WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Next-Generation Dynamic Programming Algorithms: A Survey of Modern Techniques, Applications, and Future Directions

Gorasiya Archit Jayeshbhai

Department of Computer Science and Engineering, Gyanmanjari Innovative University, Bhavnagar, Gujarat, India

ABSTRACT—

Dynamic programming (DP) has anchored sequential optimization since Bellman's work in the 1950s, but its tabular form breaks down once the state space becomes large, continuous or irregular. This survey follows how the field has answered. We sort recent work into five paradigms, each of which relaxes a different assumption of classical DP: abstract DP, which studies the Bellman operator instead of a particular problem; graph-aware DP, which prunes the state space with problem structure, as in join-order optimization; neuro-dynamic and approximate DP, which replace tables with parametric value functions; parallel DP on multicore and GPU hardware; and quantum DP for exponential-time recurrences. Beyond cataloguing methods, we compare their guarantees and failure modes and back the discussion with four small computational studies of our own: measured contraction rates of value iteration, exhaustive counts of the join pairs examined by DPsub and DPccp (915 times fewer on a twelve-relation chain), a reproduction of Baird's divergence example, and a wavefront model with a measured CPU baseline. A closed-form comparison of reported quantum speed-ups completes the picture. Applications in query optimization, robot control, energy-storage dispatch and sequence analysis are reviewed, and we close with open problems: convergence under nonlinear approximation, sample efficiency, QRAM limits and the absence of shared benchmarks.

***Index Terms*—Dynamic programming, curse of dimensionality, abstract dynamic programming, neuro-dynamic programming, join-order optimization, GPU acceleration, quantum algorithms, value iteration.**

I. Introduction

A. Motivation: Tables That Do Not Fit in Memory

Richard Bellman introduced dynamic programming in the 1950s as a way to attack multistage decision problems by working through a chain of smaller ones [1]. The idea hangs on the principle of optimality: whatever the first decision was, the remaining decisions of an optimal plan must be optimal for the state that decision produced. Over seven decades the principle has paid off in settings that have little else in common, among them protein sequence comparison [2], [3], RNA folding [4], query planning in relational databases [5] and inventory and maintenance control [6].

The classical implementation stores one number per state, and that is also where it fails. A table over d state variables with k values each has k^d entries (Fig. 1); Bellman called this effect the curse of dimensionality [1]. Three further obstacles show up in practice. Continuous states cannot be tabulated without discretization, which brings the curse back. Naive enumeration of subproblems spends time on combinations that no valid solution contains; join ordering with cross products is the textbook case [7]. And the standard recursion is sequential, which sits badly with throughput-oriented hardware such as GPUs and, for different reasons, with quantum circuits [8].

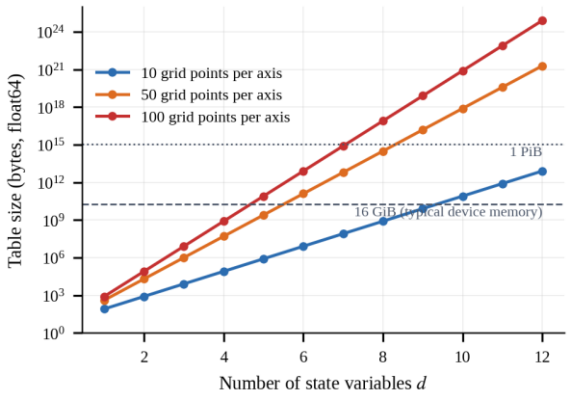


Fig. 1. Memory needed to store a tabular value function (8 bytes per entry) as the number of state variables grows, for three grid resolutions. Computed directly from $k^d \times 8$ bytes; no measurement is involved.

Each obstacle has its own literature, and those literatures are scattered across control theory, databases, reinforcement learning, parallel computing and quantum information. Authors working in one area rarely cite another. This survey puts them next to each other and asks a plain question of every method: which assumption of tabular DP does it give up, and what does it get back in return?

B. Scope and Survey Methodology

We cover work from Bellman's original formulation up to early 2026, with emphasis on the last decade. Candidate papers were collected through keyword searches on scholarly indexes and on publisher, proceedings and preprint pages, using combinations such as "dynamic programming join ordering", "approximate dynamic programming convergence", "GPU dynamic programming wavefront" and "quantum dynamic programming". An item was kept only when its bibliographic details could be confirmed on such a page; standard textbooks are cited from their published editions. Claims that come from a single preprint are described as such, and where a result has been corrected in print (see Section V-B) we say so.

A survey that only repeats published numbers can hide how fragile some of them are. We therefore ran four small studies ourselves, each cheap enough to repeat on a laptop: value iteration on random Markov decision processes (Section IV), exhaustive enumeration of join pairs (Section V), Baird's counterexample (Section VI) and an anti-diagonal parallelism model with a CPU baseline (Section VII). Their code and settings are listed in Appendix B. Timings come from a single-core 2.8 GHz Xeon virtual machine and are indicative only; no GPU or quantum hardware was used, and no figure in this paper claims otherwise.

C. Contributions

1) a taxonomy of modern DP methods organized by *which assumption of the tabular algorithm they relax* (Section III, Fig. 2), with a dated timeline (Table I);
2) a unified treatment of convergence through the operator view, checked numerically on random MDPs (Section IV);
3) a quantified account of search-space reduction in graph-aware DP, validated by exhaustive enumeration up to twelve relations (Section V);
4) a reproduction of divergence under off-policy function approximation and a discussion of what stabilizes deep value-based methods in practice (Section VI);
5) a wavefront-parallelism model for GPU-style DP with a measured CPU baseline, and a summary of reported quantum speed-ups with their assumptions (Sections VII and VIII);
6) a cross-paradigm comparison, an application review and a list of open problems (Sections IX–XI).

D. Organization

Section II recalls the foundations. Section III introduces the taxonomy. Sections IV–VIII treat the five paradigms in turn. Section IX compares them, Section X reviews applications, Section XI discusses challenges and the limits of this survey, and Section XII concludes.

