Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Search Portfolio with Sharing

Sandip Aine
Indraprastha Institute of Information Technology, New Delhi, India
sandip@iiitd.ac.in

Maxim Likhachev
Carnegie Mellon University
Pittsburgh, PA, USA
maxim@cs.cmu.edu

Abstract

Over the years, a number of search algorithms have been proposed in AI literature, ranging from best-first to depth-first searches, from incomplete to optimal searches, from linear memory to unbounded memory searches; each having their strengths and weaknesses. The variability in performance of these algorithms makes algorithm selection a hard problem, especially for performance critical domains. Algorithm portfolios alleviate this problem by simultaneously running multiple algorithms to solve a given problem instance, exploiting their diversity. In general, the portfolio methods do not share information among candidate algorithms. Our work is based on the observation that if the algorithms within a portfolio can share information, it may significantly enhance the performance, as one algorithm can now utilize partial results computed by other algorithms. To this end, we introduce a new search framework, called Search Portfolio with Sharing (SP-S), which uses multiple algorithms to explore a given state-space in an integrated manner, seamlessly combining the partial solutions, while preserving the constraints/characteristics of the candidate algorithms. In addition, SP-S can be easily adopted to guarantee theoretical properties like completeness, bounded sub-optimality, and bounded re-expansions. We describe the basics of the SP-S framework and explain how different classes of search algorithms can be integrated in SP-S. We discuss its theoretical properties and present experimental results for multiple domains, demonstrating the utility of such a shared approach.

Introduction

State space search is a widely used problem solving approach in AI with applications in classical planning (Bonet and Geffner 2001), robotics (Likhachev and Ferguson 2009), learning (Yuan and Malone 2013), bio-informatics (Keedwell and Narayanan 2005), and many other domains. Consequently, a significant amount of research effort has been put into designing efficient search algorithms, resulting in a large repository of techniques. The most well-known search algorithm is the A* (Hart, Nilsson, and Raphael 1968) algorithm, which is a best-first search that uses domain specific cost-to-goal estimates (heuristics) to guide the search-space exploration. If the heuristic is *admissible* (i.e., it provides a lower bound on the true cost-to-goal), A* is provably optimal, both in terms of the solution quality and the number of state expansions.

Unfortunately, A* often requires exponential run time and memory, making it inapplicable for large-scale problems. This motivated the development of other search methods, algorithms that improve A*'s memory footprint (Chakrabarti et al. 1989; Korf 1985), and more importantly, algorithms that attempt to attain an effective trade-off between the solution cost and run time, enabling users to solve large scale complex problems within reasonable time limits. Several such trade-off based algorithms are based on weighted A* (WA*) (Pohl 1970), where the heuristic function is inflated by a constant w ($w > 1.0$, to give the search a goal-directed focus. Other approaches include beam search (Lowerre 1976) and variants (Zhou and Hansen 2005; Vadlamudi, Aine, and Chakrabarti 2013), window A* (Aine, Chakrabarti, and Kumar 2007; Vadlamudi, Aine, and Chakrabarti 2011), algorithms that inadmissibly restrict the exploration space (using parameters like beam widths or window sizes); A*$_\epsilon$ (Pearl and Kim 1982), EES (Thayer and Ruml 2011), algorithms that use inadmissible priorities to order state expansions, but control the inadmissible expansions using admissible bounds; greedy and speedy searches (Wilt and Ruml 2014), algorithms that order state expansions using cost-to-goal or distance-to-goal estimates only; etc.

Therefore, we observe that a given planning problem can possibly be solved using multiple algorithms. Often it is *hard* to decide which particular algorithm to use, especially when we are looking for fast and accurate solutions for large scale problems, as the performance of a search algorithm critically depends on the search space characteristics, which are not easy to predict. For example, WA* approaches work very well for many domains (Zhou and Hansen 2002; Likhachev, Gordon, and Thrun 2004), but they rely heavily on the heuristic accuracy, and subsequently, perform poorly in the presence of large local minima (Wilt and Ruml 2012). In contrast, beam search can be very useful for solving large scale problems with relatively weak heuristics (Thayer and Ruml 2010), since it does not suffer from local minima as much as WA* does. However, it does not guarantee completeness and has no control over the solution quality.

Algorithm portfolios (Gomes and Selman 2001; Valenzano et al. 2010; Helmert et al. 2011; Gerevini, Saetti, and Vallati 2009) attempt to overcome this selection problem by using multiple algorithms to solve a given instance. If any of the candidate searches finds a solution, the portfolio returns a success. These approaches have been very successful in solving challenging planning/optimization problems, especially for domains where it is hard to find a single best algorithm.

Most portfolio based planners consider the candidate algorithms as separate entities (black boxes) that do not share information. Our work is motivated by the observation that majority of the search algorithms compute a solution (path from start to goal) by iteratively discovering paths to intermediates states, following a Dynamic Programming formulation. Such intermediate paths are generally qualified using the path cost only, independently of which algorithm actually computed the path. This means that the candidate algorithms can share these intermediate paths and leverage different algorithms' strengths not only when solving *different* instances (as done in portfolios), but also when solving a *single* instance. Furthermore, sharing intermediate paths may help in limiting repeated works, as one algorithm can now reuse the partial solutions computed by another algorithm, and does not need to recompute them.

We present an algorithmic framework, called Search Portfolio with Sharing (SP-S), that integrates multiple search algorithms within a single flow, enabling maximal sharing of states/paths among the candidate algorithms. This makes SP-S more powerful than a collection of algorithms, as it can use partial solutions from different algorithms to solve an instance which is not solvable (within resource constraints) by any candidate algorithm on its own. In addition, SP-S uses an (bounded) admissible algorithm to *anchor* the state expansions, so that it can guarantee completeness and bounded sub-optimality, limited re-expansions, and other efficiency properties.

