The Bench Transition System and Stochastic Exploration

Dawson Tomasz[1], **Richard Valenzano**[2]

Toronto Metropolitan University
dawson.tomasz@proton.me[1], rick.valenzano@torontomu.ca[2]

Abstract

Stochastic exploration has been shown to be an effective way to mitigate the negative impact that heuristic local minima and plateaus can have on Greedy Best First Search (GBFS). Previous work has induced exploration using type systems, which typically partition the state-space using simple features like heuristic value and depth. In this work, we introduce new type systems motivated by the Bench Transition System (BTS). The BTS is a structure used to characterize the behaviour of GBFS. It is based on high water-mark benches, which are sets of states that have made the same amount of progress towards the goal. Since the high water-mark of a state cannot be calculated during search, our type systems approximate the BTS using the notions of Heuristic Improvement and Low Water-Mark. We first identify that these approximations are exact in state-spaces with plateaus but no local minima, and also show that the resulting type systems are probabilistically complete. Our empirical evaluation shows the effectiveness of this approach on a variety of planning domains.

1 Introduction

The Greedy Best First Search (GBFS) algorithm (Doran and Michie 1966), which is core to many automated planners, searches the state space greedily according to guidance by a heuristic function. Due to its inherent greediness, any inaccuracy in the heuristic can produce *uninformed heuristic regions (UHRs)* in the state-space, which GBFS can have trouble getting past. Stochastic exploration has been shown to be an effective method of mitigating this issue (Valenzano et al. 2014; Xie et al. 2014; Kuroiwa and Beck 2022; Asai and Fukunaga 2017). When using these techniques, the search generally alternates between standard greedy expansions where the state is selected solely by heuristic value, and exploratory expansions where the state is selected stochastically and often against the advice of the heuristic.

Type-based exploration is one such stochastic exploration method. It works by bucketing states together according to state features such as heuristic value or path length (Xie et al. 2014). Exploration is then a two step process: a bucket is stochastically selected and then a state is stochastically chosen from that bucket. Type-based exploration techniques have been shown to add useful variation to the search that has been shown empirically to improve search performance.

In this work, we propose two novel type systems based upon the *Bench Transition System (BTS)* (Heusner, Keller, and Helmert 2017, 2018a,b). The BTS is a structure used to analyze the behaviour of GBFS. It identifies the regions of the space that GBFS will potentially search, and conversely identifies the parts of the space that GBFS will never search. Furthermore, it can be used to characterize the UHRs that GBFS may encounter during the search.

Below, we identify that the BTS has several properties regarding how it partitions UHRs and the state-space in general, that would seemingly make it ideal as a type system. Unfortunately, this is not possible since the BTS is based upon the notion of *high water-mark*, which cannot be computed during search. Instead, we identify two methods, called *heuristic improvement* and *low water-mark*, which can be used to approximate high water-mark and can be computed during search, meaning they can be used to define type systems. We formally show that these methods perfectly match the BTS on certain state-spaces with plateaus and no local minima, and they induce probabilistically complete type systems. Finally, we show that these new type systems can greatly outperform existing approaches when added to a basic GBFS-based planner, though find mixed results when added to a fully featured planner.

2 Background

In this section we introduce the notation used, and then review the BTS and relevant stochastic exploration techniques.

State-Spaces and The Bench Transition System

We begin with the basic concepts of a state-space and by reviewing the BTS. The notation used largely follows the original BTS work (Heusner, Keller, and Helmert 2017, 2018b).

A *state space* is defined as $\mathcal{S} = \langle S, s_{init}, S_{goal}, succ \rangle$, with S being the set of *states*, s_I being the *start state*, S_{goal} being a set of *goal states*, and $succ : S \mapsto 2^S$ is the *successor generator function*. Given a set of states S', $succ(S')$ is the successor set of every state in S'.

An s-plan is a path to a goal state starting from s. The set of all s-plans is denoted $P(s)$, and the set of all paths between two states s and s' is denoted $P(s, s')$. Note that

we ignore transition costs in this work. In a slight abuse of notation, we use $s \in \mathcal{S}$ to denote that s is in the set of states for state space $\mathcal{S}$.

A *heuristic function* is $h : S \mapsto \mathbb{R}_0^+$ and we define the heuristic value of a set of states S' as $h(S') = \min_{s' \in S'} h(s')$. A *state space topology* $\mathcal{T} = \langle \mathcal{S}, h \rangle$ is then defined as a state space $\mathcal{S}$ combined with a heuristic h.

Greedy Best-First Search (GBFS) is a classic best-first search algorithm. We largely assume the reader is familiar with the use of open and closed lists in best-first search algorithms, but recall that on every iteration, GBFS selects a state in the open list with the lowest heuristic value for expansion. Ties are broken arbitrarily. Since GBFS is a *satisficing algorithm*, we assume that states are never re-opened.

The Bench Transition System (BTS) formally characterizes the behaviour of GBFS on a given state-space topology. It uses the concept of *high-water mark*, which is defined as

$$hw_h(s) := \begin{cases} \min_{p \in P(s)}(\max_{s' \in p}(h(s'))) & P(s) \neq \{\} \\ \infty & otherwise \end{cases}$$

Notice that a state's high-water mark is lower bounded by its own heuristic. Namely, $hw_h(s) \geq h(s)$. We also define the high water-mark of a set of states S as the minimum high water-mark among them.

A GBFS is said to have made *progress* when a new lowest high water-mark state is encountered. Thus, a *progress state* is defined as follows:

Definition 2.1. Let $\langle \mathcal{S}, h \rangle$ be a state space topology. A state $s \in \mathcal{S}$ is a *progress state* if $hw_h(s) > hw_h(succ(s))$.

Notice that for any progress state, $h(s) = hw_h(s)$. We can now state the definition of a *bench*:

Definition 2.2. Let $\langle \mathcal{S}, h \rangle$ be a state space topology. For any $s \in \mathcal{S}$, a *bench*, denoted $\mathcal{B}(s)$, is a tuple $\langle I, E \rangle$. The *inner states* I is the set of all states $s'' \neq s$ that can be reached from s along paths on which all states $s' \neq s$ are non-progress states and satisfy $h(s') \leq hw_h(succ(s))$. The set E of *exit states* is the set of all progress states s' with $h(s') \leq hw_h(succ(s))$ that are successors of s or of some inner bench state of s.

