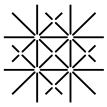


University of Basel

Bounded Suboptimal Search for Classical Planning

Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence Group

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Jendrik Seipp

Caroline Steiblin
carolinesteiblin@gmail.com
2015-615-099

09.10.2020

Acknowledgments

Over the last three months of research and writing for this thesis, I received a great deal of support and assistance. Firstly, I would like to thank my supervisor, Dr. Jendrik Seipp, for his expertise and guidance in our weekly meetings, and for the countless hours helping me navigate through my code and the Fast Downward planner. Thank you as well to Prof. Dr. Malte Helmert for the opportunity to conduct research in the Artificial Intelligence Group at the University of Basel, and for inviting the 2020 Foundations of Artificial Intelligence course to join the 13th Annual Symposium on Combinatorial Search (SoCS). Finally, a warm thank you to my parents and partner for providing me with levity and encouragement during this time.

Abstract

Suboptimal search algorithms can offer attractive benefits compared to optimal search, namely increased coverage of larger search problems and quicker search times. Improving on such algorithms, such as reducing costs further towards optimal solutions and reducing the number of node expansions, is therefore a compelling area for further research. This paper explores the utility and scalability of recently developed priority functions, XDP, XUP, and PWXDP, and the Improved Optimistic Search algorithm, compared to Weighted A*, in the Fast Downward planner. Analyses focus on the cost, total time, coverage, and node expansion parameters, with experimental evidence suggesting preferable performance if strict optimality is not desired. The implementation of priority functions in eager best-first search showed marked improvements compared to A* search on coverage, total time, and number of expansions, without significant cost penalties. Following previous suboptimal search research, experimental evidence even seems to indicate that these cost penalties do not reach the designated bound, even in larger search spaces.

Table of Contents

1 Introduction

Optimality is theoretically significant when designing search algorithms, but in practice, for solving large and complex search spaces, as found in the real world, optimality is not always feasible (Chen et al. [2]). Take two environments, the first static, deterministic, discrete, and fully observable like that of the Romanian route planning problem[1]. The second environment is dynamic, deterministic, discrete, and partially observable, like that of a real-time strategy video game. An optimal search algorithm like A* (Hart et al. [5]) can efficiently and effectively solve the route planning problem due to the predictable and easily-defined properties of the environment (Helmert [7]). The same cannot be said of the second environment, as A* is limited by its large computational needs and requires modifications to handle the more difficult properties at a speed that is enjoyable to the end user (Churchill [3]).

Suboptimal search algorithms can solve for this, as the restrictive optimality guarantee no longer holds, and computational time can be significantly reduced when even a small amount of suboptimality is allowed (Chen et al. [2]). The most common suboptimal search algorithm is Weighted A* (WA*) (Pohl [8]), which finds solutions that are at most w times larger than the optimal solution cost, with w being a predefined weight. Solutions are therefore considered w-suboptimal. Recent research in suboptimal search has aimed to develop easily implementable algorithms and supporting evaluation functions that can find bounded suboptimal solutions and do not require space-intensive node re-expansions (Chen and Sturtevant [1]). Three such evaluation functions aiming to outperform WA* will be examined further in this paper: the convex downward parabola (XDP), convex upward parabola (XUP) (Chen and Sturtevant [1]), and XX convex downward parabola (PWXDP). Additionally, the Improved Optimistic Search (IOS) algorithm (Chen et al. [2]), using a focal list for search and an open list to prove sub-optimality, is compatible with the aforementioned evaluation functions and has shown experimental performance improvement compared to WA*. In this paper, a modified version of the IOS algorithm will be analyzed. With such improved algorithms, the corresponding solution cost difference, among other properties,

[1] Route planning in Romania is a simple state space example that includes the paths of major cities in Romania to the capital, Bucharest (goal state) (Helmert [7]).

between suboptimal and optimal search becomes smaller, without significantly limiting the coverage and speed of suboptimal search. This paper aims to expand on previous research and test the IOS algorithm, with its evaluation function variants, on a different range of problems, to see if the experimental search improvements can be reproduced.
This paper will first introduce the necessary background on the planning system used, as well as the functionality of the algorithms included. The implementation of these algorithms in the Fast Downward planner will follow, the results of which will then be analyzed and evaluated in context. Finally, future work in this area will be proposed, to provide further experimental support of the utility and scalability of suboptimal search algorithms.

2
Background

This chapter will provide an overview of the terminology, definitions, and concepts used throughout this paper. The background provided here is mainly standard artificial intelligence introductory knowledge, although references to more advanced papers are given where appropriate.

2.1 Environments and State Spaces

An artificial intelligence problem is constructed through a performance measure, an agent model, and an environment. Performance measures relevant to this paper include properties like cost (optimal or w-suboptimal), coverage, and time, while an agent model represents the actions of the agent (mapping observations to actions and computing an evaluation function that guides the agent around a certain search space). The environment of the search space describes the space in which the agent exists, and allows the following classifications.

Definition 2.1.1 (Properties of Environments)**.** (Helmert [7]) An environment can be categorized as follows:

- If the state of the environment remains the same while the agent is determining its next action, it is called *static*. Otherwise, it is called *dynamic*.
- An environment is called *deterministic*, if the next state of the environment is fully determined by the current state and the agent's next action. Otherwise, it is called *stochastic*.
- An environment is called *discrete* if its state is given by finitely discrete (predefined) parameters. If its given parameters take infinitely many values, it is called *continuous*.
- If an agent's observations can completely determine the state of the environment at any given time, the environment is called fully observable. Otherwise, it is called partially observable.

These parameters help define the state space for a search problem.

A state space is a 6-tuple describing the space in which an artificial intelligence agent operates, given by the following definition.

