Simultaneously Searching with Multiple Settings: An Alternative to Parameter Tuning for Suboptimal Single-Agent Search Algorithms

Richard Valenzano, Nathan Sturtevant, Jonathan Schaeffer
University of Alberta
{valenzan, nathanst, jonathan}@cs.ualberta.ca

Karen Buro
Grant MacEwan University
burok@macewan.ca

Akihiro Kishimoto
Tokyo Institute of Technology and
Japan Science and Technology Agency
kishimoto@is.titech.ac.jp

Abstract

Many search algorithms have parameters that need to be tuned to get the best performance. Typically, the parameters are tuned offline, resulting in a generic setting that is supposed to be effective on all problem instances. For suboptimal single-agent search, problem-instance-specific parameter settings can result in substantially reduced search effort. We consider the use of dovetailing as a way to take advantage of this fact. Dovetailing is a procedure that performs search with multiple parameter settings simultaneously. Dovetailing is shown to improve the search speed of weighted IDA* by several orders of magnitude and to generally enhance the performance of weighted RBFS. This procedure is trivially parallelizable and is shown to be an effective form of parallelization for WA* and BULB. In particular, using WA* with parallel dovetailing yields good speedups in the sliding-tile puzzle domain, and increases the number of problems solved when used in an automated planning system.

1. Introduction

When constructing a single-agent search system, there are a number of decisions to be made that can significantly affect search efficiency. While the most conspicuous of these design decisions are those of algorithm and heuristic function selection, there are often subtle choices, such as tie-breaking and operator ordering, that can also greatly impact the search speed. Following the work of Hutter *et al.* (Hutter, Hoos, and Stützle 2007) we will refer to the set of choices made for a particular algorithm as the algorithm's *configuration*.

In domains in which only suboptimal problem-solving is feasible, additional options arise as most applicable algorithms involve some kind of parameterization. For example, in the weighted variants of A*, IDA* (Korf 1985), and RBFS (Korf 1993), the value of the weight must be set.

In practice, parameter values are tested offline so as to find some single setting to be used in any future search. Unfortunately, parameter tuning is an expensive process that is specific to each problem domain. This issue is of particular concern when designing general problem-solving systems, for which offline tuning time is not typically available. In practice, researchers building such systems commit to a single parameter value and hope that it will be effective over a diverse class of problems (eg. (Bonet and Geffner 2001)).

While tuning will find the setting with the best average performance, there is no guarantee that this setting will perform well on each individual problem. Instead, other parameter values may have significantly better performance on certain problems. This behaviour is evident when weighted IDA* (WIDA*) (Korf 1993) is used to solve the standard 100 15-puzzle problems (Korf 1985). WIDA* is a simple adjustment to IDA* in which the familiar cost function, $f(s)$, is changed to $f(s) = g(s) + w * h(s)$, where $g(s)$ is the length of the current path from the initial state to state s, $h(s)$ is the heuristic value of s, and $w \geq 0$ is a positive real-valued parameter called the weight.

For each of the 100 problems and each weight in the set $S = \{1, 2, ..., 25\}$, the number of nodes expanded by WIDA* was recorded. The weight of 7 achieved the best average performance over all problems. This data was also used to find the weight in S that solved any specific problem p the fastest (*ie.* requiring the fewest node expansions). In Figure 1, for each problem p, we have plotted the ratio of the number of nodes expanded by the $w = 7$ search on problem p to the number of nodes expanded by the best weight in S for p. The problems have been sorted in the order of increasing difficulty (nodes expanded) for the $w = 7$ search.

The $w = 7$ setting is best on only 6 problems (for which the fraction is 1). Furthermore, on 82 of the 100 problems, there is a weight in S that expands less than half the nodes as does $w = 7$. In fact, if there existed a system that could properly select the best weight from S for each problem, it would expand 25 times fewer total nodes than $w = 7$ alone.

These results demonstrate that correctly selecting the weight on a problem-by-problem basis can dramatically improve search speed. Therefore configuration selection is an extremely important issue. In this paper, we consider the use of *dovetailing* to deal with this issue. Dovetailing will be shown to significantly improve the speed of WIDA* and generally enhance the performance of WRBFS on two benchmark domains: the sliding-tile puzzle and the pancake puzzle. We also investigate the performance of the trivial parallelization of dovetailing as it applies to WIDA*, weighted RBFS (WRBFS), weighted A* (WA*), and the beam-search variant BULB (Furcy and Koenig 2005). We conclude with experiments that test the effectiveness of

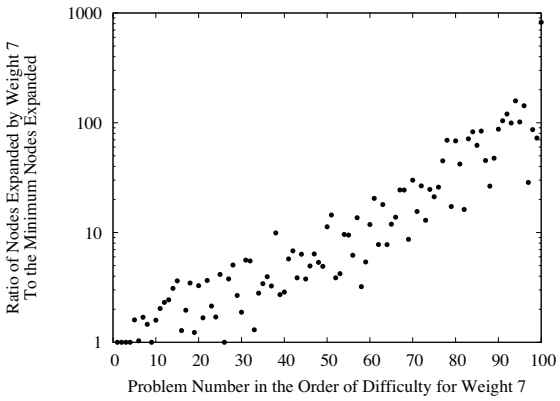

Figure 1: Comparing the performance of a $w = 7$ WIDA* search to the best of all other integer weights in the range of 1 to 25 on 100 15-Puzzle problems.

dovetailing when used with a modern WA*-based satisficing planner.

