Empirical Models of Heuristic Search in AI Planning and Neural Sequence Decoding

by

Eldan Cohen

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Abstract

Empirical Models of Heuristic Search in
AI Planning and Neural Sequence Decoding

Eldan Cohen
Doctor of Philosophy
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2021

Heuristic search algorithms are widely used in both AI planning and the decoding of sequences from deep neural networks. In recent years, several lines of work have highlighted different factors that impact the empirical performance of heuristic search algorithms. However, a principled empirical understanding of the search behavior of these heuristic search algorithms has yet to be developed.

Empirical models, such as the phase transition and the heavy-tailed behavior, have been central to the development of empirical understanding of combinatorial search problems such as constraint satisfaction problems (CSP) and satisfiability (SAT). In this dissertation, we investigate the use of empirical models to explain the behavior of heuristic search algorithms in AI planning and neural sequence decoding and support the development of more efficient search algorithms.

In AI planning, we develop empirical models for problem difficulty of greedy best first (GBFS), the most commonly used algorithm for satisficing planning. First, we establish the existence of a phase transition in the solubility of planning problems and investigate its implications to problem difficulty. Then, we demonstrate the heavy-tailed behavior of GBFS and provide a deeper understanding of the connection between constrainedness and local minima. Informed by our analysis, we develop a novel variant of GBFS that outperforms the baseline.

In neural sequence decoding, we develop empirical models for the performance of beam search, the ubiquitous algorithm for decoding deep sequence models. First, we investigate the empirical problem of performance degradation in beam search. We present an explanatory model based on search discrepancies that generalizes previous observations on the behavior of beam search. Building on our analysis, we present two heuristic techniques that eliminate the problem. Next, we study goal-oriented sequence decoding and show that, similar to GBFS, we observe heavy-tailed behavior. We present a novel variant of goal-oriented beam search that exploits our insights and outperforms the baseline.

Our work shows the importance of empirical models in the study and development of heuristic search algorithms and demonstrates that empirical models developed for CSPs and SAT can be adapted to AI planning and neural sequence decoding.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor J. Christopher Beck, for his support and guidance during my graduate studies. Thank you for giving me the freedom to pursue my research interests and for providing me with many opportunities to participate in interesting projects, attend meetings, and present my work. I have learned a lot from you about research and academia and could not have asked for a better mentor.

I would like to thank my internal committee members, Professor Fahiem Bacchus and Professor Scott Sanner, for their guidance and commitment over the years. I would also like to thank the members of the final exam committee, Professor Alan Fern and Professor Sheila McIlraith, for their time and insightful feedback.

I would like to thank my collaborators at the University of Toronto, the Fujitsu Co-Creation Research Laboratory, and the Fujitsu Laboratories of America. In particular, I would like to thank Professor Mariano P. Consens for the joint work and the interesting discussions.

Thank you to my friends at the Toronto Intelligent Decision Engineering Laboratory (TIDEL) - Tony, Chiara, Margarita, Chang, Kyle, Tanya, Wen-Yang, Arik, Michael, Buser, Filip, Ranjith, Stefana, Luke, Arnoosh, Anton, Victor, Giovanni, Alex, Jason, Ryo, Louis, Jasper - for making my time at TIDEL so enjoyable.

Finally, thank you to my family. To my parents, brother, and sister, for the love and support over the years and the encouragement to pursue my goals. To my partner, Vince, for your love and for always being there for me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Heuristic search algorithms are widely used in both AI planning [14] and the decoding of sequences from deep neural networks [173]. In these tasks, the computational cost of finding an optimal solution is high and greedy search algorithms are often used to generate high-quality solutions. In AI planning, the most commonly used search algorithm is greedy best first search [36], a complete algorithm that relies on a domain-specific or domain-independent heuristic function to guide the search. In neural sequence decoding, beam search [12] is the ubiquitous search algorithm for decoding sequences from neural networks, using the network's predictions (in most cases, conditional probabilities).

In recent years, there has been a growing interest in developing a deeper understanding of the empirical performance of heuristic search algorithms. In satisficing AI planning, a recent line of work has attempted to highlight important factors that can impact the empirical performance of greedy best first search, including the operator cost ratio [31, 32, 195, 197], the correlation between the heuristic values and the true distance-to-goal [194], and the existence of uninformative heuristic regions [204, 197]. In neural sequence decoding using beam search, several works have attempted to characterize the empirical phenomenon of performance degradation for larger beams [105, 138, 183], the impact of (lack of) diversity of decoded sequences [181, 107], and the mismatch between the decoding objective and the evaluation metric [200, 147]. However, a principled empirical understanding of the search behavior of these heuristic search algorithms has yet to be developed.

In combinatorial search problems such as constraint satisfaction problems (CSP) and satisfiability (SAT), significant work has been done on empirical models for problem difficulty and solubility, including the phase transition [18, 164, 54], heavy-tailed behavior [64, 62], the existence of backbones [132] and backdoors [192], and search discrepancies [73, 184]. These models have been extensively used in the study of search behavior of combinatorial search algorithms and have played an important role in explaining observed algorithm performance (e.g., the existence of exceptionally hard problems [52, 165]) and in identifying challenging problems to support algorithm benchmarking [90, 55]. Furthermore, these models have informed the design of more efficient search algorithms and enhancements such as randomized restarts [66], algorithm portfolios [63], limited discrepancy search [73], etc.

Despite the popularity of empirical models in combinatorial search, the use of such models in AI planning and neural sequence decoding is limited. In particular, there is little work that attempts to adapt the ideas developed in the combinatorial search literature to the fields of AI planning and neural sequence decoding. The fundamental hypothesis of this dissertation is that empirical models can be

useful in the study of these fields, and that much of the existing work in combinatorial search can be adapted to the study of AI planning and neural sequence decoding.

After formally defining our thesis statement, we introduce the approach used in this dissertation. Then, we present the outline of the dissertation and summarize our main contributions.

Thesis Statement

The central thesis of this dissertation is that empirical models of the behavior of heuristic search algorithms can both explain observed algorithm performance and help design more efficient search algorithms. Inspired by previous work in combinatorial search, we aim to provide a better understanding of heuristic search in AI planning and neural sequence decoding and exploit this knowledge to advance the state-of-the-art in these areas.

1.1 Approach

This dissertation takes an *empirical approach* to the study of heuristic search algorithms. While theoretical analysis has characterized the worst-case or average-case complexity of such algorithms, it does not usually tell us how, or why, an algorithm is actually going to work in practice [92]. In this work we develop empirical models of the behavior of heuristic search algorithms and use them to explain the observed performance of these algorithms and to inform the design of new, more efficient algorithms. Our approach is illustrated in Figure 1.1 and consists of three main phases as follows.

Develop Analytical Framework. An analytical framework contains all the components needed to conduct the extensive experiments that will support the development of our empirical models. The analytical framework typically contains several core components:

- The problem instances that will be used in our empirical analysis. These instances can be taken from an existing set of problems (e.g., a benchmark dataset) or generated from a random problem generator. The problem instances can potentially be parameterized by some problem parameters (e.g., parameters that control the size of the problem) to support an empirical analysis of the impact of the parameters.

- The heuristic search algorithm that will be used to solve the problem. The search algorithm can also be parameterized by some configuration parameters that control the behavior of the search. In heuristic search, a typical configuration parameter is the heuristic function, however depending on the algorithm, additional parameters may be applicable.

- The monitored quantities that will be recorded for each solved instance. These quantities will be the basis of the statistical analysis in the next phase. In this work, we mainly consider the quantities that represent the difficulty of solving the problem or the quality of the obtained solution.

Perform Empirical Analysis of Search Behavior. Using the analytical framework, we conduct an extensive set of experiments. Each experiment typically consists of running an algorithm on a problem instance and recording the monitored quantities. Results are aggregated and statistically analyzed in accordance with the research topic and when applicable, connections between monitored quantities are

identified and established (e.g., correlation between some characteristic of problems and the associated search effort).

Explain Observed Performance and Inform Algorithm Design. Building on the results of the empirical analysis, we develop an empirical model that explains the observed performance. When applicable, we demonstrate how the empirical model can inform the design of new algorithms.

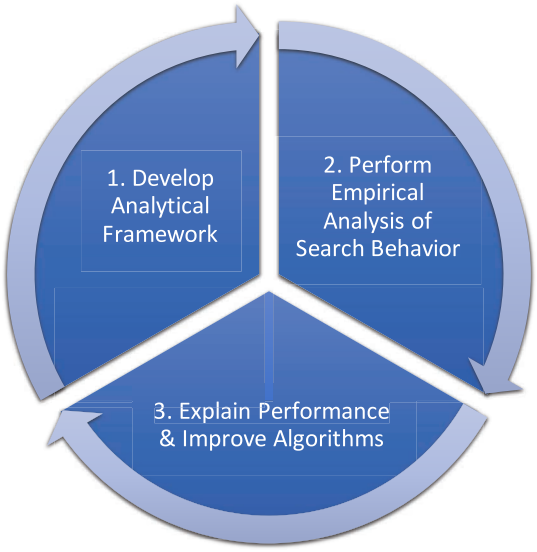


Figure 1.1: The research approach.

The above approach is used throughout the dissertation. In Chapters 3 and 4, we present an analytical framework for the analysis of GBFS behavior in domain-specific and domain-independent planning, respectively, and use it to explain empirical patterns of problem difficulty and, in Chapter 4, to inform a novel variant of GBFS. In Chapters 6 and 7, we present an analytical frameworks for the study of empirical beam search behavior in Maximum A Posteriori (MAP) and goal-oriented inference, respectively, in neural sequence models and propose algorithmic improvements.

1.2 Dissertation Outline

The dissertation is divided to two main parts, each correspond to a key AI task and the heuristic search algorithm commonly used to solve it, namely satisficing AI planning using greedy best first search and neural sequence decoding using beam search.

Part I deals with AI planning using greedy best first search (GBFS). In Chapter 2, we present the necessary background and notation on the task of automated planning and the relevant heuristic search algorithms. In Chapter 3, we conduct an empirical analysis of the phase transition phenomenon in domain-specific automated planning problems. Based on our analysis, we develop empirical models for the problem difficulty of solving such problems using GBFS. In Chapter 4, we study the heavy-tailed behavior in GBFS and present empirical models for the distribution of search effort in domain-independent automated planning. Informed by our analysis, we present a novel variant of GBFS that addresses the heavy-tailed behavior and outperforms the baseline.

Part II deals with neural sequence decoding using beam search. In Chapter 5, we present the necessary background and notation on the task of neural sequence decoding and on the beam search algorithm. In Chapter 6, we study the well-known problem of beam search performance degradation. We present an empirical exploratory model for the performance degradation and, based on our model, we develop two heuristics that mitigate the performance degradation. In Chapter 7, we study the problem of goal-oriented neural sequence decoding using beam search. Inspired by our results in Chapter 4, we present an empirical analysis of the heavy-tailed behavior in goal-oriented neural sequence decoding and a novel variant of beam search that outperforms standard beam search.

In Chapter 8, we summarize our contributions and discuss potential directions for future work.

1.3 Summary of Contributions

The main contributions of this dissertations are summarized as follows.

1.3.1 Phase Transition and Problem Difficulty in Domain-Specific Heuristic Search (Chapter 3)

Phase transitions in the solubility of random problem instances have proved useful in the study of problem difficulty for other classes of computational problems, notably SAT and CSP, and it has been shown that the hardest problems typically occur during this rapid transition. In this chapter, we perform the first empirical investigation of the phase transition phenomena in GBFS. We establish the existence of a rapid transition in the solubility of an abstract model of heuristic search problems and show that, for greedy best first search, the hardest instances are associated with the phase transition region. Then, we demonstrate that the behavior of our abstract model carries over to commonly used benchmark problems: the Pancake Problem, the Sliding Tiles Puzzle, TopSpin, and Grid Navigation. Furthermore, we investigate the interaction between the phase transition phenomenon and different factors that impact problem difficulty, including the quality of the heuristic, the re-expansion of nodes, and the operator cost ratio, and show that the effect of these factors depends on the constrainedness of problems. Building on the results of our empirical analysis, we hypothesize a relationship between the phase transition phenomenon and the existence and structure of local minima.

1.3.2 Heavy-tailed Behavior and Randomization in Satisficing Planning (Chapter 4)

In this chapter, we study the runtime distribution of satisficing planning in ensembles of random planning problems and in multiple runs of a randomized heuristic search on a single planning instance. Using common heuristic functions (such as FF) and six benchmark problem domains from the International Planning Competition (IPC), we find a heavy-tailed behavior, similar to that found in CSP and SAT. We investigate two notions of constrainedness, often used in the modeling of planning problems, and show that heavy-tailed behavior tends to appear in relatively relaxed problems, where the required effort is, on average, low. Drawing similarity from the analysis of inconsistent subtrees in CSPs, we show that there is a very strong exponential correlation between the depth of local minima and the associated search effort and that the distribution of local minima depth changes dramatically based on the constrainedness of problems. Our results support the hypothesis from Chapter 3, and provide a deeper understanding of

previous observations on the behavior of GBFS. Finally, we show that incorporating randomization in the search can help reduce the heavy-tailed behavior. Inspired by randomized restarts in CSP and SAT solving, we propose RR-GBFS, a variant of GBFS that employs randomized restarts, and show that it outperforms standard GBFS.

1.3.3 Empirical Analysis the Beam Search Performance Degradation in Neural Sequence Decoding (Chapter 6)

Beam search is the most popular inference algorithm for decoding neural sequence models. Unlike greedy search, beam search allows for non-greedy local decisions that can potentially lead to a sequence with a higher overall probability. However, work on a number of applications has found that the quality of the highest probability hypothesis found by beam search degrades with large beam widths. In this chapter, we perform an empirical study of the behavior of beam search across three sequence synthesis tasks. We find that increasing the beam width leads to sequences that are disproportionately based on early, very low probability tokens that are followed by a sequence of tokens with higher (conditional) probability. We show that, empirically, such sequences are more likely to have a lower evaluation score than lower probability sequences without this pattern. Using the notion of search discrepancies from heuristic search, we hypothesize that large discrepancies are the cause of the performance degradation. We show that this hypothesis generalizes the previous ones in machine translation and image captioning. To validate our hypothesis, we show that constraining beam search to avoid large discrepancies eliminates the performance degradation.

1.3.4 Randomized Restarting Beam Search in Goal-Oriented Neural Sequence Decoding (Chapter 7)

Recent work has demonstrated that neural sequence models can successfully solve combinatorial search problems such as program synthesis and routing problems. In these scenarios, beam search is used to produce a set of promising (high-likelihood) candidate sequences that are evaluated to determine if they satisfy the goal criteria. If none of the candidates satisfy the criteria, the beam search can be repeatedly restarted with a larger beam size until a satisfying solution is found. Following our work on GBFS, in this chapter we investigate whether heavy-tailed behavior can be observed in the search effort distribution of beam search in goal-oriented neural sequence decoding. We analyze four goal-oriented decoding tasks and find that the search effort of beam search exhibits fat- and heavy-tailed behavior. Inspired by the use of randomized restarts in GBFS, we propose a randomized restarting variant of beam search. We conduct extensive empirical evaluation, comparing different randomization techniques and restart policies, and show that the randomized restarting variant solves some of the hardest instances faster and outperforms the baseline.

Part I

Satisficing AI Planning using Greedy Best First Search

Chapter 2

Background: AI Planning and Greedy Best First Search

In Part I, we study the difficulty of satisficing planning problems solved using the greedy best first search (GBFS) algorithm. In this chapter we present relevant background and notation for the work presented in Chapter 3 and Chapter 4. Section 2.1 describes the heuristic search problem and discusses commonly used heuristic search algorithms. Section 2.2 describes the classical planning problem and how to cast it as a heuristic search problem. It also describes popular heuristics and search enhancements in classical planning using heuristic search. Finally, in Section 2.3 we describe previous work and results on problem difficulty in greedy best first search. The notation in this chapter is based on several sources [14, 81, 120].

2.1 Heuristic Search

Many approaches in artificial intelligence involve searching in large state spaces [170]. Many of the well-known algorithms for state space search, such as greedy best first search [36], A* [72], Weighted A* [142], are examples of a general family of uni-directional, expansion-based heuristic search algorithms [81]. These algorithms normally take a generative representation of the state space and a heuristic function that estimates the cost to reach the goal from a given state. In this section we formally define the different components in heuristic search problems and discuss popular algorithms for solving them.

Definition 1 (State Space [81]) *A state space is a tuple* $\langle S, s_I, S_G, \Gamma, c \rangle$

- S *is a a finite set of states,*
- $\Gamma : S \to 2^S$ *is the successor function that maps* X *to its power set,*
- $s_I \in S$ *is the initial state,*
- $S_G \subseteq S$ *is the set of goal states.*

In many cases, state spaces are also associated with a cost function $c : 2^S \to \mathbb{R}_0^+$ that defines the cost of each state transition. If there is no cost function associated with the state space, we assume a unit cost function, i.e., $c(s_i, s_j) = 1 \ \forall s_i, s_j \in S$.

State spaces contain state paths.

Definition 2 (State Path [81]) *A sequence of states* $\langle s_0, ..., s_n \rangle$ *is a state path from* s_0 *to* s_n *in* S *if* $s_i \in \Gamma(s_{i-1})$ *for* $i = 1, ..., n$.

A state path $\langle s_0, ..., s_n \rangle$ is a *solution path* if $s_0 = s_I$ and $s_n \in S_G$. The cost of a path is computed as follows:

$$cost(\langle s_0, ..., s_n \rangle) = \sum_{i=1}^{n-1} c(s_i, s_{i+1}).$$

A solution path is considered optimal if it is a minimal cost path among all solution paths.

Heuristic Function

Heuristic search algorithms use heuristic functions to guide the search in a state space for a path from the initial state to the goal state. A heuristic is a function $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$ with the intuition that $h(s)$ estimates the cost of a plan from s to a goal state [78]. The *perfect heuristic*, h^*, maps each state to the cost of an optimal plan from s to a goal state if such plan exists or to ∞ if there is no such plan.

2.1.1 Heuristic Search Algorithms

Best-First Search

Best-first search algorithms are state-space search algorithms that always expand the most promising state. Starting from the initial state, the successors of each expanded state are inserted into an *open list* that is sorted by some evaluation function $f(s)$. Once a state is expanded, it is taken out of the open list and moved to the *closed list*, a data structure that keeps track of states that were already expanded. If a goal state is expanded, the path from the initial state to the expanded goal state is computed and returned as solution. If the open list is empty before a goal state is expanded, we can determine that the state space was exhausted and there is no feasible solution. Different heuristic search algorithms can be defined that differ in their evaluation function $f(s)$. Typically, best first-search algorithms sort their open list based on evaluation function $f(s) = w_g \cdot g(s) + w_h \cdot h(s)$ that is a linear combination of $g(s)$, the lowest known cost of a path from the initial state to some state s, and $h(s)$, the estimated cost to reach a goal from state s. Specifically, greedy best first search, A*, and Weighted A* described below are all examples for such best first search algorithms. Algorithm 2.1 shows pseudo-code of a general best first search algorithm.

Algorithm 2.1 Best First Search

```
function BestFirstSearch(State space S, initial state s_I, goal states S_G, evaluation function f(.))
    CLOSED ← {}                                          ▷ Initialize the closed list
    OPEN ← {s_I}                                         ▷ Initialize the open list with the initial state
    parent(s_I) ← NONE
    g(s_I) ← 0
    while OPEN ≠ {} do
        s ← RemoveMin(OPEN, f(.))                        ▷ Select state to expand from the open list based on f(.)
        if s ∈ S_G then                                  ▷ If s is a goal state, return path to s
            return ReconstructPath(s)
        CLOSED ← CLOSED ∪ {s}
        for s' ∈ succ(s) do                              ▷ Iterate over the successors of the chosen state
            if s' ∈ OPEN then                            ▷ Update open state if cheaper path found
                if g(s') > g(s) + cost(s, s') then
                    parent(s') ← s
                    g(s') ← g(s) + cost(s, s')
            else if s' ∈ CLOSED then                     ▷ Re-open closed state if cheaper path found
                if g(s') > g(s) + cost(s, s') then
                    parent(s') ← s
                    g(s') ← g(s) + cost(s, s')
                    CLOSED ← CLOSED \ s'
                    OPEN ← OPEN ∪ {s'}
            else                                         ▷ Add successor s' to the open list
                parent(s') ← s
                g(s') ← g(s) + cost(s, s')
                OPEN ← OPEN ∪ {s'}
    return ∅                                             ▷ Exhausted space; Return no solution

function RemoveMin(open list OPEN, evaluation function f(.))
    return arg min_{s∈OPEN} f(s)

function ReconstructPath(state s)
    return ReconstructPathRecursive(s, [])

function ReconstructPathRecursive(state s, path [s_0, s_1, ..., s_k])
    if s = NONE then
        return [s_0, s_1, ..., s_k]
    return ReconstructPathRecursive(parent(s), [s, s_0, s_1, ..., s_k])
```

Greedy Best-First Search

Greedy best first search (GBFS) [36] is a variant of best first search algorithm where $w_g = 0$ and $w_h = 1$, i.e., the evaluation function $f(s) = h(s)$ is based solely on the heuristic function and the search algorithm

always expands the state with the lowest heuristic value in the open list. The algorithm is greedy since it only takes into account the (estimated) cost to reach the goal while ignoring the cost of the path from the initial state to the current state. It is common to choose not to re-open closed states or update states in the open list even if a shorter path to a state is found since state re-opening in GBFS does not guarantee any bound on solution quality. GBFS is the most commonly used algorithm for satisficing planning since it tends to find a solution faster than other best first search variants and is used by planners such as Fast Downwards [76] and LAMA [151]. In Chapter 3 and Chapter 4 we study different aspects of problem difficulty in GBFS.

A*

A* [72] is a best first search variant where $w_g = 1$ and $w_h = 1$, i.e., the evaluation function $f(s) = g(s)+h(s)$ is the sum of the cost to a state s and the estimated cost from s to a goal state. If multiple states on the open list have equal f values, states with lower h value will be expanded first. If A* is used with *admissible* heuristic, i.e., a heuristic the never overestimates the cost to the nearest goal, A* is guaranteed to return an optimal solution. Formally, admissibility is defined as follows.

Definition 3 (Admissible Heuristic) *A heuristic function $h(.)$ is considered admissible if for any state s*

$$h(s) \leq h^*(s)$$

where h^* is the perfect heuristic.

Another important property of heuristics is consistency.

Definition 4 (Consistent Heuristic) *A heuristic function $h(.)$ is considered consistent if for any state s and any successor $s' \in \Gamma(s)$*

$$h(s) \leq cost(s, s') + h(s').$$

If A* is used with a consistent and admissible heuristic, it is guaranteed not to re-open nodes [34].

Weighted A*

Weighted A* [142] is a best first search variant where $w_g = 1$ and $w_h > 1$, i.e., the evaluation function $f(s) = g(s) + w_h \cdot h(s)$ puts a higher weight on the estimated cost from s to a goal state than on the cost from the initial state to s. When used with an admissible heuristic the solution found by weighted A* is guaranteed to be no worse than $w_h \cdot C^*$ where C^* is the cost of the optimal solution.

Enforced Hill-Climbing

Enforced Hill-Climbing (EHC) [88] is a local search algorithm that is used in the FF planner. Similar to standard hill-climbing [155], the algorithm starts from an initial state and at each step the algorithm chooses a successor that has a strictly lower h value. However, unlike standard hill-climbing, if no successor has a lower h value, a breadth first search starting from the current state is invoked. The search returns the first state found with strictly lower h value or fails. If the breadth first search fails, the whole algorithm fails. Alternatively, when a goal state is reached, the algorithms stops.

2.2 Classical Planning

Classical planning consists of finding a sequence of actions from a given initial state to a goal state, assuming deterministic actions and complete information. A classical planning state space is formally defined as follows (note the similariy to Definition 1).

Definition 5 (Classical Planning State Space [14]) *A planning state space* $\langle S, s_I, S_G, A, f, c\rangle$ *consists of:*

- S *is a finite and non-empty set of states,*
- $s_I \in S$ *is the initial state,*
- $S_G \subseteq S$ *is the set of goal states,*
- $A(s) \subseteq A$ *is the set of actions applicable in each state* $s \in S$*,*
- $f(a, s)$ *is a state transition function for all* $s \in S$ *and* $a \in A(s)$*, and*
- $c(a, s)$ *is the cost of doing action* a *in state* s*.*

A *solution* of a planning state space is a sequence of actions $a_0, a_1, ..., a_n$ that generates a state trajectory $s_0, s_1 = f(a_0, s_0), ..., s_{n+1} = f(a_n, s_n)$ such that action a_i is applicable in s_i and s_{n+1} is a goal state, i.e., $a_i \in A(s_i)$ and $s_{n+1} \in S_G$. The cost of the solution is $\sum_{i=0}^{n} c(a_i, s_i)$. A solution is considered *optimal* if it has a minimum cost among all plans from the initial state to a goal state.

In large problems, it is impractical to explicitly enumerate the state space. Instead, we can use *factored* representations in which each state is a complete assignment of values to a set of variables. STRIPS [44], the most common factored representation for planning problems, is based on boolean variables called atoms (also known as fluents or facts) that state whether a proposition about the world is true in a given state.

Definition 6 (STRIPS Planning Problem [14]) *A STRIPS planning problem* $\mathcal{P} = \langle \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle$ *consists of:*

- A *is a set of atoms,*
- $\mathcal{O}$ *is a set of operators such that each* $o \in \mathcal{O}$ *is a tuple* $\langle Prec(o), Add(o), Del(o)\rangle$ *with* $Prec(o), Add(o), Del(o) \subseteq \mathcal{A}$*,*
- $\mathcal{I} \subseteq \mathcal{A}$ *encodes the initial state,*
- $\mathcal{G} \subseteq \mathcal{A}$ *encodes the goal state.*

A STRIPS problem $\mathcal{P} = \langle \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle$ implicitly defines a classical planning state space $\langle S, s_I, S_G, A, f, c\rangle$ as follows:

- The states in S are collections of atoms from, i.e., $s \subseteq \mathcal{A}$,
- The initial state $s_I \in S$ is $\mathcal{I}$,
- The goal states $s \in S_G$ are states such that $\mathcal{G} \subseteq s$,
- The actions of state s, $a \in A(s)$, are the operators $o \in \mathcal{O}$ such that $Prec(o) \subseteq s$,

- The transition function f maps a state s to the states $s' = (s \cup Add(a)) \setminus Del(a)$ for $a \in A(s)$,
- All action costs are one, i.e., $c(a, s) = 1 \ \forall s \in S, a \in A(s)$.

A more recent factored representation, the planning domain definition language (PDDL) [127, 45], has become the standard that is used in the international planning competition (IPC) [128]. The core of PDDL is the STRIPS formalism, however the languages extends beyond STRIPS to represent type structure, parameterization of actions, quantification, etc. [45].

2.2.1 Classical Planning as Heuristic Search

A planning problem (Definition 5) can be mapped to a state space (Definition 1) by defining a successor function based on the set of applicable actions $\Gamma(s) = \{s' \mid s' = f(a, s) \ \forall a \in A(s)\}$. Then, any heuristic search algorithm can be used to find a solution plan, i.e., a state path from the initial state to the goal state, given a heuristic function. Section 2.2.2 and Section 2.2.3 describe popular heuristics for classical planning and search enhancements that are commonly used in heuristic search-based classical planning, respectively.

2.2.2 Heuristics for Classical Planning

The Delete Relaxation Heuristic

Heuristics are often derived from solving a relaxation of the problem. The most commonly used relaxation in classical planning is the delete relaxation [14] where the delete effects of actions are ignored and actions can only add new facts to the set of true facts in a given state.

Definition 7 (Delete Relaxation) *Given a STRIPS planning problem $\mathcal{P} = \langle \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, the corresponding delete-relaxed STRIP planning problem is defined as $\mathcal{P}^+ = \langle \mathcal{A}, \mathcal{O}^+, \mathcal{I}, \mathcal{G} \rangle$ where the delete-relaxed set of operators $\mathcal{O}^+$ is defined as:*

$$\mathcal{O}^+ = \{\langle Prec(o), Add(o), \varnothing \rangle \mid o \in \mathcal{O}\}.$$

The cost of an *optimal* delete-relaxed plan from a state s to a goal is an admissible heuristic denoted as $h^+(s)$. Since the computation of this heuristic is NP-hard [15], it is common to use a polynomial-time approximation of h^+. Next, we describe several commonly used approximations of h^+.

The Additive Heuristic

The additive heuristic, h^{add}, is an inadmissible approximation of h^+ that is based on the costs of achieving each goal atom independently. The cost of achieving atom p from state s is denoted $g_s(p)$ and can be defined recursively as follows:

$$g_s(p) = \begin{cases} 0, & \text{if } p \in s \\ \min_{o \in O(p)}[1 + g_s(Prec(o))], & \text{otherwise,} \end{cases}$$

where $O(p)$ is the set of actions that add p, i.e., $p \in Add(o)$, and $g_s(Prec(o))$ is the estimated cost of the *set* of atoms in $Prec(o)$. The estimated cost for a set of atoms C in the additive heuristic is defined

based on the sum of the costs of the individual atoms,

$$g_s(C) = \sum_{r \in C} g_s(r).$$

The additive heuristic for a state s is then defined based on the estimated cost of achieving the set of atoms in $\mathcal{G}$ from state s:

$$h^{add}(s) = g_s(\mathcal{G}).$$

The additive heuristic assumes that subgoals (i.e., atoms in the goal) are *independent*, however this is not necessarily true as the achievement of some sub-goals can impact the difficulty of achieving other subgoals. As a result, the additive heuristic can overestimate the cost of getting to the goal and is therefore inadmissible.

The Max Heuristic

The max heuristic, h^{max}, is an admissible approximation of h^+ that is based on the maximum cost of an atom. It is defined in a similar manner to the additive heuristic, however the estimated cost for a set of atoms C is defined based on the maximal cost among the costs of achieving the individual atoms,

$$g_s(C) = \max_{r \in C} g_s(r).$$

Unlike the additive heuristic, the max heuristic is admissible since the true cost of achieving a set of atoms cannot be lower than the individual cost associated with each of the individual atoms.

The FF Heuristic

The FF heuristic [88], h^{FF}, is an inadmissible approximation of h^+ based on the cost of a suboptimal relaxed plan. To compute the heuristic for a state s, we first need to construct the relaxed planning graph, a directed layered graph that contains two types of nodes: fact nodes and action nodes. The layers alternate and each pair of fact and action layers make up a time step. In the first time step, the fact layer corresponds to the facts in state s and the action layer corresponds to the applicable actions in that state. In each subsequent time step i, the fact layer consists of all the facts that were added by the actions of the previous time step along with the facts that were already true in the previous time step, ignoring any delete effects.

Once the relaxed planning graph is constructed, a relaxed plan Π^+ is extracted from the relaxed planning graph. Starting from the goal facts, actions that add each of these facts are added to the plan, then actions are added until all preconditions of actions in the relaxed plan are supported by some action or are part of the state s. The FF heuristic then estimates the distance to the goal based on the cost of the plan Π^+. Note that unlike h^{add} and h^{max}, the FF heuristic computes a feasible, however not optimal, plan for the delete-relaxed planning problem.

Landmark-based Heuristics

Landmarks are plan features that must appear in any valid solution plan [89, 150]. Fluent landmarks are formulas over the set of fluents that must be satisfied by some state in any solution plan. Similarly, action landmarks are formulas over the set of operators that must be satisfied by any solution plan.

The problem of determining if a formula is a landmark is PSPACE-complete [89]. Therefore, practical methods for finding landmarks are either incomplete (i.e., they may fail to find some landmarks) or unsound (i.e., they may falsely declare a formula to be a landmark) [150].

The first landmark-based heuristic was the landmark count heuristic [150], an inadmissible heuristic that estimates the number of landmarks that still need to be satisfied. Given a state, s, and a plan from the initial state to s, Π, the estimate of the number of landmarks that still need to be satisfied is computed as follows.

$$h^{LM}(s, \pi) = (L \setminus \mathit{Reached}(s, \Pi)) \cup \mathit{ReqAgain}(s, \Pi)$$

where L is the set of all discovered landmarks, $\mathit{Reached}(s, \Pi)$ is the set of reached landmarks, and $\mathit{ReqAgain}(s, \Pi) \subseteq \mathit{Reached}(s, \Pi))$ is the set of reached landmarks that are required again.

Following the landmark count heuristic, several works have demonstrated that landmarks can also be used to compute admissible estimates for optimal planning, including the admissible landmark heuristic [100] and the landmark cut heuristic [78].

Abstraction Heuristics

Abstraction heuristics map each state s to an abstract state $\alpha(s)$ using a homomorphism function α [78]. The heuristic value $h^{\alpha}(s)$ is then the distance from $\alpha(s)$ to the nearest abstract goal state. Examples of abstraction heuristics include pattern databases [30, 38] and merge-and-shrink abstractions [80].

2.2.3 Notable Search Enhancements

Helpful Actions

Helpful actions [88] is a heuristic pruning techniques that allows the search to focus only on actions that are likely to lead to a solution and to reduce the number of generated successors for each state. Some heuristics, in addition to computing an estimate of the cost to the goal, also compute a subset of actions that are likely to be in the solution. For example, the FF heuristic returns, in addition to a cost estimate based on the relaxed solution, the subset of actions from that solution that are applicable to the current state. The helpful actions can be used to prune successors that are not generated from helpful actions or, alternatively, to prioritize states that were generated by helpful actions. In the Fast Downward planning system [76], Helmert proposed a technique called *preferred operators* that consists of alternating between two open lists, one containing all successors of expanded states and one containing only successors generated by helpful actions.

Deferred Heuristic Evaluation

In standard GBFS, when the search expands a state s, it needs to compute the heuristic evaluation for all of the successor states of s before adding them to the open list. If the heuristic evaluation is computationally expensive and there are many successors, this computation can make up significant part of the total search effort. For example, Bonet and Geffner [13] showed that the computation of the heuristic can take up to 85% of the computational effort of the HSP planner.

In the Fast Downward planning system [76], Helmert proposed to use *deferred heuristic evaluation*, a technique that delays the heuristic evaluation of a state until it is expanded. Each time a state s is expanded, we compute its heuristic evaluation and the successors of s are added to the open list with the

heuristic evaluation of state s. Deferred heuristic evaluation can lead to a significant speed-up when many more nodes are generated than expanded, e.g., in problems with a large branching factor [76]. Deferred heuristic evaluation was empirically shown to trade time for quality: it tends to find plans faster, however these plans tend to have a lower quality compared to plans found using standard heuristic evaluation [149].

Multi-Heuristic Best-First Search

Another enhancement that was introduced in the Fast Downward planning system [76] is the use of multi-heuristic best-first search. While there are different ways of combining more than one heuristic, e.g., by taking the maximum of the individual heuristic values, multi-heuristic bests-first search instead keeps a separate open list for each heuristic in which nodes are sorted according to the respective heuristic evaluations. The search alternates among the open lists. When expanding a state from one open list, the successors are added into each of the open lists with a heuristic evaluation based on the respective heuristic.

Non-Greedy Exploration

In satisficing planning, several works have suggested incorporating non-greedy exploration, in which the search allocates limited time to expand nodes with non-minimal h-values. We briefly describe notable recent approaches for non-greedy exploration.

Diverse Best First Search (DBFS). Imai and Kishimoto [95] proposed DBFS, a best first search variant that incorporates random exploration in two steps: first, a state is randomly selected from the open list according to a distribution that favors states with lower g value and lower h value; then, an expansion-limited GBFS is invoked from the selected state.

ϵ-GBFS. Valenzano et al. [176] proposed *ϵ-GBFS*, a simple variant of GBFS that expands a node selected uniformly at random from the open list with probability ϵ and the minimal h-value node with probability $1-\epsilon$. Empirical analysis showed that for ϵ values between 0.05 to 0.5, the number of domains for which the coverage increased was significantly higher than the number of domains for which the coverage decreased.

Type-GBFS. A type system in heuristic search is a partitioning of all the states in a state space to n disjoint sets [113]. Xie et al. [205] proposed *Type-GBFS*, a variant of GBFS that utilizes type systems of heuristic search to perform exploration. In addition to the standard open list that orders the states based on their heuristic value, Type-GBFS maintains an additional queue that performs type-based exploration using a two level type bucket structure: first it picks a bucket b uniformly from all the buckets, then it picks a state n uniformly from all the states in b [205]. The search alternates between expanding states from the standard open list and the type-based exploration queue. Empirical analysis showed that Type-GBFS significantly outperforms standard GBFS, solving almost 200 more problems out of 2112.

GBFS with Local Exploration. Xie et al. [203] proposed GBFS with local exploration (GBFS-LE) where a random walk or a local GBFS is invoked when the global GBFS has failed to improve its minimum heuristic value (i.e., the lowest heuristic value encountered so far) for a fixed number of expansions. In a

later work, Lipovetzky and Geffner [122] considered GBFS-W, a variant of GBFS-LE where the local GBFS is replaced with a width-based search [121] that explores the state space based on the *novelty* of states. In particular, the width-based algorithm IW(k) is a breadth-first search where a newly generated state s is pruned if its novelty is greater than k, where the novelty of s is the size of the smallest tuple of atoms such that s is the first state in the search that makes all these atoms true [122].

2.3 Problem Difficulty in GBFS

In the recent decade, several works have addressed the problem of developing a deeper theoretical and empirical understanding of GBFS. In this section, we review notable works on the search behavior of GBFS and different factors that impact problem difficulty for GBFS.

2.3.1 Uninformative Heuristic Regions

One of the main factors that has negative impact on the problem difficulty for GBFS is the existence of uninformative heuristic regions (UHR), which include plateaus and local minima [204].

Wilt and Ruml [197] defined local minimum as a set of states, S_{LM}, such that for each state $s \in S_{LM}$ all paths from s to a goal state include at least one state s' such that $h(s') > h(s)$ and S_{LM} is maximally connected.[1] A heuristic plateau is a maximal connected region of nodes such that all nodes in the region have the same heuristic value [197].

Wilt and Ruml [197] calculated the size of of every local minimum in an entire search space by searching backwards from the goal states, expanding nodes in increasing h order. Nodes with h value that is less than the highest h value encountered seen so far are part of a local minimum.

2.3.2 Operator Cost Ratio

Recent work has highlighted the negative impact of large operator cost ratio in cost-based search and proposed size-based search as an alternative to cost-based search [31, 32, 195, 197]. We start by defining cost-based search and operator cost ratio and then describe the main results.

Definition 8 (Cost-based search; Cushing et al. [32]) *A best first search in which $g(x) = g_c(x)$, the cost to reach state x, and $h(x) = h_c(x)$, an estimation of the cost of the cheapest path from state x to a goal state.*

Definition 9 (Operator cost ratio; Wilt and Ruml [195]) *The ratio of the largest action cost to the smallest action cost in the state space,*

$$\frac{\max_{s \in S, a \in A(s)} c(a, s)}{\min_{s \in S, a \in A(s)} c(a, s)}.$$

Wilt and Ruml [195] showed, empirically and theoretically, that a larger operator ratio can have a negative effect on the search effort of various best-first search algorithms, including GBFS. Their analysis showed that cost-based versions of sliding puzzle and the pancake problem become intractable as the operator cost ratio is increased. The cost-based grid navigation problem, however, does not suffer from a

[1] Wilt and Ruml [197] note that in directed state spaces, these definitions become more complicated.

significant increase in search effort and the authors attribute it to size-bounded local minima and the large number of duplicated states.[2]

To mitigate the negative effect of a large operator cost ratio, a number of authors suggest using size-based search [32, 195, 197].

Definition 10 (Size-based search; Cushing et al. [32]) *A best first search in which $g(x) = g_d(x)$, the distance (i.e., the number of actions) to reach state x, and $h(x) = h_d(x)$, an estimation of distance of the shortest path from state x to a goal state. Also called distance-based search.*

Wilt and Ruml [197] showed that size-based search tends to have smaller local minima compared to cost-based search.

2.3.3 Goal Distance Rank Correlation

Wilt and Ruml [196, 198, 194] performed an extensive investigation of the behavior of suboptimal heuristic search algorithms, namely GBFS and Weighted A*. They empirically show that although suboptimal search algorithms are expected to be faster than algorithms with stronger guarantees, they can often underperform. Specifically, they demonstrated how GBFS can perform worse than Weighted A* and sometimes even worse than A*. Furthermore, they show that the well-established guidelines for designing heuristic for A* can lead to poor performance in GBFS.

Based on an empirical analysis of the behavior of GBFS, Wilt and Ruml [194] make the following observations on the characteristics of effective heuristics for GBFS.

Observation 1 *All else being equal, greedy best-first search tends to work well when it is possible to reach the goal from every node via a path where h monotonically decreases along the path.*[3]

Observation 2 *All else being equal, nodes with h=0 should be connected to goal nodes via paths that only contain h=0 nodes.*[4]

Observation 3 *All else being equal, greedy best-first search tends to work well when the difference between the minimum h value of the nodes in a local minimum and the minimum h that will allow the search to escape from the local minimum and reach a goal is low.*

Based on their analysis, Wilt and Ruml [194] propose to measure and compare the effectiveness of heuristics using the *global distance rank correlation (GDRC)* defined by the Kendall's τ correlation between the heuristic values $h(s)$ and the real distance to nearest goal $d^*(s)$. Inspired by Haslum et al. [75], they present an automatic procedure to construct effective abstraction-based heuristic by searching in the space of abstractions, using GDRC as a metric for assessing the quality of a heuristic. Note that a similar notion to GDRC, called fitness-distance correlation, has been studied in the metaheuristics literature [9].

[2] Fan et al. [43] showed that in some cases diverse operator costs can lead to better performance, however their analysis is only concerned with the Dijkstra's algorithm and does not deal with GBFS or, more generally, search with heuristics.

[3] More precisely, the authors actually refer to the case where h is monotonically non-increasing.

[4] Wilt and Ruml [194] state that Observation 2 is an important specific case of Observation 1.

2.3.4 Theoretical Results on Problem Difficulty in GBFS

A recent line of work [81, 83, 82] has addressed the development of a theoretical foundation for the analysis of greedy best first search. In particular, these works characterize the set of states that are guaranteed to be expanded by GBFS, the set of states that are guaranteed not to be expanded by GBFS, and the set of states that can potentially be expanded by GBFS.

In order to characterize the behavior of GBFS, Heusner et al. [81] use the notion of high-water mark [197],

$$hwm(s) = \begin{cases} min_{p \in P(s)}(max_{s' \in p} h(s')) & \text{if P(s)} \neq \varnothing \\ \infty & \text{otherwise,} \end{cases}$$

where $P(s)$ is the set of all paths from s to the goal. The high-water mark of a set of states $S' \in S$ is defined as

$$hwm(S') = min_{s \in S'} hwm(s).$$

Intuitively, the high-water mark of a state s measures how high the heuristic values of expanded states must climb before a solution can be found starting from state s [83]. For a set of states S', the high-water mark, $hwm(S')$, is based on the minimum high-water mark of the states in S'.

Using high-water mark, Heusner et al. [83] define *progress states* as states $s \in S$ such that $hwm(s) > hwm(\Gamma(s))$. Then, they show that every GBFS run can be understood as a sequence of search episodes (called *high-water mark benches*), where each episode begins when a progress state is expanded and ends when the next progress state is expanded. Using the notion of high-water mark benches, Heusner et al. [81] shows how the state space can be partitioned into a set of states that are guaranteed not to be expanded by any GBFS run (regardless of the tie-breaking strategy) and a set of states that are expanded by at least one tie-breaking strategy [81]. Furthermore, they present criteria for states that are guaranteed to be expanded. In a later work, Heusner et al. [82] show that it is NP-complete to compute lower and upper bounds on the number of states expanded by GBFS, given an explicit representation of the state space. However they also show that for undirected spaces, best-case and worst-case tie-breaking of GBFS can be analyzed in polynomial time.

In Chapter 3 and Chapter 4 we study problem difficulty in AI planning building on the work described in Section 2.3 and two existing notions from combinatorial problem difficulty, the phase transition and heavy-tailed behavior, which will be reviewed in the respective chapters.

Chapter 3

Phase Transition and Problem Difficulty in Domain-Specific Heuristic Search

3.1 Introduction

A recent line of research in heuristic search aims to develop of an understanding of empirical problem difficulty. Several factors affecting search effort have been identified including the ratio of operator costs [195, 197, 31, 32], and the existence of uninformative heuristic regions [204].

The phase transition in problem solubility has been a central tool in the study of problem difficulty for computational problems. In seminal work, Cheeseman, Kanefsky and Taylor [18] empirically showed that several NP-complete problems exhibit an abrupt *phase transition* from underconstrained to overconstrained problems as a problem-generation control parameter is varied, changing the probability of a solution from nearly zero to nearly one. They discovered that the hardest problem instances occur during this abrupt change.

Subsequently, the phase transition was extensively studied in problems including SAT [131, 29], CSP [164, 145], quantified boolean formula [56], and classical planning [16, 152]. Interestingly, despite Cheeseman et al.'s conjecture that the phase transition was relevant to a number of AI problems, it does not appear to have been studied for heuristic search problems.

In this work, we introduce the tool of phase transition to heuristic search using an abstract model of a heuristic search problem that is based on a random graph representation of the state space. We demonstrate an abrupt transition in solubility as a parameter controlling the density of the transitions in the state space is varied and observe the accompanying easy-hard-easy pattern of problem difficulty across the transition region. Building on these results, we make the following further contributions:

- Exploiting our random graph model, we provide analytical bounds on the "mushy region" between the fully soluble and fully insoluble problems.

- We demonstrate how to transfer the abstract graph model to existing heuristic search benchmark problems, allowing the generation of versions of each problem across the phase transition and

demonstrating both the phase transition and the easy-hard-easy pattern on the five standard heuristic search benchmarks.

- We study the behavior of systematically stronger heuristics across the phase transition region and show that the reduction in search effort for strong heuristics is orders of magnitude *smaller* for the hard soluble instances at the phase transition than in the underconstrained regions.
- We show that the number of node re-expansions peaks in the phase transition and declines outside.
- We demonstrate that the effect of large operator cost ratio on the search effort is most significant in the phase transition region and diminishes outside.
- We show that *exceptionally hard problems* [52, 165] appear in the relaxed region of the phase transition where the median effort is relatively low.

The work in this chapter is based on the publications [26, 25].

3.1.1 Organization

In Section 3.2 we present the background on phase transitions and discuss related work. In Section 3.3 we describe the analytical framework we use to study the phase transition in heuristic search. In Section 3.4 we present an analysis of the phase transition using an abstract model and in Section 3.5 we demonstrate that the behavior observed for the abstract model carries over to real benchmark problems. In Section 3.6 and Section 3.7 we analyze the interaction of the phase transition with node re-expansions and operator cost ratio, two factors that were shown to have negative impact on the search effort. Finally, in Section 3.8 we discuss the implications of the analysis, as well as potential future work. In Section 3.9 we summarize the chapter.

3.2 The Phase Transition

For many NP-complete problems, we can define a *control parameter* for which a critical interval of values separates two regions: one that is underconstrained with high density of solutions and one that is overconstrained with low likelihood that a solution exists [18]. Empirically, the hardest problems occur over this critical interval. Mitchell, Selman and Levesque [131] identified a phase transition for critical values of the clause-to-variable ratio of 3-SAT problems, with search effort peaking at a ratio of approximately 4.27. Smith and Dyer [164] investigated phase transitions in CSPs and showed a phase transition in problem solubility for critical values of constraint tightness. The hardest problems occurred during this rapid transition with the median search effort peaking at the point in which 50% of the problems are soluble. Many other works through the late 1990s confirmed these results and extended them to other classes of problems [90, 55, 207, 129].

For PSPACE-complete problems, Gent and Walsh [56] investigated the phase transition phenomenon on quantified boolean formulae. They witnessed a clear phase transition in solubility and an easy-hard-easy pattern associated with a critical value of constrainedness. They therefore conjectured that similar phase transition behavior will occur in other PSPACE problems. In planning, Rintanen [152] found a transition in solubility as the operators-to-variables ratio is varied. However, the effort peak was located

earlier than expected and the analytical derivation of tight upper and lower bounds for this transition remained an open problem.

The phase transition in heuristic search does not appear to have been investigated. While a reference to heuristic search occurs in an early work [94], the work actually concerns tree search and is therefore more relevant to SAT and CSP rather than a best-first search.

3.3 Analytical Framework

To investigate the phase transition phenomenon in heuristic search, we develop an analytical framework that allows us to generate random heuristic search problems with specific levels of constrainedness. Then, we study the the impact of constrainedness on problem solubility and difficulty.

3.3.1 Abstract Model

In search problems, the problem space is often represented as a graph [142] with states as vertices and the transitions as edges. Given a finite state space S of cardinality n: $S = \{s_1, s_2, ..., s_n\}, |S| = n$, and a successor function $\Gamma : S \to 2^S$ (see Definition 1 in Section 2.1), we can use the following graph representation: $G\langle V, E\rangle$, $V = \{v_i : s_i \in S\}$, $E : \{(v_i, v_j) : s_i, s_j \in S, s_j \in \Gamma(s_i)\}$.

Random Problem Space.

With such a representation, generating a random problem requires generating a random graph with characteristics similar to the search space of heuristic search problems [42, 99, 6, 187].

Common heuristic search problems often exhibit symmetric transitions (or have a similar distribution of in- and out-degrees) along with other properties that can be well-modeled by random graphs. For example, in the Sliding Puzzle, the Pancake Problem, and the Rubik's Cube, the transition function is symmetric ($\forall i, j : s_i \in \Gamma(s_j) \iff s_j \in \Gamma(s_i)$), and most of the states have the same degree, mapping to an undirected model with nearly constant degree distribution. Grid Navigation and Vacuum World, in which a cell is blocked with probability p, as well as other similar problems, have a degree distribution centered around an expected degree that varies based on p. Map navigation problems, e.g. the Arad-Bucharest example [155], are often symmetric and have a different degree distribution for every map, with no specific structure.

As our focus is not on a specific class of problem, we use the random digraph model [99] as a general model. While not committing to symmetry, it maintains an even distribution of in- and out-degrees and a Poisson degree distribution centered around a certain degree. We expect this to be a good general model for random heuristic search problems and, in Section 3.5, we examine its applicability to real heuristic search problems. When investigating a specific class of problems, using a random graph model that better matches the nature of the class may yield more precise results.

Random Problem Instances.

Our model for generating random problem instances is based on a random directed transition graph, a randomly chosen initial state, and a randomly chosen goal state. An important technical consideration in problem generation is the avoidance of so-called "flawed" models which include many trivially insoluble instances [61]. A naive instance generator can easily generate instances for which either the initial state or

goal state has no out- or in-transitions, respectively. While such instances do not form an asymptotically dominant proportion of the instance space [1], they can nonetheless be easily filtered. Therefore, we propose the following problem generation model.

Model 1 *Let $n \in \mathbb{Z}^+$ be the number of states in the problem space $S = \{s_1, s_2, ..., s_n\}$ and $p \in [0, 1]$ be the connectivity density of the problem space. The class $Q_{n,p}$ consists of all problem instances $\langle T, S_i, S_g \rangle$ such that:*

1. *T is a random transition graph drawn from $D_{n,p}$, the probability space of all random digraphs [99].*
2. *$S_i \in S$ is a randomly chosen initial state such that $\exists k \neq i : (S_i, S_k) \in T$.*
3. *$S_g \in S$ is a randomly chosen goal state such that $S_g \neq S_i$ and $\exists k \neq g : (S_k, S_g) \in T$.*

As this model requires a minimal number of edges to provide sufficient variation, we arbitrarily consider instances that have more than 1000 edges.

The Control Parameter.

In our investigation, we use the following control parameter:[1]

$$\gamma := \frac{\text{Expected number of edges in the transition graph}}{\text{Number of states}}$$

As the numerator is in the range of $[0,\ n(n-1)]$ (no self-loops), then $\gamma \in [0, n-1]$. When no edges exists ($\gamma = 0$), no path between the initial and the goal states exists. When all edges exist ($\gamma = n - 1$), every potential path between the initial and goal states is a feasible solution. As the expected number of edges (and γ) increases, problems become less constrained and the expected number of solutions increases.

The Region of the Phase Transition.

While the phase transition is asymptotically instantaneous, in finite problems the region in which the probability that a problem is soluble changes from nearly zero to nearly one is referred to as the *mushy region* [164]. The point in which this probability is 0.5 is referred to as the *crossover point* [29]. Here, we define the mushy region and the crossover point based on the observed proportion of soluble problems as follows.

Definition 11 (Mushy region) *The range of γ in which the observed portion of soluble problems is between 0.1% and 99.9%.*

Definition 12 (Crossover point) *The γ value in which the observed portion of soluble problems is 50%.*

The Heuristic Function.

To investigate the effect of γ on search effort and whether an easy-hard-easy pattern emerges in the phase transition region, we need to use a heuristic that is general enough to apply to our abstract problems

[1] For digraphs, γ is equivalent to c, often used in the research on phase transitions in random graphs (e.g., Karp [99]).

but that also embodies useful search guidance. We therefore consider the following set of heuristics $H = \{h_0, h_1, h_2, ...\}$, based on $c(x)$, the real cost from x to goal:

$$h_i(x) = \begin{cases} c(x), & \text{if } c(x) < i \\ i, & \text{otherwise} \end{cases}$$

H is a set of admissible, increasingly informed heuristics. h_0 is the completely uninformed zero heuristic, h_1 only incorporates information on the goal state, while h_i incorporates information on states up to $i-1$ steps away from the goal. We note that $\forall i < j : h_j$ dominates h_i, and since all actions have the same cost, h_j will have a higher rank correlation with the true distance-to-go [198].

These heuristic functions are admissible and consistent and we expect they will demonstrate an easy-hard-easy pattern as γ varies: in underconstrained problems, the goal state or one of its close neighbors will be added to the open list quickly, and expanded shortly thereafter; in overconstrained problems, regardless of the heuristic, it should be easy to exhaust the relatively small part of the problem space that is accessible from the initial state; in the mushy region neither of these easy cases will apply, requiring more effort to find a path to the goal or to prove that none exists.

Analytical Bounds on the Mushy Region

In the previous work on the phase transition [164], the mushy region has been defined, as above, using arbitrary thresholds for the observed percentage of soluble instances. An immediate benefit of our random graph model is that we can derive bounds on the mushy region using the theory of random graphs.

Two well-studied phenomena in random graphs are the emergence of a giant component and the connectivity threshold [46]. Let n be the number of vertices in the digraph, $p = \frac{\gamma}{n}$ be the connectivity density and ω be a non-decreasing function such that $\omega(n) \to \infty$ as $n \to \infty$. For $\gamma < 1$, all components of $D_{n,p}$ are either single vertices or components smaller than ω for any ω, and so we expect the solubility in this region to be asymptotically zero. For $\gamma > \frac{\ln n + \omega}{n}$, the graph is fully-connected, and we expect the solubility in this region to be asymptotically one. Therefore, we can use $\gamma \approx 1$ and $\gamma \approx \frac{\ln n}{n}$ as principled bounds on the mushy region.

3.3.2 Analytical Model for Benchmark Problems

To perform an analysis of the phase transition on existing heuristic search benchmark problems, we adapt the framework in Section 3.3.1 to structured state spaces. We start by describing the benchmarks problems we consider in our analysis. Then, we describe our approaches for generating random problem instances for existing heuristic search problems.

Benchmarks

The Pancake Problem. The pancake problem [37, 48, 84] consists of sorting a sequence of objects (pancakes) through a minimal number of prefix reversals (flips). Assuming a set of n pancakes of different sizes stacked on a plate in an arbitrary order, the goal is to sort the pancakes such that that largest pancake is at the bottom and the smallest one is on top. The available actions are k-flips, $k \in \{2, ..., n\}$, that correspond to flipping k pancakes at the top of the stack. When solving the problem, we use the

gap heuristic [77], a landmark heuristic [78] based on the number of gaps in the stack:

$$h^{gap}(s) := |\{i \mid i \in \{1, ..., n\}, |s_i - s_{i+1}| > 1\}|, \tag{3.1}$$

where s_i represent the size rank of pancake i.

Sliding Puzzle. The $n \times n$ sliding puzzle has $n^2 - 1$ sliding tiles and one blank. The goal is to get from some configuration of tiles to a target configuration with the actions correspond to sliding horizontally or vertically adjacent tiles into the blank position. The heuristic function is based on the sum of Manhattan Distances between each tile and its target position.

Top Spin. The top spin puzzle includes a ring of n disks that can be shifted cyclically and a turnstile that can reverse the order of r disks. The goal is to get from an initial permutation of the n disks to a target permutation by iteratively reversing a contiguous subsequence of size r. We use a heuristic based on a pattern database (PDB) [30, 38] that stores a lower bound on the cost to goal for each abstract state (pattern) normally defined by only considering a part of the state while ignoring the rest.

Grid Navigation. Consider an $n \times m$ grid of passable cells and blocked cells (obstacles). There are four potential moves in each state (up, down, left, right), subject to obstacles and boundary constraints. The goal is to move from the initial position to the target position, traversing only passable cells. When solving the problem, we use the Manhattan Distance heuristic:

$$h(s) = |s_x - g_x| + |s_y - g_y| \tag{3.2}$$

where s_x (s_y) corresponds to the row (column) of the user in state s and g_x (g_y) corresponds to the row (column) of the target position.

Random Instance Generator

We propose a model for generating restricted or relaxed instances of an existing heuristic search problem at varying constrainedness level. We start by considering the transition graph induced by the problem's state space and create increasingly restricted or increasingly relaxed variants of this graph by removing or adding edges. These variants are proper subgraphs or supergraphs of the original transition graph and the instances near the original constrainedness level should have an almost identical connectivity structure.

Definition 13 (Observed connectivity density) *Let $G\langle V, E\rangle$ be an arbitrary transition graph. We define the observed connectivity density of this graph $\mathcal{P}(G) = \frac{|E|}{|V|\cdot(|V|-1)}$.*

Definition 14 (Restricted instance) *Let $G\langle V, E\rangle$ be an arbitrary transition graph. $\hat{G}\langle V, \hat{E}\rangle$ is considered a restricted instance of G if $\mathcal{P}(\hat{G}) < \mathcal{P}(G)$ and $\hat{E} \subseteq E$.*

Definition 15 (Relaxed instance) *Let $G\langle V, E\rangle$ be an arbitrary transition graph. $\hat{G}\langle V, \hat{E}\rangle$ is considered a relaxed version of G if $\mathcal{P}(\hat{G}) > \mathcal{P}(G)$ and $\hat{E} \supseteq E$.*

Our random model generates restricted or relaxed instances of the original problem with the required connectivity density. Consistent with the abstract model, we eliminate trivially insoluble instances.

Model 2 *Given an existing problem's transition graph $G\langle V, E\rangle$ and the required connectivity density p, the class $R_{G,p}$ consists of all problem instances $\langle T, S_i, S_g\rangle$ such that:*

1. *T, the transition graph, is a restricted instance of G if $p < \mathcal{P}(G)$, or a relaxed instance otherwise. $\mathcal{P}(T) = p$.*

2. *$S_i \in S$, a randomly chosen initial state, $\exists k : (S_i, S_k) \in T$*

3. *$S_g \in S$, a randomly chosen goal state such that $S_g \neq S_i$ and $\exists k : (S_k, S_g) \in T$*

A Domain-Specific Model

While Model 2 provides a domain-independent way to generate relaxed and restricted problems, for some domains one can find a domain-specific parameter that controls the constrainedness of the problem. For example, in the Grid Navigation domain, the probability of a blocked cell q clearly controls the constrainedness of the generated instances: a high q will produce more constrained state spaces with lower solution density while a low q produces the opposite.[2]

We therefore define a domain-specific model for Grid Navigation and use it in our analysis.

