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

**Abstract**

Many search algorithms have parameters that need to be tuned to get 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, parameter settings that are problem-instance specific can result in substantially reduced search effort. We consider an algorithm enhancement, turn-taking, which searches with multiple parameter settings simultaneously. Turn-taking is a trivially parallelizable procedure that is appropriate for use with linear-space algorithms such as weighted IDA*, RBFS, and beam search. We present experimental results for the sliding-tile and pancake puzzles showing that enhancing weighted IDA* with turn-taking can result in several orders of magnitude of speedup. We also examine the performance of similarly enhanced versions of weighted RBFS and the beam-search variant BULB.

## 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. As such, properly recognizing and evaluating all of the necessary design decisions is a vital aspect of building an effective search system.

In domains in which only suboptimal problem-solving is feasible, additional options arise as almost all 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 parameter must be set. Similarly, beam-search variants like BULB (Furcy and Koenig 2005a) and Beam-stack search (Zhou and Hansen 2005) require the selection of a beam width. Parameterization also occurs in the class of best-first search variants like KBFS (Felner, Kraus, and Korf 2003), MSC-WA* (Kitamura et al. 1998), and MSC-KWA* (Furcy and Koenig 2005b) in which a system designer can adjust the number of nodes expanded in parallel, the number of nodes to commit to, and the combination of these two ideas, respectively.

In general, the adjustment of parameters is expected to involve some exchange of solution quality for search speed. However, these metrics are not always inversely related as there often exists a set of parameter settings that will only require some minimum amount of total work. Parameter values outside this set will perform a more expensive search, regardless of the change to the solution quality.

If an algorithm exhibits such behaviour, the importance of proper parameter selection is magnified. 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. The fact that tuning information cannot effectively be transferred across domains is of particular concern when designing general problem-solving systems. In general, 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)).

Parameter tuning also suffers from another deficiency: there is no guarantee that a tuned value will perform well on each individual problem. Tuning only finds the setting that has the best average performance on some training set. However, on a per problem basis, there can be other parameter values which significantly outperform the tuned setting.

This behaviour can be seen in the following experiment which examines the performance of weighted IDA* (WIDA*) (Korf 1993) on the set of 100 15-puzzle problems used in the original IDA* paper (Korf 1985). WIDA* is a simple adjustment to IDA* in which the familiar cost function, $f(s)$, is changed to the following:

$$f(s) = g(s) + w * h(s)$$

where $g(s)$ is the length of the shortest current path from the initial state to state $s$, $h(s)$ is the heuristic value of $s$, and $w \geq 0$ is a real-valued parameter called the weight.

For each of these 100 15-puzzle problems and each of the weights in the set $S = \{1, 2, ..., 25\}$, the speed of the WIDA* search (in terms of the number of nodes expanded) was recorded. After collecting this data, it was determined that of the weights in $S$, the weight of 7 expands the fewest total number of nodes over all 100 problems. This data was also used to find the best weight (in terms of fewest nodes expanded) in $S$ for any single problem instance $p$. In Figure 1, for each problem $p$, we have plotted the ratio of the

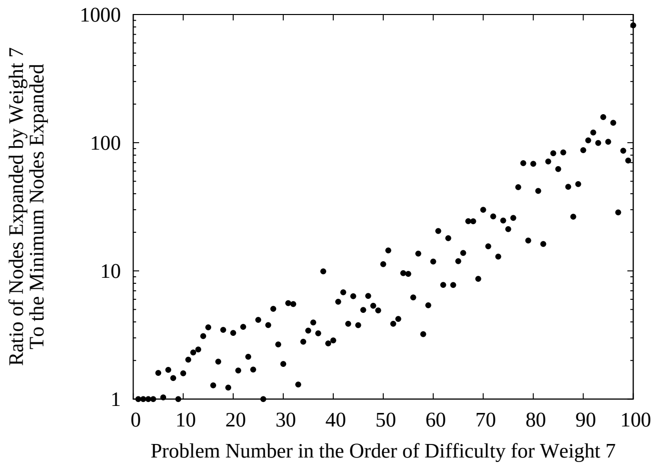

Figure 1: Comparing the Performance of WIDA* with a weight of 7 to all other integer weights in the range 1 to 25 on the 15-Puzzle.

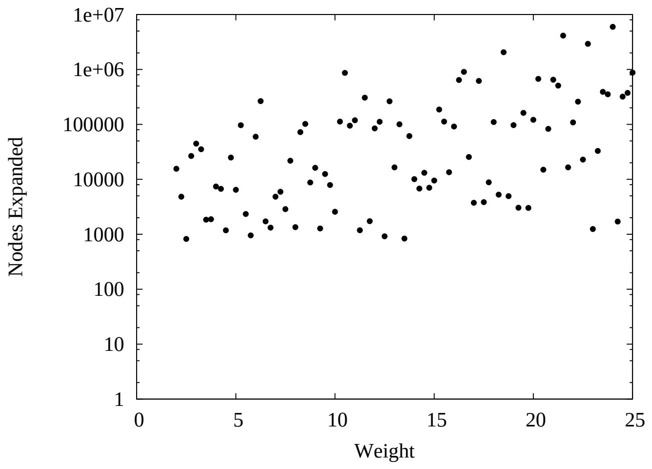

Figure 2: The number of nodes expanded by WIDA* on a single 15-puzzle problem when using weights 2 through 25 incremented by 0.25.

number of nodes expanded by weight 7 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 (in terms of nodes expanded) for weight 7.