Definition 2.1.2 (State Space)**.** (Helmert [7]) A *state space* or *transition system* is defined as $\mathcal{S} = \langle S, A, cost, T, s_0, S_* \rangle$ with

- S, the finite set of states
- A, the finite set of actions
- $cost$, $A \to \mathbb{R}_0^+$, the action costs
- $T \subseteq S \times A \times S$, the transition relation
- $s_0 \in S$, the initial state
- $S_* \subseteq S$, the set of goal state(s)

Only deterministic state spaces are considered in this paper, which means that in $\mathcal{S}$, the transition $\langle s, a, s' \rangle \in T$, written as $s \xrightarrow{a} s'$ only has $s \xrightarrow{a} s_1$ and $s \xrightarrow{a} s_2$ if $s_1 = s_2$. This condition greatly simplifies the search process. State spaces are defined for search problems in planners, where the problem is described in such a formalism that allows for algorithms to be implemented and compute solutions.

2.2 Classical Planning

Plans, the sequence of actions from the initial state to reach a goal state, offer a practical application and testing for the theoretical optimal and suboptimal search approaches discussed above. Classical plans aim to solve classical state space search problems that are static, deterministic, discrete, and fully observable (Helmert [7]). Given a state space description in a planning formalism, a planner will either find a plan for the state space (a solution) or proof that no such plan exists with the given algorithm(s) (Helmert [7]). Optimal planning guarantees optimal solutions, while suboptimal planning will find all plans in a given bound of the optimal cost (if a bound is provided), even if the solution is suboptimal.
Fast Downward is a classical planning system developed in 2003 based on heuristic search (Helmert [6]). The system solves deterministic planning problems encoded in the Planning Domain Definition Language (PDDL) formalism by first converting the problem into the Simplified Action Structures (SAS+) formalism before solving. This allows for implicit constraints of a propositional planning task to become explicit, and the use of the causal graph heuristic. Improved performance on solving planning problems has been shown experimentally, as the planner reduces the number of states for which a heuristic must be computed (Helmert [6]). All experiments in this paper were implemented in the Fast Downward planner, and computed on sciCORE, the scientific computing center at the University of Basel, offering high-performance computing and data analysis capabilities.

2.3 Heuristics

Heuristics are the basis for informed search algorithms, where aspects of the problem itself are used to find the solution. In a state space $\mathcal{S}$, an arbitrary heuristic function, $h : S \leftarrow \mathbb{R}_0^+ \cup \{\infty\}$ maps each state to a non-negative number (or ∞), to guide the algorithm through the state space more efficiently. Heuristics act as goal distance estimators, to compute how far the agent is from the goal state. An example for a heuristic for route planning in Romania could be the straight line distance from a state to Bucharest (the goal state). Heuristics can also have the following properties:

Definition 2.3.1 (Properties of Heuristics)**.** (Helmert [7]) For state space $\mathcal{S}$ with states S, a heuristic h for $\mathcal{S}$ can be

- *safe*, if the optimal solution cost $h^*(s) = \infty$ for all $s \in S$ with $h(s) = \infty$ (if the heuristic claims there is no solution, there is none)
- *goal-aware*, if $h(s) = 0$ for all goal states $s \in S_*$
- *admissible*, if $h(s) \leq h^*(s)$ for all states $s \in S$ (the heuristic is never larger than the optimal solution)
- *consistent*, if $h(s) \leq cost(a) + h(s`)$ for all transitions $s \xrightarrow{a} s`$ where $s, s' \in S$ (verifying the triangle equality, catching an inaccuracy if the path towards the goal state decreases more than the action taken)

In this paper, the safe, admissible, and consistent (and goal-aware) additive Counterexample-guided Abstraction Refinement (CEGAR) heuristic (Seipp and Helmert [9]) was used as the tie-breaker.

2.4 Optimal Search Algorithms

Using the state space and planning formalisms described above, the next step to solve search problems is developing the search algorithm. Search algorithms operate with open and closed lists. The open list organizes the leaves of a search tree, determining and removing the next node to expand, and inserting a new node that has potential for expansion (Helmert [7]). The closed list stores the expanded states to avoid duplicate expansions (reopenings) of a certain state.

2.4.1 A*

Best-first search (BFS) is a heuristic search algorithm that evaluates search nodes with an evaluation function f and always expands a node n with minimal $f(n)$ value (Helmert [7]). BFS uses a heuristic to determine node expansion, normally expanding promising nodes with low heuristic values $h(n)$ over those with higher $h(n)$ values. A sketch of BFS can be found in Algorithm 1.

A* (Hart et al. [5]) is a variant of best-first search that aims to find an optimal solution through computing $f(n) = g(n) + h(n)$, with $f(n)$ as the evaluation function, $g(n)$ as the path cost, and $h(n)$ as the heuristic cost, and expanding the nodes with minimal $f(n)$

Algorithm 1: Generic Best-First Search (Chen and Sturtevant [1])

```
initialization: (start, goal)
Push(start, OPEN)
while OPEN not empty do
    Remove best from OPEN
    if best == goal then
        return success
    end
    Move best to CLOSED
    foreach successor s_i of best do
        if s_i on OPEN then
            Update cost of s_i on OPEN if shorter
        else if s_i not on CLOSED then
            Add s_i to OPEN
        end
    end
end
return failure
```

values. A* search is complete with safe heuristics, so it is then guaranteed to find a solution if one exists and terminate if no solution exists. If reopening previously closed states is allowed, A* is optimal with admissible heuristics. Without reopening, A* is only optimal with admissible and consistent heuristics (Helmert [7]). In this paper, reopening previously closed nodes is allowed, as there would be significant limitations in algorithm coverage (algorithm finding a solution) if reopening was not allowed. A* is used in this context as the optimal search algorithm, as a baseline for comparison with suboptimal evaluation function implementations.

