} \sigma(h,p_i),\ \ p_i \in P. \tag{2.3}$$

Here h is a given heuristic function, P is a set of pathfinding problems and $p_i \in P$ is the i'th problem in the set P. The speedup of h over P is the average speedup over each problem in P. The speedup of h over a single pathfinding problem is defined as following:

$$\sigma(h,p) = \frac{\varepsilon(h_{\text{baseline}},p)}{\varepsilon(h,p)},\ p \in P. \tag{2.4}$$

where $\varepsilon(h,p)$ is the number of nodes expanded by A* with the heuristic function h on the search problem p. In line with previous works (Bulitko *et al.* 2022, Bulitko *et al.* 2021, Hernandez and Bulitko 2021, Bulitko and Lawrence 2023), we define $h_{\text{baseline}} = w(\Delta x + \Delta y)$ where the weight $w \in W = \{1.0, 1.1, \ldots, 1.9, 2.0, 3.0, \ldots, 10.0\}$ is chosen such that $\alpha(w_i(\Delta x + \Delta y), p) \leq \alpha(h,p) < \alpha(w_{i+1}(\Delta x + \Delta y), p)$ for $w_i, w_{i+1} \in W$ with α being the suboptimality measure defined in the previous section.

2.3 Heuristic Improvement and Explainability

Since this thesis focuses on iterative improvements made by a human user on a given heuristic formula, we define an improvement to be a reduction in the number of nodes expanded on a given search problem. When there are multiple pathfinding problems present, we defined an improvement to be the amount of increase in average speedups.

Formally, given a heuristic function h and a set of search problems P on which h is applied. Suppose that later h is modified to become h' by the user, we define *heuristic improvement* as

$$\kappa(h,h',P) = \begin{cases} \varepsilon(h,p) - \varepsilon(h',p) & |P| = 1 \\ \frac{1}{|P|}\sum_{p\in P}\sigma(h',p) - \sigma(h,p) & \textit{otherwise} \end{cases}. \tag{2.5}$$

When there is a single problem, we talk about the number of state expansions resulted from the user's improvement. When multiple problems are used, we talk about the speedup increase resulted from the user's improvements. If $\kappa(h,h',P) > 0$, on average h' is better than h on the set of search problems P. Otherwise the improvement fails.

A heuristic function synthesized on problems of a map can be applied to any set of search problems. Bulitko *et al.* 2022 showed that the synthesized heuristics achieve good performance on maps similar to the ones they are synthesized for. Often when a heuristic function is modified so that its quality improves for a single problem, its quality also improves for a series of similar search problems. Later during the iterative improvement process in case studies in Chapter 5, for simplicity, a single search problem is shown to the user to make improvements for the heuristic formula.

While in general explanability of AI is a broad and active research area we restrict ourselves to the following simple concept. We say that a shorter heuristic formula is more explainable than a longer one. Given an AI-generated heuristic it is important for a game-AI developer or a heuristic search researcher to understand its operation and trust that it will generally guide a search algorithm better than a baseline heuristic. So we say that a human user's understanding of a formula-based heuristic function is evidenced by the user's ability to improve it.

Chapter 3

Related Work

In this chapter we begin with the application of machine learning for search and explain the need for . We then move on to the works on formula synthesis. We then introduce the work that uses pre-trained large language models to explain written programs and end with the work of explanation by formula decomposition, on which we drew inspiration from for our heuristic explanation method.

3.1 Comparing Search Method Selection and Formula Synthesis

Leveraging deep learning, one can train a model to select suitable search methods given search problems. Sigurdson *et al.* 2019 were the first to leveraged trained neural networks in per-problem search method selection, followed by Kaduri *et al.* 2020 whose focus is on selecting from a portfolio of optimal search algorithms for multi-agent pathfinding.

Ren *et al.* 2021 added single-agent shortest path instance embeddings to the model and a more fine-grained representation of agent positions and goal positions compared to the work by Sigurdson *et al.* 2019. However Alkazzi *et al.* 2022 later proposed a simpler method that results in faster search.

Since the algorithm selectors implemented by the previous works are all deep learning models, it is unclear the reasoning applied behind the decisions made by the selectors. Furthermore, the selectors choose from a portfolio of existing algorithms, but there might be new algorithms more suitable for some problems instances than others. To facilitate human-machine collaboration, we consider formula synthesizers instead of algorithm selectors because synthesized formula can be better reasoned with than deep-learning networks and it is possible that machine can invent better algorithms through the synthesized formulae that previously do not exist. Therefore in this thesis we only consider formula synthesis.

3.2 Formula Synthesis for Search

In the domain of multi-agent pathfinding, Zhang *et al.* 2022 trained support vector machines that takes hand-crafted features to compute priority scores for each agent and then run prioritized planning (Silver 2005) to find a solution. There were 26 hand-crafted features and a trained support vector machine attached a learned weight to each of them and returned the sum as an agent's priority score. But this makes it hard to analyze the principles behind the computation of priority scores thus runs counter to our purpose of having produced results explainable.

On the other hand, Wang *et al.* 2023 synthesized algebraic formulae to compute the priority scores. The formula space allows the usage of the same hand-crafted features from the work by Zhang *et al.* 2022 and it is shown that the synthesized formulae out-perform the trained support vector machines under certain conditions while requiring less data. Furthermore, the synthesized formulae can be decomposed to explain search behaviors induced by the subformulae. However the decomposition was done manually. However in this thesis we pursue an automatic decomposition of formulae to reduce human effort.

In the domain of single-agent pathfinding, previous works on synthesizing heuristic formulae attempted to explain how formulae work. Bulitko *et al.* 2022 synthesized formulae for A* search on grid maps and Bulitko 2020 synthesized formulae to compute initial heuristic for agent-centered real-time search. They both provided visual narrative explanations of a synthesized heuristic. However we prefer a purely visual-based explanation that captures how heuristic values were computed for each region of the map and we prefer this to be done automatically.

3.3 Pretrained Large Language Model for Synthesized Code Explanation

Bashir *et al.* 2023 proposed to employ a pre-trained large language model to explain synthesized code. A measure is proposed on how explainable a given code fragment is. Their approach works as follows. The user prompts the model to give a textual explanation on a given code fragment then the textual explanation is given to another instance of the large language model to reconstruct a new code fragment. The original and the new fragments' behaviors are compared. The more similar their behaviours are, the more explainable the original code fragment is.

We have applied the method for explaining synthesized heuristic formulae. However, the model either outputs vague statements that applies to any heuristic function or gives inaccurate statements about the nature of the given formula. When transporting the generated explanations to another instance of the language model, the reconstructed formula were almost always slight variants of the Manhattan Distance or the Chebyshev Distance. The prompts used are listed in Appendix A. The results compels us to come up with another way to explain the synthesized heuristic formulae.

3.4 Symbolic Decomposition for Explanation

In the domain of reinforcement learning, Lyu *et al.* 2019 combined deep neural networks with symbolic planning for a more data-efficient and faster-converging interpretable framework for video game domains that includes delayed rewards. In particular, the deep neural networks were combined with reinforcement learning to learn low-level actions for subtasks and the rewards were used to constrain the symbolic planner's search space for a new set of subtasks. Thus the symbolic planner and the deep reinforcement learning controller benefit each other for more efficient learning.

Our work is similar in the sense that the heuristic formulae can also be decomposed into subformulae and the human expert can thus modify the parts of the heuristics to make it more efficient for search problems at hand. However the heuristic formulae are generated using program synthesis techniques and do not need domain specific knowledge as the previous work.

Similarly our automatic decomposition of synthesized formulae is inspired by decomposing a problem specification in sub-specifications as in the general field of program synthesis (Nazari *et al.* 2023). Finally the visualization part of our approach is inspired by the previous work that visualized agent movement in multi-agent pathfinding (Almagor and Lahijanian 2020) in the sense that the start and goal cells are labeled and the search behavior is visualized.

Chapter 4

Proposed Approach

In this chapter we describe our approach to explain and to make improvements upon a given heuristic formula. We first describe it informally with an intuitive example in Section 4.1 and then describe the algorithm in detail in Section 4.2. We introduce the notion of explanation-friendly formula in Section 4.3. We motivate the extended grammar used for synthesis in Section 4.4. In the end, we present a way to utilize our explanation approach for a user to improve a given heuristic formula for a set of pathfinding problems with an intuitive example in Section 4.5.

4.1 Intuition and an Example

We propose to decompose a synthesized formula into subformulae and visually associate each subformula with a map region. Given a synthesized formula, our method to explain it proceeds as shown in Algorithm 1. We illustrate the steps with a simple running example.

Our algorithm simplifies the input synthesized formula ψ using a set of hand-coded simplification rules (line 1 in Algorithm 1). To illustrate, consider a synthesized formula $\psi(x, y, x_g, y_g) = (\min(\Delta x, \Delta y) - (-\Delta x))^2$ which estimates the remaining travel cost from the grid cell (x, y) to the goal grid cell $(x_{\text{goal}}, y_{\text{goal}})$. Here $\Delta x = |x - x_{\text{goal}}|$ and $\Delta y = |y - y_{\text{goal}}|$. The formula ψ is simplified to an equivalent formula $\widehat{\psi} = (\min(\Delta x, \Delta y) + \Delta x)^2$.

Then the algorithm converts all max and min operators in $\widehat{\psi}$ to the equi-

Algorithm 1: Heuristic Explanation

input: synthesized formula ψ a goal state G

1 $\widehat{\psi} \leftarrow \texttt{SIMPLIFY}(\psi)$

2 $\widehat{\psi}' \leftarrow \texttt{MAXMIN2IF}(\widehat{\psi})$

3 $\theta, \mathcal{H}, I \leftarrow \texttt{DECOMPOSE}(\widehat{\psi}', \{0\})$

4 $\texttt{VISUALIZE}(\theta, \mathcal{H}, G)$

valent conditionals in C language's ternary notation (line 2):

$$\max(\varphi_1, \varphi_2) = \varphi_1 > \varphi_2 \ ?\ \varphi_1 : \varphi_2 \tag{4.1}$$

$$\min(\varphi_1, \varphi_2) = \varphi_1 < \varphi_2 \ ?\ \varphi_1 : \varphi_2 \tag{4.2}$$

By doing so the algorithm derives an equivalent formula:

$$\widehat{\psi}' = ((\Delta x < \Delta y \ ?\ \Delta x : \Delta y) + \Delta x)^2 \tag{4.3}$$

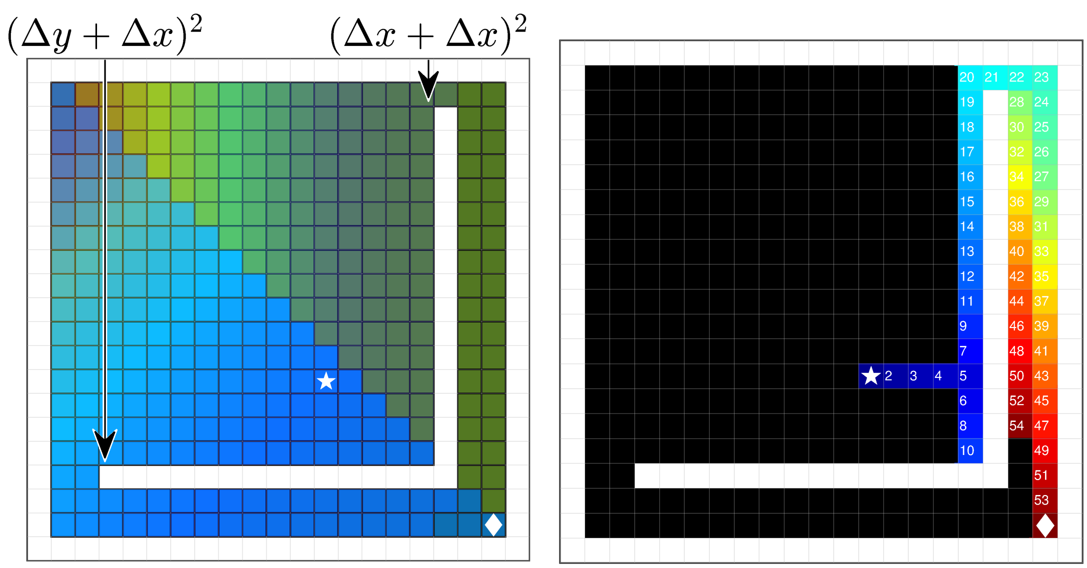

Figure 4.1: Decomposition of heuristic function $(\min(\Delta x, \Delta y) - (-\Delta x))^2$ over a simple corner map. White grid cells are impassable obstacles. The goal state is marked with a diamond and the start state a star. Starting from the start state we obtain a sequence of states expanded by A* forming the closed list. Their expansion order is indicated with numbers and colors.

The next step (line 3) of the algorithm decomposes the formula $\widehat{\psi}'$ into subformulae and associates each subformula with a map region. The subformulae and the associated regions are constructed by the algorithm in such a way that within each region R_i the original heuristic value ψ is computed solely by the corresponding subformula ψ_i. The decomposition (Algorithm 3) proceeds via building a symbolic expression based on the formula ψ's structure. First, the algorithm replaces every subformula in $\widehat{\psi}'$ that is not a conditional with a distinct integer. In the running example the formula $\widehat{\psi}' = ((\Delta x < \Delta y \mathbin{?} \Delta x : \Delta y) + \Delta x)^2$ leads to the expression $((\Delta x < \Delta y \mathbin{?} 1 : 2) + 3)^2$ assuming the integers $1, 2, 3$ are used. Then the algorithm replaces every binary operator that has a conditional in one or both of its child nodes with a hash function `hash` that takes two integer inputs and outputs an integer hash value. The expression $((\Delta x < \Delta y \mathbin{?} 1 : 2) + 3)^2$ has a single binary operator $(+)$ with a conditional in in the place of its first argument. It thus becomes $\theta = \texttt{hash}((\Delta x < \Delta y \mathbin{?} 1 : 2), 3)^2$. Depending on the value of $\Delta x < \Delta y$ the expression θ evaluates to either $\texttt{hash}(1, 3)^2$ or $\texttt{hash}(2, 3)^2$ which become indecies of the two map regions shown in Figure 4.1. Within each region the heuristic value ψ is computed by the subformulae $\psi_1 = (\Delta x + \Delta x)^2$ and $\psi_2 = (\Delta y + \Delta x)^2$.

The final step of the algorithm (line 4) visualizes the heuristic gradient of each subformula ψ_i in its map region by displaying a color scale. The different regions are shown with different background colors and the subformula corresponding to a region is indicated by a black arrow as shown in Figure 4.1.

4.2 Algorithmic Details

In this section we first detail the simplification rules in Section 4.2.1 and the decomposition process in Sections 4.2.2 and 4.2.3. Then we discuss constraining the formula synthesis process to make the resulting formula more susceptible to our explanation approach. This motivates the extension of the original grammar with conditionals which we elaborate in Section 4.4.

4.2.1 Formula Simplification

We simplify a synthesized formula to make the decomposition and resulting subformulae more concise. The full list of simplification rules are shown in Table 4.1.

Table 4.1: Simplification Rules

Before	**After**	**Before**	**After**	**Before**	**After**
$\sqrt{\varphi^2}$	$\lvert\varphi\rvert$	$\sqrt{1}$	1	$\sqrt{\varphi}^2$	φ
$(-\varphi)^2$	φ^2	$\lvert\varphi\rvert^2$	φ^2	$\lvert\varphi^2\rvert$	φ^2
$\lvert-\varphi\rvert$	φ	$\varphi/-1$	$-\varphi$	$\lvert\sqrt{\varphi}\rvert$	$\sqrt{\varphi}$
$\lvert\lvert\varphi\rvert\rvert$	$\lvert\varphi\rvert$	$--\varphi$	φ	$-(\varphi_1-\varphi_2)$	$\varphi_2-\varphi_1$
$-\varphi_1+\varphi_2$	$\varphi_2-\varphi_2$	$\varphi_1+(-\varphi_2)$	$\varphi_1-\varphi_2$	$\varphi_1-(-\varphi_2)$	$\varphi_1+\varphi_2$
$\varphi * 1$	φ	$\varphi*(-1)$	$-\varphi$	$1*\varphi$	φ
$-1*\varphi$	$-\varphi$	$\varphi/1$	φ		

The simplifications rules are hand-coded. When the procedure `SIMPLIFY` is called (line 1 of Algorithm 1) on a given formula ψ, it is simplified recursively by matching patterns under the **Before** column and rewriting them to be the ones under the **After** column.

4.2.2 Replacing min and max with Conditionals

In line 2 of Algorithm 1 we call the function `MAXMIN2IF` which replaces all occurrences of max and min with equivalent conditionals by equations 4.1 and 4.2. We want to reduce all operators in the grammar that assign heuristic subformulae to map regions to become conditional operators. This is done so that the next step of formula decomposition (e.i., the `DECOMPOSE` function) can assume that the input formula does not contain any max or min, simplifying its implementation.

Algorithm 2: `MAXMIN2IF`(ψ). Convert all max and min in ψ to conditional operators.

input : ψ
output: If-based formula ψ'

1 **if** *none of* $\max, \min$ *appears in* ψ **then**
2     $\psi' \leftarrow \psi$
3 **else**
4     $\chi(\mathbf{e}) = \psi$
5   **for** $i = 1, \ldots, |\mathbf{e}|$ **do**
6         $e_i \leftarrow$ `MAXMIN2IF`(e_i)
7   **if** $\chi = \max$ **then**
8         $\psi' \leftarrow e_1 < e_2 \ ? \ e_2 : e_1$
9   **else if** $\chi = \min$ **then**
10         $\psi' \leftarrow e_1 < e_2 \ ? \ e_1 : e_2$
11 **return** ψ'

Referring to Algorithm 2, the function takes a formula ψ as argument and checks if any of max or min appears in it (line 1). If none of them appears in ψ, no further procedure is needed and the input formula can be returned directly. Otherwise, ψ is decomposed into $\chi(\mathbf{e})$ where χ is the root node of ψ and $\mathbf{e}$ is the vector consisting of every subformula e_i of ψ. Each subformula e_i is recursively converted as shown in line 6. Depending on what χ is, ψ is converted to contain only conditional operators in a similar manner as shown in equation 4.1 to 4.2 (lines 8 and 10). Notice that we do not reduce $|\cdot|$ even though it can be converted to conditional form (i.e., $|\varphi| = \varphi \geq 0 \ ? \ \varphi : \ -\varphi$). This is because it does not create much difficulty to explain and by allowing it to be part of a subformula we in general reduce the number of regions to visualize thus formulae become simpler to explain. For the same reason, we do not convert the terminal nodes $\Delta x = |x - x_g|$ and $\Delta y = |y - y_g|$.

4.2.3 Formula Decomposition

In line 3 of Algorithm 1 we call the function `DECOMPOSE` detailed in Algorithm 3. We assume that its input is a formula containing no max or min operators. The function recursively decomposes the formula ψ and returns an assignment function $\theta : (x, y) \rightarrow \mathbb{N}^+$ that maps a location on the map (x, y) to the index of the map region it belongs to and the list of subformulae $\mathcal{H}$ of ψ. We use lists instead of sets to support element ordering and possible repeating elements. We denote lists by angle brackets $\langle$ and $\rangle$. The union operator $\cup$ is used here to concatenate two lists. Line 1 of Algorithm 3 checks for presence of a conditional in the formula. If there is none then the algorithm returns the whole formula and a single map region – the map itself.

Algorithm 3: `DECOMPOSE`(ψ, I)

input : ψ, I
output: $\theta, \mathcal{H}, I$
1 **if** *no conditional in* ψ **then**
2     $\theta \leftarrow \max(I) + 1$
3     $I \leftarrow I \cup \{\theta\}$
4     $\mathcal{H} \leftarrow \langle \psi \rangle$
5 **else**
6     $(\chi, \mathbf{e}) \leftarrow \text{parse}(\psi)$
7   **if** $|\mathbf{e}| = 1$ **then**
8         $\theta, \mathcal{K}, I \leftarrow$ `DECOMPOSE`(e_1, I)
9         $\mathcal{H} \leftarrow \langle (\chi, k) \mid k \in \mathcal{K} \rangle$
10   **else if** $|\mathbf{e}| = 2$ **then**
11         $\beta, \mathcal{K}, I \leftarrow$ `DECOMPOSE`(e_1, I)
12         $\gamma, \mathcal{L}, I \leftarrow$ `DECOMPOSE`(e_2, I)
13         $\theta \leftarrow ($`hash`$, \beta, \gamma)$
14         $\mathcal{H} \leftarrow \langle (\chi, k, l) \mid k \in \mathcal{K}, l \in \mathcal{L} \rangle$
15   **else if** $|\mathbf{e}| = 3$ **then**
16         $\beta, \mathcal{K}, I \leftarrow$ `DECOMPOSE`(e_2, I)
17         $\gamma, \mathcal{L}, I \leftarrow$ `DECOMPOSE`(e_3, I)
18         $\theta \leftarrow (\chi, e_1, \beta, \gamma)$
19         $\mathcal{H} \leftarrow \mathcal{K} \cup \mathcal{L}$
20 **return** θ, $\mathcal{H}$, I

Otherwise the formula ψ is split into a tuple $(\chi, \mathbf{e})$ where χ is the outermost operator (referred to as the *root node* due to its position in a syntax tree of the formula) of ψ and $\mathbf{e}$ is the vector consisting of every subformula e_i of ψ (line 6).