II. Background and Foundations

A. The Principle of Optimality and the Bellman Equation

Consider a discounted Markov decision process with states x, controls $u \in U(x)$, stage cost $g(x, u)$, transition probabilities $P_{xy}(u)$ and discount factor $\gamma \in [0, 1)$. The optimal cost-to-go J^* satisfies the Bellman equation [1], [6]

$$J^*(x) = \min_{u \in U(x)} \{ g(x, u) + \gamma \sum_y P_{xy}(u)\, J^*(y) \} \qquad (1)$$

Solving this equation by repeated substitution gives value iteration; alternating policy evaluation with greedy improvement gives policy iteration [6], [9]. With dense transitions one sweep costs $O(|S|^2|A|)$ operations, which is harmless for a few thousand states and hopeless for 10^{12}.

B. Exact DP over Subsets

Some problems have exponentially many states by nature and are still solved exactly with DP. The Bellman–Held–Karp algorithm for the travelling salesman problem [10], [11] keeps one value $C(S, j)$ for every subset S of cities and every end city $j \in S$:

$$C(S, j) = \min_{i \in S \setminus \{j\}} \{ C(S \setminus \{j\}, i) + d(i, j) \} \qquad (2)$$

The table has $n2^n$ entries and the total work is $O(n^2 2^n)$. Join ordering uses the same subset structure (Section V), and the quantum algorithms of Section VIII target exactly this family. Sequence alignment [2], [3] sits at the other extreme: a table of only mn cells whose dependency pattern, rather than its size, is the interesting part (Section VII).

C. Why Tables Stop Scaling

Fig. 1 shows the arithmetic. With ten grid points per axis, ten state variables already need about 75 GiB for the values alone. With a hundred points per axis, five variables are enough to pass 16 GiB, which is the memory of a mid-range accelerator. Time is no kinder than space, since every sweep touches every entry. This is why the four families of methods that follow do not simply try harder at the same table; they change what is stored, what is enumerated, or where the computation runs.

III. A Taxonomy of Modern DP Methods

We separate methods by the assumption of tabular DP they relax, because that choice determines what is lost. Fig. 2 summarizes the result. Abstract DP relaxes the *operator*: instead of one concrete Bellman equation it studies the properties that make value iteration converge. Graph-aware DP relaxes the *state space*: it enumerates only subproblems that can occur in a valid solution. Neuro-dynamic and approximate DP relax the *value table* itself and store a parametric function. Parallel DP relaxes *serial execution* while leaving the recurrence untouched. Quantum DP relaxes the *classical hardware model*,

and pays for that in assumptions about quantum memory.

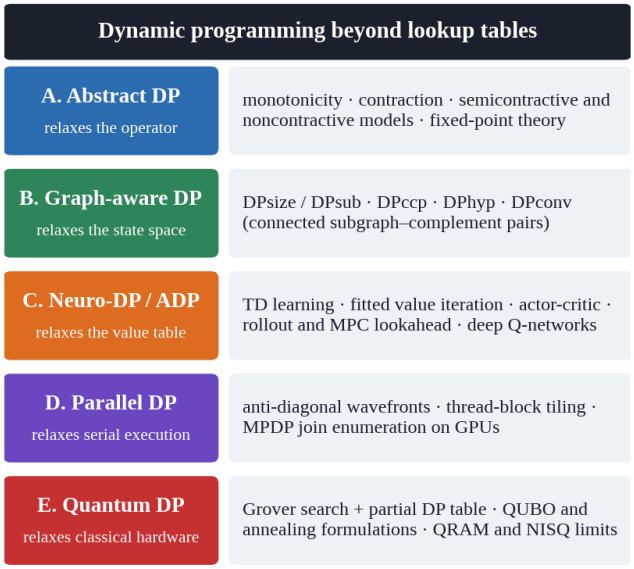


Fig. 2. Taxonomy used in this survey. Each paradigm is defined by the aspect of tabular DP it relaxes; the right-hand boxes list representative methods discussed in Sections IV–VIII.

The categories are not exclusive. Massively parallel join enumeration [12] is both graph-aware and parallel, and plan-online/learn-offline control [13] mixes lookahead search with a learned value function. Table I places the main milestones on a common time line.

TABLE I

MILESTONES IN THE DP LITERATURE

Year	Milestone	Paradigm
1957	Principle of optimality; curse of dimensionality [1]	Foundation
1962	Subset DP for the travelling salesman problem [10], [11]	Exact
1967	Contraction-mapping theory of DP [14]	Abstract
1979	DP join-order optimization [5]	Graph-aware
1996	Neuro-dynamic programming [15]	Neuro-DP
1997	TD learning with linear approximation analyzed [16]	Neuro-DP
2006	DPccp: csg-cmp pair enumeration [7]	Graph-aware
2008	DPhyp: join ordering on hypergraphs [17]	Graph-aware
2013–22	Abstract DP monograph, semicontractive models, 3rd ed. 2022 [18]	Abstract
2015	Deep Q-networks [19]	Neuro-DP
2019	Grover search combined with subset DP [8]; plan online, learn offline [13]	Quantum; Neuro-DP
2022	MPDP, GPU join enumeration [12]	Parallel
2023	Join ordering on quantum hardware [20]	Quantum
2024	DPconv: subset convolution for join ordering [21]	Graph-aware
2026	Quantum time–space trade-offs for subset DP [22]	Quantum

IV. Abstract Dynamic Programming

A. The Operator View

Abstract DP, developed most fully by Bertsekas [18] and rooted in Denardo's contraction-mapping analysis [14], moves the object of study from a problem to a mapping. Write the Bellman update as an operator on cost functions,

$$(TJ)(x) = \min_{u \in U(x)} \{\, g(x, u) + \gamma \sum_{y} P_{xy}(u)\, J(y) \,\}. \qquad (3)$$

Two structural properties then carry most of the theory. The first is monotonicity:

$$J_1 \le J_2 \implies TJ_1 \le TJ_2. \qquad (4)$$

It guarantees that the ranking of policies is preserved from one iteration to the next. The second is contraction in the sup norm:

$$\|TJ_1 - TJ_2\|_\infty \le \gamma \, \|J_1 - J_2\|_\infty. \qquad (5)$$

By Banach's fixed-point theorem a contraction has a unique fixed point J^*, and value iteration converges to it from any starting function. The point of the abstract view is that nothing in (4) or (5) refers to probabilities or to a particular cost structure. Discounted, stochastic-shortest-path, minimax and risk-sensitive problems all fit once an appropriate operator is written down, and so do approximation schemes, which can be analyzed as perturbations of the same operator [18].