2.5 Suboptimal Search Algorithms

Suboptimal search algorithms offer an alternative to optimal search with the introduction of a weight w that ensures that all found solutions are at most w times as costly as the optimal solution cost (in this case the minimal cost). These solutions are called w-suboptimal. This can be useful to boost optimal search algorithms, as the relaxed requirement for optimality means solutions can be found in less time. If w is well-chosen (even a weight of 1.1 can lead to faster results (Helmert [7])), the cost penalty of suboptimality can be managed. There are two groups of suboptimal search algorithms - ones based on BFS (for example WA*) and others based on focal search (Optimistic Search).

2.5.1 Focal Search

Focal search (Cohen et al. [4]) is a variant of bounded suboptimal search (BSS), using both an *open list* and *focal list* in parallel to separately find a solution and guarantee its w-suboptimality. The open list is like that of A*, it contains all states in increasing order of $f(n)$. The focal list contains states from the open list, although only those whose $f(n)$ values are smaller than the $w \cdot f_{min}$, where w is the weight, $w_f = 2w-1$ the weight bound, and f_{min} is the smallest $f(n)$ value on the open list. This mechanism leads to quick solution finding,

so that solutions can be tested for w-suboptimality while the focal list continues searching, potentially leading to refinement and better solutions. Evaluation functions support sorting the focal list, to optimize this process even further.

2.5.2 Weighted A*

Weighted A* (WA*) (Pohl [8]) is a suboptimal variant of A*, with an evaluation function $f(n) = g(n) + w \cdot h(n)$ with weight $w \geq 1$, guaranteeing a solution at most w times as expensive compared to A* when reopening is used. WA* carries the same properties as A*, so will provide w-suboptimal solutions if the heuristics are admissible (with reopening) or admissible and consistent (without reopening). WA* will be used in this paper as the baseline for comparing the functionality of recently developed suboptimal evaluation functions and algorithms.

2.5.3 Optimistic search

Optimistic search (OS) (Thayer and Ruml [10]) is a focal search variation of WA* that searches with $f(n) = g(n)+(2w-1)\cdot h(n)$ and can find solutions that are even w-suboptimal. The algorithm takes into account that WA* often finds solutions that are less costly (and therefore better) than w-suboptimal, given weight w. Optimistic search uses the weight bound of $2w-1$ on a focal list, which improves total search time, and runs checks in parallel through A* search that a solution found is w-suboptimal. Optimistic search will reopen states if a shorter path is found. The base algorithm is sketched in Algorithm 2.

Algorithm 2: Optimistic Search (Thayer and Ruml [10])

initialization: *initial, bound*
$OPEN_f \leftarrow \{initial\}$
$OPEN_{\hat{f}} \leftarrow \{initial\}$
$incumbent \leftarrow \infty$
repeat until $bound \cdot f(\text{first on } OPEN_f) \geq f(incumbent)$
  if $\hat{f}(\text{first on } OPEN_{\hat{f}}) < \hat{f}(incumbent)$ **then**
    $n \leftarrow$ remove first on $OPEN_{\hat{f}}$
    remove n from $OPEN_f$
  else
    $n \leftarrow$ remove first on $OPEN_f$
    remove n from $OPEN_{\hat{f}}$
  end
  add n to $CLOSED$
  if n is a goal **then**
    $incumbent \leftarrow n$
  else
    foreach child c *of* n **do**
      if c is duplicated in $OPEN_f$ **then**
        if c is better than the duplicate **then**
          replace copies in $OPEN_f$ and $OPEN_{\hat{f}}$
        end
      else if c is duplicated in $CLOSED$ **then**
        if c is better than the duplicate **then**
          add c to $OPEN_f$ and $OPEN_{\hat{f}}$
        end
      else
        add c to $OPEN_f$ and $OPEN_{\hat{f}}$
      end
    end
  end
end

3

Improved Optimistic Search

Improved Optimistic Search (IOS) and its associated non-linear evaluation functions will be introduced in this chapter, based on the ongoing research in improving suboptimal search algorithms. The IOS algorithm is covered in detail, with an accompanying pseudo code scheme.

3.1 Evaluation Functions

Referred to also as *priority functions*, evaluation functions guide informed search algorithms to a solution in various ways. Current research has explored how to improve solution quality while still avoiding node re-openings through evaluation functions in the form of $f(n) = \Phi(h(n), g(n))$, where $\Phi : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Four functions are of interest to this paper, one of which is the well-known WA*, introduced in the Background section. These functions are either linear or nonlinear, each with favorable performance depending heavily on the properties of the search problem and heuristic.

3.1.1 Linear Evaluation Functions

Linear evaluation functions keep the maximum level of suboptimality constant at the weight and are easy to implement. Taking the function form from above, WA* has the following construction, with w as the weight, h as the heuristic function $h(n)$ and g as the path cost $g(n)$:

$$\Phi_{WA*}(h, g) = \frac{1}{w} \cdot g + h. \tag{3.1}$$

Through this formulation, it has been easy to demonstrate that WA* maintains w-suboptimality without needing to reopen states (Chen and Sturtevant [1]).