2. Dovetailing for Single-Agent Search

Dovetailing is a strategy that takes as its input a problem p and a set of ordered pairs of search algorithms and configurations $A = \{(a_0, \theta_0), ..., (a_n, \theta_n)\}$ where for each i, θ_i is a configuration for algorithm a_i. The output of dovetailing is a solution to p. The set A is called an *algorithm portfolio* and each pair in A is called a *candidate algorithm*.

We will assume that each candidate algorithm performs the search in a series of steps and the work done during each step is comparable between algorithms. We also assume that all candidate algorithms share the same base algorithm (*ie.* $a_0 = a_1 = ... = a_n$) and differ only in the configuration. As such, we will refer to the input of dovetailing as being a *candidate set of configurations* Θ for an algorithm a.

Dovetailing is a technique by which a parallel algorithm is run on a single processor by interleaving the work done by each thread. This procedure consists of a number of rounds. Each round works as follows: each candidate algorithm will, in order, advance its search by a single step. If some algorithm finds a goal on its turn, the solution found will be returned and dovetailing will stop. If a round completes without having found a solution, a new round begins. Note that each candidate algorithm is performing a completely independent search. There is no memory shared between configurations, and communication is restricted to messages indicating that a solution has been found for the current problem and the search should stop.

As each algorithm advances by a single step during each round, any algorithm in A will have performed approximately as much work as any other at any time. The total problem-solving time when dovetailing on a problem p is therfore approximately $|A|$ times the problem-solving time of the candidate algorithm with the best performance on p.

For most of our experiments, each algorithm step corresponds to exactly a single node expansion. Let $exp(a, \theta, p)$ denote the number of nodes that $a(\theta)$ expands while solving problem p. For some set of configurations Θ, let $oracle(a, \Theta, p) = \min_{\theta \in \Theta} exp(a, \theta, p)$. Intuitively, this value captures the number of nodes expanded when solving problem p given an oracle that knows exactly which configuration is most efficient on p. The number of nodes expanded when dovetailing over Θ will then have $|\Theta| oracle(a, \Theta, p)$ as an upper bound. The actual number may be slightly lower due to the order in which configurations expand a node in any round. For example, if a configuration $\theta \in \Theta$ is the best for problem p, and it is the first configuration which expands a node during round $exp(a, \theta, p)$ of dovetailing over Θ, then only $|\Theta|(oracle(a, \Theta, p) - 1) + 1$ nodes will be expanded. However, as we are only interested in difficult problems, the factor of $oracle(a, \Theta, p)$ dominates this expression. Therefore, the upper bound of $|\Theta| oracle(a, \Theta, p)$ is very accurate in practice.

Many of the properties of dovetailing will be related to the properties of the candidate configurations. For example, if each of the candidate algorithms has a bound on the solution suboptimality, then the suboptimality will be bounded by the maximum of the individual bounds. Similarly, the memory requirement of dovetailing is exactly the sum of the memory requirements of each of the individual algorithms. As such, dovetailing is problematic for memory intensive algorithms such as weighted A*.

Parallel Dovetailing

As multi-core machines and computer clusters become more readily available, the importance of algorithm parallelization as a way of taking advantage of such resources becomes increased. *Parallel dovetailing* takes in an algorithm a and a candidate set Θ, and assigns a unique configuration $\theta \in \Theta$ to each of $|\Theta|$ processors. Each processor will then perform an independent search on a problem p with the algorithm configuration assigned to it. All processors will work on the same problem simultaneously and communication is limited to messages indicating that p has been solved and processors should proceed to the next problem. The time taken by parallel dovetailing with algorithm a and configuration set Θ on a problem p will be given by the time any single processor takes to expand $oracle(a, \Theta, p)$ nodes.

Dovetailing and Diversity

If a search algorithm makes an early mistake it may spend a lot of time exploring areas of the state space that do not contain a solution. By expanding multiple candidate paths in parallel, diversity is introduced into the search. This is the strategy taken by beam searches and the KBFS algorithm (Felner, Kraus, and Korf 2003). In practice, diversity helps to decrease the probability of becoming stuck in a heuristic local minima or an area with many dead-ends.

Dovetailing will achieve diversity in search if there is diversity in the behaviour of the candidate algorithms selected. If the algorithms all search the state space in a similar manner, any differences in search effort between candidate algorithms will be small and any improvement made by an oracle will be overwhelmed by the cost of running multiple algorithms simultaneously. For example, the worst case for dovetailing over k instances of an algorithm occurs when the

candidate algorithms are identical, in which case dovetailing will take a factor of k more time than is necessary.