If the number of subformulae of ψ is one (line 7), `DECOMPOSE` is called recursively on the only subformula e_1 in line 8.

The list of subformulae $\mathcal{K}$ returned by the recursive call is modified by adding χ as the root node to each subformula $k \in \mathcal{K}$ in the list formed in line 9. This is needed since χ can also be involved in the computation of heuristic values in any region of the map.

If the number of subformulae of ψ is two (line 10), both subformulae of ψ are decomposed recursively (lines 11 and 12). The sub-region-index formulae β and γ are the results of decomposing subformulae e_1 and e_2. Instead of the root node χ, we replace it with a hash function `hash` that takes two integer arguments and returns an integer as the output and combine the three of them into a formula with the root node `hash` and the two subformulae β and γ (line 13). Now θ has the root node `hash` with two sub-region-index functions β and γ. We replace χ with `hash` because β and γ might contain conditional operators that evaluates to different integer values, thus for each combination of the integer values returned by β and γ there needs to be a unique index. Therefore we compute such an index via a hash function `hash`.

If the number of subformulae of ψ is three (line 15), the only possible root node of ψ is conditional. We skip the first subformula e_1 of ψ because e_1 evaluates to a boolean value thus is not involved in the computation of the heuristic values on the map. The two subformulae e_2 and e_3 are recursively decomposed on line 16 and 17. The region-index function θ is thus constructed as $(\chi, e_1, \beta, \gamma)$ with the root node χ and the child nodes e_1, β and γ respectively. Here χ is the conditional operator, e_1 is the condition and β and γ are the sub-region-index functions decomposed from e_2 and e_3 respectively (line 18). The list of subformulae $\mathcal{H}$ is created by joining the lists of subformulae $\mathcal{K}$ and $\mathcal{L}$ of e_2 and e_3 respectively (line 19).

Now we have obtained the region-index function θ and a list of subformulae $\mathcal{H}$ that maps any location of the map to each possible value in the co-domain of θ.

4.2.4 Visualization

By decomposing the formula ψ, the algorithm computes the region-index function θ and the list of subformulae $\mathcal{H}$. In line 4 of Algorithm 1 we invoke θ on each search state (i.e., open grid cell) location. Whereas the heuristic ψ on a given state returns its estimate of the remaining distance to the goal, θ returns the index of the region to which the state belongs to. The region index is the subformula index in $\mathcal{H}$. Thus we can associate each map region to a subformula.

For each state we compute the heuristic value using the original formula ψ and display a color scale indicating the heuristic gradient.

Finally if the user supplies a start state, the algorithm can automatically create the expansion sequence and visualize it such as shown in the right plot of Figure 4.1. We also put numbers and colors on top of each state indicates the order of expansion of that state.

4.3 Explanation-Friendly Formulae

The user can likely more readily understand a subformula if it is short in size and thus readable. For instance, a simplified formula which is still long and contains neither max/min operators nor conditionals is likely to be difficult to understand since it cannot be decomposed by our algorithm.

We address this issue by constraining the synthesis process to produce formulae with short subformulae. We implement the constraint by imposing an upper bound on subformula length. This can be done simply by eliminating any candidate formula with an excessively long subformulae during synthesis. Then the constrained synthesis is no longer able to use complex subformulae inside min, max or conditional operators. The question thus becomes whether such synthesized formulae will have lower search performance (i.e., a lower speedup), which we will address in Chapter 6.

4.4 Grammar Extension

Prior work used a context-free grammar to define a space of formulae (Bulitko 2020, Bulitko *et al.* 2021, Hernandez and Bulitko 2021, Bulitko *et al.* 2022, Wang *et al.* 2023). While the grammar slightly varied from study to study, a representative example is reproduced below:

$$
\begin{aligned}
F &\rightarrow T \mid U \mid B \\
T &\rightarrow x \mid x_g \mid y \mid y_g \mid \Delta x \mid \Delta y \mid C \\
U &\rightarrow \sqrt{F} \mid -F \mid F^2 \mid |F| \\
B &\rightarrow F + F \mid F - F \mid F \times F \mid \frac{F}{F} \mid \max(F, F) \mid \min(F, F)
\end{aligned}
$$

Here, $\Delta x = |x - x_{\text{goal}}|$, $\Delta y = |y - y_{\text{goal}}|$ and C represents a set of constants.

Note the operators max and min effectively assign subformulae to map regions which allows different parts of the synthesized formula to capture specifics of different map regions, thereby offering better heuristic guidance. To illustrate, consider the heuristic formula $\max(\Delta x, \Delta y)^2$ which decomposes into Δx^2 and Δy^2. The subformula Δx^2 directs the search open list from a start node on the left side of the map horizontally towards the C-shaped wall in the middle (Figure 4.2). Then the open list follows (or "hugs") the wall. The other subformula Δy^2 then guides the search open list vertically towards the goal node G.

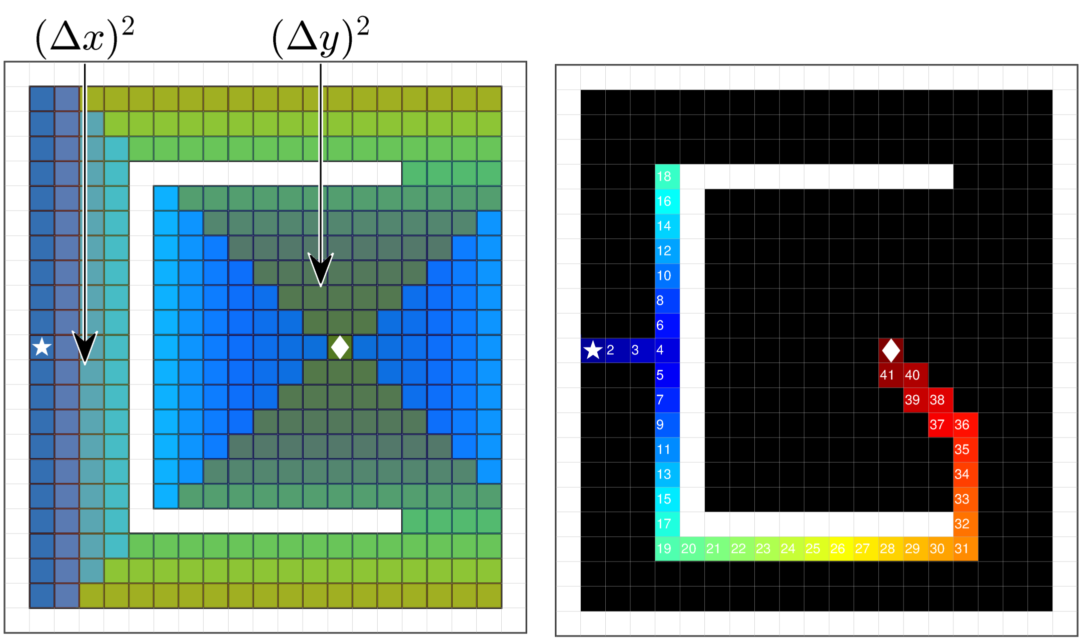

Figure 4.2: Left: decomposing the "wall-hugging" heuristic $\max(\Delta x, \Delta y)^2$. Right: expansion sequence starting from S.

We capitalize on the idea of map regioning by explicitly extending the grammar with a conditional which we represent in the grammar as $X\ ?\ S : S$. The extended grammar is shown below:

$$
\begin{aligned}
S &\rightarrow F \mid X\ ?\ S : S \\
X &\rightarrow S > S \mid S < S \mid S \geq S \mid S \leq S \mid X \wedge X \mid X \vee X \\
F &\rightarrow T \mid U \mid B \\
T &\rightarrow x \mid x_g \mid y \mid y_g \mid \Delta x \mid \Delta y \mid C \\
U &\rightarrow \sqrt{S} \mid -S \mid S^2 \mid |S| \\
B &\rightarrow S + S \mid S - S \mid S \times S \mid \frac{S}{S} \mid \max(S, S) \mid \min(S, S)
\end{aligned}
$$

We compare the quality of heuristics synthesized with both grammars and detail the results in Chapter 6.

4.5 Iterative Heuristic Formula Improvement and Adversarial Problems Display

Here we describe how our approach is used as a tool for a user to improve a given heuristic formula with. We demonstrate through two settings. First, we show how to apply our approach to improve a given heuristic formula for a single pathfinding problem in Section 4.5.1. Second, we show the application of it to improve for a set of pathfinding problems in Section 4.5.2.

4.5.1 Formula Improvement for a Single Problem

When we want to improve a heuristic formula for a single pathfinding problem, we first decompose and visualize the formula to inform the user and the user makes modifications to the formula to improve it. The modified formula is decomposed and visualized again and the user modifies it again. The cycle repeats until the user is satisfied with the result. We show an example of this approach to improve the Manhattan Distance heuristic $\Delta x + \Delta y$.

Consider the Manhattan distance heuristic $\Delta x + \Delta y$. The visualization in the upper two plots of Figure 4.3 shows that A* expands a number of states close to the start state. Note that the search problem shown in the figure is identical to the one in the previous subsection except for the exchange of start and goal state locations.

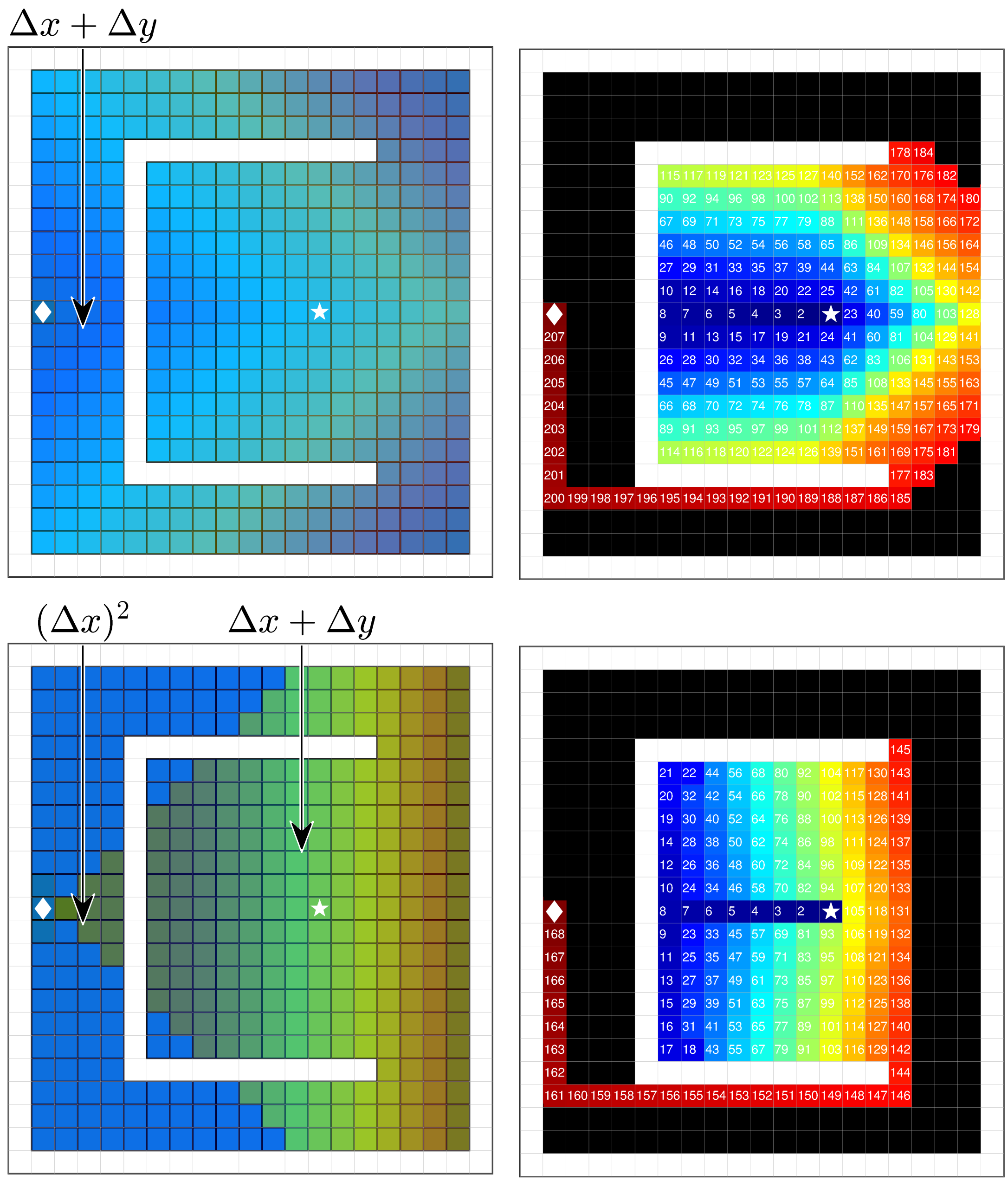

Figure 4.3: $\Delta x + \Delta y$ is incrementally modified into $\Delta x > \Delta y \ ? \ \Delta x^2 : \Delta x + \Delta y$.

Since there are no conditions in the formula, any modifications to it can affect A* on the whole map. To localize the effects the user adds a conditional to the formula, replacing the original $\Delta x + \Delta y$ with $\Delta x > \Delta y \ ? \ h_1 : \Delta x + \Delta y$ creating a choice for the subformula h_1. The user sets $h_1 = \Delta x^2$ which drives

A* away from the start state and reduces the closed list (the lower two plots in the figure). The resulting formula is $\Delta x > \Delta y \ ? \ \Delta x^2 : \Delta x + \Delta y$ which saved $\kappa(h, h', P) = 208 - 169 = 39$ state expansions.

4.5.2 Formula Improvement for Multiple Problems

When multiple pathfinding problems are present, to improve a heuristic formula, the goal is to modify it to increase the average speedup of it on the set of given pathfinding problems. Assuming that there are many pathfinding problems, visualizing all of them is unfeasible, thus we visualize the problem that contributes the most negatively towards the average speedup by selecting the problem with the least speedup when guided by the given heuristic formula. The user then makes modifications to the formula to improve it for that adversarial problem, and a new adversarial problem selected against the newly modified formula is visualized. The cycle repeats until the user is satisfied with the resulting formula.

Consider the formula $(\min(\Delta x, \Delta y) - (-\Delta x))^2$ from Section 4.1. Multiple problems were generated for the map shown in Figure 4.4, and the adversarial problem against the formula is shown in the top two plots. The average speedup of the formula over the set of problems is 0.8. Now suppose the user modifies the min operator to max thus the formula becomes $(\max(\Delta x, \Delta y) - (-\Delta x))^2$. After re-visualizing, a new adversarial problem is shown in the bottom two plots with more states expanded. However the average speedup increased to 1.2. The improvement is a $\kappa(h, h', P) = 1.2 - 0.8 = 0.4$ improvement in speedup.

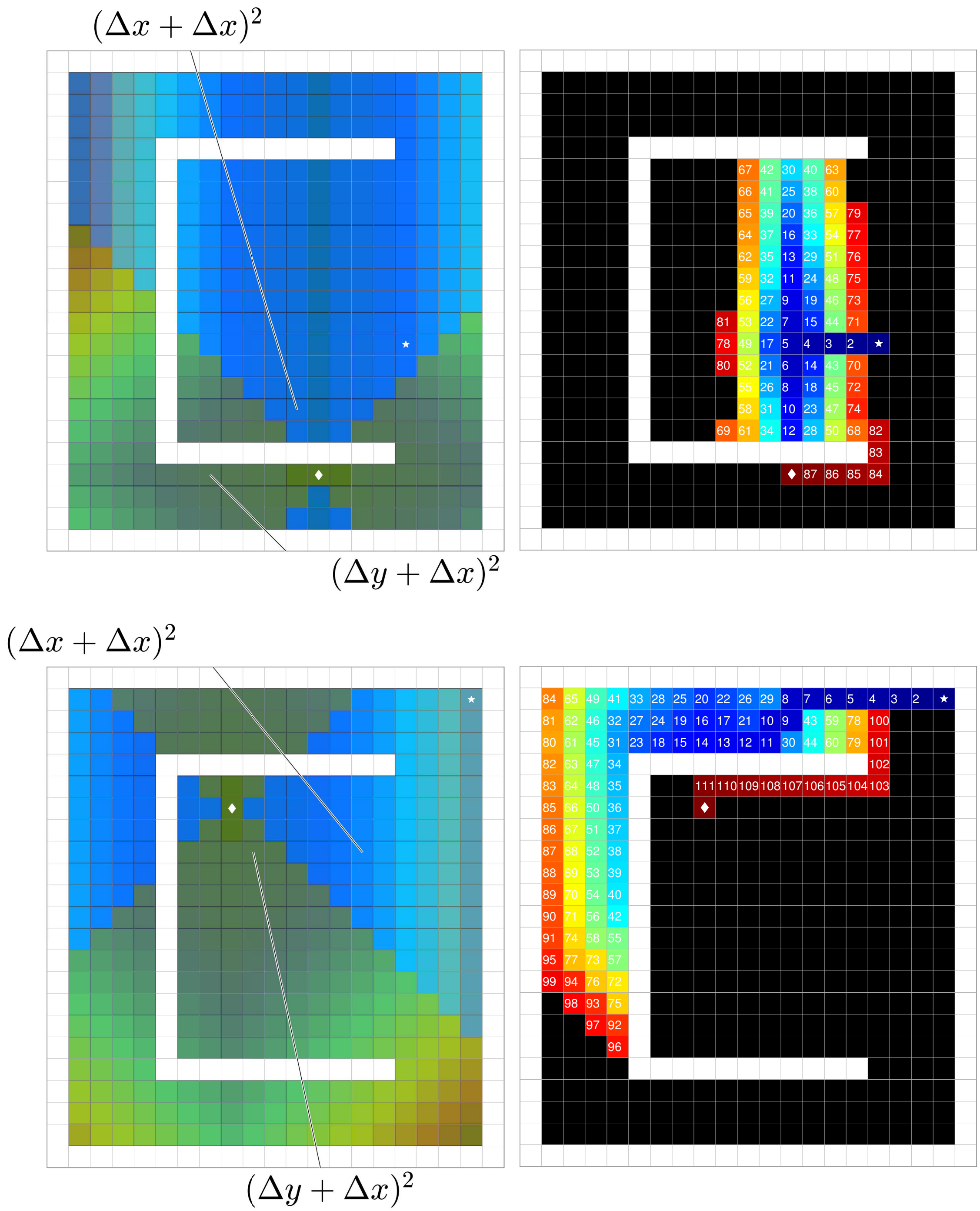

Figure 4.4: $(\min(\Delta x, \Delta y) - (-\Delta x))^2$ modified to $(\max(\Delta x, \Delta y) - (-\Delta x))^2$.

Chapter 5

Heuristic Improvement Case Studies

In this chapter we demonstrate our approach with examples of heuristic visualization and how a user might make improvements to the heuristic formulae. A synthesized heuristic formula is improved when the modification made by a user caused it to induce less states expansions during A* search.

We start with preliminary examples to demonstrate some basic principles on heuristic improvement. Then we show examples of applying our approach to improve synthesized heuristics on video-game maps from *Dragon Age: Origins* (DAO) where we show cases where our approach succeeds in improving a synthesized heuristic and cases where it fails to make any improvements. Furthermore, we demonstrate the application of our approach to improve a heuristic formula for multiple maps. In the end of this chapter, we summarize and motivate the empirical evaluations.

5.1 Single-Problem Cases

Consider the case of a single pathfinding problem. We show several examples of applying our method to improve the guiding performance of a heuristic formula.

5.1.1 Case #1: Chebyshev Distance

From the work by Bulitko *et al.* 2022, the heuristic formulae synthesized appear often as variants of the wall-hugging heuristic $\max(\Delta x, \Delta y)^2$. This heuristic formula is related to the Chebyshev Distance (Deza *et al.* 2009) $\max(\Delta x, \Delta y)$. Suppose the Chebychev Distance is given as the heuristic formula for the user to make improvements upon, we demonstrate the application of our approach.

Consider the example shown in Figure 5.1. The original Chebyshev Distance heuristic induces a substantial number of nodes in the closed list. The user chooses to increase the heuristic values in the region where the start state lies in the hope to make the region close to the start state unattractive to search so that A* may escape from the interior of the C-shape wall early on during the search. The heuristic values of the region where the start state lies are computed by the subheuristic Δx, thus the user modifies it to Δx^2.