Model 3 (q-Constrained Grid Navigation Problems) *Given grid dimensions $n \times m$, we denote by $G_{nm}\langle V, E\rangle$ the transition graph of an $n \times m$ grid navigation problem and by q the probability of a blocked cell. We define the class $R_{n,m,q}$ that consists of all problem instances $\langle T, S_i, S_g\rangle$ such that:*

1. *T, the transition graph, is an instance of G_{mn} in which each cell is blocked with probability q.*

2. *$S_i \in S$, a randomly chosen initial state, $\exists k : (S_i, S_k) \in T$.*

3. *$S_g \in S$, a randomly chosen goal state such that $S_g \neq S_i$ and $\exists k : (S_k, S_g) \in T$.*

Heuristic Functions

In our analysis, we use domain-specific heuristic functions to solve the benchmark problems. It should be noted that these heuristics remain admissible for the restricted instances but not for the relaxed instances. However, admissibility is not required for our analysis.

3.3.3 Search Algorithm

Our study is focused on greedy best-first search (GBFS) [36]. Consistent with all previous phase transition work of which we are aware, this algorithm aims to find a feasible solution. In Section 3.4 and Section 3.5, we use unicost actions and configure the search algorithm not to re-open closed nodes. Then, in Section 3.6 and Section 3.7, we study the impact of node-reopening and cost-based search, respectively.

3.4 The Phase Transition in the Abstract Model

In this section, we present an empirical analysis of the phase transition phenomenon in our abstract model. Using Model 1, we generate random problems in $Q_{n,p}$. We carried out a series of experiments for $n = \{10000,\ 100000,\ 1000000\}$, and for 35 γ values in $[0, n-1]$, non-uniformly distributed, with higher density within the mushy region boundaries. For each value of n and p, we generated 1000 random instances. For each instance, we record whether or not a solution was found and the number of

[2] With $q = 0$ the state space is still highly constrained, as there are only four actions per state. It is possible to relax the domain further by allowing more actions (e.g., diagonal moves, jumps, etc.).

nodes expanded in order to find a solution or prove that no solution exists. We present results only for $n = 100000$ as the other plots show the same behavior.

3.4.1 The Phase Transition in Solubility

Figure 3.1 shows the probability that a solution is found plotted against γ for 100K-state random problems. As we increase γ, there is a clear phase transition in solubility. The observed mushy region is approximately bounded by the analytical bounds.

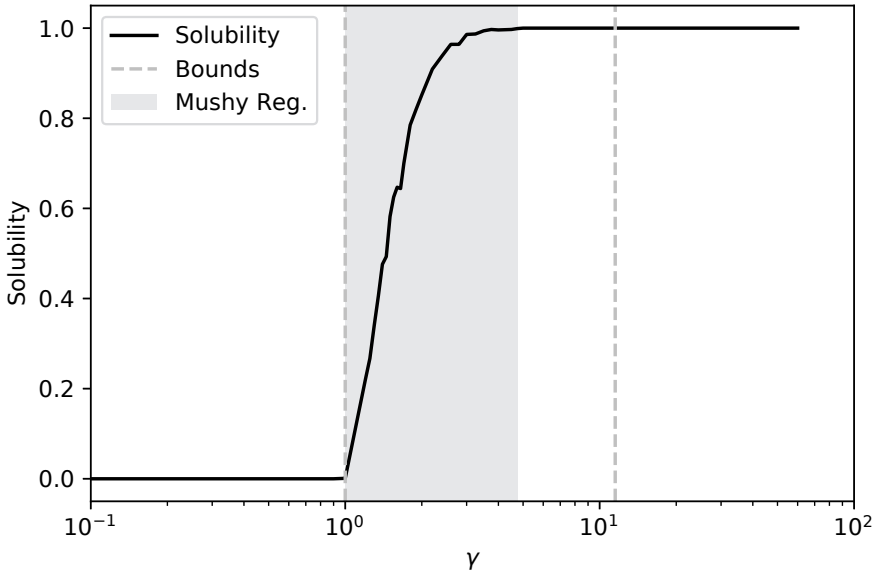


Figure 3.1: Solubility plotted against γ (log scale).

3.4.2 The Difficulty of Problems

Figure 3.2 shows the median (50%-percentile) search effort required to find a first solution using h_4 plotted against γ, for 100K-state random problems. There is a clear easy-hard-easy pattern in search effort and the hard problems are associated with the phase transition in solubility. The peak in search effort is located in close proximity to the crossover point ($\gamma \approx 1.65$). These results are consistent with the behavior observed for CSPs [164].

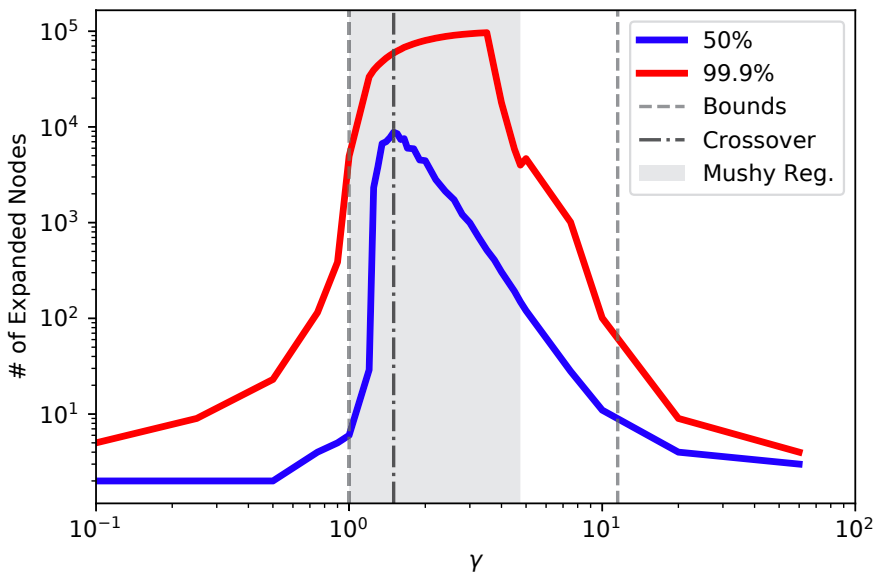


Figure 3.2: The 50%-, 99.9%-percentile effort in number of nodes expanded (log-log scale).

The hardest instances, however, do not occur near the peak of the median search effort. Figure 3.2 also shows the 99.9%-percentile search effort which peaks close to the end of the phase transition at the 99% solubility point. This can be explained, based on our model. The insoluble instances are the rare cases in which a very large giant component is formed with an initial state inside the giant component and a goal state outside. For a forward search, these are the hardest instances to solve. We expect to find similarly hard instances in the same region of high solubility for other types of search. For a backward search, the hardest instances will be the exact opposite, in which the goal state is inside a large component and the initial state outside. For bidirectional search, we expect the hardest instances to be those in which two similarly large components are formed, with the goal state in one and the initial state in another.[3] The hardest soluble instances are the rare cases of a very large giant component that is still very sparsely connected, which requires the search to exhaust most of it in order to find the solution. While we might be able to significantly reduce the required search effort for the hardest soluble instances using a better heuristic function, the hardest insoluble instances require exhausting the accessible portion of the state space to prove infeasibility and cannot be eliminated or improved using a different heuristic function.

Similar behavior of the median and higher percentiles have been observed by Gent and Walsh [52, 53] for different classes of SAT problems, including k-SAT, and by Hogg and Williams [90] for graph coloring.

3.4.3 The Impact of the Heuristic

While it is standard to use instances across the phase transition to compare heuristic quality (e.g., Gent et al. [50], McCreesh et al. [126]), to our knowledge, previous work has not studied the behavior of a set of heuristics with a known, analytical quality ranking (such as our, h_i, heuristic family) across the phase transition.

[3] A different generator is required to generate such instances.

Figure 3.3 shows the effort curve for $h_0, h_1, ..., h_4$ for the *soluble* random instances with 100K states. While for h_0 there is no reduction at all as we move towards less constrained regions, h_4 expands at the crossover approximately 68 times more nodes than at the end of the phase transition ($\gamma \approx 4.75$), and approximately 2,500 times more nodes than at $\gamma \approx 20$. Also, while h_1 expands only twice the number of nodes expanded by h_4 at the crossover, it expands approximately 90 times more nodes at $\gamma \approx 4.75$, and more than 800 times more nodes at $\gamma \approx 20$.

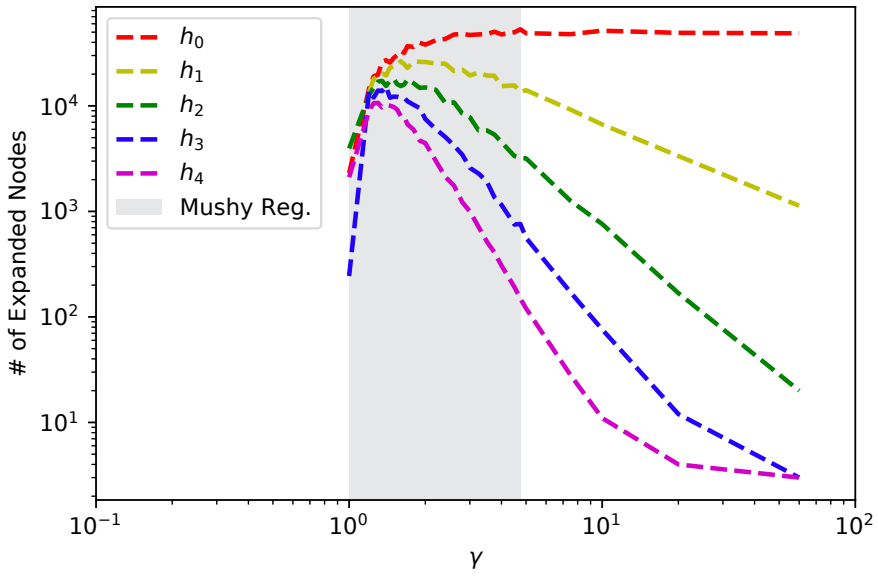


Figure 3.3: Median effort (log-log scale).

3.5 The Phase Transition in the Benchmarks

In this section, we empirically evaluate the applicability of the results of the abstract model to the analysis of existing heuristic search benchmark problems, based on the framework in Section 3.3.2. In order to perform experiments with thousands of random instances, we use relatively small versions of these problems. We do not expect to see the exact phenomena observed on the abstract model because a set of random variants of an existing problem will have a much smaller variation than our abstract problem generator. However, we expect to observe similar phenomena in close proximity to their occurrence in the abstract model.

As with our study of heuristics of systematically differing strength, we are unaware of phase transition work on other problems that has attempted to directly apply the abstract models of phase transition behavior to existing benchmarks.

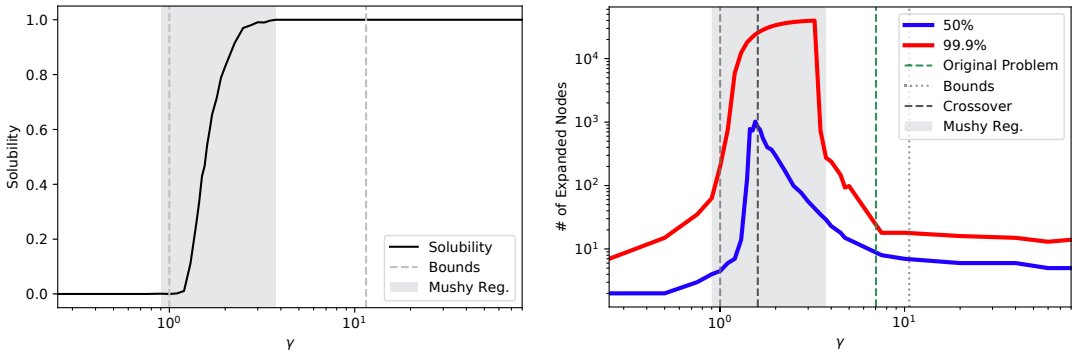
3.5.1 Solubility and Problem Difficulty

The Pancake Problem

We consider the 8-Pancake Problem, for which the observed connectivity density is $\mathcal{P} = \frac{8!\cdot 7}{8!\cdot(8!-1)} = \frac{7}{8!-1}$ ($\gamma \approx 7$). For $p < \mathcal{P}$, we generate restricted instances of the problem, in which some of the flip operators are not allowed. For $p > \mathcal{P}$, we generate relaxed instances, in which there exist additional operators that

do not correspond to valid flip. We use h^{gap} [77], a landmark heuristic based on the number of gaps in the pancake stack.

Figure 3.4 shows the probability of a solution and the required search effort. There is a clear phase transition in solubility and the mushy region is *approximately* bounded by the analytical bounds (we find one soluble instance at $\gamma = 0.9$). The median effort peaks near the crossover point with the hardest problems near the end of the mushy region. Since $\mathcal{P}$ is relatively high, we can see that restricting the problem, does not immediately reduce the solubility. These results are in agreement with our abstract model.



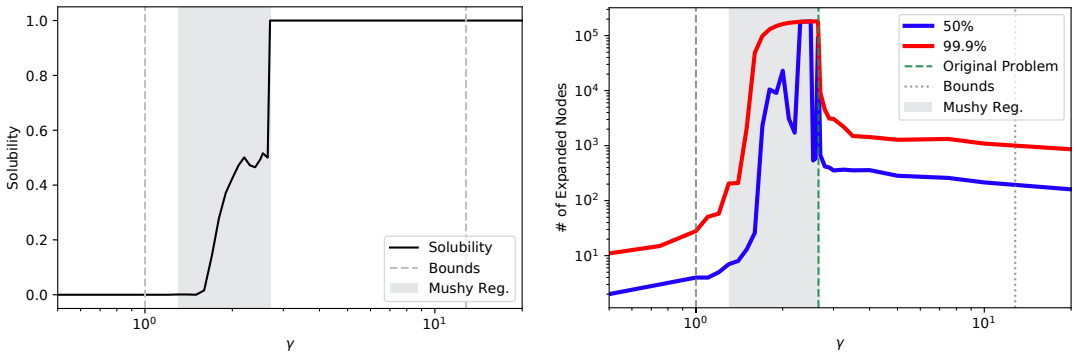
(a) Solubility plotted against γ (log scale on the x-axis).

(b) Search effort (50% and 99.9% percentile) plotted against γ (log-log scale).

Figure 3.4: Results for the 8-Pancake Problem.

Sliding Puzzle

For the Sliding Tile 8-Puzzle, we observe a different behavior (Figure 3.5). In its original form, the 8-Puzzle is not always soluble if the initial and goal states are randomly chosen because the state space consists of two equal-sized connected components. Consequently, the solubility at the original constrainedness $\mathcal{P}$ is approximately 50%. Furthermore, the behavior in the near-$\mathcal{P}$ region is different. As the problem is relaxed, it immediately becomes 100% soluble, since every edge that connects the two large components results in a fully connected state space. Alternatively, as we restrict the problem, the solubility does not immediately decline, and remains approximately 50% for a short range of γ, and the median effort therefore belongs to either a soluble or insoluble instance. Due to the unique structure of two large components, almost all the insoluble instances in the near-$\mathcal{P}$ region require near-maximal effort, as we have to exhaust one of the components. The result is a very noisy median as shown in Figure 3.5. As we further restrict the problem, the two large components break into smaller components and the effort decreases.

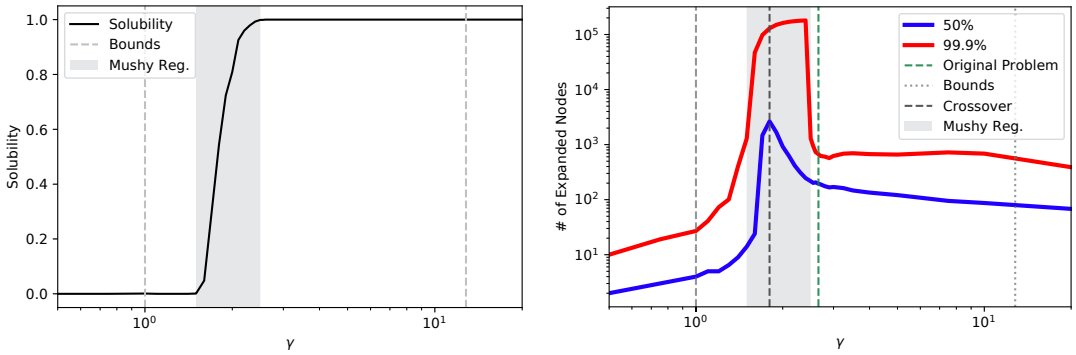


(a) Solubility plotted against γ (log scale on the x-axis).

(b) Search effort (50% and 99.9% percentile) plotted against γ (log-log scale).

Figure 3.5: Results for the 8-Puzzle Problem.

We also considered a variant of the problem where we generate problems in which the initial state and the goal state are in the same component, and the added (resp., removed) edges of the relaxed (resp., restricted) instances are within this component. This allows us to observe the full phase transition on one component and to avoid the sudden connection of two large components. Figure 3.6 shows results that are more consistent with the abstract model and the previous benchmarks.



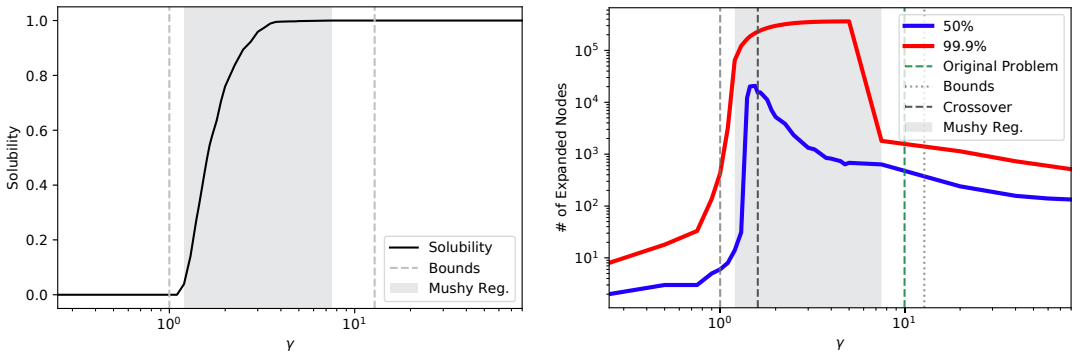
(a) Solubility plotted against γ (log scale on the x-axis).

(b) Search effort (50% and 99.9% percentile) plotted against γ (log-log scale).

Figure 3.6: Results for the 8-Puzzle Problem (one component).

TopSpin

We consider a 10-disk TopSpin with a 4-disk turnstile. The observed connectivity density is $\mathcal{P} = \frac{9!\cdot 10}{9!\cdot(9!-1)} = \frac{10}{9!-1}$ ($\gamma \approx 10$). To solve the problem we use a heuristic based on a pattern database (PDB) that abstracts the 6 disks with the highest face value. Figure 3.7 shows the probability of solution and the 50% and 99.9% percentile search effort. The results show similar patterns to other domains.



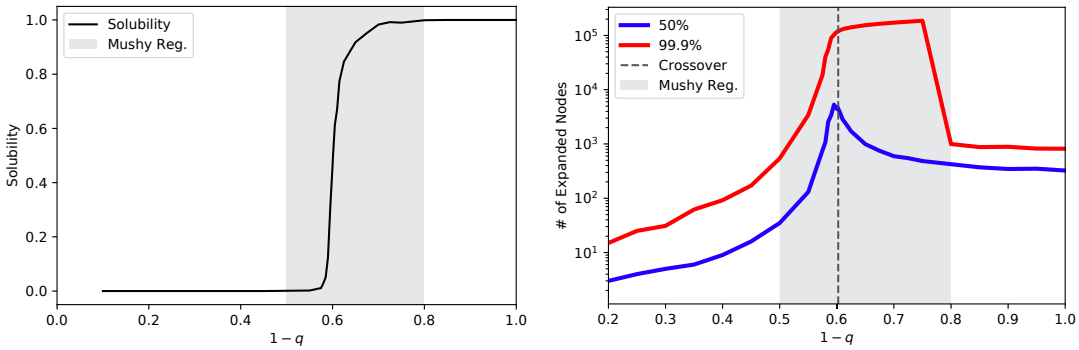
(a) Solubility plotted against γ (log scale on the x-axis).

(b) Search effort (50% and 99.9% percentile) plotted against γ (log-log scale).

Figure 3.7: Results for the TopSpin Problem.

Grid Navigation

We consider a 1000×1000 Grid Navigation Problem based on Model 3 where we vary the probability of a blocked cell q. Figure 3.8 shows the probability of solution and the required search effort plotted against the probability that a cell is not blocked $(1 - q)$. The result for the domain-specific model also match the results on the abstract model. We observe a rapid transition in the problem solubility, and the median effort peaks at the crossover point.



(a) Solubility plotted against γ (log scale on the x-axis).

(b) Search effort (50% and 99.9% percentile) plotted against γ (log-log scale).

Figure 3.8: Results for the Grid Navigation Problem (Model 3).

3.5.2 The Impact of the Heuristic Function

In order to evaluate the impact of the heuristic's quality on the search effort for existing heuristic search problems, we carried out a series of experiments on the Pancake Problem.

We consider the original h^{gap} heuristic [77], and propose $H'^{gap} = \{h_0'^{gap}, h_1'^{gap}, h_2'^{gap}, ...\}$, a set of partial and increasingly stronger versions of h^{gap}:

$$h_i'^{gap}(x) = h^{gap}(\text{bottom } i \text{ pancakes in } x)$$

For an n-Pancake problems, $h_0'^{gap}$ is the uninformed zero heuristic, while $h_n'^{gap}$ is the original h^{gap} heuristic.

Figure 3.9 and Table 3.1 show the median effort for soluble instances. Similar to the abstract model, the impact of a better heuristic is much stronger outside of the phase transition. $h_8'^{gap}$ is approximately 1.7 times better than $h_2'^{gap}$ at the crossover point ($\gamma \approx 1.55$), approximately 10.9 times better at the end of the mushy region ($\gamma \approx 3.5$), and approximately 71 times better at $\gamma \approx 60$. Note the very small difference in effort among $\{h_i'^{gap} | i > 0\}$ at the crossover point.

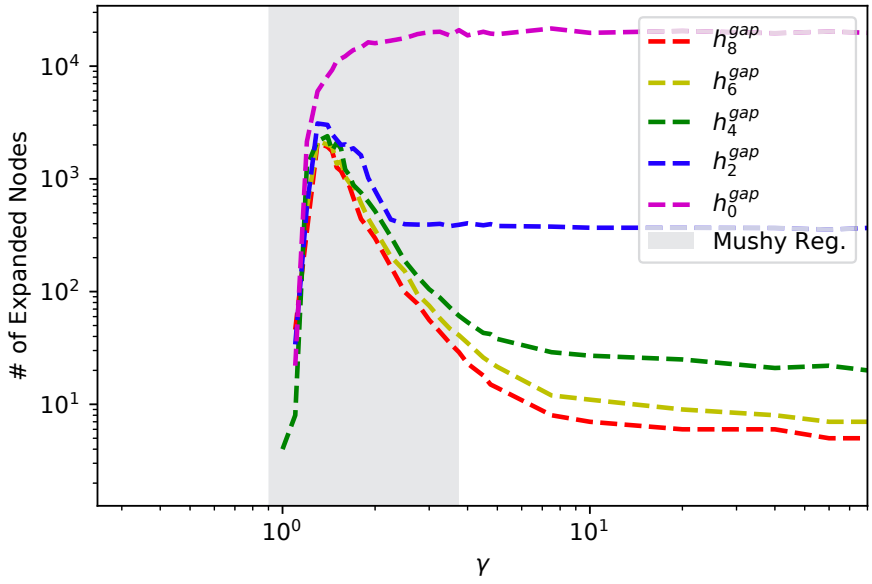


Figure 3.9: 8-Pancake Problem: Search effort for soluble instances using $h_0'^{gap}$, $h_2'^{gap}$, $h_4'^{gap}$, $h_8'^{gap}$ (log-log scale).

γ value	$h_0'^{gap}$	$h_2'^{gap}$	$h_4'^{gap}$	$h_6'^{gap}$	$h_8'^{gap}$
$\gamma \approx 1.55$	11,670	2,004	1,854	1,423	1,174
$\gamma \approx 3.5$	18,728	381	72	47	35
$\gamma \approx 20$	20,524	368	25	9	6
$\gamma \approx 60$	20,293	355	22	7	5

Table 3.1: 8-Pancake Problem: Median effort.

3.5.3 Discussion

The successful prediction of the behavior of the benchmark problems indicates some structural properties of these problems are properly modeled by our abstract model. As all these benchmark problems are essentially puzzles, their state space is symmetric, they have roughly the same number of operators for every state, and they have one goal state. The fact that the 8-Puzzle results do not match our abstract model is due to a mismatch with the connectivity assumptions in our model. The 8-Puzzle demonstrates interestingly different behavior and the use of the phase transition framework leads us to useful insights

into the structure of the problem.

While the analytical bounds on the mushy region seem to approximately hold, even on the benchmarks, predicting the location of the crossover remains an open question. For our model, the number of solutions corresponds to the number of s-t paths in a directed graph, a well-studied problem [177]. According to the theory of constrainedness [51], the crossover can be predicted at the point where the expected number of solutions is exactly one. Unfortunately, calculation of the expected number of s-t paths in random digraph remains an open problem with some potentially useful estimation results [153].

Our investigation also showed that the effort improvement due to a more informed heuristic is much smaller for more constrained problems, especially for problems inside the phase transition – exactly the hardest problems. If we can hope to generalize empirical comparisons of heuristics and develop an understanding of problem difficulty for heuristic search, we should, therefore, take into account the location of the problem sets on the phase transition as is done, for example, in CSPs [50].

In the next sections, we empirically analyze how the phase transition phenomenon interacts with two algorithm design decisions: the use of cost-based heuristics on problems with differing operator cost ratio [195, 197, 31, 32] and whether or not to allow re-opening of closed nodes [175, 159, 160]. As these two decisions have been shown to impact the problem difficulty, it will be interesting to study their impact across the phase transition region.

3.6 Node Re-Expansions

In A*, $f(n) = g(n) + h(n)$ and re-expansions of previously visited nodes only occur when using an inconsistent heuristic. In suboptimal search algorithms such as GBFS and Weighted A*, as $g(n)$ is not considered or is weighted less than $h(n)$, we are not guaranteed to avoid re-expansions even when using a consistent heuristic.

Although no theoretical or empirical analysis of which we are aware has specifically addressed re-expansions in GBFS, the authors of several recent works chose to configure GBFS to not re-open closed nodes, even if a shorter path is found [176, 203].

For Weighted A*, re-expansions can have significant negative effect on the search effort. Valenzano, Sturtevant and Schaeffer [175] presented empirical analysis of re-expansions for Weighted A* on pathfinding problems. As they increased the weight on $h(n)$, the proportional search effort spent on re-expansions of visited nodes increased. For $w = 10$, they observe that 91% of the total node expansions were re-expansions. Sepetnitsky, Felner and Stern [160] performed an empirical analysis for Weighted A* and showed that in nearly 90% of the cases, a policy of node re-opening leads to a search effort that is at least as high as no-reopening.

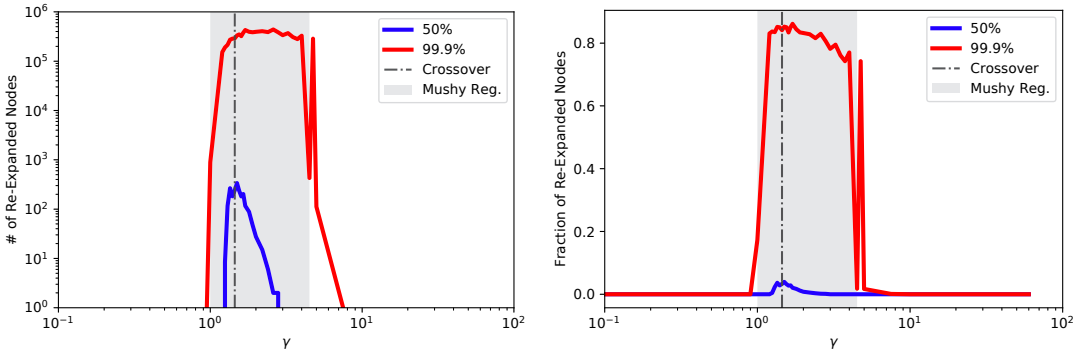
In this section we present an empirical analysis of the effect of node re-expansions across the constrainedness range. We focus on GBFS and configure the search to re-open closed nodes if a cheaper path is found. As before, we randomly break h-value ties. Similar to Section 3.4 and Section 3.5, we focus on unit-cost problems and generate 1000 random problem instances for each sampled γ values in $[0, |nodes| - 1]$. In Section 3.6.1, we present results on our abstract model and in Section 3.6.2 we present results on the benchmark problems. In each experiment, we analyze both the absolute number of node re-expansions as well as the fraction of node re-expansions out of the total number of expanded nodes [175].

3.6.1 Results on the Abstract Model

In this section, we present an analysis of node re-expansions on our abstract model (Model 1). We present results for 100K-state random problems solved with the abstract heuristic h_4.

Node Re-Expansion Across the Constrainedness Range

Figure 3.10 shows the 50% and 99.9% percentile of the absolute number of node re-expansions and the fraction of node re-expansions out of the total number of expanded nodes for 100K-state random problems. As with problem difficulty, node re-expansions has a low-high-low pattern, with the median peaking at the crossover point. For the hardest problems (99.9% percentile), we also observe a low-high-low where the peak is more widely spread across the mushy region.



(a) Number of re-expanded nodes against γ (log-log scale).

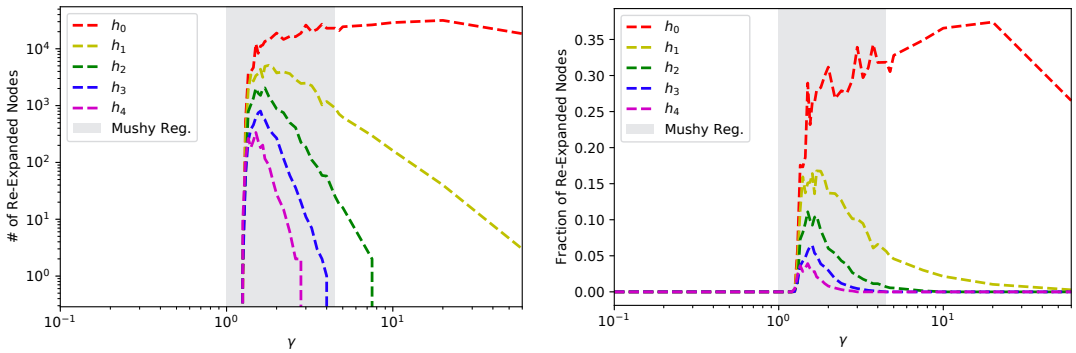
(b) Fraction of re-expanded nodes plotted against γ (log scale on the x-axis).

Figure 3.10: Node Re-expansions on the abstract model.

The Impact of the Heuristic

In our analysis of the impact of the heuristic in Section 3.4.3, the search effort for insoluble problems is not impacted by the heuristic (all accessible nodes will be visited once). When we allow re-opening of nodes, the heuristic can impact the number of re-expansions in both soluble and insoluble problems. We therefore include insoluble instances in our analysis.

Figure 3.11 shows the absolute and relative 50% percentile of node re-expansions for $h_0, ..., h_4$ for 100K-state random problems (both soluble and insoluble). Similar to problem difficulty, the reduction in node re-expansion is lower for less informed heuristics and for h_0 we observe no reduction. We cannot directly compute the ratio between heuristics since the number (and fraction) of re-expansions for the more informed heuristics is zero starting from some γ value.



(a) Number of re-expanded nodes against γ (log-log scale).

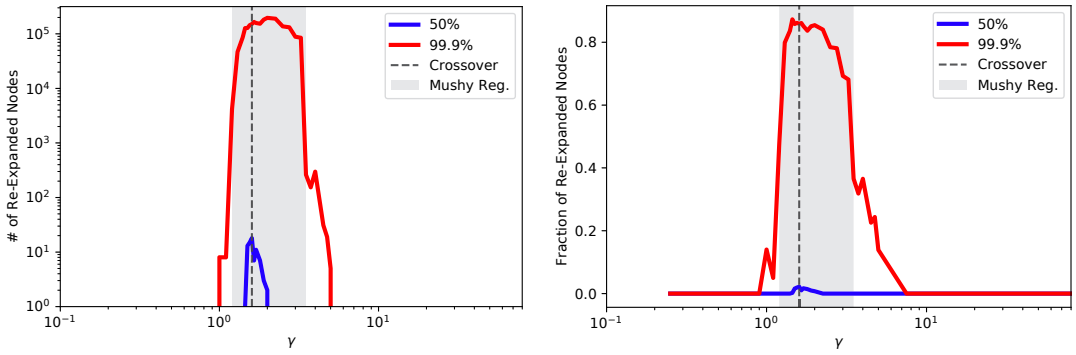
(b) Fraction of re-expanded nodes plotted against γ (log scale on the x-axis).

Figure 3.11: Node Re-expansions in the abstract model with $h_0, ..., h_4$.

3.6.2 Results on the Benchmarks

Pancake Problem

Figure 3.12 shows the absolute and relative 50% and 99.9% percentiles of node re-expansions for the 8-Pancake problem. The median is very low, which can be attributed to the quality of the *gap heuristic*. Still, we see that the median number of node re-expansions peaks inside the phase transition region and reaches zero as we move away from the phase transition. For the higher percentiles of node re-expansions, we observe a low-high-low pattern and a wider peak. Similar trends are observed for the relative number of re-expansions.



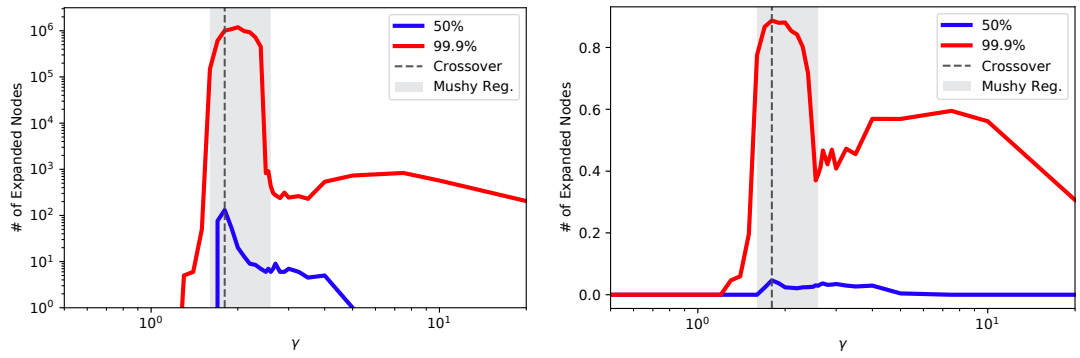
(a) Number of re-expanded nodes against γ (log-log scale).

(b) Fraction of re-expanded nodes plotted against γ (log scale on the x-axis).

Figure 3.12: Node Re-expansions in the 8-Pancake Problem.

Sliding Tiles

Figure 3.13 shows the results for the one-component variant of the 3×3 Sliding Tiles Problem. The number of re-expansions peaks inside the phase transition and the median reaches zero shortly after leaving the phase transition region. The higher percentiles do not reach zero in the sampled γ range, however we can see the decline as we move away from the phase transition. As expected, the peak of the higher percentiles is wider for higher percentiles.

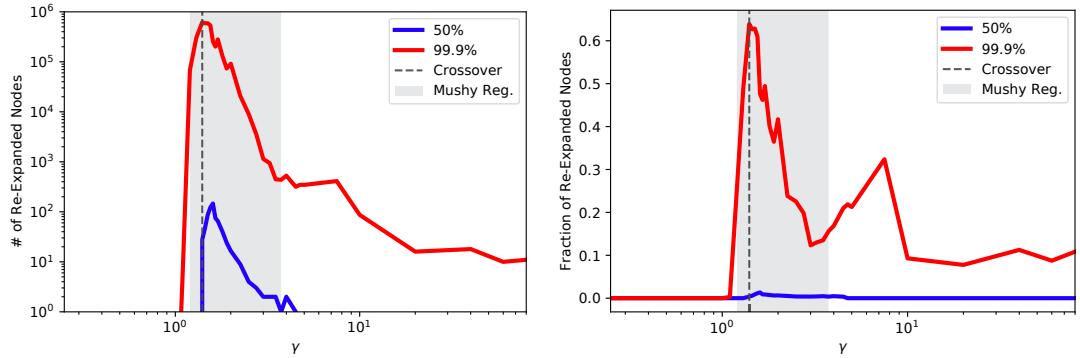


(a) Number of re-expanded nodes against γ (log-log scale). (b) Fraction of re-expanded nodes plotted against γ (log scale on the x-axis).

Figure 3.13: Node Re-expansions in the Sliding Tiles Problem.

Top Spin

Figure 3.14 shows the absolute and relative 50% and 99.9% percentiles of node re-expansions across the constrainedness range. While the results are a bit noisy, the re-expansions peak inside the mushy region in close proximity to the crossover point.

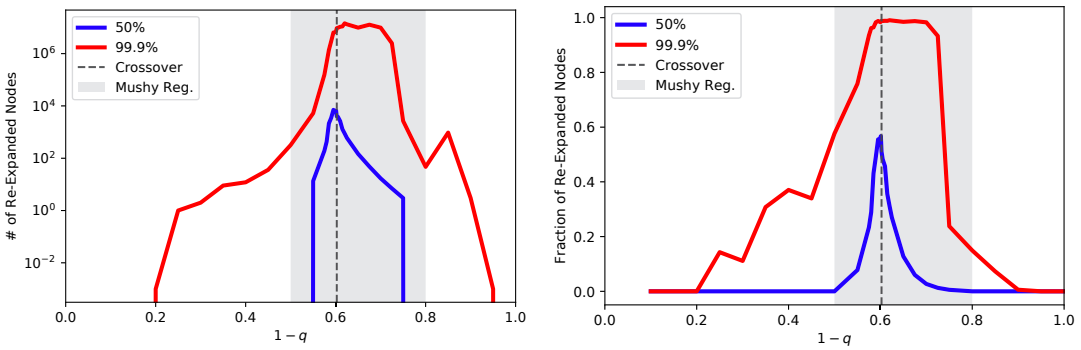


(a) Number of re-expanded nodes against γ (log-log scale). (b) Fraction of re-expanded nodes plotted against γ (log scale on the x-axis).

Figure 3.14: Node Re-expansions in the TopSpin Problem.

Grid Navigation

Figure 3.15 shows the results for the Grid Navigation Problem. On average, re-expansions occur within the mushy region and the median number of re-expansions outside of the phase transition region is zero. For the hardest problems, we see a wide peak inside the mushy region, similar to previous domains.



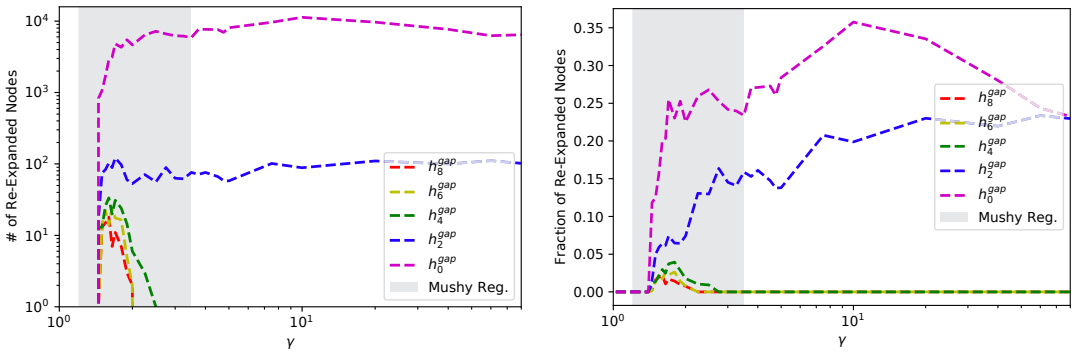
(a) Number of re-expanded nodes against γ (log-log scale).

(b) Fraction of re-expanded nodes plotted against γ (log scale on the x-axis).

Figure 3.15: Node Re-expansions in the Grid Navigation Problem.

The Impact of the Heuristic

In this section, we analyze the impact of the heuristic on node re-expansions in a benchmark problem. Similar to Section 3.5.2, we analyze the 8-Pancake problem with the set of increasingly strong heuristics $H'^{gap} = \{h_0'^{gap}, h_1'^{gap}, h_2'^{gap}, ...\}$. Figure 3.16 shows the absolute and relative 50% percentile of node re-expansions. We can see similar patterns to the abstract experiment, where the more informed heuristics exhibit a low-high-low pattern, while for the less informed heuristics, re-expansions remain high in the relaxed region. Specifically, for $\{h_i'^{gap} | 4 \leq i \leq 8\}$ we see a clear low-high-low, while for $h_2'^{gap}$ and the completely uninformed $h_0'^{gap}$ the re-expansions remain high even in the relaxed region.



(a) Number of re-expanded nodes against γ (log-log scale).

(b) Fraction of re-expanded nodes plotted against γ (log scale on the x-axis).

Figure 3.16: Node Re-expansions in the 8-Pancake Problem.

The empirical analysis in this section shows that the effect of node re-expansions depends on the constrainedness of problems. This result suggests that, in addition to the implications to the number of expanded nodes in GBFS with no re-opening of closed nodes (discussed in Sections 3.4 and 3.5), the phase transition phenomenon also interacts with other factors that impact the problem difficulty such as the re-expansion of nodes. In the next section, we study the interaction between the phase transition phenomenon and the operator cost ratio, an additional factor that was shown to be connected to problem difficulty [195, 31, 32].

3.7 Cost-based Heuristics

Previous work has highlighted the negative impact of large operator cost ratio in cost-based search and proposed size-based search as an alternative to cost-based search (see Section 2.3.2). In this section, we study the impact of cost-based heuristics on problems with differing operator cost ratio across the constrainedness range. Unlike previous sections, we are not able to demonstrate results on the abstract models due to the incompatibility of the look-ahead abstract heuristic to cost-based search. However, we perform an empirical analysis of cost-based search on the benchmark problems using the analytical model for benchmarks (Section 3.3.2). For each of the benchmark domains, we propose a cost function that is flexible enough to allow us to control the operator cost ratio and examine the change in search effort as we manipulate it.

All the experiments in this section use GBFS configured to not re-open closed nodes and to randomly break ties in h-values. We generated random problem instances for 25 γ values in $[0, |nodes| - 1]$, with higher density inside the phase transition. For each γ value we generated 1000 random instances. For each instance, we record its solubility and the number of nodes expanded in order to find a solution or prove that none exists.

3.7.1 Median-Case Analysis

Pancake Problem

We consider the 8-Pancake Problem, based on Model 2. To control the operator cost ratio, we define a cost function that is based on the bottom pancake in the sub-pile that is about to be flipped. Although somewhat artificial, this cost function is easily incorporated into the *gap heuristic* [77] and allows us to investigate the effect of the operator cost ratio on the search effort. Given z, the size of the lowest pancake in the flipped sub-pile, we define the cost of the flip to be z^m. Again, we use the parameter m to control the operator cost ratio, which is 8^m for the 8-Pancake Problem.

To directly compare the effect of the different cost functions across the phase transition, we analyze the relative number of expanded nodes. Since the operator cost ratio has no effect on the search effort required to solve insoluble instances (i.e., every node that is accessible from the initial state will be expanded exactly once to prove insolubility), we focus only on the soluble instances. To avoid bias due to small sample size, we only consider the points in the phase transition in which at least 10% of the problems are soluble.

Figure 3.17 shows the median ratio of the number of expanded nodes when using a cost function with higher operator cost ratio (i.e., $m \in \{2, 3, 4\}$) to the search effort when using a low operator cost ratio heuristic ($m = 1$). As Figure 3.17 clearly shows, the increase in search effort, associated with the large operator ratio, is centered in the region of phase transition. Outside that region, the effort ratio gradually diminishes towards a ratio of one.

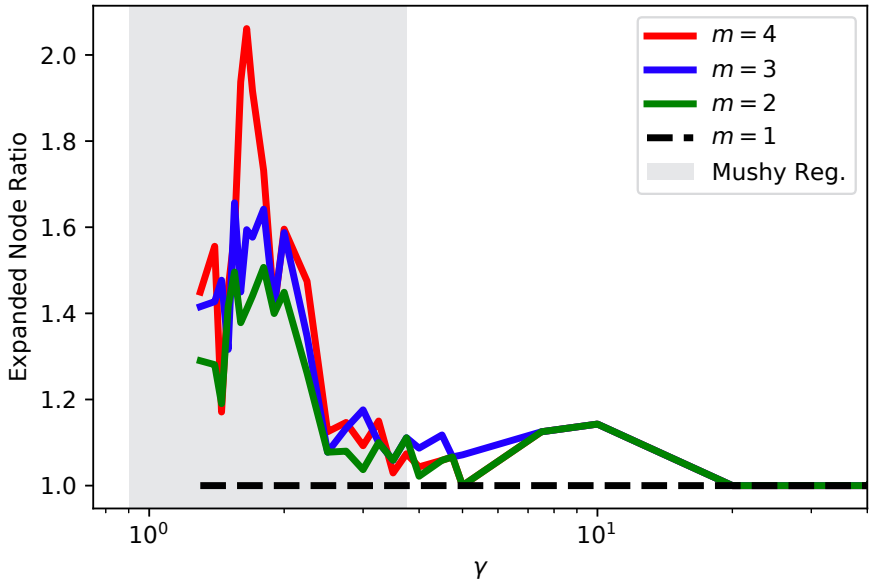


Figure 3.17: 8-Pancake Problem: Median effort ratio of soluble instance vs. γ.

Figure 3.18 shows the median search effort ratio for the 8-Pancake Problem with a large operator ratio (m=4) between the cost heuristic and the distance heuristic d. The improvement due to the distance heuristic is also concentrated in the phase transition region. In fact, the distance heuristic seems to behave similarly to a cost-based heuristic with a lower operator cost ratio (which, of course, it is).

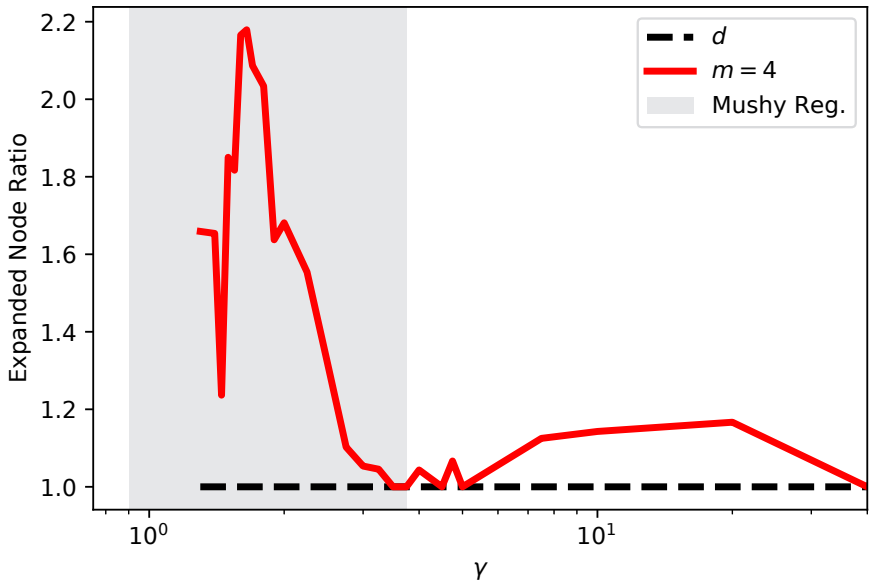


Figure 3.18: 8-Pancake Problem: Median effort ratio of soluble instance vs. γ.

Sliding Tiles

We consider the 3×3 Sliding Tiles Problem based on Model 2. As shown earlier, the state space consists of two large disconnected components [193] and we therefore generate problems in which the initial state and the goal state are in the same component.

Wilt and Ruml [195] incorporated costs for the Sliding Tiles problem by assigning different costs for moving each tile. In a later work, they showed that using an inverse cost structure, in which the cost of moving a tile is in inverse correlation to the face value of the tile, has a larger expected local minimum [197]. We therefore use a parameterized version of Wilt and Ruml's cost function in which the cost of moving a tile with a face-value of z is $\frac{1}{z^m}$. The parameter m controls the operator cost ratio, which is 8^m, for this domain. The heuristic function is based on the standard Manhattan Distance, weighted by the cost associated with each tile.

Figure 3.19 shows the median search effort ratio between a cost-based search for $m \in \{1, 1.5, 2\}$ and a search based on the distance heuristic d. In this case we see that the impact on search effort ratio that is associated with a larger operator cost ratio peaks just after the end of the observed mushy region. This can be due to the unique structural properties of this state space (the peak is located near the constrainedness of the original problem and the relaxed instances are all soluble). While the negative impact of operator cost ratio extends slightly beyond the phase transition region, it is still directly connected to the constrainedness of problems and diminishes as we move away from the phase transition.

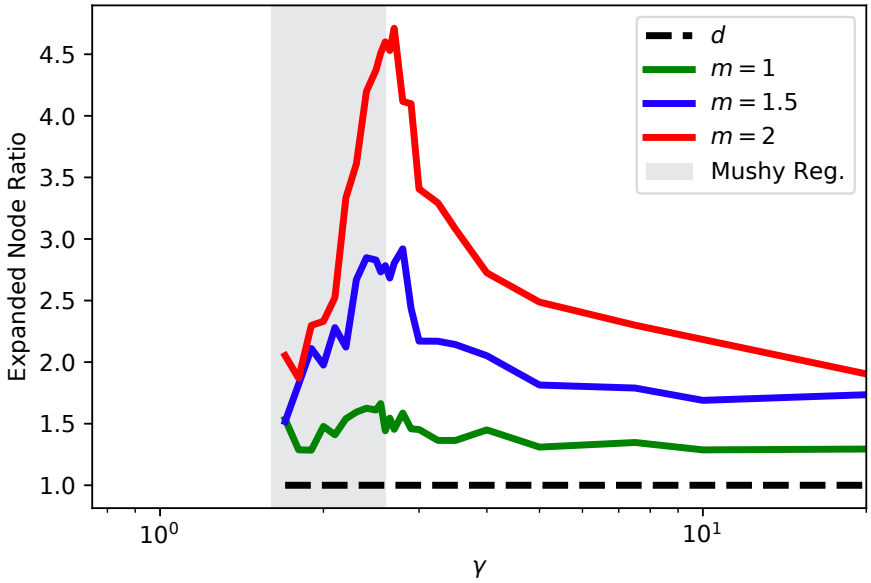


Figure 3.19: 3×3 Sliding Tiles: Median effort ratio of soluble instance vs. γ.

TopSpin

Wilt and Ruml [197] found that in some cases, using a cost-based heuristic requires less search effort than using the distance heuristic, due to smaller local minima in the cost-based heuristic. We observe such behavior for the TopSpin domain.

We consider a 10-disk TopSpin domain with a 4-disk turnstile. Our cost function is based on the sum

of faces of the disks in the turnstile:

$$C_m(s,a) = (\sum_{z \in T_a} z)^m$$

where T_a is the set of faces of the disks in the turnstile, and m is a parameter controlling the operator cost ratio.

Figure 3.20 shows the median search effort for the TopSpin problem with a PDB heuristic for $m \in \{1, 1.5, 2.0\}$, compared to a distance-based heuristic. As expected, we observe an increased effort as we increase the operator cost ratio that is centered inside the phase transition. In this case the distance heuristic is not strictly better than the cost-based heuristics. For the more relaxed instances, the cost-based heuristics actually expand fewer nodes. Furthermore for $m = 1$, we see that the distance heuristic sometimes expands more nodes even inside the phase transition region. This is consistent with Wilt and Ruml's observation.

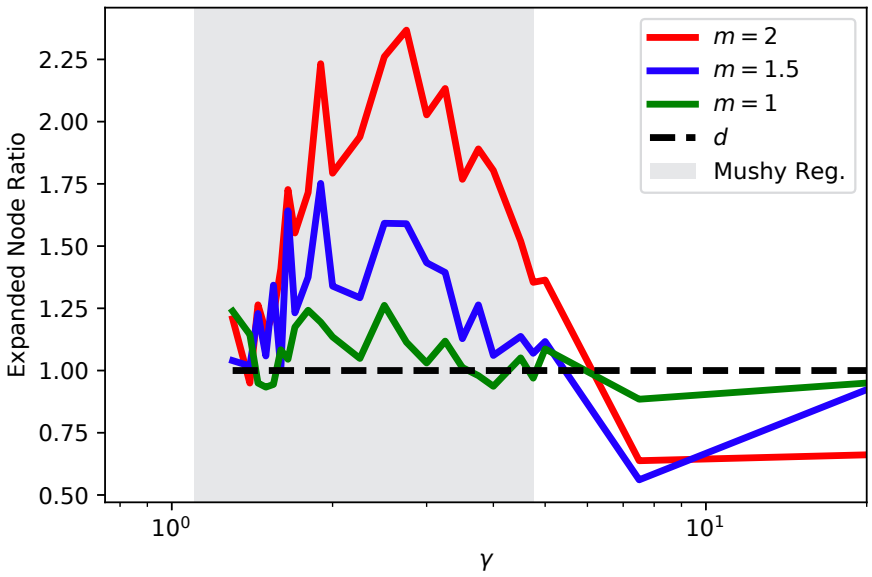


Figure 3.20: 10-disk TopSpin: Median effort ratio of soluble instance vs. γ.

Grid Navigation

We consider a 1000×1000 Grid Navigation Problem based on Model 3. We define $C_m(s,a)$, the cost of applying action a on state s, using a parameter m:

$$C_m(s,a) = \begin{cases} 1^m, & \text{if } a = up \\ 2^m, & \text{if } a = down \\ 3^m, & \text{if } a = left \\ 4^m, & \text{if } a = right \end{cases}$$

The parameter m controls the operator cost ratio. As the smallest operator cost is fixed to 1, the operator cost ratio is then 4^m. The heuristic function is based on Manhattan Distance, weighted accordingly.

Figure 3.21 shows the median search effort ratio between a cost-based search for $m \in \{2, 4, 10\}$ and a search based on the distance heuristic d. The increase in search effort due to large operator ratio is centered in the region of phase transition. Outside that region, the effort ratio gradually diminishes towards a ratio of one.

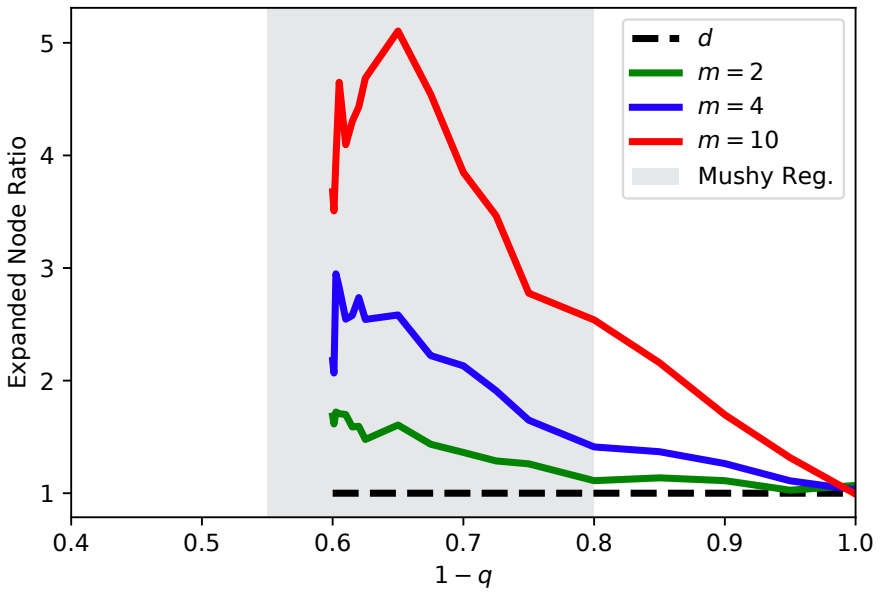
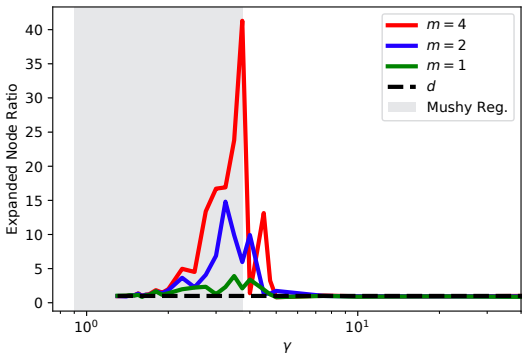


Figure 3.21: Grid Navigation: Median effort ratio of soluble instance vs. γ.

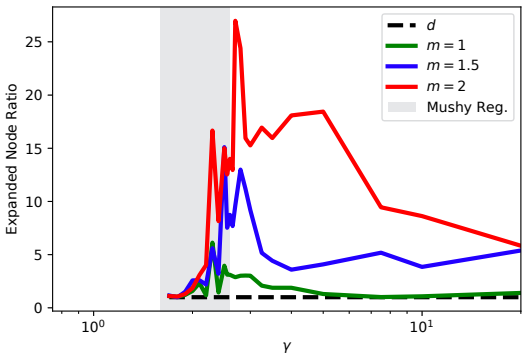
3.7.2 The Hardest Instances

In the previous section we investigated the effect of operator cost ratio on the median search effort. Here we examine the hardest instances across the constrainedness range.

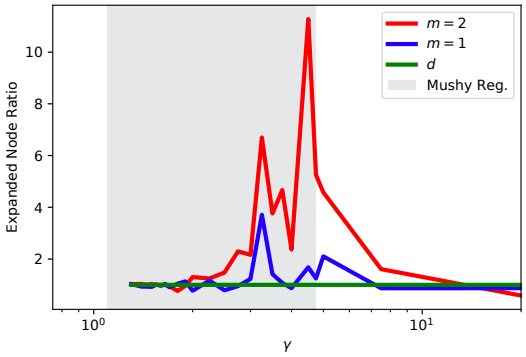
Figure 3.22 shows the 99.9%-percentile effort ratio for soluble instances for the four benchmarks. The differences are significantly larger inside the phase transition region, compared to the median case. However, high-percentile ratios too, gradually diminish as we move away from the phase transition. Interestingly, the peak of the ratio curve slightly shifts to the more relaxed areas, compared to the median case.



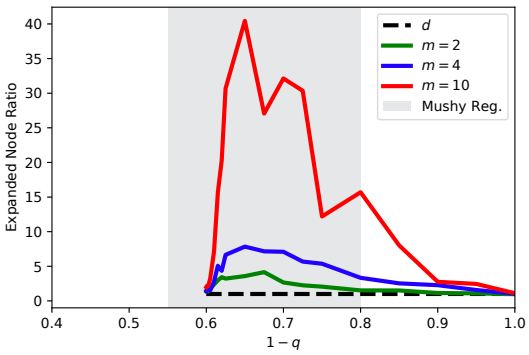
(a) 8-Pancake Problem.



(b) Sliding Tiles.



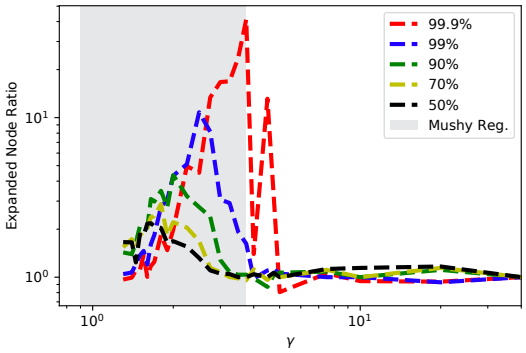
(c) TopSpin.



