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

When constructing a suboptimal single-agent search system, there are a number of decisions to be made that can significantly affect search efficiency. Each of these design decisions — including the selection of an algorithm, a heuristic function, parameter values, etc. — can greatly impact search speed. Following the work of others (Hutter, Hoos, and Stützle 2007) we refer to the set of choices made for an algorithm as the algorithm's *configuration*.

In practice, configurations are tested offline so as to find some single setting to be used in any future search. Unfortunately, tuning is an expensive process that is specific to each problem domain. Moreover, while a tuned system will perform well on average, there are often other configuratons with significantly better performance on certain problems. This is true even if we restrict the space of candidate configurations to those only varying in parameter values. For example, consider using weighted IDA* (WIDA*) (Korf 1993) to solve the standard 100 4×4 sliding tile puzzle problems (Korf 1985). From the set of weights $S = \{1, 2, ..., 25\}$, the weight (denoted w) of 7 was found to have the fastest performance on this problem set. However, on 82 of the 100 problems, there is a weight in S that expands less than half the nodes that $w = 7$ does, and on 7 of these problems, there is a weight that requires over a 100 fewer times the number of node expansions than the $w = 7$ configuration. In fact, if we could perfectly select the weight in S that performs best on each problem before beginning a search, search speed would improve by a factor of 25. These results demonstrate that correctly selecting configurations on a problem-by-problem basis can dramatically improve search speed.

Dovetailing has been shown to be an effective approach to this problem in several domains (Valenzano et al. 2010). Dovetailing is a technique from the parallel systems community that involves running a parallel algorithm on a single processor. For our purposes, we define the input of this strategy as a problem p, an algorithm a, and a set of configurations Θ for a, called the *candidate set*. As each configuration in Θ defines a unique instance of a, the candidate set can be thought of as a set of algorithm instances.

The dovetailing procedure consists of a number of rounds. Each round works as follows: each given algorithm instance 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. The process repeats until a solution is found.

A key component of dovetailing is that each algorithm instance performs a completely independent search. There is no memory shared between instances, and communication is restricted to messages indicating that a solution has been found for the current problem and the search should stop.

As each instance advances by a single step during each round, any instance in Θ will have performed approximately as much work as any other at any time. Therefore, the total problem-solving time when dovetailing on a problem p is approximately $|\Theta|$ times the problem-solving time of the candidate algorithm with the best performance on p. In the experiments presented in this paper, each algorithm step corresponds to exactly a single node expansion.

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. Again, 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 an algorithm a and a candidate configuration set Θ on a problem p is then given by the minimum time needed by any configuration in Θ to solve p.

The first test performed was of dovetailing over 15 WIDA* instances, where configurations only differ in the assigned weight. In the case of the 5×5 sliding-tile puzzle, the single WIDA* weight with the best performance over 1000 problems was the $w = 5$ configuration. Figure 1 shows the dovetailing improvement when compared to this weight. For each candidate set size k, the figure shows the ratio of the number of nodes expanded by the $w = 5$ configuration to the best of the $\binom{15}{k}$ candidate sets of containing k configurations, the worst of the candidate sets, and the average performance over all $\binom{15}{k}$ sets. This allows us to evaluate how robust dovetailing is with respect to candidate set selection. Wherever the value is greater than 1, dovetailing is outperforming the single configuration of $w = 5$ alone.

The figure indicates that dovetailing offers significant speedups. When the candidate set sizes reach 3 and 5, the

*This paper has been accepted to ICAPS 2010.