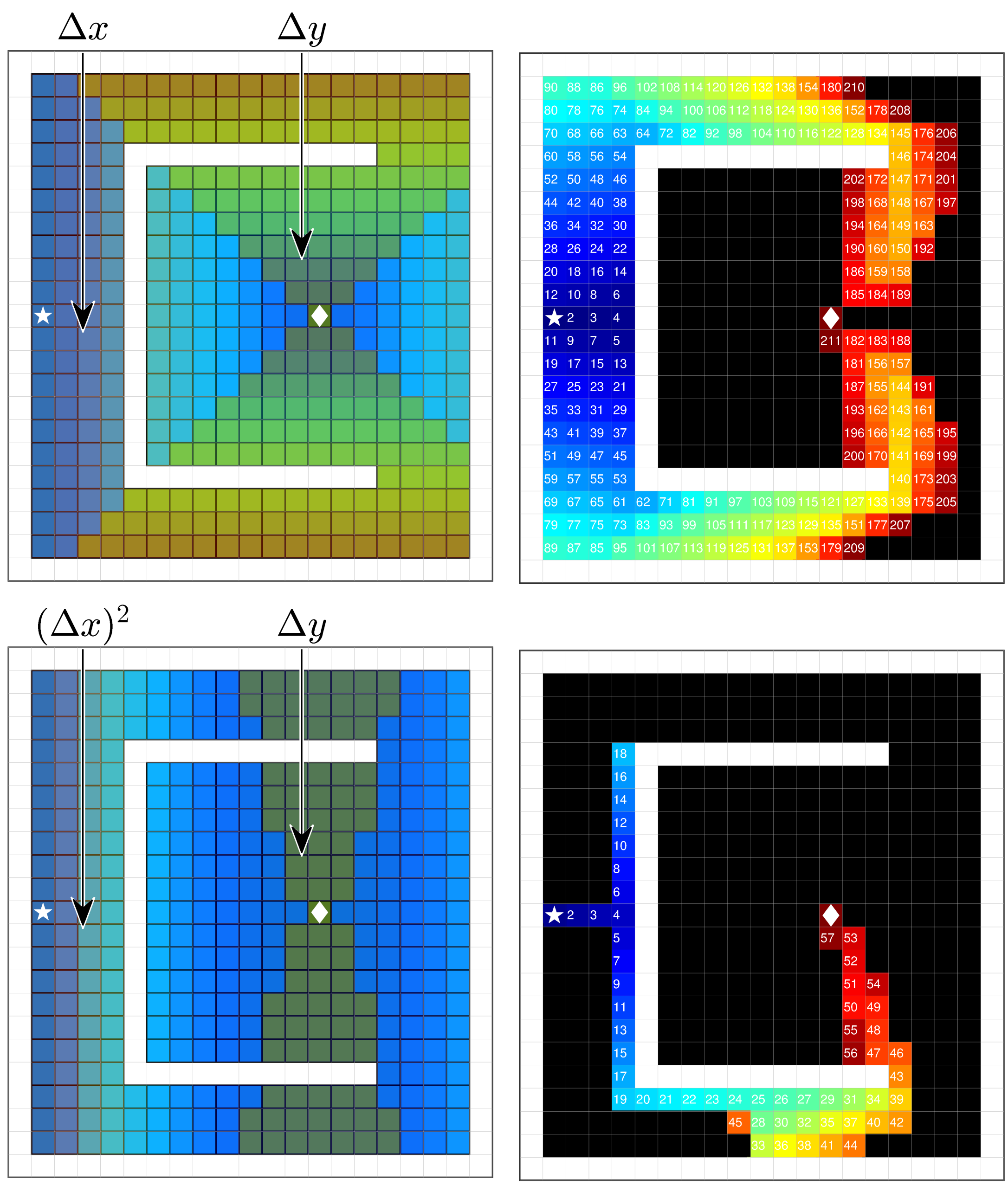

Figure 5.1: The Chebyshev Distance heuristic $\max(\Delta x, \Delta y)$ is modified into $\max(\Delta x^2, \Delta y)$. The white star and diamond denote the start and goal states.

After re-visualizing, the closed list is reduced evidencing the improvement of the heuristic's quality. The resulting heuristic formula is $\max(\Delta x^2, \Delta y)$. Note that we are improving the heuristic formula for a single pathfinding problem. If we are to improve the formula for multiple problems, we may append a square operator to Δy and end up with the wall-hugging heuristic.

The improvement from $h = \max(\Delta x, \Delta y)$ to $h' = \max(\Delta x^2, \Delta y)$ is $\kappa(h, h', P) = \epsilon(h) - \epsilon(h') = 212 - 58 = 154$ state expansions.

5.1.2 Case #2: The Wall-hugging Heuristic

Consider the wall-hugging heuristic $\max(\Delta x, \Delta y)^2$ by the work of Bulitko *et al.* 2022. We first convert it to its equivalence: $(\Delta x)^2 > (\Delta y)^2\ ?\ (\Delta x)^2 : (\Delta y)^2$. On the map in Figure 5.2 and Figure 5.3, the heuristic imposes two regions with heuristic values computed by the subformulae $(\Delta x)^2$ and $(\Delta y)^2$ (the first plot in the top row). Visualizing the A* closed list (the second plot in the top row) tells the user that a number of states inside the C-shape wall are expanded. The area is covered by the triangle shaped region associated with the subformula $(\Delta x)^2$. However, that region is substantially broader than the C-shape wall so any changes to its subformula may also affect A*'s behaviour outside of the area. Thus the user chooses to first make the region narrower, replacing the condition $(\Delta x)^2 > (\Delta y)^2$ with $(\Delta x)^2 > 2(\Delta y)^2$, narrowing the region governed by $(\Delta x)^2$ (the third plot in the top row).

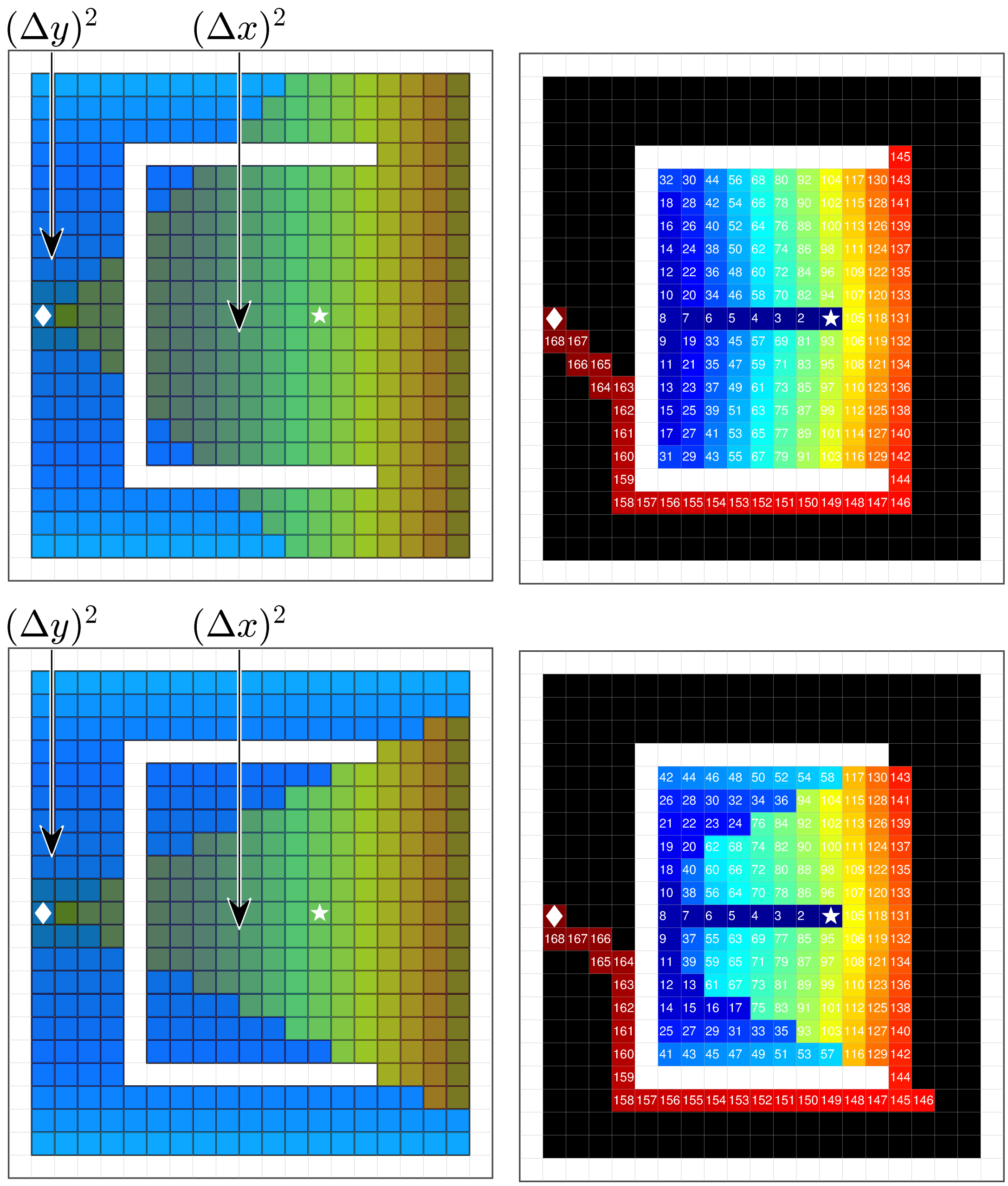

Figure 5.2: The wall-hugging heuristic $\max(\Delta x, \Delta y)^2$ is modified into $(\Delta x)^2 > 2(\Delta y)^2 \ ? \ (\Delta x)^2 : (\Delta y)^2$. The white star and diamond denote the start and goal states.

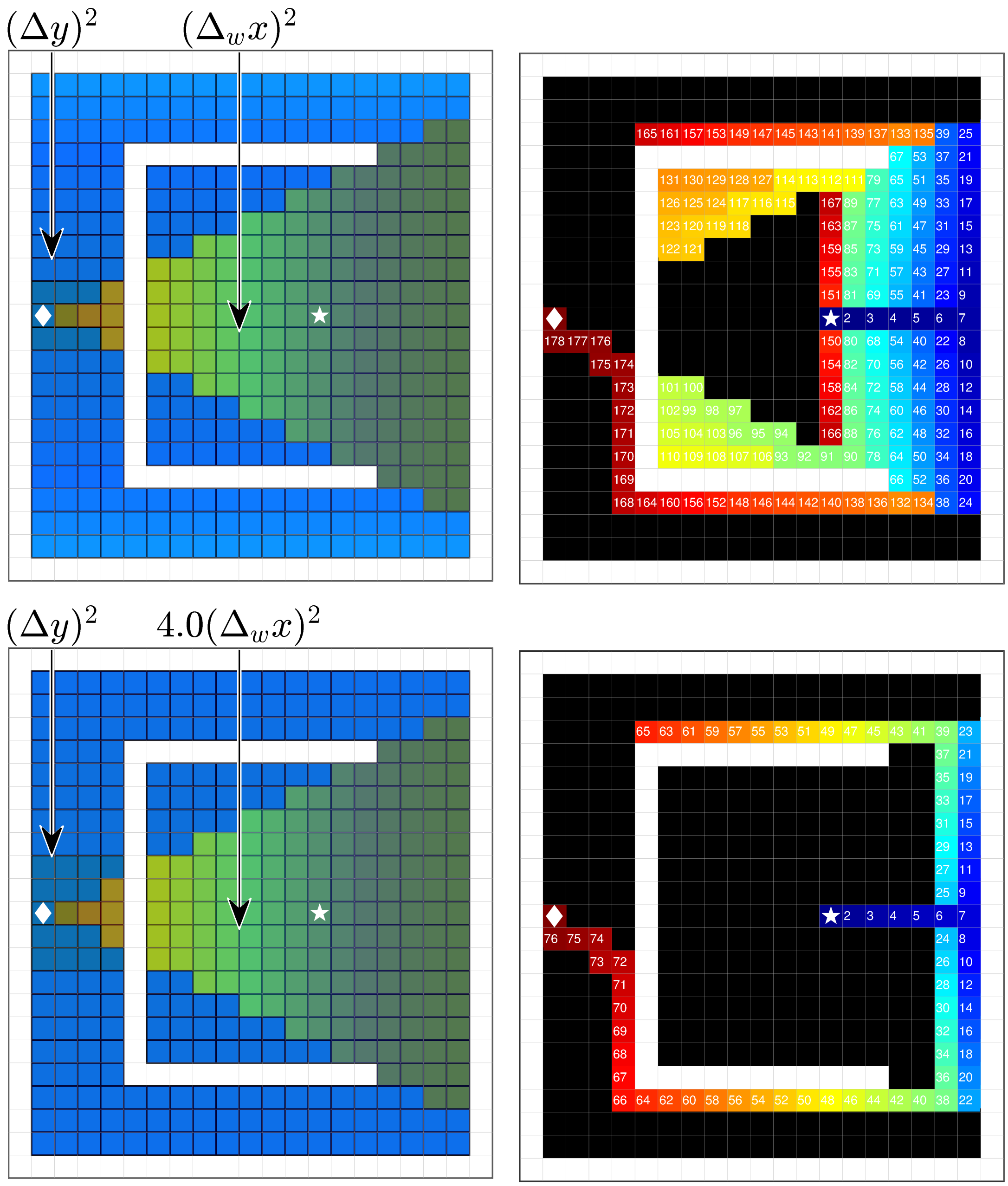

Figure 5.3: The heuristic $(\Delta x)^2 > 2(\Delta y)^2$? $(\Delta x)^2 : (\Delta y)^2$ is modified into $(\Delta x)^2 > 2(\Delta y)^2$? $4(\Delta_w x)^2 : (\Delta y)^2$.

Now the problem with $(\Delta x)^2$ is that it guides the search into the C-shape wall making A* expand many states. The user intends to guide A* out of the area by replacing it with $(\Delta_w x)^2$ where Δ_w is an extension of the regular Δ operator: $\Delta_w x = |x - x_{\text{goal}} - w|$ where w is the width of the map.

The user then visualizes the new heuristic formula $(\Delta x)^2 > 2(\Delta y)^2$? $(\Delta_w x)^2$:

$(\Delta y)^2$ (the left plot at the top) and realizes that the new A* closed list is still extensively covering the inside of the C-shape wall (the right plot at the top). This prompts the user to increase the heuristic values of the new subformula. The user chooses to add a factor 4 to $(\Delta_w x)^2$ forming: $4(\Delta_w x)^2$. A re-visualization shows a much reduced closed list (the right plot at the bottom). The formula after the iterative improvement is $(\Delta x)^2 > 2(\Delta y)^2 \ ? \ 4(\Delta_w x)^2 : (\Delta y)^2$, inducing a number of state expansion improvement of $\kappa(h, h', P) = 169 - 77 = 92$ states.

5.1.3 Section Summary

In general when there is a single pathfinding problem, it is easy to modify a given heuristic formula to guide search efficiently for the given problem. However in practice, it is costly to optimize a heuristic formula for just one search problem, instead it is common to optimize a heuristic formula for problems of a video game map, or for problems of several maps.

In the next section we present a few examples of iterative heuristic formula improvement for pathfinding problems of particular video-game maps and an example of it for problems across several maps. We list general principles of when our approach can work and when it will probably fail.

5.2 Iterative Heuristic Formula Improvement based on Adversarial Problem Selection

In this section we apply our method to improve synthesized heuristic formulae on the video maps they were synthesized on. We down sample the video game maps for better visualization. The heuristic formulae that we start with in each case study all come from the work by Bulitko *et al.* 2022.

On each down sampled map, 40×40 problems are randomly generated and evaluated by A* guided by the given heuristic formula. The search problem with the largest number of nodes expanded (the worst search problem) is selected and visualized (the adversarial search problem). The user can either make a modification to the heuristic function with the intension to make it more efficient for that problem and re-visualize or terminate the process. The goal is to improve the heuristic for any search problems on the given video game map in general.

5.2.1 Case #1: brc202d

On the map `brc202d`, we start with the synthesized heuristic function:

$$\max\left(\sqrt{\frac{y_{\text{goal}}}{x}}\Delta x, \Delta y\right)^4$$

The adversarial search problem found is shown in Figure 5.4.

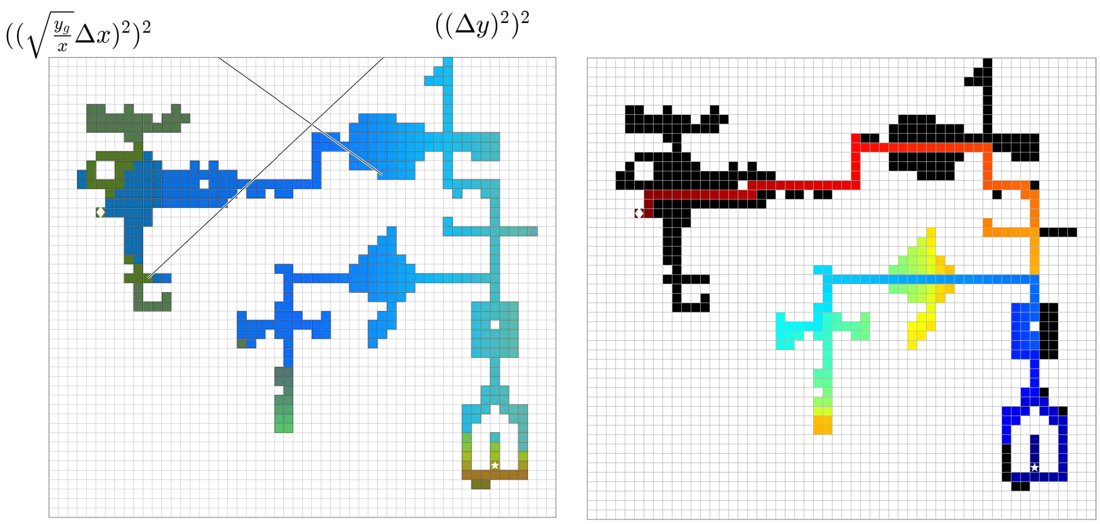

Figure 5.4: The synthesized formula $\max\left(\sqrt{\frac{y_{\text{goal}}}{x}}\Delta x, \Delta y\right)^4$ on the down sampled map.

After decomposition, the formula becomes $\left(\sqrt{\frac{y_{\text{goal}}}{x}}\Delta x\right)^4$ and Δy^4 assigned to the blue and the green regions (left of figure 5.4). Δx and Δy guide search horizontally and vertically towards the goal, the behavior of the synthesized component $\sqrt{\frac{y_{\text{goal}}}{x}}$ needs further investigation. Suppose we have a search problem, its goal location is fixed, then during the search, the value of y_{goal} remains the same and the value of x is larger for cells more to the right of the map. Taken together, the value of $\frac{y_{\text{goal}}}{x}$ are smaller for the cells on the right of the map thus guiding the search rightwards. But since it is multiplied to Δx, the search is guided to the goal location horizontally with a tendency to expand rightward.

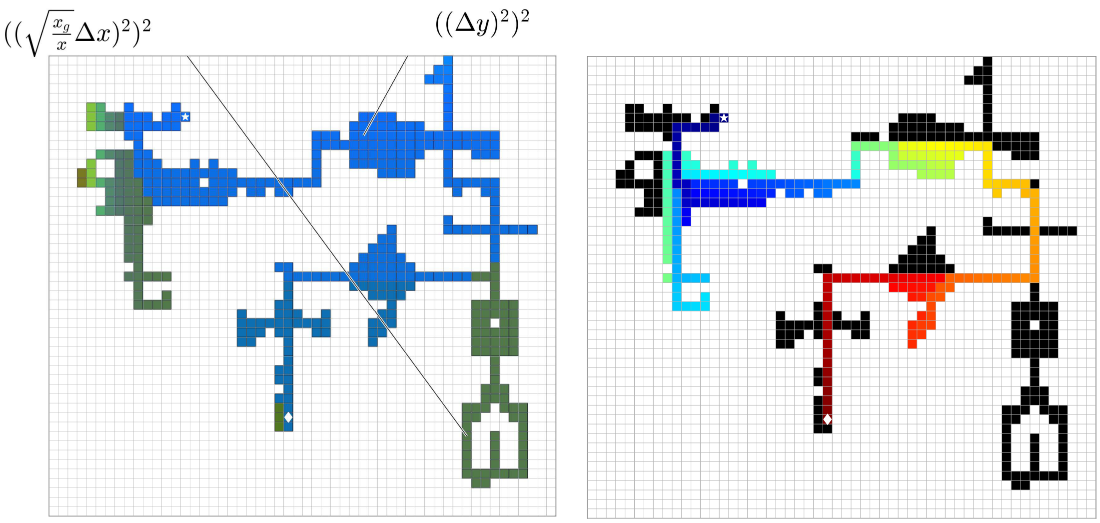

Figure 5.5: $\max\left(\sqrt{\frac{y_{\text{goal}}}{x}}\Delta x, \Delta y\right)^4$ modified to $\max\left(\sqrt{\frac{x_{\text{goal}}}{x}}\Delta x, \Delta y\right)^4$, increasing average speedup on down sampled map from 1.457 to 1.466.