If the candidate algorithms perform a diverse set of searches, there is an increased chance that one of them will avoid dead-ends or heuristic local minima. It is this aspect of dovetailing that can lead to its strong behaviour in practice.

3. Related Work

As far as we know, dovetailing has only been previously considered for suboptimal search by Kevin Knight (Knight 1993). He demonstrated that by dovetailing over many instances of Learning Real-Time A* (LRTA*), each with a lookahead of 1 but a different random tie-breaking strategy, the solutions found were shorter than those found when running a single instance. This strategy also found solutions faster than the single instance of LRTA* with a larger lookahead that achieved similar average solution quality. While Knight only considered dovetailing over random number generator seeds, this idea will be generalized to show that dovetailing can be used over other parameter spaces.

EUREKA is a search system designed to construct problem-specific parallel versions of IDA*(Cook and Varnell 1997). The system does so by collecting statistics during a breadth-first expansion of 100,000 nodes. These statistics are then fed to a decision-tree that builds a parallel IDA* instance by selecting between various methods of task distribution, load balancing, and node ordering. The decision tree is trained using a set of problem instances, each annotated with the combination of techniques found to be most effective for that problem. Note, this approach was not considered for suboptimal search and while it may prove to be effective when used in this way, it is much more complicated than parallel dovetailing.

Dovetailing is also related to the use of random restarts in SAT and constraint satisfaction solvers. Gomes *et al.* showed that the strength of this technique is due to the distribution over run-times for different configurations (even when the configurations only differ by the random seed) being of the Pareto-Lévy class, which have an infinite mean and variance (Gomes, Selman, and Crato 1997). Restarts significantly decrease the tail-lengths of these distributions.

While dovetailing interleaves the execution of different configurations, restarts run a single configuration which changes every so often. Whenever the configuration is changed, the search begins anew (perhaps with information learned along the way). Restarts can be viewed as orthogonal to dovetailing since configurations in a candidate set could include the use of restarts. Parallel dovetailing can also be viewed as a natural parallelization of restarts.

Note that the notion of running an algorithm portfolio in parallel, with each algorithm having been assigned to a separate processor, is not new. This idea has been successfully applied in SAT solvers (Hamadi, Jabbour, and Sais 2009) and Satisfiability Modulo Theories (Wintersteiger, Hamadi, and de Moura 2009). However, to the best of our knowledge, this idea has not been explored in single-agent search. Section 6. presents results showing that the use of an algorithm portfolio offers an effective form of parallelization for suboptimal single-agent search algorithms.

Finally, our work is also related to automatic configuration selection systems such as ParamILS and FocusedILS (Hutter, Hoos, and Stützle 2007). These systems replace the manual offline tuning procedure with a local search which optimizes an algorithm for a given training set (*ie.* for a single domain). As we are more interested in per-problem tuning, we did not consider this approach any further.

4. Experimentation Through Simulation

When testing the performance of dovetailing on an algorithm a, a set of configurations Ω, called the *starting configuration set*, is initially selected. For some problem set P, each problem $p \in P$ was solved using $a(\theta)$ for each $\theta \in \Omega$. Having collected this information, it is easy to calculate $oracle(a, \Theta, p)$ for any $\Theta \subseteq \Omega$ on any $p \in P$. The total number of nodes expanded when dovetailing over Θ can then be approximated by $|\Theta| \sum_{p \in P} oracle(a, \Theta, p)$.

In Section 5. all experiments were performed using this simulation technique. Testing in this way allows us to efficiently calculate the performance of dovetailing on a large number of candidate sets and thereby determine how robust dovetailing is with respect to the selection of the candidate set. Where there are n configurations in the starting configuration set, we will consider the average performance over all $\binom{n}{k}$ possible candidate sets of size k in terms of the total number of nodes expanded on the problem set, as well as the candidate sets with the best and worst performance. The figures will show how many times fewer nodes are expanded by dovetailing than are expanded by the single configuration $\theta^* \in \Omega$ with the best average performance without dovetailing (defined as $\theta^* = \arg\min_{\theta \in \Omega} \sum_{p \in P} exp(a, \theta, p)$). For example, if the value shown for the best candidate set for some k is 5, this means that the best candidate set of the $\binom{n}{k}$ candidate sets of size k expanded 5 times fewer total nodes than θ^* alone.

As we are also interested in determining over which parameter spaces dovetailing is effective, the starting configuration sets will be selected such that they are identical with the exception of a single design choice. The first choice considered will be the main algorithm parameter: the weight for weighted algorithms and the beam width for BULB.