B. Rate of Convergence, Checked Numerically

Under (5), the iterates $J_{k+1} = TJ_k$ obey

$$\|J_k - J^*\|_\infty \le \gamma^k \, \|J_0 - J^*\|_\infty, \qquad (6)$$

so reaching a tolerance ε takes about $\ln(\varepsilon / \|J_0 - J^*\|) / \ln \gamma$ sweeps. We checked this on a randomly generated MDP with 60 states and 4 actions (costs uniform on [0, 1], seed 7) for $\gamma \in \{0.5, 0.8, 0.9, 0.95, 0.99\}$. In every run the one-step inequality (5) held at each of the 200 sweeps with no violation, and the measured error followed the bound (6) closely (Fig. 3). The practical message is in the slopes. Six orders of magnitude cost 20 sweeps at $\gamma = 0.5$ and 130 at $\gamma = 0.9$; at $\gamma = 0.99$ the error was still at 13 % of its starting value after 200 sweeps. Contraction guarantees convergence, but a discount factor near one makes it slow.

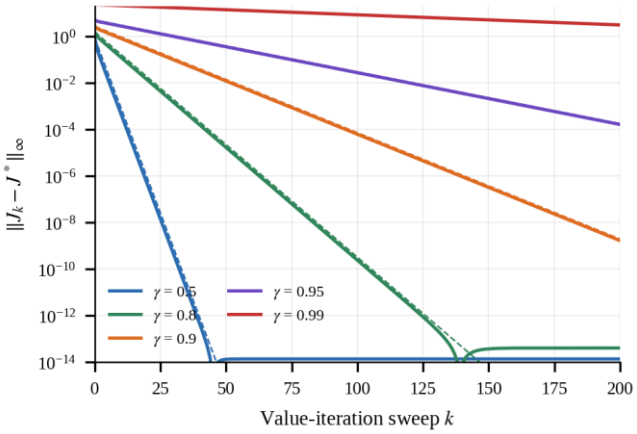


Fig. 3. Value iteration on a random 60-state, 4-action MDP. Solid lines: measured sup-norm error against the converged solution; dashed lines: the bound $\gamma^k\|J_0 - J^*\|_\infty$. Curves flatten near 10^{-14} where double-precision round-off dominates.

C. Semicontractive and Noncontractive Models

Many useful models are not contractions. Stochastic shortest-path problems, risk-sensitive control and some infinite-horizon games contract only under certain policies, or not at all. Bertsekas calls the intermediate case *semicontractive*: some but not all policies are well behaved, with shortest-path problems as the standard example [18]. The analysis there is organized around "regular" policies, those for which the relevant limits exist and are finite [23]. In noncontractive models the Bellman equation can have several solutions or none, and value iteration may settle on a fixed point that is not J^* depending on where it starts; convergence arguments then lean on monotonicity together with problem-specific conditions such as the sign of the costs. The third edition of the monograph adds a chapter on abstract DP for sequential minimax and zero-sum games [18], which indicates where the theory is currently being pushed.

V. Graph-Aware Structural Dynamic Programming

A. Join Ordering as a DP Problem

A relational query joins n relations, and the order of the joins can change the running time by orders of magnitude. Selinger et al. posed the problem as DP over relation subsets [5]. Let G be the join graph, with an edge when a predicate links two relations. The best plan for a subset S is the cheapest over all splits $S = S_1 \uplus S_2$:

$$\text{best}(S) = \min_{\text{splits}} \{ \text{best}(S_1) + \text{best}(S_2) + \text{cost}(S_1 \bowtie S_2) \}. \qquad (7)$$

The two classical enumerators differ in how they generate the splits in (7). DPsize loops over subset sizes and DPsub over bitmasks [7]. DPsub looks at every split of every subset and discards the useless ones afterwards. If each unordered split is counted once, its work is

$$N_{\text{DPsub}} = (3^n - 2^{n+1} + 1) / 2, \qquad (8)$$

independent of the shape of G. We confirmed (8) by direct summation for $n \le 13$. This is the paradox the graph-aware methods exploit: a chain of relations and a fully connected set of relations of the same size cost DPsub the same, even though almost all splits of a chain are invalid.

B. Connected Subgraph–Complement Pairs: DPccp

Moerkotte and Neumann observed that only splits (S_1, S_2) in which both sides are connected subgraphs, the two are disjoint and at least one edge joins them can occur in a cross-product-free plan. They call these csg-cmp pairs, and DPccp enumerates exactly them, each once. Their number is therefore a

lower bound for any DP algorithm over this search space [7]. The count depends strongly on topology. For chains, cycles, stars and cliques it has the closed forms in Table II, which fit all of our counts and match the analysis of [7].

Because published pseudo-code can hide errors, it is worth noting that an erratum to the DPccp paper appeared in 2018 and concerns the recursive enumeration routine [24]. Our own bitmask implementation was therefore validated against brute-force enumeration of all disjoint, connected, adjacent subset pairs for every topology up to $n = 12$, and the two counts agreed in every case.

C. Hypergraphs and Subset Convolution

Real queries contain predicates over more than two relations and non-inner joins. DPhyp lifts csg-cmp enumeration to hypergraphs so that such queries keep the benefit of structural pruning [17]. A different direction attacks the exponent itself. DPconv reformulates join ordering as a subset-convolution problem and, for the unrestricted setting studied there, is the first method to beat the $O(3^n)$ barrier of DPccp, with reported speed-ups of up to 30 times for large clique queries under a maximum-intermediate-size cost [21]. The two lines of work are complementary: pruning pays off on sparse graphs, where DPccp is already cheap, while convolution targets the dense graphs where DPccp degenerates to 3^n.

