Acknowledgment: Thomas Magnanti, Retsef Levi

Faster Primal-Dual Algorithms for the Economic Lot-Sizing Problem

Mihai Pătraşcu
AT&T Research

Dan Stratila
RUTCOR and Rutgers Business School
Rutgers University

ISMP XX
25 August 2009

Outline

1. Models:
   a. The economic lot-sizing problem.
   b. An important special case: the non-speculative condition.

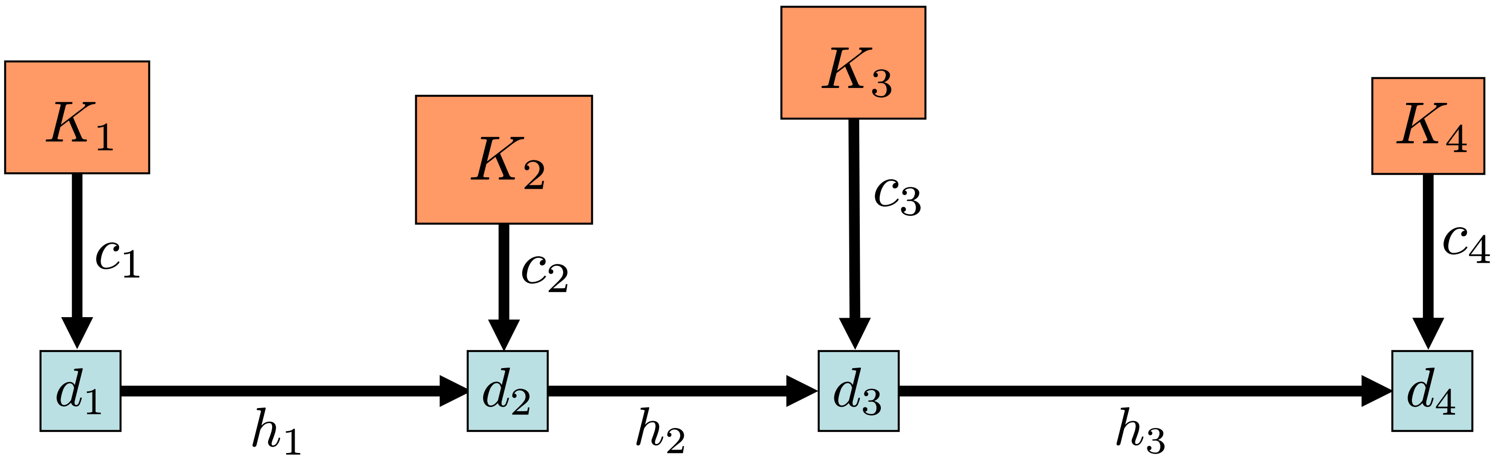
2. Overview of previous results.

3. Review of “wave” primal-dual algorithm of Levi et al (2006).

4. Main result: a faster algorithm for the lot-sizing problem.

5. Conclusions:
   a. Connection with basic algorithms problems. O(n)?
   b. Wave primal-dual applicable to other inventory problems.