Notice that weight 7 is the best of the weights on only 6 problems (for which the fraction is 1) despite expanding the fewest total nodes over the problem set. Furthermore, on 82 of the 100 problems there is a weight in $S$ that expands fewer than half as many nodes as weight 7. As a result, if there existed a system that could properly select the best weight from $S$ for each problem, the system would expand 25 times fewer total nodes than weight 7.

These results demonstrate that correctly selecting the weight on a problem-by-problem basis can dramatically improve search speed, and therefore proper parameter selection is an extremely important issue. In this end, we consider a policy referred to as turn-taking. Turn-taking is a simple strategy that will be shown to significantly improve WIDA* on two benchmark domains: the sliding-tile puzzle and the pancake puzzle. Turn-taking will also be shown to be an effective enhancement to the linear-space algorithm WRBFS when applied in one of these domains. Finally, we will also investigate the performance of a simple parallel version of turn-taking as it applies to WIDA*, RBFS, and the beam-search variant BULB.

## 2. Turn-Taking

In this section, we will introduce a new approach to parameter selection for suboptimal search algorithms called turn-taking. Before doing so, let us first consider the use of a classifier for parameter selection in suboptimal search algorithms, and the issues that can arise when this approach is applied to certain kinds of parameterization.

Consider Figure 2 which shows the nodes expanded by WIDA* when using a diverse set of weights on a single 15-puzzle problem. The figure demonstrates that the number of nodes expanded is not necessarily a smooth function of the weight, as small changes to the weight value can result in drastic changes in search effort. The noisiness of this domain suggests accurately approximating this function will be extremely difficult. Even if the function can be properly approximated, the resulting classifier will be expected to be domain specific.

Instead, we propose the policy of turn-taking. In order to introduce this idea, we will first require some notation. First, when we refer to an algorithm $a$, we will be referring to a self-contained search system that consists of the search procedure (A*, IDA*, etc.), a heuristic function, and in which any additional design decisions (parameter values, tie-breaking, etc.) have been made. Let $exp(a, p)$ denote the number of nodes expanded during the search by algorithm $a$ on a problem $p$. We can then define the batch results over a problem set $P$ of an algorithm $a$ as follows:

$$batch(a, P) = \sum_{p \in P} exp(a, p)$$

Now consider having an oracle that knows which of the algorithms in a set of algorithms $A$ performs the least amount of work on each problem in $P$. On any problem $p$, the amount of search performed by the oracle, denoted $oracle(A, p)$, is given by the following formula:

$$oracle(A, p) = \min_{a \in A} exp(a, p)$$

We will also use $oracle(A, P)$ to denote the total number of nodes expanded over an entire problem set $P$ by the oracle, formally expressed as follows:

$$oracle(A, P) = \sum_{p \in P} oracle(A, p)$$

With this notation, Figure 1 can be said to show the ratio of $exp(a, p)$ to $oracle(A, p)$ for each of the 100 problems, where $a$ is WIDA* with weight 7, and $A$ is a set of 25 identical WIDA* algorithms that only differ in the assigned weight value. In this situation, WIDA* with weight 7 was determined to perform the least amount of total work over the problem set, and will be said to have the best batch value of any algorithm in $A$.

As mentioned previously, in the application of WIDA* on the 15-puzzle with the given weight set, a search informed by an oracle will expand almost 25 times fewer nodes than the best batch value. The effectiveness of such an oracle demonstrates that there is potential for significant improvement over the use of any single parameter value. Furthermore, recall that Figure 2 shows that in some search algorithms, the parameter values that require a near minimum amount of work on any single problem need not be similar. Together, these results suggest the idea of simultaneously searching the space with multiple parameter values so as to increase the probability that at least one of the values will show good performance on each problem instance. In this end, we propose *sequential turn-taking*.

Sequential turn-taking (or simply turn-taking) is a procedure that takes as its input a set of ordered search algorithms $A = \{a_0, ..., a_n\}$, and a problem $p$. The output of turn-taking is a solution to $p$. The set $A$ will be called the candidate set, and each $a \in A$ will be called a candidate algorithm. The only pre-condition on the candidate algorithms is that each searches in a series of steps, and the work done during each step is comparable between algorithms.

The turn-taking process 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 $a_i$ finds a goal on its turn, the solution found by $a_i$ will be returned and turn-taking will stop. If a round completes without having found a solution, a new round begins.

Note that while turn-taking, each of the candidate algorithms is performing a completely independent search. As such, there is no information or memory shared by candidate algorithms.