D. Measured Search-Space Reduction

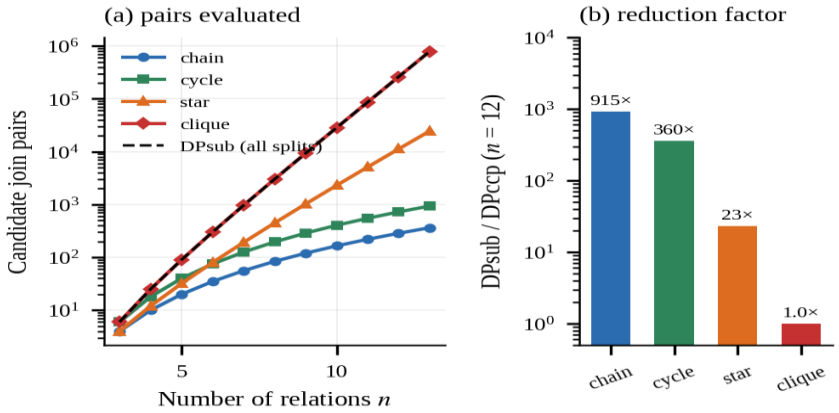


Fig. 4. Join pairs examined as the number of relations grows. (a) Exhaustively counted csg-cmp pairs for four query-graph topologies against the pair count of DPsub from (8). (b) Ratio of DPsub to DPccp work at $n = 12$.

TABLE II
CSG-CMP PAIRS PER TOPOLOGY

Topology	Closed form	$n = 10$	$n = 12$	DPsub ÷ DPccp ($n = 12$)
Chain	$(n^3 - n) / 6$	165	286	915×
Cycle	$n(n-1)^2 / 2$	405	726	360×
Star	$(n-1)\,2^{n-2}$	2,304	11,264	23×
Clique	$(3^n - 2^{n+1} + 1) / 2$	28,501	261,625	1.0×

Fig. 4 and Table II give the numbers. At twelve relations a chain needs 286 pairs instead of the 261,625 splits DPsub looks at, a factor of 915. Cycles and stars gain less, and for cliques the two methods coincide, since every split of a clique is a valid csg-cmp pair. The lesson for survey purposes is that the complexity of join enumeration is a property of the query graph as much as of n, which is why benchmark suites for this problem must report topologies and not only sizes. We count unordered pairs for both algorithms; counting ordered splits would double both without changing any slope.

VI. Neuro-Dynamic Programming and Value-Function Approximation

A. Parametric Value Functions

When states are continuous or too numerous to list, neuro-dynamic programming and approximate DP replace the table by a parametric function $\tilde{J}(x; \theta)$, where θ may be the weights of a neural network or the coefficients of a set of basis functions [15], [25]. A generic fitted value-iteration step applies the Bellman operator to the current approximation at a set of sample states X_k and projects the result back into the function class:

$$\theta_{k+1} = \operatorname{argmin}_{\theta} \sum_{x \in X_k} \left(\tilde{J}(x; \theta) - (T \tilde{J}(\cdot; \theta_k))(x) \right)^2. \qquad (9)$$

Two things can go wrong at once. The projection may not be a contraction in the same norm as T, so the composite map loses the guarantee of (5), and sampling adds noise on top of the approximation error. The model-free version updates θ one transition at a time with a temporal-difference rule,

$$\theta_{k+1} = \theta_k + \eta_k \left[g + \gamma \tilde{J}(x'; \theta_k) - \tilde{J}(x; \theta_k) \right] \nabla_\theta \tilde{J}(x; \theta_k), \qquad (10)$$

with step size η_k (written to avoid a clash with the discount factor). For on-policy sampling and linear features, convergence of such schemes has been

established [16]; the guarantee does not carry over to arbitrary sampling distributions.

B. When Approximation Diverges

Baird's counterexample [26], used as a standard illustration in [9], shows the failure in its cleanest form. Seven states share eight weights; the first six states have value $2w_i + w_8$ and the seventh $w_7 + 2w_8$. The target policy always moves to state 7 at zero reward, so the true values are all zero, but states are weighted uniformly, as if sampled by a different behaviour policy. The expected update

$$w \leftarrow w + \eta \sum_s \mu(s) \left[\gamma\, v(7; w) - v(s; w) \right] \nabla_w v(s; w) \qquad (11)$$

has no sampling noise at all, since it is a DP sweep over the whole state space. With $\gamma = 0.99$, $\eta = 0.1$, uniform μ and initial weights (1, 1, 1, 1, 1, 1, 10, 1) it still diverges: after 1,000 sweeps the largest weight has reached 6.4×10^{11}, while the same update with a tabular representation decays smoothly toward the true value of zero (Fig. 5). The divergence is therefore a property of the combination of approximation, bootstrapping and off-policy weighting, the "deadly triad" discussed in [9] and examined empirically for deep Q-networks in [27], and not an artefact of noisy samples.

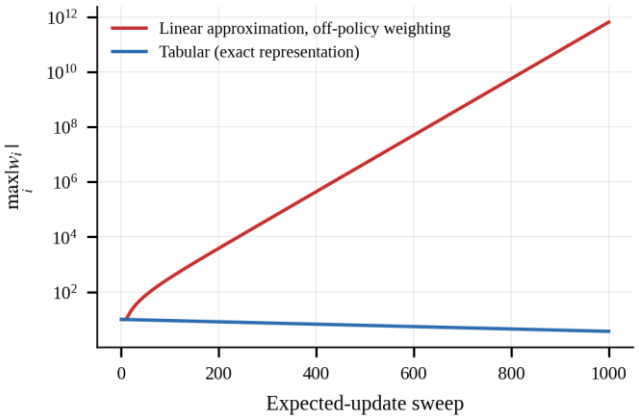


Fig. 5. Baird's counterexample as an expected (DP-style) update. The linear representation with uniform state weighting diverges geometrically; the tabular representation of the same problem converges toward the true values of zero. Our own run, parameters in the text.