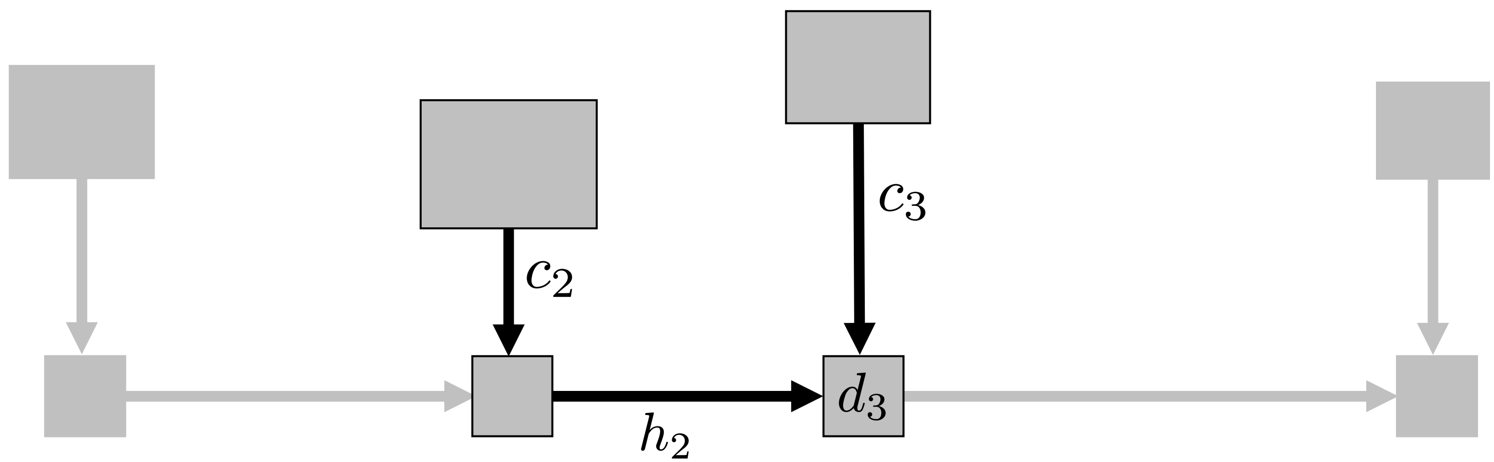
The Economic Lot-Sizing Problem



$h_{st} := c_s + \sum_{i=s}^{t-1} h_i$

$$\begin{aligned} \min \ & \sum_{t=1}^{n} K_t y_t + \sum_{s=1}^{n} \sum_{t=s}^{n} h_{st} d_t x_{st} \\ \text{s.t.} \ & \sum_{s=1}^{t} x_{st} = 1, \quad 1 \le t \le n, \\ & 0 \le x_{st} \le y_s, \quad 1 \le s \le t \le n. \end{aligned}$$

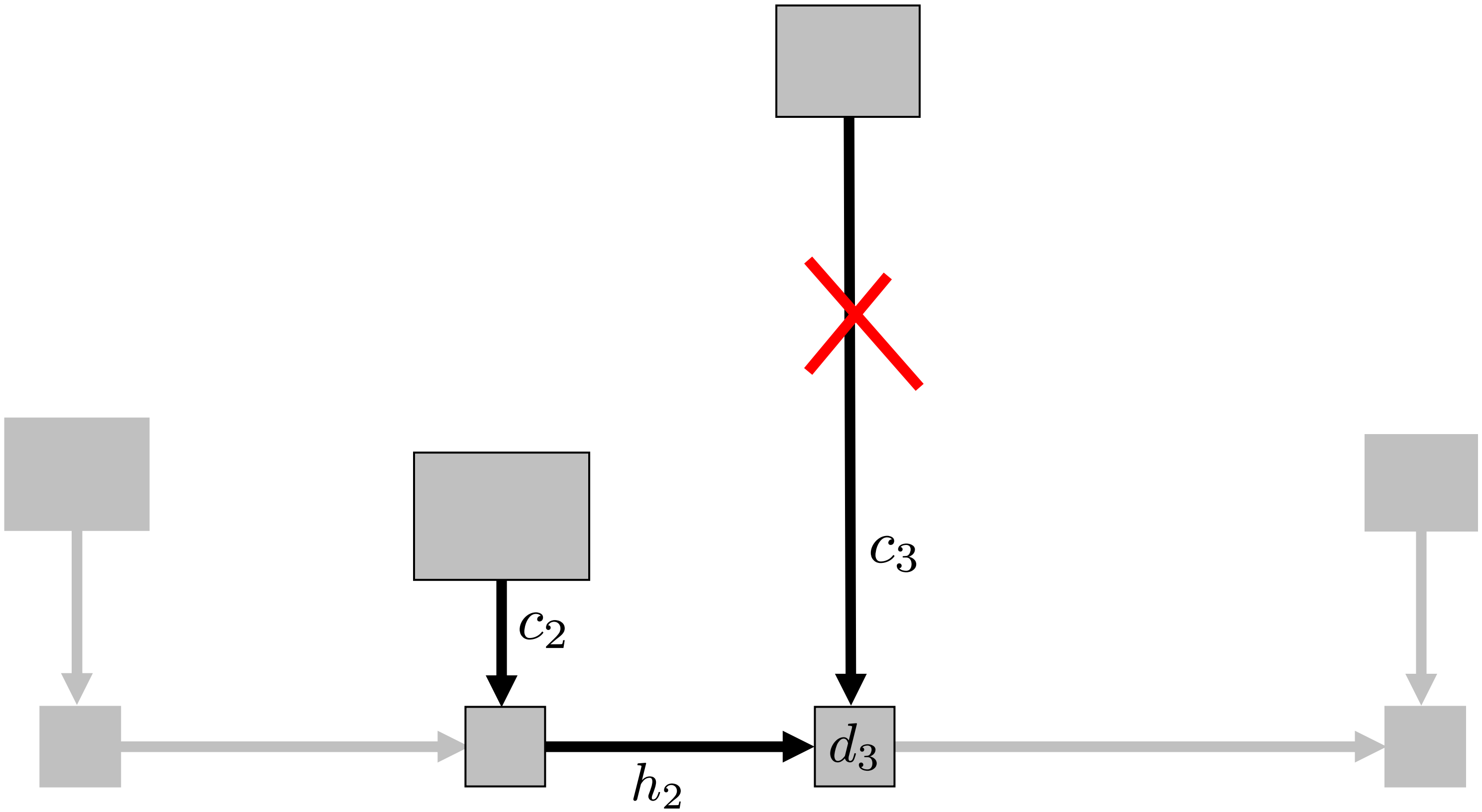
An Important Special Case



Non-speculative condition:

$$c_3 \leq c_2 + h_2$$

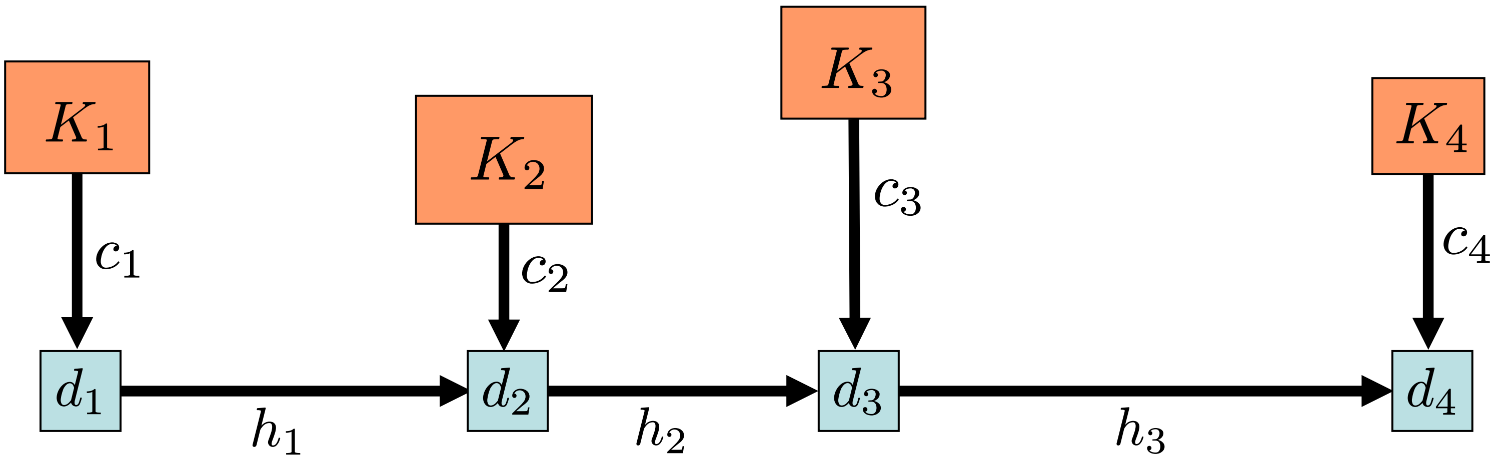
An Important Special Case



Non-speculative condition:

$$c_3 \not\leq c_2 + h_2$$

Lot-Sizing Problem with Non-Speculative Condition



$c_{t+1} \le c_t + h_t$
for $t = 1, \ldots, n-1$

$$\begin{aligned} \min \quad & \sum_{t=1}^{n} K_t y_t + \sum_{s=1}^{n} \sum_{t=s}^{n} h_{st} d_t x_{st} \\ \text{s.t.} \quad & \sum_{s=1}^{t} x_{st} = 1, \quad 1 \le t \le n, \\ & 0 \le x_{st} \le y_s, \quad 1 \le s \le t \le n. \end{aligned}$$

Outline

1. Models:
   a. The economic lot-sizing problem.
   b. An important special case: the non-speculative condition.

2. Overview of previous results.

3. Review of “wave” primal-dual algorithm of Levi et al (2006).

4. Main result: a faster algorithm for the lot-sizing problem.

5. Conclusions:
   a. Connection with basic algorithms problems. O(n)?
   b. Wave primal-dual applicable to other inventory problems.

Previous Results on Lot-Sizing

Early research:

- Introduced by Manne ('58), Wagner and Whitin ('58).
- Wagner and Whitin ('58) provide $O(n^2)$ for non-speculative case.
- Zabel ('64), Eppen et al ('69) obtain $O(n^2)$ for general case.
- Results on *heuristics,* to be more efficient than $O(n^2)$ in '70s-'80s.

The $O(n \log n)$ algorithms:

- Federgruen and Tzur ('91), forward algorithm.
- Wagelmans et al ('92), backward algorithm, also relate to dual.
- Aggarwal and Park ('93), using Monge arrays, generalizations.
- These 3 algorithms run in $O(n)$ for non-speculative case.

