Quantum dynamic programming and graph problems

Combine dynamic-programming reuse with quantum search under explicit memory and access assumptions.

Research 21 min lesson Dr. Charith Jayasekara

By the end, you'll be able to

- Trace a two-subset tour recurrence.
- Distinguish an improved exponential base from polynomial time.
- Account for coherent tables and path reconstruction.

Reuse subproblems before accelerating search

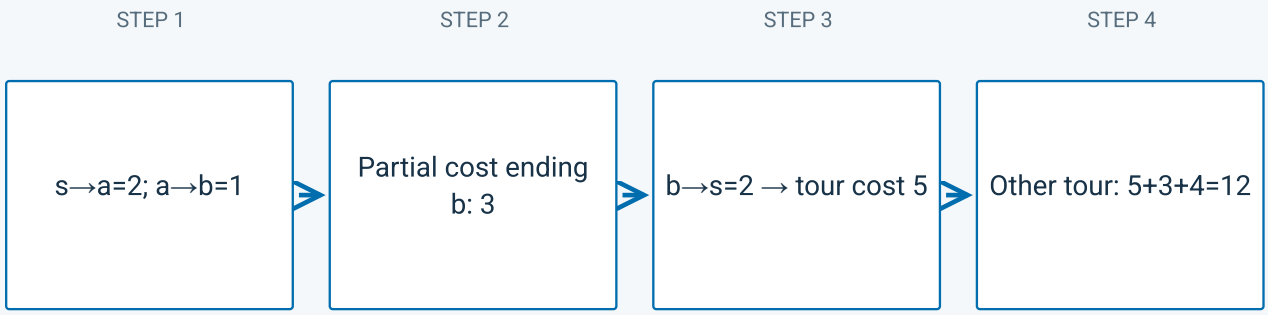
Dynamic programming solves a large problem by storing answers to overlapping smaller problems. Quantum speedups can sometimes accelerate a search or minimum inside that recurrence, or combine a partial classical table with quantum search over the remaining structure. The useful strategy depends on what is reused and how the table is accessed coherently.

This is not a rule that every loop in a dynamic program becomes quadratically faster. A recurrence may already have unavoidable input or output costs, and computing table entries coherently can be more expensive than reading them classically. The algorithm must preserve the dependency structure and account for memory, precision and error across repeated subroutines.

Review minimum finding and search ideas, reversible computation and complexity. A recurrence expresses one answer in terms of smaller answers, while memoisation stores results so they are not recomputed unnecessarily.

WORKED DIAGRAM · AA19

Reuse a partial tour cost



For the stated directed costs, s–a–b–s costs 5 and s–b–a–s costs 12. A subset recurrence stores the partial cost 3 before adding the final return edge.

Start with a graph recurrence

In a directed acyclic graph, a shortest-path distance can be computed from predecessor distances: the distance to a vertex is the minimum over predecessor distance plus edge weight. A quantum minimum-finding subroutine might help when there are many predecessors and their candidate values can be accessed coherently. But it still needs the already-computed distances and graph data in an appropriate interface.

Suppose a vertex has three predecessor candidates with total costs seven, four and nine. Its distance is four. The local minimisation is simple. On a large graph, however, the full algorithm must compute many such distances in dependency order, store them and perhaps reconstruct a path. Speeding up one minimisation does not automatically speed up every other part.

Work a small subset dynamic program

Consider travelling from a start vertex s through vertices a and b and returning to s. Let costs be s to a equal to two, s to b equal to five, a to b equal to one, b to a equal to three, a to s equal to four and b to s equal to two. The tour s–a–b–s costs five. The tour s–b–a–s costs twelve.

A subset recurrence stores the best cost for reaching a final vertex after visiting a given subset. The entry for subset {a,b} ending at b uses the entry for {a} ending at a plus the edge a to b, giving three before the return edge. The corresponding entry ending at a gives eight. Adding return costs yields five and twelve.