Part of the standard definition of a single-agent search domain is the successor function. Typically, the implementation of this function begins with an ordering over all operators applicable in the domain. The successor function constructs a list L of states by checking each operator in order for applicability. If an operator is found to be applicable to the current state, the corresponding child is constructed and appended to L. The order of states in L can often significantly impact the speed of search performed by some algorithm. For example, in weighted A*, nodes are typically added to the open list in the order in which they appear in L. As such, the underlying operator ordering can change how ties are broken between nodes with equal f-cost (and g-cost where this value is also used to break ties). Therefore, we also consider starting configuration sets which only differ in their operator ordering.

5. Dovetailing over WIDA* and WRBFS Instances

Weighted IDA* (WIDA*) and Weighted RBFS (WRBFS) are linear-space alternatives to WA* (Korf 1993). Both use $w \geq 1$; and the solution cost found is guaranteed to be no worse than w times that of the optimal value.

Unfortunately, both algorithms do not handle early mistakes in the heuristic well. In WIDA*, the search can become stuck investigating large areas of the search space with no heuristic guidance. In this algorithm, heuristic guidance is given in the form of a f-cost threshold T, such that nodes with a larger f-value are not expanded until the threshold is increased. Consider the example of a unit cost graph with a constant branching factor b, bi-directional edges, and a consistent heuristic (*ie.* the heuristic will change by at most 1 from the child to the parent). For some node c with a heuristic value exactly one greater than its parent p, the following holds: $f(c) = g(c)+w*h(c) = g(p)+1+w*(h(p)+1) = g(p)+w*h(p)+w+1 = f(p)+w+1$. This means that assuming the depth of the search tree is unbounded, the search below a node n will not be cut off until at least a depth of $\lfloor (T - f(n))/(w + 1) \rfloor$. Therefore, in the subtree of size $b^{\lfloor (T-f(n))/(w+1) \rfloor}$ below n, WIDA* will have no heuristic guidance as no pruning will occur. As a result, if a heuristic incorrectly guides the search into an area of low f-value, many nodes will have to be expanded before the search will backtrack out of the area.

WRBFS was introduced as an attempt to handle this issue that WIDA* has with inconsistency. However, as reported in the original RBFS paper (Korf 1993), WIDA* generally outperforms WRBFS in practice. We have seen similar behaviour in our own experiments. Due to these results, WIDA* should still be viewed as a valuable algorithm, even in light of the existence of WRBFS.

Dovetailing is tested with WIDA* and WRBFS on two single-agent search domains: the sliding-tile puzzle and the pancake puzzle. Several sizes of the sliding-tile puzzle were considered so as to evaluate how well the performance scales. For the sliding-tile puzzle, the Manhattan distance heuristic is used. For the pancake puzzle, pattern database heuristics are used (Culberson and Schaeffer 1996). $PDB(p_0, p_1, p_2, ..., p_i)$ is used to denote the pattern database heuristic in which the pancakes p_j for $0 \leq j \leq i$ are distinct in the abstract space. The performance of a simple parallelization of dovetailing with WA* and BULB (Furcy and Koenig 2005) is considered later in this paper.

Dovetailing over Weights in WIDA* and WRBFS

We begin by considering a starting configuration set of size 15, where configurations differ only in the assigned integer weight [2...16]. Consider the performance of dovetailing over these WIDA* configurations on the sliding-tile puzzle. In the case of the 5×5 puzzle, the single configuration with the best average performance over 1000 randomly generated problems was the $w = 5$ configuration. Figure 2 shows the dovetailing improvement when compared to this weight.

When the candidate set sizes reach 3 and 5, the average and worst configurations, respectively, outperform even the single best configuration alone. When dovetailing over all 15 configurations (*ie.* candidate set size is 15), the total number of nodes expanded is reduced by a factor of 42.5 and the average solution length only decreases by a factor of 1.8.

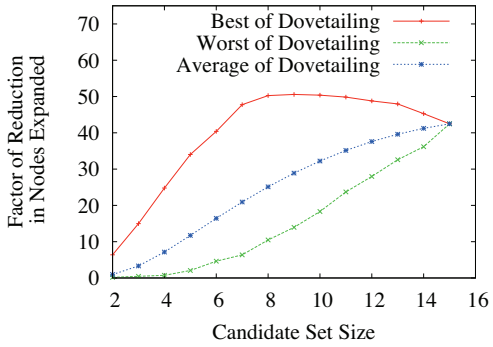

Figure 2: Dovetailing over weights in WIDA* on 1000 5×5 sliding-tile puzzles.

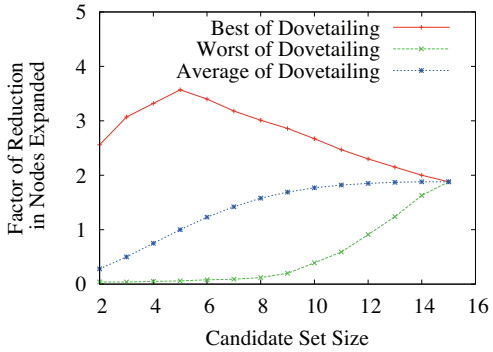

Figure 3: Dovetailing over weights in WRBFS on the 4×5 sliding-tile puzzle.

