Search Strategies for CODD

Eli M. Shattuck
School of Computing, University of Connecticut, Storrs, CT, USA

Laurent Michel
School of Computing, University of Connecticut, Storrs, CT, USA

Willem-Jan van Hoeve
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Dynamic programming solvers in existence today combine a modeling language with fixed search algorithms. While they allow model-specific heuristics, their ability to experiment with different search strategies remains limited. This paper introduces an extension to CODD that implements novel and classic search strategies, using the common modeling language of CODD. It also investigates an implicit diagram representation that reduces memory usage and improves performance. Experimental results on several classic combinatorial optimization problems demonstrate the importance of easily switching search strategies.

2012 ACM Subject Classification Mathematics of computing → Mathematical Software

Keywords and phrases Dynamic Programming, Decision Diagram, Search

Funding *Laurent Michel*: Synchrony Chair in Cybersecurity

1 Introduction

Combinatorial optimization problems can be modeled using dynamic programming (DP). These problems create exponential state spaces that are computationally challenging. Decision diagrams, when used together with relaxations and/or restrictions, aim to reduce the size of the diagrams and leverage techniques like branch and bound to derive scalable algorithms. The efficiency of such solvers depends both on the model and the type of search. This justifies the desire for a clean separation between the DP model and the search strategy. The CODD framework [8], implements a search based on branch and bound (B&B) and alternating application of relaxed and restricted diagrams.

This paper extends CODD with a suite of search strategies that all use the same modeling interface. The strategies range from being driven by relaxed diagrams (the original CODD [8]) to being driven by restricted diagrams It also explores foregoing the use of explicit edges in the diagram in a spirit reminiscent of [12]. Finally, we included a CODD-based implementation of CABS [7] to assess the competitiveness of our alternatives.

2 The Dynamic Programming Model

Optimization problems modeled with dynamic programming in frameworks such as DIDP [7], DDO [5], or CODD [8] require the definition of a state, a label generation function, a state transition function, a cost function to price transitions, and, optionally, a state merge function. The state and the functions are then used to instantiate a solver that uses the Bellman equations [2] to capture a labeled transition system. A solver then explores the labeled transition system according to some computational strategy. The objective is to find a path root-to-sink that (w.l.o.g.) minimizes the total cost of transitions. DIDP includes several strategies: A^* [6], CAASDy [7], and CABS [15]. CODD explores a strategy driven by the exact cut set extracted from the relaxed diagram used to produce dual bounds within a B&B

framework. *The purpose of this paper is to investigate alternative strategies and assess their effectiveness compared to the state-of-the-art.* In particular, the paper focuses on strategies that use restricted diagrams to populate the B&B queue.

Restricted Diagrams Restricted diagrams represent a subset of solutions by bounding the width w in any layer (all states equidistant from the root). It achieves this by discarding states whenever the limit is exceeded. They excel at delivering primal bounds.

Relaxed Diagrams Likewise, relaxed diagrams represent a superset of solutions within a given width w in any layer, by *merging* states using a merging operator $\oplus$. This leads to a relaxation, as it can create additional root-to-sink paths that are not truly feasible. The strength of the relaxation depends on w. Naturally, the states to merge also impact the relaxation. Overall, relaxed diagrams deliver dual bounds.

Integrating Restricted and Relaxed into a Search By themselves, restricted and relaxed are incomplete and should be embedded into a search to restore completeness. Two possible frameworks are *iterative widening* and *branch & bound*. Iterative widening progressively increases w until the DD becomes exact. CABS within DIDP [7] is the classic representative. *Branch & bound* incrementally investigates promising states that were unexplored by their width-limited diagrams. DDO and CODD fall in that category.

3 Search Strategies

CODD implemented one strategy. It populated the branch & bound queue from a cut set defined as the collection of exact states that parent a merged state. Restricted diagrams were used to vet states pulled from the queue and opportunistically deliver primal bounds. Details can be found in [8]. This strategy is referred to as relaxed-first (`XF`) below. The remainder of this section explores three strategies.

Restricted First (`RF`) `RF` follows the same pattern as `XF`. Yet, it populates the queue using the restricted diagram. Namely, it collates in a discarded set the states that would have exceeded w. Secondly, it uses the relaxed diagram to achieve dual-bound filtering. This strategy tends to produce smaller relaxed diagrams and reduces the number of calls to $\oplus$. Furthermore, `RF` can be made more efficient with an *aggressive* merging strategy that keeps on merging even when the desired width has been reached. Since the cut set is not needed, the relaxed graph can be made smaller by attempting to merge all states in the same layer.

Restricted Only (`RO`) `RO` is `RF` without any reliance on relaxed diagrams. This weakens (or fully removes) dual bounds but avoids the extra cost of merging or the need for $\oplus$. This is similar to the approach taken by [9].