We refer to the *bench level* of $\mathcal{B}(s)$ as $level(s) = hw_h(succ(s))$. We will also abbreviate $\mathcal{B}(s)$ as $\mathcal{B}$ when s is clear from context, use $I(\mathcal{B})$ and $E(\mathcal{B})$ to denote the inner and exit sets respectively of bench $\mathcal{B}$, and use $s' \in \mathcal{B}(s)$ to mean $s' \in I(\mathcal{B}(s)) \cup E(\mathcal{B}(s))$.

A *Bench Transition System* is then a directed graph whose vertices are benches and whose edges correspond to a progress state leading to a new bench:

Definition 2.3. Let $\mathcal{T} = \langle \mathcal{S}, h \rangle$ be a state space topology with initial state s_I. The *bench transition system* of $\mathcal{T}$, denoted $BTS(\mathcal{T})$, is a directed graph $\langle V, E \rangle$ where V and E are defined inductively as follows:
1. $\mathcal{B}(s_I) \in V$
2. If $\mathcal{B}(s) \in V, s' \in E(\mathcal{B}(s))$, and $s' \notin S_{goal}$ then $\mathcal{B}(s') \in V$ and $\langle \mathcal{B}(s), \mathcal{B}(s') \rangle \in E$

A BTS breaks a GBFS search into a set of episodes, each consisting of the expansion of a progress state and the progression to a bench with a lower level. Heusner, Keller, and Helmert (2018b) also identified that a BTS can be used to formalize different types of *uninformed heuristic regions* (UHRs), such as a *crater* (*ie.* a local minima):

Definition 2.4. Let $BTS(\mathcal{T}) = \langle V, E \rangle$ and s be a state such that $\mathcal{B}(s) \in V$. A state $s' \in \mathcal{B}(S)$ is called a *crater entry state* if $s' \in I(\mathcal{B}(s))$, $s' \notin E(\mathcal{B}(s))$, $h(s') = level(s)$, and $\exists s'' \in succ(s')$ such that $h(s'') < level(s)$. A *crater* is defined as the set of all states s' reachable from a crater entry s along a path on which for any s'', $h(s'') < level(s)$.

Once a crater entry state is expanded, the search will have made heuristic progress but not high water-mark progress. It will then have to expand every state in that crater before it can move on to the next bench (Heusner, Keller, and Helmert 2017). Craters/local minima are important state space features since the total search effort of GBFS strongly correlates with the size and depth of the largest local minima encountered during the search (Cohen and Beck 2018).

Plateau are another form of UHR. In a BTS, a plateau is a subset of states on the same bench that all have the same heuristic value (Heusner, Keller, and Helmert 2017). Large plateau obstruct search by making the heuristic no more informed than a random search (Xie, Müller, and Holte 2014).

Stochastic Exploration

Stochastic exploration can mitigate the impact of UHRs on GBFS by stochastically choosing states for expansion, often against the advice of the heuristic function. This can often helpfully diversify the search in the event that GBFS is stuck in a UHR, either by allocating some effort off the UHR or by helping find a way off of it.

The different approaches for introducing exploration generally alternate (or use a similar mechanism) between selecting nodes from the open list for expansion greedily according to h, and sampling nodes stochastically from the open list according to some defined procedure. The methods largely differ in the procedure used. *ϵ-GBFS* samples states uniformly from amongst those in the open list (Valenzano et al. 2014). However, if the open list is dominated by a single UHR, uniform sampling is biased to largely only select from within that UHR. *Type-GBFS* addresses this issue by sampling uniformly over *types* instead of the whole open list (Xie et al. 2014). This approach uses a given *type system*, which partitions the nodes into different buckets:

Definition 2.5. If $\mathcal{S}$ is a state space, then $\mathbb{T} = \langle T_1, ..., T_n \rangle$ is a *type system* of open list $OPEN \subseteq S$ if it is a disjoint partitioning of $OPEN$.

Exploration in Type-GBFS involves two steps. First, a type is selected uniformly at random from amongst those with nodes in the open list. Next, a node is selected uniformly at random from amongst those in the selected type. This removes the bias towards sampling high cardinality types. In the original work, the most effective type system bucketed nodes together if they shared the same h-cost and g-cost (Xie et al. 2014). We call this the $\langle h, g \rangle$ *type system*.

Kuroiwa and Beck (2022) later introduced biased type selection. This method also uses the $\langle h, g \rangle$ type system. It does

so by first sampling an h-cost in a way that is biased towards low h-costs. A bucket from amongst those with the selected h-cost is then selected uniformly at random, and a node is selected randomly from that bucket. This approach is motivated by the fact that while the the heuristic function may be imperfect, it still usually does well to discriminate between those nodes far away from the goal and those reasonably close. The most effective biasing scheme introduced by Kuroiwa and Beck was *Softmin-Type*, which sets the probability of selecting a heuristic value h' as

$$softmin(h') = \frac{exp(-h'/\tau)}{\sum_{h'' \in OPEN} exp(-h''/\tau)} \quad (1)$$

where the denominator is the sum over the unique h-values in $OPEN$, and $\tau > 0$ is the *temperature* parameter.

We note that under reasonable conditions, most of the existing stochastic approaches are *probabilistically complete* on infinite graphs, meaning that the probability these algorithms solve any search problem approaches 1 as the number of node expansions goes to infinity (Valenzano and Xie 2016). This is not true for standard GBFS, which can get stuck on an infinite path if the the heuristic is misleading enough. This is a desirable property even in finite graphs, since it decreases the risk that the search is led arbitrarily astray by incorrect heuristic information.