Non-algorithmic results:

- Krarup and Bilde ('77) show that above formulation is integral.
- Polyhedral results for harder versions, e.g. multi-item lot-sizing: *Production Planning by Mixed Integer Programming*, Pochet and Wolsey ('06).

Primal-Dual Algorithms for Lot-Sizing

Levi et al ('06) :

- Obtain primal-dual algorithm for lot-sizing problem.
- Proves above formulation is integral as a consequence.
- Primal-dual 2-approxim. algorithms for joint replenishment problem (JRP), and for multistage assembly problem.
- Algorithms clearly polynomial, authors do not estimate running times.

Related primal-dual algorithms for facility location:

- Primal-dual algorithms of Levi et al for inventory is rooted in primal-dual approximation algorithm for facility location of Jain and Vazirani ('01).
- Thorup ('03) obtains 1.62 algorithm approximation algorithm for facility location with running time $\tilde{O}(m+n)$ based on Jain and Vazirani ('01) and Jain et al ('03).

Differences:

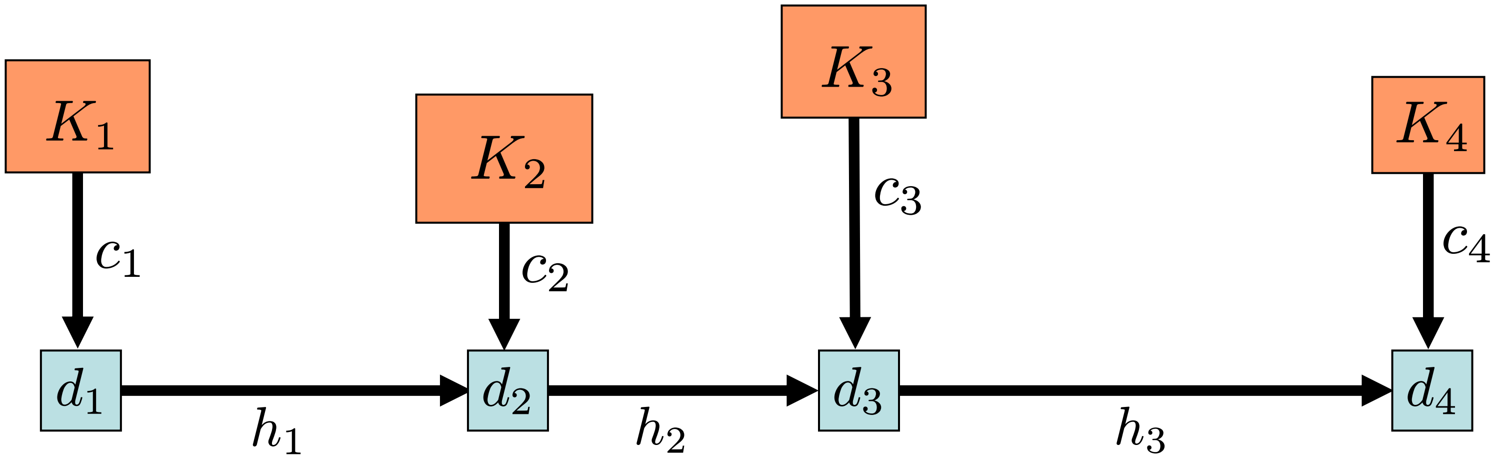
- Algorithms for facility location and lot-sizing are different.
- Input size is different: lot-sizing represented as facility would have $O(n^2)$ edges.

Outline

1. Models:
    a. The economic lot-sizing problem.
    b. An important special case: the non-speculative condition.
2. Overview of previous results.
3. Review of “wave” primal-dual algorithm of Levi et al (2006).
4. Main result: a faster algorithm for the lot-sizing problem.
5. Conclusions:
    a. Connection with basic algorithms problems. O(n)?
    b. Wave primal-dual applicable to other inventory problems.

(Modified, Levi et al '06)