Restricted Only With no B&B Queue (`RONQ`) This variant (`RONQ`) builds on `RO`, but adopts iterative widening like CABS. For a given width budget, it builds a restricted diagram to deliver a primal solution. If the diagram is inexact, the process is repeated with twice the budget. When no states are discarded, the diagram is exact, and its solution offers a proof of optimality. Clearly, `RONQ` is identical to CABS, yet framed in terms of decision diagrams.

Memory-Efficient Representation We adopt *implicit* diagrams where edges are absent. The motivation is driven by the compact formulation adapted to GPUs as presented in [14]. It is also featured in [12]. The choice delivers a compact encoding, but mandates changes to the computation of cut sets and dual bounds during the construction of relaxed diagrams (restricted is trivially adapted). In addition to space, the benefit is a runtime reduction.

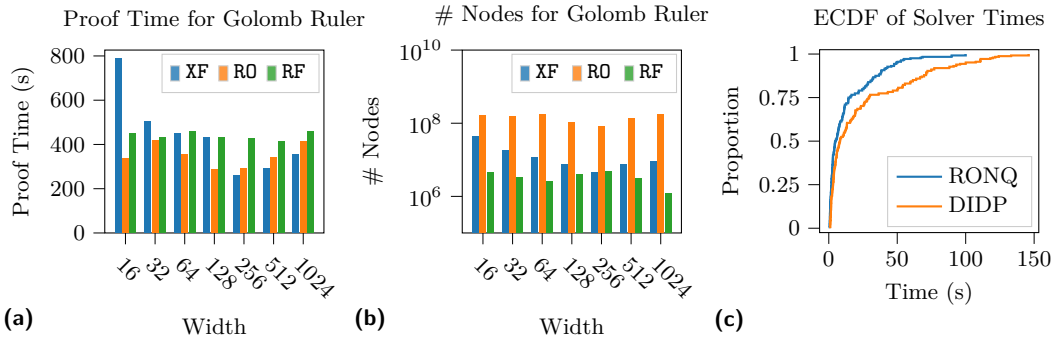

Figure 1 Results for Golomb ruler with 13 marks, and TSPTW (See Table 4 and 5 for full list of instances).

4 Experiments

We tested the four search strategies outlined above on four classic combinatorial optimization problems. In all experiments, the model is the same; we only vary the search strategy.

Maximum Independent Set Problem (MISP) Results are cataloged in Table 1. Benchmarks where all strategies finished in less than 0.1 seconds are omitted. The strategies `RF` and `RO` are dominant, beating `XF` in all instances tested.

Graph Coloring Results are shown in Table 2. Benchmarks where all strategies finished in less than 0.1 seconds are omitted. The strategies `RO` and `RF` are very close on easier instances, but on the hardest instance tested (`DSJC125.1`) `RO` struggles to scale, while `RF` is around 10 times faster. `XF` wins on only one benchmark, and struggles on some easy instances compared to the other strategies, but scales better than `RO` at around 3 times faster.

Traveling Salesman Problem with Time Windows (TSPTW) TSPTW is a problem that greatly benefits from an iterative widening. Using the `RONQ` strategy, CODD is competitive with DIDP-CABS. Figure 1c shows an empirical cumulative distribution plot comparing TSPTW performance for `RONQ` and DIDP. The plot shows the proportion of instances that can be solved in a given amount of time. On this dataset, CODD has a worst case of around 100s, and DIDP has a worst case of around 140s.

Golomb Ruler All three strategies perform similarly at $w = 64$ as seen in Table 3. Although `RO` uses more nodes than `RF` and `XF` it is still the fastest by a small margin. For a fixed instance (13 marks), we can also evaluate how the strategies vary with width. Figure 1a indicates that `XF` has the best potential time. However, the optimal width cannot be known beforehand, so `RO` may still be preferred since it is more stable across many widths.

5 Conclusion

We introduced two restricted-based searches and one iterative widening-based search for CODD. These strategies all use the same modeling language, allowing users to easily switch strategies without changing their model. We showed that, on several problem classes, these strategies are competitive with the relaxation-based strategy from CODD and with other state-of-the-art solvers. We also adopted an edgeless, memory-efficient decision diagram that increases performance across the board. This extension makes CODD more flexible for a wide array of problem classes.