It may be noted that recent works on multi-heuristic searches (Röger and Helmert 2010; Aine et al. 2014; Phillips et al. 2015) have demonstrated the benefits of a multi-pronged search with sharing and anchor based control. In a way, SP-S can be seen as an extension to the Multi-heuristic A* (Aine et al. 2014) framework for search algorithm portfolios. However, integrating multiple search algorithms within a unified flow brings forth a new set of challenges, as the individual algorithms can have different exploration characteristics and constraints. In SP-S, we address this by adapting the sharing (and control) approach for different categories of search algorithms in a fashion that maximizes the sharing of partial solutions while maintaining the individual characteristics and constraints.

The rest of the paper is organized as follows. We start with a high level description of SP-S and state some of its desirable properties. Next, we discuss different categories of search algorithms and describe how these algorithms may fit into SP-S. Then, we explain the SP-S framework in detail and describe its theoretical properties. Finally, we present experimental results evaluating an example instantiation of SP-S for challenging problem domains such as large sliding tile puzzles (unit and square cost), 6D robot arm planning, and 3D navigation.

Related Work

Algorithm portfolios

For many domains, there is no single algorithmic approach (and/or heuristic) that consistently provides the best performance. Algorithm portfolios leverage the strengths of diverse approaches by running multiple algorithms to solve a given problem. A portfolio may include different algorithms or heuristics (Gomes and Selman 2001; Helmert et al. 2011) or it may have a single algorithm but run it with different parameters (Xu, Hoos, and Leyton-Brown 2010; Valenzano et al. 2010), either sequentially or in parallel (Valenzano et al. 2012). Studies on multiple planning and optimization domain have demonstrated the efficacy of such ensemble planning, when compared to the single best algorithms (Helmert et al. 2011; Xu et al. 2008; OMahony et al. 2008). The utility of portfolios has inspired a considerable amount of research on algorithm selection and scheduling, within a given portfolio (Kotthoff 2014; Hoos et al. 2015; Kotthoff, Gent, and Miguel 2012; Malitsky and O'Sullivan 2014).

The fundamental difference between the algorithm portfolios and SP-S stems from the fact that the portfolios consider the candidate algorithms as separate entities with no (or little) communication. Although, some of the formative works on the portfolio approach (Huberman, Lukose, and Hogg 1997) discussed the possibility (and value) of cooperation among the portfolio algorithms, in practice most portfolios do not share information or they do it at a very high level (Helmert et al. 2011). Some notable exceptions are found in portfolio based SAT solvers (Hamadi, Jabbour, and Sais 2008; Malitsky et al. 2013), where sharing is more common. Unfortunately, most such techniques are domain dependent and cannot be easily generalized for classical search algorithms (which are usually domain independent). In contrast, SP-S combines multiple algorithms within a single search framework, with maximal information sharing and anchor based control. This makes SP-S more robust than portfolios in solving hard problems and enables it to guarantee a number of theoretical properties that are not generally supported by the portfolio methods.

Multi-heuristic search

Another set of algorithms that closely relates to SP-S are the multi-heuristic search algorithms (Röger and Helmert 2010; Aine et al. 2014). These algorithms utilize multiple heuristics independently to explore a search space and thus, are more powerful in solving challenging planning problems where a single heuristic may easily get stuck in a local minimum.

While the philosophy of SP-S is close to the multi-heuristic searches, especially MHA* (Aine et al. 2014), which supports similar theoretical properties, their primary difference stems from the fact that MHA* is single search algorithm with multiple heuristic functions whereas SP-S is a set of different search algorithms that are used to explore a search space in a collaborative manner. Also, to use multi-heuristic searches a user needs to provide a set of diverse (yet informative) heuristics which is a non-trivial challenge for most domains (Aine, Sharma, and Likhachev 2015). SP-S does not require such additional information and can work with a set of known algorithms, and yet benefit from multi-pronged explorations (with sharing), while preserving equivalent theoretical guarantees.

Search Portfolio with Sharing (SP-S)

Notation: In the following, S denotes the finite set of states of a given problem. s_{start} denotes the start state and s_{goal} denotes the goal state. $c(s, s')$ denotes the cost of the edge between s and s', and if there is no such edge, then $c(s, s') = \infty$. The successor function $\textsc{Succ}(s) := \{s' \in S | c(s, s') \neq \infty\}$, denotes the set of all reachable successors of s. An optimal path from s_{start} to s has cost $g^*(s)$. $g(s)$ denotes the current best path cost from s_{start} to s. $h(s)$ denotes the heuristic for s, typically an estimate of the best path cost from s to s_{goal}. A heuristic is *admissible* if it never overestimates the best path cost to s_{goal} and *consistent* if it satisfies, $h(s_{goal}) = 0$ and $h(s) \leq h(s') + c(s, s')$, $\forall s, s'$ such that $s' \in \textsc{Succ}(s)$ and $s \neq s_{goal}$.

Overview

SP-S builds on the observation that many graph searches are based on the Dynamic Programming principle, where the problem of computing a path from s_{start} to s_{goal} can be decomposed into independent sub-problems of computing a path from s_{start} to s and a path from s to s_{goal}. Thus, there is no reason why a path from s_{start} to s cannot be computed by one algorithm, whereas the path from s to s_{goal} is computed by another algorithm. Following this principle, we design SP-S with a set of diverse search algorithms, members of which are simultaneously used to explore a search space (independently). If a *new/better* path to a state is discovered by any of the algorithms, it is shared with the other searches (whenever possible). The architecture of SP-S follows the MHA* model (Aine et al. 2014), where one search is used as an anchor search (a search that controls the others expansions) and the other searches act as auxiliary searches.

