Linear-Memory Beam Search Algorithms in Domain-Independent Dynamic Programming: Extended Abstract

Yuxiao Chen
University of Toronto, Canada

J. Christopher Beck
University of Toronto, Canada

1 Introduction

Domain-Independent Dynamic Programming (DIDP) is a model-based dynamic programming paradigm for combinatorial optimization [2]. In DIDP, a variety of heuristic search algorithms have been implemented to solve the state-based models that represents the optimization problems. In a benchmark of eleven problem classes, Complete Anytime Beam Search (CABS) was shown to outperform other heuristic search algorithms implemented in DIDP and state-of-the-art Mixed-Integer Programming and Constraint Programming solvers in terms of the number of optimal solutions found and memory usage [3, 5]. However, while CABS only stores two layers of states when solving problems with an acyclic state space, it still has an exponential memory usage in the worst case. To investigate the impact of memory complexity, we implement three linear-memory complete beam search algorithms in DIDP: two from the literature, Beam Stack Search (BSS) [6] and Beam search Using Limited discrepancy Backtracking (BULB) [1], and Beam search Using Depth-bounded Discrepancy Search (BUDDS) that is a novel adaptation of Depth-bounded Discrepancy Search (DDS) [4] to beam search. BUDDS can be understood as a DDS with each state replaced by a beam of B states, where B is the given beam width.

2 Results

In all experiments, we use the same benchmark of problem instances as Kuroiwa and Beck [3]. The benchmark contains 7,940 instances from 11 problem classes including bin packing, graph clear, traveling salesman problems with time windows, and talent scheduling problems. An experiment with a time limit of 30 minutes and a memory limit of 8 GB shows that CABS has more memory-out instances but finds better average optimality gap and more optimal solutions than each of the linear-memory algorithms in all problem classes. In detail, CABS solves a larger percentage of instances to optimality than each linear-memory algorithm in each problem class. In terms of the mean percentages across all problem classes, CABS, BSS, BULB, and BUDDS prove optimality in 60.89%, 52.24%, 50.18%, and 48.41% of the instances, respectively. In contrast, CABS exhausts the memory in a larger percentage of instances than each linear-memory algorithm in each problem class, except the Single Assembly Line Balancing Problem (SALBP). The linear-memory algorithms have larger initial beam width than CABS, and CABS either solves to optimality or reaches the time limit in most of the instances in SALBP before the beam width grows larger than the initial beam width of the linear-memory algorithms. The mean percentage of instances for which each approach exhausts memory is 2.54%, 0.44%, 0.07%, and 0.07%, respectively, for CABS, BSS, BULB, and BUDDS.

To test the algorithms in a memory-restricted environment, we run a second experiment with a 10-hour time limit and a 1-GB memory limit. We conduct exhaustive hyper-parameter tuning to set the beam width of each linear-memory algorithm and problem class to the value

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:3
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that results in the most instances solved to optimality for that algorithm and problem class. The result shows BSS, BULB, and BUDDS find more optimal solutions than CABS in 7, 6, and 4 out of 11 problem classes, respectively. In detail, CABS, BSS, BULB, and BUDDS prove optimality in 58.01%, 57.70%, 56.18%, and 55.28% of the instances in terms of the mean percentage over all problem classes. CABS still exhausts the memory in the largest percentage of instances in each problem class except in SALBP for the same reason in the first experiment; CABS, BSS, BULB, and BUDDS exhaust the memory in 39.77%, 11.60%, 20.13%, and 26.28% of the instances in terms of the mean percentage over all problem classes, respectively. Moreover, BSS finds more optimal solutions than CABS in all problem classes that CABS exhausts memory in more than 30% of the instances. Despite significantly more memory-out instances, CABS still proves the most optimal solutions and finds the best average optimality gap compared to all linear-memory algorithms. With a deeper analysis, only 12.97%, 7.60%, and 6.48% of the CABS memory-out instances are solved to optimality by BSS, BULB and BUDDS, respectively. On most of such instances, the linear-memory algorithms reach the time limit.

3 Discussion and Conclusion

CABS shows a clear advantage in optimality gap and proving optimal solutions for two main reasons. First, in the instances that both CABS and the linear-memory algorithms exhaust memory, CABS can find some feasible solutions before the memory limit is reached, since it starts with small beam width and memory usage. In contrast, the linear-memory algorithms require more memory than CABS to reach the maximum depth of the search tree to find the first feasible solution, but unlike CABS, the memory usage of the linear-memory algorithms does not increase after they reach the maximum depth of the search tree. If the linear-memory algorithms exhaust memory, that means they cannot reach the leaf nodes, so they terminate without any feasible solution.

Second, for the instances on which the linear-memory algorithms reach the time limit, the advantage of CABS in terms of optimality gap mostly comes from its global dual bound. In all DIDP solvers, a global dual bound is the minimum dual bound of all unexpanded states in minimization problems or the maximum dual bound in maximization problems. Beam search algorithms have unexpanded states in every layer due to the beam width. CABS repeatedly increases the beam width to expand states in all layers. In comparison, each linear-memory algorithm prioritizes a subset of layers to reduce the memory usage and state re-expansions. This prioritization results in a weaker global dual bound that is determined by the unexpanded states in non-priority layers.

Given the extensive tuning of the linear-memory algorithms and the memory restrictions that are much tighter than the standard on even low-powered computers, it is somewhat surprising that CABS still appears to be the best heuristic search algorithm in DIDP for the tested benchmarks: we expected the exponential memory complexity of CABS to have a greater negative impact. A promising direction may be to study the other side of the memory-computation trade-off: it may be beneficial for CABS to use additional memory to reduce the repeated node expansions it does in naively increasing its beam width.

References

1 David Furcy and Sven Koenig. Limited discrepancy beam search. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 125–131. Professional Book Center, 2005.

90 **2** Ryo Kuroiwa and J. Christopher Beck. Domain-independent dynamic programming: Generic
91 state space search for combinatorial optimization. *Proceedings of the International Conference*
92 *on Automated Planning and Scheduling*, 33(1):236–244, Jul. 2023. doi:10.1609/icaps.
93 v33i1.27200.
94 **3** Ryo Kuroiwa and J. Christopher Beck. Domain-independent dynamic programming. *Artificial*
95 *Intelligence*, 354:104506, May 2026. doi:10.1016/j.artint.2026.104506.
96 **4** Toby Walsh. Depth-bounded discrepancy search. In *IJCAI*, volume 97, pages 1388–1393, 1997.
97 **5** Weixiong Zhang. Complete anytime beam search. In *Proceedings of the Fifteenth Nation-*
98 *al/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*,
99 page 425–430. American Association for Artificial Intelligence, 1998.
100 **6** Rong Zhou and Eric A. Hansen. Beam-stack search: integrating backtracking with beam
101 search. In *Proceedings of the Fifteenth International Conference on International Conference*
102 *on Automated Planning and Scheduling*, page 90–98. AAAI Press, 2005.