The user decides that it makes more sense to modify $\frac{y_{\text{goal}}}{x}$ to become $\frac{x_{\text{goal}}}{x}$ since x and x_{goal} are more related. This modification has increased the average speedup on the problems of the map from 1.457 to 1.466. Here on the right of figure 5.5, the closed list shows the adversarial search problem. The top left region of the map has heuristic values computed by $(\sqrt{\frac{x_{\text{goal}}}{x}}\Delta x)^4$ which is supposed to guide the search rightward into the light blue region yet it still incurs some state expansions.

Thus the user amplifies the heuristic values of the green region to emphasize the rightward guidance of the search by modifying $\left(\sqrt{\frac{x_{\text{goal}}}{x}}\Delta x\right)^4$ to become $\left(\frac{x_{\text{goal}}}{x}\Delta x\right)^4$. This modification has increased the average speedup from 1.466 to 1.493.

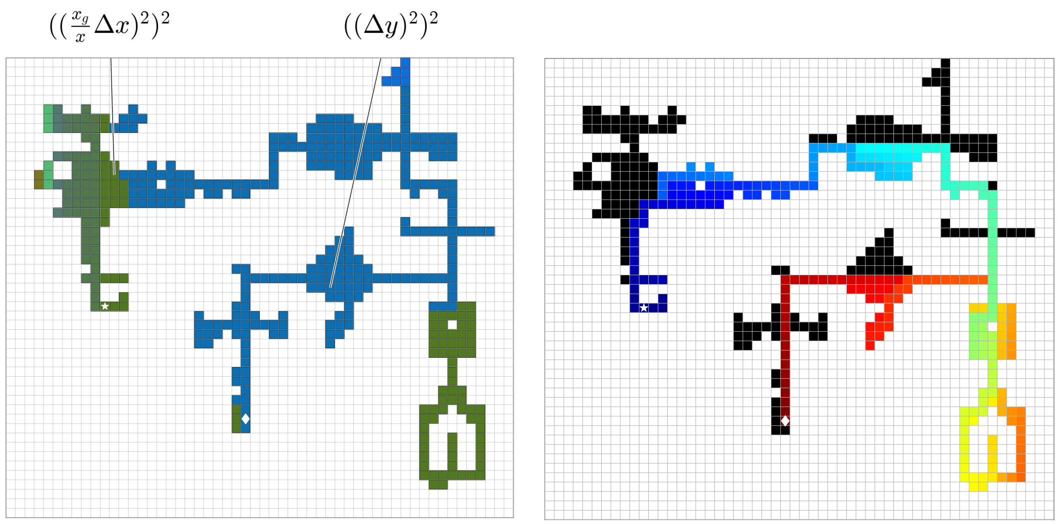

Figure 5.6: $\max\left(\sqrt{\frac{x_{\text{goal}}}{x}}\Delta x, \Delta y\right)^4$ incrementally modified to $\max\left(\frac{x_{\text{goal}}}{x}\Delta x, \Delta y\right)^4$, increasing average speedup on down sampled map from 1.466 to 1.493.

After re-visualizing, we arrive at figure 5.6. The problem shown in the figure involves two major horizontal oriented hallways. Therefore the user tries to inform the search to ignore more vertical hallways by adding a Δx component to the entire heuristic formula, modifying $\max\left(\frac{x_{\text{goal}}}{x}\Delta x, \Delta y\right)^4$ to become $\Delta x + \max\left(\frac{x_{\text{goal}}}{x}\Delta x, \Delta y\right)^4$ increasing the average speedup from 1.493 to 1.509.

Re-visualization give us figure 5.7. By testing the resulting heuristic $\Delta x + \max\left(\frac{x_{\text{goal}}}{x}\Delta x, \Delta y\right)^4$ on the 200×200 test problem set on the original map, we have improved the search performance from 3.843 to 4.191, thus making the synthesized heuristic formula even better for the map it was synthesized on.

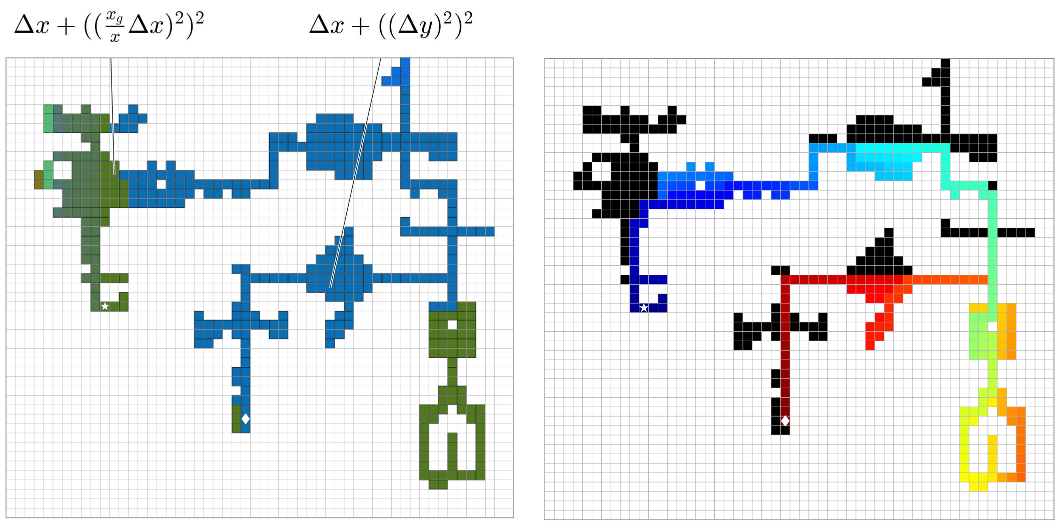

Figure 5.7: The synthesized formula $\max\left(\sqrt{\frac{y_{\text{goal}}}{x}}\Delta x, \Delta y\right)^4$ incrementally modified to $\Delta x + \max\left(\frac{x_{\text{goal}}}{x}\Delta x, \Delta y\right)^4$, increasing average speedup on down sampled map from 1.493 to 1.509 and average test speedup from 3.843 to 4.191.

The iterative improvement succeeded with an improvement in speedup of $\kappa(h, h', P) = 4.191 - 3.843 = 0.348$. We speculate several reasons causing the success. First, the map it self is mostly consistent with long narrow corridors, causing most generated pathfinding problems to be similar, thus a heuristic formula that works well for one problem likely works well for most other problems. Second, the formula we start with decompose to simple subformulae: $\left(\sqrt{\frac{y_{\text{goal}}}{x}}\Delta x\right)^4$ and $(\Delta y)^4$. They are simple in the sense that they are short and that they each contain components that only guide search towards the goal either horizontally or vertically. The user therefore can more conveniently introduce biases into the formula to explore aspects of the adversarial problem.

5.2.2 Case #2: den000d

For map `den000d` the synthesizer gave $\Delta y + 44.9 \cdot \max(\Delta x, \Delta y)$. When decomposed, the sub-formulae $\Delta y + 44.9 \cdot \Delta x$ and $\Delta y + 44.9 \cdot \Delta y$ are associated to the blue and green regions (left of figure 5.8). Obverse that the closed list mostly contains nodes around the peninsula of the map where the goal is.

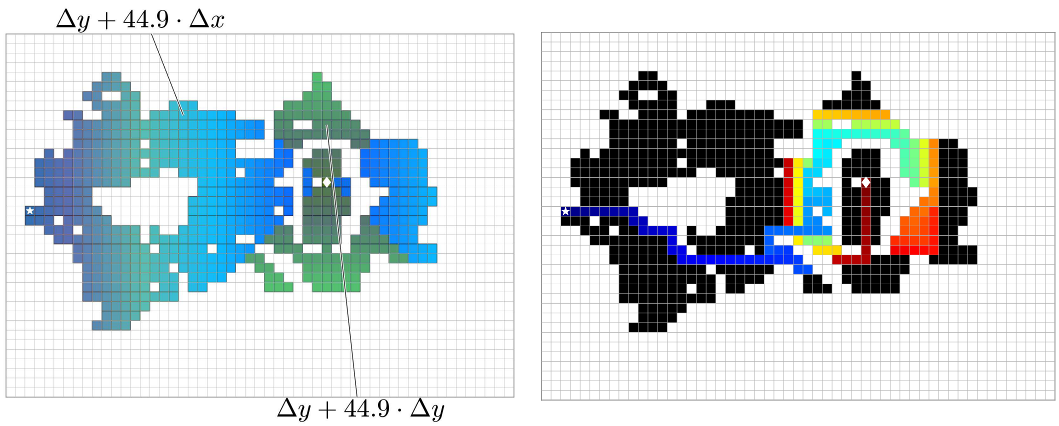

Figure 5.8: The synthesized formula $\Delta y + 44.9 \cdot \max(\Delta x, \Delta y)$ on the down sampled map.

Observe that if we modify the heuristic formula to make the blue region less attractive to search, the search will be guided to be most in the green region thus we reduce the search effort. The user modifies the formula by amplifying Δx: modifying $\Delta y + 44.9 \cdot \max(\Delta x, \Delta y)$ to become $\Delta y + 44.9 \cdot \max(2 \cdot \Delta x, \Delta y)$.

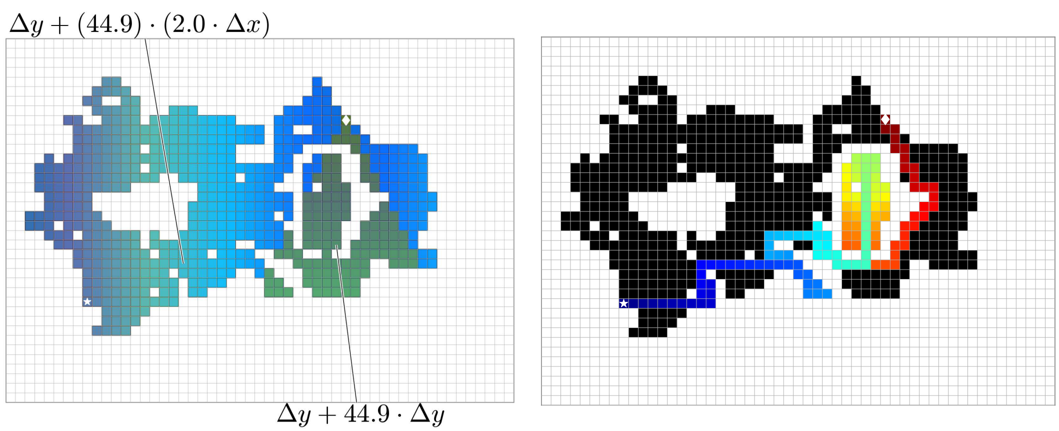

Figure 5.9: $\Delta y + 44.9 \cdot \max(\Delta x, \Delta y)$ modified to $\Delta y + 44.9 \cdot \max(2 \cdot \Delta x, \Delta y)$, decreasing average speedup on down sampled map from 1.736 to 1.734.

Re-visualization gives us figure 5.9 but the average speedup has decreased from 1.736 to 1.734. Observation of the closed list informs the user that the search has taken the longer route, passing below the peninsula, and on the way, expanded all nodes contained in the peninsula. Thus the user is prompted to amplify the Δy component to guide the search vertically towards the goal location earlier during the search. Therefore the formula is modified to become $\Delta y^2 + 44.9 \cdot \max(2 \cdot \Delta x, \Delta y)$ to enlarge the values of the Δy component to guide the search vertically towards the goal.

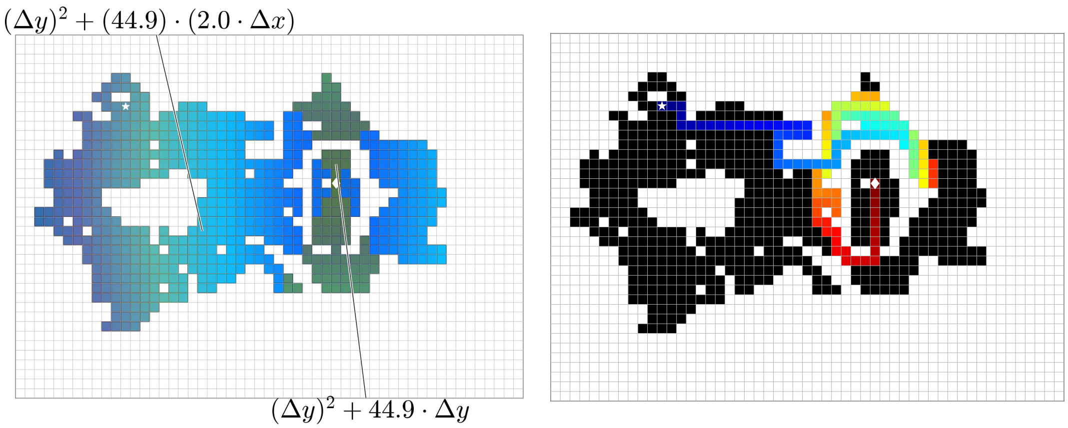

Figure 5.10: $\Delta y + 44.9 \cdot \max(2 \cdot \Delta x, \Delta y)$ modified to $\Delta y^2 + 44.9 \cdot \max(2 \cdot \Delta x, \Delta y)$, increasing average speedup on down sampled map from 1.734 to 1.737.

After re-visualizing, we get figure 5.10 with an improvement of speedup from 1.734 to 1.737. The closed list (right of figure 5.10) has the expanded nodes mostly around and close to the wall leading to an opening towards the goal location.

Observe that the multiplier 44.9 in front of the $\max(2 \cdot \Delta x, \Delta y)$ component. It is a constant chosen by the synthesizer for the original map. In order to make the formula more concise, it is reasonable to modify such amplifiers to become something such as a multiplication of 2 or a square. The user thus replaces it with a square operator and we arrive at the formula $\Delta y^2 + \max(2 \cdot \Delta x, \Delta y)^2$ as the result, increasing the average speedup from 1.737 to 1.761.

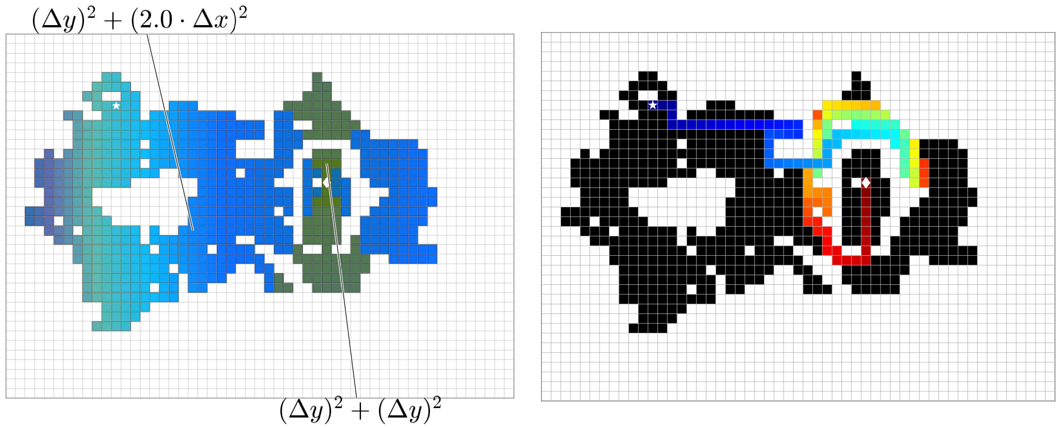

Figure 5.11: The synthesized formula $\Delta y + 44.9 \cdot \max(\Delta x, \Delta y)$ incrementally modified to $\Delta y^2 + \max(2 \cdot \Delta x, \Delta y)^2$, increasing average speedup on down sampled map from 1.737 to 1.761 but decreasing average test speedup from 5.268 to 4.073.

However, the average test speedup decreased from 5.268 to 4.073 on the original map. We suspect several reason that causes the failure. First, the modified formula might be over-fitting to the down sampled map. Second, after decomposition the subformula $\Delta y + 44.9 \cdot \Delta x$ involve components that guide search both horizontally and vertically towards the goal, thus making it difficult for the user to come up with effective ways to modify the formula.

5.2.3 Case #3: den501d

Consider the video game map `den501d`. The synthesizer gave the formula $\max\left(\frac{y_{\text{goal}}}{y-8.3} \cdot \Delta y, \Delta x\right)^2$ which is decomposed into $\left(\frac{y_{\text{goal}}}{y-8.3} \cdot \Delta y\right)^2$ and Δx^2 with map region assignment shown on the left image of figure 5.12.

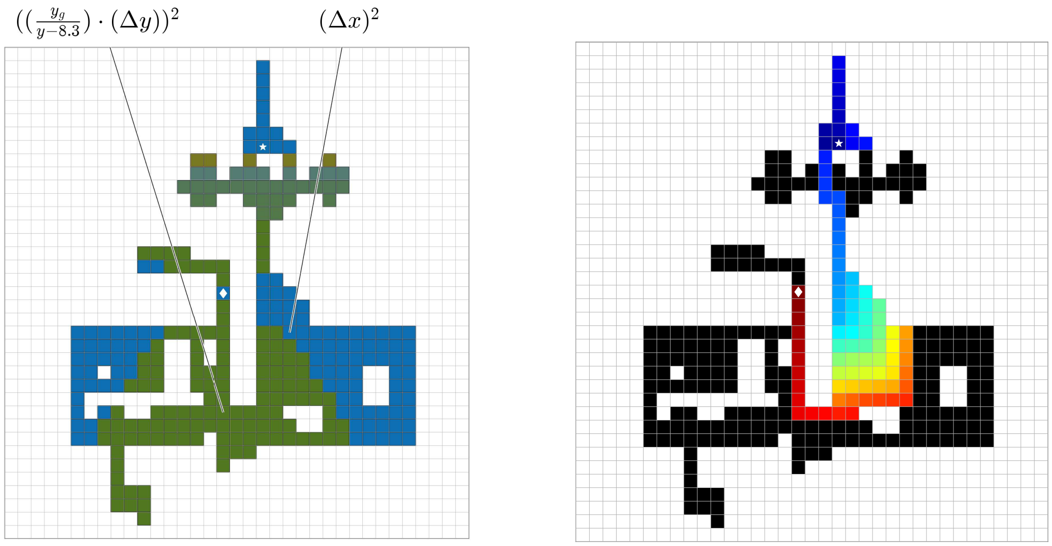

Figure 5.12: The synthesized formula $\max\left(\frac{y_{\text{goal}}}{y-8.3} \cdot \Delta y, \Delta x\right)^2$ on the down sampled map.

The closed list indicates that the blue region (associated with Δx^2) can be made less attractive to search to reduce the nodes expanded. Thus the user multiplies Δx by 2 to induce the behaviour of sticking to the wall on the left. The formula $\max\left(\frac{y_{\text{goal}}}{y-8.3} \cdot \Delta y, \Delta x\right)^2$ thus becomes $\max\left(\frac{y_{\text{goal}}}{y-8.3} \cdot \Delta y, 2 \cdot \Delta x\right)^2$, resulting in a average speedup increase from 1.326 to 1.363.

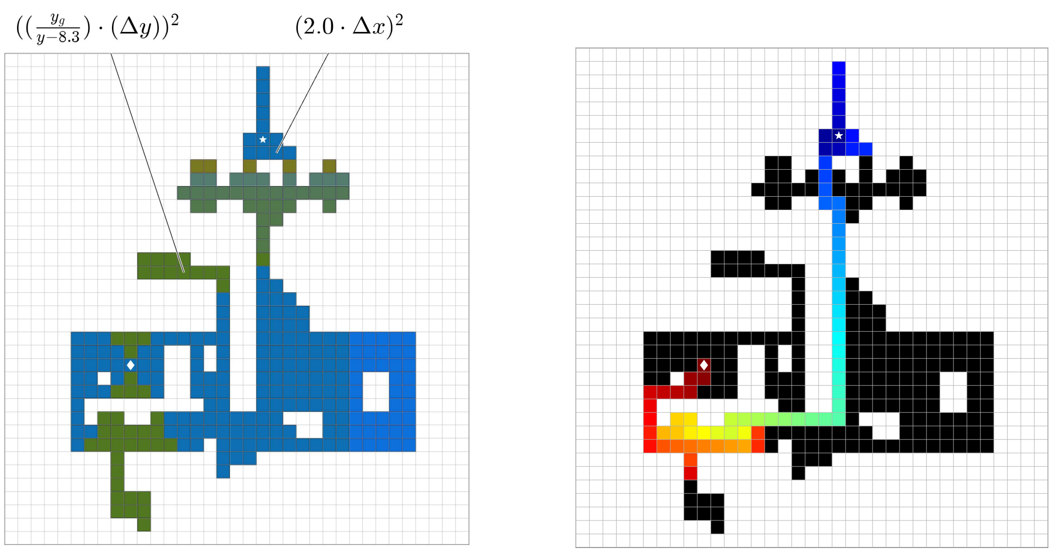

Figure 5.13: $\max\left(\frac{y_{\text{goal}}}{y-8.3} \cdot \Delta y, \Delta x\right)^2$ modified to $\max\left(\frac{y_{\text{goal}}}{y-8.3} \cdot \Delta y, 2 \cdot \Delta x\right)^2$, increasing average speedup on down sampled map from 1.326 to 1.363.