We start by outlining a set of desirable properties in SP-S. Later, we will use these properties to make the design choices. Following are the properties we would like to satisfy with SP-S,

- **Completeness and bounded sub-optimality:** SP-S should be provably complete and bounded sub-optimal, i.e., for a user defined constant w ($w \geq 1.0$), the solution returned by SP-S should have cost $\leq w.g^*(s_{goal})$.
- **Avoiding repeated/spurious work:** SP-S should try to minimize the work done (expansions) across all the searches. In particular, it should limit state re-expansions, and also avoid expanding states that cannot produce a solution within the chosen bound.
- **Individual search constraints/characteristics:** SP-S should not violate the constraints (for example memory requirements) of the candidate algorithms. In addition, SP-S should ensure that individual searches maintain their basic characteristics (state selection and pruning).

Search algorithm	DFS	BFS	UCS	A*	WA*	IDA*	GS	SS	BS	WS
(Bounded) admissibility	No	No	Yes	Yes	Yes	No	No	No	No	No
Best-first order	No	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Frontier restriction	No	No	No	No	No	Yes	No	No	Yes	Yes
SP-S category	A	A	B	B	B	A	C	C	D	D

Table 1: Classifying search algorithms in terms of admissibility, ordering and frontier restriction. Legend: DFS - depth-first search, BFS - breadth-first search, UCS - uniform cost search, IDA* - iterative deepening A*, GS - greedy search, SS - speedy search, BS - beam search, WS - window A*.

Next, we discuss a categorization of some of the widely used search algorithms with respect to certain properties, that will help us decide how these algorithms can be integrated in SP-S to support its desirable properties. Please note that these categories are not meant to be exhaustive, they only highlight certain characteristics of different search algorithms that are critical for SP-S. Following are the properties we consider for categorizing searches in SP-S,

- **(Bounded) admissibility:** A search algorithm is called (bounded) admissible (in SP-S), if it guarantees that when a state s is selected for expansion, $g(s) \leq w \cdot g^*(s)$, where w (≥ 1.0) is a chosen constant. All other algorithms are termed inadmissible. For example, uniform cost search (UCS), A* and WA* with a consistent heuristic function are (bounded) admissible searches, whereas greedy search, A*$_\epsilon$, beam search are not.

  Note that there are several algorithms which returns a provably bounded sub-optimal solutions when arbitrary state re-expansions are allowed (for example A*$_\epsilon$, EES). However, in SP-S we do not consider them as (bounded) admissible searches as we intend to restrict the number of state expansions. For this reason, we use a stricter rule ($g(s) \leq w \cdot g^*(s)$) to decide whether an algorithm is considered (bounded) admissible or not.
- **Best-first order:** Best-first searches expand states from a prioritized list, always selecting the minimum priority state for expansion (for minimization problems). A*, WA*, greedy search are some examples of best-first searches, while depth-first (DFS) and breadth-first (BFS) searches are examples of algorithms that does not expand states in a best-first manner [1]. Note that best-first search does not imply admissibility (for example greedy search).
- **Restriction of search frontier:** Unrestricted searches only impose an ordering to select states to store and expand (either admissible or inadmissible), however they do not limit the search frontier in any way. In contrast, some search algorithms (such as beam search) use external parameters (for example, beam-width in case of beam search) to restrict/prune the search frontier. In SP-S, we classify them as restricted searches. Typically, a restricted

[1]Note that both DFS and BFS can be run as a best-first search by using depth as a priority (maximum depth in case of DFS and minimum depth in case of BFS). However, here we consider the classic stack/queue based implementations of DFS/BFS that do not require prioritized lists.

search is also an inadmissible search (according to the above definition).

In Table 1, we list some well known search algorithms and show how they differ in terms of these properties. In SP-S, we partition these algorithms into 4 categories, category A for search algorithms that do not follow best-first order (DFS, BFS, IDA*), category B for (bounded) admissible algorithms (UCS, A*, WA*), category C for best-first, inadmissible and unrestricted algorithms (greedy search, speedy search), and category D for best-first restricted searches (beam search, window A*).

Algorithm

Algorithm 1 SP-S: 1 Anchor + n Auxiliary Searches

Inputs: s_{start}, s_{goal}, Sub-optimality bound w (≥ 1)
Output: A path from s_{start} to s_{goal} with cost $\leq w \cdot g^*(s_{goal})$

```
1: procedure ExpandState($s, i$)
2:     Remove $s$ from all the search frontiers
3:     if Admissible($i$) then
4:         CLOSED$_a$ ← CLOSED$_a \cup s$
5:     else
6:         CLOSED$_i$ ← CLOSED$_i \cup s$
7:     for all $s' \in$ Succ($s$) do
8:         if $s'$ was not seen before then
9:             $g(s') \leftarrow \infty$
10:        if $g(s') > g(s) + c(s, s')$ then
11:            $g(s') \leftarrow g(s) + c(s, s')$
12:            if $s' \notin$ CLOSED$_a$ then
13:                for $i' \in \{0, \ldots, n\} \bigwedge$ Admissible($i'$) do
14:                    UpdateFrontier($s', i'$)
15:                if $s' \notin$ CLOSED$_i$ then
16:                    if ¬Admissible($i$) then
17:                        UpdateFrontier($s', i$)
18:                    for $i' \in \{1, \ldots, n\} \bigwedge i' \neq i$ do
19:                        if ¬Admissible($i'$) then
20:                            if ShareWithAux($s', i'$) then
21:                                UpdateFrontier($s', i'$)
22: procedure InitializeSearches($s_{start}$)
23:     $g(s_{start}) \leftarrow 0$
24:     for $i \in \{0, \ldots, n\}$ do
25:         Insert $s_{start}$ in $i^{th}$ search frontier
26: procedure Main
27:     CLOSED$_a$ ← $\emptyset$, CLOSED$_i$ ← $\emptyset$
28:     InitializeSearches($s_{start}$)
29:     while ¬TerminationCriteria($s_{goal}$) do      ▷ Defined by anchor
30:         for $i \in \{1, \ldots, n\}$ do
31:             $s$ ← PickState($i$)  ▷ Defined by $i^{th}$ Aux. Search
32:             if WithinBound($s$) then      ▷ Defined by anchor
33:                 ExpandState($s, i$)        ▷ Aux. expansion
34:             if TerminatedAux($i$) then
35:                 ReInitializeAux($i$)          ▷ Optional
36:         $s_a$ ← PickState(0)            ▷ Defined by anchor
37:         ExpandState($s_a$, 0)           ▷ Anchor expansion
38:     return solution path
```