C. Deep Reinforcement Learning as Neuro-DP

Deep reinforcement learning is, to a first approximation, the neural instance of (10). Deep Q-networks combine a convolutional approximator with experience replay and a slowly updated target network, which together damp the instability above, and reached human-level scores on many Atari games [19]. Deterministic actor–critic methods extend the idea to continuous actions [28], and self-play systems pair a learned value with tree search [29]. That such systems work despite the triad is partly an empirical fact that theory has not caught up with. Van Hasselt et al. vary network size, bootstrapping depth and replay prioritization in deep Q-networks and study when instability appears [27].

D. Lookahead, Rollout and Model-Predictive Control

A second route to scale keeps a model and uses the approximate value only at the end of a short search. Rollout algorithms improve a base policy by one-step or multistep lookahead with simulation of the base policy [30]. Plan online, learn offline (POLO) couples local trajectory optimization with a global value function that is learned from the states the planner visits, and reports solving simulated humanoid locomotion and in-hand manipulation tasks with the equivalent of a few minutes of real-world experience [13]. The two components cover each other's weakness: planning absorbs errors of the value function, and the value function shortens the planning horizon.

E. Learned Query Optimization

The same idea has reached databases. Krishnan et al. cast join ordering as a sequential decision problem and learn a value function for it with deep reinforcement learning [31], and Neo uses a learned value network to guide plan search [32]. Compared with the exact methods of Section V, these systems give up optimality guarantees and gain the ability to learn from execution feedback and to handle very large queries.

VII. Parallel DP on Multicore and GPU Hardware

A. Dependency Structure and Wavefronts

In alignment-style recurrences, cell (i, j) depends only on its left, upper and upper-left neighbours. Every cell on an anti-diagonal $i + j = d$ is therefore independent of the others, and an $n \times n$ table can be filled diagonal by diagonal. The number of cells on diagonal d is

$$w_d = \min(d, n, 2n - d), \quad d = 1, \ldots, 2n - 1. \qquad (12)$$

With P processing elements and a barrier between diagonals, the parallel time in cell updates is $T(P) = \sum_d \lceil w_d / P \rceil$ and the speed-up over the sequential n^2 updates is

$$S(P) = n^2 / T(P) \;\le\; n^2 / (2n - 1) \approx n / 2. \qquad (13)$$

Parallelism is large in the middle of the table and nearly absent at the two corners, so even unlimited hardware gains only about $n/2$ on an $n \times n$ grid.

B. GPU Practice

The wavefront method is the usual starting point for GPU alignment codes. SW# runs it under CUDA and combines it with a linear-memory reconstruction scheme so that exact alignments of genome-scale sequences remain possible [33]. Lee et al. report a heterogeneous anti-diagonal scheme for Needleman–Wunsch, in which the CPU handles the short corner diagonals and the GPU the long ones, running about three times faster than a serial CPU version [34]. A review of the Smith–Waterman GPU literature finds that the many published implementations fall into a small set of algorithmic motifs, which also exposes gaps that have not been explored [35]. A recurring practical issue, which follows from the geometry of (12) rather than from any one paper, is that consecutive threads along a diagonal touch memory locations a full row apart, so efficient implementations tile the table into blocks held in on-chip memory.

C. Parallel Join Enumeration

The subset-structured DP of Section V parallelizes differently, because independent work comes from many subsets of the same size instead of a diagonal. MPDP combines a new enumeration technique with GPU execution and is reported to be at least an order of magnitude faster than earlier methods on large analytical queries; with it the authors raised the limit at which PostgreSQL falls back to heuristics from 12 to 25 relations within the same time budget [12].

D. A Quantitative View of the Wavefront

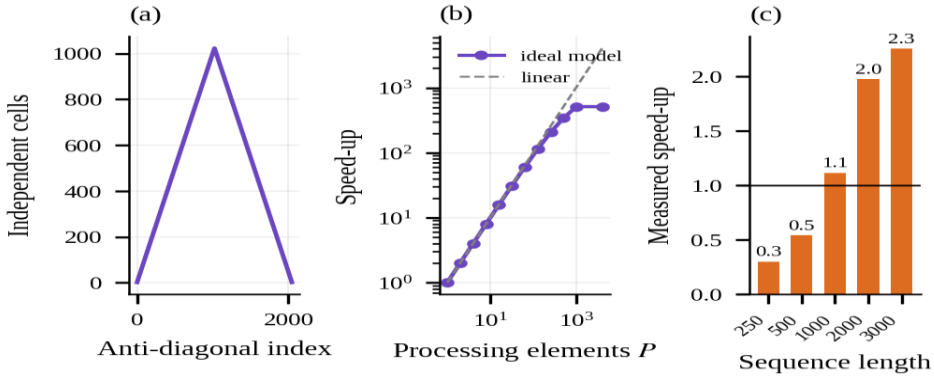


Fig. 6. Wavefront parallelism for an $n \times n$ DP table. (a) Independent cells per anti-diagonal for $n = 1{,}024$. (b) Ideal speed-up $S(P)$ from (13) against the number of processing elements. (c) Measured CPU speed-up of a NumPy per-diagonal implementation over a pure-Python double loop (longest-common-subsequence recurrence, median of three runs, single-core virtual machine). Panel (c) is not a GPU measurement.

For $n = 1{,}024$ the ideal model of (13) gives $S = 60$ at $P = 64$ (94 % efficiency) and 205 at $P = 256$ (80 %), and it saturates at 512 once P reaches n (Fig. 6b). So the first lesson is that moderate parallelism is nearly free and large parallelism is wasteful. The second lesson is about overhead, which the model leaves out. In our CPU baseline the per-diagonal NumPy version was *slower* than the plain loop for 250- and 500-element sequences (0.30× and 0.54×), broke even near 1,000 (1.12×) and reached 2.26× at 3,000. Each diagonal costs a fixed amount of setup, so short sequences cannot amortize it. GPUs shift the break-even point, since launching and synchronizing a kernel per diagonal is also costly, but the structure of the trade-off is the same, and this is why the heterogeneous schemes above hand corner diagonals to the CPU.