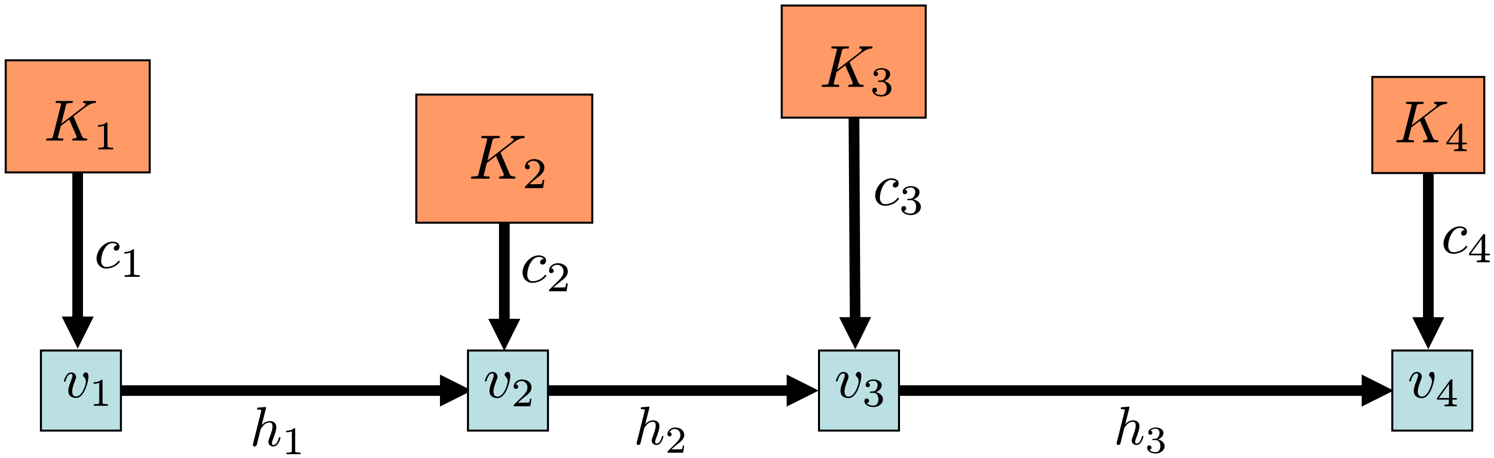
Wave Primal-Dual Algorithm for Lot-Sizing



Primal LP

$$\begin{aligned} \min \quad & \sum_{s=1}^{n} K_s y_s + \sum_{s=1}^{n} \sum_{t=s}^{n} h_{st} d_t x_{st} & \\ \text{s.t.} \quad & \sum_{s=1}^{t} x_{st} = 1, & 1 \le t \le n, \\ & 0 \le x_{st} \le y_s, & 1 \le s \le t \le n. \end{aligned}$$

Wave Primal-Dual Algorithm for Lot-Sizing

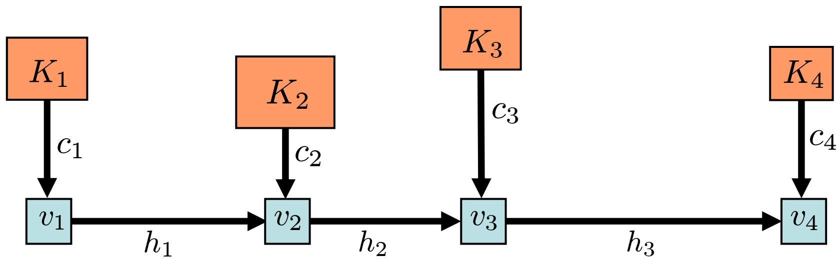


Dual LP

$$
\begin{aligned}
\max \sum_{t=1}^{n} v_t, & \\
\text{s.t. } v_t \le h_{st} d_t + w_{st}, & \qquad 1 \le s \le t \le n, \\
\sum_{t=s}^{n} w_{st} \le K_s, & \qquad 1 \le s \le n, \\
w_{st} \ge 0, & \qquad 1 \le s \le t \le n.
\end{aligned}
$$

Intuition

- v_i called budgets
- w_{st} allocate K_s to demand points t
- Feas. sol. $w_{st} = \max\{0, h_{st} d_t - v_t\}$



Dual LP

$$\max \sum_{t=1}^{n} v_t,$$

$$\text{s.t. } \underline{v_t} \le \underline{h_{st}d_t + w_{st}}, \qquad 1 \le s \le t \le n,$$

$$\underline{\sum_{t=s}^{n} w_{st}} \le \underline{K_s}, \qquad 1 \le s \le n,$$

$$w_{st} \ge 0, \qquad 1 \le s \le t \le n.$$

Note

- v_i called budgets
- w_{st} allocate K_s to demand points t
- Feas. sol. $w_{st} = \max\{0, h_{st}d_t - v_t\}$

