A polynomial-time dynamic programming algorithm for the $q/1/D/F$ problem

Andrea Cracco[1], Dario Ostuni[2], Giovanni Righini[2], Romeo Rizzi[1]

[1] University of Verona, Dept. of Computer Science
[2] University of Milan, Dept. of Computer Science

Napoli, January 16th, 2026

UNIVERSITÀ DEGLI STUDI DI MILANO

PRIN project 2022YWWETX
WOW - We Optimize Warehouses

The $q/1/D/F$ problem

Problem classified as $q/1/D/F$:

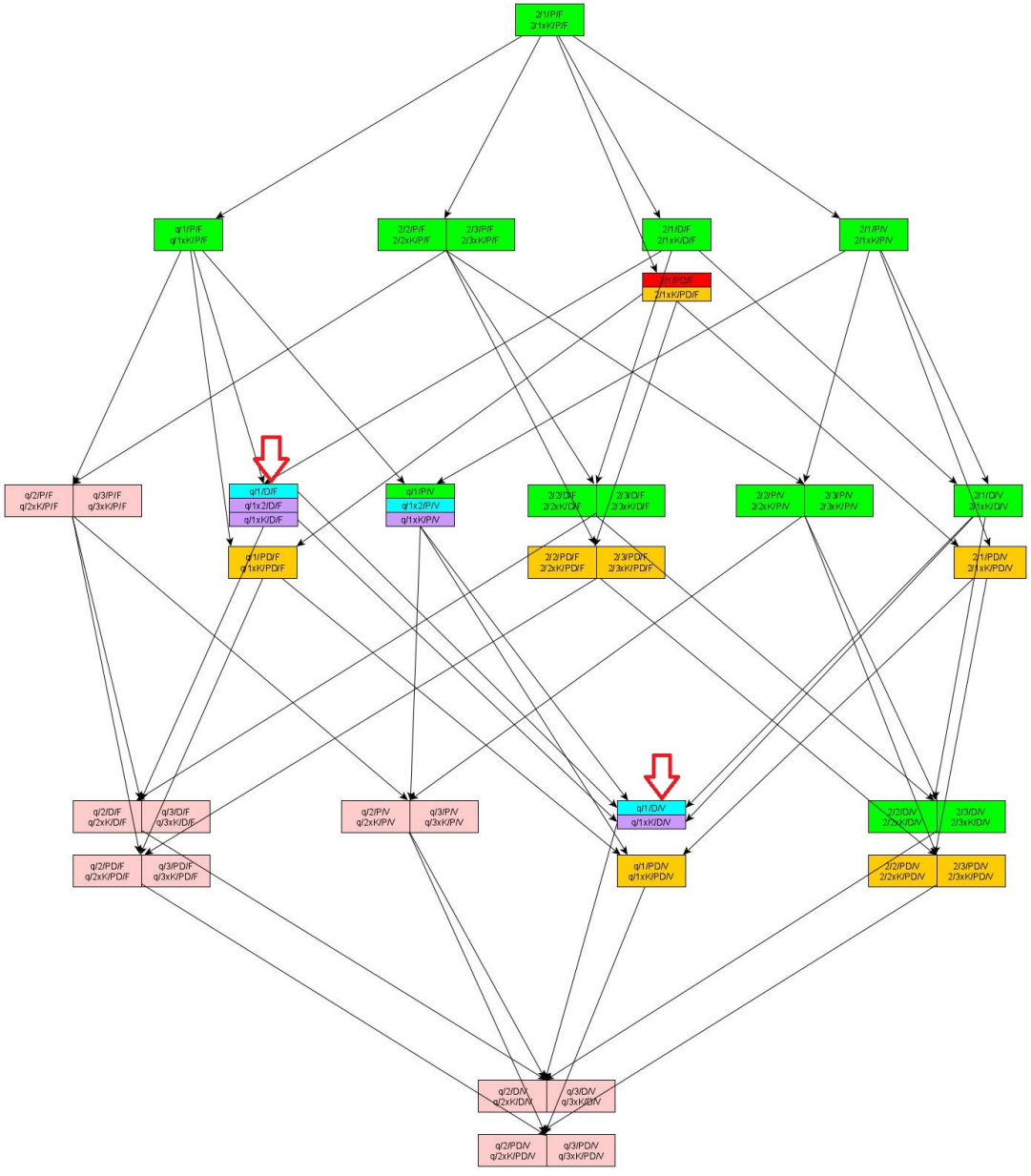
- at most q items at a time;
- mono-dimensional rail, with the origin located at an endpoint;
- only delivery operations (given arrival sequence): $N = \{1, \ldots, n\}$;
- each item has a given unique destination at a distance $d(i)$ from the origin.