By having each algorithm advance by a single step during each round, turn-taking ensures that at all times, any algorithm in $A$ will have performed approximately as much work as any other. As such, the total problem-solving time taken by turn-taking on a problem $p$ will be approximately $|A|$ times the amount of work done by the candidate algorithm with the best performance on $p$.

We will also consider the trivial parallelization of turn-taking, called *parallel turn-taking*, in which each of the candidate algorithms in $A$ is assigned to one of $|A|$ processors with distributed memory. In parallel turn-taking, each processor will perform a completely independent search on a problem $p$ and communication is limited to messages indicating that $p$ has been solved and processors should proceed to the next problem. As such, the time of search taken by parallel turn-taking using $|A|$ processors will be approximately a factor of $|A|$ less than the time taken by sequential turn-taking on a single processor.

In all experiments below, the algorithms have been implemented so that each step advances the search by exactly one node expansion. As such, the number of nodes expanded by turn-taking over a candidate set $A$ on a problem $p$ will be at most $|A| * oracle(A, p)$. While parallel turn-taking will perform the same amount of total work over all processors as sequential turn-taking, the search time is $oracle(A, p)$.

## 2.1 Turn-Taking and Memory

Many of the properties of a turn-taking search will be related to the properties of the candidate algorithms. For example, if each of the candidate algorithms has bounds on the solution suboptimality, the solution suboptimality of the turn-taking search will be the maximum of the individual bounds.

Similarly, the memory requirement of both parallel and sequential turn-taking is exactly the sum of the memory requirements of each of the individual algorithms. As such, turn-taking is not appropriate for memory intensive algorithms such as weighted A*. Due to these memory issues, the remainder of this paper will be focused on the use of turn-taking with linear-space search algorithms.

## 2.2 Turn-Taking and Diversity

If a search algorithm is misled by a heuristic it may spend a lot of time considering unneccessary areas of the state space. To address this issue, algorithms such as KBFS expand multiple candidate paths in parallel so as to introduce diversity into a search(Felner, Kraus, and Korf 2003). In practive, diversity helps to decrease the probability of becoming stuck in a heuristic local minima or an area with many dead-ends.

Turn-taking will achieve diversity in search provided there is diversity in the candidate algorithms. If there is not enough diversity in the algorithms, any differences in search effort between candidate algorithms will be small. In these situations, any improvement made by an oracle will be overwhelmed by the cost of running multiple algorithms. For example, note that the worst case for turn-taking occurs when the candidate algorithms are identical.

If the candidate algorithms do perform a diverse set of searches, there is an increased chance that at least one of these algorithms will avoid dead-ends or heuristic local minima. In this way, diversity in the candidate set allows for different algorithms to overcome the weaknesses of others.

## 3. Related Work

The issue of proper parameter selection is not unique to the field of sub-optimal search, nor is the notion of simultaneously searching with multiple algorithms a new idea. However, our work involves the first full investigation of the problem and technique as it applies to the area of suboptimal search. Below, we present some of the other work that is similar in kind or goal as our own.

Parallel Window Search (PWS) is a parallel version of IDA* (Powley and Korf 1991). In this algorithm, each processor performs an IDA* search with a different cost threshold. These cost thresholds are selected so as to correspond to a consecutive set of IDA* iterations. When a processor completes its iteration, it begins again with the next smallest available cost threshold. By simply returning the first solution found by any processor, PWS can be used to find suboptimal solutions.

PWS can be seen as a special case of parallel turn-taking as each processor can be thought of as being assigned an algorithm that performs a single iteration of IDA*, each with a different threshold. Since each iteration is a proper subset of all subsequent iterations, these searches will not be diverse. This lack of diversity explains why PWS with multiple processors outperforms a single-processor version of WIDA* (with weights selected so that the solution quality of the two algorithms is similar) in terms of search time but not in terms of total work.

EUREKA is a parallel search system designed to address the fact that different problems should be solved using different search parallelization techniques (Cook and Varnell 1997). For each problem, the EUREKA system builds a custom parallel version of IDA*. The system does so by collecting statistics during a breadth-first expansion of 100,000 nodes. These statistics are then fed into 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.

EUREKA is similar to turn-taking in that both techniques are motivated by the improvement found when properly using different choices on different problems. While EUREKA uses a classifier to construct a policy for selection, the underlying work function is expected to be relatively smooth in the choices made. Previously, we have demonstrated that not all algorithms exhibit this behaviour, and so classifiers will not be effective with all forms of parameterization.

Turn-taking is also related to the use of restarts in SAT solvers such as MiniSat (Eén and Sörensson 2003) and Chaff (Moskewicz et al. 2001). These solvers perform a depth-first like search where at each step, some variable is assigned a value. After a certain number of partial variable assignments are found to be invalid, the depth-first search restarts with all variables unassigned. Because SAT solvers can learn new constraints during search, restarts allow the decisions made near the root of the search tree to be more informed.