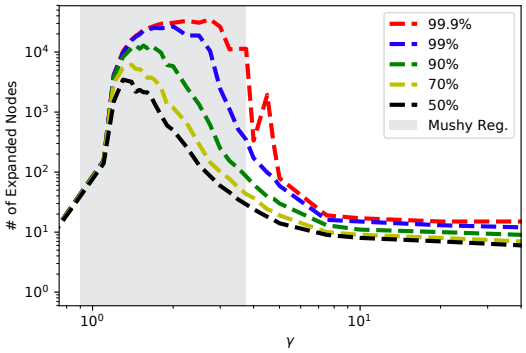
(d) Grid Navigation.

Figure 3.22: 99.9%-percentile effort ratio vs. γ.

The shift in peak suggests that the largest ratio for the hardest problems is found in a more relaxed region of the phase transition. Figure 3.23a and Figure 3.23b show the relative effort (compared to the distance heuristic) and absolute effort in the higher percentiles of the Pancake problem with $m = 4$. We can see that the peak of the highest percentiles of the absolute effort and the peak of the effort ratio both shift to the more relaxed regions in the phase transition. However, the results suggest that this phenomenon appears in the highest percentiles ($\geq 99\%$). Similar results have been observed for the other domains. Figure 3.24 shows similar analysis to Figure 3.23b for the other benchmarks.

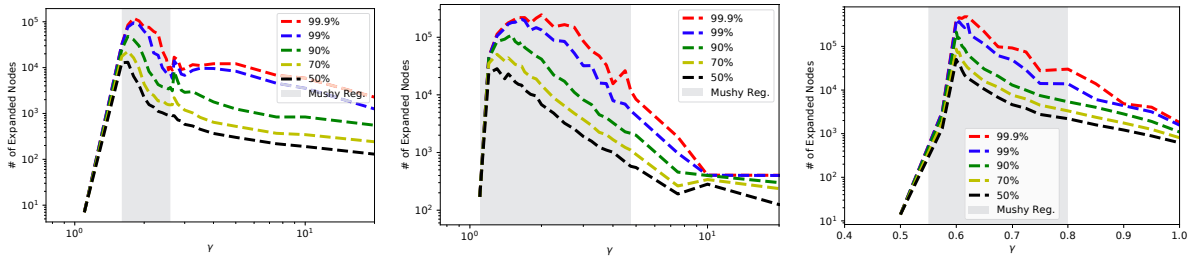


(a) Relative effort.



(b) Absolute effort.

Figure 3.23: 8-Pancake Problem.



(a) Sliding Tiles. (b) TopSpin. (c) Grid Navigation.

Figure 3.24: Higher percentiles of absolute effort.

Exceptionally Hard Problems

The anomaly of finding the hardest instances in the "easy" regions of the phase transition has been observed for other types of computational problems [52, 53, 90]. Such problem instances have been termed *exceptionally hard problems (ehps)*. The *ehps* are not simply outliers but rather outliers in an unexpected region of the phase transition and hence have been the subject of a number of investigations [52, 165].

We previously showed that, for unit-cost problems, the 99.9%-percentile peaks in the "easy" region [26]. However, that curve is dominated by insoluble instances. This result is not surprising, since heuristic search with a heuristic function that returns a finite value has to exhaust the accessible state space to prove infeasibility. However, the existence of soluble *ehps* in heuristic search is a new result.

3.8 Discussion

Our results show that the phase transition phenomenon is strongly connected to problem difficulty. First, we show that the median effort exhibits an easy-hard-easy pattern that peaks inside the phase transition region. Then, we show that two other factors that impact of two factors that were shown to be associated with higher search effort, the operator cost ratio and the re-expansion of nodes, depends on the constrainedness of problems and peaks in the phase transition region. Consistent with these observations, we also show that the distance heuristic, that can mitigate the effect of a large operator cost ratio, provides significant improvement only inside the phase transition.

It is important to clarify that while the phase transition is useful in predicting the location of the hardest problems (both the median hardest and single hardest instances), it does not predict the required search effort. Such a prediction requires taking into account other factors such as the size of the state space and the strength of the heuristic. However, our results show that the phase transition is a key factor in search effort.

3.8.1 Limitation of the Abstract Model

Our analysis of node re-expansions and operator cost ratio highlights an important limitation of our abstract model: the use of look-ahead heuristics. Our abstract heuristic function $h_i(x)$ heuristic provides an accurate distance-to-goal for nodes that are within a distance of i nodes, otherwise it returns i. The resulting search surface consists of a (potentially large) plateau of states with an h-value of i. Once a

state that is $i-1$ steps from the goal is generated, we can directly follow a path of decreasing h-values to the goal.

Real heuristic search problems tend to have a combination of local minima and plateaus (see Section 2.3.1), however our abstract model is not capable of modelling local minima. While our abstract model could support the analysis of less informed heuristics represented by a larger plateau (i.e., a larger set states we cannot distinguish w.r.t their proximity to the goal), it cannot support an analysis of misleading heuristics represented by local minima (i.e., states that will seem like they are closer to the goal while, in fact, they are not). Such model is needed, for example, to study the operator cost ratio that induces larger local minima [197]. While our analysis of the benchmark problems does consider local minima that appear due the heuristic functions we use, developing an abstract model that can model local minima is an interessting direction for future work.

3.8.2 The Phase Transition and Local Minima.

Previous work has shown that the negative effect of increasing the operator cost ratio is due to the deepening of the local minima, while the distance heuristic tends to have smaller local minima [195, 197, 31, 32]. Our results show that the effect of increasing the operator ratio, and the benefit often gained by using the distance heuristic, is significant in the phase transition region, and decreases as we move away.

These results suggest a connection between the constrainedness of a problem and the existence of local minima. A reasonable hypothesis is that the likelihood and/or extent of local minima is much larger in the phase transition and insignificant outside. This hypothesis is supported by the discovery of soluble ehps in heuristic search. Such instances in other types of computational problems are associated with large insoluble subproblems that the search has to exhaust if it enters [166, 165, 52]. Wilt and Ruml [197] defined local minima in heuristic search as a region that does not contain the goal but that the search will have to exhaust if it enters. The similarity between these definitions, as well as their similar location in the phase transition, suggests that they are analogous phenomena. As the large insoluble subproblems are directly associated with the constrainedness of the problem, we conjecture a similar relationship exists for the local minima in satisficing heuristic search.

Several methods have been suggested to mitigate the effect of local minima, including the use of randomization [176] and local exploration [203, 204]. Investigating these methods using the framework of phase transition may yield interesting new insights.

Following these research directions, in Chapter 4, we study the connection between local minima and constrainedness in GBFS. In particular, we show that the distribution of local minima depth depends on the constrainedness of problems and results in a heavy-tailed behavior that accounts for the ehps. We also investigate the impact of different randomization techniques on the distribution of local minima.

3.9 Conclusion

We performed the first empirical analysis of the phase transition in heuristic search, focusing on greedy best first search. Our results establish the existence of a rapid transition in the solubility of heuristic search problems and the occurrence of the hardest problems during this transition. We also showed that the effect on search effort associated with node re-expansions and a larger operator ratio is concentrated in the phase transition.

The phase transition phenomenon has been a central tool in the study of problem difficulty for computational problems, and our results connect heuristic search to a variety of problems and body of literature for which similar results have been obtained.

Our results suggest that many of the phenomena that are associated with larger search effort are effected by the phase transition and, therefore, that they should be studied at different levels of constrainedness. Specifically, we hypothesize that the existence of local minima in heuristic search problems is closely related to the phase transition phenomenon. In the next chapter we focus on local minima in GBFS and present evidence that supports this hypothesis.

Chapter 4

Heavy-tailed Behavior and Randomization in Satisficing Planning

In this chapter, we continue our study of empirical models of problem difficulty in GBFS. In Chapter 3, we discovered the existence of exceptionally hard problems (ehps) that appear in ensembles of easy problems where the median effort is relatively low. In this chapter, we focus on domain-independent satisficing planning and present an empirical model that accounts for such ehps and show how this model can inform the design of more efficient algorithms that can solve ehps faster. Our empirical model is inspired by work on CSPs and SAT and demonstrates how empirical models that were developed for these problems can be adapted to heuristic search problems.

4.1 Introduction

The study of runtime distributions of several computational problems (most notably CSP and SAT) found heavy-tailed behavior for both ensembles of random problems and multiple runs of a randomized backtracking search on a single problem [60]. This behavior accounted for the phenomenon of *exceptionally hard problems (ehps)* and was shown to be related to the distribution of depth of subtrees with no solution (*inconsistent subtrees*) [62]. In critically constrained problems, the probability of a deep inconsistent subtree is very high, while in relaxed problems there is a low, but non-negligible probability of a deep inconsistent subtree. These results led to the development of techniques such as randomized restarts and tailored portfolios [60] that improve search performance by jumping out of deep inconsistent subtrees.

In Chapter 3, we focused on domain-dependent satisficing heuristic search and showed a connection between problem difficulty and problem constrainedness on ensembles of random problems. Furthermore, we discovered the existence of exceptionally hard instances that suggests that a heavy-tailed behavior, similar to the one observed for CSPs and SAT, might exist for satisficing planning.

In this chapter, we demonstrate for the first time the existence of heavy-tailed behavior in satisficing planning using GBFS. Then, we show that the analysis of the distribution of inconsistent subtree depths can be applied to GBFS as an explanatory framework for the observed heavy-tailed behavior. Building on

these results, we discuss methods to reduce the heavy-tailed behavior and the phenomenon of exceptionally hard problems. Specifically, we make the following contributions:

1. We show that fat- and heavy-tailed behavior can be observed on ensembles of random planning problems across different domains and heuristic functions.
2. We present a variant of greedy best-first search that introduces a limited amount of randomization in the search procedure and show that heavy-tailed behavior can be observed in multiple runs of the randomized search procedure on a single problem instance.
3. We demonstrate how different notions of constrainedness that are commonly used in the modeling of planning problems can lead to a fat- or heavy-tailed distribution of search effort.
4. We introduce the notion of local minimum h-depth and find an exponential correlation between the h-depth of the single deepest local minimum encountered and the total search effort (i.e., number of expanded nodes).
5. We show that the distribution of local minima h-depth in planning problems depends on the constrainedness of problems, and that heavy-tailed behavior appears when there is a low, but non-negligible, probability of encountering a deep local minimum during search.
6. We show that recent methods of non-greedy random exploration can help reduce the heavy-tailed behavior in a similar manner to randomized restarts in CSPs.
7. Inspired by combinatorial search, we propose *RR-GBFS*, a randomized restarting GBFS that outperforms GBFS by escaping deep local minima.

The work in this chapter is based on the publications [27, 28].

4.1.1 Organization

In Section 4.2, we present the background on heavy tails and their implications to the analysis of the runtime distribution of computational problems. In Section 4.3 we describe the analytical framework we use to study the heavy-tailed behavior in satisficing planning. Sections 4.4 and 4.5 present an empirical analysis of the heavy-tailed behavior in satisficing planning using GBFS and its connection to the distribution of local minima encountered in the search. Finally, in Section 4.6 we study the use of randomization in GBFS in order to reduce the heavy-tailed behavior and improve GBFS performance. In Section 4.7, we summarize the chapter.

4.2 Background

The study of full runtime distributions of algorithms over a problem set, rather than just the median or the mean, often provides useful information that can contribute to the design of better algorithms. Previous work found exceptionally hard instances in many kinds of computational problems (e.g., Gent and Walsh [52]), that were attributed to a fat- or heavy-tailed behavior on ensembles of random problems in the easy region of the phase transition [64]. This behavior does not appear in ensembles of highly constrained instances, for which the median search effort is very high and all instances are uniformly hard. However, as we relax the problems, we move to a statistical regime in which the median effort is low, and

the hardest instances are *exceptionally* hard. The runtime distribution in this regime is characterized by a fat- or heavy-tailed behavior, i.e., a slow decay of the tail of the survival function.

Later, Gomes et al. [62] showed that heavy-tailed behavior can also be found in the runtime distribution of a randomized search procedure on a *single* instance, suggesting that the *ehps* can be easily solved by minor changes in the search procedure such as randomization. This result has motivated much work on eliminating the heavy-tailed behavior using randomized restarts, portfolios, etc. [60].

4.2.1 Fat- and Heavy-Tailed Distributions

The tail of a distribution is the very large (small) values of the distribution that determine the shape of its right (left) side [179]. Fat- and heavy-tailed distributions have a long tail containing a considerable concentration of mass.

Fat-tailedness can be determined based on the *kurtosis* of the distribution defined as $\kappa = \frac{\mu_4}{\mu_2^2}$, where μ_2 and μ_4 are the second and fourth moments, respectively. Since moments are very sensitive to the most extreme points in the sample's tails and many heavy-tailed distributions do not even have asymptotically finite moments, we use the L-kurtosis measure. L-kurtosis, denoted as τ_4, is based on the L-moments [93] and can be thought of as a measure similar to kurtosis that gives less weight to the extreme tails of the distribution, and is less sensitive to small sampling biases (for a detailed discussion of L-moments and specifically L-kurtosis see Hosking [93]). τ_4 is in the range $(-\frac{1}{4}, 1)$ and the Normal distribution has $\tau_4 = 0.1226$. Distributions with higher value are called leptokurtic and are considered to be fat-tailed.

Heavy-tailed distributions are considered "heavier" than fat-tailed distributions, and all the moments of a heavy-tailed distribution are infinite above some order [65]. A random variable X with distribution function $F(x)$ is considered heavy-tailed if it has a Pareto-like decay of its tail above some threshold x_l [148], i.e., there exists some $x_l{>}0$, $c{>}0$, $\alpha{>}0$ such that

$$1 - F(x) = P[X > x] = cx^{-\alpha}, x > x_l.$$

An approximately linear behavior over several orders of magnitude in the log-log plot of $1 - F(x)$ (i.e., the survival function) is a clear sign of heavy-tailed behavior with the slope providing an estimate of the stability index α [65]. In addition to the visual check, we can estimate α using the Hill estimator [85] or by fitting a generalized Pareto distribution (GPD) model to the peaks over threshold (POT) using maximum likelihood [167]. If $1 < \alpha < 2$, X has infinite variance and if $0 < \alpha \leq 1$, X has both infinite mean and variance [62]. Due to the limitations of the Hill estimator (see Embrechts et al. [41]), we use the POT method.

To demonstrate the difference between a heavy and a non-heavy tailed behavior, we use the example from Gomes et al. [65]. Figure 4.1 shows the log-log plot of $1 - F(x)$ vs. x for a normally distributed random variable with a mean of 2 and two possible values for the standard deviation. It also shows a random variable that represents the number of steps it takes for a symmetric random walk on a line to return to its starting point. The Normal distributions exhibit a fast-decay behavior, while the random walk exhibits a clear heavy-tailed behavior indicated by the approximately linear behavior on the log-log plot. The L-Kurtosis of both Normal random variables is $\tau_4 \approx 0.12$, while the random walk has $\tau_4 \approx 0.98$ and $\alpha \approx 0.5$.

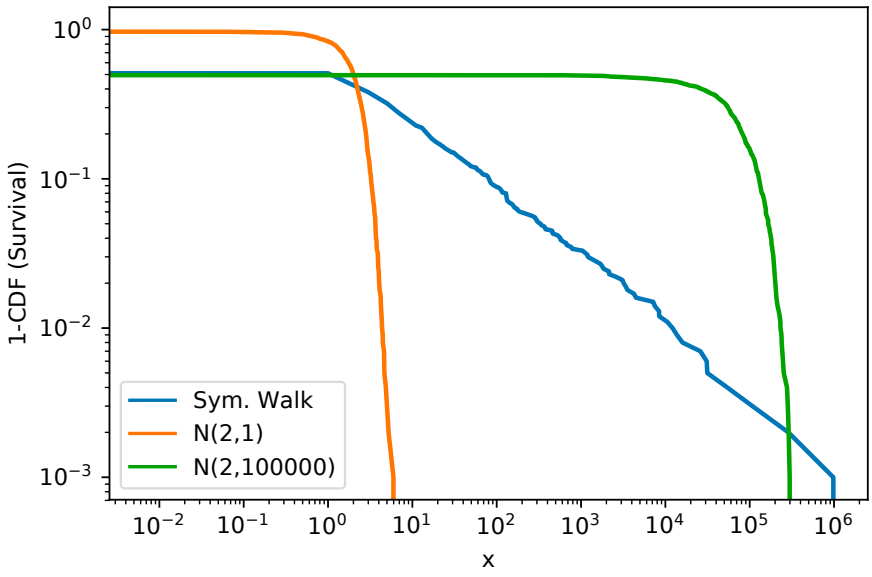


Figure 4.1: Heavy and non-heavy tailed behavior from Gomes et al. [65].

4.2.2 Randomized Search Algorithms

When analyzing the runtime distribution over an ensemble of problems, large variability (such as observed for *ehps*) can be either due to the variability among the instances in the ensemble or of the search algorithm itself. In order to isolate the latter, Gomes et al. [65] studied the runtime distribution of multiple runs of a randomized algorithm over the same instance. A fat- or heavy-tailed behavior in the runtime distribution means that there is a low (but non-negligible) probability of very long runs and suggests that using randomized restarts can dramatically reduce the variability and potentially eliminate the heavy tail.

Gomes et al. [65] proposed a method for adding randomization to complete, systematic, backtrack search algorithms such as DPLL. Traditionally, these algorithms construct a solution incrementally, and at each step a heuristic is used to decide how the partial solution will be extended (e.g., by assigning a value to a variable). Eventually, either a solution is found, or the algorithm backtracks to an earlier partial solution. If several decisions are deemed equally good, the algorithm typically applies some predefined rule to decide which decision to make. An easy way of introducing some randomization is to randomize the tie-breaking between decisions that are equally good, however, randomized tie-breaking might not be sufficient in many cases. Therefore, Gomes et al. [65] introduced H, a "heuristic equivalence" parameter, that expands the set of decisions deemed equally good. Given this modification, each run of the algorithm on a specific instance will differ in the order of decisions, and potentially in the runtime.

4.2.3 Constrainedness of Problems

Different works on combinatorial and heuristic search have used different notions of constrainedness. In our analysis of random heuristic search problems in Chapter 3, we used the density of edges in transition graph. In the analysis of random 3-SAT problems, Mitchell, Selman and Levesque [131] used clause-to-variable ratio. For random CSPs, Smith and Dyer used the tightness of the constraint graph

[164].

For structured benchmark domains, domain-specific parameters can be used. Examples of such parameters are the percent of pre-assigned colors in the quasigroup completion problem [65] or the probability of a blocked cell in Grid Navigation in Chapter 3.

Gent et al. [51] suggested a unified definition of constrainedness that is based on the expected number of feasible solutions, however it is not easy to calculate this number for a planning problem [25].

4.3 Analytical Framework

In this section we describe the analytical framework we use to investigate heavy-tailed behavior in planning. We first describe two notions of constrainedness of planning problems that we can manipulate in order to observe the different statistical regimes. Then, we propose a method to randomize a deterministic planning algorithm in order to analyze the runtime distribution on a single problem instance. Finally, we describe the benchmark problems we use.

4.3.1 Constrainedness of Planning Problems

A systematic study of fat- and heavy-tailed behaviors in satisficing planning involves analyzing the runtime distribution of instances of different constrainedness. In planning problems, there are various parameters that can effect the constrainedness of the problem (i.e., the expected number of feasible solutions). We describe two types of parameters that are often used to model planning problems.

Resource Constrainedness

When planning with consumable resource, i.e., resources that cannot be replenished, resource constrainedness is the amount by which the initial resource supply exceeds the minimum needed [87, 57]. The resource constrainedness can be measured by a parameter $C \geq 1$, namely the maximum number by which we can divide the resource supply without rendering the task unsolvable [134]. Nakhost et al. [134] studied the performance of state-of-the-art domain-independent planners as a function of C. To do so, they introduced an extended benchmark suite with three benchmark domains: *NoMystery, TPP, Rovers*.

Goal Constrainedness

The definition of a goal condition in a planning problem can also be used to control the constrainedness of a problem. For a goal condition g_i, we use G_i to denote the induced set of goal states (i.e., states that satisfy the goal condition). We consider g_j to be a relaxation of g_i if $G_i \subseteq G_j$, meaning that every state that satisfies g_i will satisfy g_j. For example, if g_i is the conjuction of a set of propositions P and g_j is the conjuction of a set of propositions P' such that $P' \subseteq P$, we consider g_j to be a relaxation of g_i. There are, however, more nuanced relaxations that are not based on dropping goal propositions and we provide an example in our benchmarks.

4.3.2 Benchmark Problems

To study the effect of resource constrainedness we use the benchmark domains used in Nakhost et al. [134]: NoMystery, Rovers, and TPP. To study the effect of goal constrainedness we use the domains Maintenance, Parking, and Freecell. Following is a brief description of each domain.

NoMystery

NoMystery is based on the domain used in IPC'11. The goal is to transport packages between locations, using a set of trucks with a limited amount of fuel.

Rovers

Rovers is based on the domain from IPC'02, without the "recharge" operator (to make the resource consumable). The goal in this domain is to take a number of rock samples and images and transfer them to a lander.

TPP

TPP was used in IPC'06. An agent is required to buy a set of items that are being sold at different prices in different markets. The limited resource is money, which is required for buying the items and for driving between markets.

Maintenance

Maintenance was used in IPC'14. Mechanics work each day at one airport. Each plane visits some airports on given days. The goal is to schedule the mechanics' visits to the airports such that each plane will be maintained once.

Parking

Parking (IPC'11, IPC'14) involves parking cars at N curbs where cars can be doubled-parked but not tripled-parked. The problem is to find a plan that moves from one configuration of cars to another by moving cars between curbs.

Freecell

Freecell (IPC'00, IPC'02) involves moving cards from initial configuration to a suit-sorted collection of stacks using the available free cells.

4.3.3 Randomized Heuristic Search

In order to analyze the runtime distribution on a single instance, we need to introduce a limited amount of randomization into the search. In greedy best-first search (GBFS), the heuristic function determines the order of expansions. However, most commonly used heuristics are deterministic and so re-running the same instance will yield the same h-values for all states (notable exceptions are heuristics that are based on a random sample of the state space, e.g., Haslum et al. [75]). Although random tie-breaking can provide some randomness, in many domains it may not be sufficient.

We therefore present a general, parameterized method to randomize a heuristic function, and use it in our empirical analysis. Given a heuristic function $h(x)$ and a parameter $p \geq 0$, that represents the extent of randomization, we consider $h_{\Delta p}(x)$ to be an p-randomized version of h if for all states s: $h_{\Delta p}(s) = h(s) + \Delta_p^h$ where Δ_p^h is a random number in the range $[-p{\cdot}h(s), p{\cdot}h(s)]$. The heuristic values are rounded to maintain integer values. When used with deferred heuristic evaluation [76], we randomize the order in which the successors of a node are generated.

This method is similar to the one proposed by Gomes et al. [65] for backtracking search, as it randomly orders nodes that are within $100{\cdot}p$ percent of their h-values, and does not effect the completeness of the search algorithm.

4.4 Empirical Analysis of Heavy-tailed Behavior in Satisficing Planning

In this section we present an empirical analysis of the runtime distribution of the benchmark problems for different levels of constrainedness. We use GBFS and configure the planner not to re-open nodes. We run experiments for several commonly-used heuristic functions: the FF heuristic [88] (denoted as h^{FF}), the landmark count heuristic [150], the landmark cut heuristic [78], and the context-enhanced additive (CEA) heuristic [79]. In our experiments, we use deferred heuristic evaluation [149], however we demonstrate similar results for standard heuristic evaluation in Appendix A.

4.4.1 Resource Constrainedness

NoMystery

We consider NoMystery problem instances with six locations and six packages. The resource constrainedness parameter (C) controls the amount of available fuel. Figure 4.2 shows the search effort distribution for 1000 random instances for different values of C, using h^{FF}. For C=1, the tail decays much faster than the more relaxed problems. For C=2 we see a clear heavy-tailed behavior (a near-linear behavior over a several orders of magnitude). Although the problems in C=2 are, on average, much easier, the hardest instances are significantly harder than the median, compared to C=1.

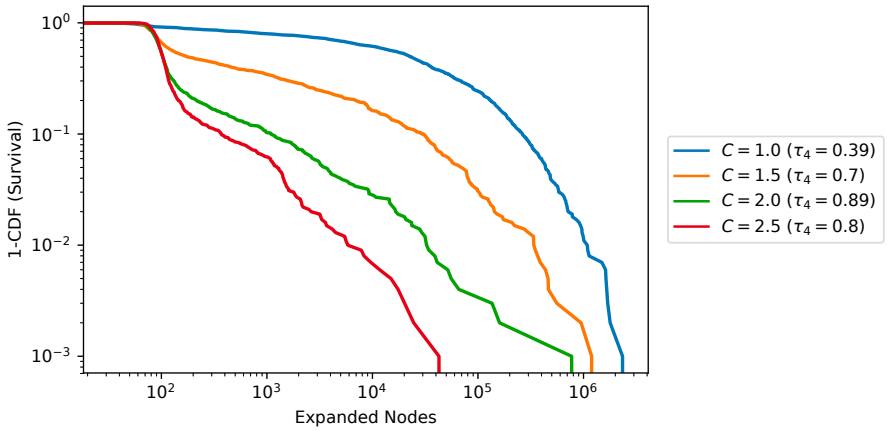


Figure 4.2: NoMystery: 1000 random instances with h^{FF}.

More interesting is the search effort distribution of a randomized search algorithm on one instance. We used the median instance from the ensemble C=1, and created relaxed instances by increasing C. Figure 4.3 shows the search effort distribution of 1000 runs of a randomized search with $h^{FF}_{\Delta 0.1}$ on the same instance for different values of C. The results clearly show a transition from a statistical regime in which the problem is very constrained and the tail decays quickly (C=1) to a regime in which the

problem is more relaxed and the tail decays much more slowly. For C=3.0, we see a clear heavy-tailed behavior with extremely long runs that are even longer than the longest run for C=1. The τ_4 values support these observations as C=1 has a lower τ_4 than the Normal distribution indicating a very thin tail. As we increase C the τ_4 value increases, reaching 0.9 for C=3.5. As we relax the problem further we start see a decline in the heavy-tailed behavior and the corresponding smaller τ_4 value.

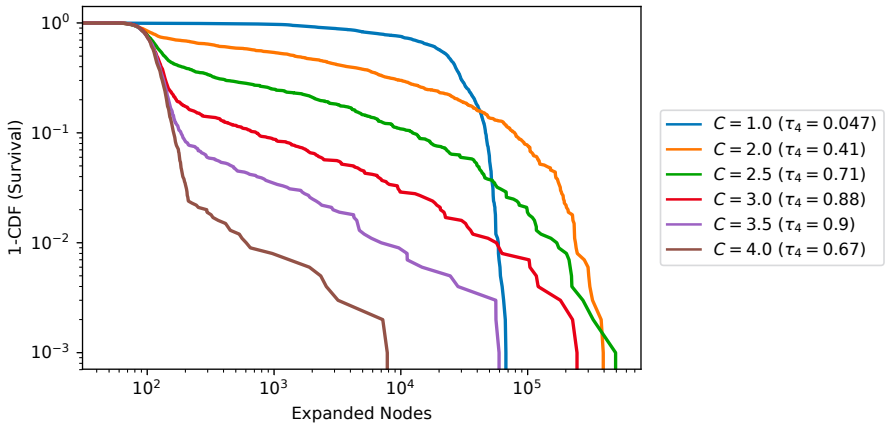


Figure 4.3: NoMystery: 1000 randomized runs on a single instance with h^{FF}.

To estimate the stability index α of the tail, we fit a GPD model to the peaks over threshold using the maximum likelihood method [17]. For C=3.0 we used a tail that corresponds to the largest 10% samples (chosen based on a visual inspection, see Figure 4.3). We estimated $\alpha \approx 0.69$, and the quality of the fit is presented in Figure 4.4. The estimated value suggests that the underlying distribution has an infinite mean and variance ($0 \leq \alpha \leq 1$).

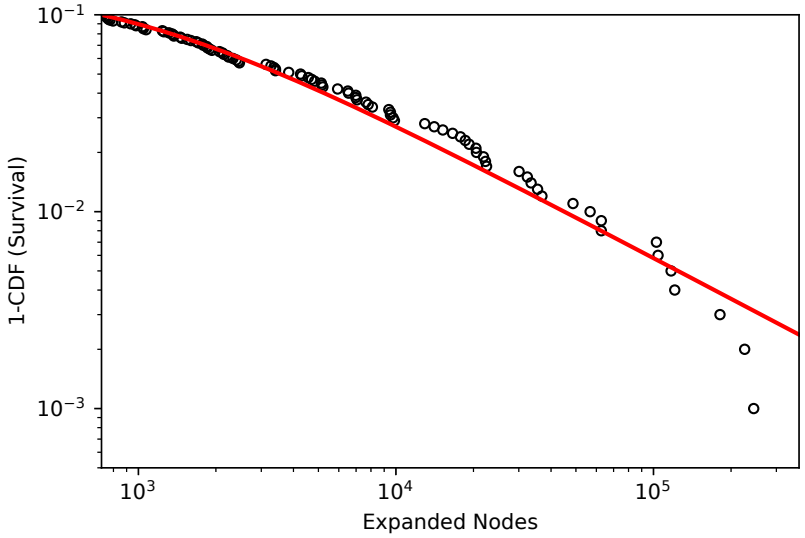
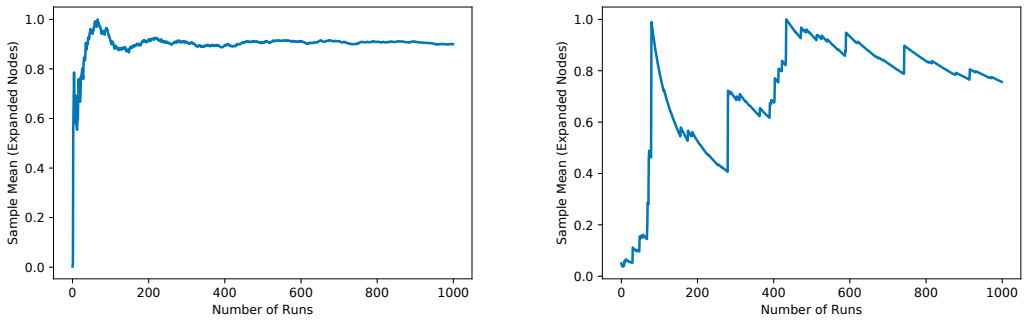


Figure 4.4: NoMystery: Fitting a GPD models with $\alpha = 0.69$ to the tail of $C = 3.0$.

Figure 4.5 shows the mean effort (normalized to a 0-1 scale) over an increasing number of runs for

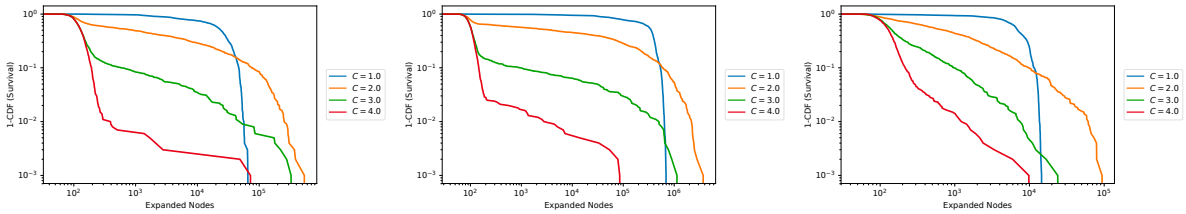
different values of C. When C=1, the mean stabilizes after a small number of samples (similar to the normal distribution). As we increase C, the mean takes longer to stabilize and still exhibits a large variance. When we move to the heavy-tailed regime, the mean does not stabilize with increasing sample size. This pattern is consistent with the erratic behavior of the mean, observed by Gomes et al. [65] in CSPs. As we increase C further, problems gradually become universally easy and the mean starts to stabilize again (not presented).



(a) Non-heavy-tailed regime ($C = 1$). (b) Heavy-tailed regime ($C = 3$).

Figure 4.5: NoMystery: Sample Mean.

To demonstrate that this phenomenon is not unique to h^{FF}, Figure 4.6 shows the runtime distribution for 1000 runs of a randomized search using the landmark count heuristic [150], the landmark cut heuristic [78], and the context-enhanced additive (CEA) heuristic [79], all with p=0.05. While the exact performance differs between the heuristics, they all exhibit fat- or heavy-tailed behavior as the problem constrainedness is relaxed.

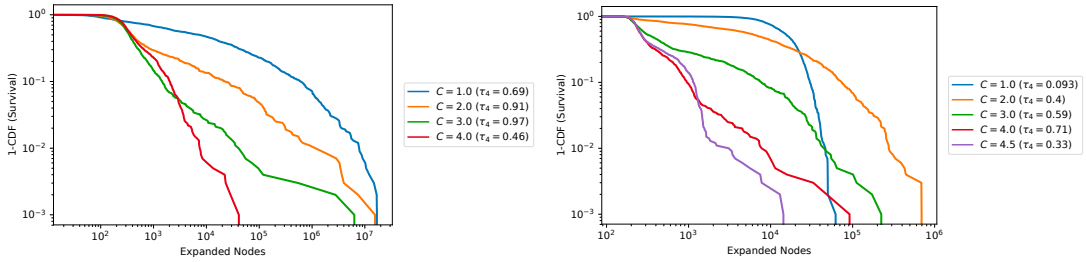


(a) Landmark cut heuristic. (b) Landmark count heuristic. (c) CEA heuristic.

Figure 4.6: NoMystery: Results for other heuristics.

Rovers

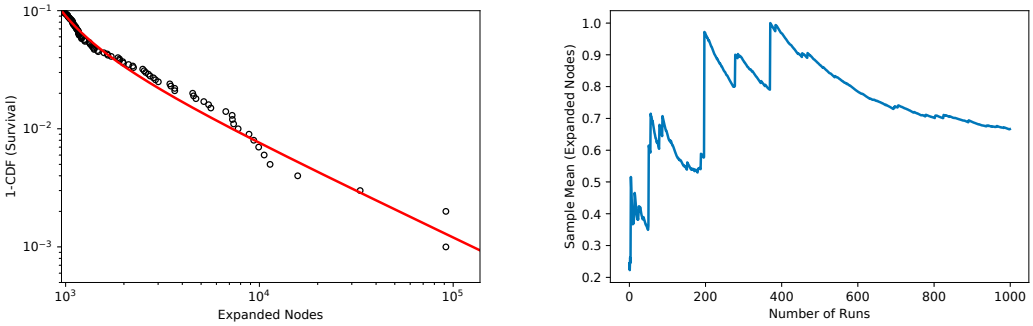
We consider Rover problem instances with a single rover, six waypoints, and six objectives. The constrainedness parameter C controls the energy level of the rover. Figure 4.7a and Figure 4.7b show the runtime distribution for 1000 random problems and 1000 randomized runs on a single problem, respectively. We observe a fat- and heavy-tailed behavior for relaxed problems associated with a higher τ_4. For Figure 4.7b, we estimated $\alpha \approx 0.78$ for for $C = 4.0$ (Figure 4.8a) and observe the associated erratic behavior of the mean (Figure 4.8b).



(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure 4.7: Results for Rovers.

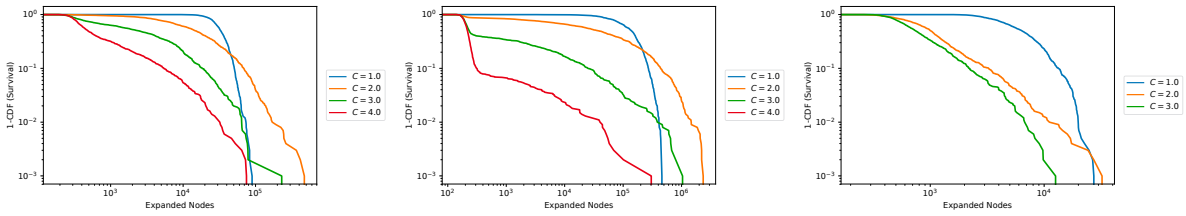


(a) Fitting a GPD models with $\alpha = 0.78$ to a tail that corresponds to the largest 10% samples.

(b) Erratic behavior of sample mean.

Figure 4.8: Rovers: Results for $C = 4.0$.

Figure 4.9a, Figure 4.9b, and Figure 4.9c show analyses similar to Figure 4.7b using the landmark cut heuristic, the landmark count heuristic, and the CEA heuristic, respectively. For the landmark cut and landmark count heuristics we used $p = 0.05$, similar to FF. For the CEA heuristic, we used $p = 0.15$ since we empirically found it required higher p to exhibit heavy tailed behavior.



(a) Landmark cut heuristic.

(b) Landmark count heuristic.

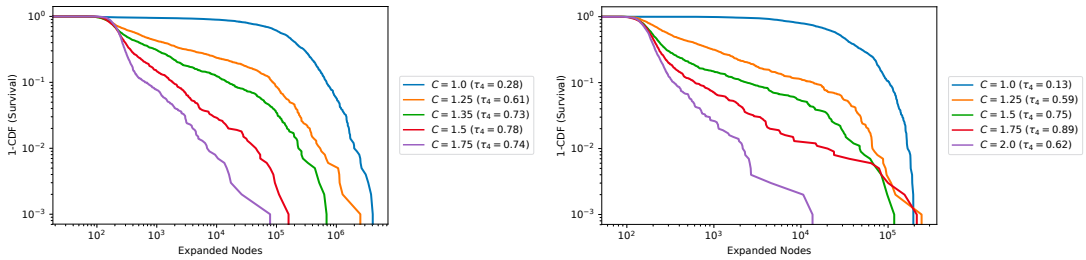
(c) CEA heuristic.

Figure 4.9: Rovers: Results for other heuristics.

TPP

We consider TPP problems with seven markets and five goods, each sold in two markets, where the constrainedness parameter (C) controls the available money. Figure 4.10a and Figure 4.10b show the runtime distribution for 1000 random problems and 1000 randomized runs on a single problem, respectively.

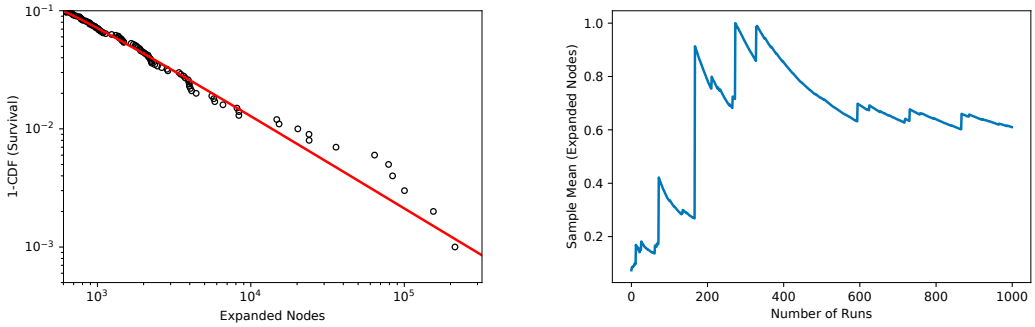
In Figure 4.10a, we observe a fat-tailed behavior for relaxed problems associated with a higher τ_4. In Figure 4.10b, we observe a heavy-tailed behavior for $C = 1.75$ with an estimated $\alpha \approx 0.78$ (Figure 4.11a) and the associated erratic behavior of the mean (Figure 4.11b).



(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure 4.10: Results for TPP.

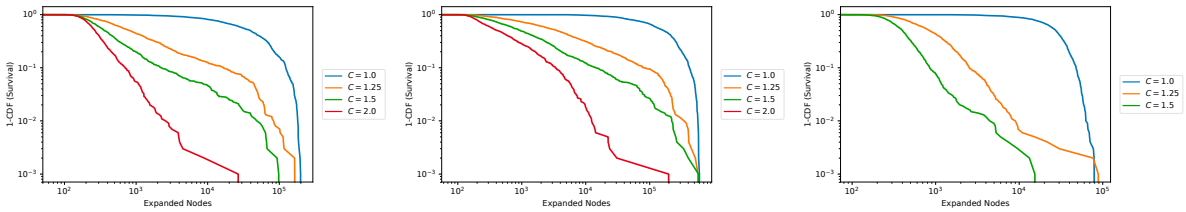


(a) Fitting a GPD models with $\alpha = 0.78$ to a tail that corresponds to the largest 10% samples.

(b) Erratic behavior of sample mean.

Figure 4.11: TPP: Results for $C = 1.75$.

Figure 4.12a, Figure 4.12b, and Figure 4.12c show analyses similar to Figure 4.10b using the landmark cut heuristic, the landmark count heuristic, and the CEA heuristic, respectively.



(a) Landmark cut heuristic.

(b) Landmark count heuristic.

(c) CEA heuristic.

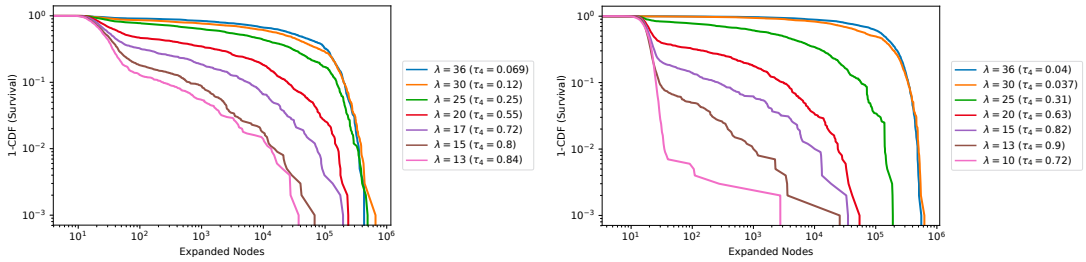
Figure 4.12: TPP: Results for other heuristics.

4.4.2 Goal Constrainedness

Maintenance

We analyze the effect of the goal constrainedness by relaxing the goal of having all n airplanes checked by a mechanic. We start by using a simple, however a bit artificial, relaxation of the goal by requiring that only a chosen subset of λ airplanes be checked ($|\lambda| \leq n$). We consider a problem with 12 days and 36 planes, and analyze the runtime distribution for different λ sets. Figure 4.13a shows the runtime distribution of 1000 random instances and exhibits fat-tailed behavior with significantly increasing τ_4.

Figure 4.13b shows the distribution for a randomized search with $h^{FF}_{\Delta 0.05}$ on one instance. Again, we used the median instance of the more constrained ensemble ($|\lambda|$=36), however this time we had to manually create relaxed variants by choosing a random subset of planes (we repeated the process several times and observed similar patterns). As we relax the problem we observe a fat tailed behavior with τ_4 increasing from approximately 0.07 to 0.84.

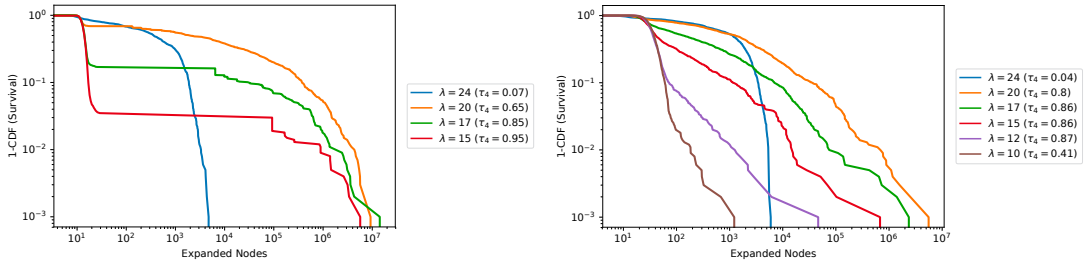


(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure 4.13: Results for Maintenance.

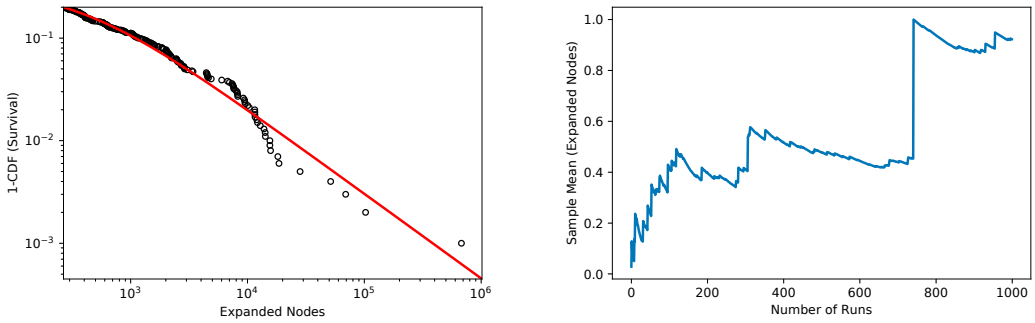
We also analyze an alternative relaxation, of a more combinatorial nature: at least λ airplanes will be checked, but we do not decide which ones. Figure 4.14 shows the results for problems with 8 days and 24 planes. Again, we see a fat- and heavy-tail behavior, with Figure 4.14a showing clear heavy-tailed behavior for relaxed instances. For $C = 15$, we estimate $\alpha \approx 0.83$ (Figure 4.15a) and observe erratic behavior of the mean (Figure 4.15b).



(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure 4.14: Results for Alternative Relaxation of Maintenance.

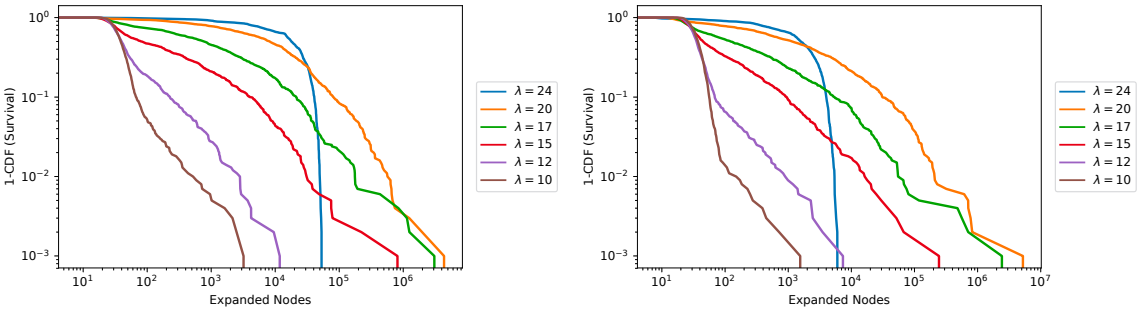


(a) Fitting a GPD models with $\alpha = 0.83$ to the tail of $C = 15$.

(b) Erratic behavior of sample mean for $C = 15$.

Figure 4.15: Results for Alternative Relaxation of Maintenance.

Figure 4.16 shows similar results to Figure 4.14b using the landmark count heuristic and the CEA heuristic. Landmark cut could not be computed due to the use of axioms in the Maintenance domain.



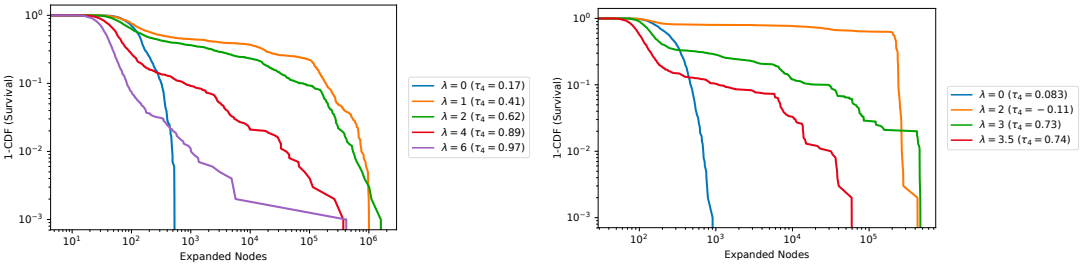
(a) Landmark count heuristic.

(b) CEA heuristic.

Figure 4.16: Maintenance (alternative): 1000 randomized runs on a single instance.

Parking

We consider Parking problem instances with 8 cars and 5 curbs and analyze the effect of the goal constrainedness by allowing each car to park in more than one location. The goal constrainedness parameter λ controls the additional goal locations for each car. When $\lambda = 0$ cars are only allowed to park in the original goal configuration and when $\lambda > 0$, we randomly add λ goal locations for each car. Figure 4.17a and Figure 4.17b show the distribution for 1000 random problems and 1000 randomized runs on a single instance ($p = 0.1$), respectively. We observe an interesting pattern: as we relax problems, we first observe a significant increase in the problem difficulty of the majority of problems followed by the more familiar reduction in difficulty. This pattern can potentially be due to less efficient pruning by the FF heuristic in the presence of more than one goal location. Still, as we relaxed the problem further we see the transition to a clear fat-tailed behavior (indicated by the τ_4 values) and the majority of the problems are solved faster, even compared to $\lambda = 0$.

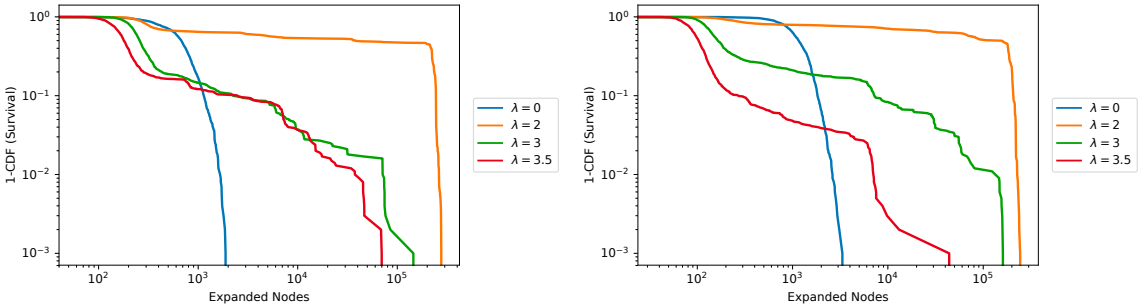


(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure 4.17: Results for Parking.

Figure 4.18 shows similar results to Figure 4.17b using the landmark cut heuristic and the CEA heuristic. We found the landmark count heuristic to be not sufficiently informative to solve this problem efficiently.



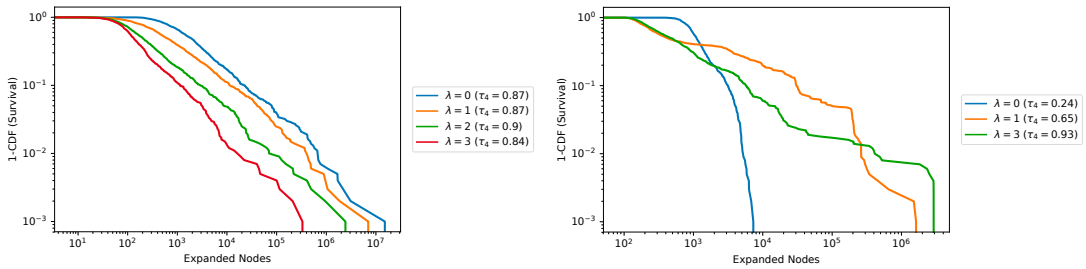
(a) Landmark cut heuristic.

(b) CEA heuristic.

Figure 4.18: Parking: 1000 randomized runs on a single instance.

Freecell

In Freecell, each card with face value i has to placed on top of a card with face value $i-1$ in each stack. We define a parameter λ that controls the number of cards that each card can be placed on top of, in each stack and analyze instances with four suits of cards, each containing 10 cards. Figure 4.19a and Figure 4.19b show the distribution for 1000 random problems and 1000 randomized runs on a single instance, respectively. Note that in Figure 4.19a, we see a clear heavy-tailed behavior for all levels of constrainedness, including for $\lambda = 1$. Heavy-tailed behavior even in $\lambda = 1$ could suggest that either these instances are relaxed enough to exhibit heavy-tailed behavior or that there is a large variance in the problem set.



(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure 4.19: Results for Freecell.

We found the landmark cut and landmark count heuristic not sufficiently informative, even in highly relaxed instances. However, for the CEA heuristic, Figure 4.20 shows a similar heavy-tailed behavior to the one observed for the FF heuristic. Note that for $\lambda = 1$, there was additional instance that could not be solved in a time limit of 60 minutes.

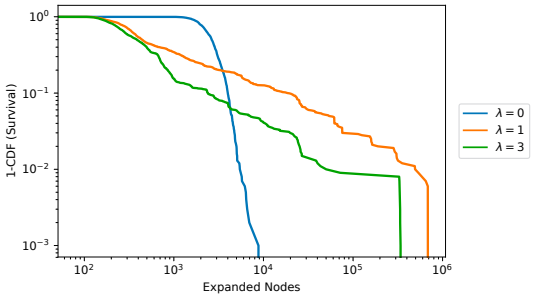


Figure 4.20: 1000 randomized runs on a single instance with the CEA heuristic.

4.4.3 Discussion

The empirical results suggest that a heavy-tailed behavior can be observed for different planning problems, using different heuristic functions. These results provide a deeper understanding of the empirical difficulty of planning problems and account for the existence of exceptionally hard problems (*ehps*) found in Chapter 3.

In our analysis, we choose to randomize GBFS by introducing uniform random noise to the heuristic function. Our empirical results show that this randomization leads GBFS to exhibit a heavy-tailed behavior on a single instance. Investigating the impact of alternative sources of random noise (e.g., non-uniform distributions) or alternative techniques for randomizing GBFS in addition to randomizing the heuristic function are directions for future work.

Heavy-tailed behavior has been shown not to be inherent to backtracking search in general, but rather to depend on the efficiency of the search procedure as well as on the level of constrainedness of the problem [62]. In the context of planning, uninformative heuristics will not exhibit a heavy-tailed behavior, even as we relax the problems, as the median effort for such problem is likely to remain high (see, for example, our analysis of uninformed heuristics in Chapter 3). We observed this behavior in some of our experiments above, e.g., for the landmark-based heuristics in the Freecell domain.

However, similar to combinatorial search, we find that heavy-tailed behavior can be observed for many problems using common heuristics. Specifically, resource constrained problems exhibit such behavior when solved with a heuristic that is based on ignoring the delete effects. For highly constrained problems, most paths do not lead to a solution due to lack of resources. As we relax the problems, we can usually find a solution easily, however one mistake can still lead to the need to exhaust a region of the state space that has no solution, resulting in an extremely long run. We also show that heavy-tailed behavior can be observed as we relax highly constrained problems by introducing more goal states. In the relatively goal-relaxed problems, while most paths will lead to a goal state, one heuristic mistake can lead the search into a region with no solution, e.g., when achieving one goal proposition has a delete effect (that the heuristic does not account for) that prevents us from achieving another.

In combinatorial search, heavy-tailed behavior has been shown to be correlated with an exponential distribution of depths of the subtrees with no solutions [62]. In satisficing planning, local minima (i.e., regions of the state space that have no solution [197]) have been shown to have a negative effect on the search effort (e.g., in Xie et al. [204]). In the next section, we investigate the connection between the observed heavy-tailed behavior and the distribution of the sizes of local minima.

4.5 Empirical Analysis of Local Minima in Satisficing Planning

In CSPs, the runtime distribution was shown to be highly correlated with the distribution of the depth of inconsistent subtrees (i.e., subtrees with no solution) discovered during the search [62]. Specifically, the observed heavy-tailed behavior was shown to be the result of an exponential distribution of the depth of inconsistent subtrees.

In this section, we adapt this analysis to GBFS, inspired by the similarity between inconsistent subtrees and local minima. In Section 4.5.1, we define local minima in GBFS and a notion of depth of a local minimum that is based on the heuristic values that guide the search. Then, in Section 4.5.2, we empirically show that the depth of a local minimum is exponentially correlated to search effort, similar to inconsistent subtrees in CSPs. Finally, in Section 4.5.3, we establish the connection between the distribution of depths of local minima and the heavy-tailed behavior we observe in Section 4.4.

4.5.1 Local Minima in Satisficing Planning

In Section 2.2, we noted that a STRIPS planning problem $\mathcal{P}=\langle \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ defines a planning state space $\langle S, s_I, S_G, A, f, c \rangle$. We use the state space to formally define a set of terms used in our analysis of local minima in satisficing planning. Informally, we consider a local minimum to be a set of states that were expanded between two consecutive expanded states on the solution path found by a search algorithm (we assume nodes are not re-opened).

Definition 16 (Solution vector) *Let $\mathcal{P} = \langle s_1, ..., s_P \rangle$ be a vector of states in S. We consider $\mathcal{P}$ to be a (feasible) solution vector if $s_1 = S_I$, $s_P \in S_G$ and for each consecutive pair of states s_i, s_j in $\mathcal{P}$ there exists action a such that $f(s_i, a) = s_j$.*

Definition 17 (Expansion vector) *Let $\mathcal{E} = \langle \bar{s}_1, ..., \bar{s}_E \rangle$ be a vector of states in S. We consider $\mathcal{E}$ to be a (feasible) expansion vector if $\bar{s}_1 = S_I$, $\bar{s}_E \in S_G$ and for every state $\bar{s}_x \in \mathcal{E}, x \geq 2$ there exists state $\bar{s}_y \in \mathcal{E}, y < x$ and action $a \in A$ such that $f(\bar{s}_y, a) = \bar{s}_x$.*

Definition 18 (Local Minimum) *Let $\mathcal{E}$ be an expansion vector and $\mathcal{P}$ be a solution vector, such that $\mathcal{P}$ is a potentially non-contiguous subsequence of $\mathcal{E}$. For every consecutive pair s_i, s_j in $\mathcal{P}$ we define the local minimum $\mathcal{L}_{s_i,s_j}$ to be the sub-vector of all states between s_i and s_j in the vector $\mathcal{E}$ (including s_i, s_j).*

Definition 19 (h-depth) *Let s_i, s_j be two consecutive states in a solution vector and let $\mathcal{L}_{s_i,s_j} = \langle s_i, ..., s_j \rangle$, a vector of states, be the local minimum between s_i and s_j. We define the h-depth of the local minimum $d^h_{s_i,s_j} = h(s_j) - min_h(\mathcal{L}_{s_i,s_j})$.*

Definition 20 (h-backtrack) *Let s_i, s_{i+1} be two consecutive states in an expansion vector. We define the expansion of s_{i+1} to be an h-backtrack if $h(s_{i+1}) > h(s_i)$.*

Our definition of a local minimum differs from Wilt and Ruml's [197] (see Section 2.3.1) as it is based on a the result of a specific GBFS run and not on the whole state space. As we are studying the distribution of search efforts across multiple runs, we are interested in modelling the local minima that were actually encountered in each run, rather then the whole state space. Analyzing only the local minima that were encountered during the search is consistent with the methodology applied to CSPs [62]. Our definition of local minima in GBFS is therefore a natural counterpart of Gomes et al.'s [62] inconsistent subtrees in CSPs.

Note that our definition of local minimum h-depth closely matches the h difference in Wilt and Ruml's [194] Observation 1 and the definition of h-backtrack is closely related to the non-monotonicity of h-values in Observation 3.

4.5.2 Problem Difficulty and Local Minima

In this section we establish the connection between the local minima encountered in the search and the problem difficulty. We use the same problem sets used in the analysis of the heavy-tailed behavior in Section 4.4. Each time we solve a problem, we record the expansion vector and the solution vector. We use these vectors to extract the local minima encountered in the search and, for each local minimum, we compute its size (i.e., the number of nodes), its h-depth, and the number of h-backtracks that occurred in that local minimum.