Objective: minimization of the total distance travelled by the shuttle.

Previous result (Barbato et al., 2019): dynamic programming algorithm; complexity $O(qn^q)$.

New result: dynamic programming algorithm; complexity $O(n^3 q^2)$.

The taxonomy graph



Definitions and properties

Definition (No delay in delivering)

An NDD-solution is a solution with no delay in delivering: each item on board is delivered as soon as its destination is visited.

Observation

For any non-NDD solution x, a non-worse NDD solution $\overline{x}$ exists.

Observation

In any NDD solution, at the end of a trip in which an item i has been delivered, no item j with $d(j) \leq d(i)$ remains on the shuttle.

Corollary

In any NDD solution, after the trip in which a farthest item is delivered no items remain on the shuttle.

Definitions and properties

Definition (Insertion position)

For any given solution, the insertion position of a generic item $i \in N$ is the maximum $j \in N$ s.t. items i and j are delivered in the same trip.

Definition (No delay in traveling)

An NDT-solution is a solution with no delay in traveling: each trip is executed as soon as its items are onboard.

Observation

For any non-NDT solution x, a non-worse NDT solution $\overline{x}$ exists.

Observation

Let t be any trip of an NDT solution x, let $N_t \subseteq N$ be the subset of items delivered in trip t and let j be their insertion position. Then, $j \in N_t$.

Corollary

In any NDT solution, no two distinct trips have the same insertion position.

Definitions and properties

Definition (Canonical solution)

A canonical solution is a solution which is both NDD and NDT.

The search for an optimal solution can be restricted to canonical solutions.

Observation

If an instance consists of delivering k items when k or more units of capacity are available, then its optimal solution consists of a single trip.

Leading items

Definition (Leading item)

Let N be any set of items. Let $\overline{N} \subseteq N$ the subset of items in N with maximum distance: $\overline{N} = \{i \in N : d(i) = \max_{j \in N}\{d(j)\}\}$. The leading item in N is item α with maximum index in $\overline{N}$: $\alpha = \max\{j \in \overline{N}\}$.