3.1.2 Non-Linear Evaluation Functions

Nonlinear evaluation functions allow the tolerance for suboptimality to vary during search. Whereas linear functions always provide the same level of suboptimality throughout the entire search, nonlinear functions like convex parabolas can strongly limit the degree of suboptimality at the beginning of the search. This ensures that suboptimality is focused

on the critical parts of the search process, with a little more tolerance available after the beginning of the path. Experimentally, this would imply that the non-linear functions would perform better with regards to degree of suboptimality than the linear functions. The first nonlinear function is the Convex Downward Parabola (XDP) (Chen and Sturtevant [1]) which can be calculated by the following, where w is the weight, h is the heuristic function $h(n)$ and g is the path cost $g(n)$:

$$\Phi_{XDP}(h,g) = \frac{1}{2w}(g + (2w-1)h + \sqrt{}((g+h)^2 + 4wgh) \tag{3.2}$$

Through its construction, paths with low g (path cost) should be near optimal (Chen and Sturtevant [1]).

The second nonlinear function is the Convex Upward Parabola (XUP) (Chen and Sturtevant [1]), whose parabola is constructed such that the path found near the goal will be near-optimal. The formula for XUP is the following, using the same weight, heuristic and path cost parameters.

$$\Phi_{XUP}(h,g) = \frac{1}{2w}(g + h + \sqrt{}((g+h)^2 + 4w(w-1)h^2) \tag{3.3}$$

The third nonlinear function of interest is a piece-wise linear function, the Piece-Wise Convex Downward Parabola (PWXDP), which is constructed through the concatenation of two linear functions. Since this evaluation function has not yet been published, only $f(n)$ formulations are available at this time.

$$f(n) = \begin{cases} g(n) + h(n) & \text{if } h(n) > g(n) \\ \frac{g(n)+(2w-1)h(n)}{w} & \text{otherwise} \end{cases} \tag{3.4}$$

3.2 Improved Optimistic Search

The recently developed Improved Optimistic Search (IOS) algorithm (Chen et al. [2]) adapts and simplifies, to some extent, the OS algorithm. IOS is deterministic and also runs two searches - one designed to find a quick solution through expansions on the focal list and the other to verify w-suboptimality through the open list. The open list search runs an A* search with $f(n) = g(n)+h(n)$, whereas the focal list uses an *evaluation function*, $f'(n)$. The theoretical pseudo code for IOS can be found in Algorithm 3, although impactful changes are also found in the dual termination conditions and the general parameterization. On the latter point, IOS is adapted to run with both linear and non-linear evaluation functions, allowing for greater testing capabilities. The termination conditions expand on those already found in OS ($c(I) \leq wf_{min}$) to prove the w-suboptimality of a solution found. To clarify terminology, $c(I)$ is the cost of I, the incumbent plan, w the weight, and f_{min} the minimum $f(n)$ value of a state in the open list. Through the rearranged $\Phi(h,g)$ functions, the $f(n)$ cost is not an estimate of the experimental solution cost, but of the optimal solution cost, so can be taken like f_{min} on the open list (Chen et al. [2]). This implies that a termination condition involving $\Phi(h,g)$, or $f'(n)$, can be used directly on the focal list. This termination condition is presented as $c(I) \leq wf'_{max}$, with f'_{max} the maximum $f'(n)$ (evaluation function) value of a state on the focal list. The dual termination conditions should result in solutions being even closer to optimal cost.

To elaborate on the algorithm's functionality, the IOS algorithm initializes by defining a clear start state and goal description, as well as a weight tolerance for the w-suboptimality of the solution. The $OPEN$ and $FOCAL$ lists are initialized and the start state is added to both lists. The initial incumbent plan (I) is not defined, so its cost, $c(I)$ is infinite. While $c(I)$ is not w-optimal, meaning $c(I)$ fails both termination conditions[2], and if the estimated path length of the best state on $FOCAL$ is less than $c(I)$ [3], the best state on $FOCAL$ is expanded. This will generate successors that will be added to $FOCAL$. If the best state is a goal state, then the path to this best state will be added to the incumbent plan. If the estimated path length of the best state on $FOCAL$ is greater or equal to $c(I)$ (indicating that an incumbent plan has been found on $FOCAL$), the w-suboptimality verification process takes place on the open list search, where the best state is then expanded on $OPEN$. For each child s of the best state, applicable operators (path costs) are computed [4]. If the path cost of s, $g(s)$ in $OPEN$ is less than the path cost of the state in $FOCAL$, three actions can take place: the path cost, g value, of state s on $FOCAL$ can be updated; state s can be re-opened on $FOCAL$; or, if state s is in the incumbent list, I, update the cost, $c(I)$ of the incumbent path. And if no path to any goal state is found, return $failure$. This can be seen in the pseudo code scheme found in Algorithm 2. Though still theoretical at this point, the next section of this paper will explore the implementation of this IOS algorithm into the Fast Downward planner.

[2] while $c(I) > weight * f_{min}$ && $c(I) > weight * f'_{max}$
[3] $c(I) \rightarrow f'_{min}$ of nodes in FOCAL
[4] $g(s)$ in OPEN = g(best state) + cost(operator)

Algorithm 3: Improved Optimistic Search (Chen et al. [2])

```
initialization: (start, goal, w)
push(start, OPEN)
push(start, FOCAL)
incumbent ← null [c(incumbent) = ∞]
while c(incumbent) not w-optimal do
    if estimated path length of best on FOCAL < c(incumbent) then
        expand best from FOCAL
        if best == goal then
            incumbent = path(best) else
                expand best from OPEN
            end
            if child s has shorter path to s on FOCAL then
                // Choose one of the following policies:
                (a) Update cost of s on FOCAL // Update
                (b) Re-open s on FOCAL // Re-open
                if s ∈ incumbent then
                    (c) update cost of incumbent // Solution update
                end
            end
    end
    return failure
end
```

4

Implementation

This chapter covers the implementation of IOS and the evaluation functions in the Fast Downward planner. The source code and corresponding experiments can be found in a personal fork and branch of the Downward repository here. The IOS algorithm discussed in Chapter 3 was modified, with changes seen in a pseudo code scheme.