We start by analyzing the connection between the h-depth and the size (i.e., the number of expanded nodes) of a local minimum in the NoMystery domain. Figure 4.21 shows the distribution of local minima size vs. h-depth in an ensemble of 1000 random problems, solved using h^{FF}, in both the non-heavy-tailed regime ($C = 1$) and the heavy-tailed regime ($C = 2$). We can clearly see an exponential correlation in both cases between the h-depth of a local minimum and the associated search effort, measured by the number of expanded nodes. This result is consistent with inconsistent subtrees in CSPs [62], and suggests that, similar to inconsistent subtrees, the depth of local minima plays an important role in the difficulty of problems.

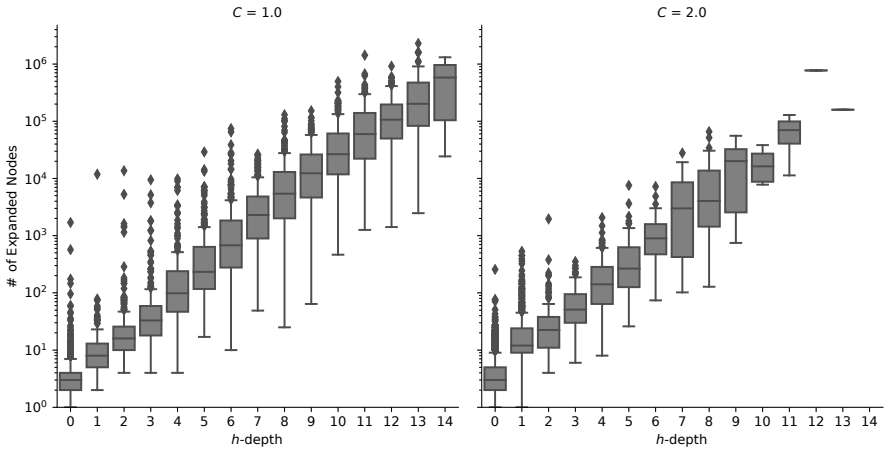


Figure 4.21: NoMystery: local minima size vs. h-depth in 1000 random instances with h^{FF}.

Figure 4.22 shows similar results for 1000 randomized runs with a randomized search procedure on one instance, solved using $h^{FF}_{\Delta 0.1}$. While the absolute numbers change between different levels of constrainedness, the qualitative trends remains similar for different C values.

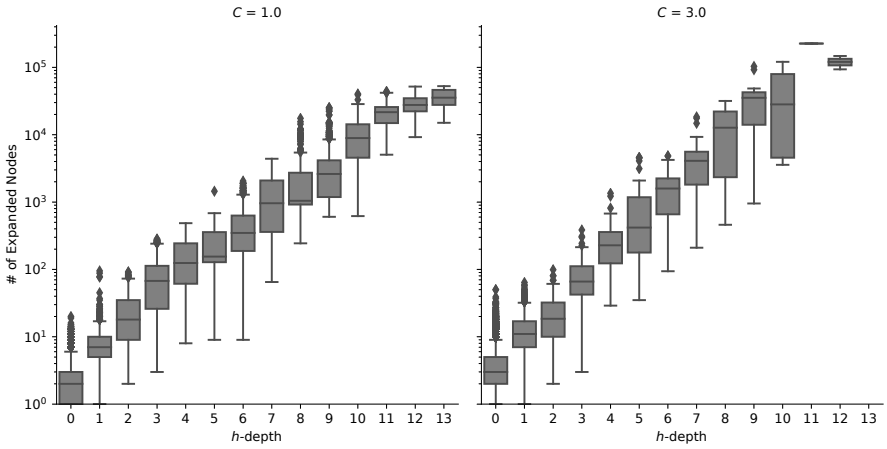
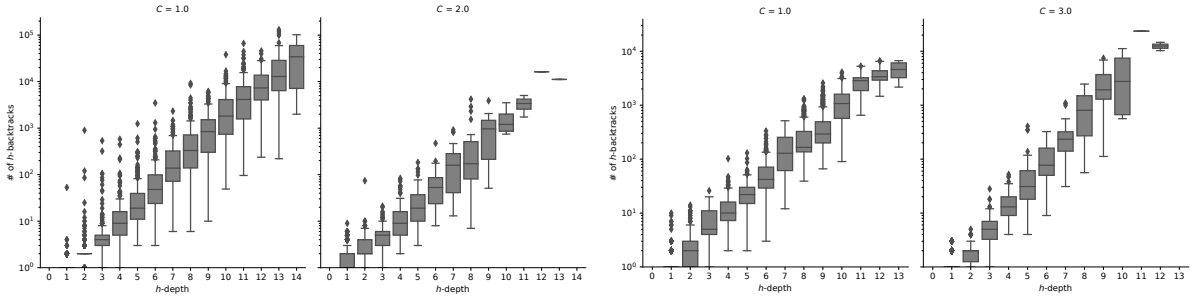


Figure 4.22: NoMystery: local minima size vs. h-depth in 1000 randomized searches on a single instance with h^{FF}.

Figure 4.23a and Figure 4.23b show the distribution of the number of h-backtracks vs. h-depth in an ensemble of 1000 random problems and in 1000 randomized searches on a single problem, respectively, in both the non-heavy-tailed and the heavy-tailed regimes. Again, we find an exponential correlation suggesting that the observed effort is due to a backtracking behavior based on h-values. The fact that we would observe such a correlation for GBFS is not a priori obvious since it is not defined as a backtracking search and is not guaranteed to exhibit such behavior. However, our results suggest that in many cases GBFS tends to exhibit a backtracking behavior, based on h-values, similar to tree search algorithms in CSPs/SAT.

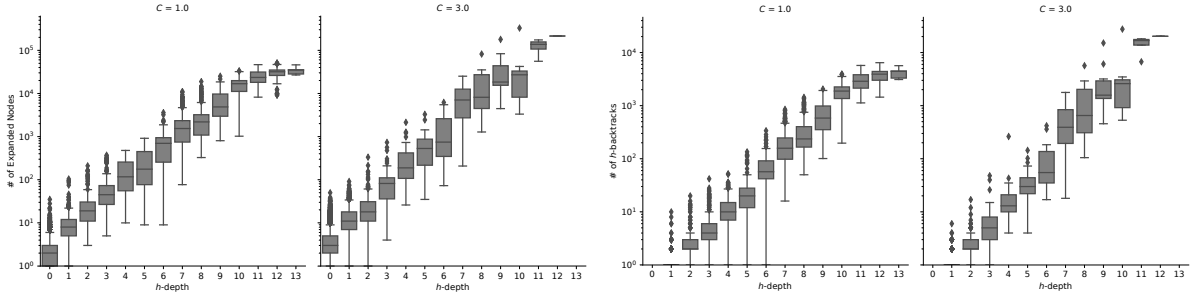


(a) 1000 random instances with h^{FF}.

(b) 1000 randomized searches on a single instance with h^{FF}.

Figure 4.23: NoMystery: number of h-backtracks vs. h-depth.

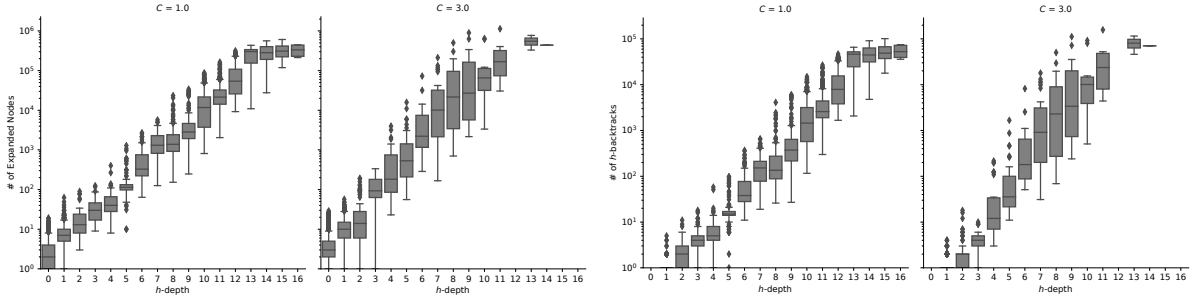
Figure 4.24, Figure 4.25, and Figure 4.26 show the local minima size and number of h-backtracks vs. h-depth for the landmark cut heuristic, the landmark count heuristic and the CEA heuristic, respectively. We observe similar trends to those observed for the FF heuristic.



(a) Local minima size vs. h-depth.

(b) Number of h-backtracks vs. h-depth.

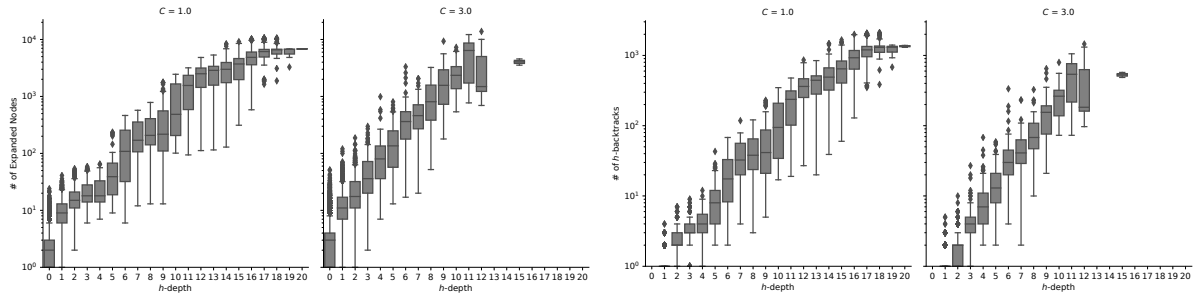
Figure 4.24: NoMystery: results for 1000 randomized runs on a single instance with the landmark cut heuristic.



(a) Local minima size vs. h-depth.

(b) Number of h-backtracks vs. h-depth.

Figure 4.25: NoMystery: results for 1000 randomized runs on a single instance with the landmark count heuristic.

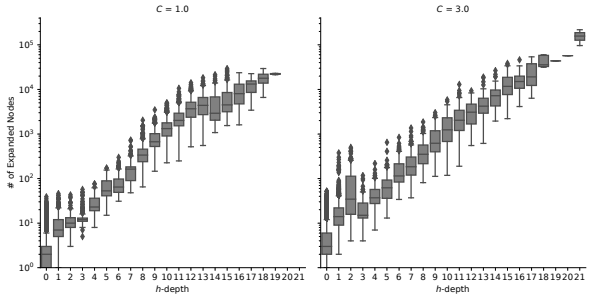


(a) Local minima size vs. h-depth.

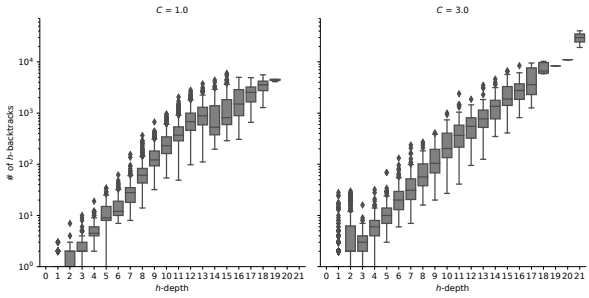
(b) Number of h-backtracks vs. h-depth.

Figure 4.26: NoMystery: results for 1000 randomized runs on a single instance with the CEA heuristic.

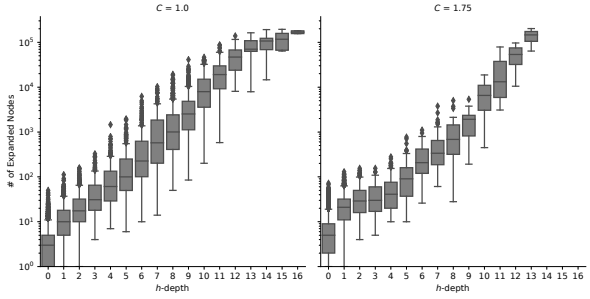
Figure 4.27 shows the distribution of local minima sizes and the number of h-backtracks vs. h-depth for the rest of the domains.



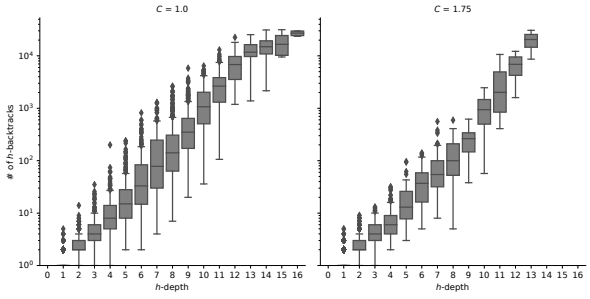
(a) Rovers: local minima size vs. h-depth.



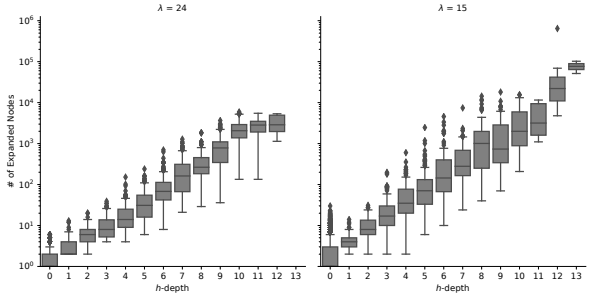
(b) Rovers: number of h-backtracks vs. h-depth.



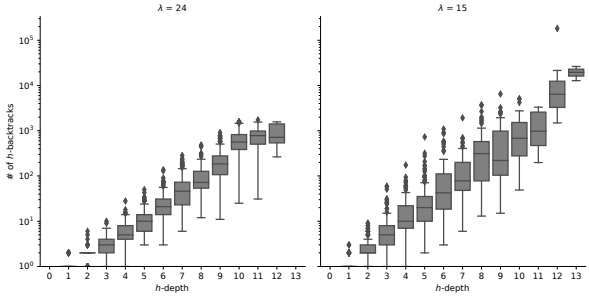
(c) TPP: local minima size vs. h-depth.



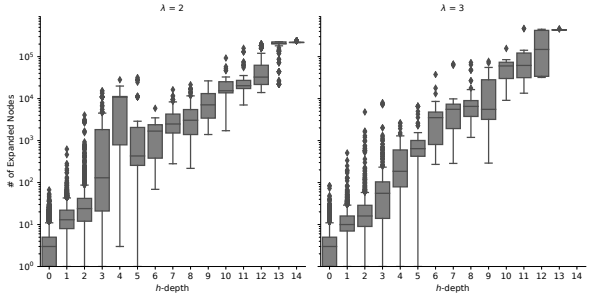
(d) TPP: number of h-backtracks vs. h-depth.



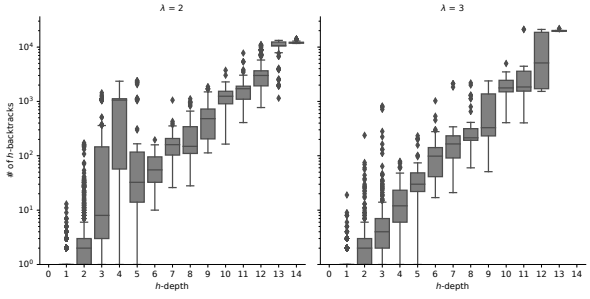
(e) Maintenance (alternative): local minima size vs. h-depth.



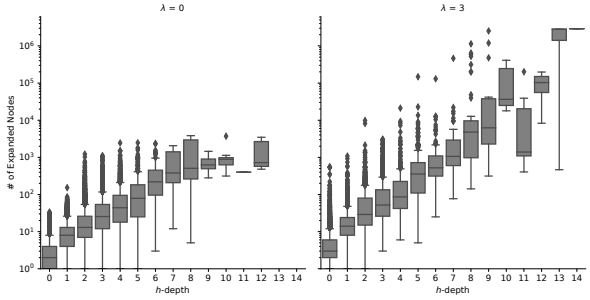
(f) Maintenance (alternative): number of h-backtracks vs. h-depth.



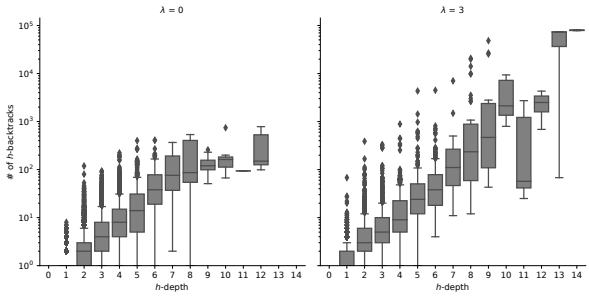
(g) Parking: local minima size vs. h-depth.



(h) Parking: number of h-backtracks vs. h-depth.



(i) Freecell: local minima size vs. h-depth.



(j) Freecell: number of h-backtracks vs. h-depth.

Figure 4.27: Results for 1000 randomized runs on a single instance with h^{FF}.

Surprisingly, we also find a strong correlation between the h-depth of the *single* deepest local minimum encountered in an instance and the *total* search effort for that instance. Table 4.1 reports this correlation for all the six benchmark domains, in both the heavy-tailed and the non-heavy-tailed regimes and over all problem instances (of mixed constrainedness values). The reported values are the Pearson correlation coefficient between the h-depth of the deepest local minimum and the log(search effort), with search effort measured by the number of expanded nodes. We use log since Pearson measures linear correlation, and linear correlation between x and $log(y)$ indicates an exponential correlation between x and y. Since the number of backtracks can be zero (when h-depth is zero), we augment the number of backtracks by adding one before computing the logarithm. To make sure we are not biased by a population that is centered on a small range of h-values we also calculated the weighted Pearson correlation (all the instances of a given h-depth sum up to the same weight). Table 4.2 shows that such results have an even higher correlation than the non-weighted data. These results indicate a strong exponential correlation and suggest that the deepest local minimum encountered in a search is an important factor in determining the total search effort. Given the complexity of heuristic search and the existence of multiple local minima, this type of strong correlation is not a priori obvious and its discovery is, to our knowledge, novel.

	log(search effort)		log(h-backtracks)	
Domain	N-HT	HT	N-HT	HT
NoMystery (1000)	0.91	0.91	0.93	0.95
NoMystery (one)	0.87	0.93	0.90	0.95
Rovers (1000)	0.96	0.77	0.98	0.94
Rovers (one)	0.70	0.93	0.69	0.96
TPP (1000)	0.85	0.88	0.87	0.96
TPP (one)	0.79	0.81	0.82	0.93
Maintenance (1000)	0.90	0.98	0.95	1.0
Maintenance (one)	0.94	0.90	0.94	0.95
Parking (1000)	0.96	0.91	0.97	0.97
Parking (one)	0.98	0.95	0.98	0.96
Freecell (1000)	0.54	0.73	0.61	0.79
Freecell (one)	0.53	0.80	0.63	0.85

Table 4.1: Pearson correlation coefficient between maximum h-depth and log(search effort) (log(h-backtracks)) in the non-heavy-tailed (N-HT) and heavy-tailed (HT) regimes for ensembles of random problems and for multiple runs on one problem.

	log(search effort)		*log*(*h*-backtracks)	
Domain	N-HT	HT	N-HT	HT
NoMystery (1000)	0.95	0.94	0.98	0.98
NoMystery (one)	0.95	0.95	0.99	0.98
Rovers (1000)	0.97	0.92	0.99	0.98
Rovers (one)	0.98	0.92	0.98	0.98
TPP (1000)	0.95	0.93	0.98	0.98
TPP (one)	0.97	0.91	0.99	0.97
Maintenance (1000)	0.92	1.0	0.96	1.0
Maintenance (one)	0.98	0.95	0.98	0.96
Parking (1000)	0.94	0.95	0.97	0.98
Parking (one)	0.95	0.95	0.98	0.98
Freecell (1000)	0.82	0.84	0.86	0.90
Freecell (one)	0.85	0.89	0.89	0.93

Table 4.2: Weighted Pearson correlation coefficient between maximum h-depth and *log*(search effort) (*log*(h-backtracks)) in the non-heavy-tailed (N-HT) and heavy-tailed (HT) regimes for ensembles of random problems and for multiple runs on one problem.

Given this strong correlation, a natural hypothesis, that is consistent with the results for CSP and SAT, is that the heavy-tailed behavior in planning problems is due to the distribution of h-depth of local minima in such problems. That is, there is a low probability of getting into a deep local minimum that leads to few, very hard, instances. In the next section, we will analyze this distribution and test this hypothesis.

4.5.3 The Distribution of Local Minima h-Depth

Based on our results on the connection between problem difficulty and local minima and inspired by the analysis of inconsistent subtrees in CSPs [62], our hypothesis is that the heavy-tailed behavior will be strongly correlated with the distribution of h-depth of the deepest local minimum encountered in search. In the heavy-tailed region, we expect to see a small, but non-negligible probability of encountering a deep local minimum, which accounts for the existence of exceptionally hard instances.

To investigate this hypothesis, we analyze the distribution of the h-depth of the deepest local minimum encountered in each problem instance for ensembles of 1000 random problem instances of different constrainedness levels. Figure 4.28 shows the distribution in the non-heavy-tailed regime ($C = 1$) and in the heavy-tailed regime ($C = 2$). In the non-heavy-tailed regime, we find that GBFS on the majority of the problem instances encounters a deep local minimum (the peak of the histogram is at 10). In the heavy-tailed regime, where most problems are easy but there are few *exceptionally hard problems*, we find that the distribution changes dramatically. The majority of the searches do not encounter a deep local minimum at all (the peak of the histogram is at 1), however a few instances do encounter local minima that are almost as deep as the deepest local minima in the non-heavy-tailed regime.

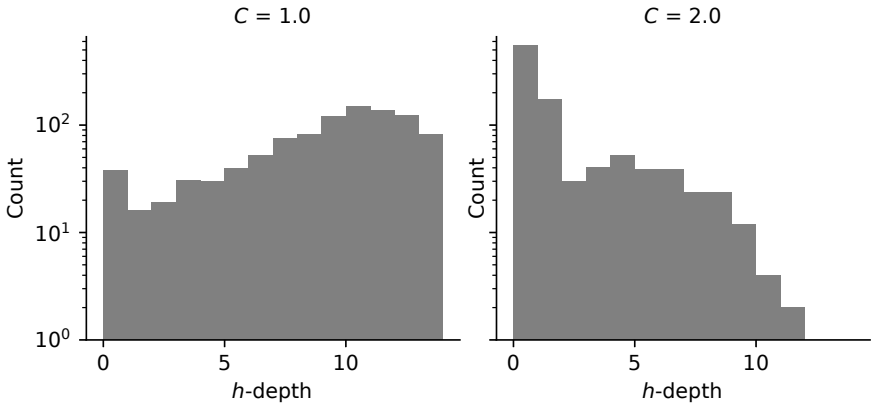


Figure 4.28: NoMystery: Distribution of deepest local minima h-depth in 1000 random instances solved with h^{FF}.

More interesting is the corresponding distribution for multiple runs of a randomized search procedure on one problem instance. Figure 4.29 shows the results when using a randomized search procedure with $h^{FF}_{\Delta 0.1}$ on the same instance for different values of C. We find similar trends: in the non-heavy-tailed regime (C=1) we find that the majority of runs encounter a deep local minimum. As we relax the problem and move to the heavy-tailed regime and easier problems on average, the majority of runs do not encounter a deep local minimum, however a few encounter a very deep local minimum.

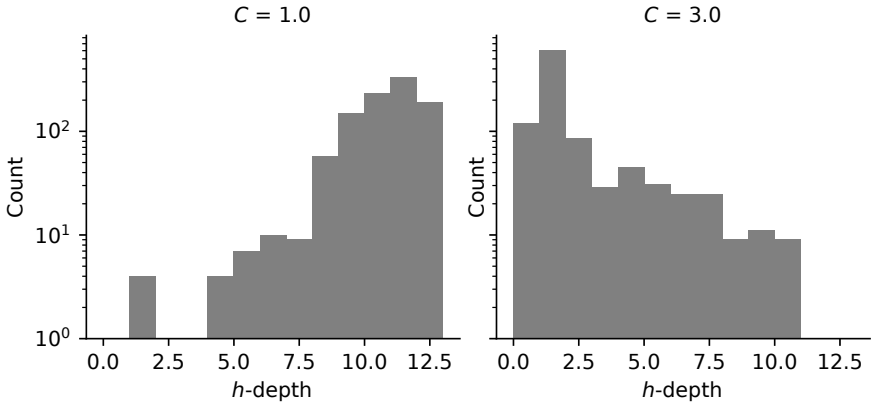
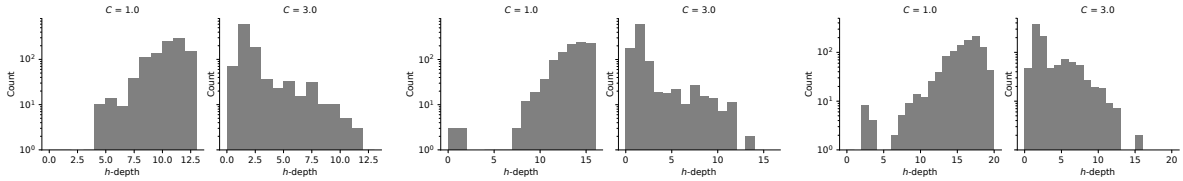


Figure 4.29: NoMystery: Distribution of deepest local minima h-depth in 1000 randomized runs on a single instance solved with h^{FF}.

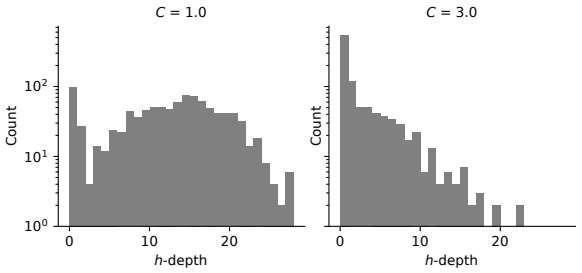
Figure 4.30a, Figure 4.30b, and Figure 4.30c show similar analysis for the landmark cut heuristic, the landmark count heuristic, and the CEA heuristic, respectively.



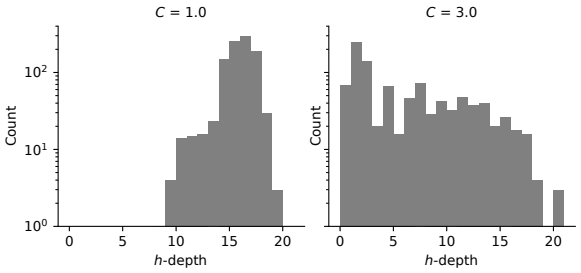
(a) Landmark cut heuristic. (b) Landmark count heuristic. (c) CEA heuristic.

Figure 4.30: NoMystery: Results for other heuristics.

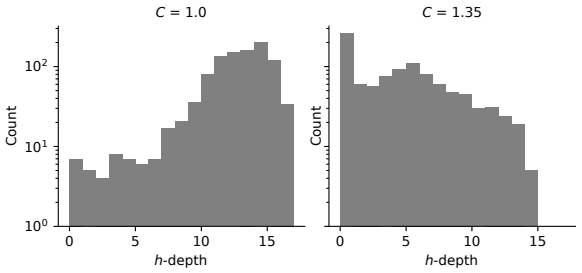
Figure 4.31 shows the distribution of deepest local minima h-depth results for the other domains in ensembles of 1000 random problems and 1000 randomized runs on a single instance. Recall that for Freecell, we observed a heavy-tailed behavior even for the most constrained instances (Section 4.4) and this is reflected in the results in Figure 4.31.



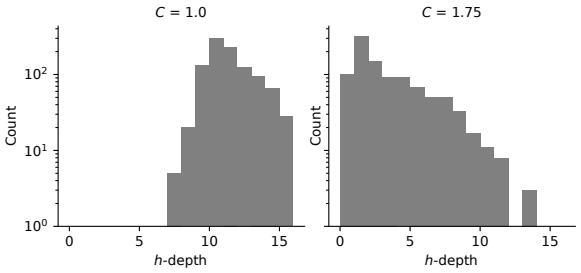
(a) Rovers: 1000 random instances.



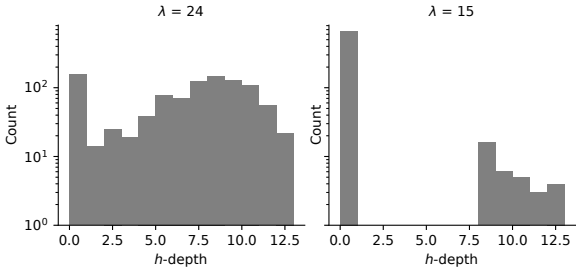
(b) Rovers: 1000 randomized runs on a single instance.



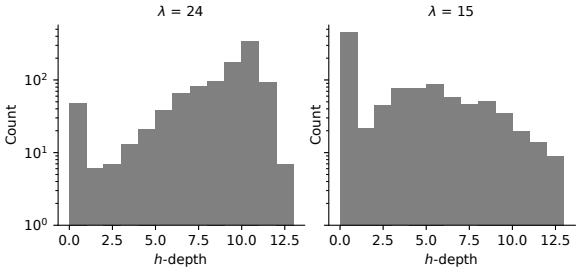
(c) TPP: 1000 random instances.



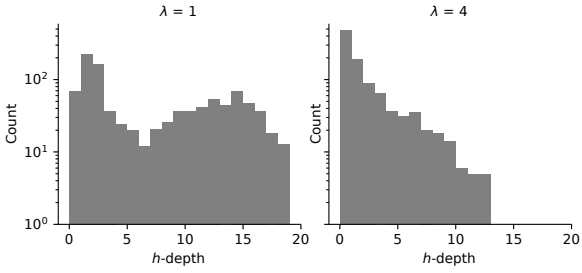
(d) TPP: 1000 randomized runs on a single instance.



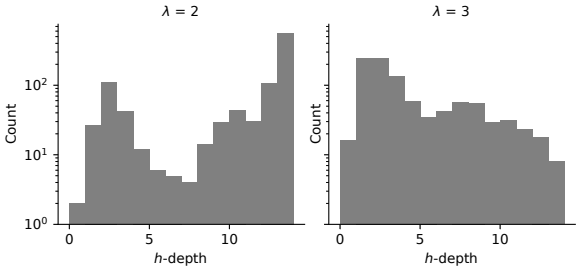
(e) Maintenance (alternative): 1000 random instances.



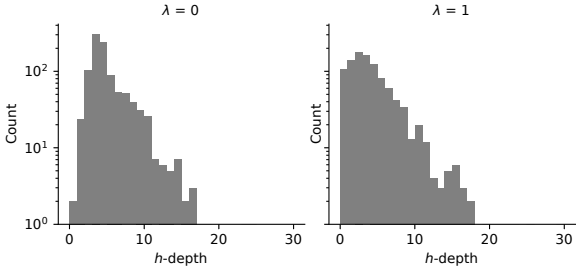
(f) Maintenance (alternative): 1000 randomized runs on a single instance.



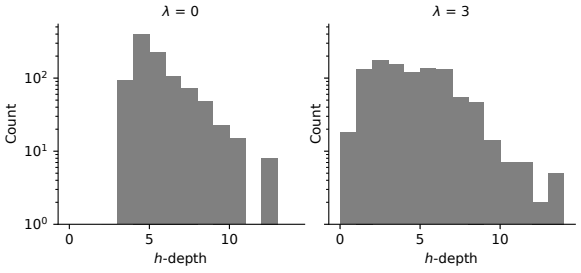
(g) Parking: 1000 random instances.



(h) Parking: 1000 randomized runs on a single instance.



(i) Freecell: 1000 random instances.



(j) Freecell: 1000 randomized runs on a single instance.

Figure 4.31: Results for 1000 randomized runs on a single instance with h^{FF}.

4.5.4 Discussion

Our analysis suggests that the h-depth of the local minima encountered in the search is a key factor in problem difficulty for different kinds of planning problems (we analyzed six domains with different characteristics and constrainedness type). Furthermore, we find an exponential correlation between the depth of a local minimum and its size, as often observed in tree search. These results are consistent with combinatorial search problems (e.g., in [62]) and further establish the connection between satisficing planning and a large body of literature on combinatorial search.

These results provide explanation and deeper understanding of several previously observed phenomena in GBFS:

1. Our results explain the heavy-tailed behavior observed in satisficing planning using GBFS. In relaxed problems, there is a low, but non-negligible, probability of encountering a deep local minimum (consistent with combinatorial search).

2. The exponential correlation between h-depth and search effort explains Wilt and Ruml's Observation 1 on the effect of large h difference in a local minimum [194].

3. The exponential correlation between h-depth and h-backtracks suggests Wilt and Ruml's Observation 3 is connected to Observation 1 [194]. Extensive backtracking behavior is the result of a deep local minimum.

4. The h-depth distribution suggests that the existence and extent of the factors highlighted by Wilt and Ruml (deep local minima and extensive backtracking behavior) depend on the constrainedness of problems. This results supports the conjecture in Chapter 3 that there is a connection between the existence and extent of local minima and the constrainedness of problems.

The heavy-tailed behavior observed in the investigated domains stems primarily from local minima, one type of uninformative heuristic region [203]. A second type, plateaus, are a different kind of search inefficiency that is not addressed in this work. The effect of constrainedness on domains with significant plateaus and no local minima in GBFS is an interesting direction for future work.

4.6 Randomization and Heavy-tailed Behavior in Satisficing Planning

In combinatorial search, the discovery of fat- and heavy-tailed behavior has inspired "boosted" search methods that employ randomized restarts in order to eliminate heavy-tailed behavior, achieving significant speedups on hard real-world problems [66]. Having established the existence of heavy-tailed behavior in satisficing planning, in this section we investigate different approaches to incorporate randomization in search and their impact on the heavy-tailed behavior. In Section 4.6.1 we analyze the impact of existing techniques for randomized exploration in heuristic search on the runtime distribution of GBFS. In Section 4.6.2 we present RR-GBFS, a novel variant of GBFS that employs randomized restarts, and analyze its impact on the heavy-tailed behavior of GBFS. RR-GBFS is inspired by similar techniques in CSPs [66] and is designed specifically to deal with heavy-tails.

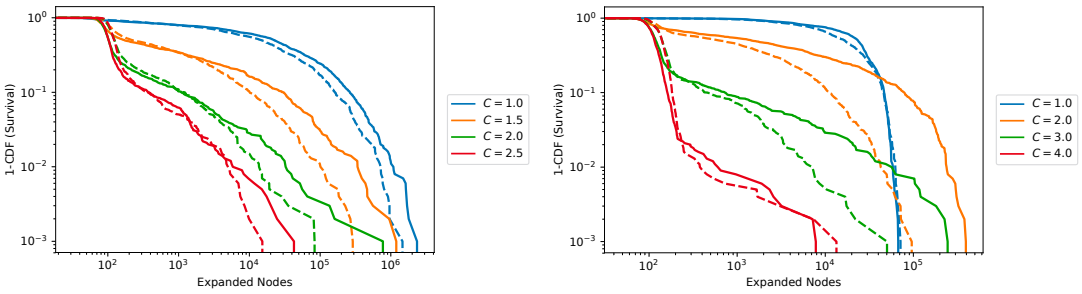
4.6.1 Existing Techniques for Randomized Exploration

In satisficing planning, several works have suggested incorporating non-greedy *random exploration*, in which the search allocates limited time to expand nodes with non-minimal h-values (see Section 2.2.3). Two recent approaches are ϵ-GBFS [176] and Type-GBFS [205].

In this section we examine the effect of these methods on the heavy tail phenomenon. Given the principled use of randomized restarts to escape heavy-tailed behavior in other combinatorial problems, an obvious question is whether the benefits of the random exploration approaches are higher in heavy-tailed regimes. Again, we start with a detailed analysis of the NoMystery domain and provide summarized results for the other domains. In our analysis, we use the ensembles of random problems used in Section 4.4 and Section 4.5 and study the behavior of ϵ-GBFS and Type-GBFS on these ensembles. We also study the behavior of these randomized variants on a single instance when using our randomized heuristic function (Section 4.3.3).

NoMystery

Figure 4.32a and Figure 4.32b show the effort distribution for an ensemble of 1000 random problems and 1000 randomized runs on a single instance with $h^{FF}_{\Delta 0.1}$, respectively, for both standard GBFS and ϵ-GBFS with $\epsilon = 0.2$. For the most constrained instances we see little to no improvement by using ϵ-GBFS. As we relax the problems and observe a fat- and heavy-tailed behavior for GBFS, we see that ϵ-GBFS manages to reduce the search effort for the hardest instances. As we further relax the problems and they gradually become easier, we see that again ϵ-GBFS offers smaller improvement. These results suggest that ϵ-GBFS can help reduce the heavy-tailed behavior and that the impact is mostly on instances that are not highly constrained or highly relaxed.

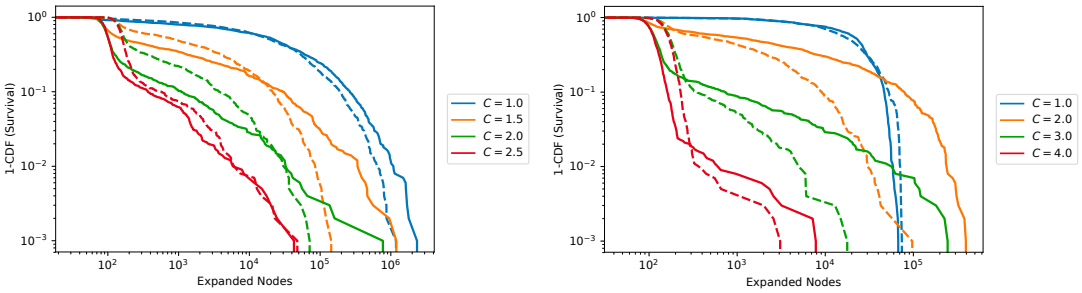


(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure 4.32: NoMystery: GBFS (solid) vs. ϵ-GBFS with $\epsilon = 0.2$ (dashed).

Figure 4.33 compares the runtime distribution of a randomized FF heuristic with $p = 0.1$ in a standard GBFS (solid lines) to a Type-GBFS (dashed lines), for both ensembles of 1000 random problems and multiple runs of a randomized search on a single instance. To directly address the impact of a non-greedy exploration, we use a simple type system that uses the h-values of the same heuristic (i.e., the type system (h) [205]) rather then a more sophisticated one that uses new information. While the exact numerical results are different from ϵ-GBFS, the results show similar trends and we observe a reduction in the heavy-tailed behavior. Again, we see a smaller impact on the most constrained instances and the most relaxed instances.



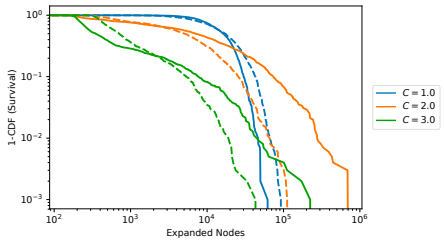
(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

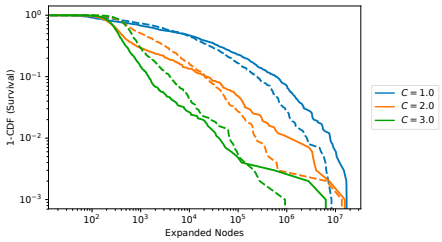
Figure 4.33: NoMystery: GBFS (solid) vs. Type-GBFS (dashed).

Rovers, TPP, Parking, Maintenance, Freecell

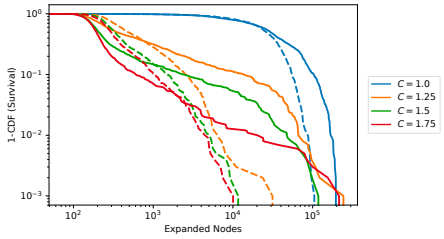
Figure 4.34 compares standard GBFS with Type-GBFS for the domains Rovers, TPP, Parking, Maintenance, and Freecell. The ensembles and the configuration of the randomized heuristic are similar to Section 4.4. We can clearly see that incorporating elements of random exploration in the search helps to reduce and even eliminate the heavy-tailed behavior. For the Rovers and TPP domains, we also observe a significant improvement for the highly-constrained problems. Still, the reduction in the tail for the more relaxed problems is significantly larger.



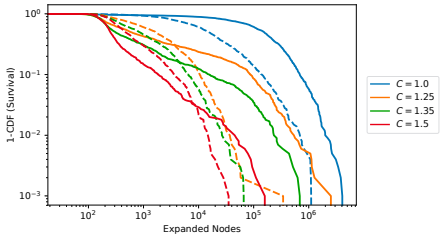
(a) Rovers: 1000 randomized runs on a single instance.



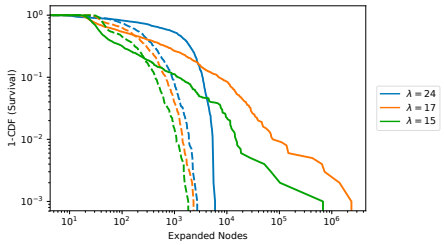
(b) Rovers: 1000 random instances.



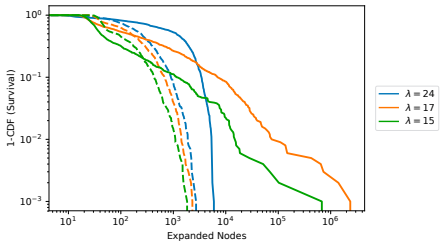
(c) TPP: 1000 randomized runs on a single instance.



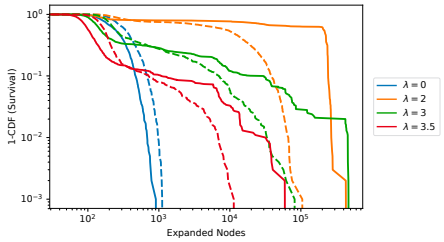
(d) TPP: 1000 random instances.



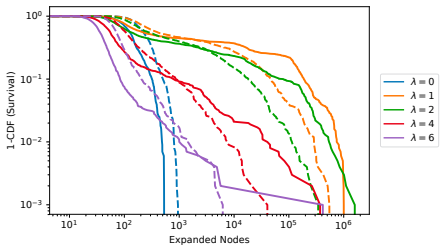
(e) Maintenance (alternative): 1000 randomized runs on a single instance.



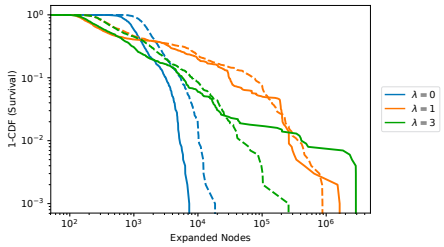
(f) Maintenance (alternative): 1000 random instances.



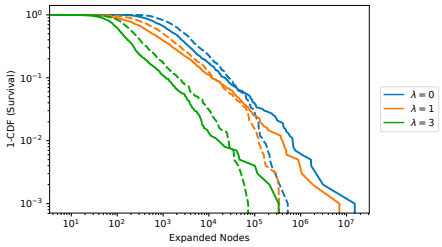
(g) Parking: 1000 randomized runs on a single instance.



(h) Parking: 1000 random instances.



(i) Freecell: 1000 randomized runs on a single instance.



(j) Freecell: 1000 random instances.

Figure 4.34: Results for 1000 randomized runs on a single instance with h^{FF}.

Our results show a clear pattern: incorporating elements of random exploration in the search helps to reduce the heavy-tailed behavior. We note that the impact of random exploration is not limited to reducing the heavy-tailed behavior and depending on the domain, the heuristic, and the randomized

variant, it can be beneficial even in highly constrained problems (e.g., in the Maintenance domain). In the next section, we consider a different solution that is designed to directly address the heavy-tailed behavior.

4.6.2 Randomized Restarting GBFS

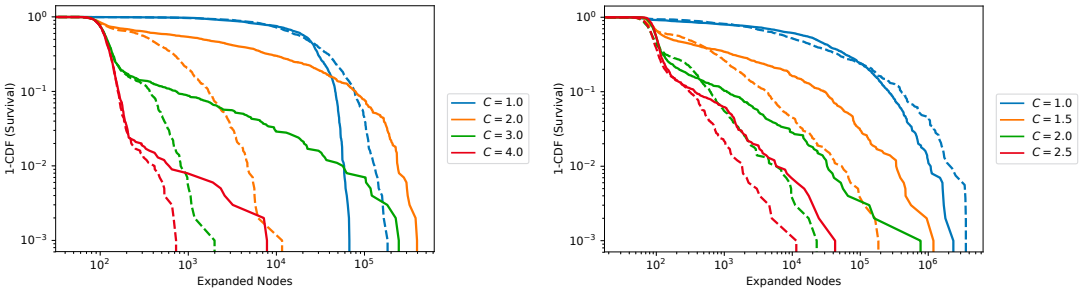
Following the work on randomized restarts in combinatorial search [66], in this section we demonstrate how such randomized restarts can be integrated into a GBFS to reduce the heavy-tailed behavior and gain significant speed-ups.

A restart strategy is a sequence $(t_1, t_2, ...)$ of cutoff values, i.e., run lengths (often expressing number of backtracks) after which the search restarts. As we have established the connection between the h-backtracks and the heavy-tailed behavior, we use cutoff values based on h-backtracks. Algorithm 4.1 presents pseudocode for a randomized restarting GBFS (RR-GBFS). In each iteration, we run a GBFS with a given h-backtrack cutoff. If the cutoff is reached before a solution is found, the randomized search is restarted with a different seed and the next cutoff value. We use randomized heuristic search with a geometric restart policy [185] with an initial value of 16, increasing with a factor of 1.5: $(16, 24, 36, ...)$.

Algorithm 4.1 Randomized restarting GBFS

```
function RR-GBFS (seed, cutoff)
    while GBFS(seed, cutoff) = NO_SOLUTION do
        seed ← ChooseRandomSeed()
        cutoff ← updateCutoff()
```

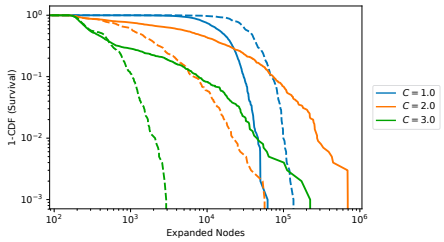
Figure 4.35a and 4.35b compare the runtime distribution of NoMystery for multiple runs on one instance and for an ensemble of random instances respectively, using randomized restarting GBFS vs. standard GBFS. Randomized-restarting GBFS manages to significantly reduce the tail in the fat- and heavy-tailed regimes, outperforming GBFS when $C > 1$. However, for the most constrained problems ($C = 1$), where the probability of a deep local minimum is very high and, therefore, restarting is likely to lead to another deep local minimum, RR-GBFS underperforms GBFS, as expected. Figure 4.36 shows similar analysis for the other domains. The experiments on multiple randomized runs on a single instance use the same p value used in Section 4.4. For ensembles of 1000 random instances, we used randomized restarts with $p = 0.05$ for all domain except for Parking where $p = 0.1$ is used.



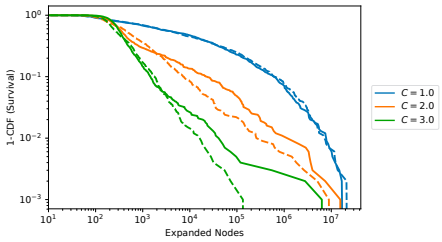
(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

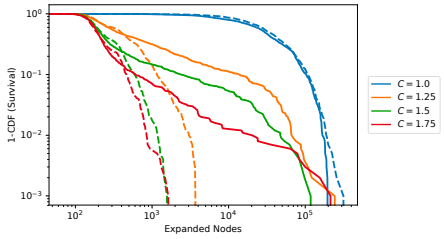
Figure 4.35: NoMystery: GBFS (solid) vs. RR-GBFS (dashed).



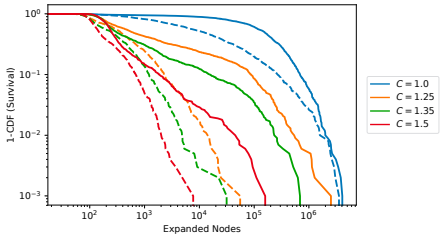
(a) Rovers: 1000 randomized runs on a single instance.



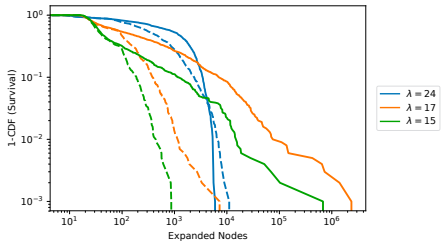
(b) Rovers: 1000 random instances.



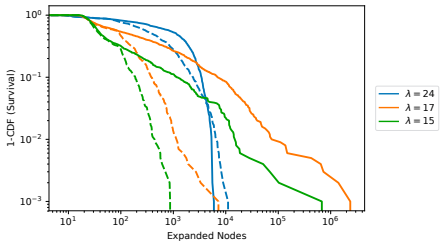
(c) TPP: 1000 randomized runs on a single instance.



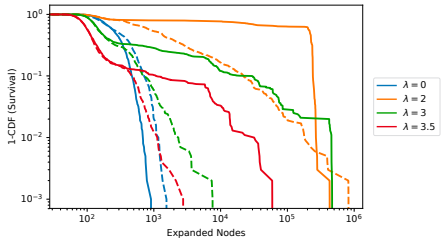
(d) TPP: 1000 random instances.



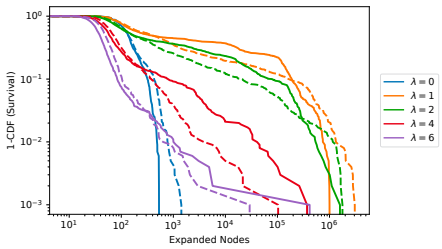
(e) Maintenance (alternative): 1000 randomized runs on a single instance.



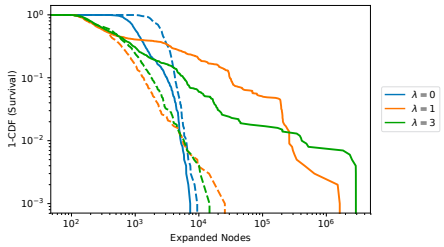
(f) Maintenance (alternative): 1000 random instances.



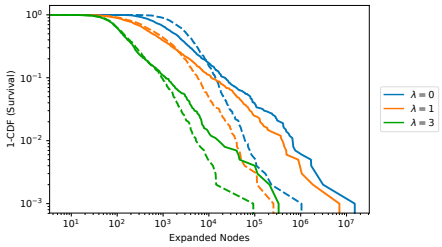
(g) Parking: 1000 randomized runs on a single instance.



(h) Parking: 1000 random instances.



(i) Freecell: 1000 randomized runs on a single instance.



(j) Freecell: 1000 random instances.

Figure 4.36: Results for RR-GBFS with h^{FF}.

Table 4.3 reports the maximum local minimum h-depth encountered for GBFS and for the last iteration of RR-GBFS, i.e., the iteration for which a solution was found, in both the heavy-tailed and non-heavy-tailed regimes. It also reports the maximum number of restarts needed by RR-GBFS. These

results explain the success of RR-GBFS for the less constrained problems. Exploiting the distribution of local minima h-depth in the heavy-tailed regime, RR-GBFS tends to require fewer restarts to successfully escape deep local minima in most of the domains, reducing the maximum h-depth significantly compared to GBFS. In the non-heavy-tailed regime, the number of restarts tends to be significantly higher and the maximum h-depth of RR-GBFS is much closer to GBFS.

	Non-Heavy-tailed			Heavy-tailed		
Domain	D_G	D_{RR}	$\#R$	D_G	D_{RR}	$\#R$
NoMystery (1000)	14	15	24	13	8	10
NoMystery (one)	13	13	16	12	5	3
Rovers (1000)	28	24	29	24	13	16
Rovers (one)	19	19	16	21	9	6
TPP (1000)	17	16	23	14	10	11
TPP (one)	16	12	18	13	7	4
Maintenance (1000)	13	12	25	15	8	5
Maintenance (one)	12	11	11	13	9	5
Parking (1000)	19	18	20	12	10	12
Parking (one)	14	13	17	13	6	6
Freecell (1000)	24	22	21	20	13	14
Freecell (one)	12	8	9	14	8	7

Table 4.3: The maximum h-depth for GBFS (D_G) and for RR-GBFS (D_{RR}) and the number of restarts in RR-GBFS ($\#R$) in the non-heavy-tailed regime vs. the heavy-tailed regime.

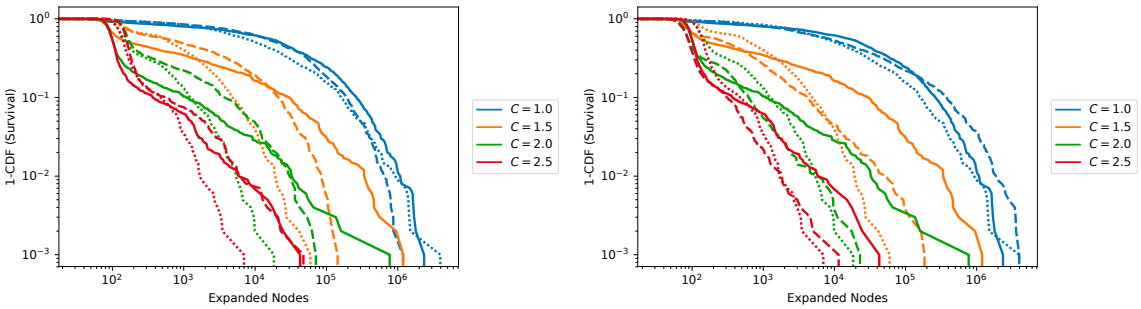
Our results support the hypothesis that randomized restarts exploit the distribution of local minima h-depth to escape deep local minima in the heavy-tailed regime. The proposed algorithm, RR-GBFS, achieves significant speed-up in the heavy-tailed regime and reduces the phenomenon of exceptionally hard problems. Consistent with our analysis in Section 4.5, RR-GBFS shows no improvement in the non-heavy-tailed regime where the probability of encountering a deep local minimum is much higher.

4.6.3 Discussion

In Section 4.5, we showed that the heavy-tailed behavior is due to a low, but non-negligible, probability of encountering a deep local minimum. Informed by this result, we investigated the use of randomization that can help escape deep local minima in GBFS. We first investigated existing general techniques for incorporating random exploration in GBFS and showed that they help reduce the heavy-tailed behavior. Then, we presented RR-GBFS, a variant of GBFS that employs randomized restarts. RR-GBFS, unlike the more general techniques for randomized exploration, is developed specifically to exploit the distribution of local minima h-depth in the heavy-tailed regime and our empirical results show that RR-GBFS is successful in reducing the heavy-tailed behavior and the phenomenon of exceptionally hard problems.

We can easily combine randomized restarts with random exploration by applying both techniques. We demonstrate the impact of applying both RR-GBFS and Type-GBFS on an ensemble of 1000 random problems in the NoMystery domain. Figure 4.37 compares the results for the combined configuration to each of the individual configurations (we also plot the results for standard GBFS for reference). We

can see that for the NoMystery domain, the combined configuration tends to outperform each of the individual ones in reducing the heavy-tailed behavior.



(a) GBFS (solid) vs. Type-GBFS (dashed) vs. RR-Type-GBFS (dotted).

(b) GBFS (solid) vs. RR-GBFS (dashed) vs. RR-Type-GBFS (dotted).

Figure 4.37: Results for combined configuration of Type-GBFS and RR-GBFS.

The work presented in this section raises a number of avenues for future work:

- A detailed investigation of the differences between RR-GBFS and the various variants of GBFS that incorporate random exploration in the search. Such variants include, in addition to ϵ-GBFS and Type-GBFS, other methods auch as GBFS-LS [203], GBFS with width-based exploration [122], DBFS [95], and IP-diversification [3]. Analyzing the effect of restarts in the presence of these exploration methods is also an interesting direction for future work.
- A detailed study of restart policies for GBFS. In particular, it is interesting to investigate how dynamic and learning restart policies (e.g., [102]) from combinatorial search can be incorporated in GBFS.

4.7 Conclusion

In this chapter, we performed an empirical analysis of the heavy-tailed behavior in ensembles of random planning problems and multiple runs of a randomized heuristic search on a single planning problem instance. We considered two notions of constrainedness in planning problems and showed that relaxed problems often exhibit fat- or heavy-tailed behavior, similar to CSPs and SAT.

Our empirical analysis indicates a strong exponential correlation between the h-depth of a local minimum and the associated search effort and a similar strong correlation between the h-depth and the number of h-backtracks. Surprisingly, we also find a strong exponential correlation between the h-depth of the *single* deepest local minimum encountered in an instance and the *total* search effort for that instance indicating that the deepest local minimum encountered in a search is an important factor in determining the total search effort. Furthermore, a key result of our empirical analysis is that the probability of entering a deep local minimum depends on the constrainedness of the problems and accounts for the observed heavy-tailed behavior.

Inspired by combinatorial optimization, we proposed a novel randomized restarting GBFS variant and show that it successfully escapes deep local minima in the heavy-tailed regime, resulting in better search performance. We also show that existing techniques for incorporating random exploration in heuristic search also reduce the heavy-tailed behavior.

These results provide a deeper understanding of GBFS and its search topology and explain several previous observations and conjectures on the behavior of GBFS, including Wilt and Ruml's [194] observations on the behavior of GBFS and our conjecture on the connection between constrainedness and local minima (see Section 3.8.2). We have demonstrated a simple way that these insights can be used to enhance search performance and believe that they can be further exploited to develop additional improvements to heuristic search algorithms.

Part II

Neural Sequence Decoding using Beam Search

Chapter 5

Background: Neural Sequence Models and Beam Search

In this dissertation, we develop empirical models for the search behavior of heuristic search algorithms in AI planning and neural sequence decoding. In Part I, we focused on satisficing AI planning using greedy best first search and developed empirical models for problem difficulty in GBFS. In Chapter 3, we established the existence of a phase transition in problem solubility of heuristic search problems and its relation to problem difficulty in GBFS. Furthermore, we made the connection to different factors that are known to impact problem difficulty, such as the operator cost ratio and the re-expansions of nodes, and we discover the existence of exceptionally hard problems in ensembles of easy problems. Then, in Chapter 4, we established the existence of a heavy-tailed behavior in GBFS that accounts for the exceptionally hard problems and presented an explanatory model for this heavy-tailed behavior that is based on the distribution of the h-depth of local minima. Based on this analysis, we showed how incorporating randomization in the search procedure can help escape deep local minima and reduce the heavy-tailed behavior and presented a novel variant of GBFS that addresses the heavy-tailed behavior and outperforms GBFS.

In Part II, we focus on neural sequence models and present empirical models for the search behavior of beam search in decoding neural sequence models. In Chapter 6, we study the well-known problem of beam search performance degradation and present an explanatory model that is based on search discrepancies. Informed by our analysis, we show how the performance degradation can be mitigated by constraining the search discrepancies considered by the beam search. In Chapter 7, we focus on goal-oriented complete beam search and observe a heavy-tailed behavior in problem difficulty, similar to the one observed for GBFS in Chapter 4. Informed by our analysis of the heavy-tailed behavior in GBFS, we introduce randomization in the beam search procedure, and show that a randomized variant of beam search can reduce the heavy-tailed behavior and outperform standard beam search.

In this chapter we present relevant background and notation to the work presented in Chapter 6 and Chapter 7. Section 5.1 describes the modeling and training of neural sequence models and Section 5.2 describes algorithms for decoding neural sequence models with an emphasis on beam search. The notation in this chapter is based on several sources [67, 107, 173, 69].