One of the motivations for restarts is the fact that it may take a long time to prove that choices made early in the depth-first search are poor. By restarting and instantiating with more information, this effect can be minimized. Instead of restarting the search, turn-taking approaches this problem by simultaneously searching with multiple algorithms so as to increase the probability that at least one of them will perform well - provided there is diversity in the candidate set.

Fast Downward is a automated planning system which uses a multi-valued state representation and a heuristic based on causal graphs (Helmert 2006). Part of this system is the multi-heuristic best-first search. In this search, there are two open lists, each of which is ordered by a different heuristic. The algorithm alternates between open lists when selecting nodes for expansion.

Both turn-taking and multi-heuristic best-first search simultaneously search the state-space with different choices. The main difference is that turn-taking completely separates the different algorithm variations while multi-heuristic best-first combines them. As this multi-heuristic enhancement is specific to the memory-intensive algorithm of A*, a comparison of this technique with turn-taking is beyond the scope of this paper.

## 4. Turn-Taking on WIDA*

In this section, we will explore the use of turn-taking and WIDA*. WIDA* is an important linear-space suboptimal algorithm because it is one of the few with provable bounds on the quality of the solutions. Any solution found by WIDA* is guaranteed to be no worse that $w$ times the optimal solution length.

Unfortunately, WIDA* searches can become stuck investigating large areas of the search space with no heuristic guidance. For example, consider a unit cost graph with uniform branching factor $b$ and a consistent heuristic function. In this domain, the heuristic value of a child $c$ an be at most 1 more than the heuristic value of the parent $p$. In this case, $f(c) = g(c) + w * h(c) = g(p) + 1 + w * (h(p) + 1) = g(p) + w * h(p) + w + 1 = f(p) + w + 1$. Similarly, the heuristic value can decreased by at most 1 in which case the child cost is $w - 1$ less than the parent cost.

Assuming the depth of the search tree is unbounded, no pruning will occur below any node $n$ until at least a depth of $\lfloor (T - f(n))/(w + 1) \rfloor$ below $n$, where $T$ is the cost threshold. Therefore, in the subtree of size $Q = b^{\lfloor (T - f(n))/(w+1) \rfloor}$ below $n$, WIDA* will have no heuristic guidance and no pruning will occur.

For any node $n$ with a low $h$, the value of $T - f(n)$ will be large, and consequently so will $Q$. Therefore, if a heuristic leads the search into an area of the state-space with low heuristic values but which actually is not near the goal, WIDA* must expand a large number of nodes before it can backtrack to $n$ in the search tree. Even if there is a goal near $n$, WIDA* may still have to expand a large number of nodes before finding it since the search has no guidance in the search tree of size at least $Q$. While these properties of WIDA* are concerning, the following results will demonstrate that the simultaneous use of different parameter values can help to mitigate these problems.

In order to test the use turn-taking on as many candidate sets as possible, most of the experiments were performed as follows: for a problem set $P$ and a set of algorithms $A$, the value of $exp(a, p)$ was determined for all $(a, p) \in A \times P$. After collecting this information, parallel turn-taking on a set $A' \subseteq A$ with $|A'|$ can be easily simulated by calculating $oracle(A', p)$ for each $p \in P$. The performance of sequential turn-taking can similarly be determined by multiplying the value of $oracle(A', p)$ on each problem by $|A'|$, and adjusting for algorithm ordering. By performing the experiments in this way, it is possible to efficiently determine the performance of turn-taking for each of the $|A|$ choose $k$ possible subsets $A' \subseteq A$ where $|A'| = k$ and $2 \leq k \leq |A|$.

For the experimental results shown in figures 3, 4, 6, and 7 the candidate set $A$ contains identical versions of WIDA* that only differ in the assigned weight. On each of these plots, the value of $batch(a, P)$ is shown for each $a \in A$. For these results, the x-axis refers to the value of the weight used in $a$. For example, a "Batch" point with x-value 3 corresponds to the batch result of WIDA* with a weight of 3.

Each of these figures also shows several results for turn-taking. For these results, the turn-taking result with x-value $k$ should be interpreted as the average number of nodes expanded over all $|A|$ choose $k$ combinations of the algorithms in $A$. As such, the x-axis refers to the number of algorithms in the candidate set. The parallel turn-taking results with x-value $k$ are also the average of all $|A|$ choose $k$ combinations of the algorithms in $A$. In this case, each point refers to the average total search time, and not average total work. Also note, in all of the following figures, the y-axis refers to nodes expanded and is shown in logarithmic scale.

For each of the following test domains there is a set of operators $O$, some subset of which is applicable in any state. In all the experiments, the children will be generated based upon a static ordering of $O$ set prior to search. In the case of WIDA*, the ordering is of particular importance since the order in which children are checked for expansion is the same as the order in which they are generated. As such, the operator ordering used in each experiment is noted.