4.1 Implementation

To explore and test the functionality of the evaluation functions and IOS algorithm described in the last chapter, a multi-stage implementation scheme was proposed:

- Implement the four relevant evaluation functions, WA*, XDP, XUP, and PWXDP, in eager BFS, in the Fast Downward planner (Helmert [6]), and evaluate results among the functions themselves, as well as an A* control search, on a standardized set of benchmarks.
- Implement the IOS algorithm (or a modified version thereof) with each evaluation function, to test both the difference between IOS and eager BFS and the evaluation function compatibility with IOS on a standardized set of benchmarks.

Three experiments were created to carry out the implementation (one for each implementation type and a final comparison experiment) to run over the `optimal-strips` benchmark suite, the standard for Fast Downward experiments. Experiments were written in a Python environment, following the protocol of Lab and Downward Lab experiments, although the evaluation functions and algorithms themselves were programmed in C++. The weight used in all experiments was held at $w = 2$ to maximize efficiency and coverage, although this is higher than weights typically used in practice. The weight bound was set at the standard $w_f = 2w - 1$ ($w_f = 3$). As well, CEGAR was used in all experiments as the heuristic function, as it is admissible, safe, and consistent.

4.1.1 Evaluation Function and Algorithm Implementation

There were no issues directly applying the theoretical evaluation functions to the Fast Downward planner, although the PWXDP function was computed in its $f(n)$ formulation, versus the $f'(n)$ formulation of the other three. This would also limit the implementation of the IOS algorithm, especially with regards to the second termination condition, $c(I) \leq wf'_{max}$, which is dependent on a $f'(n)$ function. This was not the only limitation to the IOS algorithm implementation though. The two searches - with the open and focal lists - were modified to be run sequentially versus in parallel. The capability to run two search spaces with access to one state registry (required as each search space shares the same states for each search problem) could be achieved through hard-coding the focal search space (with the weight bound $w_f = 2w - 1$), focal and open lists (these were best-first open lists in Fast Downward). A scheme of the modified IOS algorithm used can be found in Algorithm 4. Both individual searches reopen nodes in their own search space, but no information is shared between the searches. The second part of the IOS algorithm, where the algorithm switches between both search spaces, and path costs are updated across search spaces, was not able to be implemented at this point.

Algorithm 4: Improved Optimistic Search (modified)

initialization: *(start, goal, w)*
push(*start*, $OPEN$)
push(*start*, $FOCAL$)
incumbent $\leftarrow$ *null* $[c(I) = \infty]$
while $c(I)$ *not w-optimal* **do**
  if *incumbent* is *null* **then**
    expand *best* from $FOCAL$
    if *best* is *goal* state **then**
      incumbent = path(*best*)
    end
  else
    expand *best* from $OPEN$
    if *best* is *goal* state **then**
      incumbent = path(*best*)
      break
    end
  end
end
if *incumbent* is *null* **then**
  return failure
else
  return *incumbent*
end

5
Results

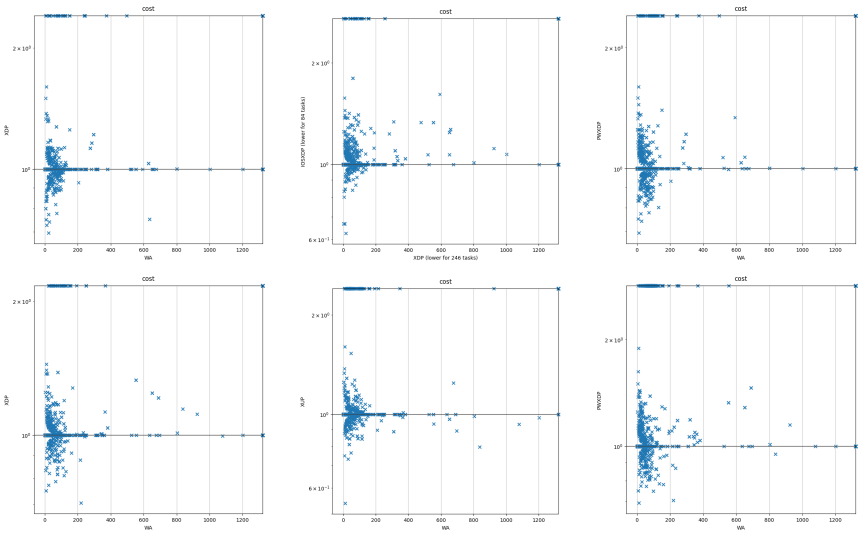
This Chapter will cover the results of the three experiments conducted, organized by attribute. Firstly, a relative cost comparison of the linear and nonlinear evaluation functions in eager best-first search (BFS) and IOS will be provided. An analysis of the number of expansions and relative time of the linear versus nonlinear evaluation functions in both algorithms will follow. Finally, the first experiments will be compared, specifically examining the cost penalties of suboptimal search and coverage of the evaluation functions.

5.1 Relative Cost, linear and non-linear functions

Figure 5.1 presents the relative costs of the nonlinear evaluation functions XDP, XUP, and PWXDP, compared to the linear WA* evaluation function, in both the eager BFS and IOS implementations. From previous research and theoretical concepts explained in Chapter 3, a nonlinear evaluation function should have some cost advantage over a linear evaluation function. As seen in Figure 5.1 though, there does not seem to be any marked cost benefit for nonlinear evaluation functions. The XUP function in the IOS implementation is the only nonlinear function showing a faster relative time compared to WA*, although the benefit is only a factor of 1.01x faster than the WA* implementation in the same algorithm.