WRBFS does not scale as well to large domains such as the 5×5 puzzle, and so the data collection phase needed for simulation was too time-consuming. Instead, we show the simulation results for the 4×5 sliding-tile puzzle. The single best weight for WRBFS in this domain is $w = 3$. Figure 3 shows the factor of improvement in nodes expanded when using dovetailing as compared to the $w = 3$ search alone. The average performance of dovetailing reaches the performance of the best of the configurations alone at a candidate set size of 5. When the candidate set is of size 15, dovetailing requires 1.9 times fewer node expansions than $w = 3$, while finding solutions that are on average 1.3 times longer.

In both algorithms, increasing the domain size also increased the effectiveness of dovetailing. For WIDA*, dovetailing over all 15 configurations decreased the number of nodes by a factor of 2.1, 7.9, and 42.5 (as reported above) when compared to the single best configuration on 1000 problems of each of the 4×4, 4×5, and 5×5 puzzles, respectively. Furthermore, on 100 6×6 puzzles, dovetailing over these 15 configurations decreased the number of nodes expanded by a factor of 121 over the $w = 5$ search (the only

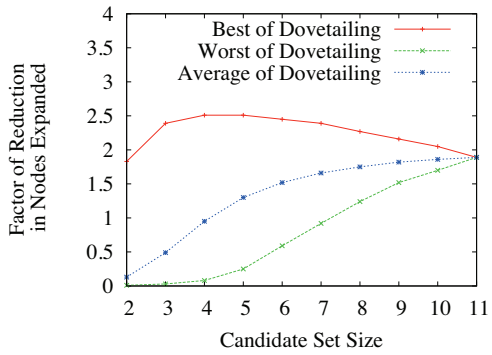

Figure 4: Dovetailing over weights in WIDA* on the 16 pancake puzzle.

configuration that could successfully solve the entire problem set in the time allotted).

On 1000 4×4 sliding-tile puzzle problems, dovetailing over all 15 WRBFS configurations required 1.7 times more node expansions than the single best weight alone. With respect to the 5×5 puzzle, while we cannot present simulation results for dovetailing with WRBFS due to the time needed for the data collection phase, dovetailing over all 15 configurations did improve upon the number of nodes expanded when compared to the single best weight of those that did successfully solve all 1000 problems in the problem set ($w = 3$) by a factor of 11.5. This provides further evidence that the effectiveness of dovetailing increases with the domain size when used with WRBFS.

Consider dovetailing with WIDA* on the 16 pancake puzzle with the heuristic given by maximizing over the $PDB(0, 1, 2, 3, 4, 5)$ and $PDB(6, 7, 8, 9)$ heuristics. Figure 4 shows the improvement factor in total nodes expanded on 1000 randomly generated problems for each of the candidate set sizes, compared to the best weight of 5 alone.

In these experiments, the starting configuration set was restricted to the 11 configurations given by the unique integer weights in the range of $[2, 12]$. This is because all other weights took too long to solve all the problems. When dovetailing over all 11 of these configurations, 1.9 times fewer nodes are expanded than the best weight of 5 alone, with a solution cost that is 1.4 times worse. If we consider dovetailing over all 15 configurations from the starting configuration set considered earlier, dovetailing also expands 1.8 times fewer nodes than $w = 5$ alone.

Similar experiments were performed with WRBFS. Again, the starting configuration set only contains the 11 configurations with integer weights in the range of $[2, 12]$. In this case, dovetailing over all 11 configurations requires 3.3 times as many nodes as the single best configuration ($w = 3$) alone. This gap narrows when we consider dovetailing over all 15 configurations given by the integer weights from 2 to 16, in which case dovetailing only expands 1.7 times as many nodes. However, in order to determine which weight alone has the best performance, we would need offline tuning. Here dovetailing has allowed us to avoid the expensive tuning phase with only a small degradation of performance.

Weight	WIDA* on the 5×5 puzzle		WRBFS on the 4×5 puzzle	
	Over Best Order	Over Av. Order	Over Best Order	Over Av. Order
3	5.6	3.46	0.9	1.6
4	14.5	127.4	1.3	3.8
5	37.1	115.4	3.3	6.3
6	43.0	223.1	2.3	6.1
7	47.8	254.8	4.6	9.0
8	33.7	217.1	4.9	13.8
9	48.3	349.0	7.4	17.2
10	142.5	380.0	5.8	16.1

Table 1: Dovetailing over operator ordering in WIDA* and WRBFS on the sliding-tile puzzle in terms of factor of improvement over the best and the average configuration.