A high level pseudocode of SP-S is included in Algorithm 1. SP-S explores the search space using the candidate algorithms in a round-robin fashion till it gets to the solution within the chosen bound or proves that such a solution does not exist. Note that, in Algorithm 1 we use a common EXPANDSTATE function for every algorithm in SP-S, which points to the assumption that the searches in SP-S must support the notion of state *expansion* (the cost update part in lines 7-11, generally applicable for all Dynamic Programming based searches). On the other hand, the searches may differ in state selection, ordering and pruning (PICKSTATE, UPDATEFRONTIER, SHAREWITHAUX functions, which are defined by the candidate algorithms).

We use index 0 to denote the anchor search, other searches are indexed from $1 \ldots n$. During a round, SP-S always expands from the anchor. However, for others, the expansions are controlled using the WithinBound function which ensures a selected state is only expanded if its cost ($g + h$) is within w-times of the optimal solution cost.

SP-S uses two CLOSED lists to distinguish between inadmissible and (bounded) admissible state expansions (admissible state expansion ensures $g(s) \leq w \cdot g^*(s)$). If a state is expanded in an admissible search, SP-S does not re-expand it. However if a state is expanded in an inadmissible search, it may get re-expanded later in an admissible search, if a better quality path to it is discovered.

Next, we discuss certain design aspects of SP-S showing how it accommodates algorithms from different categories (Table 1).

- **Anchor and auxiliary searches:** SP-S requires instantiation of an anchor search and a set of auxiliary searches. The choice of anchor is crucial as it defines the termination criteria (TERMINATIONCRITERIA function). For SP-S to be bounded sub-optimal, the anchor needs to be a (bounded) admissible algorithm (category B), as otherwise SP-S cannot guarantee bounds on solution quality. Furthermore, an admissible anchor can provide a lower bound on the solution cost at any intermediate stage, which can be used to control the expansions in auxiliary searches (WITHINBOUND function). In contrast, the auxiliary searches in SP-S can be completely arbitrary, as long as they support the notion of state expansions. However, the choice of auxiliary searches determines the sharing of states/paths, which we discuss next.

- **Path sharing:** Path sharing (g values and successor states) among searches is an important feature of SP-S which enables it to combine partial solutions from different algorithms and to limit repeated works. However, such sharing needs to be regulated, so that SP-S does not violate the constraints of an auxiliary search, nor does it alter its state expansion characteristics. Please note that whether a state (path) can be shared depends entirely on the properties of the receiver search, and not on the search that discovers the state/path.

  In SP-S, we use the following three rules to define the sharing policy. (i) Any state (path) can be shared with searches in category B and C (best-first order, unrestricted frontier). (ii) Whether a state (path) can be shared with a search in category D depends on the constraints of that search and needs to be decided dynamically. For example, for beam search with beam-width k, SP-S can only insert k states (per level) in its frontier, without violating the memory constraints. (iii) We cannot share states with searches in category A (non best-first) as these searches do not use priorities, and thus, inclusion of states from other searches may affect their characteristics. For example, in DFS (stack) or BFS (queue), the expansion order will break, if states from other searches are inserted in their frontier. In Algorithm 1, the SHAREWITHAUX function defines whether a state can be shared with an auxiliary search. Note that, the anchor (and *admissible* auxiliary searches) belong to category B, thus, any state can be shared with them.

- **Re-initialization:** As noted earlier, SP-S does not share arbitrary states with auxiliary searches in category A or D. Therefore, it may happen that such an auxiliary search gets terminated early as it has no states to expand (empty frontier/all states in the frontier violate bounds). In that case, SP-S may re-initialize the search using states from the anchor (REINITIALIZEAUX function). Note that this is optional. For searches in category B and C, re-initialization is generally not applicable. However, if we do not want to share all the states up-front (for category C), re-initialization can be used as a form of *lazy* sharing.

	Role		Functions		
Category	Anchor	Auxiliary	ADMISSIBLE	SHAREWITHAUX	REINITIALIZEAUX
A	No	Yes	No	No	Yes
B	Yes	Yes	Yes	Yes	N/A
C	No	Yes	No	Yes	Yes
D	No	Yes	No	Restricted	Yes

Table 2: Roles and functions for searches from different categories (in SP-S). Legend: N/A - not applicable.

In Table 2, we list how searches from different categories can be integrated in SP-S in terms of (possible) roles and functions (as described in Algorithm 1).

Algorithm 2 SP-S Anchor: Termination and Pruning

```
1: Assumptions: The anchor search is $w_1$-admissible. It stores
   the states in a prioritized list OPEN.
2: procedure TERMINATIONCRITERIA($s_{goal}$)
3:     if OPEN.EMPTY() then
4:         return TRUE                              ▷ Search fails
5:     $w_2 = w/w_1$
6:     if $g(s_{goal}) \leq w_2 \cdot$ OPEN.MINKEY() then
7:         return TRUE                              ▷ Search succeeds
8:     return FALSE                                 ▷ Continue
9: procedure WITHINBOUND($s$)
10:    $w_2 = w/w_1$
11:    if $s =$ null $\vee\, g(s) + h(s) > w_2 \cdot$ OPEN.MINKEY() then
12:        return FALSE
13:    return TRUE
```

Analytical Properties

The theoretical properties of SP-S rely on the anchor search which defines two key functions, TERMINATIONCRITERIA and WITHINBOUND. In Algorithm 2, we include an instantiation of these functions assuming that the anchor is a w_1-admissible search ($w \geq w_1 \geq 1.0$, where w is the user defined sub-optimality bound for SP-S), such as consistent heuristic WA* with w_1 as the weighing factor. In the following three theorems, we use this instantiation to derive the analytical results for SP-S [2].