5.1 Neural Sequence Models

A sequence model is a factorized parametric distribution over sequences [107]. The parameter θ defines the probability $p_\theta(y_t|y_{1:t-1})$ of the next token y_t, conditioned on the partial sequence $y_{1:t-1}$. Typically p_θ is defined as a softmax normalization of unnormalized log-probabilities $\phi(y_t|y_{1:t-1})$:

$$p(y_t|y_{1:t-1}) = \frac{exp(\phi(y_t|y_{1:t-1}))}{\sum_{y'} exp(\phi(y'|y_{1:t-1}))}. \tag{5.1}$$

In models that are conditioned on a context, e.g., an input sentence in machine translation, we instead model the distribution $p_\theta(y_t|x; y_{1:t-1})$ that is conditioned on both the context x and the partial sequence $y_{1:t-1}$. Note that these models are locally normalized since the normalization is w.r.t. a single token [107].

A sequence model defines a valid probability distribution over both partial and complete sequences, and the total probability of a sequence $y_{1:t}$ can be computed using the chain rule,

$$p_\theta(y_{1:t}) = p_\theta(y_t|y_{1:t-1}) \cdot p_\theta(y_{1:t-1}) = \prod_{t'=1}^{t} p_\theta(y_{t'}|y_{1:t'-1}). \tag{5.2}$$

Similarly, the total conditional probability of a sequence $y_{1:t}$ conditioned on x can be computed as follows:

$$p_\theta(y_{1:t}|x) = p_\theta(y_t|x, y_{1:t-1}) \cdot p_\theta(y_{1:t-1}|x) = \prod_{t'=1}^{t} p_\theta(y_{t'}|x, y_{1:t'-1}). \tag{5.3}$$

5.1.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) are a family of neural networks for processing sequential data [67]. Typically, an RNN is a nonlinear mapping from sequences to sequences. Given an input sequence $(i_1, i_2, ..., i_T)$, the RNN computes a sequence of hidden states $(h_1, h_2, ..., h_T)$ and a sequence of outputs $o_1, o_2, ..., o_T$ as follows [69]:

$$\begin{aligned} h_t =& f(h_{t-1}, i_t) = \sigma^h(W^{(hh)}h_{t-1} + W^{(ih)}i_t + b^{(h)}) \\ o_t =& g(h_t) = \sigma^o(W^{(ho)}h_t + b^{(o)}) \end{aligned} \tag{5.4}$$

where σ^h is the hidden layer nonlinear function, typically tanh, and σ^o is the output layer nonlinear function, typically softmax (see Table 5.1). The RNN is parameterized by the weights of the connections between hidden units $W^{(hh)}$, the weights of the connections from the input to the hidden units $W^{(ih)}$, and from the hidden units to the output units $W^{(ho)}$. Figure 5.1 illustrates the basic recurrent neural network described in Eq. (5.4).

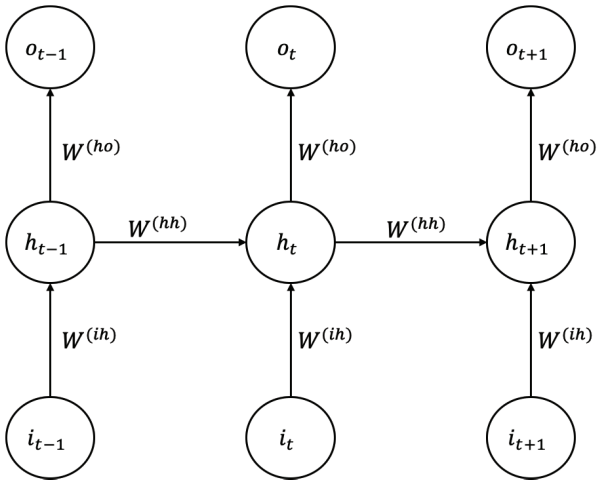


Figure 5.1: A Recurrent Neural Network.

Table 5.1: Commonly used non-linear functions.

Name	Function
Sigmoid	$f(x) = \frac{1}{1+\exp -x}$
Tanh	$f(x) = \frac{\exp^x - \exp^{-x}}{\exp^x + \exp^{-x}}$
RELU	$f(x) = max(0, x)$
Softmax	$f(x_i) = \frac{\exp^{x_i}}{\sum_{k=1}^{K} \exp^{x_k}}, i = 1..K$

It is common to model $p_\theta(y_t \mid x; \{y_0, ..., y_{t-1}\})$ using an RNN where the context x is represented by the initial state of the RNN, h_0, and in each step we set the input i_t to be the selected token in the previous step, $i_t = y_{t-1}$ (where y_0 is normally a special token that represents the beginning of a sequence).[1] In each step, the fixed length hidden state h_t therefore represents the context x and the previous tokens in y and is updated using a non-linear function f: $h_t = f(h_{t-1}, y_{t-1})$. The conditional probability $p_\theta(y_t \mid x; \{y_0, ..., y_{t-1}\})$ is then given by $g(h_t)$, assuming σ^o is a function that produces valid probabilities, e.g., softmax.

5.1.2 Sequence-to-Sequence Models

Sequence models where the input context x is a sequence as well are called *sequence-to-sequence models*. The most commonly used model in these scenarios is the encoder-decoder model [173] where the encoder takes in a sequence and generates a vector representation of fixed dimensionality called the context and the decoder generates the output sequence based on the context vector.

Given an input sequence $x = (x_1, x_2, ...x_{T^x})$, the encoder typically consists of an additional RNN such that $h_t^{enc} = f^{enc}(h_t^{enc}, x_t)$ where the last hidden state $h_{T^x+1}^{enc}$ is used as the context that is passed

[1] Alternatively, the context x can be provided as an extra input in each time step [67].

to the decoder. However, several recent works have employed alternative encoders, most notably the Transformer model [178] and convolutional encoders [49, 183].

Attention-based Decoding

One of limitations of the encoder-decoder model is the use of fixed-size context vector. For long sequences, it might not be sufficient to represent all the important information from the input sequence. To address this limitation, Bahdanau [4] proposed the mechanism of *attention* where the decoder attends to different parts of the input sequence at each time step t based on a dynamic context c_t. Each context vector c_t is computed as a weighted linear combination of the hidden states produced by the encoder RNN:

$$c_t = \sum_{k=1}^{T^x} \alpha_{tk} h_k^{enc}$$

where h_k^{enc} is the hidden state of the encoder at time step k and $\alpha_t \in \mathbb{R}^{T^x}$ is a probability distribution over the encoder hidden states computed using a Multi-Layer Perceptron (MLP) that takes in the previous decoder hidden state h_{t-1} and the hidden states of the encoder h^{enc},

$$\begin{aligned} \alpha_t &= \text{softmax}(e_t) \\ e_{tk} &= MLP(h_{t-1}, h_k^{enc}), \end{aligned} \tag{5.5}$$

where $e_t \in \mathbb{R}^{T^x}$ are unnormalized scores representing how well inputs around position k and the output at position t match [4].

5.1.3 Training Neural Sequence Models

Neural sequence models are typically trained with a cross entropy loss [67]. For an output sequence y, the cross entropy loss is calculated based on the individual losses at each time step,

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t = \sum_{t=1}^{T} \mathcal{L}(o_t, y_t^*),$$

where $\mathcal{L}(o_t, y_t^*)$ is the cross entropy loss between o_t, the output of the softmax in Eq. (5.4) at time t, and the ground truth token y_t^*. The cross entropy loss is derived as follows:

$$\mathcal{L}(o_t, y_t^*) = -\mathbb{1}_{y_t^*}^{tr} \cdot \log o_t,$$

where $-\mathbb{1}_{y_t^*}^{tr}$ is the transpose of one-hot vector of the ground truth token y_t^* and $\log o_t$ is a vector of log-probabilities for each token generated by softmax in Eq. (5.4). Typically, training is done using *teacher forcing*, where the model gets the ground truth output y_t^* as an input at time $t+1$. Training of recurrent models that have hidden-to-hidden connection (e.g., in Eq. (5.4)) requires backpropogation through time (BPTT) [189].

5.1.4 Vanishing and Exploding Gradients

Previous work has highlighted difficulties in training recurrent neural networks with long-term dependencies using gradient-based learning algorithms due to vanishing and exploding gradients [10, 141]. For example, consider the gradient of the loss with respect to the weights $W^{(hh)}$,

$$\frac{\partial \mathcal{L}}{\partial W^{(hh)}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial W^{(hh)}} \tag{5.6}$$

$$\frac{\partial \mathcal{L}_t}{\partial W^{(hh)}} = \sum_{k=1}^{t} \Big(\frac{\partial \mathcal{L}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial W^{(hh)}} \Big). \tag{5.7}$$

Each gradient component $\frac{\partial \mathcal{L}_t}{\partial W^{(hh)}}$ is the sum of components that are called the *temporal* contributions ($\frac{\partial \mathcal{L}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial W^{(hh)}}$) that measure how $W^{(hh)}$ at step k affects the cost at step $t > k$ [141]. When $k \ll t$, these components are referred to as long-term contributions. The term $\frac{\partial h_t}{\partial h_k}$ takes the form of a product of $t - k$ Jacobian matrices that, in the case of long-term components, can result in the norm growing quickly towards infinity (the exploding gradients problem) or, alternatively, shrinking quickly towards zero (the vanishing gradients problem) [141].

In order to address the problem of exploding gradients, Mikolov [130] and Pascanu et al. [141] proposed *gradient clipping*, a method of rescaling the gradients based on their norm whenever the norm goes over a given threshold.

To address the problem of vanishing gradient, Hochreiter and Schmidhuber [86] proposed the *long short-term memory (LSTM)* architecture that consists of memory cells that have a self connection of weight 1 and the flow of information to and from the cell is controlled by learned input and output gates. When the gates are shut, the gradients can flow through the memory cells without alteration for many time steps thus overcoming the vanishing gradients problem [172]. Eq. (5.8) describes a widely-used LSTM variant that includes an additional forget gate that controls the weight of the self-connection and allows the cell to clear its memory [58]:

$$\begin{aligned}
g_t^n &= sigmoid(W^{(in)} i_t + W^{(hn)} h_{t-1} + W^{(cn)} c_{t-1} + b^{(n)}) \\
g_t^f &= sigmoid(W^{(if)} i_t + W^{(hf)} h_{t-1} + W^{(cf)} c_{t-1} + b^{(f)}) \\
c_t &= g_t^f \cdot c_{t-1} + g_t^n \cdot tanh(W^{(ic)} i_t + W^{(hc)} h_{t-1} + b^{(c)}) \\
g_t^o &= sigmoid(W^{(io)} i_t + W^{(ho)} h_{t-1} + W^{(co)} c_t + b^{(o)}) \\
h_t &= g_t^o \cdot tanh(c_t),
\end{aligned} \tag{5.8}$$

where g^n, g^f, and g^o are the input gate, forget gate, and output gate, respectively. Figure 5.2 illustrates the structure of LSTM and describes the role of each weight matrix.

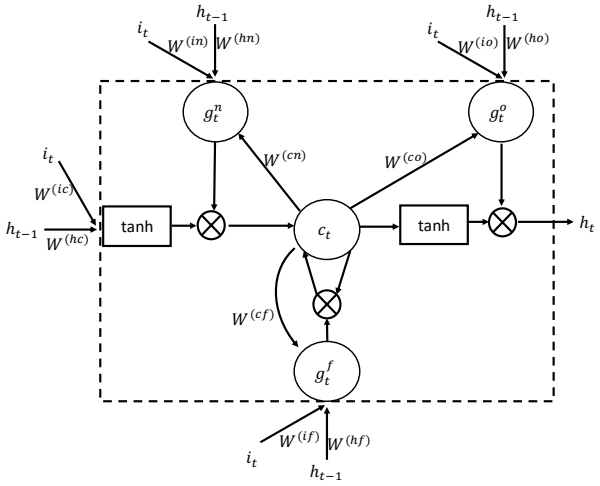


Figure 5.2: Long Short-Term Memory (LSTM).

An alternative RNN variant that was found effective in addressing the vanishing gradient problem is the *Gated Recurrent Unit (GRU)* [23]:

$$\begin{aligned} g_t^r =& sigmoid(W^{(ir)} i_t + W^{(hr)} h_{t-1} + b^{(r)}) \\ g_t^z =& sigmoid(W^{(iz)} i_t + W^{(hz)} h_{t-1} + b^{(z)}) \\ \widetilde{h}_t =& tanh(W^{(ih)} i_t + W^{(hh)} (g_t^r \cdot h_{t-1}) + b^{(h)}) \\ h_t =& (1 - g_t^z) h_{t-1} + g_t^z \widetilde{h}_t. \end{aligned} \tag{5.9}$$

Similar to LSTM, GRU has gates that control the flow of information, however it does not have a separate memory cell. The update gate g^z controls how much the unit updates its content, while the reset gate g^r allows the unit to forget its content.

5.2 Decoding Neural Sequence Models

In Section 5.1, we described neural sequence models and how they are trained. In this section, we describe how neural sequence models are decoded.

Typically, the goal of the decoding process is the find a sequence y of length T that has the highest probability, conditioned on a context x, according to our model [181],

$$\arg\max_y p_\theta(y|x), \tag{5.10}$$

where θ represents the parameters of our model and x represents the context. Since neural sequence models estimate the conditional probability $p_\theta(y_t|x, y_{1:t-1})$, we can write Eq. (5.10) as the product of the tokens' conditional probabilities (using Eq. (5.2)):

$$\arg\max_y p_\theta(y|x) = \arg\max_{y_1,\ldots,y_T} \prod_{t=1}^{T} p_\theta(y_t|x, y_{1:t-1}). \tag{5.11}$$

Solving Eq. (5.11) can be viewed as Maximum a Posteriori (MAP) inference on a T-order Markov

chain [181]. Since exact inference is NP-hard, Eq (5.10) is normally solved using approximate inference algorithms. In this section we cover the most commonly used approximate algorithms for MAP inference on neural sequence models.

5.2.1 Sampling

Random sampling is a simple inference algorithm where we iteratively sample sequences from our model, token-by-token, according to the conditional probability distribution of each token conditioned on the previous tokens and the context [137]. To approximate Eq. (5.11) we usually collect multiple samples and return the one that has the highest probability. Pseudo-code of this algorithm is provided in Algorithm 5.1.

Algorithm 5.1 Random Sampling

function RandomSampling(model θ, context x, number of samples N)
  for i=1..N **do** ▷ Collect N samples, $y^i, i = 1..N$
    for t=1..T **do**
      $y_t^i \sim p_\theta(y|x, y_{1:t-1}^i)$ ▷ Draw random token conditioned on context and $y_{1:t-1}$
  return y^i such that $p_\theta(y^i|x) \geq p_\theta^j(y^j|x)$ ▷ Return the highest probability sample

To control the randomness of the sampling, it is common to use softmax temperature [107],

$$p_\theta(y_t|x; y_{1:t-1}) = \frac{exp(\phi_\theta(y_t|x; y_{1:t-1})/\mathcal{T})}{\sum_{y'} exp(\phi_\theta(y'|x; y_{1:t-1})/\mathcal{T})}, \tag{5.12}$$

where $\mathcal{T}$ represent the softmax temperature. When $\mathcal{T} = 1$, sampling is done according to the standard softmax probabilities. Setting $0 < \mathcal{T} < 1$ will lead to a less diverse sampling, while setting $\mathcal{T} > 1$ will lead to more diverse sampling compared to standard softmax.

5.2.2 Greedy Decoding

Greedy decoding is a simple and fast algorithm: at each step, the most likely token, conditioned on the previous tokens and the context, is selected [137]. Algorithm 5.2 shows pseudo-code for greedy decoding.

Greedy decoding makes a sequence of locally optimal decisions, however it often reaches a local minima if there are higher probability sequences that start with lower probability tokens.

Algorithm 5.2 Greedy Decoding

function GreedyDecoding(model θ, context x)
  for t=1..T **do**
    $y_t = \arg\max_{y'} p_\theta(y'|x, y_{1:t-1})$ ▷ Select the most likely token conditioned on x and $y_{1:t-1}$
  return y

5.2.3 Beam Search

Beam search is any search algorithm in which a number of alternatives (the beam) are examined in parallel, while heuristic rules are used to prune non-promising alternatives [12].

In the context of neural sequence decoding, beam search is a limited-width breadth-first search that is used as an approximation to finding the (single) sequence y that maximizes Eq. (5.11), or as a way to obtain a set of high-probability sequences from the model [107]. Starting from an empty sequence, at every step $t = 0, 1, 2, ...$ beam search expands at most $\mathcal{B}$ partial sequences (those with highest probability) to compute the probabilities of sequences with length $t + 1$. It terminates with a set of $\mathcal{B}$ complete sequences, which we assume to be of equal length (as they can be padded). $\mathcal{B}$ is called the beam width or the beam size and the set of $\mathcal{B}$ complete sequences is viewed as the M-best list of the MAP inference [7]. Pseudo-code of beam search can be found in Algorithm 5.3.[2]

Algorithm 5.3 Beam Search

function BeamSearch(model θ, context x, beam width $\mathcal{B}$)
  $beams \leftarrow \{\varnothing\}$ ▷ Initialize the beam with empty sequence
  $score(\varnothing) \leftarrow 1.0$ ▷ Set probability of empty sequence to be 1.0
  for $t = 1..T$ **do**
    $candidates \leftarrow \{\}$
    for $b \in beams$ **do**
      for $v \in \mathcal{V}$ **do** ▷ For each token in the vocabulary $\mathcal{V}$
        $b' \leftarrow b \parallel v$ ▷ Extend sequence b with token v
        $scores(b') \leftarrow score(b) + p_\theta(v|x, b)$
        $candidates \leftarrow candidates \cup b'$
    $beams \leftarrow Top\text{-}\mathcal{B}(candidates, scores)$ ▷ Keep top $\mathcal{B}$ extensions of sequences in beam
  return $beams$

Vijayakumar et al. [181] provided a mathematical notation to describes the set of beam candidates in each step. We denote the set of $\mathcal{B}$ partial solutions held by the beam search at time step $t-1$ as $Y_{[t-1]}=\{\mathrm{y}_{1,[t-1]}, ..., \mathrm{y}_{B,[t-1]}\}$. At each step, beam search selects the top scoring $\mathcal{B}$ candidates from the set of all possible one token extensions of its beams $\mathcal{Y}_t=\{\mathrm{y}_{[t]} \mid \mathrm{y}_{[t-1]} \in Y_{[t-1]} \wedge y_t \in \mathcal{V}\}$. Formally, the beam search candidates are updated as follows:

$$Y_{[t]} = \underset{\mathrm{y}_{1,[t]},...,\mathrm{y}_{B,[t]} \in \mathcal{Y}_t}{\arg\max} \sum_{b \in [1..B]} \log P_\theta(\mathrm{y}_{b,[t]} \mid \mathrm{x}) \quad s.t. \quad \mathrm{y}_i \neq \mathrm{y}_j \quad \forall i \neq j; \quad i, j \in [1..B] \tag{5.13}$$

5.3 Beam Search Variants and Enhancements

Beam search is the most popular algorithm for neural sequence decoding [181], and is the focus of Chapters 6 and 7. In this section, we review notable variants of beam search and commonly used beam search enhancements.

[2] The pseudo-code is provided to conceptually explain how beam search works. In practice, there are different, and more efficient, implementations of beam search. In particular, when used for neural sequence decoding, it is common to use the GPU to compute the candidates scores' and select the top $\mathcal{B}$ candidates.

5.3.1 Complete Variants of Beam Search

Previous works have considered the use of beam search for solving combinatorial problems such as constraint satisfaction problems (CSPs) [208] and domain-specific planning [199, 47]. In these problems, we are not looking for the most likely sequence according to some learned model. Instead, we are looking for a solution that satisfies some goal criteria. In this context, we consider beam search to be *incomplete*: the use of pruning rules (e.g., limited beam width) can lead to pruning all partial solutions that lead to a solution that satisfies the goal criteria [199]. There are two types of approaches to make beam search complete. The first, called complete anytime beam search [208], consists on restarting the search with a wider beam until a solution is found. An alternative that does not require increasing the memory consumption is to use backtracking [47, 209]. In this section, we describe the two approaches in detail.

Complete Anytime Beam Search

Complete anytime beam search [208] is a complete variant of beam search that relies on iterative weakening [146]: each time a beam search fails to find a solution, the beam search is restarted with weakened pruning rules. In state spaces where all returned solutions are feasible, complete anytime beam search is an *anytime* algorithm [33] since it can quickly find a solution in the first iteration and continuously find better solutions in subsequent iterations. In the case of neural sequence decoding, beam search is a limited-width breadth-first search and the pruning is based on the beam width $\mathcal{B}$. The common variant of the complete anytime beam search in neural sequence decoding involves restarting the beam search with a larger beam width each time it fails to find a solution [199, 210, 5, 111]. Algorithm 5.4 provides pseudo-code of this variant of complete anytime beam search, using a parameter κ that controls the rate of increase in beam width in each iteration.

Algorithm 5.4 Complete Anytime Beam Search

```
function CAB(goal criteria 𝒢, beam width increase factor κ)
    beamWidth ← 1
    while not solved do
        candidates ← BeamSearch(beamWidth)
        for c ∈ candidates do
            if c satisfies 𝒢 then
                return c
        beamWidth ← κ · beamWidth
```

Beam Search with Backtracking

An alternative approach to iterative weakening is to use backtracking [209]. Unlike iterative weakening, this approach maintains a similar memory complexity to standard beam search.

Zhou and Hansen [209] proposed beam-stack search, an anytime complete variant of beam search that is based on backtracking and pruning of nodes using an admissible heuristic. Beam-stack starts with an initial upper bound based on an initial solution found by an approximation algorithm, however Wilt et al. [199] modified it to take infinity as the upper bound in case we do not have such initial solution.

Furcy and Koenig [47] proposed Beam Search Using Limited Discrepancy Backtracking (BULB), another complete, backtracking-based variant of beam search, however instead of using chronological

backtracking, they use limited discrepancy search [73]. Table 5.2 shows Furcy and Koenig's [47] taxonomy of beam search methods. Note that depth-first beam search [47] is a simplified version of Zhou and Hansen's [209] beam-stack search.

Table 5.2: Taxonomy of beam search methods [47].

Beam	Type of backtracking		
width	None	Chronological	Limited discrepancy
1	Greedy search	Guided depth-first search	Limited discrepancy search
$1 < n < \infty$	Beam search	Depth-first beam search	Beam search using limited discrepancy
∞	Breadth-first search	Breadth-first search	Breadth-first search

5.3.2 Diversity in Beam Search

Despite the popularity of beam search, previous work has highlighted that the M-best list generated by beam search tends to include similar sequences (often differing by only one token), and is a poor surrogate for the entire search space [59, 114, 115]. Different methods have been proposed to increase the diversity in beam search results (e.g., [181, 107, 114, 115]), many of them exclusive to natural language applications. In this section, we describe in detail two recent techniques that can be used across applications.

Diverse Beam Search (DBS)

Vijayakumar et al. [181] proposed to modify the objective of beam search (Eq. (5.13)) to account for diversity. They considered a partition of the beams $Y_{[t]}$ into G groups, $Y^g_{[t]}, g \in [1, G]$, each containing $\mathcal{B}' = \mathcal{B}/G$ elements. At each time step, each group is greedily updated by selecting extensions of the current partial sequences $Y^g_{[t]} = \{Y^g_{1,[t]}, ..., Y^g_{B',[t]}\}$ that maximize a linear combination of sequence likelihood and diversity with respect to previous groups. To measure diversity, Vijayakumar et al. introduced a diversity function $\Delta(\mathrm{y}_{[t]}, Y^g_{[t]})$ that measures the dissimilarity between a sequence $\mathrm{y}_{[t]}$ and a group $Y^g_{[t]}$,

$$\Delta(\mathrm{y}_{[t]}, Y^g_{[t]}) = \sum_{b=1}^{B'} \delta(\mathrm{y}_{[t]}, \mathrm{y}^g_{b,[t]}),$$

where $\delta(\cdot, \cdot)$ is a measure of sequence dissimilarity, e.g., Hamming distance. DBS optimizes each group while holding previously extended groups fixed and incorporating diversity into the beam search objective (Eq. (5.13)). In time step t, the objective for updating each group g is as follows:

$$Y^g_{[t]} = \underset{\mathrm{y}^g_{1,[t]},...,\mathrm{y}^g_{\mathcal{B}',[t]} \in \mathcal{Y}_t}{\arg\max} \sum_{b \in [1..\mathcal{B}]} \log P_\theta(\mathrm{y}^g_{b,[t]} \mid \mathrm{x}) + \lambda \sum_{h=1}^{g-1} \Delta(\mathrm{y}^g_{b,[t]}, Y^h_{[t]}) \quad s.t. \quad \lambda > 0, \quad \mathrm{y}^g_{i,[t]} \neq \mathrm{y}^g_{j,[t]} \quad \forall i \neq j. \tag{5.14}$$

Algorithm 5.5 presents Vijayakumar et al.'s beam search procedure modified to reflect the objective in Eq. (5.14). Each *argmax* can be computed in a similar manner to Algorithm 5.3.

Algorithm 5.5 Diverse Beam Search [181]

function DBS(model θ, context x, beam width $\mathcal{B}$, number of groups G)
  $\mathcal{B}' \leftarrow \mathcal{B}/G$
  for $t = 1..T$ **do**
    $Y^1_{[t]} \leftarrow \arg\max_{\mathrm{y}^1_{1,[t]},\ldots,\mathrm{y}^1_{\mathcal{B}',[t]} \in \mathcal{Y}_t} \sum_{b\in[1..\mathcal{B}]} \log P_\theta(\mathrm{y}^1_{b,[t]} \mid \mathrm{x}) \quad s.t. \quad \mathrm{y}^1_{i,[t]} \neq \mathrm{y}^1_{j,[t]} \forall i \neq j$
    for $g = 2..G$ **do**
      $\log P_\theta(Y^g_{[t]}) \leftarrow \log P_\theta(Y^g_{[t]}) + \lambda \sum_{h=1}^{g-1} \Delta(\mathrm{y}^g_{b,[t]}, Y^h_{[t]})$
      $Y^g_{[t]} \leftarrow \arg\max_{\mathrm{y}^g_{1,[t]},\ldots,\mathrm{y}^g_{\mathcal{B}',[t]} \in \mathcal{Y}_t} \sum_{b\in[1..\mathcal{B}]} \log P_\theta(\mathrm{y}^g_{b,[t]} \mid \mathrm{x}) \quad s.t. \quad \mathrm{y}^g_{i,[t]} \neq \mathrm{y}^g_{j,[t]} \forall i \neq j$
  $Y_{[T]} \leftarrow \bigcup_{g=1}^{G} Y^g_{[T]}$
  return $Y_{[T]}$

Stochastic Beam Search (SBS)

The Gumbel-Max trick [70, 125] allows sampling from the categorical distribution by perturbing the log-probability for each category. The process involves adding Gumbel-distributed noise and selecting the category with the highest perturbed log-probability. Selecting the k categories with the highest perturbed log-probabilities is equivalent to sampling k items without replacement from the categorical distribution (Gumbel-Top-k trick; [180]).

Stochastic Beam Search (SBS) [107] applies the Gumbel-Top-k trick to a sequence model, without instantiating the entire search tree, by using top-down sampling. At each step, the beam search expands the k partial sequences that have the highest perturbed log-probabilities, where the Gumbel-perturbed log-probabilities are sampled conditioned on their maximum being equal to their parent perturbed log-probability.

Given a partial sequence S with log-probability ϕ_S, for each extension $S' \in \mathit{children}(S)$, we start by computing their Gumbel-perturbed log-probabilities $G_{\phi_{S'}} \sim \mathit{Gumbel}(\phi_{S'})$ independently, and set $Z = max_{S'} G_{\phi_{S'}}$. Then,

$$\tilde{G}_{\phi_{S'}} = -\log(\exp(-G_{\phi_S}) - \exp(Z) + \exp(-G_{\phi_{S'}}))$$

is a set of Gumbels with a maximum equal to G_{ϕ_S}. Algorithm 5.6 presents pseudo-code of SBS.[3]

[3]The pseudo-code is provided to conceptually explain SBS. In the original paper [107], Kool et al. describe a numerically stable implementation of SBS.

Algorithm 5.6 Stochastic Beam Search [107]

function SBS(conditional probability $\log P_\theta$, beam width $\mathcal{B}$)
  $beam \leftarrow \{\}$
  $beam \leftarrow beam \cup (y^N = \varnothing, \phi_N = 0, G_{\phi_N} = 0)$ ▷ Initialize beam with empty sequence
  for $t = 1..T$ **do**
    $expansions \leftarrow \{\}$
    for $(y^S, \phi_S, G_{\phi_S}) \in beams$ **do** ▷ For each (partial) sequence in beam
      $Z \leftarrow -\infty$
      for $S' \in children(S)$ **do** ▷ For each one-token extension of the sequence
        $\phi_{S'} \leftarrow \phi_S + \log P_\theta(y^{S'} \mid y^S)$
        $G_{\phi_{S'}} \sim Gumbel(\phi_{S'})$ ▷ Compute Gumbel-perturbed log-probabilities
        $Z \leftarrow \max(Z, G_{\phi_{S'}})$
      for $S' \in children(S)$ **do**
        $\tilde{G}_{\phi_{S'}} = -\log(\exp(-G_{\phi_S}) - \exp(Z) + \exp(-G_{\phi_{S'}}))$ ▷ $\tilde{G}_{\phi_{S'}}$ conditioned on maximum G_{ϕ_S}
        $expansions \leftarrow expansions \cup (y^{S'}, \phi_{S'}, \tilde{G}_{\phi_{S'}})$
    $beams \leftarrow Top - \mathcal{B}(expansions, \tilde{G})$

5.3.3 Beam Search Enhancements

In this section, we describe notable enhancements and commonly used techniques to improve beam search performance.

Length Normalization

In many natural language generation tasks, it is common to apply length normalization to the (partial) sequences in beam search [105, 49]. When using length normalization, the score of a (partial) sequence is its likelihood divided by the length of the sequence,

$$score(y) = \frac{p_\theta(y|x)}{|y|}, \tag{5.15}$$

where $|y|$ denotes the length of sequence y.

In Google's NMT system, Wu et al. [202] improved over Eq. (5.15) by dividing the likelihood by $|y|^\alpha$ where $\alpha \in [0.6, 0.7]$. Eventually, Wu et al. [202] used the following length normalization scheme that was empirically found to be better:

$$score(y) = \frac{p_\theta(y|x)}{\left(\frac{5+|y|}{5+1}\right)^\alpha}.$$

Beam Search Re-ranking

Beam search re-ranking (or re-scoring) [171, 157] refers to the following two-step process:

1. First, a beam-search is run to compute the M-best list.

2. Then, sequences in the M-best list are re-ranked according to a different scoring function.

In this setting, the beam search is used to generate a set of high quality candidate sequences, however the final ranking of these sequences is done using a different scoring function that may lead to a different top-1 sequence being returned.

Common examples for additional scoring functions are global scoring functions that can only score complete sequences or additional trained models that can be utilized for re-ranking the beam search M-best list of a base model [162, 137]. In order to use a set of additional scoring models $m'_1, m'_2, ..., m'_z$ to re-rank the M-best list of a base model m_{base}, we can use the following procedure:

1. Use beam search to generate the M-best list of promising candidates from m_{base}.
2. Compute the score of each sequence in the M-best list according to each of the z additional models.
3. Combine the score of all models by a simple average or a weighted combination with learned weights (e.g., using MIRA [19]).
4. Re-rank the M-best list according to the new combined scores and return the top candidate in the re-ranked list.

Previous works considered a variety of additional models, including ensemble models [171], right-to-left models [157], etc.

Constrained Beam Search

In recent years, several works have considered incorporating constraints into beam search. We review notable extensions of beam search that incorporate different types of constraints that the solution sequences must satisfy.

Anderson et al. [2] proposed a constrained beam search that forces inclusion of selected tokens in the output. The authors consider two types of constraints: (1) token sets that at least one token of each set must appear in the output; (2) sub-sequences that must appear in the output. The authors construct a finite-state machine (FSM) that is able to keep track of which constraints are currently satisfied by each partial sequence during the beam search decoding. For each FSM state s and each decoding step t, a corresponding search beam is maintained containing at most $\mathcal{B}$ partial sequences. In each step, the FSM state-transition function determines for each potential expansion of a partial sequence the corresponding state in the FSM. The accepting state represent the conjunction of all constraints and the algorithm terminates when an accepting state contains a completed sequence. In a later work, Hasler et al. [74] adapted Anderson et al.'s [2] constrained beam search to machine translation and proposed employing alignment information between target-side constraints and their corresponding source words to reduce the computational complexity.

Hokamp and Liu [91] proposed *grid beam search (GBS)*, a variant of beam search that must satisfy constraints in the form of sub-sequences that must appear in the solution sequences. To generate sequences that satisfy the constraints, GBS maintains a grid-based data structure of beams where the beams are indexed by t, the time step of the search, and c, the number of constraint tokens that are covered by the partial sequences in this beam. The partial sequences in each beam are separated into two types:

1. *open* sequences that can either generate tokens from the model's distribution or generate a token that starts an available constraint.
2. *closed* sequences that can only generate the next token for a currently unfinished constraint.

The beams at the top level of the grid, i.e., beams for which c is equal to the total number of tokens in all constraints, contain partial sequences that satisfy all the constraints and can be returned once they are completed. Post and Vilar [144] proposed to change GBS such that the beam slots are dynamically allocated across the constraints, reducing the complexity of the algorithm to $T \times \mathcal{B}$, i.e., the complexity does not scale with the number of constraints.

Building on the background presented in this chapter, in Chapter 6 and Chapter 7 we present empirical models for the search behavior of beam search in neural sequence decoding. In Chapter 6 we analyze the problem of performance degradation in beam search with large beam widths [105] and in Chapter 7 we study the problem difficulty of beam search in goal-oriented neural sequence decoding.

Chapter 6

Empirical Analysis of Beam Search Performance Degradation in Neural Sequence Decoding

In this chapter, we develop an empirical model for the behavior of beam search in neural sequence decoding. Our model is based on the notion of search discrepancies [73], adapted from combinatorial search. Consistent with our thesis statement, we show that this model can explain the well-known problem of performance degradation in beam search with large beams [105] and demonstrate how this model can be used to devise heuristic techniques that eliminate the performance degradation.

6.1 Introduction

Neural sequence models are among the most popular tools for modeling sequential data and have been applied to a range of applications including machine translation [49], summarization [21], image captioning [183], and conversation modeling [182]. The most commonly used inference algorithm for decoding neural sequence models is beam search, as discussed in Chapter 5.

Recently, several works have reported the problem of performance degradation in beam search. In machine translation, Koehn and Knowles [105] found that beam search "only improves translation for narrow beams and deteriorates when exposed to a larger search space" (p. 28). While length-normalization can alleviate the problem somewhat, it does not eliminate it. Koehn and Knowles [105] chose this problem as one of six central challenges in machine translation.

To explain the performance degradation in length-normalized machine translation models, Ott et al. [138] proposed the existence of training pairs in which the target is a copy of the source. For larger beams, they found that more of the predictions are "copies"[1] and that pruning these copies from the beam search results reduces the performance degradation.

In image captioning, Vinyals et al. [183] observed performance degradation for wider beams and highlighted the use of a narrower beam search as one of the most significant improvements in their model. They hypothesized that the degradation is either due to overfitting or that the objective used in training

[1] "Copies" are predictions that share at least 50% of their unigrams with their source [138].

(likelihood) is not aligned with human judgement. Their analysis found that wider beams exhibited more predictions that repeat training captions and fewer novel ones. This observation is used to support the hypothesis that the model is overfitted and therefore they propose the use of smaller beam width as "another way to regularize" (Vinyals et al. [183], p. 660).

In this chapter, we study the phenomenon of beam search performance degradation in neural sequence models. Based on an extensive empirical analysis across different neural sequence decoding tasks, we develop an explanatory model of the beam search performance degradation. We make the following contributions:

- We introduce an explanatory model of the beam search performance degradation in neural sequence models that is based on the concept of *search discrepancies* that represent deviations from greedy choices.
- We conduct an extensive empirical study of the distribution of search discrepancies and show that increasing the beam width leads to solutions with more and larger discrepancies early in the sequence. These sequences often have lower evaluation score, leading to the observed performance degradation. As we increase the beam width, the differences in the distribution of discrepancies that are associated with improved vs. degraded solutions grow substantially.
- We show, empirically, that our explanatory model generalizes previous observations on "copies" in machine translation and predictions that repeat training set targets in image captioning and accounts for more of the degraded predictions.
- Based on our empirical analysis, we propose two heuristics for constraining the beam search from considering large search discrepancies and show that these heuristics eliminate the performance degradation.

The work in this chapter is based on the publication [24].

6.1.1 Organization

In Section 6.2, we provide the necessary notation and formally define search discrepancies in neural sequence decoding using beam search. In Section 6.3, we describe the experimental setup we use in our analysis. In Section 6.4, we conduct an extensive empirical analysis of the search discrepancies in neural sequence decoding using beam search. Based on the results of the analysis, in Section 6.4.7 we hypothesize that the performance degradation is due to early and large search discrepancies. Then, in Section 6.5, we empirically verify our hypothesis by introducing two heuristics for constraining the beam search from considering large search discrepancies and show that they eliminate the performance degradation. Finally, in Section 6.6 we present a detailed discussion on the implications of our results and the connections to other related works and in Section 6.7 we summarize the chapter.

6.2 Preliminaries

In this section we present the notation and definitions used in our work. Chapter 5 includes detailed background on neural sequence models and beam search. For convenience, we repeat the main definitions that are relevant for this chapter. Then, in Section 6.2.1, we present the definition of search discrepancies we use in our empirical analysis.

Neural Sequence Models A sequence model is a factorized parametric distribution over sequences. The parameter θ defines the probability $p_\theta(y_t|x, y_{1:t-1})$ of the next token y_t, conditioned on the input x and the partial sequence $y_{1:t-1}$. A sequence model defines a valid probability distribution over both partial and complete sequences, and the total probability of a sequence $y_{1:t}$ conditioned on input x can be computed using the chain rule,

$$p_\theta(y_{1:t}|x) = p_\theta(y_t|x, y_{1:t-1}) \cdot p_\theta(y_{1:t-1}|x) = \prod_{t'=1}^{t} p_\theta(y_{t'}|x, y_{1:t'-1}). \tag{6.1}$$

Beam Search Beam search is a limited-width breadth-first search that is used as an approximation to finding the (single) sequence y that maximizes Eq. (6.1), or as a way to obtain a set of high-probability sequences from the model. Following Vijayakumar et al. [181], we denote the set of $\mathcal{B}$ solutions held by the beam search at time step $t-1$ as $Y_{[t-1]}=\{\mathrm{y}_{1,[t-1]}, ..., \mathrm{y}_{B,[t-1]}\}$. At each step, beam search selects the top scoring $\mathcal{B}$ candidates from the set of all possible one token extensions of its beams $\mathcal{Y}_t=\{\mathrm{y}_{[t]} \mid \mathrm{y}_{[t-1]} \in Y_{[t-1]} \wedge y_t \in \mathcal{V}\}$. Formally, the beam search candidates are updated as follows:

$$\begin{aligned} Y_{[t]} = \underset{\mathrm{y}_{1,[t]},\ldots,\mathrm{y}_{B,[t]} \in \mathcal{Y}_t}{\arg\max} & \sum_{b \in [1..B]} \log P_\theta(\mathrm{y}_{b,[t]} \mid \mathrm{x}) \\ s.t. \quad & \mathrm{y}_i \neq \mathrm{y}_j \quad \forall i \neq j; \quad i, j \in [1..B] \end{aligned} \tag{6.2}$$

6.2.1 Search Discrepancies in Neural Sequence Generation

In combinatorial search, a search discrepancy is a decision made by the search algorithm that is not the most highly rated one according to the heuristic [73]. Search discrepancies have been used as the basis of different combinatorial search algorithms such as limited discrepancy search [73, 108], depth-bound discrepancy search [184], discrepancy-bounded depth-first search [8], and beam search using limited discrepancy backtracking (BULB) [47]. However, these methods are aimed at finding feasible solutions in a constrained combinatorial optimization or path-finding problems.

In the context of search for neural sequence generation, we define a *search discrepancy* as extending a partial sequence with a token that is not the most probable one. For each search discrepancy, we denote the difference in log-probability between the most likely token and the chosen token as *discrepancy gap*. We formally define *search discrepancy* and *discrepancy gap* as follows:

Definition 21 (Search Discrepancy) *Consider a sequence model parameterized by θ and an input context x. A sequence* y *is considered to have a search discrepancy at time step t if*

$$\log P_\theta(y_t \mid \mathrm{x};\{y_0,\ldots,y_{t-1}\}) < \max_{y \in \mathcal{V}} \log P_\theta(y \mid \mathrm{x};\{y_0,\ldots,y_{t-1}\})$$

Definition 22 (Discrepancy Gap) *Consider a sequence model parameterized by θ and an input context x. At time step t, the discrepancy gap is defined as*

$$\max_{y \in \mathcal{V}} \log P_\theta(y \mid \mathrm{x};\{y_0,\ldots,y_{t-1}\}) - \log P_\theta(y_t \mid \mathrm{x};\{y_0,\ldots,y_{t-1}\})$$

Note that if there is no discrepancy at time step t then $\log P_\theta(y \mid \mathrm{x};\{y_0,\ldots,y_{t-1}\}) = \log P_\theta(y_t \mid \mathrm{x};\{y_0,\ldots,y_{t-1}\})$ and the token at index t is considered to have a discrepancy gap of zero.

To demonstrate how the discrepancy gap is computed, Figure 6.1 shows the extension of a partial hypothesis in machine translation. The candidate with the highest conditional probability has a discrepancy gap of zero, by definition, while the gap of the other candidates is the difference in log-probability.

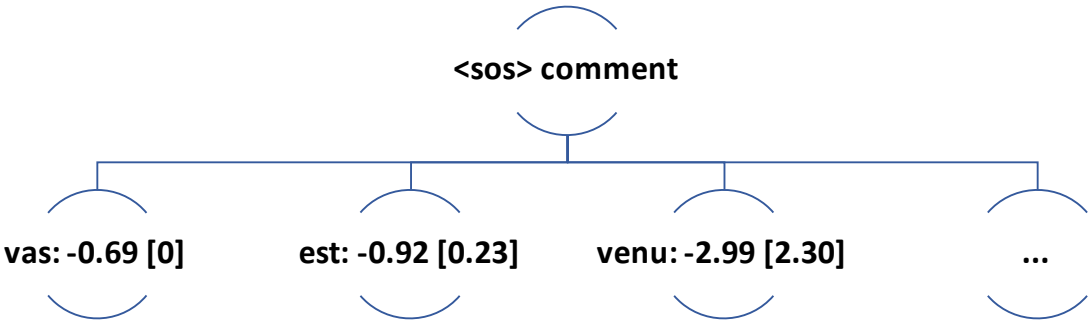


Figure 6.1: Example: expanding a partial hypothesis in the translation of "How are you?" to French. Discrepancy gap in brackets.

In this work we investigate if the notion of search discrepancies we adapt from combinatorial search can help explain the performance degradation in neural sequence decoding using beam search.

6.3 Experimental Setup

We perform an extensive empirical evaluation over multiple tasks, models, datasets, and evaluation metrics. Following is a description of the experimental setup for each task.

6.3.1 Sequence Decoding Tasks

Machine Translation

Neural machine translation (NMT) is an end-to-end approach to machine translation [4]. Most of the approaches to neural machine translation are sequence-to-sequence models [173], with an encoder and a decoder for each language. The encoder network reads a source sentence and encodes it into a fixed-length hidden vector. The decoder takes in the encoded vector and outputs a translation. The whole encoder–decoder system is jointly trained to maximize the probability of a correct translation given a source sentence.

We use Gehring et al.'s [49] model that includes a convolutional encoder and a recurrent decoder based on LSTM, implemented in the *fairseq-py* toolkit [139]. We present results for two models:

1. English-to-French translation model, trained on the WMT'14 En-Fr dataset and evaluated on the newstest2014 En-Fr dataset.

2. English-to-German translation model, trained on the WMT'14 En-De dataset and evaluated on the newstest2014 En-De dataset.

Both models use a vocabulary that is based on byte pair encoding (BPE) [158] that allows for open-vocabulary translation by encoding rare and unknown words as sequences of subword units.

Abstractive Summarization

Automatic Summarization is the task of generating a condensed version of a passage while preserving its meaning. Extractive summarization systems generate summaries by extracting important segments from the source text and putting them together to form a summary. Abstractive summarization systems generate summaries from scratch without being constrained to using phrases from the source text.

We use a sequence-to-sequence abstractive summarization model [21] that includes two-layer LSTMs for the encoder and the decoder, implemented in the OpenNMT toolkit [104]. The model is trained and evaluated using Rush et al.'s [154] test split of the Gigaword corpus [68].

Image Captioning

Image captioning is the task of automatically describing the content of an image. Vinyals et al. [183] proposed a end-to-end approach, inspired by neural machine translation systems. Their model includes a deep convolution neural network encoder that takes in an image and produces a rich representation of the image by embedding it in a fixed-length vector. This vector is used as the input to a recurrent LSTM decoder.

The model is trained on the MSCOCO dataset [119]. We evaluate the model using a test set of 5000 images based on Karpathy and Fei Fei's [101] splits.

In machine translation and summarization, we apply length normalization on the hypotheses log-likelihood, as it was shown to reduce the performance degradation by not prioritizing short sentences [105, 49]. For image captioning, consistent with previous works, we do not use length normalization (we also found that such normalization reduces the overall performance).

6.3.2 Evaluation Metrics

While beam search finds the (approximately) most probable sequence, the quality of a sequence is evaluated based on human references using a task-specific evaluation metric.

BLEU-N

For machine translation and image captioning, we use the bilingual evaluation understudy (BLEU) score [140] to evaluate the sequence returned by the beam search. Given a candidate sentence and a reference sentence, BLEU-N is a geometric average of precision over 1- to N-grams multiplied by a brevity penalty for short sentences,

$$\text{BLEU-}N = BP \cdot exp(\sum_{n=1}^{N} w_n \log p_n),$$

where p_n is the n-gram precision and the brevity penalty BP is computed based on the reference length r and the candidate length c:

$$BP = \begin{cases} 1 & \text{if } c > r \\ \mathrm{e}^{(1-\frac{r}{c})} & \text{if } c \leq r \end{cases}$$

As in recent literature, we present results for BLEU-4. Corpus-level BLEU is reported without smoothing, while for sentence-level BLEU we use smoothed n-gram counts for $n > 1$ [118].

ROUGE-N

For summarization, we use the recall-oriented understudy for gisting evaluation (ROUGE) score [116]. ROUGE-N is the N-gram recall between candidate summary c and a set of reference summaries $R = \{r_1, r_2, ..., r_m\}$, such that ROUGE-1 represents unigram recall, ROUGE-2 represents bigram recall, etc. In recent works, ROUGE-N recall has been replaced by F_1 scores of ROUGE-N [135, 21]. In addition, Lin and Och [117] proposed ROUGE-L that is based on the longest common subsequence (LCS) between a candidate and a reference. Given a candidate summary c of length l_c and a reference summary r of length l_r, ROUGE-L is defined as the LCS-based F-measure:

$$R_{LCS} = \frac{LCS(c,r)}{l_c}$$

$$P_{LCS} = \frac{LCS(c,r)}{l_r}$$

$$F_{LCS} = \frac{(1+\beta^2)R_{LCS}P_{LCS}}{R_{LCS} + \beta^2 P_{LCS}}$$

where $LCS(c,r)$ is the longest common subsequence of c and r and $\beta = P_{LCS}/R_{LCS}$. We report the ROUGE-1 F_1 score, however similar trends were observed for ROUGE-L.

6.4 Empirical Analysis of Search Discrepancies in Beam Search

In this section, we present an empirical analysis of the search discrepancies in neural sequence decoding using beam search. We analyze and compare the most likely hypotheses found by a beam search for the following beam widths: {1, 3, 5, 25, 100, 250}.

6.4.1 Baseline Results

Table 6.1 presents the performance of beam search with different beam widths, based on the chosen evaluation metrics. The performance degradation for larger beam widths appears for all tested tasks based on their task-specific evaluation metric. These results are consistent with the existing reports of such performance degradation [105, 138, 183].

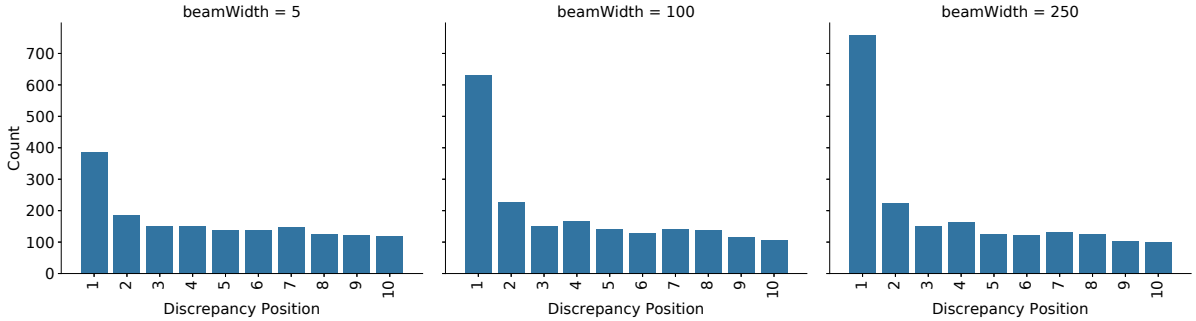
Table 6.1: Baseline results for different beam widths (higher values are better, best results in bold).

Task	Dataset	Size	Metric	B=1	B=3	B=5	B=25	B=100	B=250
Translation	En-De	3003	BLEU4	25.27	26.00	**26.11**	25.11	23.09	21.38
	En-Fr	3003	BLEU4	40.15	40.77	**40.83**	40.52	38.64	35.03
Summariz.	Gigaword	1751	ROUGE-1	33.56	**34.22**	34.16	34.01	33.67	33.23
Captioning	MSCOCO	5000	BLEU4	29.66	**32.36**	31.96	30.04	29.87	29.79

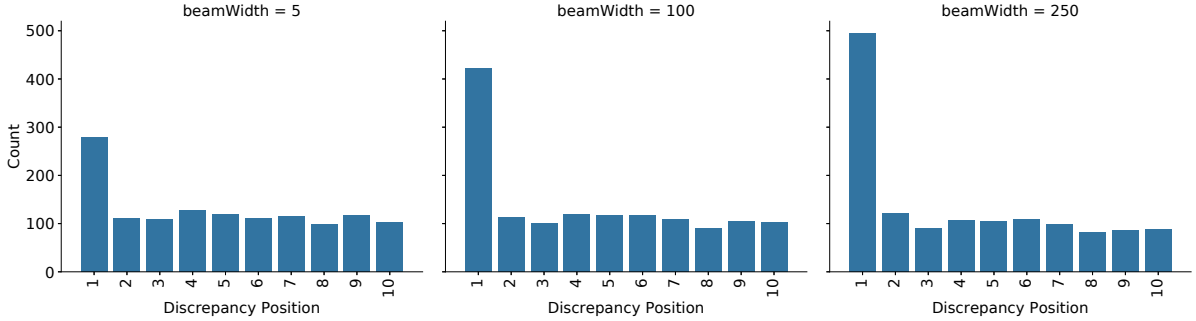
6.4.2 The Distribution of Search Discrepancies

Figure 6.2a shows the number of discrepancies per position (index) for the most likely hypotheses generated by a beam search on the WMT'14 En-De test set for different beam widths. All graphs are

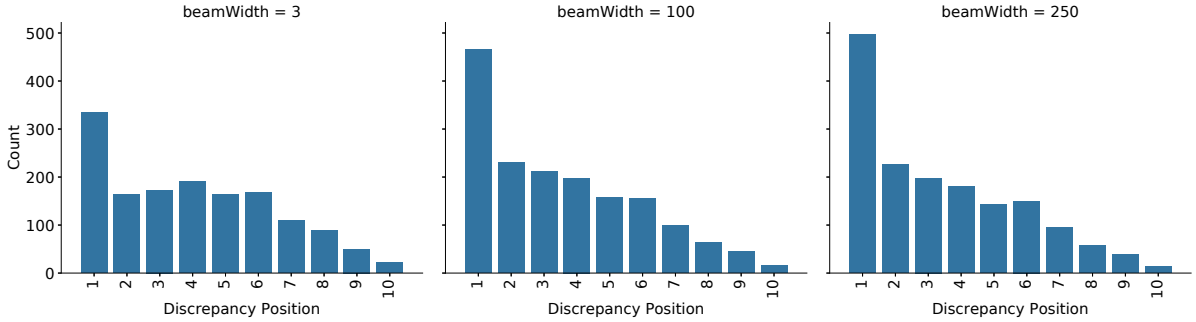
based on the same number of solutions, however the total number of discrepancies in the generated solutions is not necessarily the same for different beam widths. In general, the majority of discrepancies happen in early positions. More interestingly, for larger beams, the number of early discrepancies grows significantly while the number of later discrepancies stays approximately constant. Larger beams allow the search to find solutions with higher overall probability by exploring less probable early tokens, however they do not seem to lead to more probable sequences that share a prefix with solutions found for a smaller beam width.



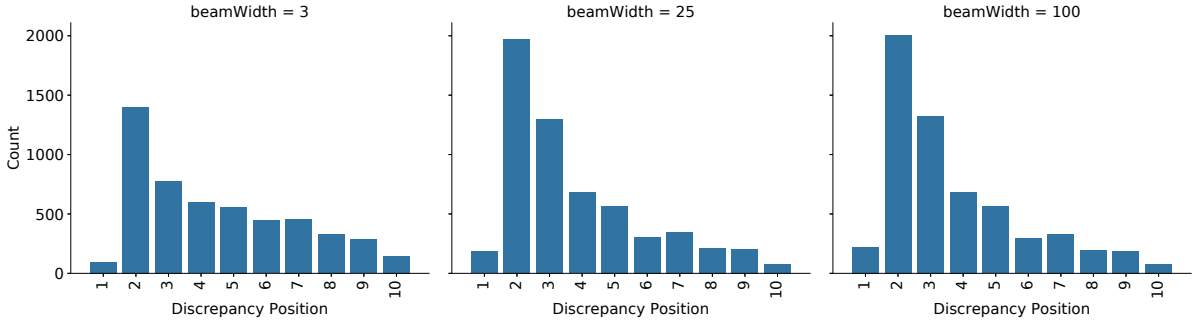
(a) WMT'14 En-De: Distribution of discrepancy positions for different beam widths.



(b) WMT'14 En-Fr: Distribution of discrepancy positions for different beam widths.



(c) Gigaword: Distribution of discrepancy positions for different beam widths.



(d) MSCOCO: Distribution of discrepancy positions for different beam widths.

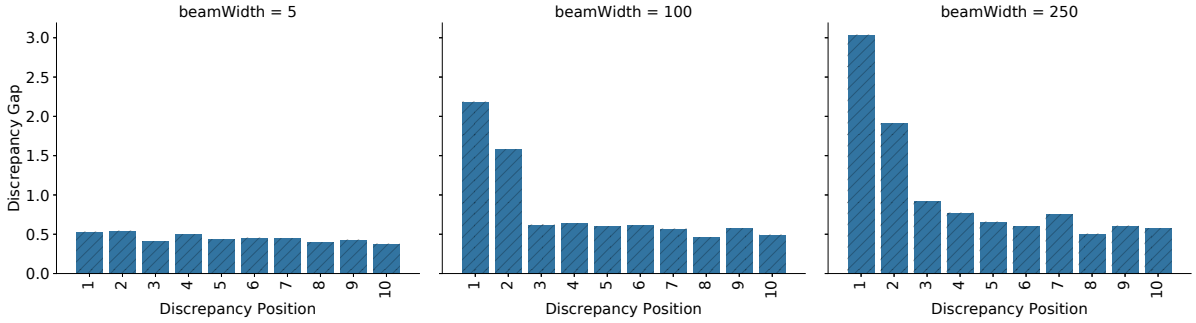
Figure 6.2: Distribution of discrepancy positions for different beam widths.

Figure 6.2b, Figure 6.2c, and Figure 6.2d show similar results for WMT'14 En-Fr translation, Gigaword summarization, and MSCOCO image captioning, respectively. For image captioning (MSCOCO), we find the majority of early search discrepancies appear on the second token due to the first token being

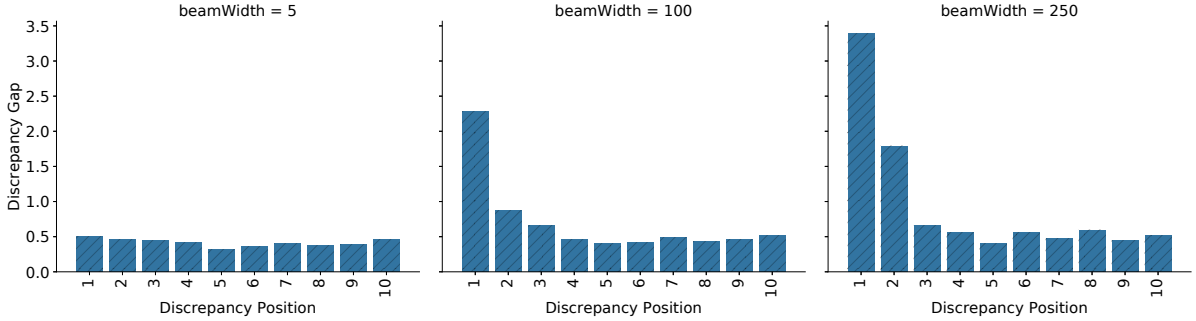
"a" with high probability in almost all sentences (in greedy search, for example, 99% of the generated captions start with "a").

Next, we analyze the discrepancy gap vs. sequence position. Figure 6.3a presents the mean discrepancy gap per position for WMT'14 En-De for different beam widths. Again, we can see that the differences are mainly in the early positions: for larger beams, the search tends to find solutions with larger early discrepancy gap, i.e., the early tokens are relatively less likely. The gap of the other tokens remains similar.

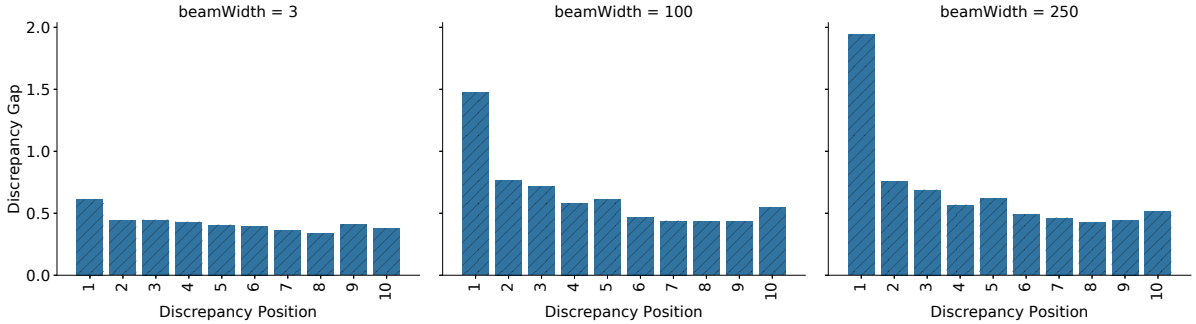
Figure 6.3b, Figure 6.3c, and Figure 6.3d show similar results for WMT'14 En-Fr translation, Gigaword summarization, and MSCOCO image captioning, respectively.