VIII. Quantum Dynamic Programming

A. Grover Search on Top of a Partial Table

Grover search alone is not competitive with DP. For the travelling salesman problem it reduces the $n!$ routes to roughly $\sqrt{(n!)}$, which is still worse than the classical $O^*(2^n)$ of Bellman–Held–Karp. Ambainis et al. instead compute a small part of the DP table classically and use quantum search or minimum finding for the large middle layer of the subset lattice, recursing on the sub-cubes [8]. For problems that fit the "path in the hypercube" template, including many vertex-ordering problems, this gives $O^*(1.817^n)$ against $O^*(2^n)$, and for the travelling salesman problem, minimum set cover and feedback arc set $O^*(1.728^n)$ (Table III).

TABLE III

REPORTED QUANTUM SPEED-UPS FOR SUBSET DP [AMBAINIS ET AL.]

Problem	Best classical	Quantum
Path in the hypercube	$O^*(2^n)$	$O^*(1.817^n)$
Vertex-ordering problems	$O^*(2^n)$	$O^*(1.817^n)$
Travelling salesman	$O(n^2 2^n)$	$O^*(1.728^n)$
Feedback arc set	$O^*(2^n)$	$O^*(1.728^n)$
Minimum set cover	$O(nm2^n)$	$O(\mathrm{poly}(m, n)\ 1.728^n)$
Graph bandwidth	$O^*(4.383^n)$	$O^*(2.946^n)$

The exponent of the asymptotic advantage over 2^n is

$$(2 / 1.7274)^n \approx 1.158^n, \qquad (14)$$

which is about 80 at $n = 30$ and about 1,500 at $n = 50$ for the 1.728 bound, and about 18 and 120 for 1.817 (Fig. 7). Two caveats limit what these numbers say about practice. First, they hide polynomial factors and constants. Second, and more important for this survey, the algorithms work in a QRAM model and still need exponential space for the partial table [8]. A 2026 preprint studies what happens when quantum memory is limited and derives time–space trade-offs that keep a speed-up over classical algorithms with smaller tables [22]. Statements that quantum DP avoids memory allocation by holding intermediate histories in superposition are therefore not supported by this literature.

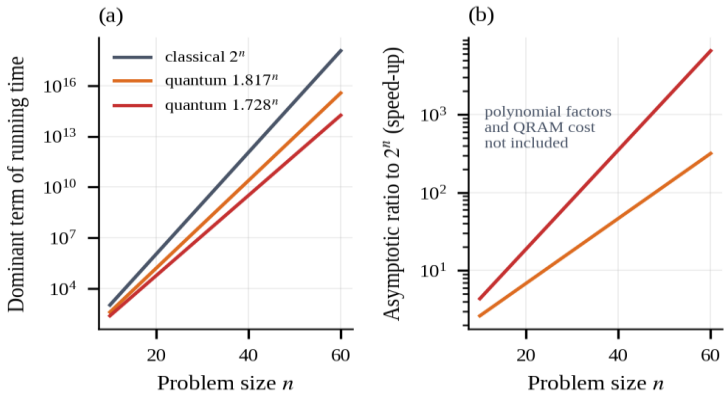


Fig. 7. Reported asymptotic running-time exponents for subset DP. (a) Dominant exponential terms of the classical and the two quantum bounds in Table III. (b) Ratio to 2^n. Both panels plot bounds from [8] only; polynomial factors and QRAM cost are not included.

B. QUBO and Annealing Formulations

A second, structurally different route encodes the whole optimization as a quadratic unconstrained binary optimization (QUBO) or Ising problem [36], which quantum annealers address [37]. Join ordering was the first database problem to receive a quantum implementation of this kind; the authors found that the limits of early-stage hardware are reached quickly and derived co-design recommendations for future processors [20]. Quantum-inspired digital annealers are a nearer-term alternative and have been applied to the same problem [38]. We note that these formulations do not exploit the principle of optimality at all, since the optimum is searched globally; they sit next to DP and not within it.

C. The NISQ Constraint

Today's devices are noisy and small, the setting Preskill termed NISQ [39]. The algorithms in Table III assume fault-tolerant machines with large quantum memory, while the annealing results above are limited by qubit counts and connectivity. Quantum DP is thus best read as a theoretical benchmark for what could be gained, not as a deployable method.

IX. Comparative Analysis

TABLE IV

COMPARISON OF DP PARADIGMS

Paradigm	Relaxes	Guarantee	Cost driver	Main limitation
Exact tabular DP	nothing	exact optimum	$\|S\|^2\|A\|$ per sweep (dense)	curse of dimensionality
Abstract DP	the operator	convergence under monotonicity and contraction	depends on model	slow near $\gamma = 1$; pathologies without contraction
Graph-aware DP	the state space	exact, within the valid search space	#csg-cmp pairs: n^3 to 3^n	needs graph structure; dense graphs stay hard
Neuro-DP / ADP	the value table	few; may diverge off-policy	$O(\|\theta\|)$ per update	approximation error; instability
Parallel DP	serial execution	same as underlying DP	diagonal width, memory traffic	corners; launch overhead
Quantum DP	classical hardware	bounded-error, asymptotic	QRAM size; 1.73^n–1.82^n	needs fault tolerance; exponential memory

Table IV lines up the paradigms. Three observations follow. First, guarantees and scale pull in opposite directions: methods that keep exactness

(abstract, graph-aware, parallel) either assume structure or stay exponential in the worst case, while methods that scale to continuous problems (neuro-DP) have the weakest guarantees. Second, the relaxations are complementary and the strongest systems combine them: MPDP joins graph-aware pruning to GPUs [12], and POLO joins search to learned values [13]. Third, communities evaluate differently. Database papers count enumerated pairs and optimization time, reinforcement-learning papers report returns over environment steps, and quantum papers state asymptotic exponents. Without common benchmarks, cross-paradigm claims remain qualitative.

X. Applications

A. Database Query Optimization

This is the area where DP is most firmly embedded in production software. Optimizers descend from the DP join enumeration of Selinger et al. [5], use csg-cmp enumeration for exact search, and fall back on heuristics past a size threshold; the work of [12] pushes that threshold from 12 to 25 relations in PostgreSQL. Learned optimizers [32], [31] and quantum formulations [20] are active research directions, not replacements.