References

1. Norbert Ascheuer. *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems*. PhD thesis, 1996.
2. Richard Ernest Bellman. *The Theory of Dynamic Programming*. RAND Corporation, Santa Monica, CA, 1954.
3. Yvan Dumas, Jacques Desrosiers, Marius Solomon, and Éric Gélinas. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43:367–371, 04 1995. doi:10.1287/opre.43.2.367.
4. Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Oper. Res.*, 46(3):330–335, March 1998. doi:10.1287/opre.46.3.330.
5. Xavier Gillard, Pierre Schaus, and Vianney Coppé. Ddo, a generic and efficient framework for mdd-based optimization. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 5243–5245. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Demos. doi:10.24963/ijcai.2020/757.
6. Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.
7. Ryo Kuroiwa and J. Christopher Beck. Domain-independent dynamic programming. *Artificial Intelligence*, 354:104506, 2026. URL: https://www.sciencedirect.com/science/article/pii/S0004370226000329, doi:10.1016/j.artint.2026.104506.
8. Laurent Michel and Willem-Jan van Hoeve. CODD: A decision diagram-based solver for combinatorial optimization. In *ECAI*, Frontiers in Artificial Intelligence and Applications, pages 4240–4247. IOS Press, 2024.
9. Ryan J. O'Neil and Karla Hoffman. Decision diagrams for solving traveling salesman problems with pickup and delivery in real time. *Operations Research Letters*, 47(3):197–201, 2019. URL: https://www.sciencedirect.com/science/article/pii/S0167637718305364, doi:10.1016/j.orl.2019.03.008.
10. Jean-Yves Potvin and Samy Bengio. The vehicle routing problem with time windows - part ii: Genetic search. *INFORMS Journal on Computing*, 8, 03 2002. doi:10.1287/ijoc.8.2.165.
11. Omar Rifki and Christine Solnon. On the phase transition of the euclidean travelling salesman problem with time windows. *Journal of Artificial Intelligence Research*, 82:2167–2188, 04 2025. doi:10.1613/jair.1.18334.
12. Isaac Rudich and Louis-Martin Rousseau. Implicit decision diagrams, 2026. URL: https://arxiv.org/abs/2602.20793, arXiv:2602.20793.
13. Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35:254–265, 1987. URL: https://api.semanticscholar.org/CorpusID:15346313.
14. Fabio Tardivo, Laurent Michel, and Willem-Jan van Hoeve. Gpu-accelerated relaxed decision diagrams for branch-and-bound optimization. In *32nd International Conference on Principles and Practice of Constraint Programming (CP 2026)*, 2026. To appear.
15. Weixiong Zhang. Complete anytime beam search. In *Proceedings of the Fifteenth National/-Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, page 425–430, USA, 1998. American Association for Artificial Intelligence.

A Additional Results

Benchmark	Opt	XF(s)	RO(s)	RF(s)	CODD (s)
brock200_1	21	**395.53**	397.02	557.63	709.35
brock200_2	12	0.74	**0.38**	0.49	1.28
brock200_3	15	5.69	**4.23**	5.70	13.75
brock200_4	17	**16.17**	20.70	28.24	35.94
hamming6-2	32	**0.09**	0.28	0.28	0.013
hamming8-4	16	**8.00**	65.19	81.69	12.16
keller4	11	6.00	**5.65**	7.62	13.50
p_hat300-2	25	**73.01**	103.18	140.06	136.94

Table 1 MISP results ($w = 128$). Best time in bold. CODD's times use explicit edges from [8].

Benchmark	Opt	XF	RO	RF
2-FullIns_3	5	1.13	**0.16**	0.31
2-Insertions_3	4	0.91	**0.47**	0.53
DSJC125.1	5	48.32	165.99	**16.13**
anna	11	0.66	**0.34**	0.35
david	11	0.21	**0.11**	**0.11**
jean	10	0.16	**0.11**	**0.11**
miles250	8	0.27	0.15	**0.14**
miles500	20	1.33	**0.68**	**0.68**
miles750	31	1.96	**1.18**	1.30
queen7_7	7	0.86	**0.69**	**0.69**
r125.1c	46	**0.04**	1.20	0.26
school1	14	4.43	**1.12**	1.19

Table 2 Results for graph coloring ($w = 64$). Best time shown in bold.

# Marks	Opt	XF		RO		RF	
		Time (s)	# Nodes	Time (s)	# Nodes	Time (s)	# Nodes
8	34	**0.02**	548	0.06	8,198	0.04	306
9	44	**0.07**	3,335	0.27	22,007	0.24	1,915
10	55	**0.44**	12,268	0.78	126,275	1.21	2,694
11	72	6.32	230,515	**4.44**	1,700,144	8.85	55,598
12	85	20.69	406,981	**14.90**	7,055,891	27.20	182,774
13	106	426.81	11,434,923	**396.37**	170,043,265	408.55	2,474,553

Table 3 Results for Golomb ruler ($w = 64$). Best time shown in bold.