3 The BTS as a Type System?

Suppose we had a type system that bucketed together nodes if and only if they were on the same bench. We refer to this hypothetical system as the *BTS type system*. Below, we discuss the impossibility of realizing this type system, but first identify several positive attributes that such a type system would have. This will help motivate the new type systems introduced in Section 4, which are based on approximations of the BTS. We begin with the following observation:

Observation 3.1. *In the BTS type system, all nodes within the same plateau or crater will reside in the same bench.*

A consequence of this observation is that the exploration induced by the BTS type system will mitigate the cardinality bias of large UHRs when used as part of Type-GBFS. That is, since all the nodes in an UHR will be of the same type, the likelihood of selecting a node from a large UHR will be the same as selecting from a small UHR. Thus the exploration will not be dominated by these large UHRs.

Asai and Fukunaga (2017) distinguished between *intra-UHR* exploration[1] (exploring within a UHR) and *inter-UHR* exploration (exploring between UHRs). These different modes were each found to be beneficial, while also orthogonal to each other (Asai and Fukunaga 2017). By Observation 3.1, a search using the BTS type system is explicitly choosing between UHRs, and is thus doing inter-plateau exploration. When selecting from amongst nodes of a given type, it is selecting within a UHR, and thus is doing intra-plateau exploration. This means that type-based exploration using the BTS type can easily incorporate more complex strategies for each type of exploration by simply changing the sampling strategy for each phase of exploration.

Our next observation relates to the fact the BTS type system only buckets together nodes that are "near" each other in the state-space:

Observation 3.2. *Nodes n_1 and n_2 can only be in the same type in the BTS type system, if the deepest progress state on the paths common to both n_1 and n_2 is the same.*

This property means that nodes found along very different paths — and thus likely to be in very different locations of the search space — are unlikely to be grouped together unless they are both in a massive UHR. The standard $\langle h, g \rangle$ type system does not have this property: states in very different parts of the state-space may have the same h and g values, and thus may be bucketed together. In the terminology of Asai and Fukunaga 2017, the BTS type system would induce *breadth*-based diversification since it would distinguish between nodes at the same depth but in different parts of the state-space. It would also induce *depth*-based diversification, meaning that exploration across different depths would be encouraged. While nodes at different depths may be bucketed together, depth is implicitly a part of the BTS type system since each new deepest progress state creates a new type just as each new depth does in the $\langle g, h \rangle$ type system. Asai and Fukunaga (2017) showed that neglecting to perform both modes of diversification can result in pathological behaviour.

Unfortunately, except in certain cases such as those in the next section, the BTS type system described above cannot be used in practice. The first issue is that in its original formulation, any node that is not possibly expanded by GBFS is not included in the BTS. As there are very likely cases where the heuristic is misleading enough that the search would be better off explicitly trying to "get off the BTS", this is not desirable behaviour for exploration in satisficing search.

The above issue can be addressed by adding the nodes omitted from the BTS to the same type as the nearest ancestor that is a progress state. We use a similar approach when defining our type systems. However, the next issue does not obviously have such a solution. Recall that the BTS is built upon the notion of high-water mark. This property requires knowledge of the state-space that is unavailable at the time that a node is generated. After all, the high water-mark of s is calculated from the heuristic values along all s-plans, which are precisely what's being searched for. Therefore, the BTS cannot be immediately used to construct a type system.

4 BTS-Approximate Type Systems

In this section we first introduce *subspaces* to facilitate BTS approximations. We then introduce two such approximations and show they are equivalent to the BTS in state spaces without local minima. Finally, we define type systems based on these approximations and show that Type-GBFS using these type systems is probabilistically complete.

From Benches to Subspaces

Before introducing our approximations, we begin with several observations about benches. First, the states in bench

[1] The original work refers to inter- and intra-plateau exploration, but extending these concepts to UHRs is straightforward.

$\mathcal{B}(s)$ are contiguous in that they are all reachable from the *root* s of the bench. Second, any s' on the bench must satisfy a *bench candidacy test*, namely that $h(s') \leq level(s)$. Finally, any exit state s' must satisfy an *exit test*, namely that s' is a progress state and $h(s') \leq level(s)$. Using these observations, we can now generalize the concept of a bench as a *state-space subspace*, or *subspace* for short, by allowing for arbitrary candidacy and exit tests.

Definition 4.1. Let $\tau = \langle \beta, \pi \rangle$ be a tuple of boolean functions from $S \times S$ to $\{\top, \bot\}$ for state-space topology $\langle \mathcal{S}, h \rangle$. π is called the *exit test* and β is the *candidacy test*. For any $s \in S$, the *τ-based subspace of s*, denoted by $\mathbb{B}_\tau(s)$, is the tuple $\langle I, E \rangle$, where I is a set of *inner subspace states* and E is a set of *subspace exit states*. I and E are defined as:

$$\begin{aligned} I =& \{s' \in S \mid \exists p \in P(s, s') : \\ & \forall s'' \in p, \pi(s, s') = \bot \wedge \beta(s, s'') = \top\} \\ E =& \{s' \in S \mid (s' \in succ(I) \vee s' \in succ(s)) \\ & \wedge (\pi(s, s') = \top \wedge \beta(s, s') = \top)\} \end{aligned}$$

We will use $\mathbb{B}_\tau$ to refer to $\mathbb{B}_\tau(s)$ when s is clear from context, and $I(\mathbb{B}_\tau)$ and $E(\mathbb{B}_\tau)$ to denote $\mathbb{B}_\tau$'s inner and exit sets, respectively.

We now note that Definition 4.1 simply replaces the specific candidacy and exit tests from Definition 2.2, with generic β and π tests. In particular, a bench can be expressed as a subspace using

$$\beta(s, s') = \begin{cases} \top, & h(s') \leq hw_h(succ(s)) \\ \bot, & otherwise \end{cases}$$

$$\pi(s, s') = \begin{cases} \top, & hw_h(s') > hw_h(succ(s')) \\ \bot, & otherwise \end{cases}$$

That is, a bench is a special case of a subspace.