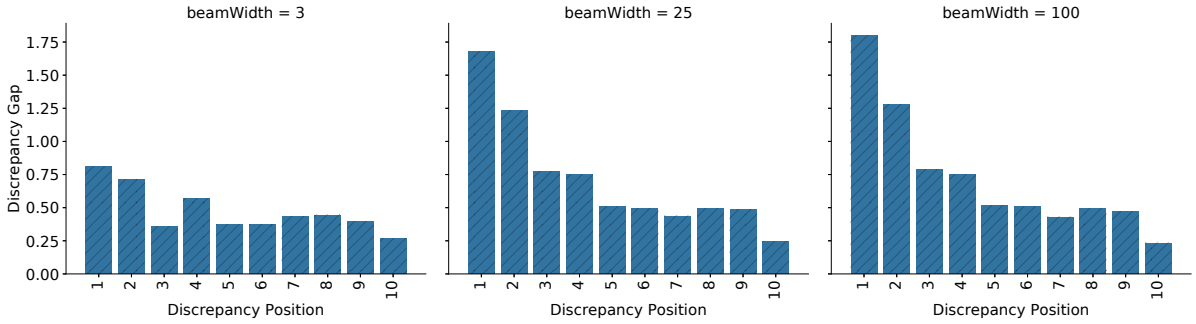
(a) WMT'14 En-De: Mean discrepancy gap per position for different beam widths.



(b) WMT'14 En-Fr: Mean discrepancy gap per position for different beam widths.



(c) Gigaword: Mean discrepancy gap per position for different beam widths.



(d) MSCOCO: Mean discrepancy gap per position for different beam widths.

Figure 6.3: Mean discrepancy gap per position for different beam widths.

The increase in number and size of early discrepancies for larger beams means that the search manages to find solutions with higher overall probability when starting from a large discrepancy. However, these solutions are not necessarily better according to the evaluation metric. The observed performance

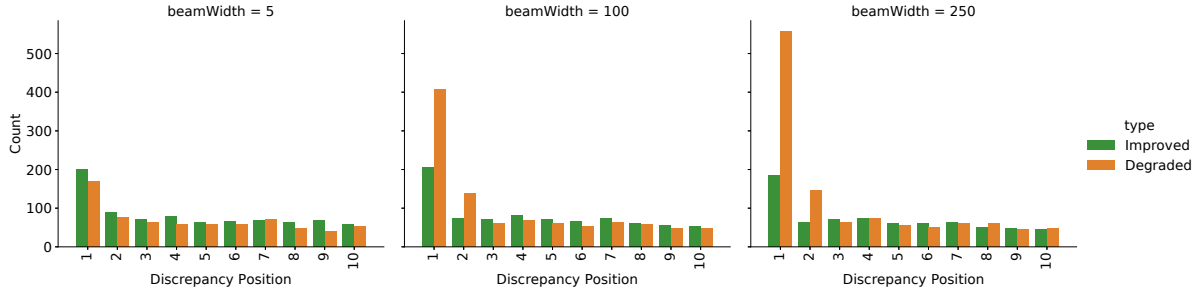
degradation in Section 6.4.1 suggests that the more probable solutions found by larger beams are, in fact, worse in terms of the evaluation metric. Identifying discrepancies that are likely to lead to a worse solution is therefore a key task in addressing the performance degradation.

6.4.3 Discrepancies in Improved vs. Degraded Solutions

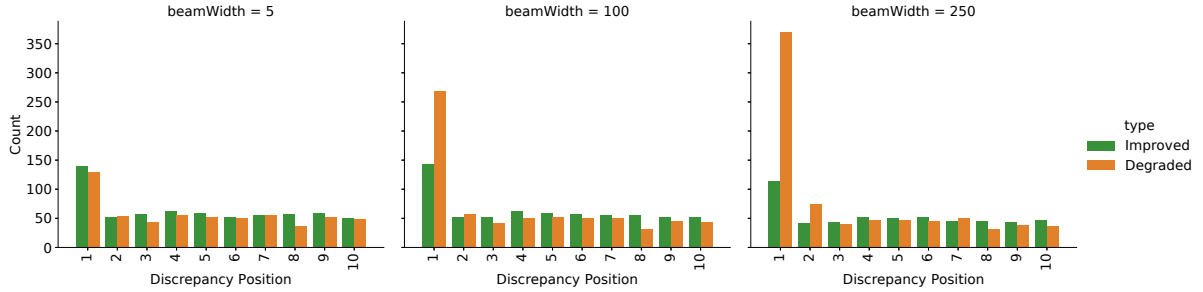
We now compare the solutions generated by a greedy search with the solutions generated by beam search with different widths. We then analyze the discrepancies in solutions that were improved by increasing the beam width (based on the evaluation metric and with respect to the greedy solution) vs. solutions that were degraded.

Figure 6.4a shows the number of discrepancies per position for WMT'14 En-De, comparing solutions that were improved vs. solutions that were degraded. For B=5 there are 386 solutions in which the first token is not based on a greedy decision. Of those, 200 have a better evaluation than the greedy solution and 169 have a worse evaluation. However, as we increase the beam width, the increase in early discrepancies observed in Figure 6.2a is associated almost entirely with degraded solutions. This result leads to the observed performance degradation for larger beam widths.

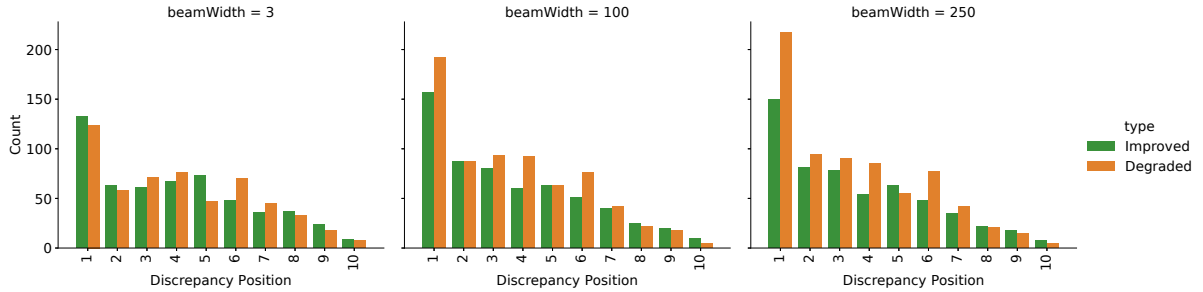
Figure 6.4b, Figure 6.4c, and Figure 6.4d show similar results for WMT'14 En-Fr translation, Gigaword summarization, and MSCOCO image captioning, respectively.



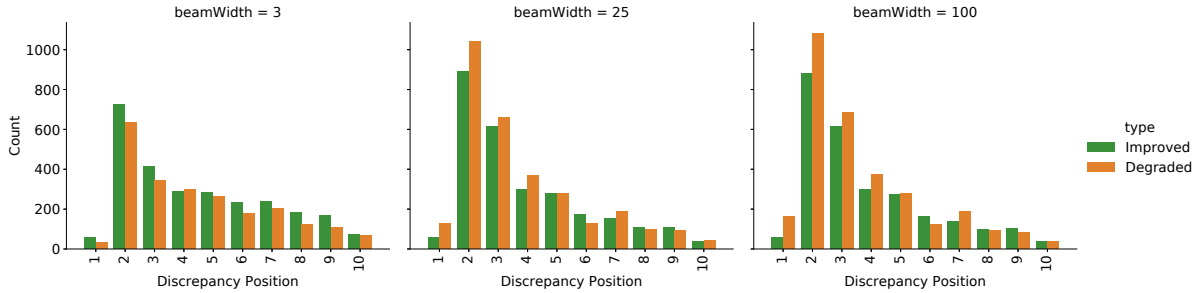
(a) WMT'14 En-De: Distribution of discrepancy positions for improved vs. degraded solutions.



(b) WMT'14 En-Fr: Distribution of discrepancy positions for different beam widths.



(c) Gigaword: Distribution of discrepancy positions for different beam widths.

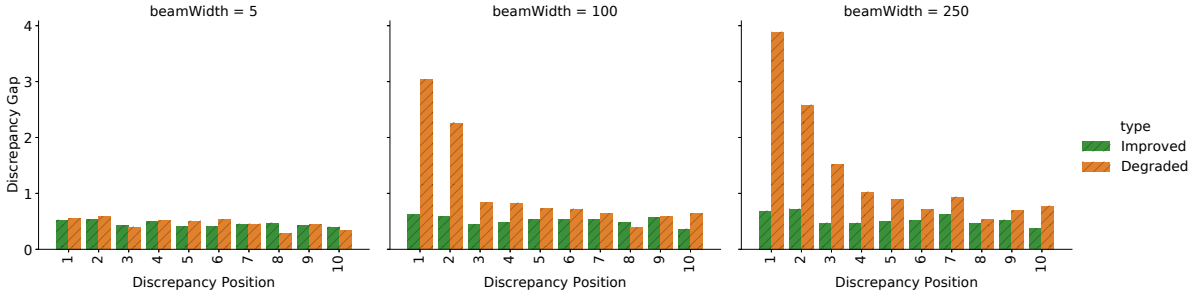


(d) MSCOCO: Distribution of discrepancy positions for different beam widths.

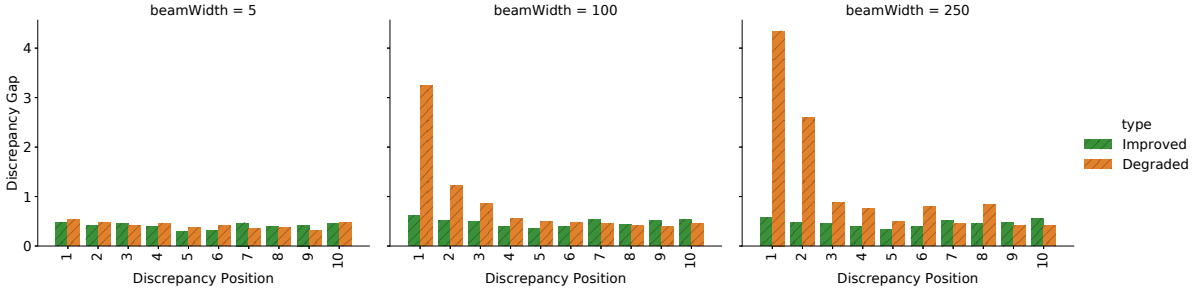
Figure 6.4: Distribution of discrepancy positions for improved vs. degraded solutions.

Next, we compare the discrepancy gaps in degraded vs. improved solutions. Figure 6.5a presents the mean discrepancy gap per position in the WMT'14 En-De dataset, for both the improved and the degraded solutions. Interestingly, we find that the additional early discrepancies that are associated with degraded solutions tend to have a much higher discrepancy gap compared to the ones associated with improved solutions.

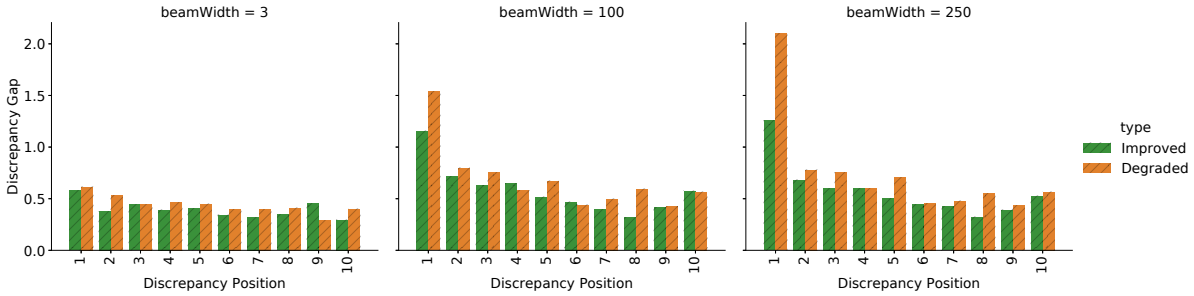
Figure 6.5b, Figure 6.5c, and Figure 6.5d show similar results for WMT'14 En-Fr translation, Gigaword summarization, and MSCOCO image captioning, respectively.



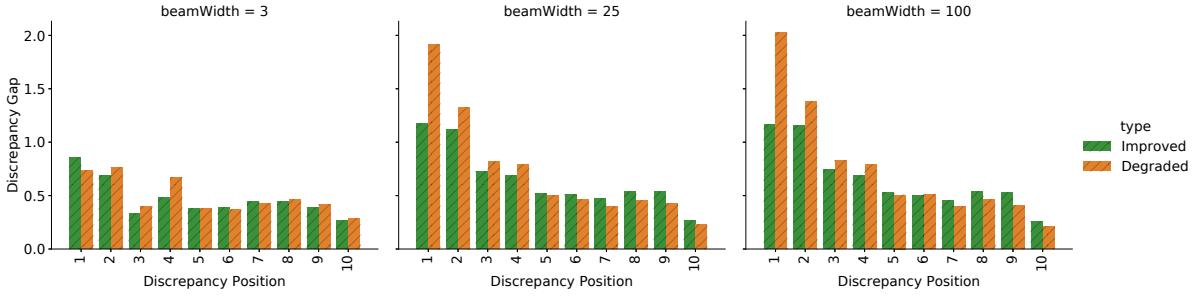
(a) WMT'14 En-De: Mean discrepancy gap per position for improved vs. degraded solutions.



(b) WMT'14 En-Fr: Mean discrepancy gap per position for different beam widths.



(c) Gigaword: Mean discrepancy gap per position for different beam widths.



(d) MSCOCO: Mean discrepancy gap per position for different beam widths.

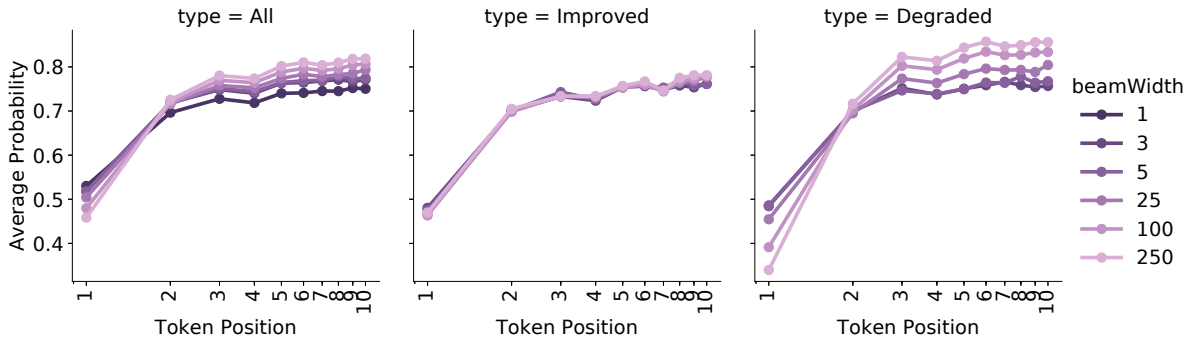
Figure 6.5: Mean discrepancy gap per position for improved vs. degraded solutions.

6.4.4 Discrepancies and the Most Likely Hypothesis

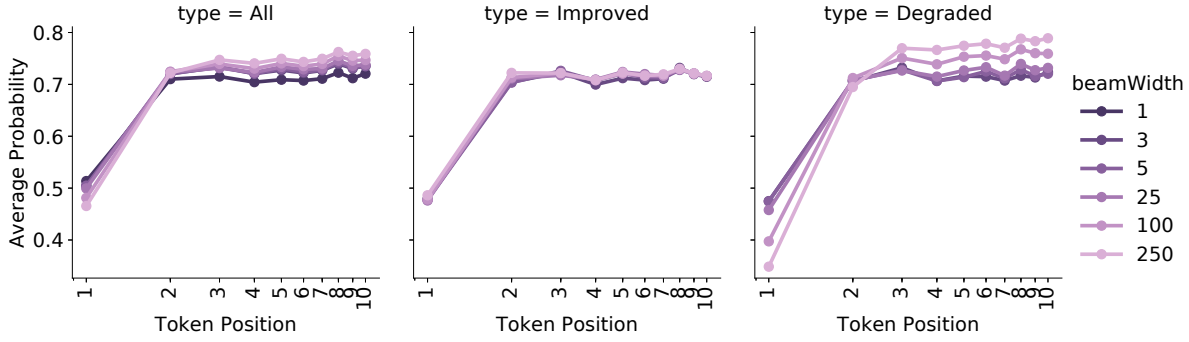
In order for a sequence with an early large discrepancy to be selected by a beam search as (approximately) the most likely hypothesis, it must be followed by tokens with higher (conditional) probability. Figure 6.6a shows the average (conditional) token probability for WMT'14 En-De (we use log-scale on the x axis to highlight the early positions). For larger beams, the average probability of early tokens decreases (due to larger discrepancy gaps) while the average probability of later tokens increases explaining the overall

higher probability.

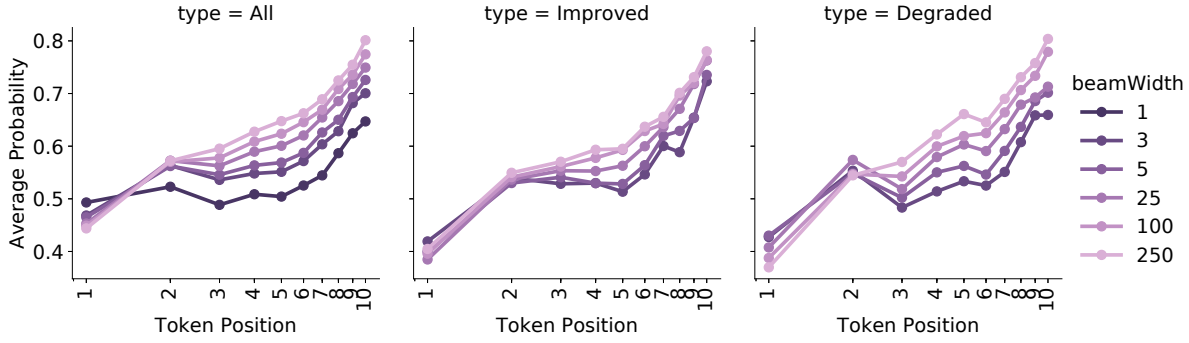
Figure 6.6a also shows the same graph for the improved vs. degraded solutions (compared to greedy search). For improved solutions, we do not see significant change as we increase the beam width. For degraded solutions, however, as we increase the beam width we find more and more early discrepancies that lead to an overall higher probability but a worse evaluation metric value.



(a) WMT'14 En-De: Average token probability per position for different beam widths.



(b) WMT'14 En-Fr: Average token probability per position for different beam widths.



(c) Gigaword: Average token probability per position for different beam widths.

Figure 6.6: Average token probability per position for different beam widths.

Figure 6.6b and Figure 6.6c show similar analyses for WMT'14 En-Fr and Gigaword summarization. Again, we find that for larger beams, the average probability of early tokens decreases while the average probability of later tokens increases explaining the overall higher probability. We also find that the changes in the tokens' average probability for increased beam width are larger in the case of degraded solutions than for improved solutions.

For image captioning, length normalization is not used. We therefore need to compare the product

of token probabilities rather than the average token probabilities. Table 6.2 shows the mean sum of log-probabilities (that corresponds to the product of probabilities) of the first two tokens and the mean sum of log-probabilities of the rest of the tokens. As we increase the beam width, the probability of the early tokens decreases, while the probability of the rest of the tokens increases. Furthermore, the change in probabilities is larger for degraded sequences compared to improved sequences.

Table 6.2: MSCOCO: Log-probability of the early (first two) tokens vs. the log-probability of the rest.

	All		Improved		Degraded	
Beam	Early	Rest	Early	Rest	Early	Rest
B=1	-1.48	-6.98	N/A	N/A	N/A	N/A
B=3	-1.68	-5.11	-1.76	-5.32	-1.78	-5.09
B=25	-2.02	-4.09	-2.04	-4.26	-2.21	-3.99
B=100	-2.07	-4.02	-2.06	-4.21	-2.30	-3.91
B=250	-2.08	-4.01	-2.06	-4.20	-2.31	-3.90

6.4.5 Generalizing Copies and Training Set Predictions

Ott et al. [138] observed the pattern observed above for copies, i.e., they have low first token probability and higher probabilities for subsequent tokens. Our analysis accounts for this behavior and suggests that copies are one instance of a more general pattern that leads to degraded sequences. In this section, we show that our analysis generalizes copies, as well as training set predictions in image captioning, and even accounts for additional degraded sequences.

Table 6.3 shows the number of predictions that were copies in machine translation and training set predictions in summarization and image captioning. For larger beams, the number of copies and training set predictions grows. Table 6.3 also reports the mean discrepancy gap of the first token (second token for MSCOCO, see Section 6.4.2). As our analysis predicts, the early gap of these predictions also grows significantly.

Table 6.3: Number of copies and training set examples and the average first token discrepancy gap.

		B=1	B=3	B=5	B=25	B=100	B=250
En-De	# Copies	23	40	49	179	385	567
En-De	First token gap (copies)	0.0	0.12	0.28	1.79	3.05	3.71
En-De	First token gap (all)	0.0	0.05	0.07	0.18	0.46	0.77
En-Fr	# Copies	25	28	41	89	227	358
En-Fr	First token gap (copies)	0.0	0.12	0.31	1.69	3.68	4.38
En-Fr	First token gap (all)	0.0	0.04	0.05	0.10	0.32	0.60
Gigaword	# Training set predictions	81	86	86	115	163	224
Gigaword	First token gap (train pred.)	0.0	0.07	0.07	0.98	1.84	2.61
Gigaword	First token gap (all)	0.0	0.12	0.12	0.29	0.39	0.55
MSCOCO	# Training set predictions	163	260	371	588	582	576
MSCOCO	Second token gap (train pred.)	0.0	0.39	0.87	1.76	1.82	1.82
MSCOCO	Second token gap (all)	0.0	0.20	0.29	0.49	0.51	0.51

Note that copies and training set predictions only partially account for the performance degradation. In WMT'14 En-De translation with B=25, we find that copies account for $\approx 40\%$ of degraded solutions with first token gap. In Gigaword summarization with a similar beam width, training set examples account for $\approx 68\%$ of degraded solutions with first token gap. Furthermore, in MSCOCO, since many of the improved sequences are training set captions, eliminating them all together is not desired. Instead, we are interested in avoiding only the training captions in the larger beams that lead to the performance degradation. These, as Table 6.3 shows, have a larger difference in the discrepancy gap.

6.4.6 An Illustrative Example

Consider the following example of training set predictions in Gigaword. As we increase the beam width, we find more predictions with the structure: "⟨weekday⟩'s sports scoreboard" (Table 6.4).[2] As expected, these predictions have a large early discrepancy, followed by highly (conditionally) probable tokens. For B=100, the average first token discrepancy gap for these summaries is ≈3.6 compared to ≈0.4 in the full test set. As none of the test references includes "sports scoreboard", these summaries have low evaluation.

Table 6.4: Number of "⟨weekday⟩'s sports scoreboard" predictions.

$B = 3$	$B = 5$	$B = 25$	$B = 100$	$B = 250$
0	1	17	19	19

As a potential explanation for this phenomenon, we find that all texts that were summarized as "⟨weekday⟩'s sports scoreboard" included the corresponding weekday. In the training set, we found that 2962 of the 2971 texts that were summarized to "⟨weekday⟩'s sports scoreboard" included the corresponding weekday. The existence of the weekday in the text can lead to the ⟨weekday⟩ token being

[2]Without length normalization, the numbers are higher as this sequence is shorter than most summaries.

suggested as a first token with a low, but sufficiently high, probability to get into the top B tokens. Followed by high probability tokens, it can, in some cases, have an overall probability that is higher than the alternatives.

6.4.7 Search Discrepancies and the Performance Degradation in Beam Search

Our baseline results support the observations that performance degradation is a significant problem that occurs across different neural sequence tasks, using different models and evaluation metrics. Consistent with previous results we find this problem even in length-normalized models.[3]

Based on our empirical analysis, we hypothesize that large search discrepancies are the cause of the previously reported performance degradation in beam search. To test this hypothesis, we modify the beam search algorithm to prevent it from considering large discrepancies, with the prediction that it will eliminate the observed performance degradation.

6.5 Discrepancy-Constrained Beam Search

We evaluate two heuristic methods of constraining the beam search from considering large search discrepancies.

Discrepancy gap: Given a threshold $\mathcal{M}$, we modify beam search to only consider candidates with a discrepancy gap smaller or equal to $\mathcal{M}$. Formally, we modify Eq. 6.2 to include the constraint

$$\max_{y \in \mathcal{V}} logP_\theta(y|\mathrm{x};\{y_0,...,y_{t-1}\}) - logP_\theta(y_t|\mathrm{x};\{y_0...y_{t-1}\}) \leq \mathcal{M}$$

Beam candidate rank: Given a threshold $\mathcal{N}$, we modify $\mathcal{Y}_t$ to only include the top $\mathcal{N}$ one-token extensions in each beam. Note that the beam search still retains (at most) B candidates, however it will not consider more than $\mathcal{N}$ candidates from each beam.

Using the setup in Section 6.4.1, we compare these methods to the baseline. Although the analysis in Section 6.4 was done on the test set (to account for the performance degradation that was previously observed on the test set), $\mathcal{M}$ and $\mathcal{N}$ are tuned on a held-out validation set and no information from the test set was used to tune our methods.

[3] We further show that this problem is not due to length bias in Section 6.6.4.

Table 6.5: A comparison of the baseline results vs. the constrained beam search methods (higher values are better, best baseline results in bold).

Dataset	Method	Threshold	B=1	B=3	B=5	B=25	B=100	B=250
En-De (BLEU-4)	Baseline		25.27	26.00	**26.11**	25.11	23.09	21.38
	Constr. Gap	$\mathcal{M} = 1.5$	25.27	26.00	26.18	26.18	26.22	26.29
	Constr. Rank	$\mathcal{N} = 2$	25.27	26.07	26.01	26.08	26.10	26.10
En-Fr (BLEU-4)	Baseline		40.15	40.77	**40.83**	40.52	38.64	35.03
	Constr. Gap	$\mathcal{M} = 2.0$	40.15	40.78	40.86	40.98	41.05	41.06
	Constr. Rank	$\mathcal{N} = 3$	40.15	40.77	40.81	40.99	41.05	41.02
Gigaword (ROUGE-1)	Baseline		33.56	**34.22**	34.16	34.01	33.67	33.23
	Constr. Gap	$\mathcal{M} = 0.85$	33.56	34.27	34.29	34.43	34.33	34.32
	Constr. Rank	$\mathcal{N} = 2$	33.56	34.48	34.45	34.25	34.23	34.32
MSCOCO (BLEU-4)	Baseline		29.66	**32.36**	31.96	30.04	29.87	29.79
	Constr. Gap	$\mathcal{M} = 0.45$	29.66	32.24	32.33	32.36	32.35	32.35
	Constr. Rank	$\mathcal{N} = 2$	29.66	32.52	31.97	30.88	30.87	30.87

As shown in Table 6.5, both methods significantly reduce, and in some cases completely eliminate, the performance degradation. In machine translation and summarization, we improve performance compared to the baseline with the best test beam width. In general, the discrepancy gap constraint seems to perform better (most notably, for MSCOCO). The gap constraint allows for a finer-grained control over the accepted search discrepancies, however the rank constraint is simpler and easier to tune.

We repeated the analysis in Section 6.4 on discrepancy-constrained beam search and found that both constrained beam search variations substantially reduce the discrepancy phenomena detailed in Section 6.4. Complete results for both constrained methods on WMT'14 En-De are in Appendix B (other tasks exhibited similar results).

Ott et al. [138] proposed to add an inference constraint that prunes copies in the beam search and showed that it significantly reduces the performance degradation in machine translation. However, their empirical analysis still found a drop of approximately a point in the BLEU evaluation for $B = 200$, consistent with our observation that copy predictions do not fully account for the performance degradation in machine translation (Section 6.4.5). Our inference constraints, more general and not limited to copies, completely eliminate the performance degradation (and even slightly improve the evaluation) in machine translation.

6.5.1 Copies and Training Set Predictions in Discrepancy-Constrained Beam Search

In this section, we analyze the impact of the two discrepancy-constrained variants of beam search on the number of copies and training set predictions. Table 6.6 compares the number of copies in the baseline vs. the discrepancy-constrained methods for the machine translation tasks for each beam width. For the baseline, we can see that as we increase the beam width, the number of copies grows significantly. However, both discrepancy-constrained methods significantly reduce this growth.

Table 6.6: Number of copies in machine translations for the baseline and the two types of discrepancy-constrained beam search for different beam widths.

Dataset	Method	Parameter	B=1	B=3	B=5	B=25	B=100	B=250
En-De	Baseline		23	40	49	179	385	567
En-De	Constr. Gap	$\mathcal{M} = 1.5$	23	39	42	50	53	55
En-De	Constr. Rank	$\mathcal{N} = 2$	23	38	44	46	54	55
En-Fr	Baseline		25	28	41	89	227	358
En-Fr	Constr. Gap	$\mathcal{M} = 2.0$	25	27	37	43	46	45
En-Fr	Constr. Rank	$\mathcal{N} = 3$	25	28	38	42	42	46

Table 6.7 compares the number of training set predictions in the baseline vs. the discrepancy-constrained methods for the summarization and image captioning tasks for each beam width. For the baseline, we can see that as we increase the beam width, the number of training set predictions grows significantly. However, as with copies, both discrepancy-constrained methods significantly reduce the growth in training set predictions.

Table 6.7: Number of predictions that are in the training set for the baseline and the two types of discrepancy-constrained beam search for different beam widths.

Dataset	Method	Parameter	B=1	B=3	B=5	B=25	B=100	B=250
Gigaword	Baseline		81	86	86	115	163	224
Gigaword	Constr. Gap	$\mathcal{M} = 0.85$	81	81	77	79	78	78
Gigaword	Constr. Rank	$\mathcal{N} = 2$	81	81	79	79	79	79
MSCOCO	Baseline		163	260	371	588	582	576
MSCOCO	Constr. Gap	$\mathcal{M} = 0.45$	163	265	271	271	271	271
MSCOCO	Constr. Rank	$\mathcal{N} = 2$	163	242	262	231	231	231

The above results show that our methods reduce the growth in the number of both copies and training set predictions, supporting the claim that our hypothesis is a generalization of the previous explanations.

6.5.2 Generation of Non-English Text

To show that our results extend beyond generating text in English or in European languages that share some similarity to English, we present results for the WMT'17 En-Zh dataset (that involves generating translations in Chinese), using the Nematus toolkit [156]. Table 6.8 shows the results for the baseline vs. the constrained methods for different beam widths. Consistent with WMT'17 instructions for evaluating Chinese output, we report BLEU-4 scores computed on characters.[4] The results show a clear performance degradation for the baseline, with BLEU-4 score dropping by more than 3 points. Note that the performance degradation on this dataset is not due to copies, as there are none in the translations. Similar to the other tasks, our constrained methods successfully eliminate the performance degradation and even lead to a higher evaluation.

[4] http://www.statmt.org/wmt17/translation-task.html

Table 6.8: A comparison of the baseline results vs. the constrained beam search methods for the WMT'17 En-Zh dataset based on the BLEU-4 metric (higher values are better; best baseline result in bold).

Dataset	Method	Parameter	B=1	B=3	B=5	B=25	B=100	B=250
En-Zh	Baseline		32.41	**33.17**	33.16	33.01	31.33	29.61
En-Zh	Constr. Gap	$\mathcal{M} = 1.0$	32.41	33.15	33.22	33.43	33.45	33.50
En-Zh	Constr. Rank	$\mathcal{N} = 2$	32.41	33.20	33.17	33.35	33.28	33.30

Figure 6.7 shows the distribution of discrepancy positions for different beam widths and Figure 6.8 shows the mean discrepancy gap per position for different beam widths. We can see that the results for WMT'17 En-Zh exhibit the same phenomena observed for the other datasets in Section 6.4.

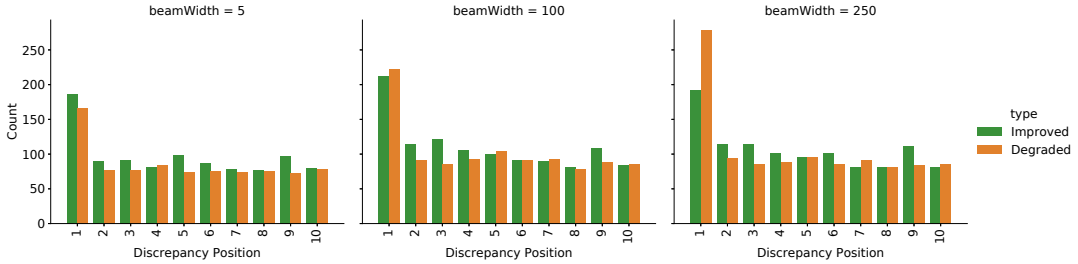


Figure 6.7: WMT'17 En-Zh: Distribution of discrepancy positions for different beam widths.

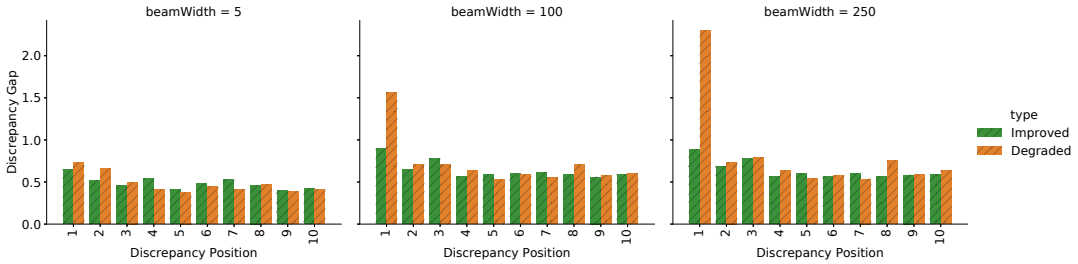


Figure 6.8: WMT'17 En-Zh: Mean discrepancy gap per position for different beam widths.

6.6 Discussion

6.6.1 Connection to Exposure and Label Bias

Our results show that larger beam width leads to increasingly large early discrepancies. These very unlikely early tokens are later compensated by subsequent tokens with a much higher (conditional) probability compared to the subsequent tokens of the more probable early tokens. The large difference in the conditional probability of the subsequent tokens is at the heart of the observed performance degradation.

Previous work has highlighted two potential biases that can account for this difference.

- *Exposure bias* [147] occurs since the model is only exposed to the training data distribution instead of its own predictions: Due to the use of teacher forcing training (see Section 5.1.3), the model is

trained to predict the next token conditioned on the ground truth prefix. However, during inference the model is conditioned on *predicted* prefix that could also be erroneous. Models trained using teacher forcing can be overly reliant on previously predicted tokens which, during inference, can exacerbate error propagation [186]. In the illustrative example in Section 6.4.6, over-relying on an erroneous prefix leads to a conditional distribution that is biased to a specific repetitive pattern in the training data.

- *Label bias* [110] occurs in locally-normalized sequence models: The token probabilities at each time step are locally normalized and therefore the successors of incorrect histories receive the same probability mass as the successors of a correct history [200]. In the illustrative example in Section 6.4.6, the conditional distribution over the successors of a low probability partial sequence is more concentrated compared to more probable partial sequences. Since the conditional distributions at each time step are locally normalized to sum to one, in some cases, a concentrated distribution can compensates a low probability prefix and lead to an overall higher probability sequence.

These biases help explain the observed behavior with large beam width: a biased (conditional) probability that concentrates high probability mass on one token and is locally normalized to sum to one compensates for earlier low probability tokens. The negative effects of these biases have been discussed before [147, 200], however the connection to the performance degradation in beam search and the explanatory framework to allow such analysis is, to our best knowledge, novel.

6.6.2 Connection to Previous Results on Heuristic Search

While beam search is used to perform (approximate) inference, it is a heuristic search algorithm [12]. We therefore believe it is natural to address the performance degradation from a heuristic search perspective. Our analysis, based on the search discrepancy concept from heuristic and combinatorial search, views the probabilities predicted by the neural network as a heuristic value to guide search for a solution. Early mistakes have been shown to have a large negative effect on the performance of combinatorial search [52] and substantial work has analyzed and proposed techniques to mitigate the phenomenon [62, 28], including limited discrepancy search [73]. In combinatorial search, early mistakes were associated with deep inconsistent subtrees [29, 184, 52] and can account for a large variability in performance between different heuristics [201]. In the next chapter, we consider goal-oriented neural sequence decoding in which, similar to combinatorial search, we need to find a solution that satisfies a goal condition. Inspired by work on combinatorial search and our analysis of GBFS in Chapter 4, we investigate whether high variability in the performance of goal-oriented beam search can be observed and, subsequently, exploited to boost the search performance.

6.6.3 Alternative Models and Training Schemes for Sequence Models

In this work, we studied the previously reported beam search performance degradation in the most commonly used neural sequence models that are based on an RNN decoder and are trained to maximize the word-level likelihood, conditioned on the input sequence and the reference history [173, 4]. Recently, there have been several proposals for alternative models and training schemes that rely on sequence-level losses and that can potentially mitigate the effects of the biases described above [147, 200, 39]. These works only report the results for small beam widths, and it is not clear if they reduce, or even eliminate,

the performance degradation in beam search. Analyzing the performance of these models for larger beam widths, and the associated distribution of search discrepancies in these models is a direction for future work. We are also interested in analyzing the recent Transformer model [178] as it represents a different type of sequence-to-sequence model.

6.6.4 Analysis of Length Bias

A recent line of work in machine translation suggested that performance degradation is due to length bias [206, 133]. For larger beams, an end-of-sentence token with a lower probability that leads to an overall more probable hypothesis is more likely to be considered by the beam search. However, we showed performance degradation above even when using length normalization and in tasks where length bias does not appear.

Table 6.9 shows the mean length of generated sentences for different beam widths for the baseline, normalized to the best tested beam width. All values are very close 1.0, which suggest that the observed performance degradation is not due to length bias. We note that for machine translation and summarization this pattern is due to the use of length normalization on the hypotheses log-likelihood, as suggested by Koehn and Knowles [105] (without normalization, the performance degradation would have been worse).[5] In image captioning, however, there is no observed length bias even when length normalization is not used.[6]

Table 6.9: Analysis of the mean length, normalized to best test width (in bold).

Task	Dataset	B=1	B=3	B=5	B=25	B=100	B=250
Translation	En-De	0.99	1.0	**1.0**	1.0	0.99	0.98
	En-Fr	0.99	1.0	**1.0**	1.0	0.99	0.91
Summarization	Gigaword	1.03	**1.0**	0.99	0.99	1.0	1.01
Captioning	MSCOCO	1.04	**1.0**	0.99	0.98	0.98	0.98

In Section 6.4.1, we showed substantial performance degradation as we increase the beam width. As the results in Table 6.9 demonstrate that there is no significant change in the length of generated sequences, the observed performance degradation cannot be attributed to length bias.

6.7 Conclusion

In this work, we perform an empirical analysis of performance degradation in beam search across three neural sequence decoding tasks. We find that the performance degradation for large beam widths is associated with increasing number of early and large search discrepancies. We hypothesize that the fact that beam search exhibits large discrepancies is the cause of the performance degradation and that avoiding such discrepancies will eliminate the performance degradation. We show that this hypothesis generalizes previous results including the existence of copy predictions in machine translation and the training set predictions in image captioning, and accounts for additional degraded sequences. To validate

[5] This is consistent with Ott et al.'s [138] results on performance degradation even when using length normalization.
[6] In fact, as we stated earlier, we found that length normalization reduces the overall performance.

this hypothesis, we show that methods that prevent the search from considering large search discrepancies eliminate the performance degradation in beam search.

Consistent with our thesis statement, the work in this chapter demonstrates how an empirical model of beam search behavior, that is based on the notion of search discrepancies, can explain the observed performance degradation and inform the design of algorithmic solutions that mitigate it. In the next chapter, we continue our investigation of empirical models for beam search. We consider a complete variant of beam search and investigate whether it exhibits a high variability in search performance and whether, similar to our analysis of GBFS, randomized restarts can be used to reduce this variability and improve the performance.

Chapter 7

Randomized Restarting Beam Search in Goal-Oriented Neural Sequence Decoding

Continuing our investigation of empirical models for neural sequence decoding using beam search, in this chapter we focus on beam search for *goal-oriented* neural sequence decoding. Informed by our analysis of the fat- and heavy-tailed behavior of GBFS in planning problems, we investigate whether a similar behavior can be observed for goal-oriented neural sequence decoding and whether our use of randomized restarts can be adapted to boost the performance of beam search in this setting.

7.1 Introduction

Neural sequence models are commonly used in the modeling of sequential data. As demonstrated in Chapter 6, neural sequence models are the state-of-the-art approach for tasks such as machine translation [49], text summarization [21], and image captioning [183]. Beam search (see Section 5.2.3) is the most commonly used algorithm for decoding neural sequence models by (approximately) finding the most likely output sequence conditioned on the input.

Recently, neural sequence models have been successfully applied to different combinatorial search problems such as program synthesis and routing problems. Unlike machine translation and image captioning, such problems often have a goal criteria that can be used to evaluate candidate solution and we wish to generate solutions that satisfy the goal criteria. For example, in resource-constrained combinatorial routing problems, we may wish to find a tour that satisfies some resource constraint (e.g., limited fuel or budget). In these scenarios, beam search is used to produce a set of promising (high-likelihood) candidate sequences that are evaluated to determine if they satisfy the goal criteria. If none of these satisfy the criteria, the beam search can be restarted with a larger beam size until a satisfying solution is found.

In Chapter 4, we focused on GBFS, another goal-oriented heuristic search algorithm, and showed that the distribution of search efforts exhibits a fat- and heavy-tailed behavior that can be exploited to boost its performance by incorporating randomization in the search. In this chapter, we investigate

whether a heavy-tailed behavior can also observed for goal-oriented beam search. We focus on complete anytime beam search (CAB; see Section 5.3.1), a complete variant of beam search commonly used in goal-oriented neural sequence decoding, and perform an extensive empirical study of the heavy-tailed behavior. Specifically, we make the following contributions:

1. We consider the setting of goal-oriented neural sequence decoding, where the decoded sequence must satisfy a goal condition. We consider four neural decoding tasks for which a selected evaluation metric is available at decoding time and develop a goal-oriented variant of these problems where the goal condition enforces bounded suboptimality with respect to the evaluation metric.

2. We show that for goal-oriented neural sequence problems, complete anytime beam search exhibits a fat- or heavy-tailed behavior on ensembles of relaxed problems, similar to the behavior exhibited for GBFS.

3. We consider a randomized variant of beam search that is based on noise injection to the inputs of the neural network and show that randomized complete anytime beam search exhibits fat- or heavy-tailed behavior on ensembles of multiple runs on a single instance.

4. Inspired by our results for GBFS, we introduce a randomized restarting variant of complete anytime beam search and show that it solves some of the hardest instances faster and outperforms the baseline.

5. We conduct extensive empirical evaluation and analyze the impact of different parameters including the constrainedness of the goal-criteria, the restart policy, and the type of randomization on the effectiveness of our method.

7.1.1 Organization

In Section 7.2 we define the problem of goal-oriented neural sequence decoding using beam search. Section 7.3 describes the benchmark problems used in our analysis and how they can be posed as goal-oriented neural sequence decoding tasks. In Section 7.4, we show that complete anytime beam search exhibits a fat- and heavy-tailed behavior on goal-oriented neural sequence decoding tasks. Then, in Section 7.5, we consider a randomized variant of complete anytime beam search that is based on noise injection to the input of the neural network and show that it exhibits a fat- and heavy-tailed behavior on a single instance. Finally, in Section 7.6 we introduce RR-CAB a randomized restarting variant of complete anytime beam search and in Section 7.7 we perform an extensive experimental evaluation of RR-CAB and show it outperforms the baseline. In Section 7.8 we discuss the impact of different design choices, potential limitations, and directions for future work and in Section 7.9 we summarize the chapter.

7.2 Beam Search for Goal-Oriented Decoding of Neural Sequence

As discussed in Section 5.2, beam search is a limited-width breadth-first search that can be used to decode neural sequence models token-by-token while keeping a fixed number of active candidates at each step. In tasks such as machine translation and image captioning, we use beam search to find an approximation

of the most likely sequence y conditioned on the input x according to some learned model θ,

$$\arg\max_{y} p_\theta(y|x). \tag{7.1}$$

In goal-oriented decoding of neural sequence models, we are not looking for the most-likely sequence according to the learned model. Instead, we are looking for a solution that satisfies the goal criteria. In such scenarios, we use beam search to generate a set of $\mathcal{B}$ high-quality candidates that are then evaluated to determine if they satisfy the goal criteria. Once a candidate satisfies the goal criteria, it is returned as the solution of the beam search.

This setting allows us to perform a complete search, that is guaranteed to find a satisfying solution, if one exists. Previous work on goal-oriented neural sequence decoding considered a variant of the complete anytime beam search (CAB) [208] in which failing to find a satisfying solution results in doubling the beam width and re-running the beam search [210, 5, 111]. As the beam width increases, a larger portion of the hypothesis space is explored and the search is guaranteed to find a solution, if one exists, provided the hypothesis space is finite. Algorithm 7.1 shows a pseudo-code for this variant of complete anytime beam search. For more detailed discussion on complete variants of beam search, including CAB, see Section 5.2.

Algorithm 7.1 Complete Anytime Beam Search

```
function CAB(goalCriteria)
    beamWidth ← 1
    while not solved do
        candidates ← BeamSearch(beamWidth)
        for cand ∈ candidates do
            if Satisfy(cand, goalCriteria) then
                return cand
        beamWidth ← 2 · beamWidth
```

7.3 Goal-Oriented Benchmark Problems

We introduce a set of four goal-oriented benchmark problems that will be used in our analysis. Following is a description the problems and the goal criteria for each problem.

7.3.1 Combinatorial Routing Problems

Several recent works have demonstrated the potential of using deep learning to solve combinatorial optimzation problems [106, 103, 35, 136]. A recent work [106] proposed an architecture based on attention layers and trained using REINFORCE [191] to generate solutions for combinatorial routing problems that minimize the solution cost. The authors use this architecture to generate solutions to the Travelling Salesman Problem (TSP), two variants of the Vehicle Routing Problem (VRP), the Orienteering Problem (OP), and the Prize Collecting TSP (PCTSP) and show it outperforms a wide range of baselines. Decoding can be done using sampling or beam search, and the best solution among the generated candidates is returned. To eliminate infeasible solutions, e.g., re-visting the same node in TSP, the authors use

masking (setting the log-probabilities of infeasible solutions to $-\infty$). In our work, we use Kool et al.'s [106] architecture and problem instances and run experiments on two combinatorial routing problems:

- The Travelling Salesman Problem (TSP) consists of constructing a tour that starts at the depot, visits all nodes exactly once, and returns to the depot.

- The Capacitated Vehicle Routing Problem (CVRP) consists of constructing multiple routes, each starting and ending at the depot, such that the total demand of the nodes in each route does not exceed the vehicle capacity.

The cost of solution in both problems is the sum of pairwise euclidean distances of consecutive nodes in the solution path (including the depot).

Goal Criteria.

As the current model is trained to minimize the solution cost, we consider the goal-oriented problem of finding a solution with a bounded optimality gap. Assuming a minimization problem with cost function $\mathcal{C}$, our goal criteria for a candidate solution x is $\frac{\mathcal{C}(x)-\mathcal{C}(x^*)}{\mathcal{C}(x^*)} \leq \varepsilon$, where x^* is an optimal solution and ε controls the constrainedness of problems (increasing ε leads to a higher expected number of feasible solutions). We note that this notion of constrainedness matches the notion of resource-constrainedness used in Chapter 4 that was previously used to evaluate planning algorithms in resource-constrained environments [134].

7.3.2 Visual Program Synthesis

Several recent works have considered the problem of synthesizing programs for images using deep neural networks [161, 174, 123]. These networks take an image as input and output a program that generates the image. The quality of a candidate solution program can be evaluated using a metric of projection loss, typically a distance measure between the generated image and the original one. In our experiments, we use CSGNet [161], a neural architecture that takes in a 2D or 3D shape image and outputs a program to generate the shape using instructions based on constructive solid geometry (CSG). CSGNet is trained using a combination of supervised learning and reinforcement learning (using REINFORCE [191]) to minimize the visual distance between the generated solutions and the input images.

Goal Criteria.

Our goal criteria is based on Chamfer Distance ($\mathcal{CD}$), a measure of visual similarity between two shapes that is used in the original paper [161],

$$\mathcal{CD}(X,Y) = \frac{1}{2|X|}\sum_{x\in X}\min_{y\in Y}\|x-y\|_2 + \frac{1}{2|Y|}\sum_{y\in Y}\min_{x\in X}\|x-y\|_2,$$

where X and Y are the sets of points on the edges of the two shapes. We define our goal criteria for a candidate solution x to be $\mathcal{CD}(x,i) \leq \gamma$ where i is the input image and the parameter γ controls the constrainedness of problems.

7.3.3 Conditional Molecular Design

A recent line of work focuses on generating molecules with specific properties [98, 97, 96], such as the molecular weight (MolWt), the Wildman-Crippen partition coefficient (LogP) [190], and a quantitative estimation of drug-likeness (QED) [11]. Kang and Cho [98] proposed a semi-supervised variational autoencoder that is trained on a set of existing molecules from the ZINC dataset [169] with only a partial annotation (i.e., only a fraction of the molecules are labelled with the property values). The model represent a generative process in which the input molecule x is generated from the distribution $p(x|z, y)$ that is conditioned on the molecule properties y and a latent variable z. The proposed architecture includes three RNNs with Gated Recurrent Units (GRU):

1. The property prediction network represents the conditional probability $p(y|x)$ and is used to predict the properties of unlabelled molecules.

2. The encoder network represents the conditional probability $p(z|x, y)$.

3. The decoder network represents the conditional probability $p(x|y, z)$.

The molecules are represented using SMILES [188] strings and are generated character-by-character. For the conditional generation of molecule with specific property, we sample z from its prior and y from its prior conditioned on the specific property. A molecule representation $\hat{x}$ is obtained from y and z using the decoder's conditional probability $p(x|y, z)$,

$$\hat{x} = \arg\max p(x|y, z), \tag{7.2}$$

where Eq. (7.2) is approximated by a beam search.

Goal Criteria.

Our goal criteria is based on the QED property [11],

$$QED = exp(\frac{1}{n}\sum_{i=1}^{n} \ln d_i),$$

where $d_i, i = 1..n$ are n desirability functions [71], each corresponding to a chosen molecular descriptor. We use RDKit [112] to compute the QED of the generated molecule and define our goal criteria based on the absolute difference between its QED and the desired QED. Formally, we define our goal criteria for a candidate solution x to be $|QED(x) - q| \leq \rho$ where q is the desired value of QED and the parameter ρ represents a bound on the deviation from the desired QED value and controls the constrainedness of the criteria.

7.4 Fat- and Heavy-tailed Behavior in Goal-Oriented Neural-Guided Search

In this section we demonstrate the existence of heavy-tailed behavior and the associated exceptionally hard problems in goal-oriented neural sequence decoding.

7.4.1 The Travelling Salesman Problem (TSP)

We consider a collection of 500 randomly generated TSP problem instances with 100 nodes solved using beam search with a beam width of 10. Figure 7.1 shows the distribution of solution quality presented as optimality gap ($\frac{\mathcal{C}(x)-\mathcal{C}(x^*)}{\mathcal{C}(x^*)}$) to match our goal criteria. The center of the distribution is around 0.03 with the mean (marked in a dashed line) at approximately 0.034. However, there is a small number of problems for which the optimality gap can be much higher (up to approximately 0.1).

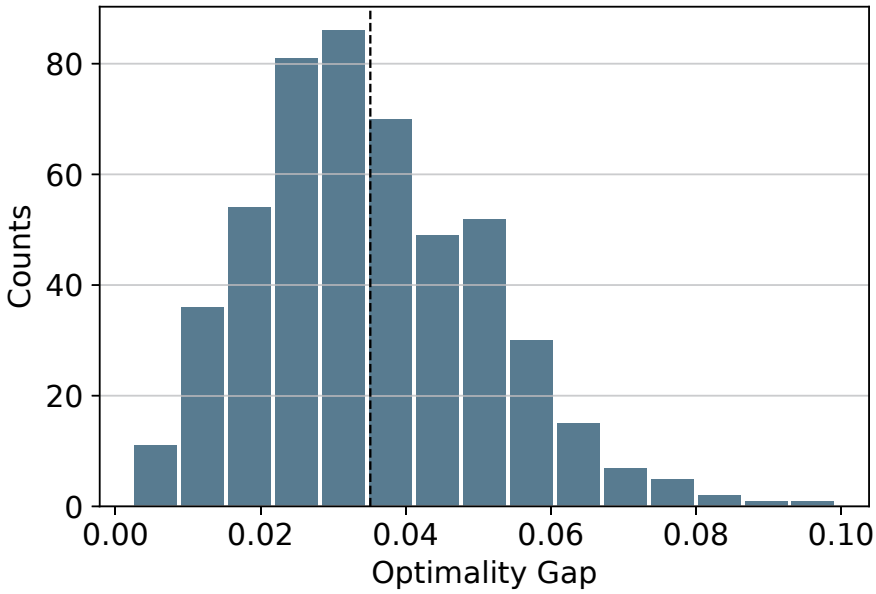


Figure 7.1: TSP (100 nodes): Histogram of of solution quality for 500 random instances.

We consider the case of solving the goal-oriented problem where solutions must satisfy a bound on the optimality gap denoted as ε (as discussed in Section 7.3.1). We then use complete anytime beam search (Algorithm 7.1) to solve the problems with the given bound as goal criteria. We start with a beam width of 1, and double the beam width in each iteration if no solution that satisfies the goal criteria is found. We record the beam width for which a satisfying solution was found representing the required search effort.

Figure 7.2 shows the the search effort distribution for three different goal criteria $\varepsilon = 0.04$, $\varepsilon = 0.05$, $\varepsilon = 0.06$. The y-axis represents the number of unsolved problems in log-scale, while the x axis represents the search effort (i.e., beam width) in discrete log_2-scale (i.e., in steps of $2^i, i = 0, 1, ...$) to match the behavior of the complete anytime beam search. We artificially add the step 0 (i.e., no search effort) to denote the total number of problems. For $\varepsilon = 0.05$ and $\varepsilon = 0.06$, there is a clear heavy-tailed behavior with a very low median (beam width of 1) and a slow decay of the tail over multiple orders of magnitude. In fact, not all problems were solved for the maximum beam width of $32,768$. Note that when $\varepsilon = 0.05$, 332 of the 500 problems are solved with a beam width of 1, while five *exceptionally hard problems* could not be solved for a beam width of $32,768$. For a more constrained goal criteria of $\varepsilon = 0.04$, we still observe a fat-tailed behavior, however we see a noticeable increase in the difficulty of problems and the number of problems that could not be solve in the search effort limits is significantly higher. We could not analyze more constrained goal criteria due to the high computational cost, however we hypothesize

that problem will become significantly harder and the heavy-tailed behavior will reduce, consistent with the results for GBFS in Chapter 4. Since we cannot solve the hardest instances in the search effort limit, i.e., we do not fully observe the tail of the distribution, we could not compute the quantitative metrics we use in Chapter 4 such as the L-Kurtosis and the stability index α of the fitted GPD model.

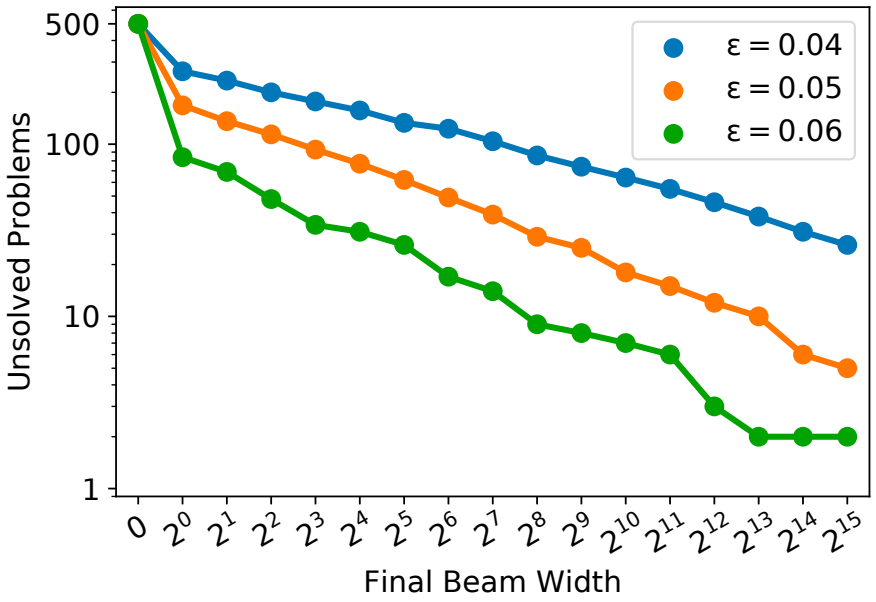
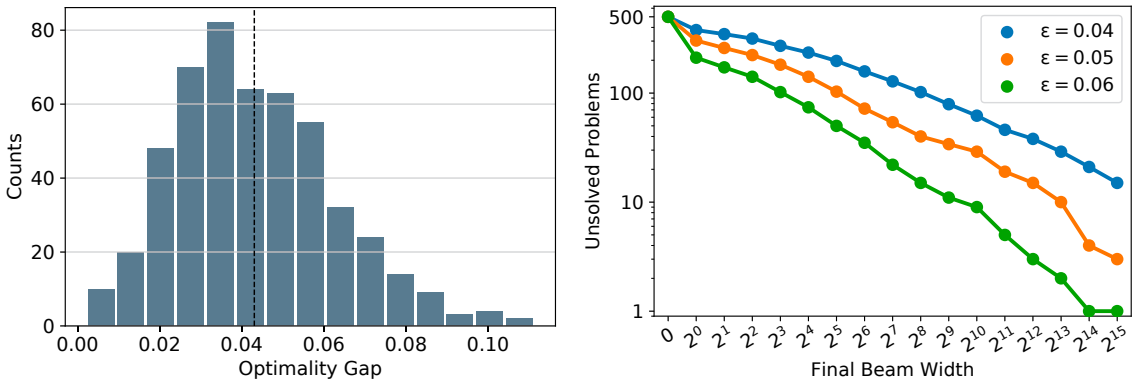


Figure 7.2: TSP (100 nodes): Distribution of beam widths for 500 random instances.

7.4.2 The Capacitated Vehicle Routing Problem (CVRP)

We consider a collection of 500 randomly generated CVRP problems with 50 nodes. Figure 7.3a shows the histogram of optimality gap for beam search with a beam width of 10. The center of the distribution is around 0.04 with the mean (marked in a dashed line) is at approximately 0.043, however there is a small number of problems for which the optimality gap can be much higher.

Similar to TSP, we run experiments for $\varepsilon = 0.04$, $\varepsilon = 0.05$, $\varepsilon = 0.06$ and present the distribution of search efforts in Figure 7.3b. We observe a clear heavy tailed behavior for the more relaxed problems ($\varepsilon = 0.06$), with most problems solved for a beam width of 1 and one problem that could not be solved in the maximum beam width of $32,768$. As problems become more constrained we observe an increase in the median problem difficulty and the number of problems that could not be solved in the maximum beam width increases.



(a) Histogram of solution quality.

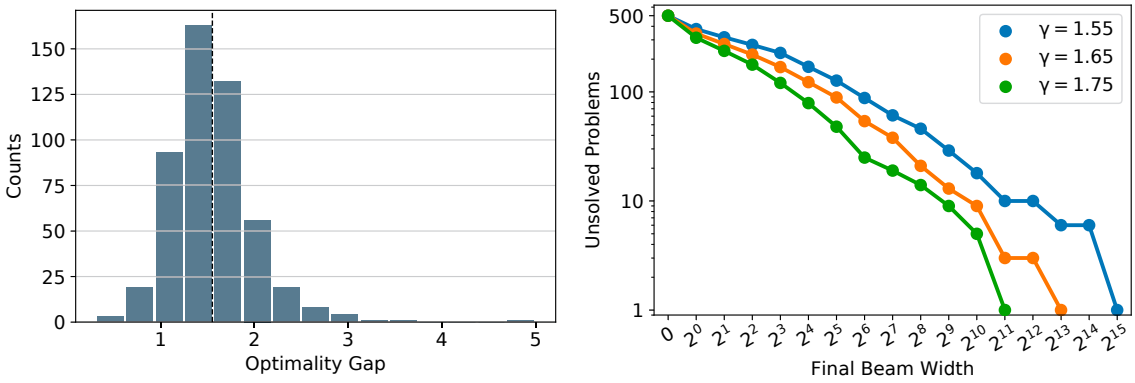
(b) Distribution of beam widths.