Finally, note that although we will often compare the average performance of turn-taking with the performance of the algorithm $a_{best}$ with single best batch value, $a_{best}$ can only be found by offline parameter tuning. In the case of turn-taking, tuning can be used to find candidate sets with a high performance. However, we only present the average results so as to demonstrate that even in the absence of any pre-computation, turn-taking can often significantly outperform algorithms found with tuning.

## 4.1 Turn-Taking and WIDA* for the Sliding-Tile Puzzle

The sliding-tile puzzle is a standard single-agent benchmark domain, and we will assume the reader's familiarity with it. The static ordering of the operators used in all experiments is `down`, `right`, `left`, and `up`, where each operator corresponds to the direction a tile is slid into the empty space. The heuristic being used is that of manhattan distance.

Figures 3 and 4 show a comparsion of turn-taking with standard IDA* on the 15-puzzle and 24-puzzles respectively. For each puzzle, experiments were performed on 1000 randomly generated problem instances. In the case of the 15-puzzle, the average total number of nodes expanded by turn-taking reaches a minimum for candidate set sizes of 8 to 10, while the single best batch value occurs when the weight is 3. Even when using all 15 candidate algorithms, turn-taking expands 2.05 times fewer nodes than the weight of 3. As such, the search time required by parallel turn-taking when using 15 processors is 30.75 times faster than the single best batch value which represents a super-linear speedup on the number of processors over standard WIDA*.

For the 24-puzzle, the average number of nodes expanded reaches a minimum when using all 15 weights, and again the single best batch value occurs when the weight is 3. In this case, simultaneous turn-taking expands 42 times fewer nodes than the best batch value found by the weight of 3.

For the 35-puzzle, only weight 5 had completed a set of 100 randomly generated problems after several days of computation, and so turn-taking could not be simulated. Instead,

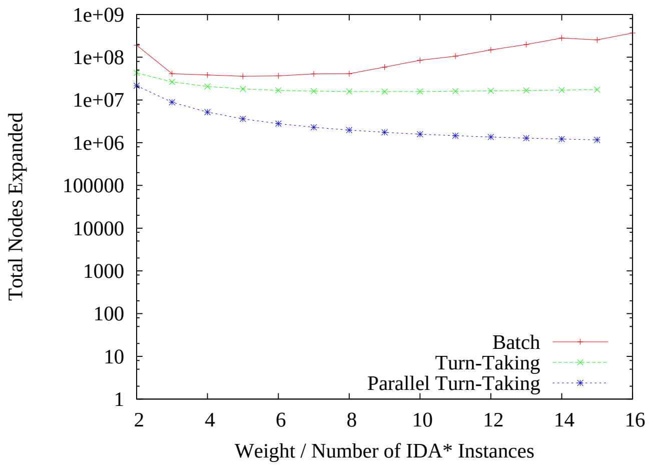

Figure 3: Regular and Parallel Turn-taking with WIDA* on the 15-Puzzle.

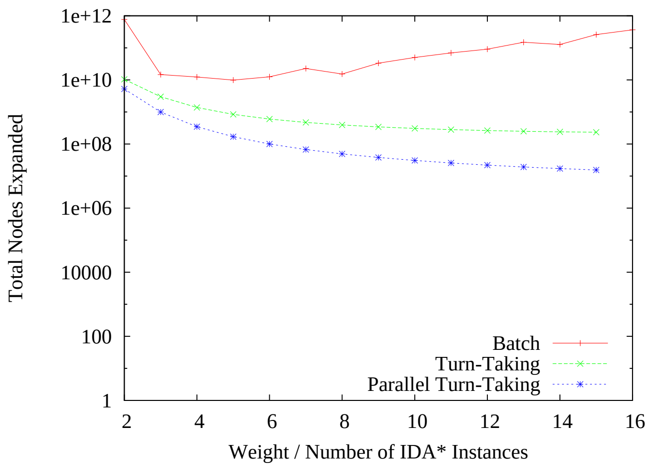

Figure 4: Regular and Parallel Turn-taking with WIDA* on the 24-Puzzle.

the actual turn-taking algorithm was run with the a single candidate set of 15 WIDA* instances each with an integer weight from 2 to 16. Figure 5 shows the search time needed by the weight of 5, turn-taking, and parallel turn-taking on each of the 100 puzzles. The puzzles have been sorted in increasing difficulty for weight 5. While the weight of 5 does outperform simultaneous turn-taking on a few of the problems, these are almost exclusively those problems that are easiest for the weight of 5. On the problems for which the performance of the weight of 5 is poor, turn-taking shows the most improvement. Over the entire problem set, turn-taking expanded 121 times fewer nodes than weight 5.