The Heuristic Improvement and Low Water-Mark Subspaces

We can now use alternative candidacy and exit tests to define new subspaces that approximate benches. The first is the *heuristic improvement subspace*, which considers "progress" to be made any time that a state s has a successor with a lower heuristic value than s:

Definition 4.2. The *heuristic improvement subspace of state s*, denoted $\mathbb{B}_{hi}(s)$, is the τ_{hi}-based subspace of s where $\tau_{hi} = \langle \beta_{hi}, \pi_{hi} \rangle$, and where $level_{hi}(s) = h(succ(s))$, these tests are defined as follows:

$$\beta_{hi}(s, s') = \begin{cases} \bot, & s' \in succ(s) \wedge h(s') > level_{hi}(s) \\ \top, & otherwise \end{cases}$$

$$\pi_{hi}(s, s') = \begin{cases} \top, & h(succ(s')) < h(s') \\ \bot, & otherwise \end{cases}$$

In a heuristic improvement subspace, the exit states each have a child which makes heuristic improvement, and the inner states are those that are reachable from s on a path along which the heuristic never decreases.

Our next new subspace is based on the *low water-mark* of a path. Given a path $p = \langle s_0, ..., s_n \rangle$ where s_0 is the initial state, the *low water-mark* of p is $lw_h(p) = min_{s' \in p} h(s')$. A low water-mark subspace then defines exit states as those with a successor which sets a new lower water-mark:

Definition 4.3. The *low water-mark subspace of s*, denoted $\mathbb{B}_{lw}(s)$, is the τ_{lw}-based subspace of s where $\tau_{lw} = \langle \beta_{lw}, \pi_{lw} \rangle$, and where $level_{lw}(s) = h(succ(s))$, these tests are defined as follows:

$$\beta_{lw}(s, s') = \begin{cases} \bot, & s' \in succ(s) \wedge h(s') > level_{lw}(s) \\ \top, & otherwise \end{cases}$$

$$\pi_{lw}(s, s') = \begin{cases} \top, & h(succ(s')) < level_{lw}(s) \\ \bot, & otherwise \end{cases}$$

We note that we are slightly abusing notation, as the low water-mark for s can only be defined based on a path from the initial state to s, not using s alone. The subspace definition could be extended to be path-dependent to account for this, but we defined it in terms of a state for the sake of readability. Given the conditions considered in the next section, the analysis below applies for this path-dependent case.

Recall that the BTS is not suitable as a type system since it relies on the high water-mark of states, which is unknown during search. The new subspaces only depend on information available when a state is generated. So while high water-mark, by definition, accounts for deadends and craters, in our subspace approximations, craters will register as progress and deadends won't be recognized (short of a heuristic that recognizes them) misaligning our transition systems and the BTS. However, in the next section we identify a case where our approximations are exact.

BTS Equivalence in Simple Local-Minima Free Domains

We will now show that both of the new subspaces are equivalent to benches for a simple family of state-spaces that contain plateaus but not local minima. This type of state-space is formalized as follows:

Definition 4.4. A state-space topology $\langle \mathcal{S}, h \rangle$ is *monotonically non-increasing* if for every path $p(s_0, s_n) = \langle s_0, s_1, ...s_n \rangle$ through $\mathcal{S}$ starting with the initial state, $\forall s_j, s_{j+1} \in p(s_0, s_n)$, with $0 \leq j < n$, $h(s_j) \geq h(s_{j+1})$.

Where a *dead-end* is a state s with no successors, we show that the heuristic of s is equal to its high and low water-marks in such state spaces:

Lemma 4.1. *If $\langle S, h \rangle$ is a monotonically non-increasing state space topology with no dead-ends, then for all $s \in S$, $h(s) = hw_h(s) = lw_h(s)$.*

Proof. Consider path $\langle s_0, ..., s_i, ..., s_n \rangle$ where s_0 is the initial state, s_n is a goal state, and $s = s_i$. By Definition 4.4, the heuristic value of any state from s_{i+1} to s_n must be less than or equal to $h(s_i)$. Since this is true of any such path, $hw_h(s) = h(s)$. Similarly, $h(s_i)$ is no greater than the heuristic value of any state from s_0 to s_{i-1}, so $lw_h(s) = h(s) = hw_h(s)$. □

These state-space also have no craters (*ie.* local minima):

Lemma 4.2. *If $\langle S, h\rangle$ is a monotonically non-increasing state space topology with no dead-ends, then for any state s, $\mathcal{B}(s)$ does not contain a crater and for $\mathcal{P} = I(\mathcal{B}(s)) \cup E(\mathcal{B}(s))$, it is true that $\forall s' \in \mathcal{P}, h(s') = level(s)$.*

Proof. Suppose there is a state s' that is in a crater on $\mathcal{B}(s)$. By Definition 2.4, this means that $h(s') < level(s)$. Since Lemma 4.1 states that $hw_h(s') = h(s')$, this means that $hw_h(s') < level(s)$. But this is only possible if s' was the descendent of a progress state and thus not on $\mathcal{B}(s)$. Thus, there can be no craters by contradiction.

The second part of the statement follows since $h(s') \leq level(s)$ by the bench candidacy test. □

We can now show the equivalence of benches and the new subspaces on these types of state spaces:

Lemma 4.3. *Let $\mathcal{T} = \langle S, h\rangle$ be a monotonically non-increasing topology with no dead-ends and let $s \in S$ be progress state in $BTS(\mathcal{T})$. Then $\mathcal{B}(s) = \mathbb{B}_{hi}(s) = \mathbb{B}_{lw}(s)$.*

Proof. Below, $\mathcal{B}(s)$, $\mathbb{B}_{hi}(s)$, and $\mathbb{B}_{lw}(s)$ are abbreviated as $\mathcal{B}$, $\mathbb{B}_{hi}$, and $\mathbb{B}_{lw}$, respectively. By Lemma 4.2, $\forall s' \in I(\mathcal{B}) \cup E(\mathcal{B}), h(s') = level(s)$. From Lemma 4.1, $h(succ(s)) = hw_h(succ(s)) = lw_h(succ(s))$ and so $level_{hi}(s) = level(s) = level_{lw}(s)$.