Dovetailing over Operator Orderings in WIDA* and WRBFS

Consider dovetailing over configurations that only differ in the operator ordering being used. For the sliding-tile experiments, we consider 8 different starting configuration sets for each algorithm. Each starting configuration set will contain configurations with a different weight, and each will contain 24 configurations (one for each of the 24 possible operator orderings). For example, the $w = 3$ set will contain 24 configurations each with a weight of 3 but a different operator ordering, while the $w = 4$ set is defined analogously, except all configurations are set to have a weight of 4.

Table 1 summarizes the operator ordering experiments with WIDA* and WRBFS in the 5×5 and 4×5 sliding-tile puzzles, respectively. The test sets contain 100 randomly generated problems. For each of the configuration sets, we show the factor of the decrease in the number of nodes expanded when using dovetailing over all 24 configurations when compared to the best configuration in the set, and when compared to the average performance of all 24 configurations. For example, dovetailing over the 24 $w = 5$ WIDA* configurations required 37.1 times fewer node expansions than the single best operator ordering, and 115 times fewer than the number of nodes expanded on average over all 24 configurations. We include the comparison with the average configuration since it is generally not well understood as to how this design choice will affect search speed. As such, a system designer is left to perform a lot of tuning, or, more commonly, randomly select a operator ordering and stick with it.

The table shows that dovetailing over operator orderings generally significantly improves both algorithms in the sliding-tile puzzle domain. This is particularly true of the higher weights which generally have much worse performance on their own. In this way, dovetailing over orderings reduces the effect of a system designer selecting a poor weight. For example, the average performance of each of the 24 operator orderings alone is 3.9 times worse when the weight is set as 10 as opposed to 5. However, if dovetailing is used, the weight of 10 starting configuration set performance is only 1.18 times worse than the performance of the weight 5 configuration set, and it is still 97.7 times

Weight	WIDA* Over Best Order	WIDA* Over Av. Order	WRBFS Over Best Order	WRBFS Over Av. Order
3	1.1	1.8	0.2	0.2
4	1.4	2.4	0.2	0.3
5	2.6	4.7	0.3	0.5
6	2.3	4.3	0.4	0.7
7	2.5	8.2	0.7	1.1
8	4.4	14.3	0.7	1.2
9	15.4	63.0	1.8	3.6
10	77.9	279.6	2.1	4.4

Table 2: Dovetailing over operator ordering in WIDA* and WRBFS on the 16 pancake puzzle (factor of improvement over the best and the average configuration).

faster than the average single weight 5 configuration. Therefore, the penalty for mistakenly using the weight of 10 is not nearly as severe.

For the pancake puzzle, we still consider different starting configuration sets, each containing 15 of the 15! possible operator orderings. The operator orderings were selected randomly under the constraint that each of the 15 begins with a different operator. Table 2 summarizes the results of dovetailing over operator orderings with WIDA* and WRBFS on the 16 pancake puzzle. Again the test set used contains 100 randomly generated problems.

In this puzzle, dovetailing is clearly less effective than it was in the sliding-tile puzzle (which was also evident when considering dovetailing over weights). However, it still improved WIDA* with all starting configuration sets considered, and improved WRBFS on the larger weights.

This improvement in search speed also comes at almost no cost in terms of solution quality. When dovetailing over WIDA* instances all with the same weight, the difference in solution quality when compared to the average solution quality of the configurations in the starting configuration set was virtually neglible. When dovetailing over WRBFS configurations, the solution quality generally improved, whith the largest improvements being found on the higher weight configuration sets. For example, on the 4×5 sliding-tile puzzle, dovetailing over all 24 $w = 3$ configurations improved the solution quality by 1% over the average $w = 3$ configuration, while dovetailing over all $w = 10$ configurations improved the quality by 8% over the average $w = 10$ configuration.

6. Parallel Dovetailing

In this section we will experiment with parallel dovetailing. Single-agent search algorithms are notoriously difficult to parallelize, and we will show that despite the fact that parallel dovetailing is a very simple procedure, it often yields significant gains.

Parallel Dovetailing with WIDA* and WRBFS

To determine the effectiveness of parallel dovetailing with WIDA* and WRBFS, the speedups reported in Section 5. are simply multiplied by the candidate set size being used. For example, parallel dovetailing with 24 cores in the 5×5 sliding-tile puzzle over 24 WIDA* configurations each with a weight of 6 but a different operator ordering, results in a speedup of $43\times24 = 1032$ over the single best configuration alone. This means that in most of our experiments we are seeing *super-linear speedup*, which occurs when the improvement factor is greater than the number of processors being used. Any dovetailing point with a value greater than 1 in any of Figures 2, 3, or 4 are indicative of a super-linear speedup. The same is true for entries in Tables 1 and 2.

While there have been many attempts at parallelizing IDA*, we are unaware of any work on parallelizing WIDA*. While the techniques used to parallelize IDA* may work with WIDA*, they are generally much more complicated than this simple algorithm portfolio approach which we have shown to often yield several orders of magnitude of speedup with only a small number of cores. We are also unaware of any attempts at parallelizing either WRBFS or RBFS, but have shown that parallel dovetailing often yields super-linear speedups in the former.