For many vertices, the number of subsets grows exponentially. Classical dynamic programming improves on enumerating every permutation by reusing these subset answers. Quantum methods can combine such reuse with search over carefully chosen decompositions. A naive Grover search over all tours ignores that existing classical improvement and may be the wrong baseline.

The subset recurrence reuses the best smaller route instead of enumerating all full tours.

$$D(S,j)=\min_{i\in S\setminus\{j\}}\left[D(S\setminus\{j\},i)+w_{ij}\right],\qquad D(\{j\},j)=w_{sj},\qquad D(\{a,b\},b)+w_{bs}=2+1+2=5.$$

$D(S, j)$ is the minimum cost from start s through all vertices of S, ending at j; w_{ij} is the directed edge cost. The recurrence applies when $|S| > 1$. Returning to the start completes the tour, giving the chapter's optimal route cost five.

What an exponential-time speedup means

Ambainis and colleagues develop quantum speedups for several exponential-time dynamic-programming problems by combining partial tables with quantum search. Their results improve exponential bases for specified problems and access constructions. They do not turn those hard problems into polynomial-time tasks. Read the dynamic-programming paper.

This distinction is easy to miss. Changing a growth rate from two to the power n to a smaller constant raised to n can be a substantial asymptotic improvement while remaining exponential. It may also require exponential memory or significant preprocessing. State time and space together, and include the cost of preparing the portion of the table that supports coherent search.

Graph-query models also matter

An adjacency matrix supplies potential edges between every pair of vertices; an adjacency list supplies neighbours. Quantum graph algorithms can have different query complexities in those models. A method efficient in one representation may be costly in another. Weighted graphs add arithmetic precision and comparison costs, especially when path costs are nearly tied.

The output can be a yes-or-no reachability decision, a distance value, one path or all-pairs information. Printing a path has a length cost, and printing all distances has an output-size cost. A theorem for deciding whether a path exists is not a theorem for returning every path. Match the user's required result before choosing an algorithm.

Follow a disciplined adaptation

Write the classical recurrence and identify its genuine bottleneck. Determine which candidate set can be searched or minimised quantumly, and whether candidate evaluation can use stored subproblem answers coherently. Allocate error across subroutine calls so a small failure probability does not accumulate into an unreliable overall result. Then account for table preparation, memory, reconstruction and verification.

If a recurrence reuses an answer many times, rebuilding it inside each quantum query may destroy the expected benefit. Conversely, storing a large table in a coherent-access data structure may require resources absent from the device model. These are central design trade-offs. An asymptotic theorem should state which route it assumes.

Compare beyond the simplest recurrence

Classical graph and optimisation methods exploit sparsity, heuristics, approximation, preprocessing and problem-specific structure. A quantum improvement over one textbook dynamic program does not establish superiority over every practical solver. Benchmark matched solution quality and include setup costs. For exact exponential algorithms, also compare memory and the range of instance sizes that fit.

Approximate numerical values require another layer of care. If a quantum minimum finder compares rounded candidate costs, a near tie can be misordered. Define a tolerance, a gap promise or a bounded objective error. More precise arithmetic may increase circuit depth and workspace enough to affect the application comparison.

Practice with answers

Exercise 1. In the worked tour example, calculate the best partial cost visiting a and b and ending at b, before returning to s.

Check the worked answer

The route s–a–b costs two plus one, or three. Adding b–s gives the full tour cost five. The partial entry separates reusable subproblem information from the final return edge.

Exercise 2. Why is Grover over all permutations not automatically the best quantum travelling-salesperson approach?

Check the worked answer

It ignores classical dynamic-programming reuse and other structure. A fair comparison starts from strong classical exact or approximate methods and studies how quantum search interacts with their subproblems, memory and access costs. The best representation can matter more than applying a square root to the largest brute-force space.

Check your understanding

PAUSE & THINK

What is the partial cost of s–a–b before returning to s?

- ☐ 3
- ☐ 5
- ☐ 12

Answer: 3. Add s→a=2 and a→b=1 to obtain 3. The return edge adds another 2 only at the final step.

Sources

- Read the dynamic-programming paper