We start by showing that the inner sets of the subspaces are equal. Recalling Definition 4.1, to show the inner sets are equal, it is sufficient to show that for each subspace, the condition $\pi(s', s) = \bot \wedge \beta(s, s') = \top$ is equivalent.

First note that $\pi_{hi}(s', s) = \top$ when $h(succ(s')) < h(s)$ and $\pi_{hw}(s', s) = \top$ when $hw_h(succ(s')) < hw_h(s)$. With $h(s) = hw_h(s)$, π_{hi} and π_{hw} are equivalent. From Lemma 4.2 we know that $\forall s' \in \mathcal{B}(s), h(s') = level(s)$. Any state $s' \in \mathbb{B}_{hi}(s)$ with $h(s') < level(s)$ would necessarily have been reached by a progress state and so cannot exist; furthermore, any state $s' \in \mathbb{B}_{hi}(s)$ with $h(s') > level(s)$ would violate the monotonicity of the space given the β_{hi} test, meaning $\forall s' \in \mathbb{B}_{hi}, h(s') = level(s')$. Therefore, all states $s' \in I(\mathbb{B}_{hi})$ also satisfy $h(s') = level(s)$.

The argument for low water-mark is nearly identical. The low water-mark notion of progress is equivalent to high water-mark and heuristic improvement in these spaces, and the low water-mark candidacy test is identical to heuristic improvement's. Therefore $I(\mathcal{B}) = I(\mathbb{B}_{hi}) = I(\mathbb{B}_{lw})$.

We next show that the exit sets are equivalent. We know from above that $I(\mathcal{B}) = I(\mathbb{B}_{hi}) = I(\mathbb{B}_{lw})$ and that the three progress tests are all equivalent; therefore to show that $E(\mathcal{B}) = E(\mathbb{B}_{hi}) = E(\mathbb{B}_{lw})$ we need only consider the β test for the cases that $s' \in succ(s)$ or $s' \in succ(I)$.

The β predicates of each will prohibit successors of s that do no satisfy $h(s') = level(s)$.

If $s' \in succ(I)$ then the β predicates are irrelevant. In all cases, due to the monotonicity of the space, if a successor of I, s', has a lower heuristic value than $level(s)$ then a progress state must have been traversed to reach s', which is impossible since I contains no progress states. Therefore exit states in the three subspaces will satisfy $h(s') = level(s)$, making $E(\mathcal{B}) = E(\mathbb{B}_{hi}) = E(\mathbb{B}_{lw})$. □

Now notice that the inductive procedure in Definition 2.3 can be used to create a transition system for any subspace definition, not just benches. We refer to the new transition systems for $\mathcal{T}$ as the *Heuristic Improvement Transition System (HITS)*, denoted $HITS(\mathcal{T})$, and the *Low Water-Mark Transition System (LOTS)*, denoted $LOTS(\mathcal{T})$. The following theorem formalizes the fact that these are all equivalent in monotonically non-increasing state-spaces.

Theorem 4.4. *For a monotonically non-increasing state-space topology $\mathcal{T}$ with no dead-ends and any state s, $\mathcal{B}(s) \in BTS(\mathcal{T}) \iff \mathbb{B}_{hi}(s) \in HITS(\mathcal{T}) \iff \mathbb{B}_{lw}(s) \in LOTS(\mathcal{T})$.*

Proof. We can now show the statement is true by considering "synchronizing" the generation of $BTS(\mathcal{T})$, $HITS(\mathcal{T})$, and $LOTS(\mathcal{T})$ according to the procedure in Definition 2.3. That is, every time a state is selected from an exit set in a bench, that same exit state is selected to generate a new subspace for $HITS(\mathcal{T})$ and $LOTS(\mathcal{T})$. We will now show by induction that the vertex sets of these transition systems are identical after each bench/subspace is added.

The base case is after the bench for the initial state s_I is added. Clearly, all three will have a single bench after this step. If s_I is a progress state (*ie.* it has a successor with a lower heuristic value than it), then the fact that the bench and subspaces are equal follows immediately by Lemma 4.3. If not, then for any $s'' \in succ(s)$, $h(s'') = h(s)$ since the state-space is monotonically non-decreasing. Thus, $level(s) = h(s) = level_{lw}(s) = level_{hi}(s)$, and so the same argument as in Lemma 4.3 can be used to show that $\mathcal{B}(s_{init}) = \mathbb{B}_{hi}(s_{init}) = \mathbb{B}_{lw}(s_{init})$.

We now assume the vertex sets are identical after $k \geq 1$ benches have been added. If there are no more exit vertices in $BTS(\mathcal{T})$ to handle, then the same must be true of $HITS(\mathcal{T})$, and $LOTS(\mathcal{T})$. This holds since the inductive hypothesis guarantees the vertex sets and exit sets are identical and so none could have been missed. Thus, the transition system generating procedure would terminate for all three, and the statement holds in this case.

If there is a remaining exit state to handle in the BTS, then it must also be in the other transition systems, since the vertex sets are identical by the inductive hypothesis. The new bench generated will then be identical to the new subspaces generated by Lemma 4.3. Therefore the vertex sets will be the same after the $k+1$-st step, and thus the statement holds. □

Thus, HITS and LOTS are perfect approximations of the BTS in the case of montonically non-increasing state-spaces with no dead-ends. While exact equivalence only holds in this restricted setting, recall that the motivation is to produce type systems that approximate the BTS while being computable during search. Even if the state-space was not exactly monontonically non-increasing, we would still expect the subspaces to be fairly similar if the deviation from this requirement was not too large. However, future work is needed to verify this hypothesis.

New Type Systems

We can now consider using the heuristic improvement and low water-mark subspaces to define type systems. Since type systems must cover the entire state space, we must deviate slightly from the subspaces they are based upon. In Definitions 2.2 and 4.1, the states that are in $I \cup E$ must satisfy a candidacy condition and cannot include any successors of an exit state. Moreover, only those successors of a progress state that actually make progress are included in the subspace. Both of these causes of states being discarded from a subspace must be relaxed in order to ensure the state space is covered by the type system. And so in our type systems, a notion of progress is used, bench candidacy is not used, and the non-improving children of progress states are included.