Figure 7.3: CVRP (50 nodes): Results for 500 random instances.

7.4.3 Visual Program Synthesis

We consider a set of 500 visual program synthesis problems from the CAD dataset of CSGNet [161] solved using beam search with a beam width of 10. Figure 7.4a shows the distribution of solution quality presented as the Chamfer distance between the candidate image x and the input image i, $\mathcal{CD}(x, i)$. The center of the distribution is around 1.5 with the mean (marked in a dashed line) at approximately 1.55, however there is a small number of problems for which the Chamfer distance can be much higher.

Next, we consider the goal-oriented setting in which solutions must satisfy a bound on the Chamfer distance from the input image, $\mathcal{CD}(x, i) \leq \gamma$. We run experiments for $\gamma = 1.55$, $\gamma = 1.65$, $\gamma = 1.75$ and present the distribution of search efforts in Figure 7.4b. Again, we observe a very slow decay of the tail, with most problems solved for a relatively small search effort (beam width between 2 and 8, depending on γ), while some problems require several orders of magnitude larger beam width. For $\gamma = 1.55$, we even have an unsolved problem at the maximum beam width of $32,768$.



(a) Histogram of solution quality.

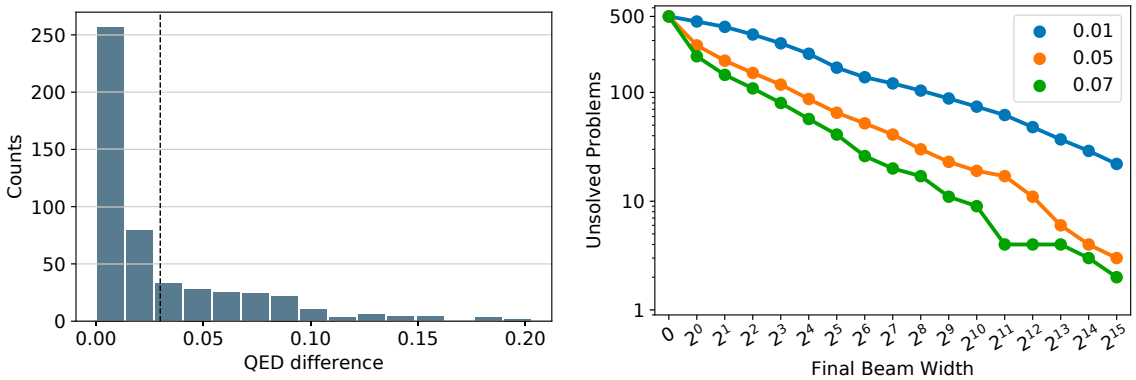
(b) Distribution of beam widths.

Figure 7.4: Visual Program Synthesis: Results for 500 random instances.

7.4.4 Conditional Molecule Generation

We consider a set of 500 random problem instances of conditional molecule generation conditioned on QED of 0.9 [98] solved using beam search with a beam width of 10. Recall that our goal condition is that the difference between the QED of the generated molecule and the desired level is at most ρ. Figure 7.5a shows the distribution of solution quality presented as the difference in QED from the desired level of 0.9, $|QED(x) - 0.9|$. The mean (marked in a dashed line) is at approximately 0.03, however there is a small number of problems for which the difference can be much higher.

Next, we consider the goal-oriented setting in which solutions must satisfy a bound on the solution quality, $|QED(x) - q| \leq \rho$. We run experiments for $\rho = 0.01$, $\rho = 0.05$, $\rho = 0.07$ and present the distribution of search efforts in Figure 7.5b. For $\rho = 0.07$, we observe a clear heavy-tailed behavior, with the majority of problems were solved for a beam width of 1 while two problems remain unsolved even at the maximum beam width of $32,768$. For the more constained problems $\rho = 0.01$, we see that an increase in problem difficulty even in the lower quantiles of search efforts with fewer problems solved for small beam widths.



(a) Histogram of solution quality.

(b) Distribution of beam widths.

Figure 7.5: Conditional Molecule Generation: Results for 500 random instances.

The above results suggest that goal-oriented beam search exhibits a heavy-tailed behavior in ensembles of random problems, similar to the one we observe in Chapter 4 for GBFS. In GBFS, we found that the large variability in the search effort in ensembles of random problems was often associated with the algorithm, rather than problem instances. To isolate the variability of the search algorithm, we studied the search effort distribution of a randomized variant of GBFS on a single instance. In the next section, we propose a method to randomize a goal-oriented beam search and perform an empirical analysis of the distribution of search effort, similar to our analysis in Section 4.4.

7.5 Fat- and Heavy-tailed Behavior on a Single Instance

In this section, we study the distribution of search effort of a randomized goal-oriented beam search on a single problem instance. In order to introduce randomization into beam search decoding of neural sequence models, we inject random noise in the input of the neural network that is being decoded using beam search. Injecting random noise in the inputs of a neural network is a known technique in the

training of neural networks in order to improve their robustness [67].[1] Note that the noise injected to the network's inputs does not impact the goal test that is still based on the original input, i.e., the noise does not change the problem we are solving. The sole purpose of the noise is to introduce some randomness in the network's predicted probabilities and, as a result, in the beam search decoding process.

7.5.1 The Travelling Salesman Problem (TSP)

For TSP instances, the inputs to the network consist of the locations of all nodes, expressed as two-dimensional coordinates normalized in the range [0, 1]. We inject noise to the network inputs by *adding* small random noise drawn from a uniform distribution, $U(-0.1, 0.1)$. We note, again, that the change in the neural network outputs does not impact the actual problem being solved, i.e., the computation of the cost of solution and the goal criteria are based on the real problem inputs without the noise injection.

Figure 7.6 shows the distribution of search effort for 500 randomized runs (i.e., runs with different random injected noise) for different values of ε. We can see a fat-tailed behavior that indicates a significant variability that is associated with the search method.

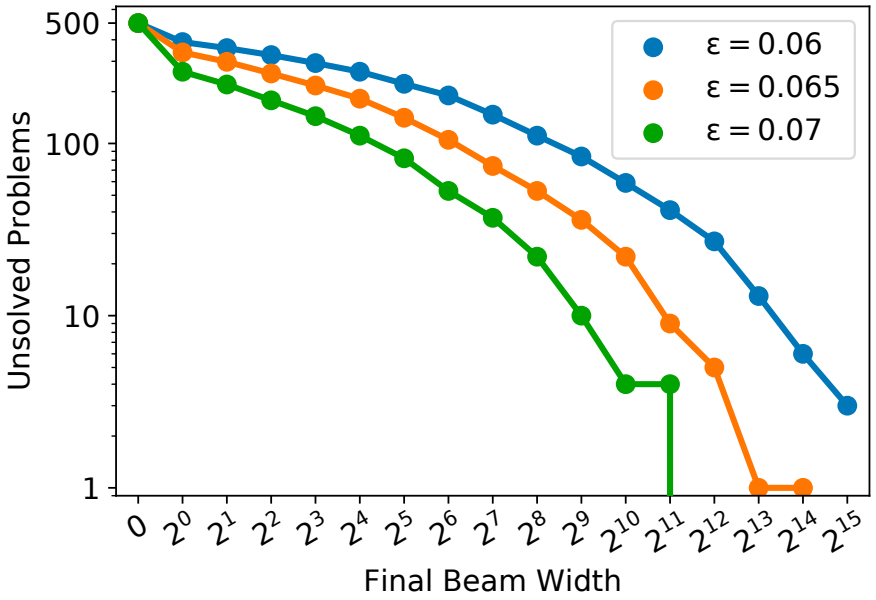


Figure 7.6: TSP (100 nodes): Distribution of beam widths for 500 randomized runs on a single instance.

The results in Figure 7.6 were observed for a specific instance chosen arbitrarily. Experiments with different instances also yielded fat- and heavy-heavy tailed behavior, however we found larger differences among instances compared to GBFS instances in Chapter 4: instances exhibited different levels of fat- and heavy-tailedness for different levels of goal-condition constrainedness. We note that this observation is relevant for the other problems described below.

[1] Note that we are not aware of any direct connection between noise injection in training to increase robustness and our use of noise injection in testing to introduce randomness in the decoding process. However, it might be interesting to consider whether there is some underlying connection.

7.5.2 The Capacitated Vehicle Routing Problem (CVRP)

For CVRP instances, the inputs to the network consist of the location and demand of each of the nodes. We inject noise to the network inputs by *multiplying* the location and demand values by random factor that is close to 1 drawn from uniform distribution $U(0.95, 1.05)$, i.e., we create random variants of the input values that are at most 5% higher or lower than the original values. The differences in the type and amount of injected noise are due to the need to tailor problem-specific noise injection as we discuss later in Section 7.8.1.

Figure 7.7 shows the distribution of search effort for 500 randomized runs on a single instance for different values of ε. For $\epsilon = 0.7$ or $\varepsilon = 0.8$, we observe a very fat-tailed behavior with the hardest problems require several order of magnitude larger beam width to solve. As problems become more constrained we observe an increase the median effort and more and more problems become hard to solve.

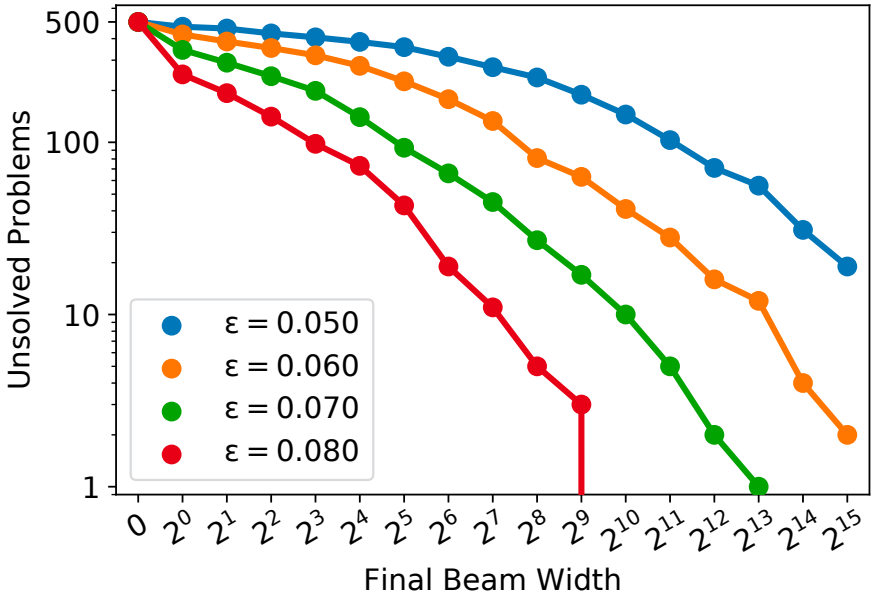


Figure 7.7: CVRP (50 nodes): Distribution of beam widths for 500 randomized runs on a single instance.

7.5.3 Visual Program Synthesis

For visual program synthesis instances, the input to the network is a two-dimensional binary shape image, i.e., a matrix whose values are either zero or one. To add noise to images we flip, with small probability, the bits that are close to the edges of the shape. The detailed procedure of noise injection is described and demonstrated in Appendix C.

Figure 7.8 shows the distribution of search effort for 500 randomized runs on a single instance for different values of γ. For example, for $\gamma = 1.45$, we observe a very fat-tailed behavior with more than half of the problem instances solved for a beam width of 16, while the hardest instance could not be solved with a beam width of 32,768.

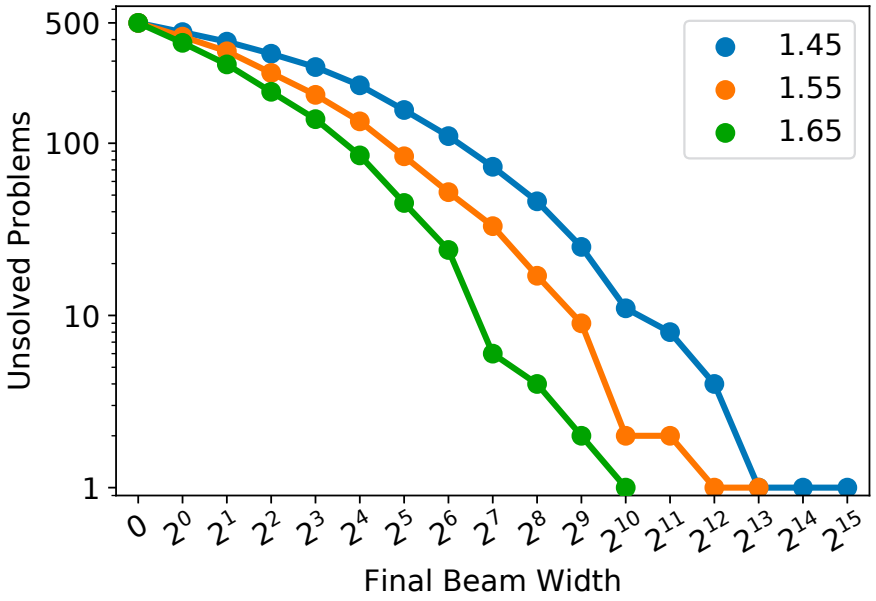


Figure 7.8: Visual Program Synthesis: Distribution of beam widths for 500 randomized runs on a single instance.

7.5.4 Conditional Molecule Generation

In conditional molecule generation, the input to the network consists of y that represents the molecule properties and z that represents a point in the latent space. We inject noise to the network inputs, both z and y, by adding random zero-centered Gaussian noise drawn from $N(\mu, \sigma^2)$ with $\mu = 0$ and $\sigma = 0.3$.

Figure 7.9 shows the distribution of search effort for 500 randomized runs on a single instance for different values of ρ. The results show a fat-tailed behavior. For example, for $\epsilon = 0.4$, the majority of instances are solved for a beam width of 8, while the hardest instance requires a beam width of 16,384 to be solved. As we increase the constrainedness of the goal criteria, problems become harder. For $\rho = 0.02$, the median problem is only solved for a beam width of 128.

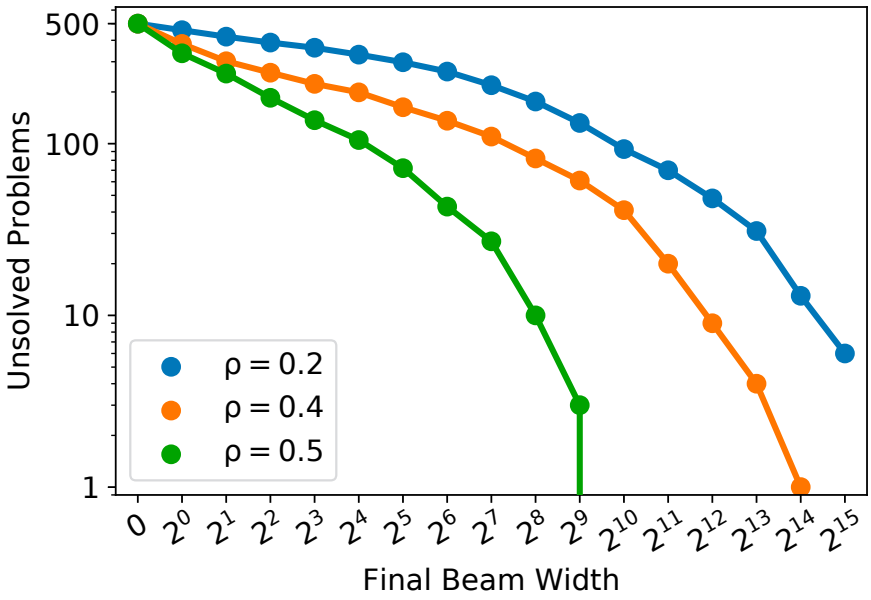


Figure 7.9: Conditional Molecule Generation: Distribution of beam widths for 500 randomized runs on a single instance.

The results in this section indicate that significant variability can be associated with the search algorithm itself. This result suggests that a significant part of the variability observed for the ensembles of random problems in Section 7.4 might be due to the search algorithm. Our work on GBFS in Chapter 4 and previous work on CSPs/SAT [62, 66] has exploited the variability associated with the search algorithm by introducing randomized restarts. In the next section, we propose a complete variant of beam search that incorporates randomized restarts and evaluate its impact on the distribution of search effort. Based on our results on GBFS in Chapter 4, we expect that randomized restarts will exploit the high variability observed in Section 7.5 and reduce the search effort for the hardest problem instances.

7.6 Randomized Restarting Neural-Guided Beam Search for Goal-Oriented Combinatorial Problems

In this section, we present randomized-restarting complete anytime beam search (RR-CAB). A variant of complete anytime beam search (Algorithm 7.1) that uses randomized beam search and a custom restart strategy.

Algorithm 7.2 presents the pseudo-code of RR-CAB, where the goal criteria and the restart strategy are passed as parameters. In each iteration the algorithm runs a randomized beam search (using a random seed) with a beam width that is determined by the restart strategy. The algorithm returns when one of the candidate solutions generated by the beam search satisfies the goal criteria.

Algorithm 7.2 Randomized Complete Beam Search

```
function RR-CAB(goalCriteria, restartStrategy)
    iteration ← 1
    while not solved do
        beamWidth ← restartStrategy(iteration)
        seed ← RandomSeed()
        candidates ← RandomizedBeamSearch(beamWidth, seed)
        for cand ∈ candidates do
            if Satisfy(cand, goalCriteria) then
                return cand
        iteration ← iteration + 1
```

In order to randomize the results of a beam search, we consider the following two options.

Beam search with injected input noise. Following the methodology in Section 7.5, we inject noise to the inputs of the neural networks.

Stochastic beam search (SBS) [107]. SBS is a stochastic variant of beam search that samples k elements without replacement from a sequence model and is therefore a variant of beam search that produces randomized output (see Section 5.3.2 for a detailed description). Note that we could not perform the analysis in Section 7.5 using SBS since, unlike input noise injection, we cannot guarantee that repeated runs with different beam widths will maintain similar conditional probability distributions (see discussion in Section 7.8.1). However, in RR-CAB, we are not interested in maintaining the same probability distributions across runs and therefore SBS can be used as a randomized variant of beam search.

7.6.1 Restart Strategies

A restart strategy is a sequence $(t_1, t_2, t_3, ...)$ of run lengths after which the search restarts. In goal-oriented neural sequence decoding, the sequence length is either fixed (e.g., in TSP and CVRP) or predicted by the network (e.g., in visual program synthesis or conditional molecule generation). If we want to allocate more search effort, we simply extend the beam width thus allowing more sequences to be tested against the goal condition.

In each iteration, we run a beam search with a given beam width until a solution if found. In deterministic complete anytime beam search (Algorithm 7.1), the beam width is increased in each iteration. In RR-CAB, running a search with the same beam width multiple times leads to different results and can sometimes be more efficient than increasing the beam width. We therefore employ a custom restart strategy to determine the beam width in each iteration. We consider two popular restart strategies from the literature.

Fixed-Cutoff Strategy. Fixed-cutoff strategies [66] are simple strategies of the form $(t_c, t_c, ...)$ where t_c is a constant. This strategy is often not robust enough: a small t_c value might not be sufficient to solve all problems, while a larger value will be computationally inefficient.

Geometric Strategy. Geometric strategies [185] take the form $(r^0, r^1, r^2, r^3, ...)$ where the geometric factor r controls how fast the cutoff values grow. When $r = 2$ and randomization is not applied, this strategy has a similar behavior to the complete beam search procedure described in Section 7.2.

7.7 Empirical Results

In this section, we present empirical analysis of the performance of RR-CAB on the goal-oriented benchmarks. We compare results for the two randomization techniques (input noise injection and SBS) and the two restart policies (geometric and fixed-cutoff) described in Section 7.6.

7.7.1 Results for the Travelling Salesman Problem (TSP)

We consider the same collection of 500 randomly generated TSP problems with 100 nodes used in Section 7.4. We analyze the results of RR-CAB with random noise injection and the two restart strategies: geometric with $r = 2$ and fixed-cutoff with beam width $\mathcal{B} = 8$. In order to directly compare the performance of a fixed-cutoff strategy and a geometric strategy, we organize the results of fixed-cutoffs beam search in batches of multiple beam searches with a constant beam width, such that they sum to the beam width of the corresponding beam search with geometric restarts. For example, we present results for a geometric restart policy for the beam width thresholds 1, 2, 4, 8, 16, 32, etc. In comparison, for fixed-cutoff restarts, the result for a threshold of 16 represents a batch of two beam searches, each with a constant beam width of 8.

Figure 7.10 compares the distribution of search effort of standard CAB and RR-CAB in the configurations described above.[2] In general, the randomized variants tend to under perform for the very small beam width: problems that were easily solved without randomization do not benefit, and even suffer, from adding randomization. In particular, since we use a beam width $\mathcal{B} = 8$ for the fixed-cutoff strategy, solutions are only found starting from a threshold of 8. However, as we increased the search effort, we see that the randomized variants outperform standard CAB. For the more constrained problems, we see that the fixed-cutoff strategy significantly outperforms the geometric restarts strategy. This could be due the use of relatively large r (in Chapter 4, we used $r = 1.5$ for GBFS) chosen for fair comparison with CAB. For $\varepsilon = 0.6$, geometric restarts seem to have similar performance to fixed cut-offs.

[2]Detailed numeric results for all the graphs in this section appear in Appendix D.

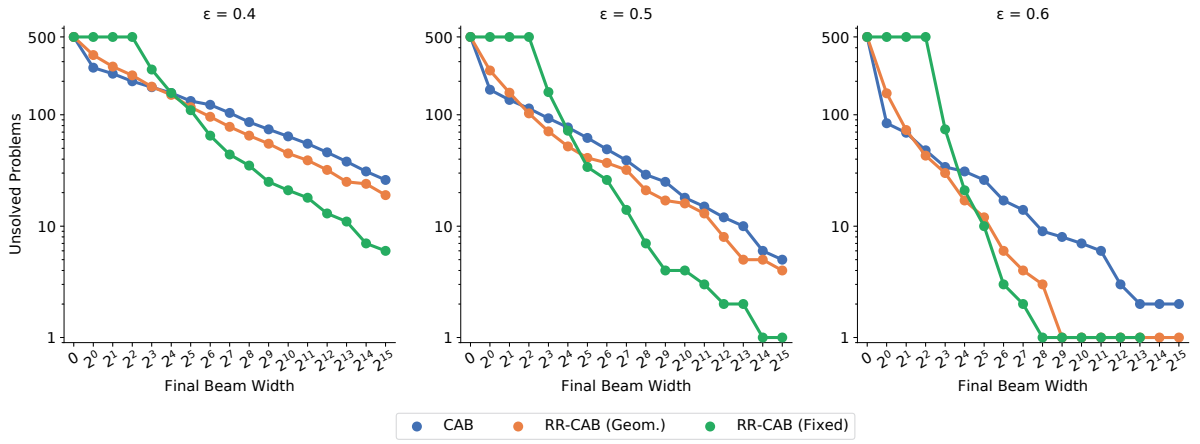


Figure 7.10: TSP (100 nodes): Distribution of beam widths for 500 random instance for RR-CAB with input noise injection.

The inherent differences between the two restart policies result in an apparent inferiority of fixed-cutoffs in smaller beam widths: in addition to having no solutions for beam widths smaller than 8, even for a beam width of 8 it underperforms since RR-CAB with geometric restarts has already made three randomized runs (for beam width 1, 2, and 4) that can lead to solutions. In practice, this is easily mitigated by using a restart policy that starts with geometric restarts before changing to fixed-cutoffs: $1, 2, 4, 8, 8, \ldots$ To maintain simple and clear comparison we do not adopt this enhancement in our evaluation.

Figure 7.11 shows similar comparison to Figure 7.10 where the beam search is randomized using SBS. Again, we see that introducing randomization to CAB leads to better performance. However, the performance gain from SBS is smaller compared to noise injection (we discuss this issue in more detail in Section 7.8.2). Interestingly, for SBS we find that geometric restarts are at least as good as fixed-cutoff strategy.

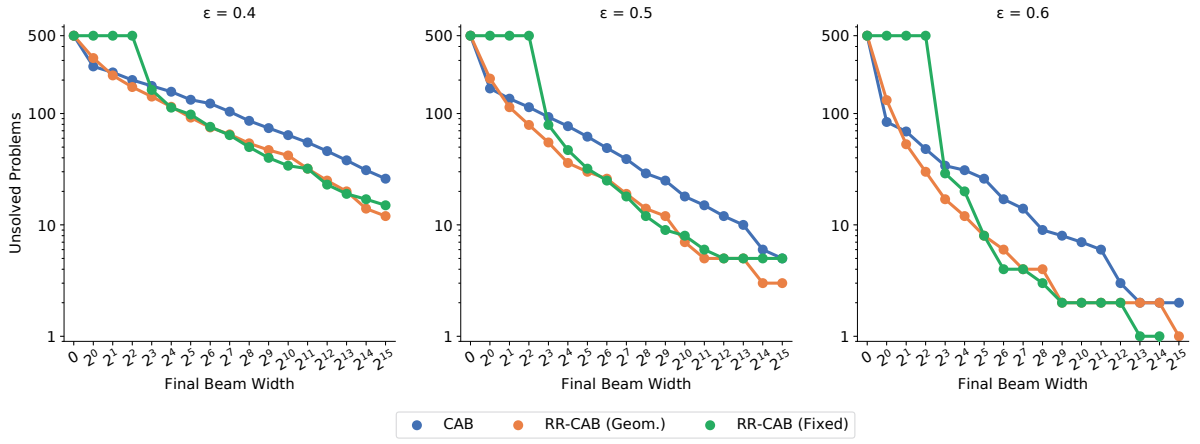


Figure 7.11: TSP (100 nodes): Distribution of beam widths for 500 random instance for RR-CAB with SBS.

The above results show that introducing randomization in the search can help solve some of the hardest instances faster. Consistent with our results on GBFS in Chapter 4, the impact on the more relaxed instances tends to be more significant. However, we note that we cannot analyze the impact of

RR-CAB on more constrained instances due to computational limitations and even for $\varepsilon = 0.4$, using randomization seems to have positive impact on the performance.

7.7.2 Results for the Capacitated Vehicle Routing Problem (CVRP)

We consider the collection of 500 randomly generated CVRP problems with 50 nodes used in Section 7.4. Figure 7.12 compares the performance of RR-CAB with input noise injection to standard CAB. We can see that RR-CAB solves more problems within the search effort limit, and solves some of the hardest problems for a smaller beam width. As with TSP, the fixed-cutoff policy tends to work better for input noise injection.

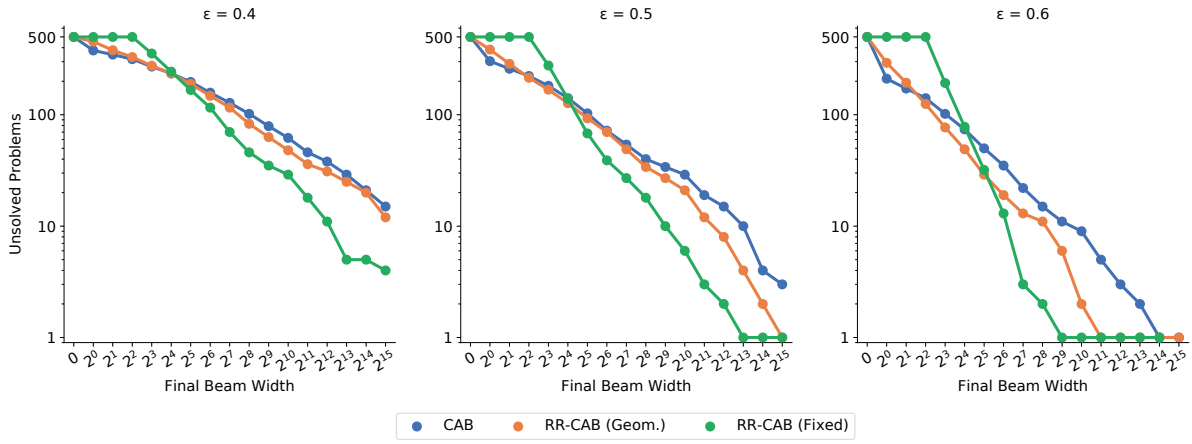


Figure 7.12: CVRP (50 nodes): Distribution of beam widths for 500 random instance for RR-CAB with input noise injection.

Figure 7.13 shows the performance of RR-CAB with beam search randomization based on SBS. RR-CAB with SBS also outperforms the baseline, and as with TSP, geometric restarts tend to work better in this setting. As with TSP, the performance gain when using SBS is smaller compared to noise injection.

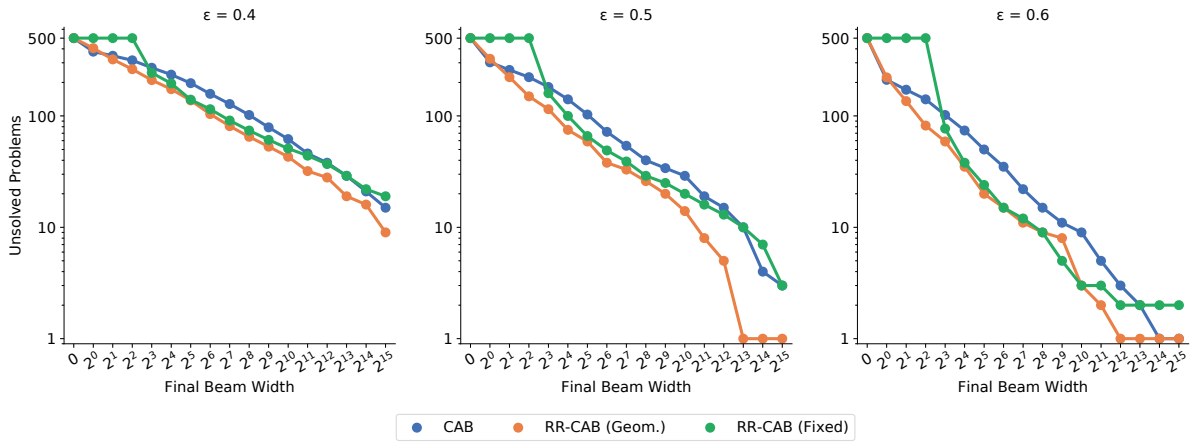


Figure 7.13: CVRP (50 nodes): Distribution of beam widths for 500 random instance for RR-CAB with SBS.

The results for CVRP show that, similar to TSP, using randomization can help solve some of the hardest instances faster. As we constrain the instances the performance gain tends to be smaller.

7.7.3 Results for Constrained Visual Program Synthesis

We consider the collection of 500 visual program synthesis problem instances that was used in Section 7.4. We compare the result of a standard complete anytime beam search to RR-CAB for three levels of γ that represent the constrainedness of the goal-condition. Figure 7.14 shows the results for RR-CAB with input noise injection, as described in Section 7.5. We can see that RR-CAB solves the hardest problems with a smaller search effort, reducing the heavy-tailed behavior. Again, we see that fixed-cutoff restarts tend to work better when using input noise injection.

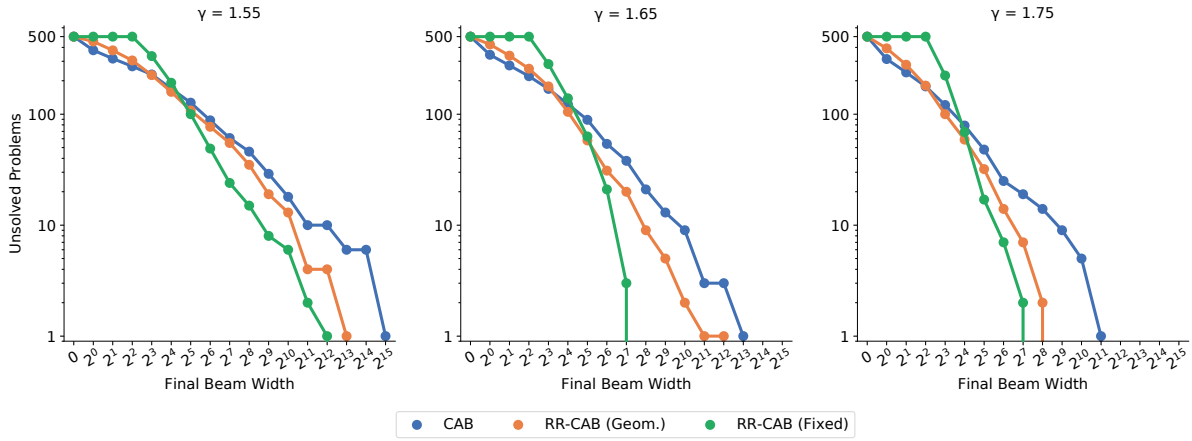


Figure 7.14: Visual Program Synthesis: Distribution of beam widths for 500 random instance for RR-CAB with input noise injection.

Next, we analyze the results for RR-CAB with beam search randomization based on SBS. Note that the number of potential expansions of each of the beam's candidate in this problem is much higher than the other problems (approximately 400, compared to 50-100 in the previous problems). Therefore, when using SBS for this problem, we only consider the top 50 extensions of each candidate. Practically, it is very unlikely that an extension of partial hypothesis that is not in the most likely 50 extensions will lead to a hypothesis that will be returned by the beam search. However, when applying randomization it may have the undesired outcome of promoting very low-ranked hypotheses and we therefore consider only the top 50 hypotheses.

Figure 7.15 shows the results for RR-CAB with SBS randomization. Again, we find that RR-CAB solves the hardest instances for a smaller search effort. Geometric restarts and fixed-cutoffs have approximately similar performance.

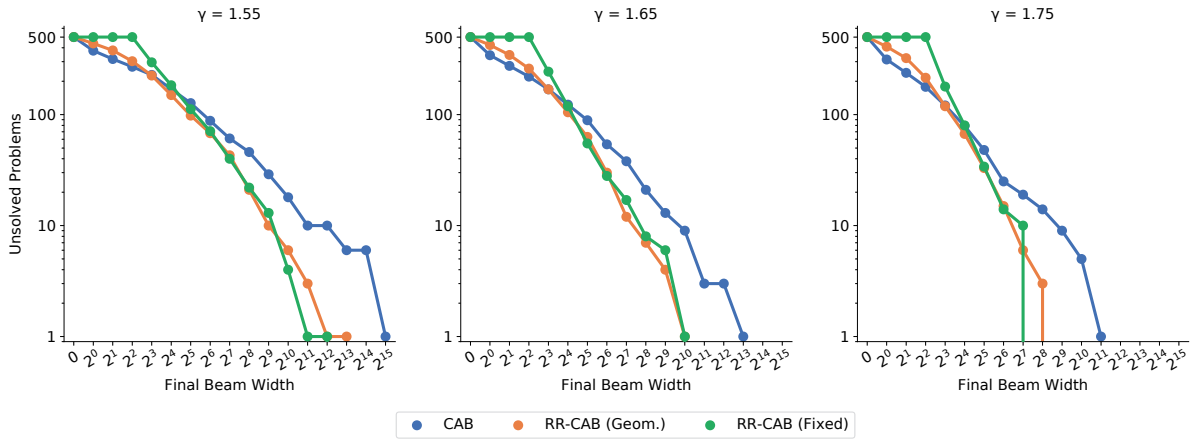


Figure 7.15: Visual Program Synthesis: Distribution of beam widths for 500 random instance for RR-CAB with SBS.

Consistent with previous results, we find that RR-CAB outperforms the baseline and solves some of the hardest instances faster. Unlike with TSP and CVRP, we find that the performance of SBS is close to the performance of noise injection (we discuss this difference in more detail in Section 7.8.2).

7.7.4 Results for Conditional Molecule Generation

We consider the set of 500 instances of conditional molecule generation with $QED = 0.9$, used in Section 7.4. We analyze the results for RR-CAB with three values of ρ that represent different levels of constrainedness of the goal-condition. Figure 7.16 shows the results for RR-CAB with input noise injection as described in Section 7.5. As with previous problems, RR-CAB solves the hardest problems for a smaller search effort, reducing the heavy-tailed behavior. And, as with previous problems, a fixed-cutoff restart policy tends to work better for input noise injection.

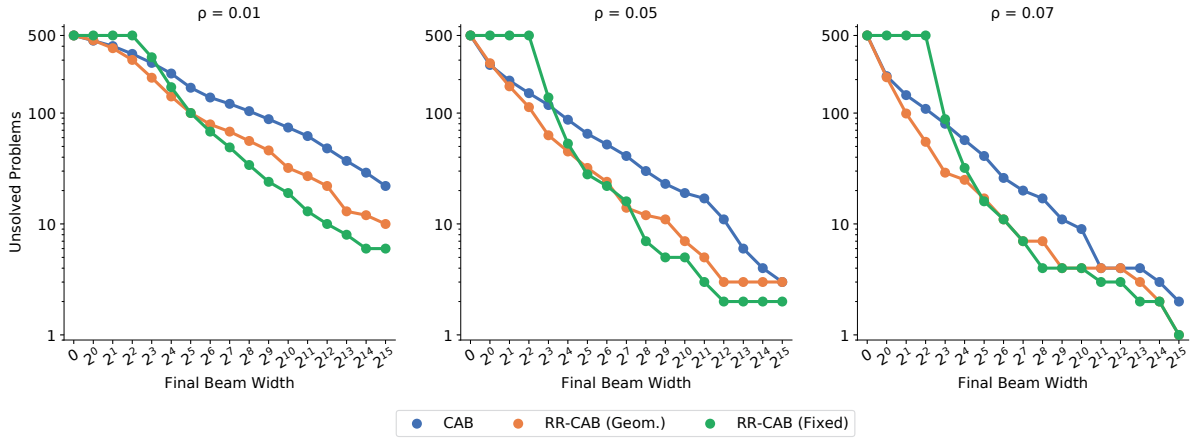


Figure 7.16: Conditional Molecule Generation: Distribution of beam widths for 500 random instance for RR-CAB with input noise injection.

Figure 7.17 shows similar analysis to Figure 7.16 when using beam search randomization based on SBS. While RR-CAB with SBS does seem to solve some hard problems for a smaller beam width, the

performance gain is relatively small compared to input noise injection and compared to previous problems. Still, we see that geometric restarts tend to work better with SBS.

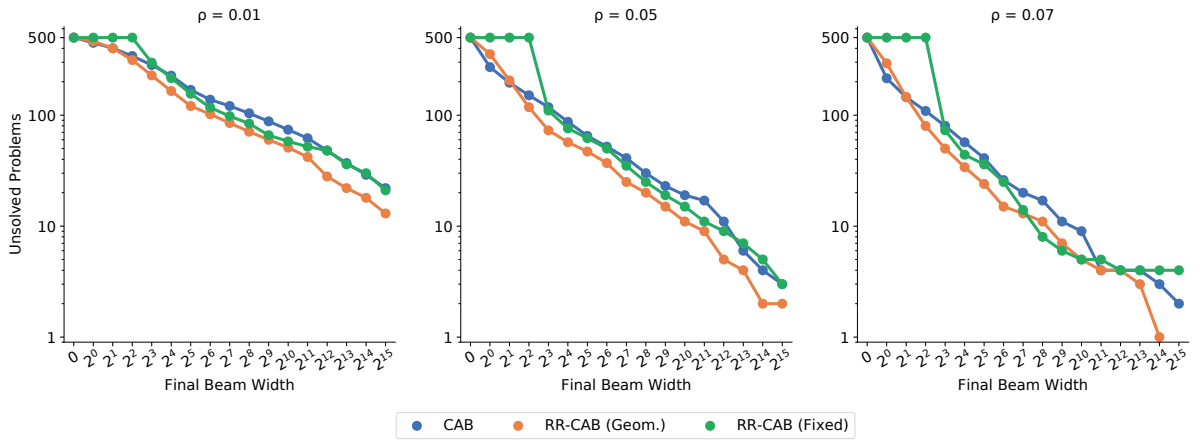


Figure 7.17: Conditional Molecule Generation: Distribution of beam widths for 500 random instance for RR-CAB with SBS.

The above results show that, similar to previous problems, RR-CAB outperforms the baseline and solves some of the hardest instances faster. In conditional molecule generation, we find that the performance gain from SBS is significantly smaller compared to noise injection.

7.8 Discussion and Future Work

Our empirical analysis shows that a fat- and heavy-tailed behavior, similar to the one observed for GBFS on planning problems in Chapter 4, can be observed for complete anytime beam search on goal-oriented neural sequence decoding problems. Inspired by our work on randomized restarts in GBFS, we propose RR-CAB, a randomized restarting variant of CAB, and show that it outperforms the baseline.

In Chapter 4, we showed that the heavy-tailed behavior of GBFS is due to the distribution of local minima h-depth. For goal-oriented beam search, we have yet to develop a well-defined notion of a local minimum. Whether the observed heavy-tailed behavior in beam search can be associated with a well-defined pattern of search behavior, similar to our analysis of local minima in GBFS, remains an open question (see detailed discussion in Section 8.3).

While RR-CAB reduces the search effort for hard problems, our results show that, unlike GBFS, the relaxed ensembles still contain very hard problems that could not be solved in the search effort limit even by RR-CAB (see, for example, Figures 7.10 and 7.11). This difference in behavior can potentially be associated with the relatively large differences observed within the ensembles: as noted in Section 7.5, not all instances exhibit similar variability in the performance of the randomized search procedure. This can be due to the difference in heuristic information (learned probabilities vs. cost-to-go) that may not be similarly effective across instances, differences in the constrainedness criteria, or differences in the search algorithm. Further investigation of the difference in behavior is an interesting direction for future work that may interact with the analysis of local minima described above.

In the remainder of this section, we discuss different modeling choices and their implications, as well as directions for future work.

7.8.1 Randomization and Restart Strategies

In Chapter 4, we used a randomized heuristic in order to randomize the heuristic search algorithm. In neural sequence decoding, the heuristic values that guide the search are conditional probabilities predicted by a deep neural network (see discussion in Section 6.6.2). Unlike with planning heuristics that represent distance to goal, we cannot simply add uniform noise to predicted probabilities. We therefore consider two techniques that can naturally lead to randomization of the predicted probabilities: input noise injection and SBS. While both techniques introduce randomization to the predicted probabilities, there are some important differences between them. A key limitation of the noise injection technique is that it needs to be tailored for each problem. In our work, we had to manually try different randomization approaches in order to find one that would generate sufficient variability on a single instance without making the problem significantly harder across different runs. In particular, the choices regarding the use of additive or multiplicative noise and the use of uniform or Gaussian distribution are tailored specifically to each problem and different choices are likely to have impact on the performance of our approach. Alternatively, an inherent limitation of SBS is that we are unable to guaranteed that repeated runs with different beam widths will maintain similar conditional probability distributions. The implication of this limitation is that we cannot analyze the search effort distribution of SBS on a single problem instance, as we do for beam search with noise injection in Section 7.5.

The differences in performance between the two techniques and in particular, the differences in performance for each of the restart strategies suggest that different types of randomization may interact differently with problems and restart strategy. Development of empirical models for a deeper analysis of the impact of each randomization technique is an interesting direction for future work.

As our approach of injecting noise to the inputs of the neural network requires a tailored solution for each problem, it is interesting to investigate generic ways of introducing noise into the decoding process. Potential research directions include applying noise to hidden units [143, 20] or using dropout [168] in inference.

In this work, we focused on two well known restart strategies: fixed-cutoff and geometric restarts. Research work in the area of combinatorial optimization has considered more advanced restart strategies such as dynamic and learning restart policies (e.g., [102]) that are dynamically updated in real-time. Investigating ways to incorporate such policies in RR-CAB is an interesting direction for future work.

7.8.2 Softmax Temperature

When using sampling for decoding, it is common to use softmax temperature (see Section 5.2.1) to control the randomness. In SBS, Kool et al. [107] noted that the softmax temperature can control the diversity of decoded solutions: higher temperature results in a less concentrated distribution and higher diversity. Even in standard beam search, softmax temperature has been used to change the conditional distributions during the decoding to generate more or less diversity in the outputs (e.g., [22]).

In our study, we find that RR-CAB with SBS tends to underperform RR-CAB with noise injection. A potential explanation is that RR-CAB with SBS exhibits smaller variability compared to noise injection and therefore is less likely to significantly reduce the search effort of the hardest instances. Using softmax temperature will increase the level of randomization of SBS and can, potentially, increase the variability in the distribution of search effort. While our current study is conducted using the default temperature value ($\mathcal{T} = 1$), investigating the impact of softmax temperature on the performance of RR-CAB with

different restart strategies is an interesting direction for future work.

To further motivate the interest in softmax temperature, we note that our analysis found that the different models in our study exhibit different levels of concentration in their predicted (conditional) distributions. As an example, Table 7.1 reports the mean token conditional probability in the solutions of a greedy search for each of the problems. We can see that the average token probability in visual program synthesis has a much lower mean and higher variance, while the average token probability in TSP and CVRP has very high mean and lower variance. These results can potentially explain the difference in the performance of SBS between visual program synthesis and the other problems: the high level of uncertainty is likely to result in higher variability in the sequences generated by SBS and potentially in the distribution of search effort.

Table 7.1: Mean (conditional) token probability in greedy search solutions.

Problem	Token Probability (Mean ± SD)
Travelling Salesman Problem (TSP)	0.955 ± 0.108
Capacitated Vehicle Routing Problem (CVRP)	0.948 ± 0.115
Visual Program Synthesis	0.382 ± 0.300
Conditional Molecule Generation	0.849 ± 0.209

7.8.3 Parallelization Implementation

A key feature of using beam search for decoding sequences from deep sequence models is its ability to be parallelized on a GPU. In our empirical evaluation, we present the results for the fixed-cutoff restart strategy by batching together beam searches and comparing these results to the corresponding final beam width of a geometric strategy. As we start investigating more complicated restart strategies, such as Luby's universal strategy [124], we will not be able to batch the results together to maintain comparability. Furthermore, even in our comparison, it is not clear that a set of four beam search instances, each with a beam width of 8 and executed together on a GPU, are comparable to one beam search with a beam width of 32.

Our work raises the need for well-defined evaluation metrics that can be used to compare the results of complete beam searches with different restart strategies, even when it not possible to batch together runs as we currently do. Furthermore, due the centrality of parallelization in neural sequence decoding, we believe that investigating parallelization-friendly restart strategies is an important direction for future work.

7.9 Conclusion

In this chapter, we investigate whether a fat- and heavy-tailed behavior, similar to the one we observe in Chapter 4 for GBFS in planning problems, is observed for complete anytime beam search in goal-oriented neural sequence decoding. We perform an extensive empirical analysis, across four goal-oriented benchmark problems, and find fat- and heavy-tailed behavior in the distribution of search efforts of beam search. Inspired by our analysis of randomized restarts in GBFS, we propose a randomized restarting variant of complete anytime beam search, RR-CAB, and study the impact of different randomization

techniques and restart strategies. Our experiments show that RR-CAB reduces the search effort for some of the hardest instances and outperforms the baseline. The work in this chapter further support our thesis statement by demonstrating how empirical models can help explain algorithm performance and inform more efficient search algorithms. Furthermore, this chapter demonstrates that our analysis on GBFS for satisficing planning can be adapted to neural sequence decoding using beam search and suggests that it may be applicable to other tasks and additional heuristic search algorithms.

Chapter 8

Concluding Remarks

8.1 Summary

In this dissertation, we developed several empirical models for the behavior of heuristic search algorithms in AI planning and neural sequence decoding. Inspired by work on CSPs and SAT and consistent with our thesis statement, we showed that empirical models can help explain the performance of greedy best first search and beam search and inform the design of more efficient search algorithms.

In Part I, we developed empirical models for problem difficulty in satisficing AI planning using GBFS. We established the existence of a rapid phase transition in problem solubility as we vary the constrainedness of problem and an associated pattern of easy-hard-easy that peaks at the phase transition. We also showed that the phase transition phenomenon interacts with other factors that impact the difficulty of planning problems. To further understand the connection between constrainedness and problem difficulty, we studied the distribution of local minima encountered in the search. We established an exponential correlation between the depth of encountered local minima and the associated search effort. The constrainedness of problems controls the distribution of local minima depths that, in turn, impacts the problem difficulty. Our analysis explains many of the observed phenomena on the behavior of GBFS, including the impact of node re-expansions, operator cost ratio, the existence of exceptionally hard problems, and the correlation between the heuristic values and the distance to the goal. Building on our analysis, we proposed a randomized variant of GBFS that exploits the distribution of local minima depths and outperforms the baseline.

In Part II, we developed empirical models for the behavior of beam search in neural sequence decoding. We first studied the well-known phenomenon of performance degradation in beam search. We presented an explanatory model that is based on search discrepancies and demonstrate how this model can inform the design of heuristic techniques that successfully mitigate the performance degradation. Then, in Chapter 7, we studied the patterns of problem difficulty in goal-oriented neural sequence decoding using a complete variant of beam search. Inspired by our analysis in Part I, we showed that complete anytime beam search exhibits a heavy-tailed behavior similar to GBFS. Based on our analysis, we proposed a randomized variant that, similar to our randomized variant of GBFS, outperforms the baseline.

Throughout the dissertation, we followed the approach described in Section 1.1. We started by designing an analytical framework that contains all the components needed to conduct experiments that study the behavior of the search algorithm. Using this framework we conducted an extensive set of

experiments that involve varying different parameters of the search algorithm and recording the results. Then, building on our empirical analysis, we developed empirical models that explain the observed performance and support the design of more efficient heuristic search algorithms.

We note that all of the empirical models presented in the work are inspired by existing works in the combinatorial search literature. The work in this dissertation suggest that many of the ideas developed for combinatorial search problems that consist of searching for an assignment of variables and are typically solved by a tree search algorithm could be adapted to heuristic search algorithms such as greedy best first search and beam search.

8.2 Contributions

The main contributions of this dissertation are summarized as follows.

8.2.1 Phase Transition and Problem Difficulty in Domain-Specific Heuristic Search (Chapter 3)

1. We introduce the tool of phase transition to heuristic search using an abstract model of a heuristic search problem that is based on random graphs and demonstrate an abrupt transition in problem solubility as a parameter controlling the density of the transitions in the state space is varied.
2. We show that the abrupt transition in the problem solubility is accompanied by an easy-hard-easy pattern of problem difficulty across the transition region.
3. Exploiting our random graph model, we provide analytical bounds on the "mushy region" between the fully soluble and fully insoluble problems.
4. We demonstrate how to transfer the abstract graph model to existing heuristic search benchmark problems, allowing the generation of versions of each problem across the phase transition and demonstrating both the phase transition and the easy-hard-easy pattern on five standard heuristic search benchmarks.
5. We study the behavior of systematically stronger heuristics across the phase transition region and show that the reduction in search effort for strong heuristics is orders of magnitude *smaller* for the hard soluble instances at the phase transition than in the underconstrained regions.
6. We show that the number of node re-expansions peaks in the phase transition and declines outside.
7. We demonstrate that the effect of large operator cost ratio on the search effort is most significant in the phase transition region and diminishes outside.
8. We show that *exceptionally hard problems* [52, 165] appear in the relaxed region of the phase transition where the median effort is relatively low.

8.2.2 Heavy-tailed Behavior and Randomization in Satisficing Planning (Chapter 4)

1. We show that fat- and heavy-tailed behavior can be observed on ensembles of random planning problems across different domains and heuristic functions.

2. We present a variant of greedy best-first search that introduces a limited amount of randomization in the search procedure and show that heavy-tailed behavior can be observed in multiple runs of the randomized search procedure on a single problem instance.

3. We demonstrate how different notions of constrainedness that are commonly used in the modeling of planning problems can lead to a fat- or heavy-tailed distribution of search effort.

4. We introduce the notion of local minimum h-depth and find an exponential correlation between the h-depth of the single deepest local minimum encountered and the total search effort (i.e., number of expanded nodes).

5. We show that the distribution of local minima h-depth in planning problems depends on the constrainedness of problems, and that heavy-tailed behavior appears when there is a low, but non-negligible, probability of encountering a deep local minimum during search.

6. We show that recent methods of non-greedy random exploration can help reduce the heavy-tailed behavior in a similar manner to randomized restarts in CSPs.

7. Inspired by combinatorial search, we propose *RR-GBFS*, a randomized restarting GBFS that outperforms GBFS by escaping deep local minima.

8.2.3 Empirical Analysis the Beam Search Performance Degradation in Neural Sequence Decoding (Chapter 6)

1. We introduce an explanatory model of the beam search performance degradation in neural sequence models that is based on the existing concept of *search discrepancies* that represent deviations from greedy choices.

2. We conduct an extensive empirical study of the distribution of search discrepancies and show that increasing the beam width leads to solutions with more and larger discrepancies early in the sequence. These sequences often have lower evaluation score, leading to the observed performance degradation. As we increase the beam width, the differences in the distribution of discrepancies that are associated with improved vs. degraded solutions grow substantially.

3. We show, empirically, that our explanatory model generalizes previous the observations on "copies" in machine translation and predictions that repeat training set targets in image captioning and accounts for more of the degraded predictions.

4. Based on our empirical analysis, we propose two heuristics for constraining the beam search from considering large search discrepancies and show that these heuristics eliminate the performance degradation.

8.2.4 Randomized Restarting Beam Search in Goal-Oriented Neural Sequence Decoding (Chapter 7)

1. We consider the setting of goal-oriented neural sequence decoding, where the decoded sequence must satisfy a goal condition. We consider four neural decoding tasks for which a selected evaluation metric is available at decoding time and develop a goal-oriented variant of these problems where the goal condition enforces bounded suboptimality with respect to the evaluation metric.

2. We show that for goal-oriented neural sequence problems, complete anytime beam search exhibits a fat- or heavy-tailed behavior on ensembles of relaxed problems, similar to the behavior exhibited for GBFS.

3. We consider a randomized variant of beam search that is based on noise injection to the inputs of the neural network and show that randomized complete anytime beam search exhibits fat- or heavy-tailed behavior on ensembles of multiple runs on a single instance.

4. Inspired by our results for GBFS, we introduce a randomized restarting variant of complete anytime beam search and show that it solves some of the hardest instances faster and outperforms the baseline.

5. We conduct extensive empirical evaluation and analyze the impact of different parameters including the constrainedness of the goal-criteria, the restart policy, and the type of randomization on the effectiveness of our method.

8.3 Future Work

As immediate directions for future work have been discussed in the respective chapters (see Sections 3.8, 4.6.3, 6.6, and 7.8), in this section, we take a broader view and propose several lines of research that arise from the work presented in the dissertation.

8.3.1 Local Minima vs. Plateaus in AI Planning

Plateaus and local minima are two of the main factors that have negative impact on the problem difficulty for GBFS [204]. Both local minima and plateaus are instances of uninformative heuristic regions (UHR). However, they represent different challenge for GBFS. Wilt and Ruml [197] note that heuristic plateaus can sometimes be mitigated by tie breaking, but local minima cannot be avoided by standard greedy best-first search. Still, plateaus represent a significant challenge in satisficing planning [176, 205]. While several works have studied the negative impact of these factors and proposed different techniques to overcome local minima and plateaus, little work has focused on understanding when each of these phenomena appear.

In this dissertation, we have provided a deeper understanding of the local minima phenomenon. In Chapter 3, we hypothesized a connection between the constrainedness of problems and the structure of local minima in GBFS. Our hypothesis was informed by our analysis of the operator cost ratio and the discovery of exceptionally hard problems. In Chapter 4 we establish our hypothesis by showing that the distribution of local minima h-depth depends on the constrainedness of problems. Since h-depth is shown to be highly correlated with search effort, this result has a direct impact on problem difficulty.

Similar to local minima, heuristic plateaus are one of the central factors that impact problem difficulty in GBFS and their study is crucial to the development of a deep and comprehensive understanding of problem difficulty in satisficing planning. Our work in Chapter 4 suggests that developing empirical models for heuristic plateaus can be a useful way of deepening our understanding of this phenomenon and inspiring new algorithmic enhancements that address such plateaus.

Informed by our analysis of local minima, a particularly interesting research question is whether the structure of heuristic plateaus encountered in the search is also empirically connected to the constrainedness

of problems. In Chapter 3, we compare a set of systematically stronger heuristics across the phase transition and find that the impact of more informed heuristics is orders of magnitudes higher in the relaxed region of the phase transition. According to our definition of the set of systematically stronger heuristics, both for the abstract model and for the benchmarks, a less informed heuristic has a bound on its heuristic value that leads to a large plateau of states whose heuristic value is above some threshold. While these results are not sufficient to characterize the connection between plateaus and problem constrainedness, they suggest that a connection is likely to exist and that the negative impact of plateaus, unlike local minima, could be stronger in relaxed problems. We believe that a careful investigation of the connection between the structure of plateaus and problem constrainedness is an interesting direction for future work.

8.3.2 Understanding the Impact of Different Randomization Techniques

In this dissertation, we show that randomization can be used to boost the performance of GBFS and beam search by reducing the fat- and heavy-tailed behavior. In Chapter 4 we compared both randomized restarts (using a randomized heuristic) and two variants of GBFS that incorporate random exploration in the search. Our results shows that these techniques reduce the heavy-tailed behavior, however the performance for each of the approaches was different. In particular, we found that, in some domains, the variants that incorporate randomized exploration reduced the search effort even in the non-heavy-tailed regime.

In Chapter 7, we compare two techniques to randomize the beam search in the randomized restarting complete anytime beam search (RR-CAB) algorithm: input noise injection and stochastic beam search (SBS). Our empirical analysis shows that while both techniques could help reduce the search effort for the hardest instances, their performance is not identical. In particular, we found that input noise injection tends to work better with a restarting strategy based on fixed-cutoff values, while SBS tends to work better with geometric restarts. However, we found that the differences between the different restart strategies change as we vary the goal-condition constrainedness.

The results in both Chapters 4 and 7 suggest that different randomization techniques may benefit heuristic search algorithm in different ways. Specifically, it seems that the impact of randomization on the search effort depends not only on the constrainedness of problems, but also on the type of randomization. Investigating the interaction between constrainedness and randomization is an interesting direction for future work. Another potential research direction concerns the impact of the different randomization techniques on problems that, instead of deep local minima, suffer from large heuristic plateaus (see above discussion on heuristic plateaus).

8.3.3 Local Minima in Neural Sequence Decoding using Beam Search

In Chapter 4, we formally defined the notion of local minima that were encountered in search. These local minima represent regions of the state spaces that do not participate in the solution, however, they need to be exhausted by GBFS due to their heuristic evaluation. The search effort associated with these local minima is shown to be highly correlated to the backtracking behavior of the search.

Unlike GBFS, beam search is not a complete search algorithm in itself. If the search is misled into local minima, i.e., all the nodes in the beam do not lead to a goal, beam search will not be able to find a solution. Therefore, complete anytime beam search, the complete variant of beam search considered in

Chapter 7, consists of repeatedly restarting beam search with a wider beam until a solution is found. In this setting, the definition of local minima from GBFS is not applicable. First, there is no backtracking as done in GBFS and the way to overcome a local minimum is to restart with a large enough beam width. Second, at each step the beam consists of multiple nodes, each can be in a different local minimum. Third, neural sequence decoding uses predicted probabilities that are different from the estimations of distance-to-goal used in GBFS.