General

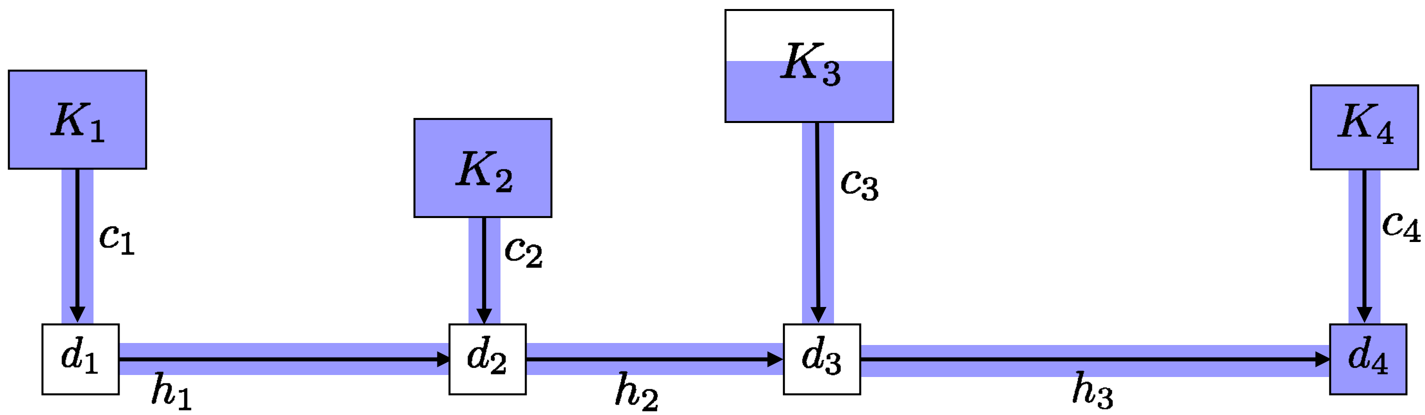
- Start with (v,w)=0 and (x,y)=0
- Iteratively increase dual solution, increasing dual objective
- At the same time, construct primal solution
- When dual objective cannot be increased, primal is feasible
- Post-processing step, $O(n)$

More details

- To increase dual objective, we increase budgets v_t
- At some point, some of the constraints become tight
- To keep increasing v_t we begin increasing w_{st} for those constraints
- At some point some of the constraints become tight
    - Open s in the primal
    - Freeze all corresponding v_t
    - Demands t with w_{st}>0 assigned to order s