Parallel Dovetailing with WA*

While dovetailing is ill-suited for use with WA* due to the large memory requirements of the algorithm, parallel dovetailing can still be used, particularly in the presence of distributed memory machines. In this section, we will compare the performance of parallel dovetailing with a recent technique for parallelizing WA*, called wPBNF (Burns et al. 2009b), on the 4×4 sliding-tile puzzle. Specifically, we will compare the speedups reported in the wPBNF paper to our own speedups when using parallel dovetailing over operator orderings in this domain.

As the wPBNF evaluation shares memory between processors, we will do the same. In our setup, we will not allow the cumulative number of states held in memory by all instances to exceed 1×10^6. As wPBNF was tested with a 15 GB memory system, we suspect this is a significant underestimate of the number of nodes that wPBNF was allowed to hold in memory. We then assume that once memory has been exhausted by the k instances, $k - 1$ instances stop, clear their memory usage, and sit idle as the processor continues on to solve the problem using the 1×10^6 node limit as needed. We will also assume that the algorithm always "guesses" wrong and always continues with the instance that will take longest to complete its search.

Finally, the speedup results that we present for some candidate set size k will correspond to the average performance over $min(10,000, \binom{24}{k})$ candidate sets tested when compared to the single best configuration alone. As such, it should be clear that we are using a pessimistic evaluation of parallel dovetailing.

The speedups for wPBNF, taken from a combination of a conference paper (Burns et al. 2009b) and a workshop paper (Burns et al. 2009a), and the simulated speedups of parallel dovetailing are shown in Table 3. Notice that wPBNF outperforms parallel dovetailing for small weights, but actually degrades performance from a serial WA* for larger weights. This suggests that techniques for parallelizing A* cannot necessarily be expected to yield similar speedups when applied to WA*.

	Speedup With Different Numbers of Processors					
	wPBNF			Parallel Dovetailing		
Weight	2	4	8	2	4	8
1.4	**1.12**	**1.65**	**2.62**	0.99	0.98	0.97
1.7	0.76	**1.37**	**1.49**	**0.98**	1.02	1.03
2.0	0.62	1.10	**1.46**	**1.01**	**1.15**	1.23
3.0	0.62	0.90	0.78	**1.06**	**1.62**	**2.05**
5.0	0.60	0.76	0.64	**1.16**	**1.62**	**2.02**

Table 3: The speedup of wPBNF and the average speedup of parallel dovetailing over operator orderings on 42 4×4 sliding-tile puzzle problems.

In contrast, parallel dovetailing outperforms wPBNF for the larger weights while slightly underperforming a serial WA* on the smallest weights (mostly because of the comparison of average performance to the best configuration). Keep in mind that serial WA* requires the least amount of time with the largest weights. As such, in this domain parallel dovetailing is able to effectively improve on the best performance of WA* (in terms of search speed) despite its simplicity, unlike wPBNF which has difficulties doing so.

Parallel Dovetailing and BULB

Beam Search using Limited Discrepancy Backtracking (BULB) is a beam-search variant that is capable of quickly finding solutions in very large state-spaces (Furcy and Koenig 2005). While it does not have proven bounds on suboptimality like other algorithms previously discussed, it is quite effective in practice. As it uses a large amount of memory, we only consider parallelizing this algorithm with parallel dovetailing. To the best of our knowledge, there have been no other attempts to parallelize BULB or any other beam-search-like algorithms.

Consider dovetailing over 12 BULB instances with beam widths ranging from 7 to 200 on 100 7×7 sliding-tile puzzles. When dovetailing over all 12 configurations with 12 processors, we achieved a 2.2 times speedup when compared to the single best beam width of 7. However, the average solution length is 4 times shorter than that found by the beam width of 7. When parallel dovetailing with BULB is compared with the beam width with the most similar solution quality, we see a 22 times speedup.

When dovetailing over 24 operator orderings, all with a beam width of 7, the speedup achieved and the solution quality improvement were both 3.8. An analogous size 24 candidate set with beam width 10 saw both a speedup and solution quality improvement of 5.9. When compared to the single beam width with the most similar solution quality, the beam width 7 and 10 candidate sets saw speedups of 44 and 43 times respectively.

7. Applications To Planning

We also tested the use of parallel dovetailing for classical planning by using the WA*-based Fast Downward planner (Helmert 2006) on a set of 846 problems taken from the satisficing track of the last planning competition. 36 different configurations were considered. The first 11 configurations used only the FF heuristic and were identical with the exception of the weight value used. Specifically, the 10 integer weights from 1 to 10 inclusive were used, as was the weight of infinity (thereby producing a pure heuristic search). The next 11 configurations used only the landmark cut heuristic (Helmert and Domshlak 2009) and used the same set of weights. A third set of 11 configurations used only the context-enhanced additive (CEA) heuristic (Helmert and Geffner 2008), each with a different weight from the set described above.