The formal definition and empirical characterization of local minima in CAB remain open questions. While the definition used in GBFS cannot be applied to beam search, we believe the core idea behind local minima is still relevant in the analysis of problem difficulty in CAB. These local minima would represent regions of the state space, induced by the heuristic estimations, that have to be exhausted by a sufficiently large beam width. In GBFS, we found that the existence and structure of local minima is connected to the constrainedness of problems, and that the change in the distribution of local minima depth accounts for the observed heavy-tailed behavior. In Chapter 7, we showed the impact of constrainedness on problem difficulty of complete anytime beam search, as well as the existence of heavy-tailed behavior. Building on our results in Chapter 4, we believe that an analysis that is based on an adaptation of the local minima notion to beam search can help explain the observed patterns of problem difficulty.

A related direction for future work consists of analyzing the patterns of problem difficulty in complete variants of beam search that are based on backtracking (see Section 5.3.1). These variants use backtracking instead of restarting with a wider beam in order to make beam search complete. As a first step, we need to check whether the heavy-tailed behavior observed for CAB also exists in backtracking beam search variants. Then, we can see if our definitions of local minima and h-backtracks could be adapted to a backtracking beam search.

8.3.4 The Use of MLE Training and MAP Inference in Neural Sequence Decoding

Recent work has highlighted challenges that are associated with inference on neural sequence models including the length bias [105, 206, 133], "copies" in machine translation [138], and lack of novelty in image captioning [183]. Our analysis in Chapter 6 shows that several of these known issues in neural sequence decoding are instances of a more general problem in MAP inference using beam search on MLE-trained models. We also make the connection between the observed problem and the existence of two well-known biases in the models: exposure bias and label bias. Investigating new models and training schemes that are robust to these biases is a direction for future work. In particular, it is interesting to investigate whether recently proposed models such as the transformer model [178] and new training methods such as minimum risk training [163] are robust, or at least less sensitive, to these biases.

Our study of goal-oriented neural sequence decoding in Chapter 7 highlights a different type of task for which the use of MLE training and MAP inference might not be the most appropriate: first, the beam search algorithm evaluates all the candidates in the beam and does not only choose the most likely one; second, the combinatorial nature of many of these problems can lead to different sequences that may represent equivalent solutions and, alternatively, we can have a pair of very similar sequences with one satisfying the goal criteria and the other does not. However, existing approaches consist of models that are trained to maximize the probability of a reference solution and are known to generate M-best lists that tend to have very similar solutions [59, 114, 115]. It is therefore interesting to investigate alternative training methods that might be more suitable for goal-oriented neural sequence decoding tasks. It is

particularly interesting to study the impact of training techniques that are aware of the whole set of candidates in the beam, e.g., Wiseman and Rush [200], on the performance of goal-oriented tasks and the heavy-tailed behavior we observe in Chapter 7.

In a recent work on neural machine translation (NMT), Eikema and Aziz [40] suggest that some of the known pathologies in NMT are due to the use of MAP decoding and not to due to NMT as a model or due to its training algorithm, maximum likelihood estimation (MLE). They claim that the core problem is the attempt to identify the highest-probability hypothesis, i.e., the mode, under the model distribution. To support their hypothesis, they show, empirically, that the most likely hypotheses according to the model distribution accumulate a very small probability mass and claim that the mode can therefore be considered essentially arbitrary. Instead, they suggest that we should base decisions on statistics gathered from the distribution holistically. They present experimental results for minimum Bayes risk (MBR) decoding [109] and show that it outperforms beam search. It is interesting to investigate whether this type of approach is useful beyond NMT, in particular in goal-oriented neural sequence decoding. Furthermore, developing empirical models for such new approaches that are able to explain the observed improvement and account for the mistakes that still appear is an interesting direction for future work.

Bibliography

[1] Dimitris Achlioptas, Michael SO Molloy, Lefteris M Kirousis, Yannis C Stamatiou, Evangelos Kranakis, and Danny Krizanc. Random constraint satisfaction: A more accurate picture. *Constraints*, 6(4):329–344, 2001.

[2] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Guided open vocabulary image captioning with constrained beam search. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 936–945, 2017.

[3] Masataro Asai and Alex Fukunaga. Exploration among and within plateaus in greedy best-first search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2017.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Matej Balog, Alexander Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. In *International Conference on Learning Representations (ICLR)*, 2017.

[6] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[7] Dhruv Batra, Payman Yadollahpour, Abner Guzman-Rivera, and Gregory Shakhnarovich. Diverse m-best solutions in markov random fields. In *European Conference on Computer Vision (ECCV)*, pages 1–16. Springer, 2012.

[8] J Christopher Beck and Laurent Perron. Discrepancy-bounded depth first search. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, pages 8–10, 2000.

[9] J Christopher Beck and Jean-Paul Watson. Adaptive search algorithms and fitness-distance correlation. In *The Fifth Metaheuristics International Conference*, volume 83. Citeseer, 2003.

[10] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[11] G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90, 2012.

[12] Roberto Bisiani. Beam search. In SC Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 56–58. Wiley & Sons, 1987.

[13] Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *European Conference on Planning*, pages 360–372. Springer, 1999.

[14] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001.

[15] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[16] Tom Bylander. A probabilistic analysis of propositional STRIPS planning. *Artificial Intelligence*, 81(1):241–271, 1996.

[17] René Carmona. *Statistical analysis of financial data in R.* Springer, second edition, 2014.

[18] Peter Cheeseman, Bob Kanefsky, and William M Taylor. Where the really hard problems are. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 91, pages 331–337, 1991.

[19] Colin Cherry and George Foster. Batch tuning strategies for statistical machine translation. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 427–436. Association for Computational Linguistics, 2012.

[20] Kyunghyun Cho. Noisy parallel approximate decoding for conditional recurrent language model. *arXiv preprint arXiv:1605.03835*, 2016.

[21] Sumit Chopra, Michael Auli, and Alexander M Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 93–98, 2016.

[22] Jan Chorowski and Navdeep Jaitly. Towards better decoding and language model integration in sequence to sequence models. In *Annual Conference of the International Speech Communication Association (Interspeech)*, pages 523–527, 2017.

[23] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[24] Eldan Cohen and Christopher Beck. Empirical analysis of beam search performance degradation in neural sequence models. In *International Conference on Machine Learning (ICML)*, pages 1290–1299, 2019.

[25] Eldan Cohen and J Christopher Beck. Cost-based heuristics and node re-expansions across the phase transition. In *International Symposium on Combinatorial Search (SoCS)*, pages 11–19, 2017.

[26] Eldan Cohen and J Christopher Beck. Problem difficulty and the phase transition in heuristic search. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 780–786, 2017.

[27] Eldan Cohen and J Christopher Beck. Fat- and heavy-tailed behavior in satisficing planning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 6136–6143, 2018.

[28] Eldan Cohen and J Christopher Beck. Local minima, heavy tails, and search effort for GBFS. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 4708–4714, 2018.

[29] James M Crawford and Larry D Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81(1):31–57, 1996.

[30] Joseph C Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[31] William Cushing, J Benton, and Subbarao Kambhampati. Cost based search considered harmful. In *International Symposium on Combinatorial Search (SoCS)*, pages 140–141, 2010.

[32] William Cushing, J Benton, and Subbarao Kambhampati. Cost based satisficing search considered harmful. *arXiv preprint arXiv:1103.3687*, 2011.

[33] Thomas L Dean and Mark S Boddy. An analysis of time-dependent planning. In *National Conference on Artificial Intelligence (AAAI)*, volume 88, pages 49–54, 1988.

[34] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.

[35] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, pages 170–181, 2018.

[36] James E Doran and Donald Michie. Experiments with the graph traverser program. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 294(1437):235–259, 1966.

[37] Harry Dweighter. Problem e2569. *American Mathematical Monthly*, 82(10), 1975.

[38] Stefan Edelkamp. Planning with pattern databases. In *European Conference on Planning*, 2014.

[39] Sergey Edunov, Myle Ott, Michael Auli, David Grangier, et al. Classical structured prediction losses for sequence to sequence learning. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 355–364, 2018.

[40] Bryan Eikema and Wilker Aziz. Is map decoding all you need? the inadequacy of the mode in neural machine translation. *arXiv preprint arXiv:2005.10283*, 2020.

[41] Paul Embrechts, Thomas Mikosch, and Claudia Klüppelberg. *Modelling Extremal Events: For Insurance and Finance*. Springer-Verlag, 1997.

[42] Paul Erdős and Alfréd Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[43] Gaojian Fan, Martin Müller, and Robert Holte. The two-edged nature of diverse action costs. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 98–106, 2017.

[44] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[45] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[46] Alan Frieze and Michał Karoński. *Introduction to random graphs*. Cambridge University Press, 2015.

[47] David Furcy and Sven Koenig. Limited discrepancy beam search. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 125–131, 2005.

[48] William H Gates and Christos H Papadimitriou. Bounds for sorting by prefix reversal. *Discrete mathematics*, 27(1):47–57, 1979.

[49] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning (ICML)*, pages 1243–1252, 2017.

[50] Ian P Gent, Ewan MacIntyre, Patrick Presser, Barbara M Smith, and Toby Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 179–193. Springer, 1996.

[51] Ian P Gent, Ewan MacIntyre, Patrick Prosser, and Toby Walsh. The constrainedness of search. In *National Conference on Artificial Intelligence (AAAI)*, pages 246–252, 1996.

[52] Ian P Gent and Toby Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70(1-2):335–345, 1994.

[53] Ian P Gent and Toby Walsh. The hardest random SAT problems. In *KI-94: Advances in Artificial Intelligence*, pages 355–366. Springer, 1994.

[54] Ian P Gent and Toby Walsh. The SAT phase transition. In *European Conference on Artificial Intelligence (ECAI)*, pages 105–109, 1994.

[55] Ian P Gent and Toby Walsh. The TSP phase transition. *Artificial Intelligence*, 88(1):349–358, 1996.

[56] Ian P Gent and Toby Walsh. Beyond NP: the QSAT phase transition. In *National Conference on Artificial Intelligence (AAAI)*, pages 648–653, 1999.

[57] Alfonso E Gerevini, Alessandro Saetti, and Ivan Serina. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence*, 172(8-9):899–944, 2008.

[58] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 2000.

[59] Kevin Gimpel, Dhruv Batra, Chris Dyer, and Gregory Shakhnarovich. A systematic exploration of diversity in machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1100–1111, 2013.

[60] Carla Gomes. Randomized backtrack search. In Michela Milano, editor, *Constraint and integer programming: Toward a unified methodology*, pages 233–291. Springer Science & Business Media, 2003.

[61] Carla Gomes and Toby Walsh. Randomness and structure. In Francesca Rossi, Peter Van Beek, and Toby Walsh, editors, *Handbook of constraint programming*, pages 639–664. Elsevier, 2006.

[62] Carla P Gomes, Cèsar Fernández, Bart Selman, and Christian Bessière. Statistical regimes across constrainedness regions. *Constraints*, 10(4):317–337, 2005.

[63] Carla P Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[64] Carla P Gomes, Bart Selman, and Nuno Crato. Heavy-tailed distributions in combinatorial search. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 121–135. Springer, 1997.

[65] Carla P Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of automated reasoning*, 24(1):67–100, 2000.

[66] Carla P Gomes, Bart Selman, Henry Kautz, et al. Boosting combinatorial search through randomization. *National Conference on Artificial Intelligence (AAAI)*, 98:431–437, 1998.

[67] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[68] David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword. *Linguistic Data Consortium, Philadelphia*, 4(1):34, 2003.

[69] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[70] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1948.

[71] Edwin C Harrington. The desirability function. *Industrial quality control*, 21(10):494–498, 1965.

[72] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[73] William D Harvey and Matthew L Ginsberg. Limited discrepancy search. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 607–615, 1995.

[74] Eva Hasler, Adrià de Gispert, Gonzalo Iglesias, and Bill Byrne. Neural machine translation decoding with terminology constraints. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 506–512, 2018.

[75] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *National Conference on Artificial Intelligence (AAAI)*, pages 1007–1012, 2007.

[76] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[77] Malte Helmert. Landmark heuristics for the pancake problem. In *International Symposium on Combinatorial Search (SoCS)*, pages 109–110, 2010.

[78] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: what's the difference anyway? In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

[79] Malte Helmert and Héctor Geffner. Unifying the causal graph and additive heuristics. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 140–147, 2008.

[80] Malte Helmert, Patrik Haslum, Jörg Hoffmann, et al. Flexible abstraction heuristics for optimal sequential planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 176–183, 2007.

[81] Manuel Heusner, Thomas Keller, and Malte Helmert. Understanding the search behaviour of greedy best-first search. In *International Symposium on Combinatorial Search (SoCS)*, 2017.

[82] Manuel Heusner, Thomas Keller, and Malte Helmert. Best-case and worst-case behavior of greedy best-first search. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018.

[83] Manuel Heusner, Thomas Keller, and Malte Helmert. Search progress and potentially expanded states in greedy best-first search. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018.

[84] Mohammad H Heydari and I Hal Sudborough. On the diameter of the pancake network. *Journal of Algorithms*, 25(1):67–94, 1997.

[85] Bruce M Hill. A simple general approach to inference about the tail of a distribution. *The annals of statistics*, 3(5):1163–1174, 1975.

[86] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[87] Jörg Hoffmann, Carla P Gomes, Bart Selman, and Henry A Kautz. SAT encodings of state-space reachability problems in numeric domains. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1918–1923, 2007.

[88] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[89] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

[90] Tad Hogg and Colin P Williams. The hardest constraint problems: A double phase transition. *Artificial Intelligence*, 69(1-2):359–377, 1994.

[91] Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In *Association for Computational Linguistics (ACL)*, pages 1535–1546, 2017.

[92] John N Hooker. Needed: An empirical science of algorithms. *Operations research*, 42(2):201–212, 1994.

[93] Jonathan RM Hosking. L-moments: Analysis and estimation of distributions using linear combinations of order statistics. *Journal of the Royal Statistical Society. Series B (Methodological)*, 52(1):105–124, 1990.

[94] Bernardo A Huberman and Tad Hogg. Phase transitions in artificial intelligence systems. *Artificial Intelligence*, 33(2):155–171, 1987.

[95] Tatsuya Imai and Akihiro Kishimoto. A novel technique for avoiding plateaus of greedy best-first search in satisficing planning. In *International Symposium on Combinatorial Search (SoCS)*, 2011.

[96] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning (ICML)*, pages 2323–2332, 2018.

[97] Wengong Jin, Kevin Yang, Regina Barzilay, and Tommi Jaakkola. Learning multimodal graph-to-graph translation for molecule optimization. In *International Conference on Learning Representations (ICLR)*, 2018.

[98] Seokho Kang and Kyunghyun Cho. Conditional molecular design with deep generative models. *Journal of chemical information and modeling*, 59(1):43–52, 2018.

[99] Richard M Karp. The transitive closure of a random digraph. *Random Structures & Algorithms*, 1(1):73–93, 1990.

[100] Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1728–1733, 2009.

[101] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3128–3137, 2015.

[102] Henry Kautz, Eric Horvitz, Yongshao Ruan, Carla Gomes, and Bart Selman. Dynamic restart policies. *National Conference on Artificial Intelligence (AAAI)*, pages 674–681, 2002.

[103] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 6348–6358, 2017.

[104] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. Opennmt: Open-source toolkit for neural machine translation. *ACL: System Demonstrations*, pages 67–72, 2017.

[105] Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. In *Workshop on Neural Machine Translation and Generation (WNMT)*, pages 28–39, 2017.

[106] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.

[107] Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning (ICML)*, pages 3499–3508, 2019.

[108] Richard E Korf. Improved limited discrepancy search. In *National Conference on Artificial Intelligence (AAAI)*, pages 286–291, 1996.

[109] Shankar Kumar and William Byrne. Minimum bayes-risk decoding for statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 169–176, 2004.

[110] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

[111] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2019.

[112] Greg Landrum. Rdkit: Open-source cheminformatics. `http://www.rdkit.org`.

[113] Levi HS Lelis, Sandra Zilles, and Robert C Holte. Stratified tree search: a novel suboptimal heuristic search algorithm. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 555–562, 2013.

[114] Jiwei Li and Dan Jurafsky. Mutual information and diverse decoding improve neural machine translation. *arXiv preprint arXiv:1601.00372*, 2016.

[115] Jiwei Li, Will Monroe, and Dan Jurafsky. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*, 2016.

[116] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

[117] Chin-Yew Lin and Franz Josef Och. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Association for Computational Linguistics (ACL)*, page 605. Association for Computational Linguistics, 2004.

[118] Chin-Yew Lin and Franz Josef Och. Orange: a method for evaluating automatic evaluation metrics for machine translation. In *International Conference on Computational Linguistics (COLING)*, page 501, 2004.

[119] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

[120] Nir Lipovetzky. Structure and inference in classical planning. *AI Access*, 2014.

[121] Nir Lipovetzky and Hector Geffner. Width and serialization of classical planning problems. In *European Conference on Artificial Intelligence (ECAI)*, pages 540–545, 2012.

[122] Nir Lipovetzky and Hector Geffner. Best-first width search: Exploration and exploitation in classical planning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

[123] Yunchao Liu, Zheng Wu, Daniel Ritchie, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to describe scenes with programs. In *International Conference on Learning Representations (ICLR)*, 2018.

[124] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

[125] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 3086–3094, 2014.

[126] Ciaran McCreesh, Patrick Prosser, and James Trimble. Heuristics and really hard instances for subgraph isomorphism problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 631–638, 2016.

[127] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.

[128] Drew M McDermott. The 1998 ai planning systems competition. *AI magazine*, 21(2):35–35, 2000.

[129] Stephan Mertens. Phase transition in the number partitioning problem. *Physical Review Letters*, 81(20):4281, 1998.

[130] Tomáš Mikolov. *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology, 2012.

[131] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In *National Conference on Artificial Intelligence (AAAI)*, pages 459–465, 1992.

[132] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400(6740):133–137, 1999.

[133] Kenton Murray and David Chiang. Correcting length bias in neural machine translation. In *Conference on Machine Translation: Research Papers*, pages 212–223, 2018.

[134] Hootan Nakhost, Jörg Hoffmann, and Martin Müller. Resource-constrained planning: A monte carlo random walk approach. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

[135] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar GuÌ‡lçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, pages 280–290, 2016.

[136] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 9839–9849, 2018.

[137] Graham Neubig, Makoto Morishita, and Satoshi Nakamura. Neural reranking improves subjective quality of machine translation: Naist at wat2015. In *Proceedings of the 2nd Workshop on Asian Translation (WAT2015)*, pages 35–41, 2015.

[138] Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. Analyzing uncertainty in neural machine translation. In *International Conference on Machine Learning (ICML)*, pages 3953–3962, 2018.

[139] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *NAACL-HLT 2019: Demonstrations*, 2019.

[140] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Association for Computational Linguistics (ACL)*, pages 311–318, 2002.

[141] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning (ICML)*, pages 1310–1318, 2013.

[142] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3):193–204, 1970.

[143] Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli. Analyzing noise in autoencoders and deep networks. *arXiv preprint arXiv:1406.1831*, 2014.

[144] Matt Post and David Vilar. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324, 2018.

[145] Patrick Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1):81–109, 1996.

[146] Foster J Provost. Iterative weakening: Optimal and near-optimal policies for the selection of search bias. In *National Conference on Artificial Intelligence (AAAI)*, pages 749–755, 1993.

[147] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[148] Sidney I Resnick. *Heavy-tail phenomena: probabilistic and statistical modeling*. Springer Science & Business Media, 2007.

[149] Silvia Richter and Malte Helmert. Preferred operators and deferred evaluation in satisficing planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

[150] Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 975–982, 2008.

[151] Silvia Richter and Matthias Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2010.

[152] Jussi Rintanen. Phase transitions in classical planning: An experimental study. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 2004, pages 101–110, 2004.

[153] Ben Roberts and Dirk P Kroese. Estimating the number of s-t paths in a graph. *Journal of Graph Algorithms and Applications*, 11(1):195–214, 2007.

[154] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 379–389, 2015.

[155] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2nd edition, 2003.

[156] Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. Nematus: a toolkit for neural machine translation. In *EACL: Software Demonstrations*, pages 65–68, 2017.

[157] Rico Sennrich, Barry Haddow, and Alexandra Birch. Edinburgh neural machine translation systems for wmt 16. In *Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 371–376, 2016.

[158] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Association for Computational Linguistics (ACL)*, pages 1715–1725, 2016.

[159] Vitali Sepetnitsky and Ariel Felner. To reopen or not to reopen in the context of Weighted A*. classifications of different trends. In *International Symposium on Combinatorial Search (SoCS)*, 2015.

[160] Vitaly Sepetnitsky, Ariel Felner, and Roni Stern. Repair policies for not reopening nodes in different search settings. In *International Symposium on Combinatorial Search (SoCS)*, pages 81–88, 2016.

[161] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5515–5523, 2018.

[162] Libin Shen, Anoop Sarkar, and Franz Josef Och. Discriminative reranking for machine translation. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 177–184, 2004.

[163] Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum risk training for neural machine translation. In *Association for Computational Linguistics (ACL)*, pages 1683–1692, 2016.

[164] Barbara M Smith and Martin E Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1):155–181, 1996.

[165] Barbara M Smith and Stuart A Grant. Sparse constraint graphs and exceptionally hard problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 646–654, 1995.

[166] Barbara M Smith and Stuart A Grant. Modelling exceptionally hard constraint satisfaction problems. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 182–195. Springer, 1997.

[167] Richard L Smith. Estimating tails of probability distributions. *The annals of Statistics*, pages 1174–1207, 1987.

[168] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[169] Teague Sterling and John J Irwin. Zinc 15–ligand discovery for everyone. *Journal of chemical information and modeling*, 55(11):2324–2337, 2015.

[170] Roni Stern and Levi HS Lelis. What's hot in heuristic search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[171] Meng Sun, Bojian Jiang, Hao Xiong, Zhongjun He, Hua Wu, and Haifeng Wang. Baidu neural machine translation systems for WMT19. In *Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 374–381, 2019.

[172] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, Canada, 2013.

[173] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 3104–3112, 2014.

[174] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. In *International Conference on Learning Representations (ICLR)*, 2019.

[175] Richard Anthony Valenzano, Nathan R Sturtevant, and Jonathan Schaeffer. Worst-case solution quality analysis when not re-expanding nodes in best-first search. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 885–892, 2014.

[176] Richard Anthony Valenzano, Nathan R Sturtevant, Jonathan Schaeffer, and Fan Xie. A comparison of knowledge-based gbfs enhancements and knowledge-free exploration. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 375–379, 2014.

[177] Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[178] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

[179] John Verzani. *Using R for introductory statistics*. CRC Press, 2014.

[180] Tim Vieira. Gumbel-max trick and weighted reservoir sampling, 2014. `https://timvieira.github.io/blog/post/2014/08/01/gumbel-max-trick-and-weighted-reservoir-sampling`.

[181] Ashwin K Vijayakumar, Michael Cogswell, Ramprasaath R Selvaraju, Qing Sun, Stefan Lee, David J Crandall, and Dhruv Batra. Diverse beam search for improved description of complex scenes. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[182] Oriol Vinyals and Quoc V. Le. A neural conversational model. In *ICML Deep Learning Workshop*, 2015.

[183] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):652–663, 2017.

[184] Toby Walsh. Depth-bounded discrepancy search. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1388–1393, 1997.

[185] Toby Walsh. Search in a small world. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1172–1177, 1999.

[186] Chaojun Wang and Rico Sennrich. On exposure bias, hallucination and domain shift in neural machine translation. In *Association for Computational Linguistics (ACL)*, pages 3544–3552, 2020.

[187] Duncan J Watts and Steven H Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393:440–442, 1998.

[188] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

[189] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[190] Scott A Wildman and Gordon M Crippen. Prediction of physicochemical parameters by atomic contributions. *Journal of chemical information and computer sciences*, 39(5):868–873, 1999.

[191] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[192] Ryan Williams, Carla P Gomes, and Bart Selman. Backdoors to typical case complexity. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, volume 3, pages 1173–1178, 2003.

[193] Richard M. Wilson. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory (Series B)*, 16(1):86–96, 1974.

[194] Christopher Wilt and Wheeler Ruml. Effective heuristics for suboptimal best-first search. *Journal of Artificial Intelligence Research*, 57:273–306, 2016.

[195] Christopher Makoto Wilt and Wheeler Ruml. Cost-based heuristic search is sensitive to the ratio of operator costs. In *International Symposium on Combinatorial Search (SoCS)*, pages 172–179, 2011.

[196] Christopher Makoto Wilt and Wheeler Ruml. When does weighted A* fail? In *International Symposium on Combinatorial Search (SoCS)*, pages 137–144, 2012.

[197] Christopher Makoto Wilt and Wheeler Ruml. Speedy versus greedy search. In *International Symposium on Combinatorial Search (SoCS)*, pages 184–192, 2014.

[198] Christopher Makoto Wilt and Wheeler Ruml. Building a heuristic for greedy search. In *International Symposium on Combinatorial Search (SoCS)*, pages 131–139, 2015.

[199] Christopher Makoto Wilt, Jordan Tyler Thayer, and Wheeler Ruml. A comparison of greedy search algorithms. In *International Symposium on Combinatorial Search (SoCS)*, 2010.

[200] Sam Wiseman and Alexander M Rush. Sequence-to-sequence learning as beam-search optimization. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1296–1306, 2016.

[201] Huayue Wu and Peter Van Beek. On universal restart strategies for backtracking search. In *International Conference on Principles and Practice of Constraint Programming*, pages 681–695. Springer, 2007.

[202] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[203] Fan Xie, Martin Müller, and Robert Holte. Adding local exploration to greedy best-first search in satisficing planning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 2388–2394, 2014.

[204] Fan Xie, Martin Müller, and Robert Holte. Understanding and improving local exploration for GBFS. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 244–248, 2015.

[205] Fan Xie, Martin Müller, Robert Holte, and Tatsuya Imai. Type-based exploration with multiple search queues for satisficing planning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 2395–2402, 2014.

[206] Yilin Yang, Liang Huang, and Mingbo Ma. Breaking the beam search curse: A study of (re-) scoring methods and stopping criteria for neural machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3054–3059, 2018.

[207] Lenka Zdeborová and Florent Krzakała. Phase transitions in the coloring of random graphs. *Phys. Rev. E*, 76(3):031131, 2007.

[208] Weixiong Zhang. Complete anytime beam search. In *National Conference on Artificial Intelligence (AAAI)*, pages 425–430, 1998.

[209] Rong Zhou and Eric A Hansen. Beam-stack search: Integrating backtracking with beam search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 90–98, 2005.

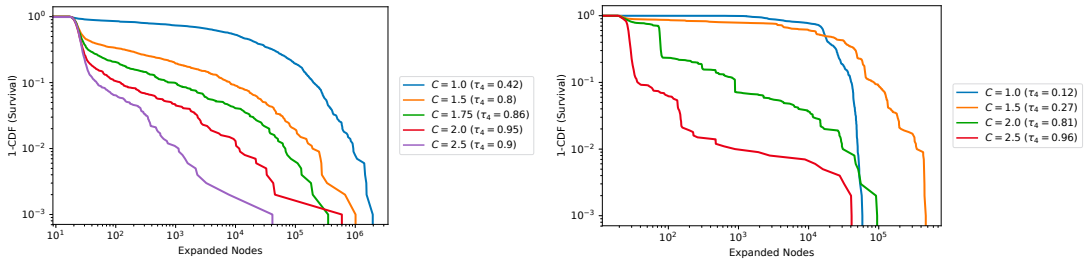
[210] Amit Zohar and Lior Wolf. Automatic program synthesis of long programs with a learned garbage collector. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 2094–2103, 2018.

Appendix A

NoMystery: Results for Standard Evaluation

A.1 Heavy-tailed Behavior

Figure A.1a and Figure A.1b show the runtime distribution for 1000 random problems and 1000 randomized runs (with $p = 0.05$) on a single problem, respectively. We observe a transition from a non-heavy-tailed regime to a fat- and heavy-tailed regime, consistent with the results in Section 4.4.

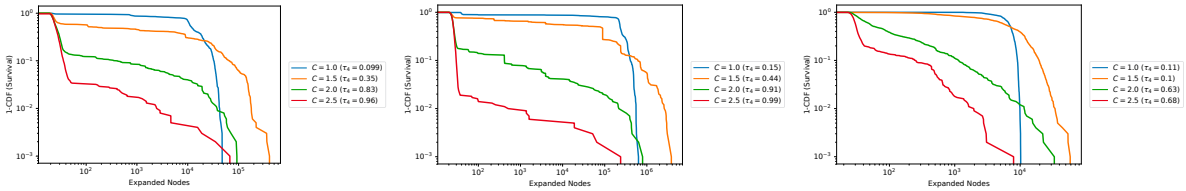


(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure A.1: Results for NoMystery with standard evaluation.

Figure A.2a, Figure A.2b, and Figure A.2c show similar analysis to Figure A.1b using the landmark cut heuristic, the landmark count heuristic, and the CEA heuristic, respectively, all with $p = 0.05$.



(a) Landmark cut heuristic.

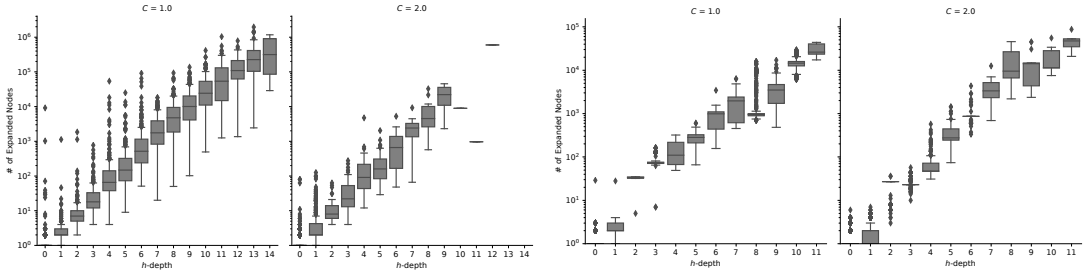
(b) Landmark count heuristic.

(c) CEA heuristic.

Figure A.2: NoMystery: Results for other heuristics with standard evaluation.

A.2 Local Minima Distribution

Figure A.3a shows the distribution of local minima size vs. h-depth in ensemble of 1000 random problems in both the non-heavy-tailed regime ($C = 1.0$) and the heavy-tailed regime ($C = 2.0$). Figure A.3b shows a similar analysis for 1000 randomized runs on a single instance. We can clearly see an exponential correlation, consistent with the results in Section 4.5.

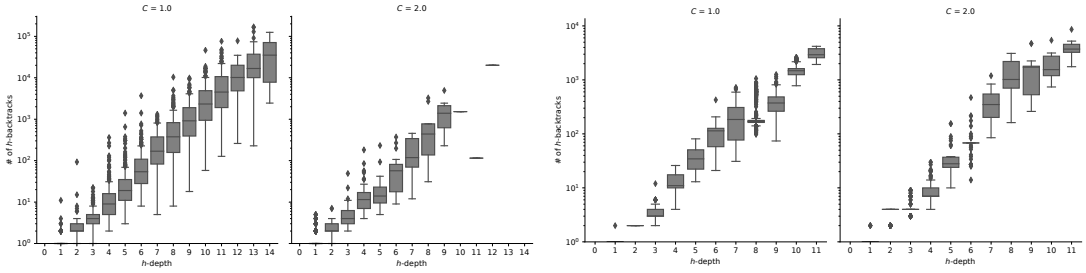


(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure A.3: NoMystery: local minima sizes vs h-depth in standard evaluation.

Figure A.4a and Figure A.4b show the distribution of number of h-backtracks vs. h-depth in an ensemble of 1000 random problems and in 1000 randomized searches on a single problem, respectively, in both the non-heavy-tailed and the heavy-tailed regimes. Again, we observe an exponential correlation, consistent with the results in Section 4.5.



(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure A.4: NoMystery: number of h-backtracks vs h-depth in standard evaluation.

Table A.1 and Table A.2 shows similar analysis to Table 4.1 and Table 4.2, for standard evaluation. We find strong exponential correlation between the h-depth of the deepest local minimum and the search effort (number of h-backtracks), consistent with the results in Section 4.5.

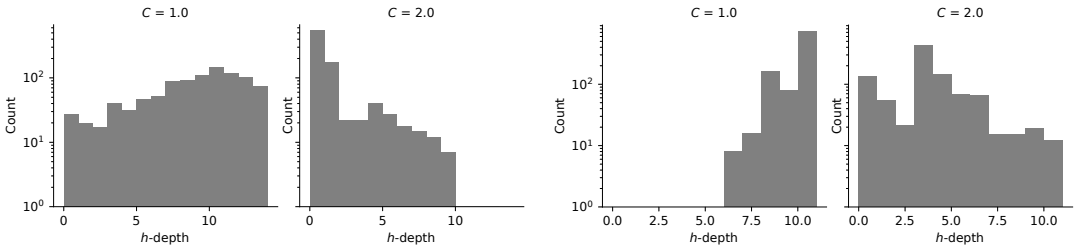
	log(search effort)		*log*(*h*-backtracks)	
Domain	N-HT	HT	N-HT	HT
NoMystery (1000)	0.93	0.93	0.93	0.96
NoMystery (one)	0.89	0.97	0.91	0.96

Table A.1: Pearson correlation coefficient between h-depth and *log*(search effort) (*log*(h-backtracks)) in the non-heavy-tailed (N-HT) and heavy-tailed (HT) regimes for ensembles of random problems and for multiple runs on one problem (standard evaluation).

	log(search effort)		*log*(*h*-backtracks)	
Domain	N-HT	HT	N-HT	HT
NoMystery (1000)	0.97	0.95	0.98	0.94
NoMystery (one)	0.99	0.98	0.99	0.98

Table A.2: Weighted Pearson correlation coefficient between h-depth and *log*(search effort) (*log*(h-backtracks)) in the non-heavy-tailed (N-HT) and heavy-tailed (HT) regimes for ensembles of random problems and for multiple runs on one problem (standard evaluation).

Figure A.5a shows the distribution of of h-depth of the deepest local minimum in the non-heavy-tailed regime ($C = 1.0$) and in the heavy-tailed regime ($c = 2.0$), for 1000 random problems. Figure A.5b shows similar analysis for 1000 randomized problems.



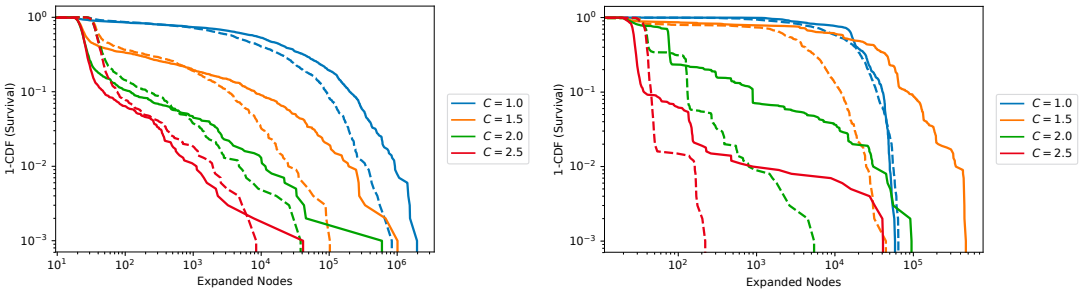
(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure A.5: NoMystery: Distribution of deepest local minima h-depth in standard evaluation.

A.3 Randomization and Heavy-tailed Behavior

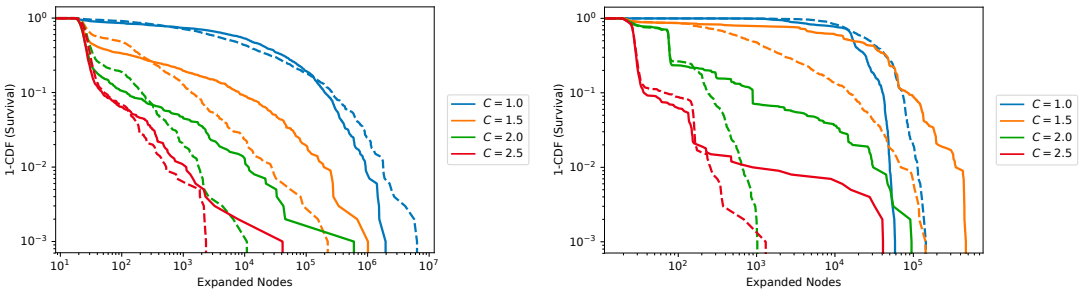
Figure A.6 compares GBFS to Type-GBFS for both 1000 randomized instances and 1000 randomized runs on a single instance ($p = 0.05$). Similarly, Figure A.7 compares GBFS to RR-GBFS. The results are consistent with the analysis in Section 4.6.



(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure A.6: NoMystery: GBFS (solid) vs. Type-GBFS (dashed).



(a) 1000 random instances with h^{FF}.

(b) 1000 randomized runs on a single instance with h^{FF}.

Figure A.7: NoMystery: GBFS (solid) vs. RR-GBFS (dashed).

Appendix B

Results for Constrained Beam Search on WMT'14 En-De

B.1 Results for Discrepancy Gap Constrained Beam Search ($\mathcal{M} = 1.5$)

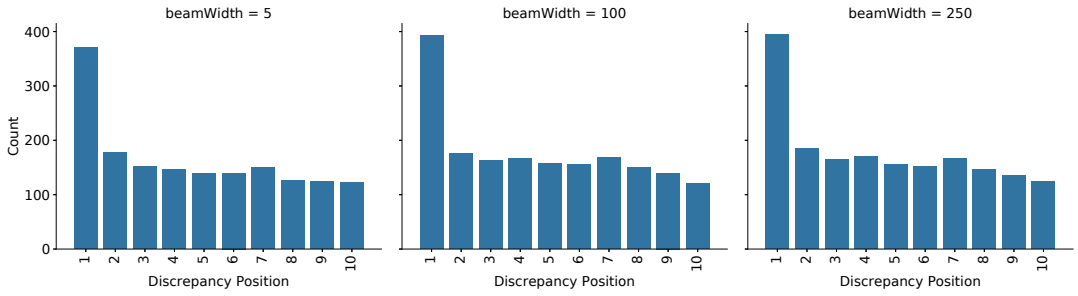


Figure B.1: WMT'14 En-De: Distribution of discrepancy positions ($\mathcal{M} = 1.5$).

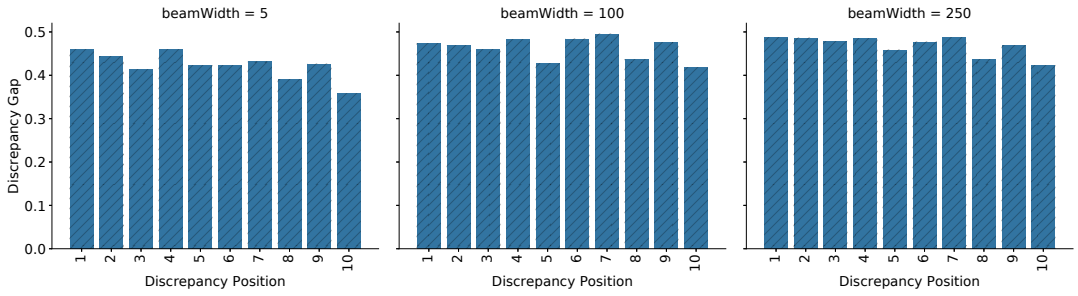


Figure B.2: WMT'14 En-De: Mean discrepancy gap per position ($\mathcal{M} = 1.5$).

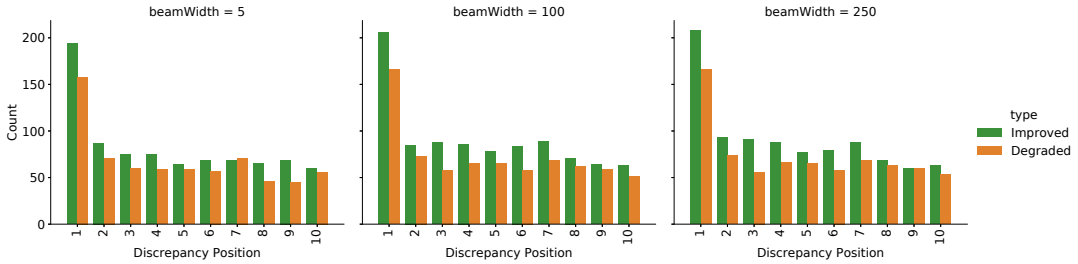


Figure B.3: WMT'14 En-De: Distribution of discrepancy positions ($\mathcal{M} = 1.5$).

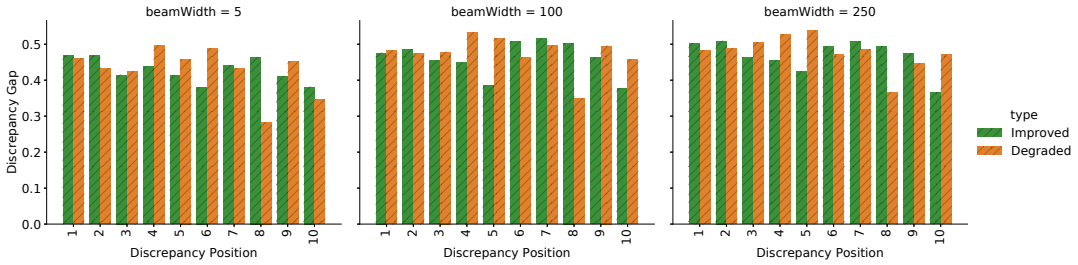


Figure B.4: WMT'14 En-De: Mean discrepancy gap per position ($\mathcal{M} = 1.5$).

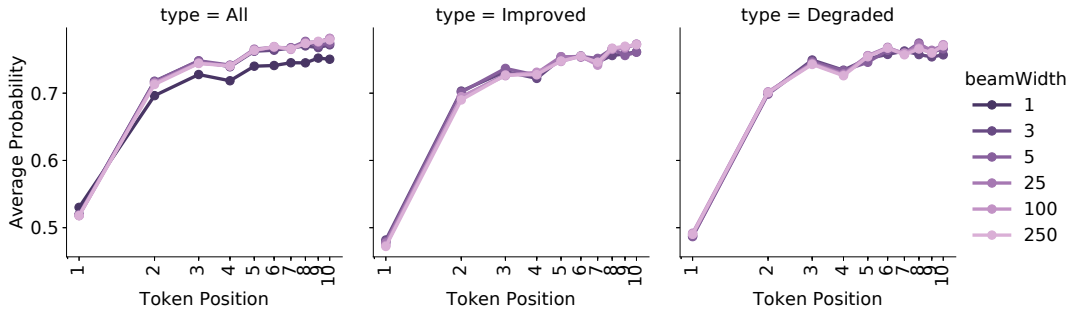


Figure B.5: WMT'14 En-De: Average token probability per position ($\mathcal{M} = 1.5$).

B.2 Results for Rank Constrained Beam Search ($\mathcal{N} = 2$)

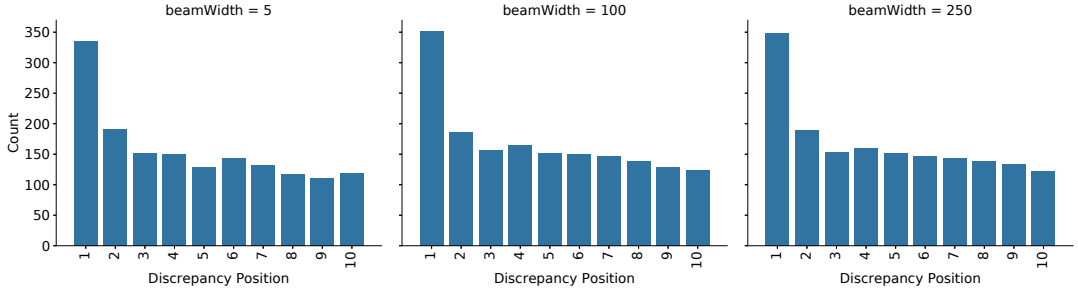


Figure B.6: WMT'14 En-De: Distribution of discrepancy positions ($\mathcal{N} = 2$).

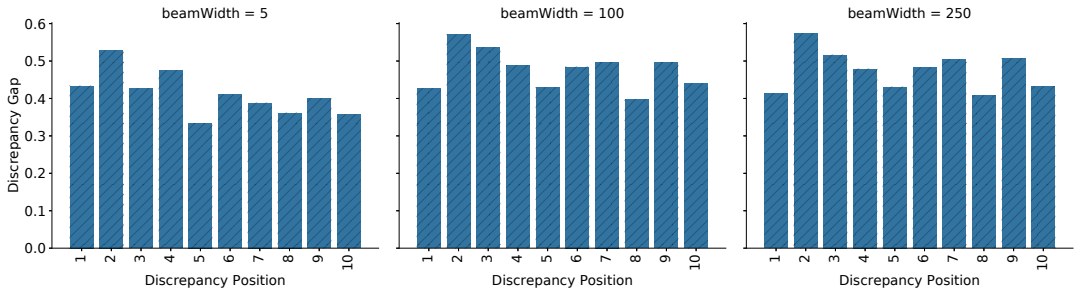


Figure B.7: WMT'14 En-De: Mean discrepancy gap per position ($\mathcal{N} = 2$).

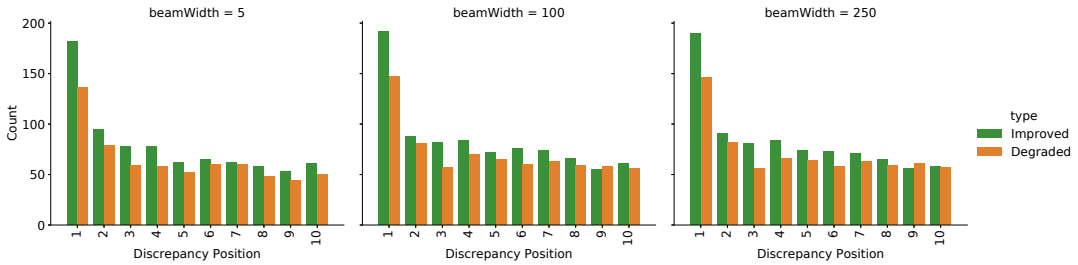


Figure B.8: WMT'14 En-De: Distribution of discrepancy positions ($\mathcal{N} = 2$).

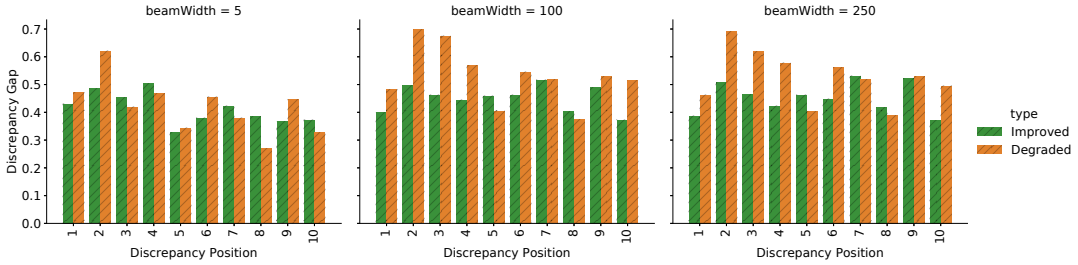


Figure B.9: WMT'14 En-De: Mean discrepancy gap per position ($\mathcal{N} = 2$).

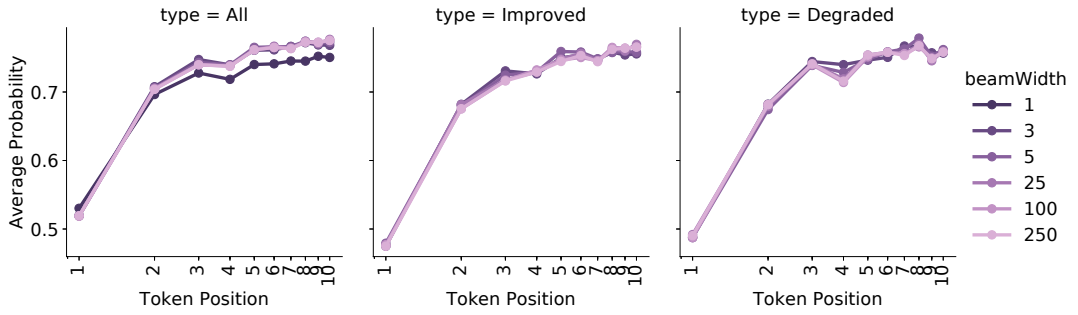


Figure B.10: WMT'14 En-De: Average token probability per position ($\mathcal{N} = 2$).

Appendix C

Input Noise Injection for CSGNet

For visual program synthesis instances, the input to the network is a two-dimensional binary shape image, i.e., a matrix whose values are either zero or one. To add noise to images we flip, with small probability, the bits that are close to the edges of the shape. The process is described below, and demonstrated on an image of a circle shape.

We consider a binary image of a circle presented in Figure C.1. In order to identify a set of pixels that are close to the edges of the shape, we apply a transformation based on two-dimensional randomized kernels. We use a 3×3 kernel where the element in center position is one and the other elements are zero, as in Figure C.2. We then randomize an additional element from the other 8 positions and set it to one as well. Figures C.3 and C.4 demonstrate the pixels detected as edges for two potential randomized kernels.

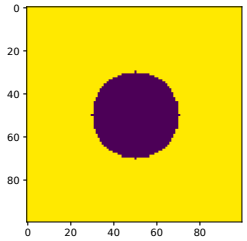
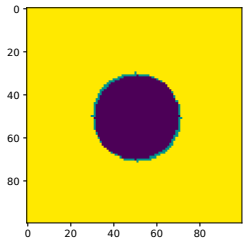


Figure C.1: Binary image of a circle shape.

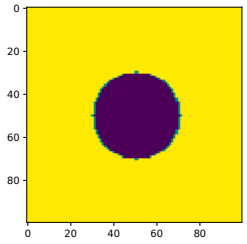
$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Figure C.2: Base 2-dimensional kernel.



$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Figure C.3: Example #1 of edge detection using a randomized kernel.



$$\begin{matrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Figure C.4: Example #2 of edge detection using a randomized kernel.

We consider only the set of pixels whose value is not the highest (the shape itself) or lowest (the background) as the edge pixels and flip the binary value of each pixel with a probability p. In our experiments, we use $p = 0.1$.

Appendix D

Detailed Empirical Results for Goal-Oriented Neural Sequence Decoding

D.1 Detailed Results for TSP

D.1.1 Results for RR-CAB with Input Noise Injection

Table D.1: TSP: Results for RR-CAB with Input Noise Injection

	CAB			RR-CAB (Fixed)			RR-CAB (Geom.)		
ϵ	0.4	0.5	0.6	0.4	0.5	0.6	0.4	0.5	0.6
Threshold									
0	500	500	500	500	500	500	500	500	500
1	265	168	84	500	500	500	344	250	156
2	234	136	69	500	500	500	271	158	73
4	200	114	48	500	500	500	226	103	43
8	177	93	34	255	160	74	179	71	30
16	157	77	31	157	72	21	151	52	17
32	133	62	26	110	34	10	117	41	12
64	123	49	17	65	26	3	96	37	6
128	104	39	14	44	14	2	78	32	4
256	86	29	9	35	7	1	65	21	3
512	74	25	8	25	4	1	55	17	1
1024	64	18	7	21	4	1	45	16	1
2048	55	15	6	18	3	1	39	13	1
4096	46	12	3	13	2	1	32	8	1
8192	38	10	2	11	2	1	25	5	1
16384	31	6	2	7	1	0	24	5	1
32768	26	5	2	6	1	0	19	4	1

D.1.2 Results for RR-CAB with SBS

Table D.2: TSP: Results for RR-CAB with SBS

	CAB			RR-CAB (Fixed)			RR-CAB (Geom.)		
ϵ	0.4	0.5	0.6	0.4	0.5	0.6	0.4	0.5	0.6
Threshold									
0	500	500	500	500	500	500	500	500	500
1	265	168	84	500	500	500	315	206	132
2	234	136	69	500	500	500	220	114	53
4	200	114	48	500	500	500	173	79	30
8	177	93	34	163	79	29	142	55	17
16	157	77	31	113	47	20	115	36	12
32	133	62	26	98	32	8	92	30	8
64	123	49	17	76	25	4	75	26	6
128	104	39	14	64	18	4	65	19	4
256	86	29	9	50	12	3	54	14	4
512	74	25	8	40	9	2	47	12	2
1024	64	18	7	34	8	2	42	7	2
2048	55	15	6	32	6	2	32	5	2
4096	46	12	3	23	5	2	25	5	2
8192	38	10	2	19	5	1	20	5	2
16384	31	6	2	17	5	1	14	3	2
32768	26	5	2	15	5	0	12	3	1

D.2 Detailed Results for CVRP

D.2.1 Results for RR-CAB with Input Noise Injection

Table D.3: CVRP: Results for RR-CAB with Input Noise Injection

	CAB			RR-CAB (Fixed)			RR-CAB (Geom.)		
ϵ	0.4	0.5	0.6	0.4	0.5	0.6	0.4	0.5	0.6
Threshold									
0	500	500	500	500	500	500	500	500	500
1	378	303	211	500	500	500	454	385	293
2	347	259	172	500	500	500	380	287	194
4	316	223	141	500	500	500	330	215	125
8	271	182	102	355	277	193	276	168	77
16	235	141	74	244	139	78	235	127	49
32	197	103	50	167	68	32	189	93	29
64	158	72	35	116	39	13	148	70	19
128	128	54	22	70	27	3	116	49	13
256	102	40	15	46	18	2	83	34	11
512	79	34	11	35	10	1	63	27	6
1024	62	29	9	29	6	1	48	21	2
2048	46	19	5	18	3	1	36	12	1
4096	38	15	3	11	2	1	31	8	1
8192	29	10	2	5	1	1	25	4	1
16384	21	4	1	5	1	1	20	2	1
32768	15	3	1	4	1	0	12	1	1

D.2.2 Results for RR-CAB with SBS

Table D.4: CVRP: Results for RR-CAB with SBS

	CAB			RR-CAB (Fixed)			RR-CAB (Geom.)		
ϵ	0.4	0.5	0.6	0.4	0.5	0.6	0.4	0.5	0.6
Threshold									
0	500	500	500	500	500	500	500	500	500
1	378	303	211	500	500	500	406	326	222
2	347	259	172	500	500	500	322	223	136
4	316	223	141	500	500	500	263	150	82
8	271	182	102	244	160	77	210	115	59
16	235	141	74	196	100	38	174	75	35
32	197	103	50	140	66	24	138	59	20
64	158	72	35	115	49	15	104	38	15
128	128	54	22	91	39	12	81	33	11
256	102	40	15	74	29	9	65	26	9
512	79	34	11	61	25	5	53	20	8
1024	62	29	9	51	20	3	43	14	3
2048	46	19	5	44	16	3	32	8	2
4096	38	15	3	37	13	2	28	5	1
8192	29	10	2	29	10	2	19	1	1
16384	21	4	1	22	7	2	16	1	1
32768	15	3	1	19	3	2	9	1	1

D.3 Detailed Results for Visual Program Synthesis

D.3.1 Results for RR-CAB with Input Noise Injection

Table D.5: Visual Program Synthesis: Results for RR-CAB with Input Noise Injection

	CAB			RR-CAB (Fixed)			RR-CAB (Geom.)		
γ	1.55	1.65	1.75	1.55	1.65	1.75	1.55	1.65	1.75
Threshold									
0	500	500	500	500	500	500	500	500	500
1	377	343	314	500	500	500	452	425	392
2	317	275	238	500	500	500	376	337	279
4	270	220	178	500	500	500	305	257	181
8	228	169	121	334	283	223	225	178	100
16	170	123	79	192	139	69	159	105	59
32	127	89	48	100	63	17	108	58	32
64	88	54	25	49	21	7	77	31	14
128	61	38	19	24	3	2	55	20	7
256	46	21	14	15	0	0	35	9	2
512	29	13	9	8	0	0	19	5	0
1024	18	9	5	6	0	0	13	2	0
2048	10	3	1	2	0	0	4	1	0
4096	10	3	0	1	0	0	4	1	0
8192	6	1	0	0	0	0	1	0	0
16384	6	0	0	0	0	0	0	0	0
32768	1	0	0	0	0	0	0	0	0

D.3.2 Results for RR-CAB with SBS

Table D.6: Visual Program Synthesis: Results for RR-CAB with SBS

	CAB			RR-CAB (Fixed)			RR-CAB (Geom.)		
γ	1.55	1.65	1.75	1.55	1.65	1.75	1.55	1.65	1.75
Threshold									
0	500	500	500	500	500	500	500	500	500
1	377	343	314	500	500	500	438	424	410
2	317	275	238	500	500	500	379	345	323
4	270	220	178	500	500	500	303	260	215
8	228	169	121	296	244	179	225	169	119
16	170	123	79	184	119	80	150	105	67
32	127	89	48	112	55	34	98	63	33
64	88	54	25	71	28	14	68	30	15
128	61	38	19	40	17	10	43	12	6
256	46	21	14	22	8	0	21	7	3
512	29	13	9	13	6	0	10	4	0
1024	18	9	5	4	1	0	6	1	0
2048	10	3	1	1	0	0	3	0	0
4096	10	3	0	1	0	0	1	0	0
8192	6	1	0	0	0	0	1	0	0
16384	6	0	0	0	0	0	0	0	0
32768	1	0	0	0	0	0	0	0	0

D.4 Detailed Results for Conditional Molecule Generation

D.4.1 Results for RR-CAB with Input Noise Injection

Table D.7: Conditional Molecule Generation: Results for RR-CAB with Input Noise Injection

	CAB			RR-CAB (Fixed)			RR-CAB (Geom.)		
ρ	0.01	0.05	0.07	0.01	0.05	0.07	0.01	0.05	0.07
Threshold									
0	500	500	500	500	500	500	500	500	500
1	449	271	215	500	500	500	452	281	210
2	401	196	145	500	500	500	384	174	99
4	341	151	109	500	500	500	301	113	55
8	283	118	80	318	138	88	208	63	29
16	227	87	57	171	53	32	141	45	25
32	169	65	41	100	28	16	100	32	17
64	138	52	26	68	22	11	79	24	11
128	121	41	20	49	16	7	68	14	7
256	104	30	17	34	7	4	56	12	7
512	88	23	11	24	5	4	46	11	4
1024	74	19	9	19	5	4	32	7	4
2048	62	17	4	13	3	3	27	5	4
4096	48	11	4	10	2	3	22	3	4
8192	37	6	4	8	2	2	13	3	3
16384	29	4	3	6	2	2	12	3	2
32768	22	3	2	6	2	1	10	3	1

D.4.2 Results for RR-CAB with SBS

Table D.8: Conditional Molecule Generation: Results for RR-CAB with SBS

	CAB			RR-CAB (Fixed)			RR-CAB (Geom.)		
ρ	0.01	0.05	0.07	0.01	0.05	0.07	0.01	0.05	0.07
Threshold									
0	500	500	500	500	500	500	500	500	500
1	449	271	215	500	500	500	464	355	293
2	401	196	145	500	500	500	401	206	146
4	341	151	109	500	500	500	314	118	80
8	283	118	80	297	110	73	228	73	50
16	227	87	57	216	76	44	165	57	34
32	169	65	41	156	62	36	121	47	24
64	138	52	26	117	50	25	102	37	15
128	121	41	20	98	35	14	85	25	13
256	104	30	17	84	25	8	71	20	11
512	88	23	11	66	19	6	60	15	7
1024	74	19	9	58	15	5	51	11	5
2048	62	17	4	52	11	5	42	9	4
4096	48	11	4	48	9	4	28	5	4
8192	37	6	4	36	7	4	22	4	3
16384	29	4	3	30	5	4	18	2	1
32768	22	3	2	21	3	4	13	2	0