In the case of heuristic improvement, a state s' has made progress from its parent s if $h(s') < h(s)$. And so upon expanding state s, if s has a child with an improved heuristic value, then a a single new type is created and all improving members of $succ(s)$ are added to the new type. When a state s' is generated that does not improve upon the heuristic value of its parent s, it is assigned to the same type as s. For a non-progress state s, this corresponds to s' being in the same subspace as s. However, if s is a progress state, this is a relaxation of the inner bench set condition that states be reachable by non-progress states only, which is needed to ensure the type system covers the entire state space.

An analogous approach is used for the low water-mark type system. A state s' has made progress from its parent s if $lw_h(s') < lw_h(s)$. Upon expanding s, if it's a low water-mark progress state, a new type is created for each new low water-mark value seen with a state in $succ(s)$ (*ie.* lower than the one associated with s's type). States that do not result in an improved low water mark will be assigned s's type. Importantly, since the type of a state (either heuristic improvement or low water-mark) does not change after it is first generated, these processes are both incremental and thus enable the efficient maintenance of the type buckets.

We note that these type systems create *type trees* which roughly correspond to the transition systems they're based on. However, they deviate slightly from the way the associated subspaces are defined above in the case of non-unit cost domains or with an inconsistent heuristic. In these cases there can be varying amounts of progress from a progress state to its children. Since the type systems cover the open list they must include all amounts of progress. In the heuristic improvement type system, this was achieved by including all improving successors of a progress state into the new type, not just the most improving. In the low-water mark type system, this was achieved by creating a new type for each new low water-mark encountered among the children, instead of just those with the most improvement. This was done as opposed to putting them all into the same type to avoid the case of having states with different low water-mark values in the same low water-mark type.

Finally, we note that type-based exploration using these type systems remains probabilistically complete under reasonable conditions. These results follow from the fact that type-based exploration is probabilistically complete if there are a finite number of types (Valenzano and Xie 2016). In the case of the heuristic improvement type system, this will hold if every path through the state space has no more than $N < \infty$ heuristic improvements along it (*ie.* there are no endlessly oscillating heuristic values). In the case of the low water-mark type system, there are a finite number of types if there is a finite number of low water-mark values between the initial state and any goal state. This will hold if the heuristic values can only take on non-negative integer values, for example.

5 Empirical Evaluation

To evaluate our new type systems, we used benchmark problems from the satisficing tracks of IPC 2011, 2014, and 2018. In domains that were included in both 2011 and 2014 sets, the 2014 version was taken. In total, 496 instances were used, all of which were treated as unit-cost problems. Experiments were run on a VMware Virtual Platform using an 8 core, Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz with a 16 KiB L1 cache, with a 10 minute time limit and a 3.5 GB memory limit per instance. All methods were implemented in Fast Downward (Helmert 2006; Artificial Intelligence Group - University of Basel 2023), using the FF heuristic (Hoffmann and Nebel 2001). For stochastic methods, we ran each 5 times and averaged the results.

Adding Exploration to GBFS

In the first set of experiments, we consider adding exploration to standard GBFS using *queue alternation*. This method uses a standard GBFS queue and an exploration queue, both containing the same states. The search alternates between which queue is selected from on each search step.

We test against 3 baselines from the literature: (1) *standard GBFS*, (2) *Type-GBFS* using the $\langle h, g \rangle$ type system, and (3) *Softmin-Type* using the default temperature of $\tau = 1$ for all instances. For evaluating our new type systems, we exploit the fact that since each type consists of a contiguous subspace that often contains a UHR, it is easy to use different strategies for inter and intra-UHR exploration separately. The former is accomplished by selecting a type, and the latter by selecting a node from that type. For these strategies, we consider 3 sampling techniques. The first is the standard *Uniform selection (U)*. The second is *Biased Heuristic Selection (H)*. Here, we use the softmin in Equation 1 from Softmin-Type. When this approach is used for type selection, the heuristic value of the type is set as the minimum value of any state in the type. The third selection approach used is *Depth selection (D)*. This method was only used for type selection. It is analogous to the softmin biasing, except we bias in favour of types deeper in the type tree. In the rest of this section, we use the following notation for referring to different configurations: $[\text{type system}]^{[\text{type selection}]}_{[\text{state selection}]}$. For example, HI^D_U denotes a heuristic improvement type system with depth type selection and uniform state selection, and LW^D_H denotes a low water-mark type system with depth type selection and heuristic state selection.

Due to the extensive runtime, we first evaluated our new type systems on just the 2011 and 2014 domains. The results are shown in Table 1. The per-domain results are omitted in

Algorithm	Coverage
GBFS	198.0
Type-GBFS	224.2
HI^U_U-GBFS	206.4
LW^U_U-GBFS	204.2
HI^H_U-GBFS	255.6
HI^H_H-GBFS	263.4
HI^D_U-GBFS	**295.0**
HI^D_H-GBFS	284.2
LW^H_U-GBFS	256.4
LW^H_H-GBFS	254.4
LW^D_U-GBFS	258.6
LW^D_H-GBFS	263.6

Table 1: Coverage over 376 problems from the IPC 2011 and 2014 domains.

the interest of space. Of the configurations that use uniform selection for both type selection and state selection (just below GBFS in Table 1), Type-GBFS was the best. It on average solved 17 more instances than the runner-up and was roughly as good or better in all tested domains.

Doing a non-uniform type selection step made the biggest difference, as seen in the bottom part of Table 1. In general, it appears that heuristic improvement types have higher coverage than low water-mark types, depth selection performs better than heuristic selection, and HI$^D_U$is substantially better than all the other configurations, with the runners up being HI^D_H, LW^D_H, and HI^H_H. Interestingly, the improved performance of depth sampling over the other configurations mostly arose in just a couple domains such as visitall-14 and transport-14 with HI^D_Udominating those while performance was relatively similar in the other tested domains.