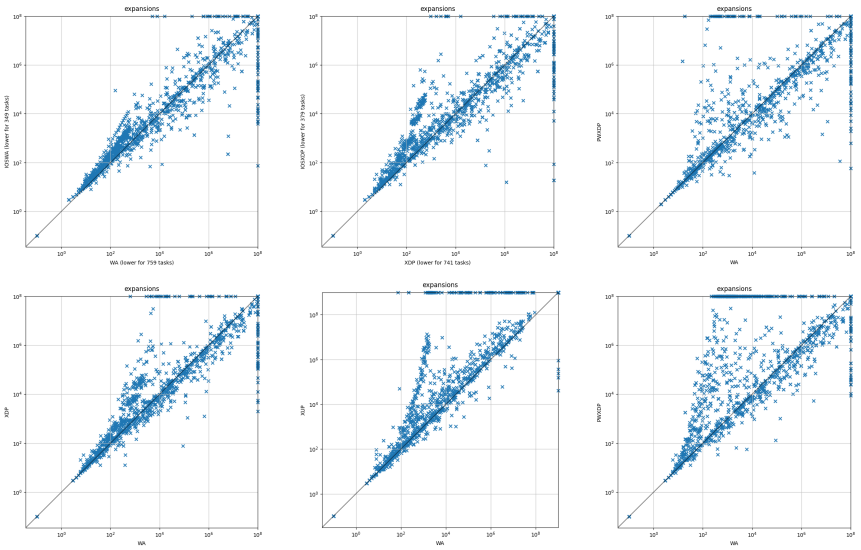
5.2 Number of Expansions and Relative Time, linear and non-linear functions

Figure 5.2 presents the number of expansions of the nonlinear evaluation functions XDP, XUP, and PWXDP, compared to the linear WA* evaluation function, in both the eager BFS and IOS implementations. A nonlinear evaluation function should have, according to previous experiments, fewer expansions (and reopenings) versus a linear evaluation function. As seen in Figure 5.2 though, the opposite seems to be the case. In both algorithms, WA* has fewer expansions, although significantly so with regards to the XUP and PWXDP functions in the IOS algorithm. The PWXDP function seems to have gotten lost in larger search problems in the benchmark suite, with an average of 11.6 times more expansions in its IOS implementation than WA*.



WA* vs. XDP WA* vs. XUP WA* vs. PWXDP

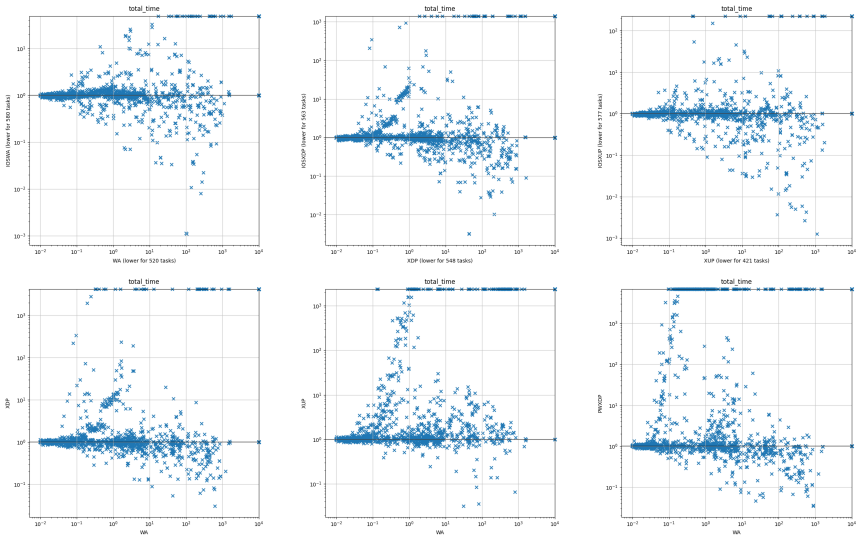
Table 5.1: Relative cost comparison of linear (WA*) and nonlinear evaluation functions in eager BFS algorithm (top) and IOS algorithm (bottom)



WA* vs. XPD WA* vs. XUP WA* vs. PWXDP

Table 5.2: Number of expansions comparison of linear (WA*) and nonlinear evaluation functions in eager BFS algorithm (top) and IOS algorithm (bottom)

A similar trend can be seen in Figure 5.3, comparing the relative time of the nonlinear evaluation functions XDP, XUP, and PWXDP, to the linear WA* evaluation function, in both the eager BFS and IOS implementations. This correlation makes sense, as a larger number of expansions will mean greater time. Referring back to the PWXDP implementation in IOS, the PWXDP function was 2.5 times slower than the WA* function for the same algorithm.



WA* vs. XPD WA* vs. XUP WA* vs. PWXDP

Table 5.3: Relative time comparison of linear (WA*) and nonlinear evaluation functions in eager BFS algorithm (top) and IOS algorithm (bottom)

5.3 Algorithm Comparison

Algorithm	Cost (Weight)	Coverage (%)
A*	1.00x	48.5
WA*	1.08x	62.3
XDP	1.08x	63.2
XUP	1.07x	56.3
PWXDP	1.10x	63.3
IOS-WA*	1.11x	63.7
IOS-XPD	1.12x	64.5
IOS-XUP	1.10x	59.2
IOS-PWXDP	1.13x	56.2

Table 5.4: Effective weight and Coverage Summary for each algorithm- evaluation function pair, in percentage coverage (out of 1827 search problems)

Table 5.4 shows the effective weights of the solution costs of each algorithm and evaluation function pair. As mentioned in previous experiments, this trend was to be expected, and validates the concept on which suboptimal focal search algorithms are based. This finding was one of the two most positively surprising results of this research, as the average effective weights are, even in the IOS algorithm, are under 1.15 times the cost of the optimal solution. The increase in coverage provided by the nonlinear evaluation functions was expected, but a pleasant result, as it validates the existence of suboptimal search algorithms to solve larger and more complex state spaces.