B. Robotics and Autonomous Control

Continuous control is the natural home of neuro-DP. Deterministic actor–critic learning handles continuous actions [28], and lookahead with a learned terminal value reduces the data requirement for complex simulated bodies [13]. The open concern in this area is not capability on benchmarks but stability guarantees, which Section VI showed to be fragile.

C. Energy Storage and Power Systems

Storage dispatch under uncertain prices, wind and load is a multistage stochastic problem with a continuous state. Powell and Meisel organize the solution methods into four classes of policies and show with a storage example that, depending on the problem, each class can be best [40], [41]. Jiang et al. compare approximate DP techniques on benchmark storage problems [42], and Nascimento and Powell prove convergence of an approximate DP algorithm that exploits concavity of the value function in a scalar-storage setting [43]. Here the structure of the problem, and not generic function approximation, is what delivers guarantees.

D. Computational Biology

Sequence alignment [2], [3] and RNA secondary-structure prediction [4] are textbook DP workloads, and GPU wavefront codes accelerate the alignment part [33], [34], [35]. We found no source in the surveyed literature that supports the broader claim that grid DP on GPUs accelerates molecular docking, and so we do not repeat it.

TABLE V

APPLICATION SUMMARY

Domain	Dominant paradigm	Evidence in cited work	Maturity
Query optimization	Graph-aware; parallel	order-of-magnitude faster enumeration; limit 12 → 25 relations [12]	Production
Robot control	Neuro-DP; lookahead	minutes of simulated experience on humanoid tasks [13]	Research
Energy storage	Approximate DP	convergence proof for concave scalar storage [43]	Pilot / research
Sequence analysis	Parallel DP	≈3× over serial CPU for heterogeneous NW [34]	Production

XI. Challenges, Limitations and Future Directions

TABLE VI

OPEN CHALLENGES AND CANDIDATE MITIGATIONS

Challenge	Why it matters	Candidate mitigation
Nonlinear convergence	approximation can break contraction; values diverge	target networks, multistep returns, lookahead [27], [13]
Sample efficiency	real interaction is costly in control	model-based planning with learned values [13]
Dense join graphs	DPccp is 3^n on cliques	subset convolution [21]; GPUs [12]
QRAM and noise	exponential quantum memory; NISQ limits	time–space trade-offs [22]; co-design [20]
Benchmarks	no shared yardstick across fields	report topology, seeds and code; see Appendix B

A. Convergence Under Nonlinear Approximation

The divergence reproduced in Fig. 5 is a linear example. For deep networks theory offers even less, and practice relies on stabilizers whose effect is documented empirically [27]. A convergence theory that covers common deep architectures with replay and target networks would be a major contribution.

B. Sample Efficiency and Safety

Methods that mix planning and learning cut data needs [13], but safe exploration on physical systems remains open, because a value function that is wrong in unvisited states can steer a planner into them.

C. Reproducibility of Algorithms

The 2018 erratum to DPccp [24] is a reminder that even carefully analyzed pseudo-code can be wrong, and that implementations should be tested against brute force, as we did in Section V. We recommend that DP papers publish enumeration counts and code, not only timings.

D. Hardware Realism

Our wavefront study suggests that overhead, not available parallelism, is what limits small and medium problems, and that reported GPU speed-ups need the problem size stated next to them. For quantum DP the corresponding demand is a resource estimate, including QRAM size, next to every asymptotic exponent.

E. Limitations of This Survey

This survey has four limits. It covers literature that could be verified on a bibliographic page and therefore omits some relevant work. The computational studies use small random instances and a CPU baseline; they illustrate mechanisms and do not benchmark systems. Claims taken from preprints, such as [21] and [22], were not independently verified. Finally, the field moves quickly and the review stops in early 2026.

XII. Conclusion

Dynamic programming has moved far past the lookup table. Abstract DP explains when iteration converges and where it does not; graph-aware DP shows that the cost of exact enumeration depends on the shape of the problem, with our exhaustive counts giving a 915-fold saving on a twelve-relation chain and none on a clique; neuro-DP scales to continuous control but, as Baird's example shows, at the price of guarantees; parallel DP is limited by the geometry of dependencies and by overhead; and quantum DP promises a constant-base exponential speed-up that depends on memory assumptions no current device meets. The next steps lie in combining these relaxations, in theory for nonlinear approximation, and in benchmarks that let the communities compare their results.

Appendix A: Nomenclature

TABLE VII
ABBREVIATIONS AND SYMBOLS

Symbol	Meaning
DP	dynamic programming
ADP	approximate dynamic programming ("abstract DP" is always spelled out here to avoid the clash)
NDP	neuro-dynamic programming
MDP	Markov decision process
TD	temporal difference
csg-cmp	connected subgraph and connected complement
MPDP	massively parallel dynamic programming
QRAM	quantum random-access memory
NISQ	noisy intermediate-scale quantum
QUBO	quadratic unconstrained binary optimization
γ, η	discount factor; step size
T, J^*	Bellman operator; optimal cost-to-go
$O^*(\cdot)$	running time up to polynomial factors

Appendix B: Reproducibility Notes

Value iteration (Section IV): random MDP, 60 states, 4 actions, transition rows proportional to uniform random numbers cubed and normalized, costs

uniform on [0, 1], NumPy seed 7, 200 sweeps, reference solution iterated to 10^{-14}. *Join enumeration* (Section V): graphs for chain, cycle, star and clique with $2 \le n \le 13$; csg-cmp pairs counted by exhaustive search over disjoint connected adjacent subset pairs for $n \le 12$ and by a bitmask implementation of DPccp for all n; both agree wherever both were run. *Baird* (Section VI): features, initial weights, $\gamma = 0.99$, $\eta = 0.1$ and uniform state weighting as in the text; 1,000 sweeps. *Wavefront* (Section VII): ideal model for $n = 1{,}024$; CPU timings are medians of three runs of a longest-common-subsequence recurrence on random 4-letter sequences, Python 3.12 and NumPy 2.4 on a single-core 2.8 GHz Intel Xeon virtual machine. The scripts accompany this manuscript.