Having identified the best performing of our new configurations, HI$^D_U$and LW^D_U, we then compare to Softmin-GBFS on all domains from 2011, 2014, and 2018. Table 2 summarizes the results. We first note that the three biased modes all improved upon Type-GBFS. Softmin-Type was slightly better than LW^D_H, but HI$^D_U$was the best performer, solving 40 more problems than the runner up. Importantly, the advantage of HI^D_Uholds on the 2018 domains, even though it was selected based only on the 2011 and 2014 domains. In general, all three biased type systems performed similarly well on most domains and HI^D_Uperformed much better on a select few—like visitall-14, transport-11, data-networks-18, and tetris-14; the reason for this discrepancy is left as future work. Without any parameter tuning, HI$^D_U$and LW^D_Hproved effective exploration enhancements for GBFS.

While HI^D_Uhad the best coverage, it also finds longer plans. Figure 1 summarizes the results of a plan length comparison with Softmin-GBFS. Each point is a problem that both algorithms solved with the red line indicating where the plan lengths are equal. Softmin-GBFS dominated HI^D_U, while LW^D_Uappeared to slightly out perform Softmin-GBFS. This perhaps suggests that heuristic improvement and low water-mark type systems are somewhat orthogonal.

	Type	Soft	HI^D_U	LW^D_H
barman-14	9.0	15.4	20.0	19.0
childsnack-14	0.2	0.0	0.0	0.2
elevators-11	14.4	16.0	16.0	16.0
floortile-11	8.2	7.4	6.8	7.4
ged-14	17.8	20.0	20.0	20.0
hiking-14	20.0	20.0	20.0	20.0
nomystery-11	18.4	12.2	8.6	9.4
openstacks-14	0.0	14.6	16.6	16.6
parcprinter-11	20.0	20.0	19.8	19.8
parking-11	14.2	19.8	15.8	20.0
pegsol-11	20.0	20.0	20.0	20.0
scanalyzer-11	18.8	19.6	19.2	19.8
sokoban-11	18.6	19.0	17.0	18.8
tetris-14	6.6	15.8	20.0	10.0
thoughtful-14	14.0	9.6	8.6	10.8
tidybot-11	16.2	16.2	17.6	16.4
transport-11	0.0	0.8	13.2	1.2
visitall-14	0.0	0.0	19.6	1.4
woodworking-11	7.8	19.4	16.2	17.0
2011-2014 Total:	224.2	265.8	**295**	263.8
agricola-18	10.8	10.0	10.8	11.8
data-network-18	5.6	10.0	15.6	8.8
organic-synthesis-18	3.0	3.0	3.0	3.0
organic-synth-split-18	9.8	9.6	9.6	8.4
snake-18	5.2	19.2	19.2	7.4
spider-18	10.0	5.0	9.2	9.4
termes-18	13.2	12.8	12.0	14.6
2018 Total:	57.6	69.6	**79.4**	63.4
Total:	281.8	335.4	**374.4**	327.2

Table 2: Average Coverage across domains from IPC 2011, 2014, and 2018. Type and Softmin abbreviate Type-GBFS and Softmin-GBFS respectively.

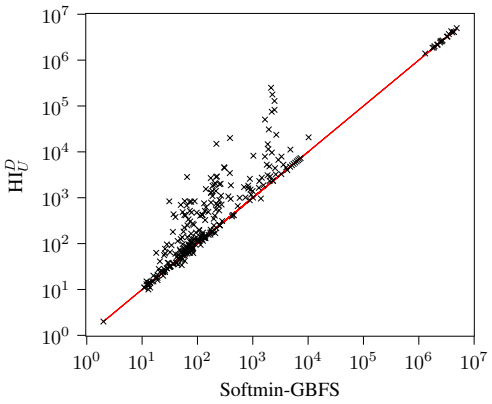
Adding Exploration to Lama-2011

In our next set of experiments, we experiment with adding exploration to a fully featured planner in LAMA-2011 (Richter and Westphal 2010). For this investigation, we considered 4 planners. The first is the original version of LAMA which uses 4 search queues: two for the different heuristics and 2 for the different preferred operators. The second is *Type-Lama*, which adds a fifth queue for type-based exploration using $\langle h, g \rangle$. The third is *Softmin-LAMA* which adds a fifth queue using Softmin-Type. Finally, we test *HI^D_U-LAMA*, which adds a fifth queue employing HI^D_Uselection

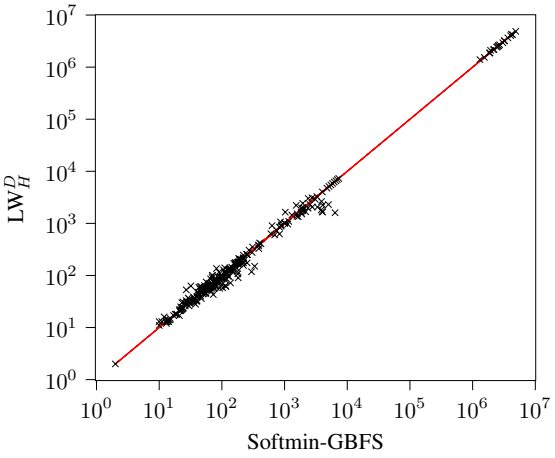
The results are summarized in Table 3. HI^D_U, the best GBFS enhancement, did not perform well as a LAMA enhancement. LAMA employing a HI^D_Uqueue in fact performed worse than standard LAMA, suggesting it was simply added overhead. Type-LAMA and Softmin-LAMA performed very similarly with Type-LAMA being slightly better, and both performed better than LAMA. It is interesting to note that biased selection was not dominant here like it was in GBFS enhancements. This suggests that GBFS and LAMA benefit from different kinds of exploration.

6 Related Work

Outside type systems, other GBFS exploration techniques have been proposed. *Invasion Percolation (IP)* targets intra-plateau exploration by inducing exploration that approxi-



(a) HI_U^D vs. Softmin-GBFS



(b) LW_H^D vs. Softmin-GBFS

Figure 1: Plan length comparison: Softmin, HI_U^D, and LW_H^D.

mates building a spanning tree of a random graph (Asai and Fukunaga 2017). *Local exploration* methods, such as a local GBFS or random walk have also been explored for finding exits to a local minimum (Xie, Müller, and Holte 2014).