The new worst problem is selected and the closed list with the start-goal pair locations shown in the right image of figure 5.13. Now the $\frac{y_{\text{goal}}}{y-8.3}$ component is similar to the $\frac{x_{\text{goal}}}{x}$ component from subsection 5.2.1 and with similar reasoning, the $\frac{y_{\text{goal}}}{y-8.3}$ component enables the search the tendency to expand vertically downward. However, to make the component more archetypal, the user decides to modify it to become $\frac{y_{\text{goal}}}{y}$ thus one may use the same reasoning from subsection 5.2.1 to directly explain its behaviour. After modifying the formula from $\max\left(\frac{y_{\text{goal}}}{y-8.3} \cdot \Delta y, 2 \cdot \Delta x\right)^2$ to $\max\left(\frac{y_{\text{goal}}}{y} \cdot \Delta y, 2 \cdot \Delta x\right)^2$, the average speedup drops from 1.363 to 1.330 as shown in Figure 5.14.

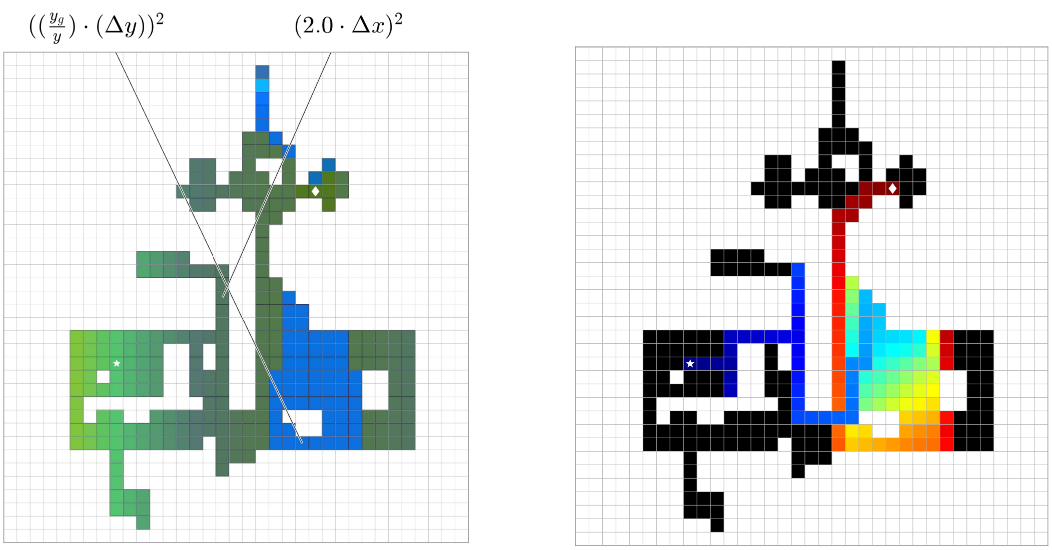

Figure 5.14: $\max\left(\frac{y_{\text{goal}}}{y-8.3} \cdot \Delta y, 2 \cdot \Delta x\right)^2$ modified to $\max\left(\frac{y_{\text{goal}}}{y} \cdot \Delta y, 2 \cdot \Delta x\right)^2$, decreasing the average speedup on down sampled map from 1.363 to 1.330.

Observation of the closed list in the image on the right of figure 5.14 prompts the user to amplify the heuristic values in the blue region governed by the sub-formula $\left(\frac{y_{\text{goal}}}{y} \cdot \Delta y\right)^2$ in order to make the region less attractive to search. Thus the user modifies the previous formula $\max\left(\frac{y_{\text{goal}}}{y} \cdot \Delta y, 2 \cdot \Delta x\right)^2$ to become the formula $\max\left(2 \cdot \frac{y_{\text{goal}}}{y} \cdot \Delta y, 2 \cdot \Delta x\right)^2$, resulting in an increase of average speedup from 1.330 to 1.386.

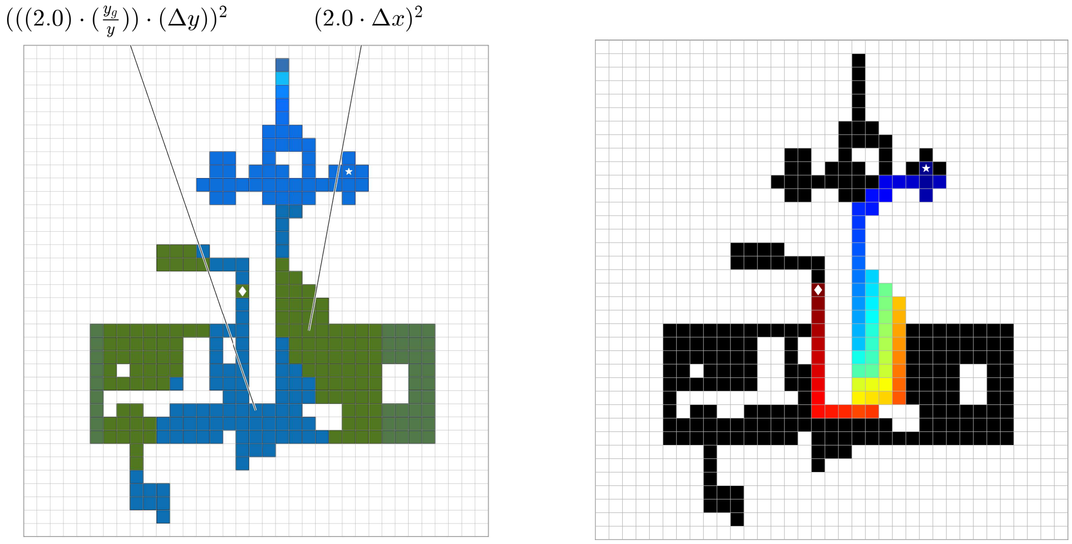

Figure 5.15: The synthesized formula $\max\left(\frac{y_{\text{goal}}}{y-8.3} \cdot \Delta y, 2 \cdot \Delta x\right)^2$ incrementally modified to $\max\left(2 \cdot \frac{y_{\text{goal}}}{y} \cdot \Delta y, 2 \cdot \Delta x\right)^2$, increasing average speedup on down sampled map from 1.330 to 1.386 and the average test speedup from 3.821 to 3.895.

The user is satisfied with the result and terminates the heuristic improvement process. The resulting heuristic formula on the test problems yields an increase of speedup from 3.821 to 3.895.

The improvement process again proves to be successful and we observe that the subformulae are again simple and the user is thus more capable of introducing biases into the formula to exploit aspects of the adversarial problem.

5.2.4 Case #4: lak505d

Moving on to the map `lak505`. Starting from the formula synthesized

$$\max\left(100, \min\left(y_{\text{goal}}, \Delta y\right) + y\right)^2 \cdot \max\left(\Delta x, \Delta y\right)$$

which decomposes into four sub-formulae. However, since on the down sample map, for the search problem shown in Figure 5.16, $\min\left(y_{\text{goal}}, \Delta y\right) + y$ is less than 100 for all cells. Therefore only two sub-formulae are shown: $100^2 \cdot \Delta x$ and $100^2 \cdot \Delta y$.

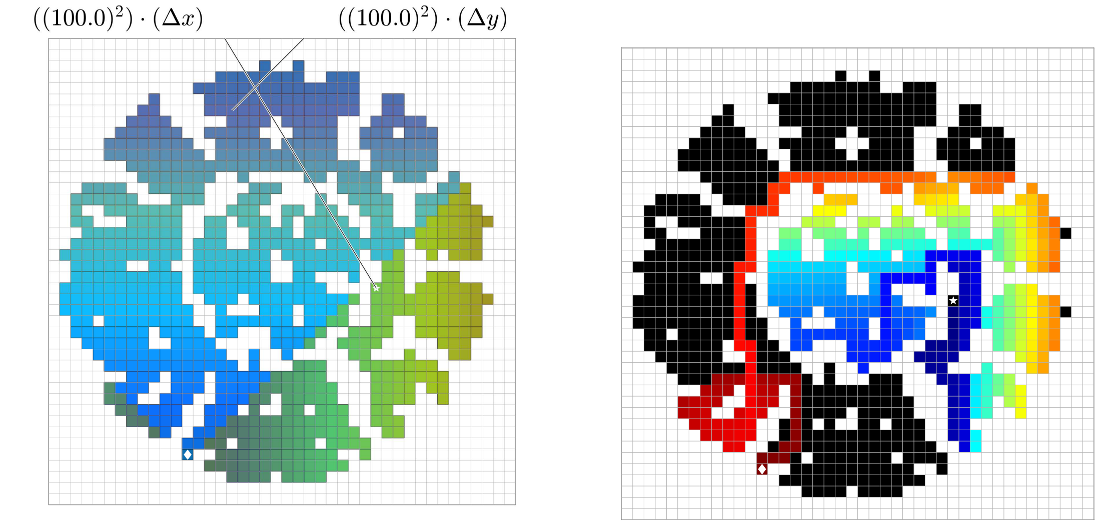

Figure 5.16: The synthesized formula $\max(100, \min(y_{\text{goal}}, \Delta y) + y)^2 \cdot \max(\Delta x, \Delta y)$ on the down sampled map.

From previous example in section 5.1.1, the Chebyshev Distance does not lead to efficient search, thus the user decisdes to add a square to it, modifying the formula into $\max(100, \min(y_{\text{goal}}, \Delta y) + y)^2 \cdot \max(\Delta x, \Delta y)^2$, however, the average speedup has not changed (figure 5.17).

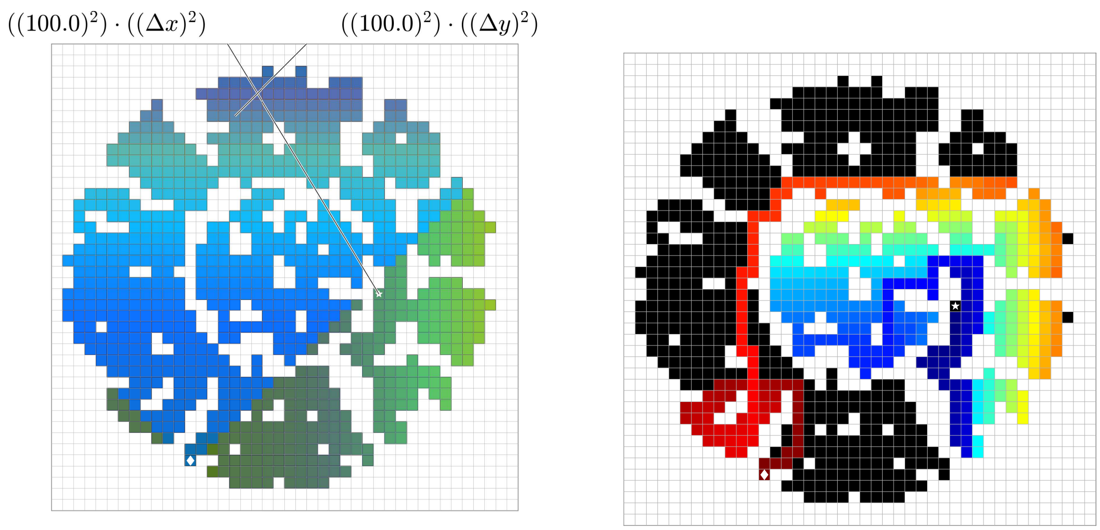

Figure 5.17: Formula modified to $\max(100, \min(y_{\text{goal}}, \Delta y) + y)^2 \cdot \max(\Delta x, \Delta y)^2$, speedup not changed.

Observe that the number 100 is produced by the synthesizer for the original map and probably does not generalize well onto other maps. Thus in order to make the formula more portable, the user replaces 100 with x_{goal}: $\max\left(x_{\text{goal}}, \min\left(y_{\text{goal}}, \Delta y\right) + y\right)^2 \cdot \max\left(\Delta x, \Delta y\right)^2$, increasing the average speedup on the down sampled map from 1.54 to 1.71.

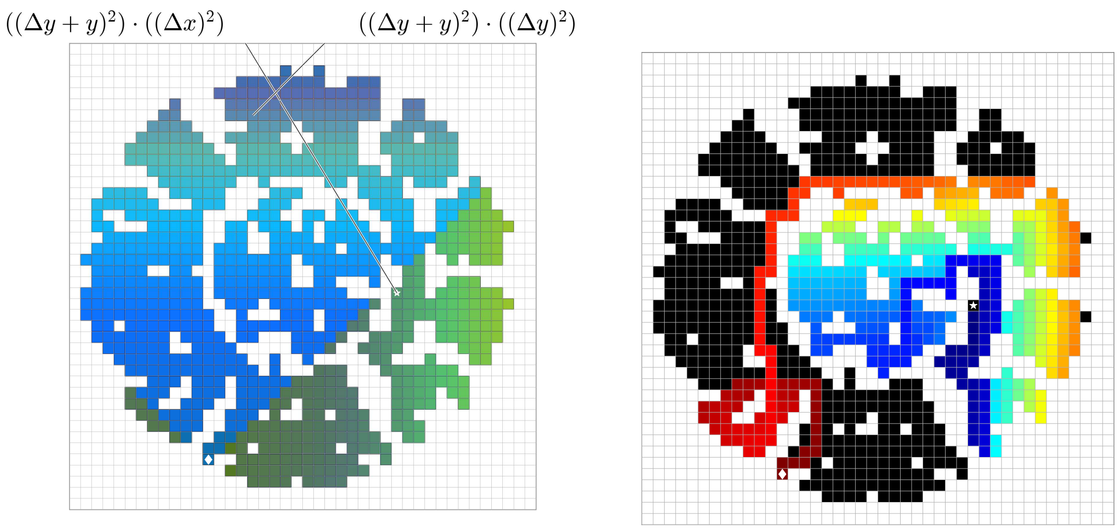

Figure 5.18: The synthesized formula $\max\left(100, \min\left(y_{\text{goal}}, \Delta y\right) + y\right)^2 \cdot \max\left(\Delta x, \Delta y\right)$ incrementally modified to $\max\left(x_{\text{goal}}, \min\left(y_{\text{goal}}, \Delta y\right) + y\right)^2 \cdot \max\left(\Delta x, \Delta y\right)^2$, increasing average speedup on down sampled map from 1.54 to 1.71 but decreasing average test speedup from 3.894 to 3.232.

The user is satisfied with resulting formula $\max\left(x_{\text{goal}}, \min\left(y_{\text{goal}}, \Delta y\right) + y\right)^2 \cdot \max\left(\Delta x, \Delta y\right)^2$ and terminates the improvement process. However the test speedup on the original map decreases from 3.894 to 3.232.

The improvement failed. Although the subformulae displayed on the map initially were $100^2 \cdot \Delta x$ and $100^2 \cdot \Delta y$, which would be considered simple, the actual subformulae also consist of $\left(y_{\text{goal}} + y\right)^2 \cdot \Delta x$ and $\left(\Delta y + y\right)^2 \cdot \Delta x$ which are not simple. These components made cause the user less able to effectively introduce bias since parts of subformulae interact with each other.

5.2.5 Case #5: ost000a

The map `ost000a` is one of the larger maps that even after down sampling to the tenth of its original size, still appears to be quite large compared to other maps. The user starts with the formula synthesized: $\max(\Delta y, \Delta x +$

$\min(\Delta x, x_{\text{goal}}))^2$, which is decomposed into three sub-formulae: Δy^2, $(\Delta x + \Delta x)^2$ and $\Delta x + x_{\text{goal}}$.

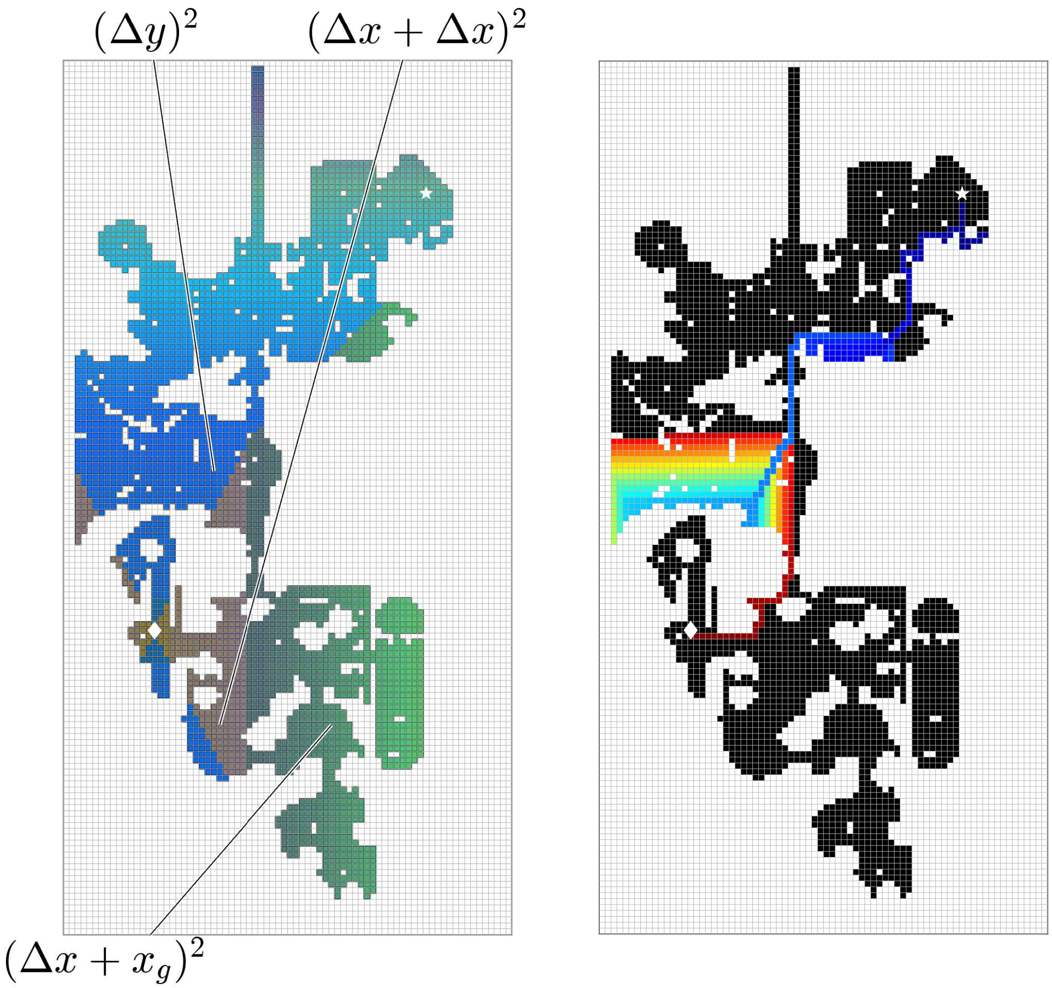

Figure 5.19: The synthesized formula $\max(\Delta y, \Delta x + \min(\Delta x, x_{\text{goal}}))^2$ on the down sampled map.

As shown by the closed list on the right of figure 5.19, there is a cluster of nodes in the middle of the map at the region governed by Δy^2. Thus the user decided to make that region less attractive to search by increasing the heuristic values in that region, adding a multiplication of two in front of Δy^2, thus the formula becomes $\max(2 \cdot \Delta y, \Delta x + \min(\Delta x, x_{\text{goal}}))^2$. The average speedup on the down sampled map improved from 1.67 to 1.87. An adversarial search problem is selected for the improved formula, shown in Figure 5.20.