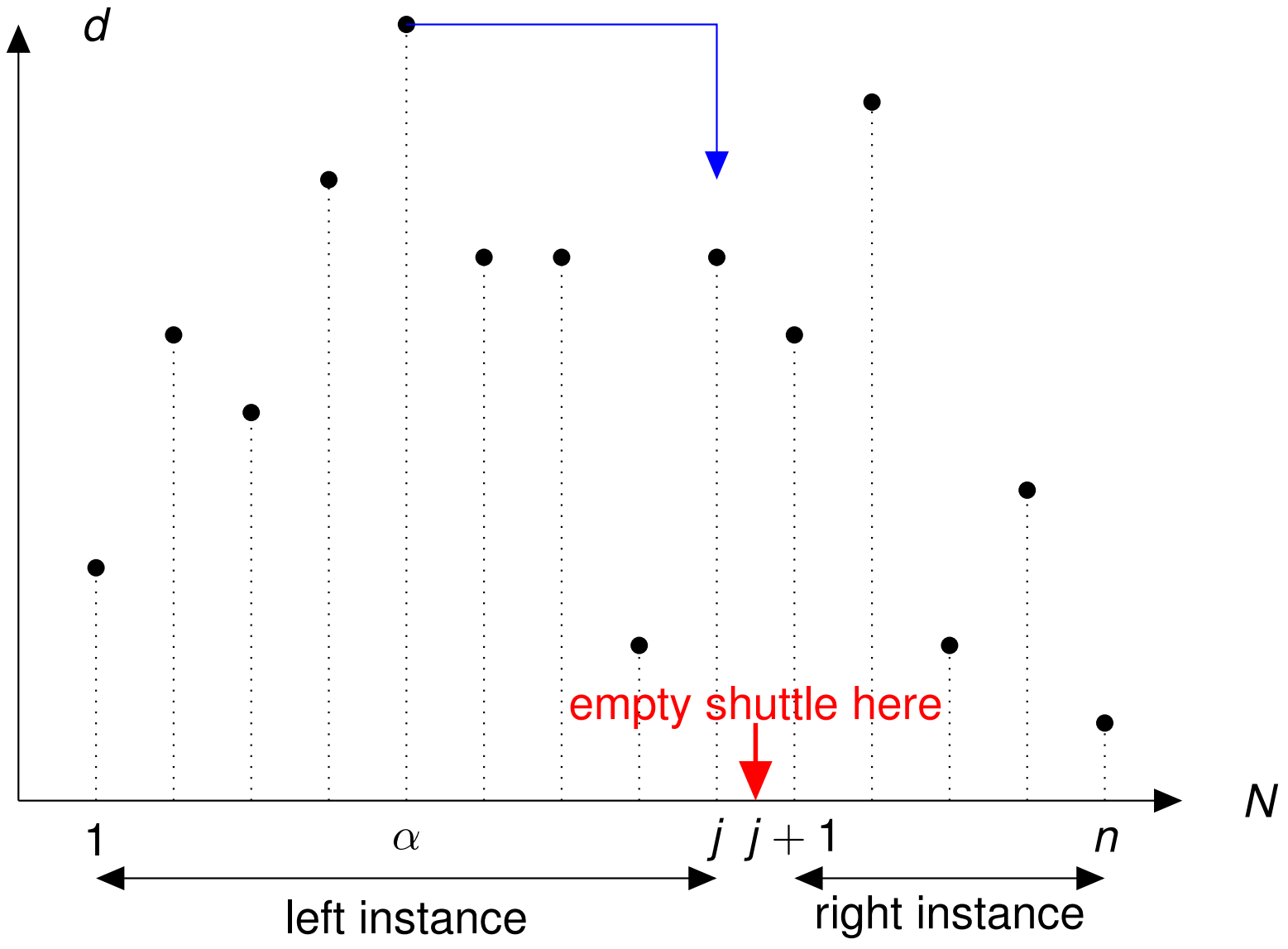
The same definition applies to any sub-interval $[i, \ldots, j]$ in N.

For any subset of items, the leading item exists and is unique.

Definition (Leading subsets)

Let N be any set of items. Let ℓ^1 be the leading item in N and let $L^1 = \{\ell^1\}$. Now, recursively, let ℓ^k be the leading item in $N \backslash L^{k-1}$ for $k \geq 2$. We define L^k as the k-order leading subset of N.

Recursive decomposition



Recursive decomposition

Interval: $[1, \ldots, n]$.
Leading item: $\alpha \in [1, \ldots, n]$.
Insertion position of the leading item: $j \in [\alpha, \ldots, n]$.

The shuttle is empty after delivering the leading item.