Theorem 1 (Bounded sub-optimality)**.** *SP-S is complete and bounded sub-optimal, i.e, SP-S is guaranteed to terminate and when it does, $g(s_{goal}) \leq w \cdot g^*(s_{goal})$.*

Proof. The w-admissibility of SP-S stems from the fact that the anchor is always a (bounded) admissible search. Borrowing from (Aine et al. 2014), we can show that if anchor is w_1-admissible, then OPEN.MINKEY() $\leq w_1 \cdot g^*(s_{goal})$, and if $g^*(s_{goal}) \neq \infty$, OPEN can never be empty before termination. Which in turn ensures that if $g(s_{goal}) \leq w_2 \cdot$ OPEN.MINKEY(). It follows that $g(s_{goal}) \leq w_1 \cdot w_2 \cdot g^*(s_{goal}) = w \cdot g^*(s_{goal})$. On the other hand if OPEN is empty, then $g^*(s_{goal}) = \infty$, i.e., there is no finite cost solution. ■

Theorem 2 (Bounded re-expansions)**.** *No state is expanded more than twice in SP-S. Also, a state expanded in an inadmissible (auxiliary search) can only be re-expanded in the anchor if its g value is lowered since the last expansion.*

Proof. If a state is expanded in an admissible search, it is removed from all frontiers and put into CLOSED$_a$. Such a state is never put back to any search frontier (check at line 12, Algorithm 1), and thus can never be re-expanded. If a state is expanded in an inadmissible (auxiliary) search, it is removed from all the frontiers and stored in CLOSED$_i$. Now, the check at line 15, Algorithm 1 ensures that it can only be put into an admissible frontier, that too if only its g-value gets lowered (line 10, Algorithm 1). Therefore, a state can at most be expanded twice. ■

Theorem 3 (Efficiency)**.** *In SP-S, any state s with $g(s) + h(s) > w \cdot g^*(s_{goal})$ will never be expanded.*

Proof. This is ensured by the WITHINBOUND function. As, OPEN.MINKEY() $\leq w_1 \cdot g^*(s_{goal})$, and SP-S only expands a state s if $g(s) + h(s) \leq w_2 \cdot$ OPEN.MINKEY(), it follows that a state s with $g(s) + h(s) > w \cdot g^*(s_{goal})$ is never expanded. ■

Scheduling

In Algorithm 1, SP-S explores the candidate searches in a round-robin fashion. Such a uniform strategy may not be an efficient one if we have many algorithms in the portfolio and only few of them perform well for a particular instance. Algorithm portfolios generally use offline training to profile the performance of candidate algorithms for a given domain,

[2]Note that, if the anchor is w_1-admissible, an auxiliary search from category B is considered admissible, only if it guarantees w_1'-admissibility, where $w_1' \leq w_1$. Otherwise, it is used as an inadmissible search.

Algorithm 3 SP-S with DTS

Inputs: s_{start}, s_{goal}, Sub-optimality bound w (≥ 1)
Output: A path from s_{start} to s_{goal} with cost $\leq w \cdot g^*(s_{goal})$

```
1:  procedure InitializeSchedule($s_{start}$)
2:      $h_{min} \leftarrow h(s_{start})$
3:      for $i \in \{0, \dots, n\}$ do
4:          $\alpha[i] \leftarrow 1; \beta[i] \leftarrow 1$
5:  procedure ChooseSearch
6:      $r[0] \sim Beta(\alpha[0], \beta[0])$
7:      for $i \in \{1, \dots, n\}$ do
8:          $s \leftarrow$ PickState($i$)
9:          if WithinBound($s$) then
10:             $r[i] \sim Beta(\alpha[i], \beta[i])$
11:         else
12:             $r[i] \leftarrow -\infty$
13:             if TerminatedAux($i$) then
14:                 ReInitializeAux($i$)
15:     return $\arg\max_i r[i]$
16: procedure UpdateSchedule($i$)
17:     $h_{cur} \leftarrow min(h(s), \forall s \in$ OPEN$)$
18:     if $h_{cur} < h_{min}$ then
19:         $h_{min} = h_{cur}$
20:         $\alpha[i] \leftarrow \alpha[i] + 1$                    ▷ Get reward 1
21:     else
22:         $\beta[i] \leftarrow \beta[i] + 1$                      ▷ Get reward 0
23:     if $\alpha[i] + \beta[i] > C$ then
24:         $\alpha[i] \leftarrow \frac{C}{C+1}\alpha[i]; \beta[i] \leftarrow \frac{C}{C+1}\beta[i]$
25: procedure Main()
26:     CLOSED$_a \leftarrow \emptyset$, CLOSED$_i \leftarrow \emptyset$
27:     InitializeSearches($s_{start}$)
28:     InitializeSchedule($s_{start}$)
29:     while $\neg$TerminationCriteria($s_{goal}$) do
30:         $i \leftarrow$ ChooseSearch()
31:         $s \leftarrow$ PickState($i$)
32:         ExpandState($s, i$)
33:         UpdateSchedule($i$)
34:     return solution path
```

and use such profiles to select/order the candidates for sequential exploration (Helmert et al. 2011). These methods are not directly applicable for SP-S, as SP-S allows sharing of states (which means we cannot create profiles by running the candidate algorithms individually and need to consider all possible combination of algorithms).