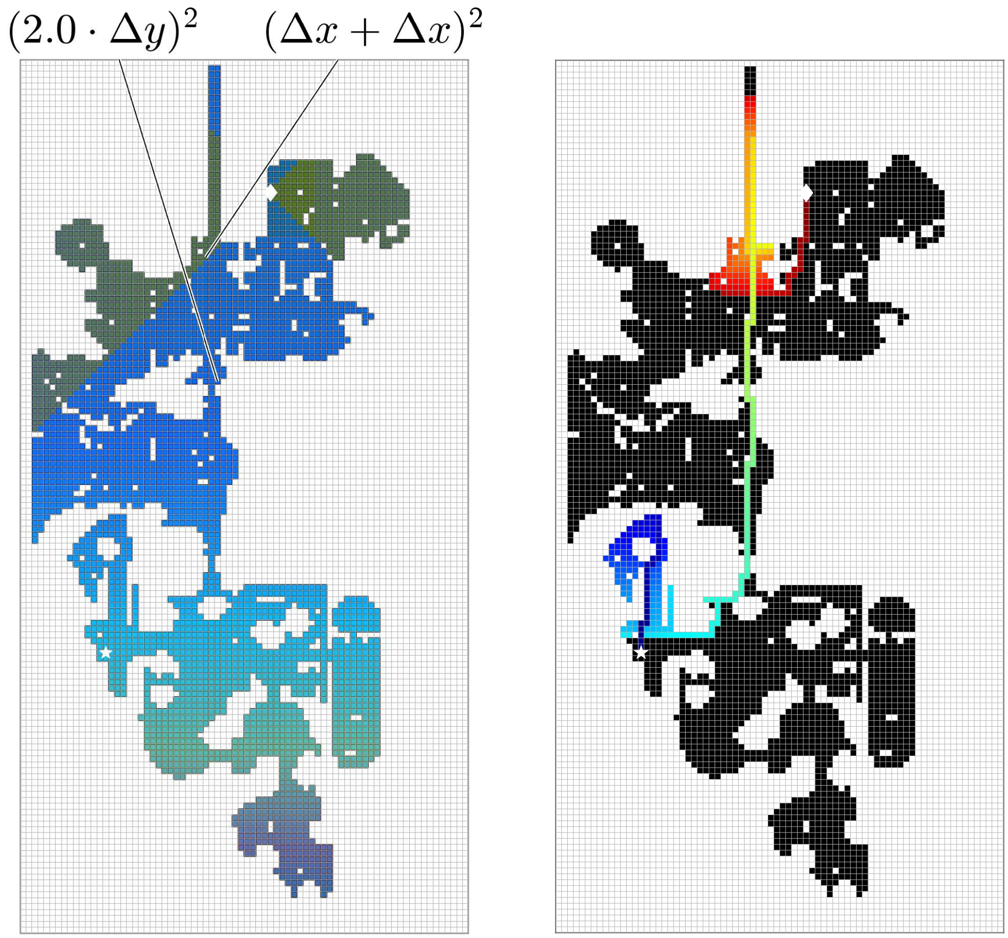

Figure 5.20: $\max(\Delta y, \Delta x + \min(\Delta x, x_{\text{goal}}))^2$ modified to $\max(2 \cdot \Delta y, \Delta x + \min(\Delta x, x_{\text{goal}}))^2$, increasing speedup from 1.67 to 1.87.

Now a new problem is adversarially selected for the newly improved heuristic formula and the expanded nodes in the middle of the map disappeared. However there are nodes expanded at the top of the map (figure 5.21). The problematic region is governed by the sub-formula $(\Delta x + \Delta x)^2$ and the user thus increases its heuristic values by modifying it to become $(\Delta x, 1.5 \cdot \Delta x)^2$, and consequentially the formula becomes $\max(2 \cdot \Delta y, \Delta x + 1.5 \cdot \min(\Delta x, x_{\text{goal}}))^2$, increasing the average speedup on the down sampled map from 1.67 to 1.88 and that of the test set on the original map from 3.665 to 3.792 (see figure 5.21).

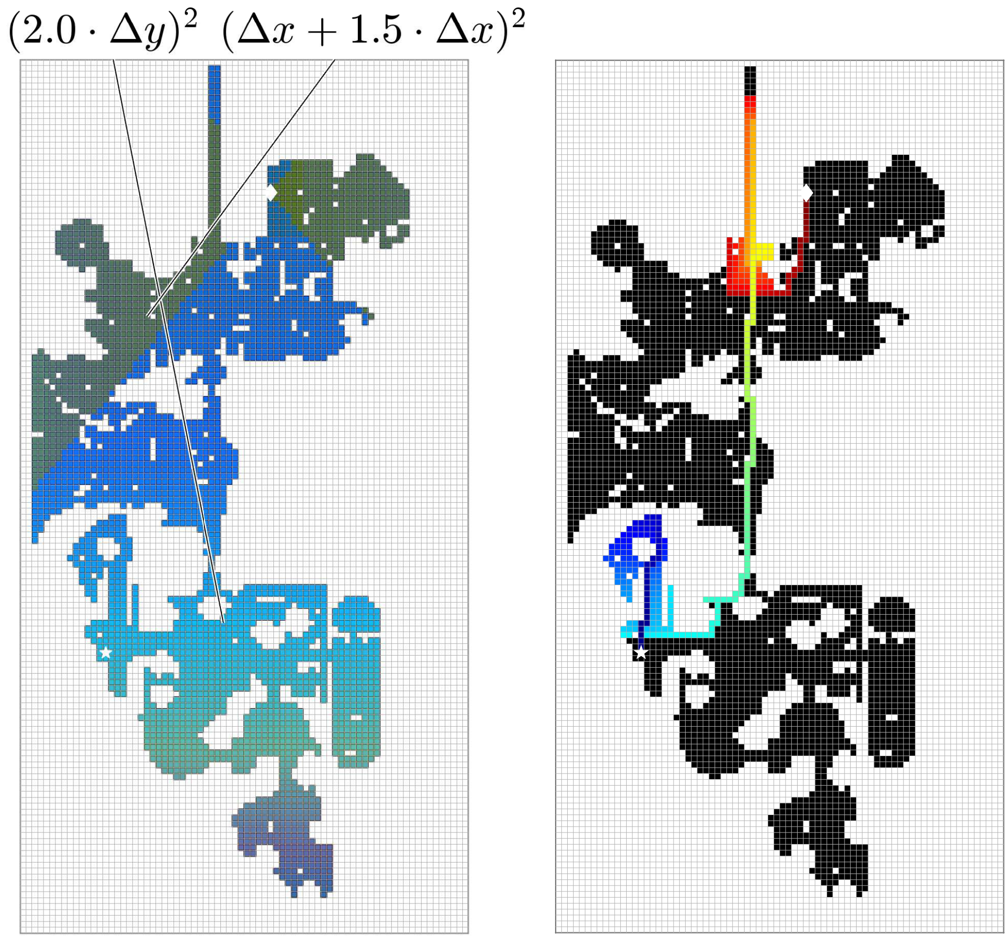

Figure 5.21: $\max(\Delta y, \Delta x + \min(\Delta x, x_{\text{goal}}))^2$ incrementally modified to $\max(2 \cdot \Delta y, \Delta x + 1.5 \cdot \min(\Delta x, x_{\text{goal}}))^2$, increasing average speedup on down sampled map from 1.67 to 1.88 and average test speedup from 3.665 to 3.792.

We again see a case where the subformulae are simple: each subformula guides search towards the goal either horizontally or vertically. The user was able to improve the formula's speedup merely by amplifying the components' magnitudes (by multiplication of constants).

5.2.6 Section Summary

The case studies show that in general when the given heuristic formula is composed of simple subformulae, the user can better introduce bias for the adversarial problem and thus improve the formula's guiding performance. When the subformulae are complicated, the user is less able to effectively introduce modifications that improve it, because the modifications may affect the formula's guidance on problems in unexpected ways.

5.3 Adversarial Map Selection

In this section we apply our method to improve a synthesized heuristic formula for a set of video-game maps. We take the same five maps from `DragonAge:Origins` from Section 5.2. Similar to the previous section, we downsample the maps for better visualizations. On each of the down sampled map, a set of 40×40 path finding problems were generated. We take the five synthesized heuristic formulae Bulitko *et al.* (2022) (one synthesized for each of the five maps) and take the formula with the highest average speedup over all $5 \times 40 \times 40$ path finding problems to start our method with.

In the previous section, for each map we selected the problem which causes the formula to expand the most number of states. However, in this section we select the problem that causes the formula to produce the lowest speedup because the sizes of the maps vary thus this way we prevent bias towards selecting problems from larger maps.

Furthermore, since the human user only perceives the one path finding problem, the modification made to the formula may not be beneficial to all problems on average. Therefore we provide the user with the option to run a light-weight genetic algorithm seeded with the current formula to optimize the average speedup over all $5 \times 40 \times 40$ pathfinding problems. The genetic algorithm runs once with a population of 20 and terminates after 3 generations. The time it takes is about 30 seconds.

The formula $\Delta y + 44.9 \cdot \max(\Delta x, \Delta y)$ was selected among the five with the highest average speedup over all $5 \times 40 \times 40$ path finding problems. The problem that causes it to produce the lowest speedup is selected and visualized in Figure 5.22. This is the same map from Section 5.2.5.

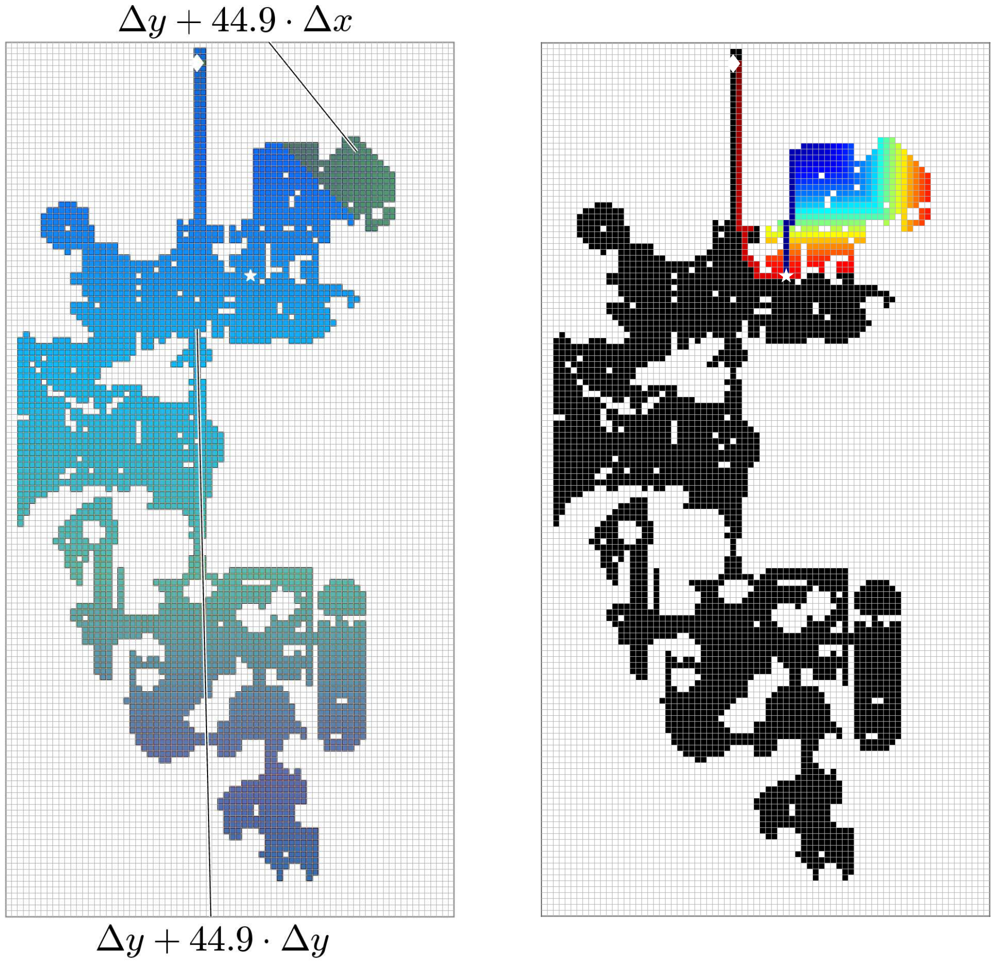

Figure 5.22: $\Delta y + 44.9 \cdot \max(\Delta x, \Delta y)$ decomposed on map `ost000a` with the problem that causes it to produce the lowest speedup.

The formula is decomposed into sub-formulae $\Delta y + 44.9 \cdot \Delta x$ and $\Delta y + 44.9\Delta y$. As shown in the closed list plot, most states expanded lie in the region governed by $\Delta y + 44.9 \cdot \Delta y$ because this sub-formula considers only the vertical distance between the current state and the goal state. The user therefore modified the formula to become $\Delta x + 44.9 \cdot \max(\Delta x, \Delta y)$ however, after reevaluation, there is a sharp drop in the average speedup. Therefore the user undid the modification and ran the genetic algorithm which returns an improve heuristic formula $\Delta x + \Delta y + 44.9 \cdot \max(\Delta x, \Delta y)$ visualized in Figure 5.23.

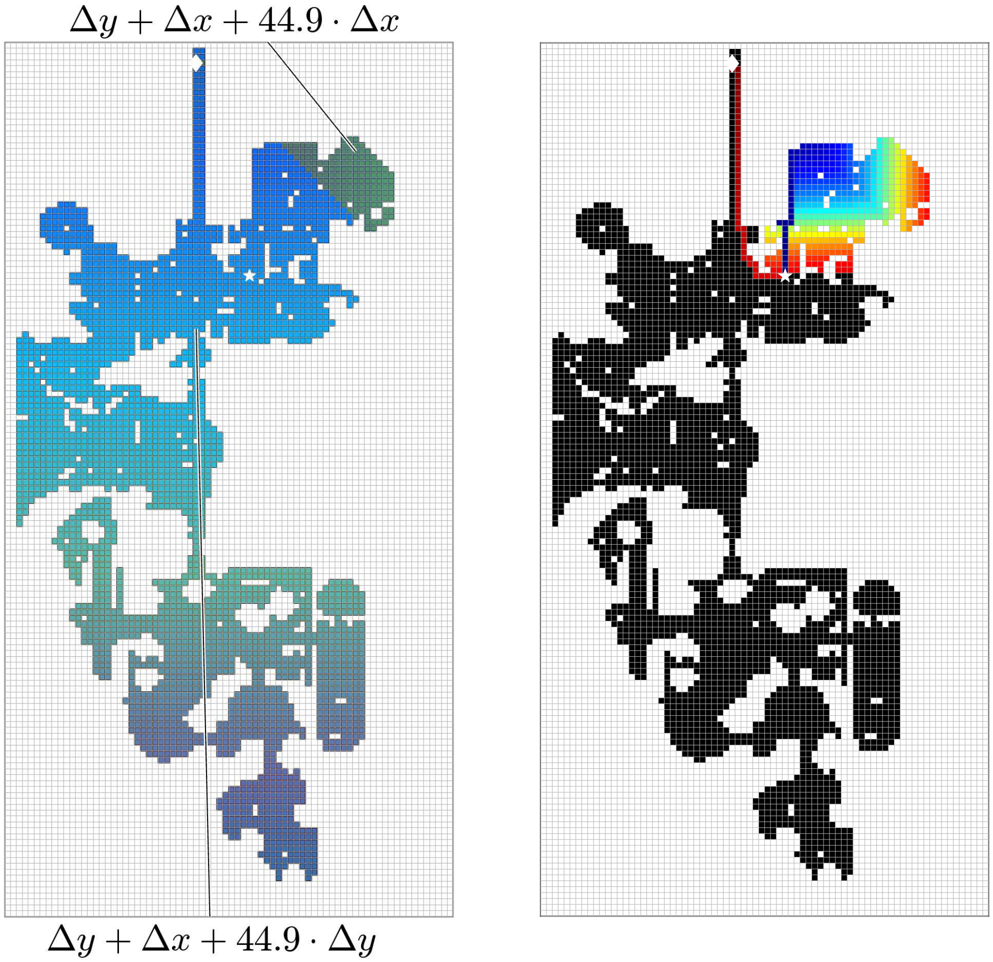

Figure 5.23: The genetic algorithm improved $\Delta y + 44.9 \cdot \max(\Delta x, \Delta y)$ to $\Delta y + \Delta x + 44.9 \cdot \max(\Delta x, \Delta y)$ that improved the average speedup from 1.569 to 1.602.

Although the problem selected did not change, the average speedup over all problems increased. The new formula generated by the genetic algorithm appropriately inserted the Δx component into the formula which nudges the search to explore horizontally towards the goal states.

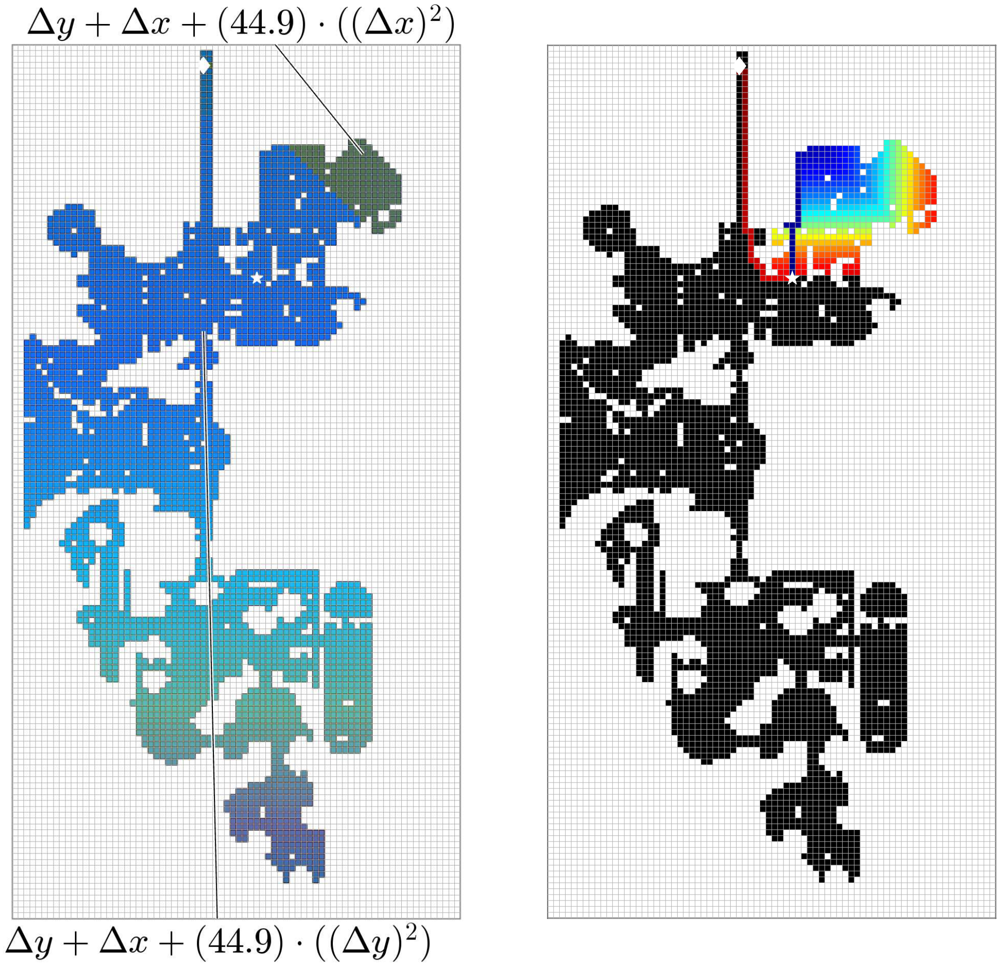

Figure 5.24: The genetic algorithm improved $\Delta y + 44.9 \cdot \max(\Delta x, \Delta y)$ to $\Delta y + \Delta x + 44.9 \cdot \max(\Delta x, \Delta y)^2$ that improved the average speedup above 1.602.

The user invokes the genetic algorithm again and got the newly improved formula $\Delta y + \Delta x + 44.9 \cdot \max(\Delta x, \Delta y)^2$. The modification proposed by the genetic algorithm amplifies the $\max(\Delta x, \Delta y)$ component by applying a square operator to it. The average speedup has increased.

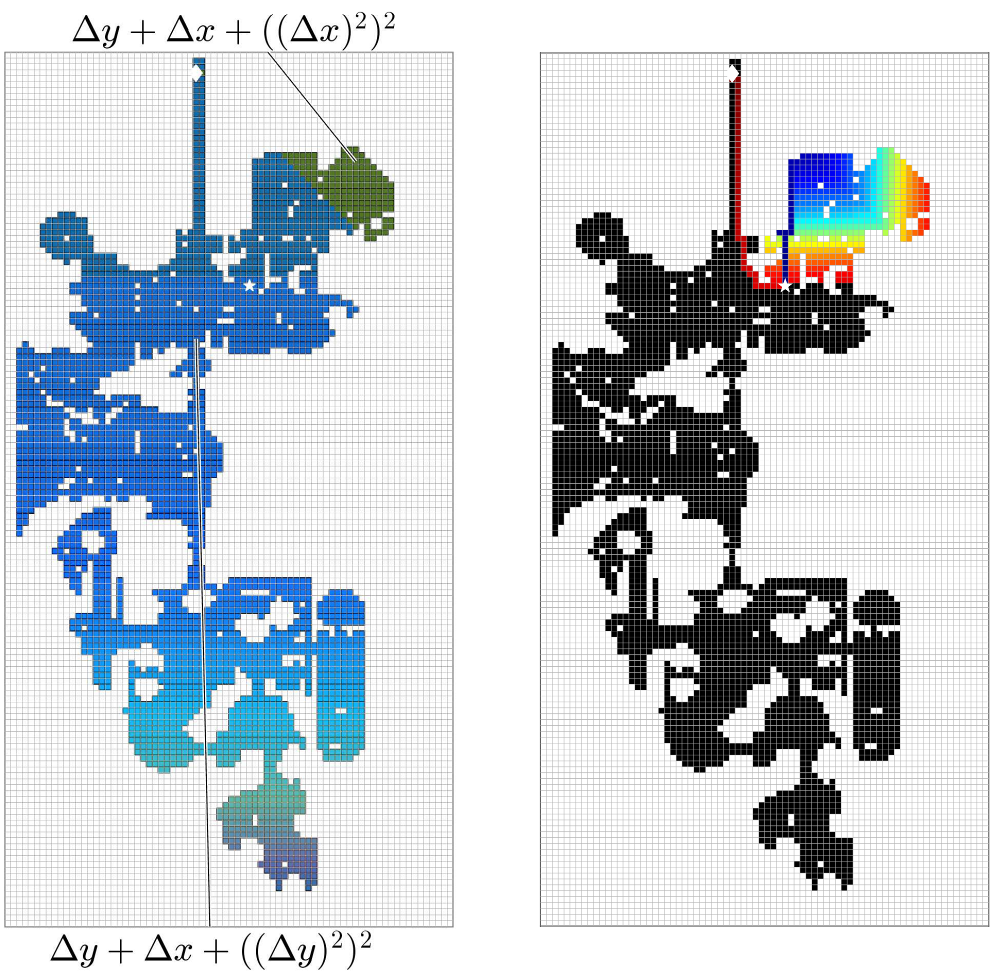

Figure 5.25: The user modifies $\Delta y + \Delta x + 44.9 \cdot \max(\Delta x, \Delta y)^2$ to become $\Delta y + \Delta x + \max(\Delta x, \Delta y)^4$ increasing the average speedup.