The plots in Appendix A show comparisons of each linear and nonlinear evaluation function in both the eager BFS and IOS algorithms.

6
Future Work

This paper can be viewed as a non-comprehensive introduction of modern suboptimal search algorithms to the Fast Downward planner, lightly laying a foundation for further research in suboptimal search. As well, this research has been a somewhat entrepreneurial attempt to extract theoretical functions and algorithms into practice. The first, and probably most important, area for future work would be in implementing a full version of the IOS algorithm as described in Algorithm 3, providing the capability to switch between search spaces continuously versus just once sequentially. This switching mechanism posed feasibility challenges and was finally dismissed, as even the authors of the initial research paper chose to use a one-time sequential switch from one search space to the other. Secondly, a lazy implementation of the IOS algorithm could be beneficial for comparison to the eager implementation described in this paper. A first attempt at incorporating the nonlinear evaluation functions into a lazy best-first search would also test these nonlinear evaluation functions in a different search environment. In this context, observing how high the suboptimality of the solution plan compares to the search time required would be informative.

To extend the validity of already existing implementations, further testing is recommended. Using a larger suite of benchmarks, testing the implementations with a larger weight range (e.g. weights in increments of 0.1 from 1.1x to 3.0x optimal cost), or even implementing the nonlinear functions and IOS algorithm in other planners would provide more quantitative and qualitative insight into the functionality and usefulness of the research done so far.

7
Conclusion

The evaluation functions showed marked benefits with regards to total time and expansions versus the optimal algorithm with little cost penalty (the average suboptimal solution was only 1.1x the optimal cost, even with a weight of 2). With that, if optimality is not required to solve certain problems, these implementations may have a use case. There was no clear "best" evaluation function, although the XDP function had slightly better overall performance, and the XUP function showed the lowest costs in all suboptimal implementations. Disappointingly, there was no significant advantage to implementing nonlinear evaluation functions over WA* in the standardized benchmarks chosen from the Fast Downward planner. Furthermore, this paper only covers a small part of the testing needed to validate the utility and scalability of nonlinear evaluation functions and the IOS algorithm for suboptimal search.

On a positive note, it was encouraging to reproduce an effect described in some of the referenced research papers - when given a defined weight, suboptimal search implementations return solution paths that are "better" than the weight. This is an encouraging step for suboptimal focal search algorithms to exploit, although more research is needed to truly provide breakthrough improvements to already existing algorithms.

In the greater context of artificial intelligence, even though there was a clear increase in coverage compared to optimal search - from 48.6% in A* to around 61.0% in suboptimal implementations - these implementations of suboptimal search still cannot find solution plans for large search spaces. That said, current research tackling all sides of the issue, from algorithm development to more realistic state space descriptions, may sometime in the future reach significant usability and scalability.

Bibliography

[1] Jingwei Chen and Nathan R Sturtevant. Conditions for avoiding node re-expansions in bounded suboptimal search. *puzzle*, 40:39–753, 2019.

[2] Jingwei Chen, Nathan R Sturtevant, William J Doyle, and Wheeler Ruml. Revisiting suboptimal search. In *SOCS*, pages 18–25, 2019.

[3] David G Churchill. *Heuristic search techniques for real-time strategy games*. PhD thesis, University of Alberta, 2016.

[4] Liron Cohen, Matias Greco, Hang Ma, Carlos Hernández, Ariel Felner, TK Satish Kumar, and Sven Koenig. Anytime focal search with applications. In *IJCAI*, pages 1434–1441, 2018.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107, 1968.

[6] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[7] Malte Helmert. Foundations of artificial intelligence, February - April 2020.

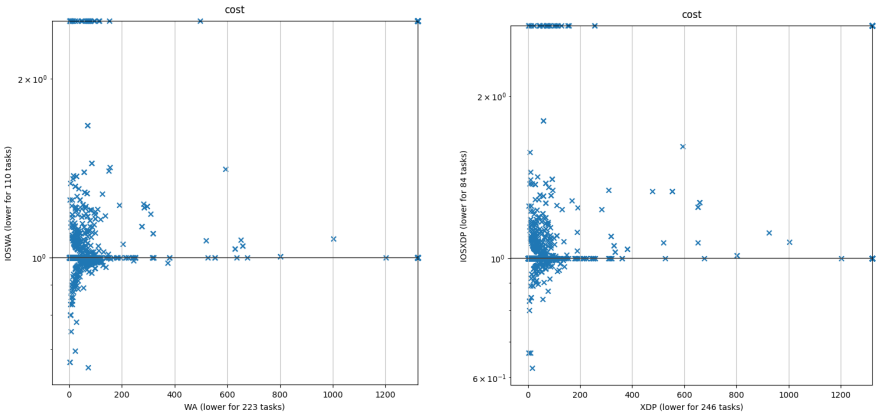
[8] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1 (3-4):193–204, 1970.

[9] Jendrik Seipp and Malte Helmert. Counterexample-guided cartesian abstraction refinement. In *ICAPS*, pages 347–351, 2013.

[10] Jordan Tyler Thayer and Wheeler Ruml. Faster than weighted a*: An optimistic approach to bounded suboptimal search. In *ICAPS*, pages 355–362, 2008.