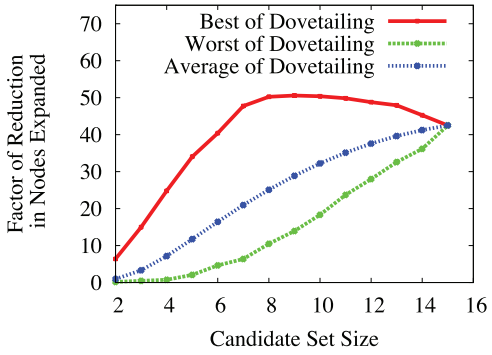

Figure 1: Dovetailing over weights in WIDA* on 1000 5×5 sliding-tile puzzles.

average and worst configurations, respectively, outperform even the single best configuration alone (*ie.* have a ratio greater than 1). An analogous set of experiments were run when using WRBFS (Korf 1993) with the same starting configuration set and on 1000 4×5 sliding-tile puzzle problems. While still evident, the speedups are not as dramatic as with WIDA*. For example, when the candidate set is of size 15, dovetailing uses 1.9 times fewer node expansions than the single best configuration of $w = 3$.

When dovetailing over candidate sets in which the configurations differ only in the operator ordering used, the speedups are similarly dramatic. With a candidate set containing 24 configurations, all with a weight of 5 but a different order, the speedup seen when dovetailing over WIDA* instances on the 5×5 sliding-tile puzzle is by a factor of 37.1 over the single best configuration. When all configurations have a weight of 10, the speedup factor is 142.5. Similar behaviour is seen when using WRBFS instead of WIDA*.

To determine the effectiveness of parallel dovetailing with WIDA* and WRBFS, the speedups reported above are 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 5 but a different operator ordering, results in a speedup of $37 \times 24 = 888$ over the single best ordering alone. As such, in most of the experiments parallel dovetailing results in a *super-linear speedup*, which occurs when the speedup is greater than the number of processors being used.

Dovetailing is less suitable for use with high-memory algorithms such as WA* and BULB (Furcy and Koenig 2005) since the memory demands grow linearly in the candidate set size. However, the parallel version of dovetailing can be useful when applied to these algorithms in a distributed memory system. For example, in the 4×4 sliding-tile puzzle domain, dovetailing with WA* over instances differing only in operator ordering performs comparably with a state-of-the-art parallelization of WA* known as wPBNF (Burns et al. 2009). In this domain, wPBNF outperforms parallel dovetailing when small weights are used. However, dovetailing scales much better to larger weights for which wPBNF is unable to achieve any speedups. For such weights, parallel dovetailing does not achieve super-linear speedups but it does effectively speed up the search. This performance is even more impressive when we consider that parallel dovetailing is very simple and easy to implement.

When parallel dovetailing is used with BULB, the speedups are modest when compared to the single fastest configuration. For example, when using 24 instances with a beam width of 7 but different operator orderings, the speedup factor is 3.8 on 7×7 sliding-tile puzzle problems. However, the solution quality improves substantially as the speedup is by a factor of 44 when compared to the beam width with the most similar solution quality.

Parallel dovetailing was also found to be a useful when used with an automated planner. 36 different configurations of the WA*-based Fast Downward planner (Helmert 2006) were constructed by varying the weight, the heuristic, and the use of preferred operators. Each configuration was given 30 minutes and 2 GB of memory to solve each of 846 problems taken from the satisficing track of the last planning competition. The average configuration solved 653 of problems and the best solved 746. When using parallel dovetailing over all 36 configurations in a distributed setting, 789 of the 846 problems were solved — 43 more than the single best configuration alone.

These experiments demonstrate the effectiveness of dovetailing, and its corresponding parallelization. Additional experiments indicate that speedups are also seen when the technique is used in the pancake puzzle domain with WIDA* or WRBFS (Valenzano et al. 2010). Parallel dovetailing has also been shown to offer super-linear speedups when used with these algorithms and has been shown to be an effective parallelization of WA* and BULB. In addition, it can be used to increase the number of problems that a WA*-based planner can solve. Therefore, dovetailing should be viewed as an attractive form of parallelization for such systems, and as an effective enhancement to the linear-space algorithms described above.

Acknowledgments

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

References

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

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

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.

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.

Valenzano, R.; Sturtevant, N.; 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.