Note, the factor by which turn-taking improves over the best batch value increases with the puzzle size. This behaviour occurs because as the state-space becomes larger, the penalty for making a poor decision at any branch increases. Therefore, the importance of overcoming the mis-

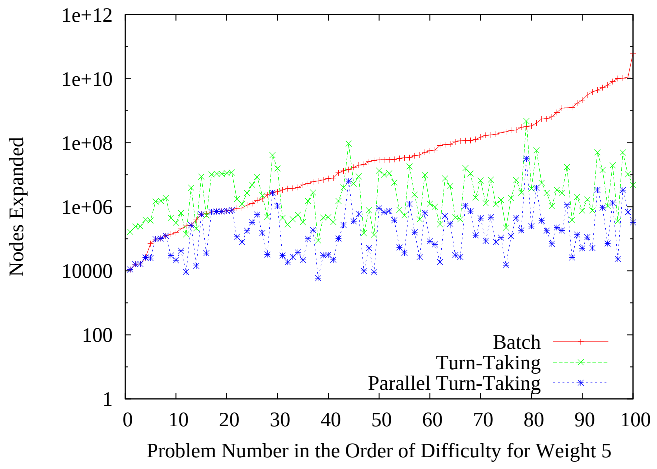

Figure 5: Regular and Parallel Turn-taking with WIDA* on the 35-Puzzle.

takes of individual algorithms is increased.

While WIDA* enhanced with turn-taking has been shown to exhibit impressive improvements in search efficiency in this puzzle, the average solution quality found with turn-taking does degrade slightly. In th 15 and 24-puzzles, the average solution length found by turn-taking were very similar to the weights of 8 and 9, respectively. For the 35-puzzle, the average solution length of turn-taking was 1.7 times worse than those found by the weight of 5. On individual problems, the solution quality could be much worse or better than the average of these weights, as the quality will depend on which algorithm found the solution.

## 4.2 Turn-Taking and WIDA* for the Pancake Puzzle

In this section, we discuss the application of turn-taking and WIDA* to the pancake puzzle. Each state in the $n$-pancake puzzle consists of a permutation of the numbers (or ”pancakes”) $0, 1, ..., n-1$. In any state, all $n-1$ operators, denoted by one of the integers from $2, 3, ..., n$, are applicable and of cost 1. The application of an operator $i$ reverses the order of the first $i$ elements in the permutation. The goal of the search is to sort the integers of the permutation in ascending order. In the experiments below, the operator ordering used is $n, n-1, ..., 2$. Experiments were performed on randomly generated test sets of 1000 problems.

For these experiments, the search was guided by pattern database heuristics(Culberson and Schaeffer 1996). We will use $PDB(p_0, p_1, p_2, ..., p_i)$ to denote the pattern database heuristic in which the pancakes $p_j$ for $0 \leq j \leq i$ are distinct in the abstract space.

In the 14-pancake experiment, $PDB(0, 1, 2, 3, 4, 5, 6)$ was used as the heuristic. The candidate set included 15 identical WIDA* each assigned a unique integer weight from the range 2 to 16. In this domain, the single best weight was 7 which expanded 1.6 times as many nodes as turn-taking.

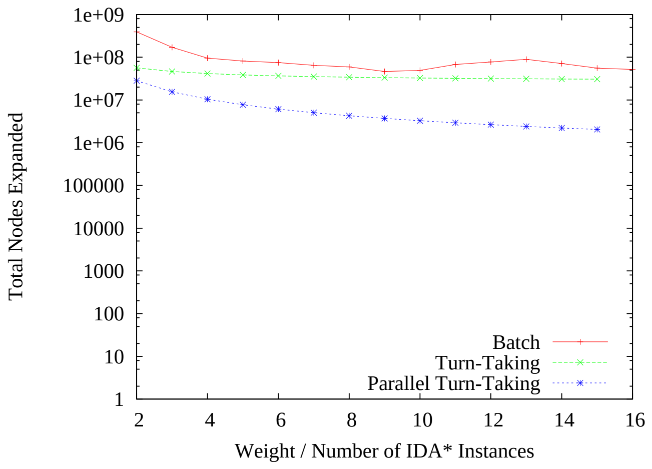

Figure 6: Regular and Parallel Turn-taking with WIDA* on the 14-Pancake Puzzle.

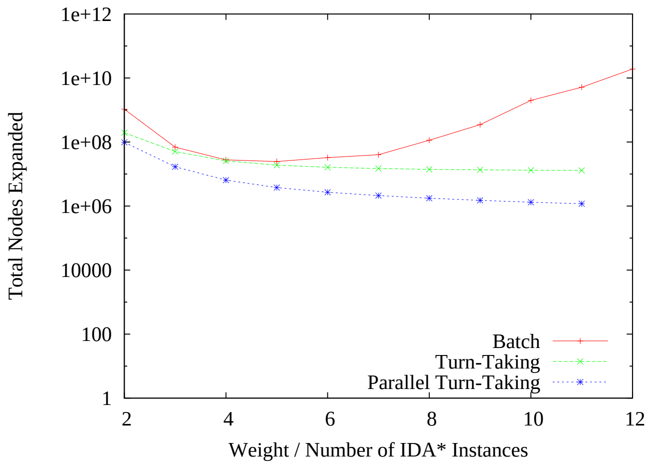

Figure 7: Regular and Parallel Turn-taking with WIDA* on the 16-Pancake Puzzle.