Remaining free choice—order in which v_t are increased

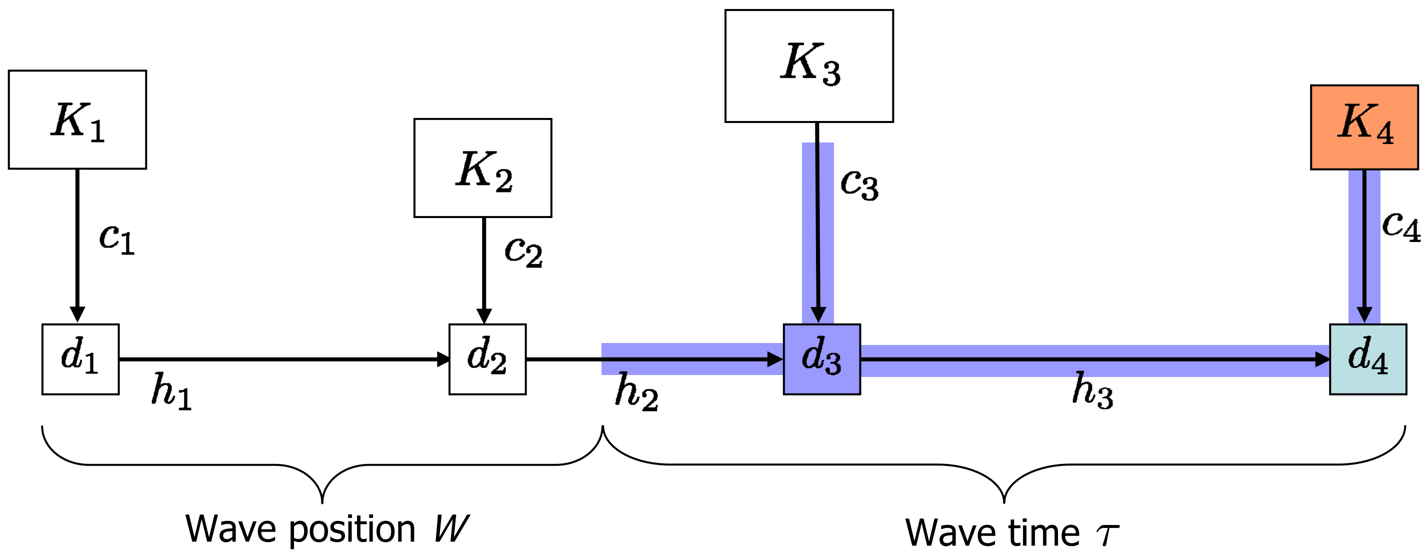
The Wave

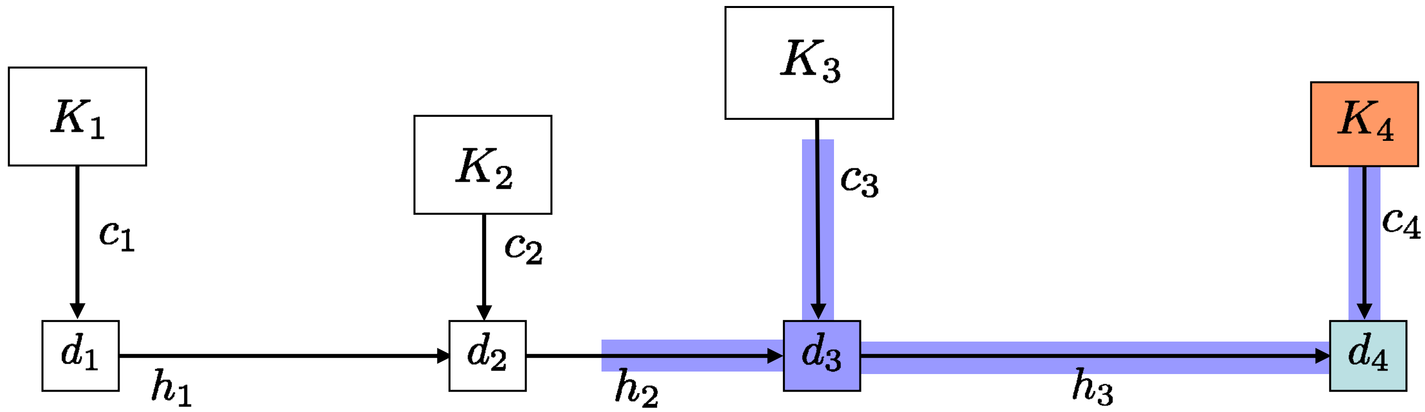


- start increasing v_4
- start increasing w_{44}
- start increasing v_3
- order 4 opens, budget 4 freezes
  demand 4 assigned to order 4
- start increasing w_{33}
- start increasing v_2
- start increasing w_{22} and w_{32}
- order 2 opens, budgets 2 & 3 freeze
  demands 2 & 3 assigned to order 2
- start increasing v_1
- start increasing w_{11}
- order 1 opens, budgets 1 freezes,
  demand 1 assigned to order 1

Types of events:
- Demand point becomes active
- Budget begins contributing to K_t
- Order point becomes tight & assoc.

Wave Time and Position





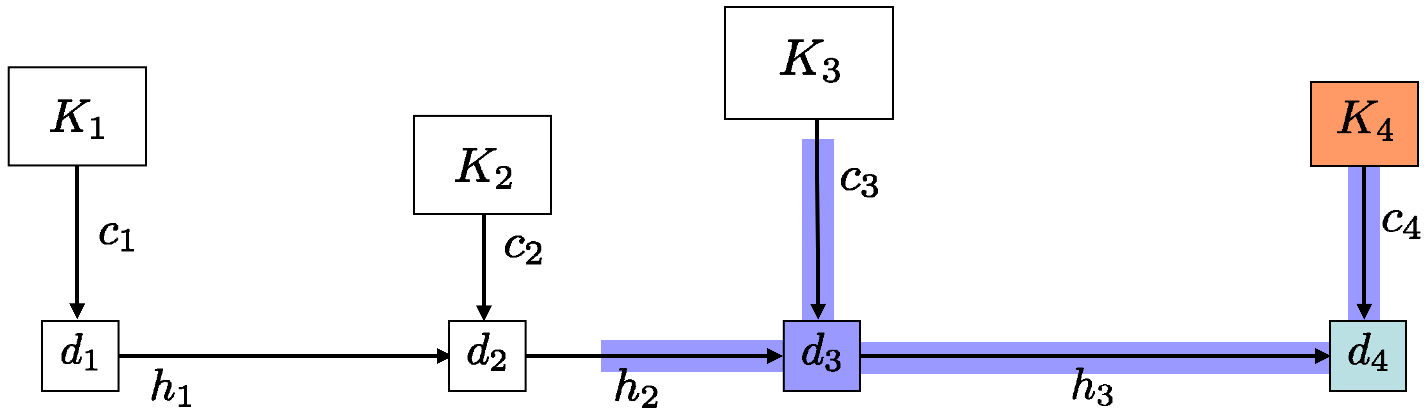
ALGORITHM LSPD($n \in \mathbb{Z}_+$; $c, f, d \in \mathbb{Q}^n_+, h \in \mathbb{Q}^{n-1}_+$)

(1) Start with the wave at $W = h_{1n}$ and the dual solution $(v, w) = 0$. All orders are closed, and all demand points are unserved, i.e. $(x, y) = 0$.
(2) **While** there are unserved demand points:
(3) Decrease W continuously. At the same time increase v_t and w_{st} for unserved demand points t so as to maintain $v_t = \max\{0, d_t(h_{1t} - W)\}$ and $w_{st} = \max\{0, v_t - (c_s + h_{st})d_t\}$. The wave stops when an order becomes tight.
(4) Open the order s that became tight. For each unserved demand point t contributing to s, serve t from s.
(5) **For** each open order s from 1 to n:
(6) If there is a demand point t that contributes to s and to another open order s' with $s' < s$, close s. Reassign all demand points previously served from s to s'.
(7) Return (x, y) and (v, w).