Instead, in SP-S we adopt the Dynamic Thompson Sampling (DTS) based approach described in (Phillips et al. 2015), to schedule the searches online without using any prior knowledge. DTS scheduling is simple. It models the search selection problem as a multi-armed bandit problem and always expands from the search which is expected to "progress" quickly. In SP-S, we store the minimum heuristic value of the states in the anchor frontier in parameter h_{min}, i.e., $h_{min} = min(h(s), \forall s \in$ OPEN$)$. If an expansion from a search reduces this h_{min}, it gets a reward 1, otherwise it gets a reward 0. This way, the searches that help SP-S progress toward goal get selected more often, as compared to searches that do not. However, when a search gets stuck (say at a local minima), DTS quickly adjusts its distribution and other candidates are preferred.

We include the pseudocode for SP-S with DTS in Algorithm 3. The changes introduced are quite simple. Instead of exploring all the searches in round-robin manner, SP-S with DTS selects a particular search in each iteration, using the ChooseSearch routine (line 30) following DTS. Once a search is selected, it expands from that search and updates its distribution depending on whether the minimum h value of the anchor frontier is reduced or not (UpdateSchedule routine, line 33). The parameter C is used to control the temporal history of the scheduler, i.e., it determines allowed range of the $\alpha+\beta$ values. If $\alpha+\beta$ exceeds C, it is normalized to C.

Note that the ChooseSearch routine only considers an auxiliary search for selection if it satisfies the bound check (line 9), on the other hand, the anchor search is always considered. This ensures that the analytical properties of SP-S with DTS remain identical to that of SP-S with round-robin scheduling, i.e., the Theorems described in the previous section are equally applicable for SP-S with DTS.

Experimental Results

We built a search portfolio using the following algorithms, WA* (anchor search), greedy search (GS), beam search (BS) and window A* (WS). We compared SP-S with the individual algorithms of the portfolio, portfolio of the same algorithms without sharing (SP-wS), and EES (Thayer and Ruml 2011), which is a composite bounded sub-optimal search algorithm. In all the experiments, we used a sub-optimality bound of 5.0 for the bounded sub-optimal searches (WA*, EES, SP-S) [3]. We ran beam search with beam-width 300 and window A* with window-size 5. In SP-S, we dynamically restricted the state sharing for beam search and window A*, according to the beam and window-size constraints. For the portfolios (SP-wS and SP-S) we investigated two scheduling strategies, namely round-robin (RR) and DTS. We evaluated SP-S for the following domains.

Sliding Tile Puzzle (Unit Cost)

For this domain, we benchmarked the algorithms on randomly generated test-suites of 100 8×8, 9×9 and 10×10 puzzles (all solvable). We used the Manhattan distance plus linear conflicts as a consistent heuristic. Each planner was given a time limit of 180 seconds for solving a given instance.

We include the results for this domain in Table 3. From the table, we observe that among the individual algorithms beam search performs the best, however the other algorithms also contribute in coverage by solving different instances. For the portfolio methods, first thing we note that the DTS scheduling generally outperforms round-robin scheduling (for both SP-wS and SP-S), highlighting the importance of dynamic selection of algorithms. Finally, the results clearly show

[3]Note that, we used SP-wS without any information sharing among the algorithms (neither the states nor the bounds), thus SP-wS was not a bounded sub-optimal algorithm as the portfolio had algorithms like greedy search, beam search and window A*, none of which are bounded sub-optimal.

		Indiv. Algo.						SP-wS		SP-S	
Size		WA*	GS	BS	WS	∪	EES	RR	DTS	RR	DTS
8×8	IS	52	15	73	61	92	38	88	89	94	98
	RT	31.31	41.93	14.42	31.32	-	51.04	52.46	44.59	27.97	22.76
	SC	794	1451	712	734	-	1145	771	791	776	783
9×9	IS	25	7	48	46	78	19	60	66	73	82
	RT	63.42	63.83	21.19	38.78	-	114.60	126.08	73.51	47.70	51.27
	SC	1039	2157	983	1007	-	1453	983	1004	987	990
10×10	IS	7	2	22	18	38	5	14	19	35	48
	RT	36.67	174.00	30.22	40.35	-	121.28	116.92	95.15	81.04	64.62
	SC	1130	1719	1234	1155	-	1583	1059	1055	1119	1174

Table 3: Comparison of different algorithms and portfolios for unit cost sliding tile puzzles (time limit 180 seconds). Legend: IS - instances solved (out of 100), RT - average run time, SC - average solution cost. Column marked by ∪ denotes the total number of unique instances solved by any of the individual algorithm.

		Indiv. Algo.						SP-wS		SP-S	
Size		WA*	GS	BS	WS	∪	EES	RR	DTS	RR	DTS
8×8	IS	2	19	66	73	90	12	81	84	95	95
	RT	16.00	28.11	11.06	16.39	-	43.92	31.69	27.28	24.70	20.08
	SC	3666	6917	3368	3838	-	4403	3619	3731	3709	3732
9×9	IS	1	5	39	55	72	8	62	67	77	84
	RT	59.23	69.75	17.24	24.85	-	29.72	84.67	70.55	46.77	49.02
	SC	5886	10811	5529	5661	-	7138	5428	5396	5660	5536
10×10	IS	0	5	30	32	51	2	39	42	61	61
	RT	-	84.50	19.69	29.00	-	134.82	111.54	101.38	119.03	95.15
	SC	-	13311	7190	7569	-	10782	7227	7338	7455	7502

Table 4: Comparison of different algorithms and portfolios for square cost sliding tile puzzle problems (time limit 180 seconds). Note that, here GS, BS and WS use distance based priorities. Legend: IS - instances solved (out of 100), RT - average run time, SC - average solution cost.

the superiority of SP-S over SP-wS (and the others). Not only it provides better coverage/trade-off compared to all the other approaches, in most cases (especially with DTS), SP-S solves more instances than the union of all portfolio algorithms. Note that, the selection and scheduling techniques for portfolio methods attempt to get as close as possible to this union number (Helmert et al. 2011), as portfolios (w/o sharing) can never solve more instances. In contrast, by sharing paths, SP-S can and does solve instances that are not solved by any candidate algorithm, corroborating our claim that SP-S can be more powerful than a collection of individual algorithms. Note that, as SP-S solves more (hard) instances than the other algorithms, the average run time results are somewhat biased against it. Even then, we observe that SP-S with DTS is faster than the other portfolio solvers.