For the 16-pancake puzzle, the heuristic used was given by the maximization over $PDB(0, 1, 2, 3, 4, 5)$ and $PDB(6, 7, 8, 9)$. Unfortunately, this domain is too large to find all of the batch values of the weights from 2 to 16. In figure 7, we show the batch, turn-taking, and parallel turn-taking results for the integer weights from 2 through 12. In the case of turn-taking on the weights from 2 through 16, turn-taking improved upon the number of nodes expanded by the single best weight of 3 by a factor of 1.8.

In terms of solution quality on the 14-pancake puzzle, the average solution length found by turn-taking was similar to that found by weight 7. When using turn-taking over 15 WIDA* algorithms on the 16-pancake puzzle, the average solution quality was similar to the weight of 6.

On both pancake puzzles, sequential turn-taking performed slightly better than the single best batch value alone, and so parallel turn-taking again exhibited super-

| Weight | Expanded By Best Order | Expanded By Turn-Taking | Magnitude of Improvement |
|---|---|---|---|
| 3 | 273,782,770 | 49,318,913 | 5.6 |
| 4 | 246,230,595 | 16,968,531 | 14.5 |
| 5 | 411,505,075 | 11,092,577 | 37.1 |
| 6 | 441,818,307 | 10,280,071 | 43.0 |
| 7 | 487,919,186 | 10,216,976 | 47.8 |

Table 1: Turn-Taking on Operator Ordering in the 24-Puzzle.

linear speedup. Turn-taking does not perform as effectively in this domain as it does in the sliding tile puzzle in part because the graph width of this puzzle is smaller. This characteristic will result in shorter solution lengths and shallower search trees. As a result, the difference between the cost threshold and the f-cost of any node will never become too large, and so less blind search is performed.

### 4.3 Turn-Taking, WIDA* and Operator Ordering

Operator ordering is another design choice that can greatly impact the speed of a WIDA* search. In this section, we consider turn-taking over a set of identical WIDA* algorithms, each with the same weight but a different operator ordering.

In the sliding tile puzzle, turn-taking was tested with a candidate set of 24 WIDA* instances, each with one of the 24 possible operator orderings. Table 1 shows a summary of the results for 5 different weight values on 100 24-puzzle problems. The second column of this table shows the fewest number of nodes expanded by any single order for the indicated weight. The third column shows the total number of nodes expanded by sequential turn-taking on these problems. Note, these results are those for turn-taking over all 24 operator orderings, and not for any subset of the candidate set. The final column shows how many times fewer nodes are expanded by turn-taking than by the single best ordering for that weight.

Table 1 demonstrates that in this domain, turn-taking on operator ordering is quite effective. It should be noted that this improvement also occurs without any loss in solution quality. Table 1 also indicates that the magnitude of improvement increases with the weight. This behaviour occurs because the size of the last iteration grows as the weight increases, and so the importance of overcoming mistakes made by individual orderings is magnified.

In the 14-pancake puzzle, turn-taking was tested on 13 of the 13! possible orderings. So as to maximize the diversity of the searches, each ordering was assigned a different first operator. A random permutation of the remaining operators was then used to complete each ordering.

Turn-taking over all 13 orderings was similarly tested on weights 3, 4, 5, 6, and 7. Turn-taking required between 2 and 3 times fewer node expansions on all 5 of these weights with minimal change in the search quality. Again, due to the graph widths of the domains, turn-taking is much more effective in the sliding-tile puzzle.

## 5. Turn-Taking and Other Linear-Space Algorithms

In this section, we test the use of turn-taking with other linear-space search algorithms.

### 5.1 Weighted Recursive Best First Search (WRBFS)

RBFS is a linear-space algorithm in which nodes are first expanded in the same best-first order as A* (Korf 1993). The cost function that guides the search is also the familiar function from A*.

In WRBFS, a weight parameter is added to this cost function just as in WIDA*. As such, WRBFS also has the same bound on the suboptimality of the solution as does WIDA*.

For the 15-puzzle, the constructed candidate set contains 15 identical WRBFS instances, each with a different integer weight in the range from 2 to 16. Turn-taking over all algorithms in this candidate set was found to expand 1.67 times as many nodes as the single best batch value achieved with weight 3. While in this experiment turn-taking is not outperforming the single best batch weight, it remains competitive with this single weight and did so without any tuning. Similarly, while parallel turn-taking is not exhibiting super-linear speedup, it is still an effective form of parallelization as the total time is still decreasing by a factor of 9 when using 15 processors. Preliminary results do suggest that turn-taking with WRBFS also scales well to larger puzzles in which turn-taking again outperforms the best batch value and super-linear speedups return. However, even on the larger puzzles, the improvement from turn-taking pales in comparison to the improvement found in WIDA*. This effect is largely related to the best-first nature of WRBFS which decreases the amount of diversity between different weighted searches.