Decomposition:

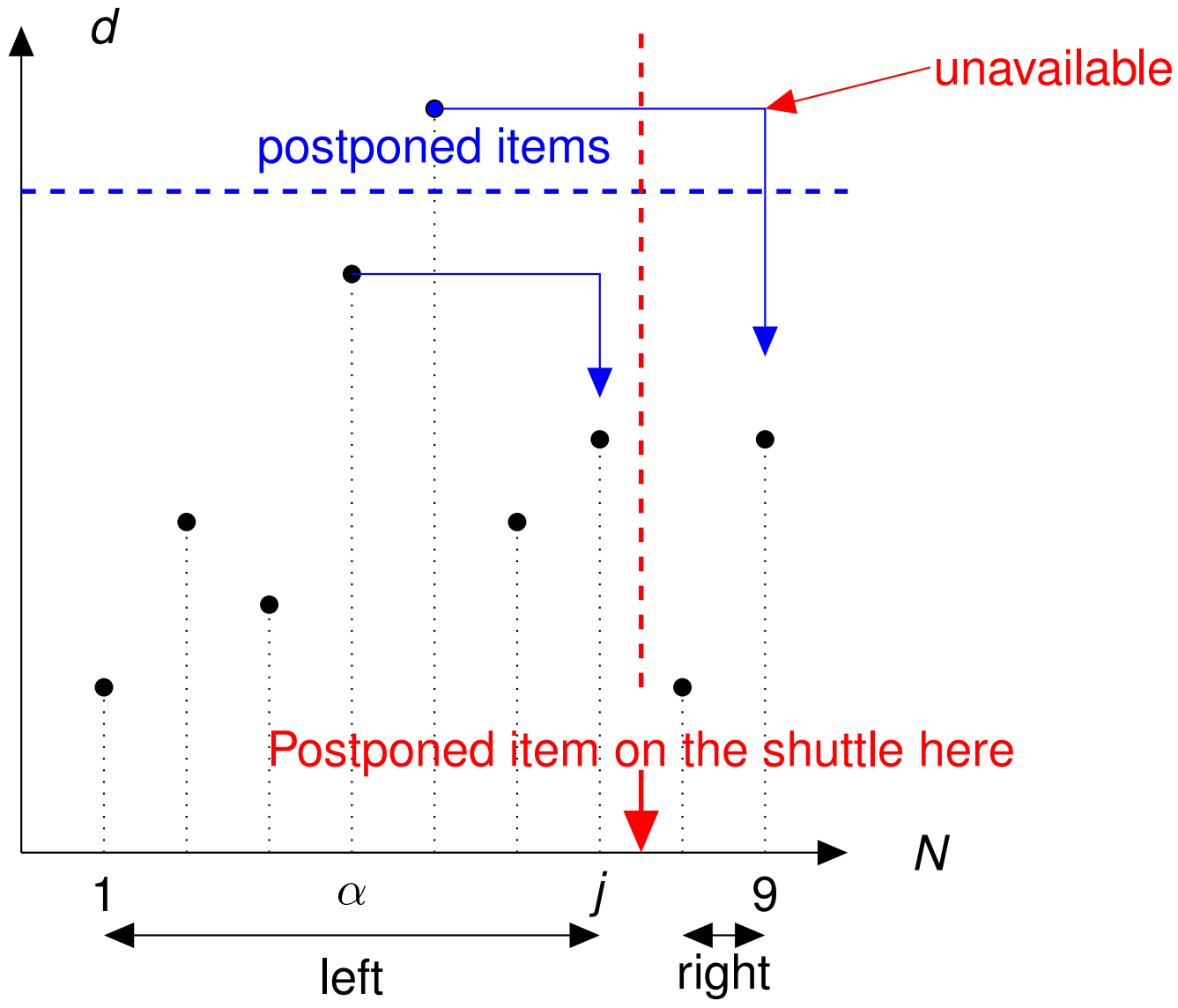
- *left instance* with the items in $[1, j]$;
- *right instance* with the items in $[j + 1, n]$.

The two instances can be solved independently.

Each of them is recursively decomposed.

Base of the recursion: instance with k items and at least k available capacity units.