References

[1] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 1957.

[2] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Mol. Biol.*, vol. 48, no. 3, pp. 443–453, 1970.

[3] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, no. 1, pp. 195–197, 1981.

[4] M. Zuker and P. Stiegler, "Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information," *Nucleic Acids Res.*, vol. 9, no. 1, pp. 133–148, 1981.

[5] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1979, pp. 23–34.

[6] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2005.

[7] G. Moerkotte and T. Neumann, "Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products," in *Proc. 32nd Int. Conf. Very Large Data Bases (VLDB)*, Seoul, South Korea, 2006, pp. 930–941.

[8] A. Ambainis, K. Balodis, J. Iraids, M. Kokainis, K. Prūsis, and J. Vihrovs, "Quantum speedups for exponential-time dynamic programming algorithms," in *Proc. 30th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*, 2019, pp. 1783–1793.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[10] M. Held and R. M. Karp, "A dynamic programming approach to sequencing problems," *J. Soc. Ind. Appl. Math.*, vol. 10, no. 1, pp. 196–210, 1962.

[11] R. Bellman, "Dynamic programming treatment of the travelling salesman problem," *J. ACM*, vol. 9, no. 1, pp. 61–63, 1962.

[12] R. Mancini, S. Karthik, B. Chandra, V. Mageirakos, and A. Ailamaki, "Efficient massively parallel join optimization for large queries," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2022.

[13] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, "Plan online, learn offline: Efficient learning and exploration via model-based control," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[14] E. V. Denardo, "Contraction mappings in the theory underlying dynamic programming," *SIAM Rev.*, vol. 9, no. 2, pp. 165–177, 1967.

[15] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA, USA: Athena Scientific, 1996.

[16] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Trans. Autom. Control*, vol. 42, no. 5, pp. 674–690, May 1997.

[17] G. Moerkotte and T. Neumann, "Dynamic programming strikes back," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 539–552.

[18] D. P. Bertsekas, *Abstract Dynamic Programming*, 3rd ed. Belmont, MA, USA: Athena Scientific, 2022.

[19] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[20] M. Schönberger, S. Scherzinger, and W. Mauerer, "Ready to leap (by co-design)? Join order optimisation on quantum hardware," *Proc. ACM Manage. Data*, vol. 1, no. 1, Art. 92, May 2023.

[21] M. Stoian and A. Kipf, "DPconv: Super-polynomially faster join ordering," arXiv:2409.08013, Sep. 2024 (to appear at SIGMOD 2025).

[22] S. Caroppo, J. Vihrovs, D. Zajakina, and A. Zajakins, "Quantum time-space tradeoffs for exponential dynamic programming," arXiv:2604.02233, Apr. 2026.

[23] D. P. Bertsekas, "Regular policies in abstract dynamic programming," *SIAM J. Optim.*, vol. 27, no. 3, pp. 1694–1727, 2017.

[24] A. Meister, G. Moerkotte, and G. Saake, "Errata for 'Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products'," *Proc. VLDB Endow.*, vol. 11, no. 10, pp. 1069–1070, 2018.

[25] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, 2nd ed. Hoboken, NJ, USA: Wiley, 2011.

[26] L. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *Proc. 12th Int. Conf. Mach. Learn. (ICML)*, 1995, pp. 30–37.

[27] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep reinforcement learning and the deadly triad," arXiv:1812.02648, Dec. 2018.

[28] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[29] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[30] D. P. Bertsekas, *Rollout, Policy Iteration, and Distributed Reinforcement Learning*. Belmont, MA, USA: Athena Scientific, 2020.

[31] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica, "Learning to optimize join queries with deep reinforcement learning," arXiv:1808.03196, 2018.

[32] R. Marcus *et al.*, "Neo: A learned query optimizer," *Proc. VLDB Endow.*, vol. 12, no. 11, pp. 1705–1718, 2019.

[33] M. Korpar and M. Šikić, "SW#–GPU-enabled exact alignments on genome scale," *Bioinformatics*, vol. 29, no. 19, pp. 2494–2495, 2013.

[34] Y. S. Lee, Y. S. Kim, and R. L. Uy, "Serial and parallel implementation of Needleman-Wunsch algorithm," *Int. J. Adv. Intell. Informat.*, vol. 6, no. 1, pp. 97–108, Mar. 2020.

[35] R. Barnes, "A review of the Smith-Waterman GPU landscape," Dept. EECS, Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2020-152, Aug. 2020.

[36] A. Lucas, "Ising formulations of many NP problems," *Front. Phys.*, vol. 2, Art. no. 5, 2014.

[37] T. Kadowaki and H. Nishimori, "Quantum annealing in the transverse Ising model," *Phys. Rev. E*, vol. 58, no. 5, pp. 5355–5363, 1998.

[38] M. Schönberger, I. Trummer, and W. Mauerer, "Quantum-inspired digital annealing for join ordering," *Proc. VLDB Endow.*, 2024.

[39] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[40] W. B. Powell and S. Meisel, "Tutorial on stochastic optimization in energy—Part I: Modeling and policies," *IEEE Trans. Power Syst.*, vol. 31, no. 2, pp. 1459–1467, Mar. 2016.

[41] W. B. Powell and S. Meisel, "Tutorial on stochastic optimization in energy—Part II: An energy storage illustration," *IEEE Trans. Power Syst.*, vol. 31, no. 2, pp. 1468–1475, Mar. 2016.

[42] D. R. Jiang, T. V. Pham, W. B. Powell, D. F. Salas, and W. R. Scott, "A comparison of approximate dynamic programming techniques on benchmark energy storage problems: Does anything work?," in *Proc. IEEE Symp. Adapt. Dyn. Program. Reinforcement Learn. (ADPRL)*, 2014, pp. 1–8.

[43] J. Nascimento and W. B. Powell, "An optimal approximate dynamic programming algorithm for concave, scalar storage problems with vector-valued controls," *IEEE Trans. Autom. Control*, vol. 58, no. 12, pp. 2995–3010, Dec. 2013.