Sliding Tile Puzzle (Square Cost)

For square cost puzzles, the cost of moving a tile is the square root of the value in the tile face, i.e, unlike the classical sliding tile puzzle, here the operator costs are not uniform.

It has been noted (Wilt and Ruml 2011), that such non-uniform cost problems are generally harder to solve, compared to the uniform cost ones. In fact, as shown in Table 4, WA* performs very poorly in this domain. In SP-S (and SP-wS), we modified the inadmissible algorithms to use a distance based priority instead of the cost based ones (as suggested in (Wilt and Ruml 2011)), i.e., these searches mimic the priorities used for the unit cost domain. It may be noted that this change does not alter the sub-optimality guarantee of SP-S as the bound is derived using the anchor search (WA*) which uses a cost based priority.

Results for this experiment are included in Table 4, which show that while the individual algorithms' performances change significantly from the unit cost domain, SP-S consistently provides the best results, and in most cases is able to solve more problems than the union of all candidate algorithms. These results highlight the importance of portfolios, as a slight alteration in the domain (from unit to square cost) can cause large variation in an algorithm's performance (for example, WA*), and also show that when using portfolios, sharing of states and bounds can provide considerable improvement over running the algorithms independently.

6D Robot Arm Planning

We used a planar 6-DOF robot arm with a fixed base situated at the center of a 2D environment (100×100 grid) with obstacles. The environment is the same as described in the publicly available SBPL library (http://www.sbpl.net). The planning objective here is to compute obstacle-free path to move the end-effector from its initial configuration to the goal configuration. We evaluated the algorithms on a test-suite of 100 instances with randomly placed obstacles (we used three obstacles, one circular, one T-shaped and one rectangular) and random start-goal pairs. A consistent heuristic was computed by running a 2D Dijkstra search from goal to start on an underlined grid. We set a time limit of 60 seconds for solving each problem instance.

	Indiv. Algo.						SP-wS		SP-S	
	WA*	GS	BS	WS	∪	EES	RR	DTS	RR	DTS
IS	51	38	46	42	63	44	59	62	62	71
RT	2.76	2.02	3.52	4.36	-	2.49	2.87	1.41	0.84	0.79
SC	70.86	77.61	66.58	61.69	-	73.63	72.51	73.95	68.20	67.42

Table 5: Comparison of different algorithms and portfolios for 6D robot arm planning (time limit 60 seconds). Legend: IS - instances solved (out of 100), RT - average run time, SC - average solution cost.

We include the results of this experiment in Table 5. From the results we notice that for this domain the diversity in coverage is not much, there are only 12 instances that are not solved by WA* but solved by other algorithms, which diminishes the scope of SP-S. However, SP-S with DTS still performs best, solving more problem instances and providing considerable run time improvement over other algorithms, while computing equivalent (and sometimes better) quality solutions with the same (provable) bounds as WA*/EES.

3D Navigation

For 3D (x, y, heading) navigation, the planning objective was to compute smooth paths that satisfy the minimum turning radius constraints for a car-like robot. We generated 100 maps of size 1000×1000 with random start-goal states and obstacles. The test suite was built using a combination of maps (50 each), simulating indoor and outdoor environments (following (Aine and Likhachev 2016)). We computed a consistent heuristic by running a 16-connected 2D Dijkstra search from goal to start, after inflating the obstacles with the robot's in-radius. Each planner was given 30 seconds to solve an instance.

	Indiv. Algo.						SP-wS		SP-S	
	WA*	GS	BS	WS	∪	EES	RR	DTS	RR	DTS
IS	97	94	89	92	99	96	97	99	99	99
RT	2.12	0.98	3.88	2.42	-	2.76	3.85	1.94	1.87	0.92
SC	62.21	471.52	56.31	57.95	-	74.60	301.91	274.32	57.92	59.75

Table 6: Comparison of different algorithms and portfolios for 3D path planning (time limit 30 seconds). Legend: IS - instances solved (out of 100), RT - average run time, SC - average solution cost.

The results of this experiment are included in Table 6. From the results, we observe that coverage is not an issue here (all algorithms were able to solve most instances), whereas cost vs run time trade-off is, and in that front, SP-S (with DTS) performs better than others, leveraging the strengths of candidate algorithms while minimizing state re-expansions. However, the improvement here is not as pronounced as observed in other domains. Also note, that in this domain, the inadmissible searches (especially greedy search) can sometimes produce arbitrarily bad solutions, affecting the quality of results provided by SP-wS.

Conclusions

We have presented a search portfolio (SP-S) that integrates multiple search algorithms within a single flow, enabling maximal sharing of intermediate paths, and described how different type of search algorithms can be used in SP-S. We have shown how SP-S can be adopted to guarantee useful theoretical properties and experimentally demonstrated its efficacy for multiple domains.

In the future, we would like to explore different sharing and scheduling strategies, analyzing their impact on the performance of SP-S. We also plan to apply SP-S for other domains, such as satisficing planning. Another future direction is to extend SP-S beyond classical discrete search algorithms and explore how can we efficiently incorporate other algorithmic techniques (for example sampling based search) within the SP-S flow. We are also interested in efficient parallelization of SP-S.