There are also many deterministic enhancements to GBFS aimed at introducing diversity to the search. *Multiple heuristic search* alternates between multiple queues, each one ordered by a different heuristic (Röger and Helmert 2010). *Best First Width Search* introduces exploration based on *novelty*, with novel states being preferred (Lipovetzky and Geffner 2017). Identifying how to best combine these ideas with our BTS-inspired type systems remains future work.

Recent approaches for parallel GBFS algorithms have explicitly attempted to ensure the search stays on the BTS so as to minimize the number of bench paths explored. Kuroiwa and Fukunaga (2019) proposed the parallel GBFS algorithm LG, which, using knowledge from the BTS, empirically greatly reduced the number of bench paths expanded during search. Later work expanded on this to create parallel GBFS algorithms which are guaranteed to only expand states that are in the BTS (Kuroiwa and Fukunaga 2020; Shimoda and Fukunaga 2023). Bench-based type systems may have useful applications in BTS inspired parallel GBFS techniques.

	LAMA	**Type**	**Soft**	**HI_U^D**
barman-14	20.0	20.0	20.0	20.0
childsnack-14	5.0	6.0	7.0	11.0
elevators-11	16.0	16.0	16.0	16.0
floortile-11	6.0	7.0	6.2	5.0
ged-14	20.0	20.0	20.0	20.0
hiking-14	20.0	20.0	20.0	20.0
nomystery-11	11.0	17.4	10.0	15.0
openstacks-14	20.0	20.0	20.0	19.4
parcprinter-11	20.0	20.0	20.0	20.0
parking-11	20.0	20.0	20.0	20.0
pegsol-11	20.0	20.0	20.0	20.0
scanalyzer-11	20.0	20.0	20.0	20.0
sokoban-11	19.0	18.0	19.0	17.0
tetris-14	18.0	14.0	19.6	16.4
thoughtful-14	15.0	17.0	15.0	18.0
tidybot-11	16.0	17.4	16.0	17.2
transport-11	18.0	17.6	17.0	18.2
visitall-14	20.0	20.0	20.0	20.0
woodworking-11	20.0	20.0	20.0	1.0
agricola-18	10.0	12.0	11.8	12.4
data-network-18	12	14.4	15.8	10.6
organic-synthesis-18	3.0	3.0	3.0	3.0
organic-synth-split-18	10.0	11.6	10.0	11.2
snake-18	4.0	5.6	4.8	4.4
spider-18	16.0	16.2	16.0	16.0
termes-18	14.0	14.0	14.0	14.0
Total Solved:	393.0	**407.2**	401.2	385.8

Table 3: Average Coverage across domains from IPC 2011, 2014, and 2018. Type, Soft and HI_U^D abbreviate Type-LAMA, Softmin-LAMA and HI_U^D-LAMA.

7 Conclusion

In this paper, we observed that the BTS may serve as a suitable basis for constructing type systems for stochastic exploration. In noting that benches in the BTS cannot be used during search, we introduced the more general notion of subspaces and used Heuristic Improvement and Low Water-Mark to approximate the BTS in a search friendly way. These BTS approximations were then used to build BTS-approximate type systems that were empirically shown to effectively enhance standard GBFS.

We conducted no parameter tuning in our empirical evaluation — this is left as future work. The LAMA planner was only tested with a single BTS-based configuration with lackluster results; it may benefit more from a different type system or different selection policies. The actual BTS as a type system may be tested in small problems for which it can be calculated; this we leave as future work. More generally, BTS-approximate type systems suggest a family of *path dependent* type systems. These systems, and their underlying type trees, may be useful for other kinds of selection policy approaches such as that in Monte Carlo Tree Search, or they may be useful to inform load balancing in parallel GBFS.

Acknowledgements

We thank the anonymous reviewers for their helpful feedback on this work. We also gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Artificial Intelligence Group - University of Basel. 2023. Fast Downward. https://github.com/aibasel/downward.

Asai, M.; and Fukunaga, A. 2017. Exploration among and within plateaus in greedy best-first search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, 11–19.

Cohen, E.; and Beck, J. C. 2018. Local Minima, Heavy Tails, and Search Effort for GBFS. In *IJCAI*, 4708–4714.

Doran, J. E.; and Michie, D. 1966. Experiments with the Graph Traverser Program. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 294(1437): 235–259.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Heusner, M.; Keller, T.; and Helmert, M. 2017. Understanding the search behaviour of greedy best-first search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 8, 47–55.

Heusner, M.; Keller, T.; and Helmert, M. 2018a. Best-case and worst-case behavior of greedy best-first search. International Joint Conferences on Artificial Intelligence.

Heusner, M.; Keller, T.; and Helmert, M. 2018b. Search progress and potentially expanded states in greedy best-first search. International Joint Conferences on Artificial Intelligence.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Kuroiwa, R.; and Beck, J. C. 2022. Biased Exploration for Satisficing Heuristic Search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 213–221.

Kuroiwa, R.; and Fukunaga, A. 2019. On the pathological search behavior of distributed greedy best-first search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 255–263.

Kuroiwa, R.; and Fukunaga, A. 2020. Analyzing and avoiding pathological behavior in parallel best-first search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 175–183.

Lipovetzky, N.; and Geffner, H. 2017. Best-first width search: Exploration and exploitation in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Richter, S.; and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Röger, G.; and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 20, 246–249.

Shimoda, T.; and Fukunaga, A. 2023. Improved Exploration of the Bench Transition System in Parallel Greedy Best First Search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 16, 74–82.

Valenzano, R.; Sturtevant, N.; Schaeffer, J.; and Xie, F. 2014. A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, 375–379.

Valenzano, R.; and Xie, F. 2016. On the completeness of best-first search variants that use random exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Xie, F.; Müller, M.; and Holte, R. 2014. Adding local exploration to greedy best-first search in satisficing planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.