Three additional configurations were also used. The first performs a pure heuristic search using preferred operators and the FF heuristic. The second performs a pure heuristic search using preferred operators and the CEA heuristic. The final configuration performs a pure heuristic search version of multi-heuristic best-first search with preferred operators and both the FF and CEA heuristics.

In our experiments, each configuration was given 2 GB of memory and 30 minutes to solve each of the problems on a single processor of a dual quad-core 2.33 GHz Xeon E5410 machine with 6MB of L2 cache. Under this setup, each configuration solved 653 of the 846 problems on average. The single best configuration solved 746 of the problems.

After collecting this data, we simulated the use of parallel dovetailing on a distributed cluster of 36 identical machines, each having been assigned a different configuration. Such a system would solve 789 problems, 43 more than even the single best configuration alone. In comparison, recent work in optimal parallel planning (Kishimoto, Fukunaga, and Botea 2009) showed an increase of 11 problems solved when using their architecture on 128 machines. While our results are not directly comparable since they were considering optimal planning and used a different test suite, they do suggest that parallel dovetailing offers an effective alternative to the development and implementation of complicated parallelization techniques for distributed parallel planning.

8. Conclusion

We have demonstrated that while offline tuning can find values with good average performance, systems that properly find problem-specific parameters can outperform those that rely on tuned parameters. Dovetailing was shown to significantly improve WIDA* and generally improve WRBFS by taking advantage of this fact.

A trivial parallelization of dovetailing was also considered. It usually achieved super-linear speedups when applied with WIDA* and WRBFS. With WA*, it was shown to outperform a state-of-the-art parallelization of this algorithm for higher weights in the 4×4 puzzle. It also helped improve the solution quality of BULB with a significant speedup and it increased the number of problems solved for a modern automated planning system.

While dovetailing requires the selection of a candidate set, we have shown that it is still effective over a large range of candidate sets. Moreover, we assert that the use of a single configuration is more error-prone approach than is the selection of a candidate set. This is because the use of multiple

configurations helps overcome the deficiencies of any single configuration at a linear cost in the candidate set size.

When using only a single configuration, there is no such backup. Even if a strong single configuration is found, it cannot be expected to do well on all problems. Having said that, investigating policies for identifying good candidate sets, either with offline or online computation, remains an area of future work.

Acknowledgments

This research was supported by iCORE, NSERC, and JST Presto.

References

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

Burns, E.; Lemons, S.; Ruml, W.; and Zhou, R. 2009a. Parallel Best-First Search: Optimal and Suboptimal Solutions. In *Proceedings of the International Symposium on Combinatorial Search (SoCS-09)*.

Burns, E.; Lemons, S.; Ruml, W.; and Zhou, R. 2009b. Suboptimal and anytime heuristic search on multi-core machines. In *ICAPS*.

Cook, D. J., and Varnell, R. C. 1997. Maximizing the Benefits of Parallel Search Using Machine Learning. In *AAAI*, 559–564.

Culberson, J. C., and Schaeffer, J. 1996. Searching with Pattern Databases. In McCalla, G. I., ed., *Canadian Conference on AI*, volume 1081 of *Lecture Notes in Computer Science*, 402–416. Springer.

Felner, A.; Kraus, S.; and Korf, R. E. 2003. Kbfs: K-best-first search. *Ann. Math. Artif. Intell.* 39(1-2):19–39.

Furcy, D., and Koenig, S. 2005. Limited Discrepancy Beam Search. In *IJCAI*, 125–131.

Gomes, C. P.; Selman, B.; and Crato, N. 1997. Heavy-tailed distributions in combinatorial search. In Smolka, G., ed., *CP*, volume 1330 of *Lecture Notes in Computer Science*, 121–135. Springer.

Hamadi, Y.; Jabbour, S.; and Sais, L. 2009. ManySAT: a Parallel SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation* 6:245–262.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In *ICAPS*, 140–147.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Hutter, F.; Hoos, H. H.; and Stützle, T. 2007. Automatic Algorithm Configuration Based on Local Search. In *AAAI*, 1152–1157. AAAI Press.

Kishimoto, A.; Fukunaga, A.; and Botea, A. 2009. Scalable, Parallel Best-First Search for Optimal Sequential Planning. In *ICAPS*, 201–208.

Knight, K. 1993. Are Many Reactive Agents Better Than a Few Deliberative Ones? In *IJCAI*, 432–437.

Korf, R. E. 1985. Iterative-Deepening-A*: An Optimal Admissible Tree Search. In *IJCAI*, 1034–1036.

Korf, R. E. 1993. Linear-Space Best-First Search. *Artif. Intell.* 62(1):41–78.

Wintersteiger, C. M.; Hamadi, Y.; and de Moura, L. M. 2009. A Concurrent Portfolio Approach to SMT Solving. In Bouajjani, A., and Maler, O., eds., *CAV*, volume 5643 of *Lecture Notes in Computer Science*, 715–720. Springer.