A

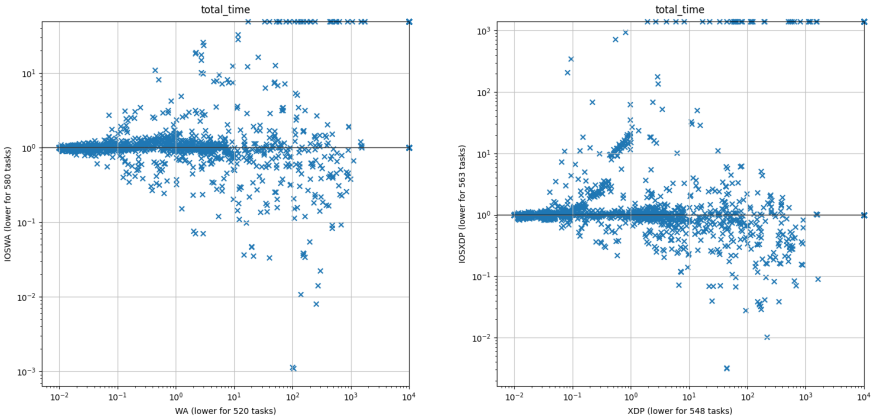
Appendix



"row 1, column 1" "row 1, column 2"

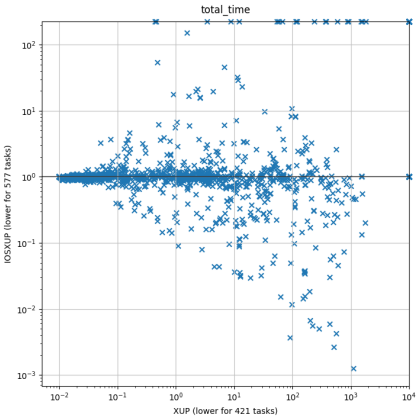
"row 2, column 1" "row 2, column 2"

Table A.1: Relative cost comparison of evaluation functions in eager BFS algorithm and IOS algorithm (preprended "IOS")

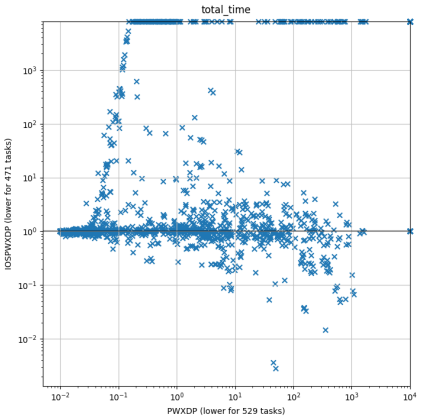


"row 1, column 1"

"row 1, column 2"

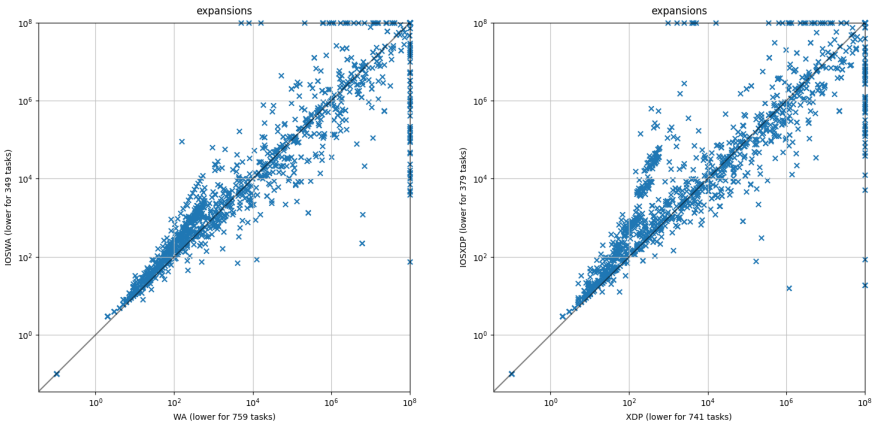


"row 2, column 1"



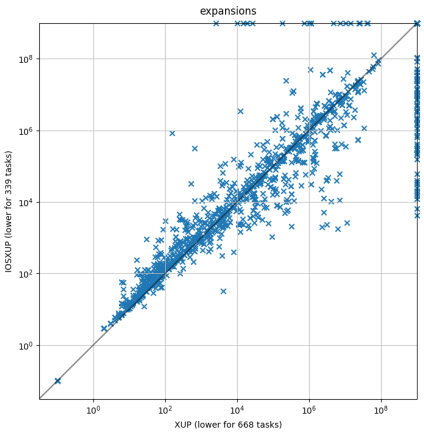
"row 2, column 2"

Table A.2: Relative time comparison of evaluation functions in eager BFS algorithm and IOS algorithm (preprended "IOS")

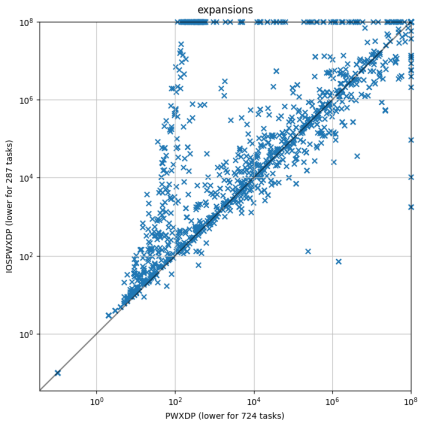


“row 1, column 1”

“row 1, column 2”



“row 2, column 1”



“row 2, column 2”

Table A.3: Number of expansions comparison of evaluation functions in eager BFS algorithm and IOS algorithm (preprended ”IOS”)

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Caroline Steiblin

Matriculation number — Matrikelnummer
2015-615-099

Title of work — Titel der Arbeit
Bounded Suboptimal Search for Classical Planning

Type of work — Typ der Arbeit
Bachelor Thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 09.10.2020

Signature — Unterschrift