Acknowledgement

This work was supported by NSF grant IIS-1409549.

References

Aine, S., and Likhachev, M. 2016. Truncated incremental search. *Artificial Intelligence*.

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and Likhachev, M. 2014. Multi-Heuristic A*. In *Proceedings of the Robotics: Science and Systems (RSS)*.

Aine, S.; Chakrabarti, P. P.; and Kumar, R. 2007. AWA* - A window constrained anytime heuristic search algorithm. In Veloso, M. M., ed., *IJCAI*, 2250–2255.

Aine, S.; Sharma, C.; and Likhachev, M. 2015. Learning to search more efficiently from experience: A multi-heuristic approach. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel.*, 141–145.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Chakrabarti, P. P.; Ghose, S.; Acharya, A.; and Sarkar, S. C. D. 1989. Heuristic search in restricted memory. *Artificial Intelligence* 41(2):197–221.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: Pbp. In *ICAPS*.

Gomes, C. P., and Selman, B. 2001. Algorithm portfolios. *Artificial Intelligence.* 126(1-2):43–62.

Hamadi, Y.; Jabbour, S.; and Sais, L. 2008. Manysat: a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation* 6:245–262.

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

Helmert, M.; Röger, G.; Seipp, J.; Karpas, E.; Hoffmann, J.; Keyder, E.; Nissim, R.; Richter, S.; and Westphal, M. 2011. Fast downward stone soup. *Seventh International Planning Competition* 38–45.

Hoos, H.; Kaminski, R.; Lindauer, M.; and Schaub, T. 2015. aspeed: Solver scheduling via answer set programming. *Theory and Practice of Logic Programming* 15(01):117–142.

Huberman, B. A.; Lukose, R. M.; and Hogg, T. 1997. An economics approach to hard computational problems. *Science* 275(5296):51–54.

Keedwell, E., and Narayanan, A. 2005. *Intelligent Bioinformatics: The Application of Artificial Intelligence Techniques to Bioinformatics Problems*. John Wiley and Sons.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Kotthoff, L.; Gent, I. P.; and Miguel, I. 2012. An evaluation of machine learning in algorithm selection for search problems. *AI Communications* 25(3):257–270.

Kotthoff, L. 2014. Algorithm selection for combinatorial search problems: A survey. *AI Magazine* 35(3):48–60.

Likhachev, M., and Ferguson, D. 2009. Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles. *I. J. Robotic Res.* 28(8):933–945.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2004. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press.

Lowerre, B. 1976. The Harpy Speech Recognition System. *PhD thesis, Carnegie Mellon University*.

Malitsky, Y., and O'Sullivan, B. 2014. Latent features for algorithm selection. In *Seventh Annual Symposium on Combinatorial Search*.

Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2013. Boosting sequential solver portfolios: Knowledge sharing and accuracy prediction. In *Learning and Intelligent Optimization*. Springer. 153–167.

OMahony, E.; Hebrard, E.; Holland, A.; Nugent, C.; and OSullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish Conference on Artificial Intelligence and Cognitive Science*, 210–216.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE Trans. Pattern Anal. Mach. Intell.* 4(4):392–399.

Phillips, M.; Narayanan, V.; Aine, S.; and Likhachev, M. 2015. Efficient search with an ensemble of heuristics. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 784–791.

Pohl, I. 1970. Heuristic Search Viewed as Path Finding in a Graph. *Artif. Intell.* 1(3):193–204.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *ICAPS*, 246–249.

Thayer, J. T., and Ruml, W. 2010. Anytime heuristic search: Frameworks and algorithms. In *Third Annual Symposium on Combinatorial Search*.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, 674–679.

Vadlamudi, S. G.; Aine, S.; and Chakrabarti, P. P. 2011. MAWA*—A Memory-Bounded Anytime Heuristic-Search Algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 41(3):725–735.

Vadlamudi, S. G.; Aine, S.; and Chakrabarti, P. P. 2013. Incremental beam search. *Inf. Process. Lett.* 113(22-24):888–893.

Valenzano, R. A.; Sturtevant, N. R.; Schaeffer, J.; Buro, K.; and Kishimoto, A. 2010. Simultaneously Searching with Multiple Settings: An Alternative to Parameter Tuning for Suboptimal Single-Agent Search Algorithms. In *ICAPS*, 177–184.

Valenzano, R.; Nakhost, H.; Müller, M.; Schaeffer, J.; and Sturtevant, N. 2012. Arvandherd: Parallel planning with a portfolio. *European Conference on Artificial Intelligence (ECAI)*.

Wilt, C. M., and Ruml, W. 2011. Cost-based heuristic search is sensitive to the ratio of operator costs. In *Proceedings of the Fourth Annual Symposium on Combinatorial Search, SOCS 2011, Castell de Cardona, Barcelona, Spain, July 15.16, 2011*.

Wilt, C. M., and Ruml, W. 2012. When does weighted A* fail? In *SOCS*. AAAI Press.

Wilt, C. M., and Ruml, W. 2014. Speedy versus greedy search. In *Seventh Annual Symposium on Combinatorial Search*.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. Satzilla: Portfolio-based algorithm aelection for sat. *Journal of Artificial Intelligence Research* 565–606.

Xu, L.; Hoos, H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In *AAAI*, volume 10, 210–216.

Yuan, C., and Malone, B. 2013. Learning optimal bayesian networks: A shortest path perspective. *J. Artif. Intell. Res.(JAIR)* 48:23–65.

Zhou, R., and Hansen, E. A. 2002. Multiple sequence alignment using anytime A*. In *Proceedings of 18th National Conference on Artificial Intelligence AAAI'2002*, 975–976.

Zhou, R., and Hansen, E. A. 2005. Beam-stack search: Integrating backtracking with beam search. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, 90–98.