AFG [1]	Dumas [3]	GendreauDumas [4]	
rbg021.6	n100w60.001	n100w100.001	n40w180.005
rbg021.6	n100w60.001	n100w100.001	n40w180.005
rbg021.7	n150w40.001	n100w100.002	n40w200.001
rbg021.7	n150w40.001	n100w100.002	n40w200.001
rbg048a	n150w40.004	n100w100.003	n40w200.003
rbg048a	n150w40.004	n100w100.003	n40w200.004
rbg049a	n150w40.005	n100w100.004	n40w200.004
rbg049a	n150w40.005	n100w100.004	n60w120.001
rbg050b	n150w60.001	n100w120.004	n60w120.001
rbg050b	n150w60.001	n100w80.001	n60w120.002
rbg132.2	n150w60.002	n100w80.001	n60w120.002
rbg172a	n150w60.002	n100w80.002	n60w120.003
rbg172a	n150w60.003	n100w80.002	n60w120.003
rbg193	n150w60.003	n100w80.003	n60w120.004
rbg193	n150w60.004	n100w80.003	n60w120.004
rbg201a	n150w60.004	n100w80.004	n60w120.005
rbg201a	n150w60.005	n100w80.004	n60w120.005
rbg233	n150w60.005	n100w80.005	n60w140.001
rbg233	n200w40.001	n100w80.005	n60w140.001
SolomonPotvinBengio [10]	n200w40.001	n40w140.002	n60w140.002
rc_203.2	n200w40.002	n40w140.002	n60w140.002
rc_203.2	n200w40.002	n40w140.004	n60w140.003
rc_204.3	n200w40.003	n40w140.004	n60w140.005
rc_204.3	n200w40.003	n40w140.005	n60w140.005
rc_207.1	n200w40.004	n40w140.005	n60w160.001
rc_207.1	n200w40.004	n40w160.001	n60w160.004
rc_207.2	n200w40.005	n40w160.001	n60w160.005
rc_207.2	n200w40.005	n40w160.002	n80w100.001
rc_207.3	n60w100.004	n40w160.002	n80w100.001
rc_207.3	n60w100.004	n40w160.003	n80w100.002
SolomonPesant [13]	n80w60.002	n40w160.003	n80w100.002
rc207.0	n80w60.002	n40w160.005	n80w100.003
rc207.0	n80w80.002	n40w160.005	n80w100.003
rc207.1	n80w80.002	n40w180.001	n80w100.004
rc207.1	n80w80.003	n40w180.001	n80w100.004
rc207.2	n80w80.003	n40w180.003	n80w120.002
rc207.2	n80w80.004	n40w180.003	n80w120.003
	n80w80.004	n40w180.004	n80w120.003
		n40w180.004	n80w120.005

Table 4 Instances used for TSPTW

Solnon [11]			
n21g100b50.004	n31g60b40.004	n41g100b40.005	n51g100b30.003
n21g100b50.004	n31g60b40.005	n41g100b40.005	n51g100b30.004
n21g60b50.001	n31g60b40.005	n41g60b30.001	n51g100b30.004
n21g60b50.001	n31g60b50.005	n41g60b30.002	n51g100b30.005
n31g100b40.005	n31g60b50.005	n41g60b30.002	n51g60b20.001
n31g100b40.005	n31g80b40.001	n41g60b30.003	n51g60b20.001
n31g100b50.001	n31g80b40.001	n41g60b30.003	n51g60b20.002
n31g100b50.001	n31g80b40.002	n41g60b30.004	n51g60b20.002
n31g100b50.002	n31g80b40.002	n41g60b30.004	n51g60b20.003
n31g100b50.002	n31g80b40.003	n41g80b30.001	n51g60b20.003
n31g100b50.003	n31g80b40.003	n41g80b30.001	n51g60b20.004
n31g100b50.003	n31g80b40.005	n41g80b30.002	n51g60b20.004
n31g100b50.004	n31g80b40.005	n41g80b30.002	n51g60b20.005
n31g100b50.004	n31g80b50.002	n41g80b30.003	n51g60b20.005
n31g100b50.005	n31g80b50.002	n41g80b30.003	n51g80b20.001
n31g100b50.005	n31g80b50.003	n41g80b30.004	n51g80b20.001
n31g60b30.001	n31g80b50.003	n41g80b30.004	n51g80b20.005
n31g60b30.001	n41g100b40.001	n41g80b30.005	n51g80b20.005
n31g60b30.003	n41g100b40.001	n41g80b30.005	n51g80b30.001
n31g60b30.003	n41g100b40.002	n51g100b30.001	n51g80b30.001
n31g60b30.005	n41g100b40.002	n51g100b30.001	n51g80b30.002
n31g60b30.005	n41g100b40.003	n51g100b30.002	n51g80b30.002
n31g60b40.002	n41g100b40.004	n51g100b30.002	n51g80b30.003
n31g60b40.002	n41g100b40.004	n51g100b30.003	n51g80b30.003
n31g60b40.004			

Table 5 Instances used for TSPTW (Continued.)