The performance of WRBFS with turn-taking on the pancake puzzle is actually quite poor due to anomolous behaviour by WRBFS in this domain. For example, in the case of the 14-pancake puzzles, with the $PDB(0, 1, 2, 3, 4, 5, 6)$ heuristic, all weights of size 10 and larger produce identical search trees. Aside from the smallest weights in the candidate set, the search trees are also quite similar. As such, even an oracle is only able to achieve very minimal gains in this domain.

### 5.2 Beam Search Using Limited Discrepancy Backtracking (BULB)

BULB is a beam-search variant that is capable of quickly finding solutions in very large state-spaces (Furcy and Koenig 2005a). Unlike the other two linear-space algorithms described, BULB does not have any guarantees on the suboptimality of the solution. In any case, here we provide preliminary results on the use of turn-taking with BULB.

Traditional beam-search is characterized by the parallel expansion of a set of open nodes. Once all the children have been generated, they are sorted by heuristic cost. The $B$ children with the lowest heuristic cost form the new open list, called the beam, and all other children are discarded. $B$ is referred to as the beam width. The process is repeated until the goal is found or memory is filled. As at most $B$ nodes at any depth are expanded, standard beam-search is an incomplete algorithm.

The BULB algorithm will initially perform a standard beam-search until memory is exhausted. BULB then applies a policy called limited discrepancy backtracking so as to make the algorithm complete. By using this style of backtracking, BULB first reconsiders decisions where the heuristic is expected to be the least accurate: near the root of the search tree.

For the experiments involving BULB and turn-taking, the candidate set contained identical BULB algorithms that differed only in the beam width. The beam widths included in the candidate set are as follows: $\{3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 75, 100, 200, 300\}$. Each BULB instance was also given a memory limit of 100,000 nodes.

In the case of the 35-puzzle, turn-taking over all 16 beam widths expands 8.3 times more nodes than the best batch beam width of 8. However, the average solution quality found by BULB improves upon this beam width by a factor of 2.2. The single beam width with the most similar solution quality to BULB with turn-taking was that of 30, which still expanded 2.5 times fewer nodes.

In the 16-pancake puzzle with the maximization over $PDB(0, 1, 2, 3, 4, 5)$ and $PDB(6, 7, 8, 9)$ as the heuristic, turn-taking showed similar behaviour even though no backtracking was required for any of the 100 problems tested. Turn-taking over all beam widths performed 4.9 times as much work as the single best beam width of 15 but the solution quality improved over this beam width by a factor of 2.61. The single beam width with the most similar solution quality to BULB with turn-taking was that of 50 in this case. This beam width also expanded 3.8 times fewer nodes than BULB enhanced with turn-taking.

These results indicate that turn-taking is less effective when used with the BULB algorithm than with the other two linear-space algorithms. In BULB, the beam widths that have the best batch values also seem to do very well on most of the individual problems. As such, an investigation into techniques for diversifying beam-search is needed before turn-taking can be used to effectively enhance this algorithm.

## 6. Conclusion

To the best of our knowledge, there has been only a limited discussion on the issue of proper parameter selection in the sub-optimal search literature. The standard approach to this problem has been to tune offline. In this paper, we have demonstrated that while tuning can find values with good average performance, there is no guarantee that these settings will perform well on every problem instance. As such, systems that properly select parameters on a problem-by-problem basis can outperform systems that rely on tuned parameters, often by a large margin.

In this end, we have proposed sequential turn-taking as an enhancement for suboptimal search algorithms. When turn-taking has been applied to WIDA*, it has been shown to outperform the best weight found by parameter tuning, sometimes by several orders of magnitude. We have also described a parallel version of this algorithm which can offer super-linear speedups over standard WIDA*. Finally, we have considered the application of turn-taking to other linear-space search algorithms and shown it to be a promising enhancement for WRBFS in certain domains.

While turn-taking can significantly improve the efficiency of search, open questions remain regarding how to determine the suitability of turn-taking for an algorithm or domain and how to use offline computation to find good sets of candidate algorithms. These topics are left as future work.

## References

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

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.

Eén, N., and Sörensson, N. 2003. An Extensible SAT-solver. In Giunchiglia, E., and Tacchella, A., eds., *SAT*, volume 2919 of *Lecture Notes in Computer Science*, 502–518. 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. 2005a. Limited Discrepancy Beam Search. In *IJCAI*, 125–131.

Furcy, D., and Koenig, S. 2005b. Scaling up WA* with Commitment and Diversity. In *IJCAI*, 1521–1522.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Kitamura, Y.; Yokoo, M.; Miyaji, T.; and Tatsumi, S. 1998. Multi-state commitment search. In *Tools for Artificial Intelligence*, 431–439.

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.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. In *Design Automation Conference*, 530–535. ACM.

Powley, C., and Korf, R. E. 1991. Single-Agent Parallel Window Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 13(5):466–477.

Zhou, R., and Hansen, E. A. 2005. Beam-Stack Search: Integrating Backtracking with Beam Search. In Biundo, S.; Myers, K. L.; and Rajan, K., eds., *ICAPS*, 90–98. AAAI.