Taking inspiration of amplifying the $\max(\Delta x, \Delta y)$ component, the user replaced the multiplication of 44.9 by a square operator in hope of both achieving the same magnitude amplification effect and making the formula more portable (by eliminating the seemingly map-specific multiplier 44.9). Thus the user modified the formula to become $\Delta y + \Delta x + \max(\Delta x, \Delta y)^4$ which increased the average speedup. The user attempted to add another square operator to the $\max(\Delta x, \Delta y)$ component in the hope to keep increasing the average speedup, however the new formula caused a drop in average speedup.

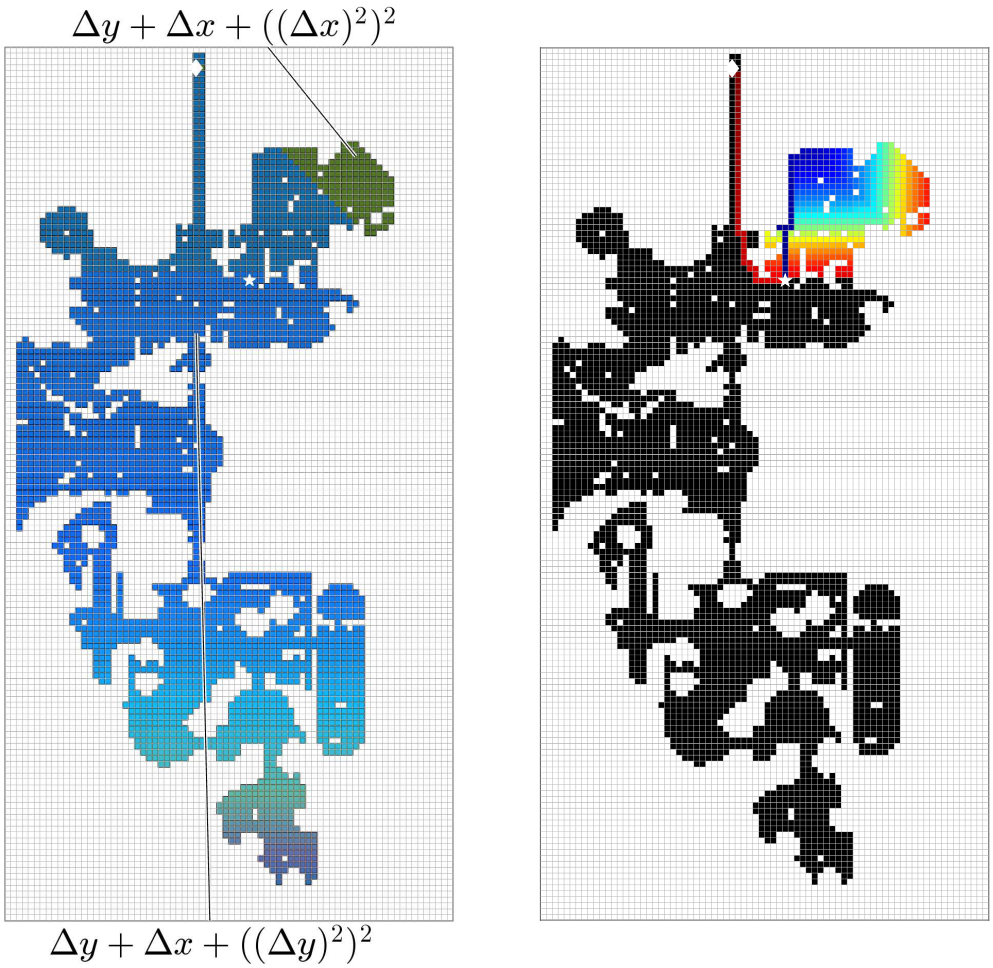

Figure 5.26: Increasing the test speedup from 3.721 to 3.835.

The user undid the modification and consulted the genetic algorithm to improve the formula, however the genetic algorithm returned the same formula:

$$\Delta y + \Delta x + \max(\Delta x, \Delta y)^4$$

Thus the user terminates the improvement cycle with the resulting formula. The new formula was evaluated on the held-out $5 \times 200 \times 200$ pathfinding problems over all original video game maps and has increased the average speedup from 3.721 to 3.835.

In this case we see that the complicated subformulae $\Delta y + 44.9 \cdot \Delta x$ again caused the user to be unable to make improving modifications. Almost all attempts made by the user failed to improve the speedup of the formula.

Assisted by the genetic algorithm, the user was able to improve the speedup of the formula by replacing the constant 44.9 with a square operator.

From the case studies it is clear that the user can be more likely to make improvements to the formula when the initial formula has simple subformulae. Thus it is beneficial if the synthesiser can be constrained to synthesize formulae with short subformulae. However if such constraints on subformula size causes the speedup of synthesized formulae to drop substantially, it is no better to use formulae with long subformulae and better speedup. To answer the question "will constraining on subformula size hurt average performance?" requires empirical work which we present in Chapter 6.

Chapter 6

Synthesizing User Improvement Conducive Heuristics

In the previous chapter we applied our explanation algorithm to some heuristic formulae synthesized for pathfinding problems on video game maps used in the work by Bulitko *et al.* 2022, where they took 24 MovingAI (Sturtevant 2012) maps from the video games *Dragon Age: Origins* (`DAO`) and *StarCraft* (`SC`) split into four sets `DAO-A`, `DAO-B`, `SC-A` and `SC-B` each containing six maps. We adopt the same maps as our test bed.

We adopt the genetic programming algorithm from the work of Wang *et al.* 2023 as our formula synthesizer. Since our focus is on the effect of restricting subformulae sizes on A* speedup of the synthesized heuristics, we keep the synthesizer largely identical to the previously published algorithm (Wang *et al.* 2023) with the removal of the limit of consecutive stagnate generations μ as a termination condition. Thus a trial only terminates if the budget b (i.e., allowed number of state expansions) runs out. We set our baseline heuristic, the weighted Manhattan distance, with the set of weights $\{1, 1.1, \ldots, 1.9, 2, 3, 4, \ldots, 9, 10\}$.

Our empirical evaluation seek to address the following two questions. First, will constraining on subformula size hurt average performance? Second, will extending the grammar with a conditional increase speedup of synthesized formulae?

To address the two questions we run formula synthesis under four settings: (1) $\mathbf{A} + \mathbf{\Gamma}$: original grammar Γ with no subformula size restriction (this is in line with published work); (2) $\mathbf{A}^{\eta} + \mathbf{\Gamma}$: original grammar with an upper bound on subformula size; (3) $\mathbf{A} + \mathbf{\Gamma}^{+}$: extended grammar and no subformula size restrictions and (4) $\mathbf{A}^{\eta} + \mathbf{\Gamma}^{+}$: extended grammar and an upper bound on subformula size.

In generating problem sets to be used during synthesis, we consider two settings: different problems with varied goal locations and all problems shareing a single goal location. Intuitively the latter is easier to synthesize a heuristic formula for.

A single synthesis trial was run on a quad-core cluster node with 16Gb of RAM.

6.1 Synthesis on Problems with Multiple Goal Locations

For each map from each set, we randomly initialize 3 goal locations and 3 start locations for each goal location ($3 \times 3 = 9$ start-goal pairs) as the training set. We run a total of 16 trials for each map and the synthesis output of each trial is evaluated on a 100×100 validation problem set on that map. The formulae with the highest/best validation speedup out of the 16 outputs is evaluated on the 200×200 test problem set for that map (this is in line with the work of Bulitko *et al.* 2022). The speedup of that formula on the test problem set is referred to as the *test speedup*. We run synthesis four times and average the test speedups.

There are two grammars and four values of the upper bound on the subformula size $\eta \in \{1, 3, 5, \infty\}$. An upper bound of ∞ means no size restriction is imposed on subformula size during synthesis.

As shown by Figure 6.1, upper bounds of 3 and 5 on subformula size does not appear to decrease the resulting speedup. Setting the upper bound to 1, however, does have a negative effect as the subformulae are forced to be one token in length (i.e., too limited). However, synthesis with the extended grammar does not appear to find better heuristics than that with the original grammar.

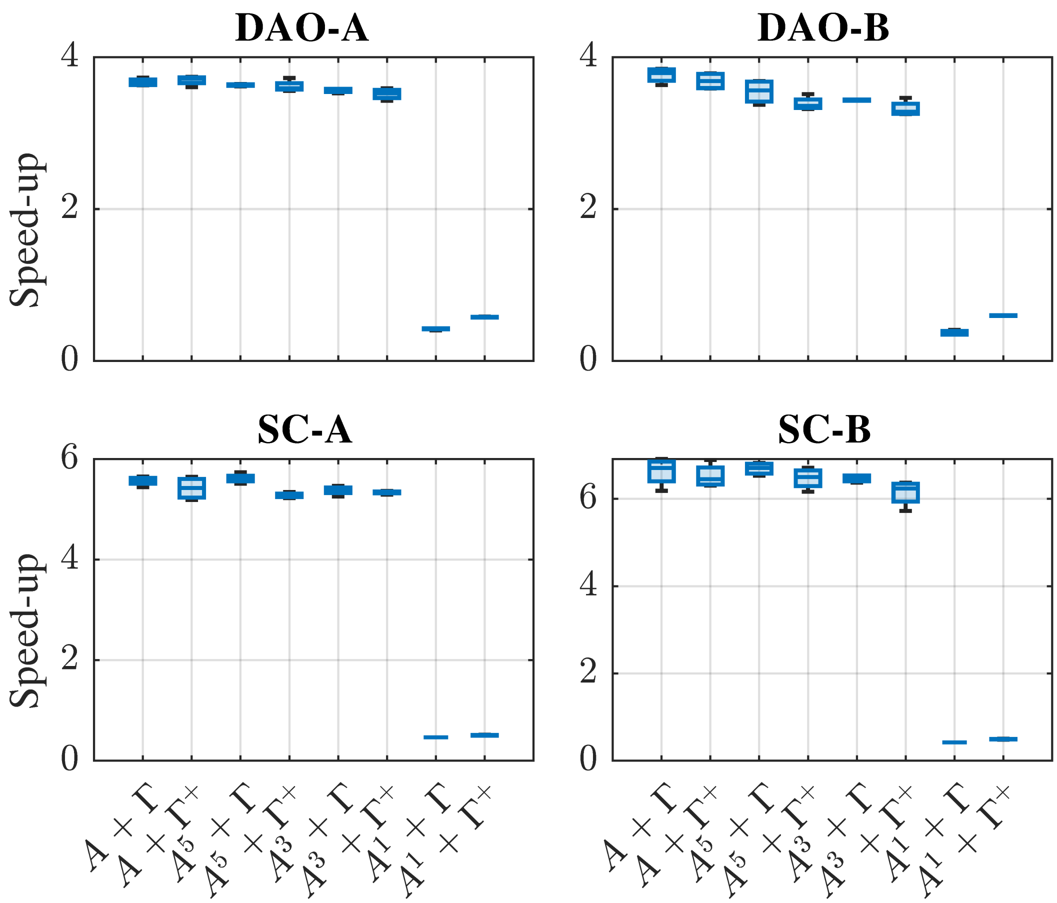

Figure 6.1: Test speedup for various synthesis configurations. A indicates lack of subformula size restriction. A^1 indicates the upper bound $\eta = 1$ and so forth. The boxes show the quantiles and the whiskers show the min and max values over the four trials.

Tables 6.1, 6.2 and 6.3 list all the distinct subformulae synthesized for different maps when setting η to 5, 3 and 1 respectively. The left column details the map the subformula was synthesized on and the right column details all subformulae synthesized for that map. We removed duplicate subformulae from the tables and manually simplified subformulae for readability. Note that the manual simplifications only changed the formulae's apparent size, so $\eta = 3$ for instance was a limitation before simplification.

Table 6.1: Synthesized subformulae for the multi-goal setting for $\eta = 5$.

Map	**Heuristic subformula**
brc202d	$\mathbf{f_1} = x_{\text{goal}} \cdot \Delta x$, $\mathbf{f_2} = (y + y_{\text{goal}}) \cdot \Delta y$
den000d	$\mathbf{f_3} = \Delta y^4$, $\mathbf{f_4} = \Delta x^4$
den501d	$\mathbf{f_5} = \Delta y^2$, $\mathbf{f_6} = \Delta x^2 \cdot \Delta x$
lak505d	$\mathbf{f_7} = \Delta x^2$, $\mathbf{f_8} = x_{\text{goal}} \cdot y$
orz103d	$\mathbf{f_9} = y \cdot \Delta y$
brc100d	$\mathbf{f_{10}} = \Delta x^8$, $\mathbf{f_{11}} = (x_{\text{goal}} \cdot \Delta y^2)^2$
brc201d	$\mathbf{f_{12}} = \Delta x^2 \cdot \sqrt{\Delta x}$
den505d	$\mathbf{f_{13}} = y^4$
lak100c	$\mathbf{f_{14}} = \Delta y \cdot x_{\text{goal}} - x$, $\mathbf{f_{15}} = \Delta x^2 + y_{\text{goal}}$
orz701d	$\mathbf{f_{16}} = \Delta y^2 + \Delta x$
orz702d	$\mathbf{f_{17}} = (y_{\text{goal}} \cdot \Delta y)^2$
ShroudPlatform	$\mathbf{f_{18}} = x_{\text{goal}} \cdot \Delta y$
SpaceAtoll	$\mathbf{f_{19}} = \Delta x + \Delta y^2$
Triskelion	$\mathbf{f_{20}} = \Delta y \cdot x_{\text{goal}}$, $\mathbf{f_{21}} = \Delta x \cdot x_{\text{goal}}$
Archipelago	$\mathbf{f_{22}} = \Delta x^2 - \Delta x$
Brushfire	$\mathbf{f_{23}} = \Delta y \cdot (y + \Delta y)$
Caldera	$\mathbf{f_{24}} = \sqrt{x} \cdot \Delta x$
CatwalkAlley	$\mathbf{f_{25}} = \sqrt{y_{\text{goal}}} \cdot \Delta x$

Table 6.2: Synthesized subformulae for the multi-goal setting for $\eta = 3$.

Map	**Heuristic subformula**
brc202d	$\mathbf{f_{26}} = \Delta x^2$, $\mathbf{f_{27}} = \Delta y^2$
den501d	$\mathbf{f_{28}} = \Delta y \cdot x_{\text{goal}}$, $\mathbf{f_{29}} = \Delta x^4$
lak505d	$\mathbf{f_{30}} = \Delta x \cdot y_{\text{goal}}$
brc100d	$\mathbf{f_{31}} = x_{\text{goal}} \cdot \Delta x$, $\mathbf{f_{32}} = \Delta y \cdot y_{\text{goal}}$
brc201d	$\mathbf{f_{33}} = \Delta x \cdot x_{\text{goal}}$, $\mathbf{f_{34}} = \Delta y^4$
den505d	$\mathbf{f_{35}} = y^2$
orz702d	$\mathbf{f_{36}} = x \cdot \Delta x$

Table 6.3: Synthesized subformulae for the multi-goal setting for $\eta = 1$.

Map	**Heuristic subformula**
brc202d	$\mathbf{f_{38}} = \Delta x$, $\mathbf{f_{39}} = y$, $\mathbf{f_{40}} = \Delta y$, $\mathbf{f_{41}} = x$, $\mathbf{f_{42}} = x_{\text{goal}}$
ost000a	$\mathbf{f_{43}} = y_{\text{goal}}$

With upper bound $\eta = 1$, a synthesized subformula can only be a terminal in our grammar which means that the A* search may overemphasize the g values when computing the priority score $f = g + h$. With larger upper bounds $\eta = 3$ or $\eta = 5$, the synthesis can use more complex subformula (e.g., $\Delta x + \Delta y^2$) which may prevent the dominance of g values when computing priority scores.

Our motivation for imposing an upper bound on subformula size was to achieve an easier connection between a subformula and the direction it guides the search in a region of the map. To illustrate this with an actually synthesized subformula, consider $\mathbf{f_1} = x_{\text{goal}} \cdot \Delta x$. Given a goal coordinate x_{goal}, the subformula $\mathbf{f_1}$ simply computes a scaled horizontal distance between a state and the goal. Therefore the search will be guided in the horizontal direction towards the goal until an obstacle is encountered.

6.2 Synthesis on Problems with a Shared Goal Locations

For the shared goal location experiments, we pick the goal location at the center of each map for all problems. If the center location of the map is not empty we pick the available location closest to the center by Manhattan distance (with arbitrary tie breaking).

During training, we randomly initialize 9 start locations for each synthesis trial. We randomly generate 400 and 800 start locations for the validation and test problem sets respectively on each map.

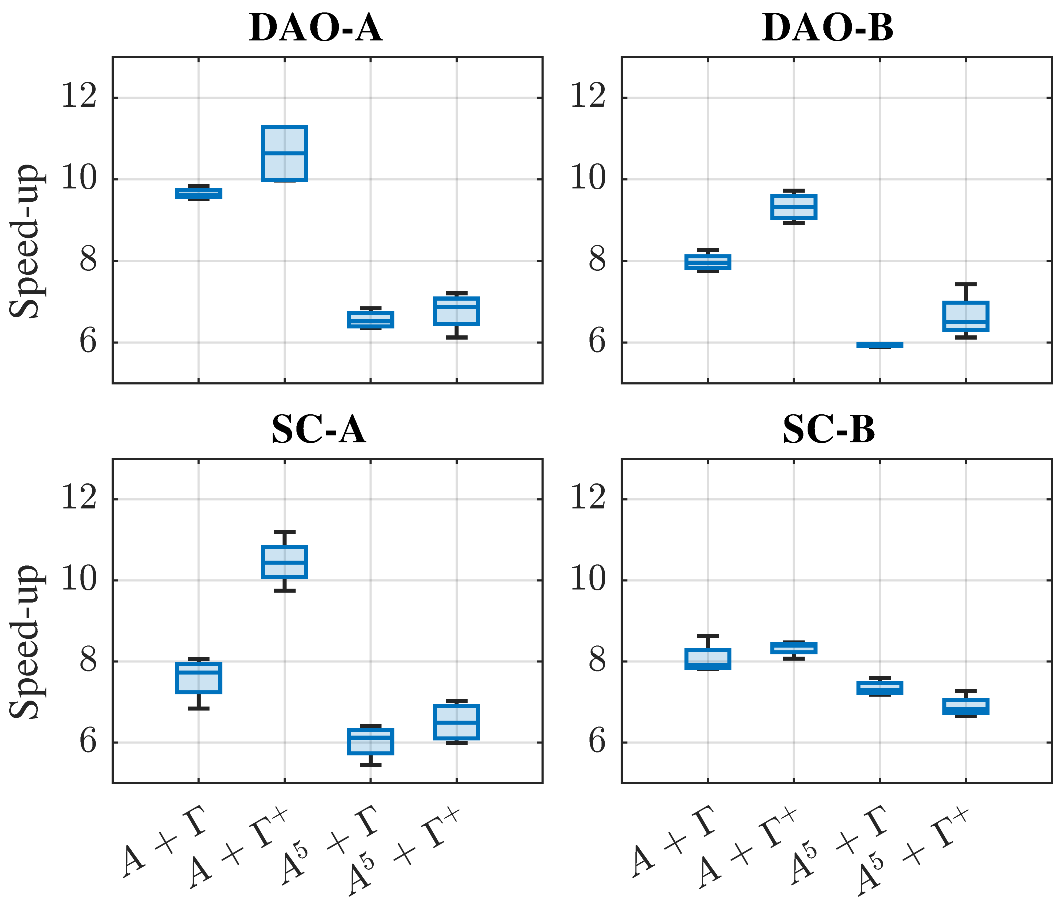

Figure 6.2: Speedup comparison under the single-goal setting with subformula size limit set to $(\infty, 5)$.

In the shared goal setting, an upper bound on subformula size appears to have a more negative impact on speedup. On the positive side, extending the original grammar with conditionals allows the synthesis to produce formulae with higher speedups.