Postponed items and unavailable units of capacity



Postponed items and unavailable units of capacity

Definition (Significant range)

A non-empty range $[i, j]$ *of consecutive items is significant iff* j *is an insertion position.*

Definition (Postponed item)

When an item in $[i, j]$ *has been assigned an insertion position* $j' \geq j$*, it is postponed for the range* $[i, j]$*.*

All postponed items in a range $[i, j]$ form a leading subset of $\{i, \ldots, j\}$.

Observation

If an item α *is postponed to position* $j > \alpha$*, a corresponding unit of capacity is made unavailable for all ranges* $[i, j]$ *with* $i > \alpha$*.*

Each sub-instance produced by the recursive decomposition, is characterized by a significant range, a n. of postponed items and a n. of unavailable capacity units.

Dynamic programming: state

State: (l, r, u, p), where

- $l, r \in N$: significant range $[l, r]$;
- $u \geq 0$: n. of unavailable units of capacity,
- $p \geq 0$: n. of items in $[l, r]$ to be postponed.

Feasibility conditions (incomplete list):

(i) $l \leq r$: only non-empty ranges are significant;

(ii) $u \leq q - 1$: $u = q$ implies full shuttle;

(iii) $u \leq l - 1$: u items are already on board when l arrives;

(iv) $p \leq q - u$: every postponed item requires one unit and $q - u$ units are available;

(v) $p \leq l - r + 1$: all postponed items of a range belong to it.

States that do not comply with these restrictions are not generated.

Dynamic programming: extension rules

Base of the recursion: (l, r, u, p), with $r - l + 1 \leq q - u$.

All items are delivered in a trip in position r.

Recursive step: (l, r, u, p) with $r - l + 1 > q - u$.

All feasible insertion positions for the leading non-postponed item are tried.

For each state (l, r, u, p) and each feasible insertion position j the corresponding cost $c_j(l, r, u, p)$ is given by the sum of three terms.

$$c(l, r, u, p) = \min_{j=\alpha,\ldots,\bar{j}} \{c_j(l, r, u, p)\} = \min_{j=\alpha,\ldots,\bar{j}} \{C_j^{\alpha} + C_j^{left} + C_j^{right}\}.$$

Bottom-up: states with smaller ranges are evaluated first.

Optimal value: cost of $(1, n, 0, 0)$.

Dynamic programming: cost

First cost term: $d(\alpha)$ iff the insertion position j is used for the first time (α is the farthest item in its trip).

$$C_j^{\alpha} = \begin{cases} d(\alpha) & \text{if } (j < r) \vee ((j = n) \wedge (u = 0) \wedge (p = 0)) \\ 0 & \text{otherwise.} \end{cases}$$

Second cost term: cost of the left sub-instance.

$$C_j^{left} = c(l, j, u, p' + 1).$$

p': n. of postponed items in $[l, r]$ preceding j.

Third cost term: cost of the right sub-instance.

$$C_j^{right} = \begin{cases} c(j + 1, r, u + p', p'') & \text{if } (j < r) \\ 0 & \text{otherwise.} \end{cases}$$

p'': n.of postponed items in $[l, r]$ following j.

Complexity

Preprocessing: $O(n^2 q)$ both in time and space.

- N. of possible $[l, r]$ pairs: $O(n^2)$;
- u and p are non-negative integers $\leq q$;

N. of states: $O(n^2 q^2)$, upper bounded by $O(n^4)$ (space complexity).

The time needed to compute the cost for each state: $O(n)$ (n. insertion positions).

Time complexity of the algorithm: $O(n^3 q^2)$, upper bounded by $O(n^5)$ (strongly polynomial).

WOW!

Computational results

100 random instances $\forall(n, q)$: $5 \leq n \leq 100$, $2 \leq q \leq 0.95n$.

n	q	time (s)
10	2	0.000
10	3	0.000
10	5	0.001
20	2	0.001
20	5	0.001
20	10	0.001
50	2	0.002
50	5	0.003
50	15	0.016
50	25	0.034
100	2	0.006
100	10	0.057
100	25	0.286
100	50	1.115