We suspect this is because under the shared goal setting, the specifics of the map can be better captured by the subformulae since the goal location is fixed. Therefore the quality of subformulae is more dominant than that of the map region assignment functions derived from the formula. By extending the grammar with the conditionals, more flexibility is given to the subformulae because they become more independent from other components of the formula, resulting in higher average speedups. On the other hand, by setting an upper bound on the size of the subformulae, the large part of expression power of the formula is cut off thus we see a more negative impact on the speedup for the shared goal setting than the multiple goal setting.

6.3 Chapter Summary and Iterative Improvement of a Constrained Formula

Heuristic formula synthesis constrained by an upper bound on subformula size does not appear to decrease the resulting A* speedup unless the upper bound is too low (Figure 6.1) under the multiple goal setting. However, under the shared goal setting, the restriction on subformula size impact the speedup more negatively.

Additionally, when pathfinding problems on a map share a goal location, grammar extended with conditionals allows the synthesizer more flexibility to compose heuristic formulae with higher speedups than those with the original grammar (Figure 6.2).

Now we test the iterative improvement approach with a synthesized heuristic formula with constraint of 3 on subformula size. We pick the failed case from Section 5.2.2 of Chapter 5 on the video-game map `den000d`. Compared to the formula synthesized without constraints from the work by Bulitko *et al.* 2022, the constrained synthesizer generated the formula $\max(\Delta x^2, \Delta y^2)$ with subformulae Δx^2 and Δy^2 each having size no greater than 3.

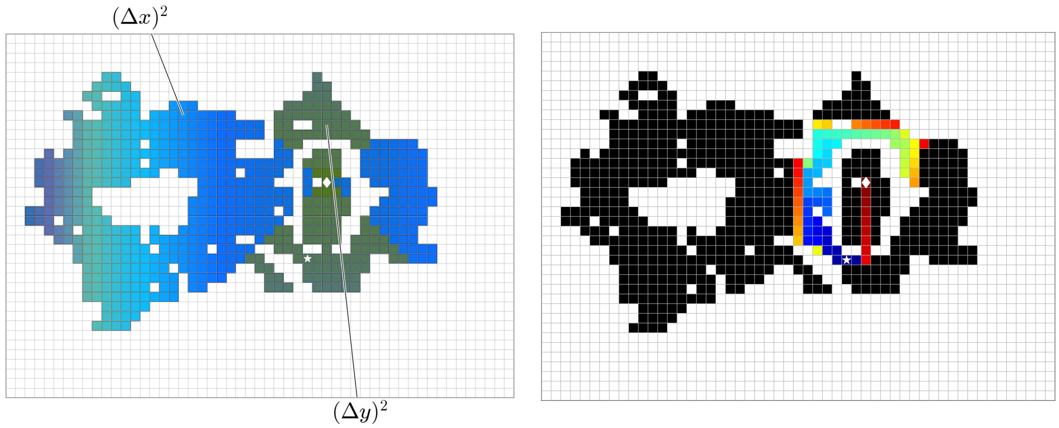

Figure 6.3: The decomposed subformulae with the closed list on the down-sampled map.

The formula's average test speedup is 5.258, slightly lower than 5.268, that of the previous formula. The formula's average speedup on the 40×40 pathfinding problems on the down sampled map is 1.673. Consider the adversarial problem selected against the formula shown in Figure 6.3. Most needlessly expanded states are to the left of the start state in the region governed by Δx^2. In order to make that region less attractive to search, the user multiplies the subformula by a constant 1.5 thus end up with the new formula $\max(1.5 \cdot \Delta x^2, \Delta y^2)$.

Figure 6.4 shows the newly selected adversarial problem. The closed list seems substantially reduced and the speedup on the down-sampled map increased from 1.673 to 1.713. The user is satisfied with the result and terminates the process. After evaluating the new formula on the held-out problem set, the average test speedup increased from 5.258 to 5.319. The resulting formula performs even better than the formula synthesized without constraints yet it is more compact.

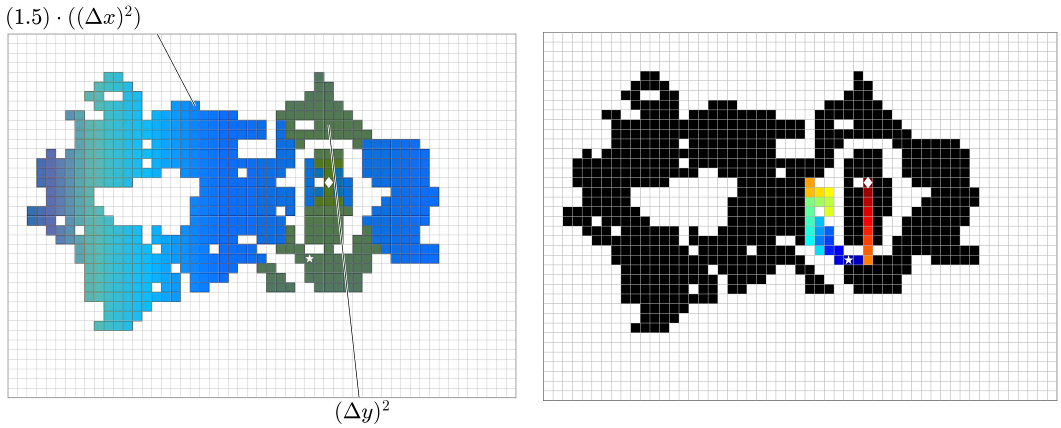

Figure 6.4: Increasing the speedup on downsampled map from from 1.673 to 1.713 and the test speedup from 5.258 to 5.319. Even greater than 5.268, the test speedup of synthesized formula before.

To conclude, we have shown that in general synthesis with constrained subformula size does not substantially hurt the guiding performance of produced heuristic formulae. Furthermore, the synthesized formulae with constrained subformula size, when applied in the iterative improvement approach aforementioned, can allow the human user to introduce biases more effectively. Therefore the answer to the first research question is that by constraining subformula size, performance is not hurt substantially. The answer to the second research question is that by extending the grammar with a conditional, speedup of synthesized formula increase on average under the condition that all problems on a map share a common goal location.

Chapter 7

Future Work

Future work will investigate explaining and improving synthesized heuristics in other settings such as agent-centered real-time heuristic search (Bulitko 2020) or multi-agent pathfinding (Wang *et al.* 2023). For instance, in real-time heuristic search a formula-based heuristic provides the initial heuristic values for an agent. However, since the agent modifies its heuristic as it moves about the map, our explanation method would need to be extended to take the modifications into account. A related future research direction is to extend our method to explain reinforcement learning policies.

Future work will also apply formula synthesis in other domains. A promising direction is to apply the synthesis techniques to generate formulae to simulate bird songs (Bistel *et al.* 2022). Traditionally domain experts handcraft formulae to simulate bird songs in order to produce more training data for classifiers. This process is tedious and expensive. By applying the formula synthesis method one can potentially generate formulae for any bird species thus expanding training data for species that lack them. Follow-up work will investigate the explanability of the synthesized formulae.

The method we presented in this thesis relied on visualizations. An alternative is to use a large language model to attempt to generate a natural language explanation of each subformula's operation in the spirit of Bashir *et al.* 2023. Although large language models are not good at explaining the entire heuristic formula (as shown in Appendix A), it can be applied to explain subformulae to facilitate the human user in understanding them.

Chapter 8

Conclusions

We presented a novel method for explaining a synthesized heuristic formula via automatically decomposing it into subformulae and associating each with a region of the search map. We also presented an application of our approach for a user to iteratively improve a given formula. We presented case studies in single-problem, multi-problem and multi-map settings and identified the main reason behind the failed cases: complicated subformulae, causing useful modifications more difficult to add by a user.

Therefore the iterative improvement approach will likely work better when the subformulae are short. Thus we introduced constrained heuristic synthesis to produce heuristics with short subformulae and tested the iterative improvement on a previously failed case with the newly synthesized constrained formula. The new case study was successful.

We empirically showed that a restriction on subformula size, unless too restrictive, does not adversely affect heuristic speedup. Finally we showed that by extending the original grammar with conditionals we allow the synthesizer to find heuristic formulae with better performance when the problems share a common goal location.

Furthermore we applied a pre-trained large language model to explain a synthesized formula but it's explanations are too general for a reconstructor to come up with a formula resembling the original. However it is likely that pre-trained large language models can well explain short subformulae thus facilitate a user in understanding the entire formula.

References

[1] J.-M. Alkazzi, A. Rizk, M. Salomon and A. Makhoul, 'Mapfaster: A faster and simpler take on multi-agent path finding algorithm selection,' in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[2] S. Almagor and M. Lahijanian, 'Explainable Multi Agent Path Finding,' in *Proceedings of Autonomous Agents and Multiagent Systems conferece (AAMAS)*, 2020, pp. 34–42.

[3] Z. Bashir, M. Bowling and L. H. Lelis, 'Assessing the Interpretability of Programmatic Policies with Large Language Models,' *arXiv preprint arXiv:2311.06979*, 2023.

[4] R. A. Bistel, A. Martinez and G. B. Mindlin, 'Neural networks that locate and identify birds through their songs,' *The European Physical Journal Special Topics*, pp. 185–194, 2022.

[5] V. Bulitko, 'Evolving Initial Heuristic Functions for Agent-Centered Heuristic Search,' in *Proceedings of the IEEE Conference on Games (CoG)*, 2020, pp. 534–541.

[6] V. Bulitko, Y. Björnsson, N. R. Sturtevant and R. Lawrence, 'Real-Time Heuristic Search for Pathfinding in Video Games,' in *Artificial Intelligence for Computer Games*, 2011.

[7] V. Bulitko, S. P. Hernandez and L. H. S. Lelis, 'Fast Synthesis of Algebraic Heuristic Functions for Video-game Pathfinding,' in *Proceedings of the IEEE Conference on Games (CoG)*, 2021, pp. 1–5.

[8] V. Bulitko and R. Lawrence, 'Game-map Pathfinding with Per-Problem Selection of Synthesized Heuristics,' in *Proceedings of the IEEE Conference on Games (CoG)*, 2023, pp. 1–4.

[9] V. Bulitko, S. Wang, J. Stevens and L. H. Lelis, 'Portability and Explainability of Synthesized Formula-based Heuristics,' in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 2022, pp. 29–37.

[10] J. Chen and N. R. Sturtevant, 'Necessary and sufficient conditions for avoiding reopenings in best first suboptimal search with general bounding functions,' in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[11] E. Deza, M. M. Deza, M. M. Deza and E. Deza, *Encyclopedia of distances*. Springer, 2009.

[12] S. P. Hernandez and V. Bulitko, 'Speeding Up Heuristic Function Generation via Automatically Extending the Formula Grammar,' in *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2021, pp. 233–235.

[13] O. Kaduri, E. Boyarski and R. Stern, 'Algorithm selection for optimal multi-agent pathfinding,' in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2020.

[14] D. Lyu, F. Yang, B. Liu and S. Gustafson, 'SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning,' in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 2970–2977.

[15] A. Nazari, Y. Huang, R. Samanta, A. Radhakrishna and M. Raghothaman, 'Explainable Program Synthesis by Localizing Specifications,' in *Proceedings of the Association for Computing Machinary (ACM) on Programming Languages*, 2023, pp. 2171–2195.

[16] J. Ren, V. Sathiyanarayanan, E. Ewing, B. Senbaslar and N. Ayanian, 'MAPFAST: A deep algorithm selector for multi agent path finding using shortest path embeddings,' in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, F. Dignum, A. Lomuscio, U. Endriss and A. Nowé, Eds., 2021.

[17] P. Saunders, V. Bulitko, S. Ondrčková and R. Barták, 'Formula- and memory-based heuristics in video-game pathfinding,' in *2024 IEEE Conference on Games (CoG)*, 2024, pp. 1–4.

[18] D. Sigurdson, V. Bulitko, S. Koenig, C. Hernandez and W. Yeoh, 'Automatic Algorithm Selection in Multi-Agent Pathfinding,' *arXiv preprint arXiv:1906.03992*, 2019.

[19] D. Silver, 'Cooperative pathfinding,' in *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment (AIIDE)*, 2005.

[20] N. R. Sturtevant, 'Benchmarks for Grid-based Pathfinding,' *IEEE Transactions on Computational Intelligence and AI in Games*, no. 2, pp. 144–148, 2012.

[21] N. R. Sturtevant, A. Felner, M. Barrer, J. Schaeffer and N. Burch, 'Memory-based Heuristics for Explicit State Spaces,' in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

[22] G. Team, R. Anil, S. Borgeaud *et al.*, 'Gemini: A family of highly capable multimodal models,' *arXiv preprint arXiv:2312.11805*, 2023.

[23] S. Wang, V. Bulitko, T. Huang, S. Koenig and R. Stern, 'Synthesizing Priority Planning Formulae for Multi-Agent Pathfinding,' in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2023, pp. 360–369.

[24] S. Wang, V. Bulitko and W. Yeoh, 'Explaining Synthesized Pathfinding Heuristics via Iterative Visualization and Modification,' in *Proceedings of the anual IEEE Conference on Games (CoG)*, 2024, pp. 1–4.

[25] S. Zhang, J. Li, T. Huang, S. Koenig and B. Dilkina, 'Learning a priority ordering for prioritized planning in multi-agent path finding,' in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 2022.

Appendix A

Large Language Model Prompts for Explaining Heuristic Formulae

Here we present the prompts we used for the pre-trained large language model to explain a synthesized heuristic formula. We used Gemini (Team *et al.* 2023) for the experiment. We tested the heuristic formula:

$$\max\left(100, \min\left(y_{\text{goal}}, \Delta y\right) + y\right)^2 \cdot \max\left(\Delta x, \Delta y\right)$$

from Section 5.2.4 of Chapter 5. Gemini generated a textual explanation with the prompts shown in Listing A.1. The explanation is shown in Listing A.2.

```
Heuristic functions guide search algorithms and have a profound impact on their performance. Consider A* search where the algorithm maintains a search frontier and the f-score computed by f = h + g determines which node in the frontier to expand next.

You are provided with the following preliminaries and helper functions in C:

int delta(int a, int b) {
  return abs(a - b);
}

int sqr(int a) {
  return a * a;
}

Here, I provide a context free grammar (CFG) of heuristic functions inside the <CFG></CFG> tag written bellow:
```

```

<CFG>
F -> T | U | B
T -> x | xg | y | yg | delta(xs, xg) | delta(ys, yg) | C
U -> sqrt(F) | |F| | -F | sqr(F)
B -> F + F | F - F | F * F | F / F | max(F, F) | min(F, F)
</CFG>

Here C is a numeric constant. x is the x coordinate of the current state, xg is the x coordinate of the goal state, y is the y coordinate of the current state and yg is the y coordinate of the goal state.

Now you have the background knowledge you need to know. I want you to write an explanation for the following heuristic function written in C that goes with the A* algorithm by explaining how it guide the search:

int heuristic(int x, int y, int xg, int yg) {
  return sqr(max(100, min(yg, delta(y, yg)) + y));
}

Note: the output of the heuristic function will be rounded to an integer.

Keep in mind the following about the search map:
1. The map's coordinates starts with (0, 0). There are no negative coordinates.
2. There are only walls and empty spaces in the map.
3. The goal position is always reachable from the start position.
4. The values of x, xg, y and yg will ALWAYS be nonnegative.
5. The action selection logic is ALWAYS to selection the state with the minimum f-score.
6. When you explain the heuristic function, explain how it will guide the search if we use the A* search algorithm.

The following 8 are the guidelines for writing an explanation for this heuristic function:
1. Write a high-level explanation and do not describe the heuristic function term by term.
2. Try to understand how this heuristic function guides search and explain in natural language for someone who wants to apply this heuristic function for their
```

```
   search.
3. DO NOT USE any quotation marks in writing the
   explanation.
4. You can mention numbers in your explanation.
5. There should be no appearance of the heuristic
   function anywhere in your explanation.
6. Avoid using any adjectives.
7. Write the explanation inside the "<explanation></
   explanation>" tag. Your explanation should start with
   "<explanation>" and end with "</explanation>".
8. Make the explaination simple and be objective.

So, following the instructions above, provide a high-
   level explanation of the provided heuristic function
   so that someone can rewrite this program from that
   summary.
```

Listing A.1: Prompts for the explainer.

The explanation is vague and it applies to almost any heuristic that contains the Δx and Δy components. After giving the explanation to the reconstructor instance, Gemini outputs the formula $\max(\Delta x, 100)$, which is not even close to the original.

```
Heuristic functions guide search algorithms and have a
   profound impact on their performance. Consider A*
   search where the algorithm maintains a search frontier
    and the f-score computed by f = h + g determines
   which node in the frontier to expand next.

You are provided with the following preliminaries and
   helper functions in C:

int delta(int a, int b) {
  return abs(a - b);
}

int sqr(int a) {
  return a * a;
}

Here, I provide a context free grammar (CFG) of heuristic
    functions inside the <CFG></CFG> tag written bellow:

<CFG>
F -> T | U | B
T -> x | xg | y | yg | delta(xs, xg) | delta(ys, yg) | C
```

```
U -> sqrt(F) | |F| | -F | sqr(F)
B -> F + F | F - F | F * F | F / F | max(F, F) | min(F, F)
</CFG>

Here C is a numeric constant. x is the x coordinate of the current state, xg is the x coordinate of the goal state, y is the y coordinate of the current state and yg is the y coordinate of the goal state.

Note: the output of the heuristic function will be rounded to an integer.

Keep in mind the following about the search map:
1. The map's coordinates starts with (0, 0). There are no negative coordinates.
2. There are only walls and empty spaces in the map.
3. The goal position is always reachable from the start position.
4. Because of the nature of the map, x, xg, y, yg will ALWAYS be nonnegative.

Now you have the background knowledge you need to know.

Here is a summary of a heuristic function generated by Gemini. Let's call it "LLM Explanation" which was generated as an explanation of a heuristic function to compute the heuristic for A* search. The natural language explanation is between the "<explanation></explanation>" tags:

<explanation>
The heuristic function estimates the distance to the goal based on the current position's y-coordinate and the goal's y-coordinate. It prioritizes exploration in the direction of the goal's y-coordinate while considering a minimum distance threshold. The function ensures a minimum estimated distance of 100 units, regardless of the actual position. This prevents the algorithm from prematurely focusing on nearby positions and encourages exploration in a broader area.
</explanation>

Your tasks are the following 6:
1. Given the instructions on how to write a heuristic formula and the explanation from Gemini (i.e., the "
```

```
LLM Explanation") provided earlier, write down the formula encoded in the explanation and reconstruct the program in the language for the heuristic formulae (you must only use the language provided).
2. You must not use any symbols (for example: &&, || etc.) outside of the given CFG. You have to strictly follow this CFG while writing the formula.
3. Write only the formula inside the "<formula></formula>" tag and the heuristic function you write should be a C++ function.
4. Do not write unneccessary symbols of the CFG such as "->", "|" etc.
5. When you encounter parentheses in an expression or code, their opening and closing positions are crucial as they indicate the inclusion of some statements within others.
6. Check the formula and ensure that it does not violate the rules of the CFG or the guidelines of writing the formula.

IMPORTANT:
For the sake of formula readability, write a shorter formula instead of a long one.
```

Listing A.2: Prompts for the reconstructor.

The verifier outputs “no” almost always, which is true most of the time.

```
Heuristic functions guide search algorithms and have a profound impact on their performance. Consider A* search where the algorithm maintains a search frontier and the f-score computed by f = h + g determines which node in the frontier to expand next.

Here, I provide a context free grammar (CFG) of heuristic functions inside the <CFG></CFG> tag written bellow:

<CFG>
F -> T | U | B
T -> x| x_g | y | y_g | \Delta x | \Delta y | C
U -> sqrt(F) | |F| | -F | F^2
B -> F + F | F - F | F * F | F / F | max(F, F) | min(F, F)
</CFG>

Here, \Delta x = |x - x_g|, \Delta y = |y - y_g| and C is a numeric constant. Now you have the background knowledge you need to know.

```

```
I have a heuristic function F written in the language provided above. It computes heuristic for A* search. Here is the heuristic function:

int heuristic(int x, int y, int xg, int yg) {
  return sqr(max(100, min(yg, delta(y, yg)) + y));
}

Then I asked the first instance of Gemini for a high-level explanation so that another instance of Gemini can regenerate the code using that explanation. The first Gemini gave me this:

<explanation>
The heuristic function estimates the distance to the goal based on the current position's y-coordinate and the goal's y-coordinate. It prioritizes exploration in the direction of the goal's y-coordinate while considering a minimum distance threshold. The function ensures a minimum estimated distance of 100 units, regardless of the actual position. This prevents the algorithm from prematurely focusing on nearby positions and encourages exploration in a broader area.
</explanation>

Now I want to know if the explanation provides term by term description of the heuristic function or an exact instruction of how to write the heuristic function. Answer with yes or no first and then explain why.
```

Listing A.3: Prompts for the verifier.