Events:
- Demand point becomes active
- Budget begins contributing to K_t
- Order point becomes tight & assoc.

Algorithms for digital computers:
- Goal at end of execution to obtain primal *y* and dual *v*.
- Execute (3) from event to event, instead of continuously

Step (3):
- Compute wave position W^* when next order point becomes tight.
- Update W:=W^*, then update *v* and *w*.



ALGORITHM LSPD($n \in \mathbb{Z}_+$; $c, f, d \in \mathbb{Q}_+^n$, $h \in \mathbb{Q}_+^{n-1}$)

(1) Start with the wave at $W = h_{1n}$ and the dual solution $(v, w) = 0$. All orders are closed, and all demand points are unserved, i.e. $(x, y) = 0$.

(2) **While** there are unserved demand points:

(3) Decrease W continuously. At the same time increase v_t and w_{st} for unserved demand points t so as to maintain $v_t = \max\{0, d_t(h_{1t} - W)\}$ and $w_{st} = \max\{0, v_t - (c_s + h_{st})d_t\}$. The wave stops when an order becomes tight.

(4) Open the order s that became tight. For each unserved demand point t contributing to s, serve t from s.

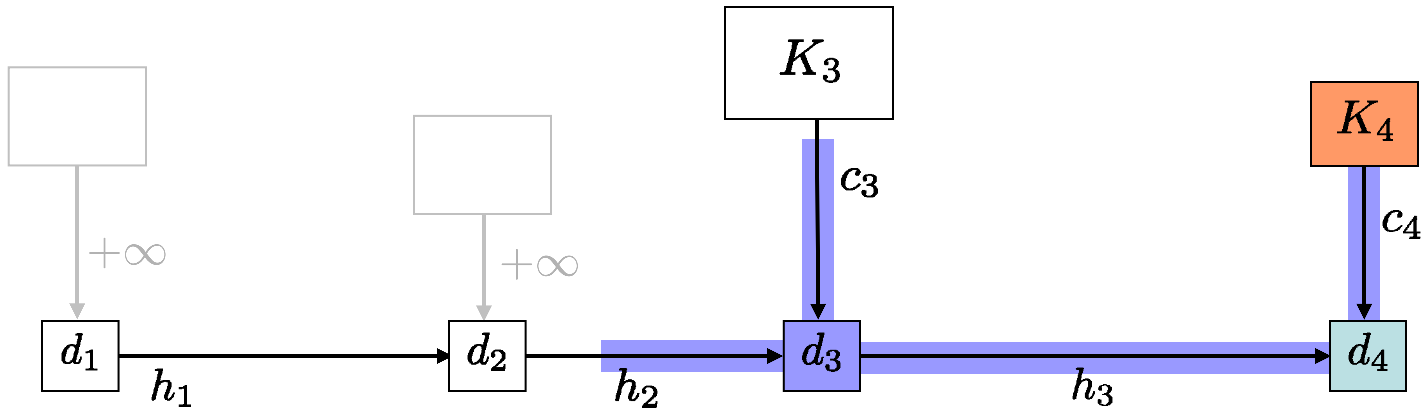
(5) **For** each open order s from 1 to n:

(6) If there is a demand point t that contributes to s and to another open order s' with $s' < s$, close s. Reassign all demand points previously served from s to s'.

(7) Return (x, y) and (v, w).

One iteration of (3):

- For each unfrozen order point t, compute position W_t when it becomes tight, assuming no other unfrozen order points become tight in the meantime.



ALGORITHM LSPD($n \in \mathbb{Z}_+$; $c, f, d \in \mathbb{Q}_+^n, h \in \mathbb{Q}_+^{n-1}$)

(1) Start with the wave at $W = h_{1n}$ and the dual solution $(v, w) = 0$. All orders are closed, and all demand points are unserved, i.e. $(x, y) = 0$.

(2) **While** there are unserved demand points:

(3) Decrease W continuously. At the same time increase v_t and w_{st} for unserved demand points t so as to maintain $v_t = \max\{0, d_t(h_{1t} - W)\}$ and $w_{st} = \max\{0, v_t - (c_s + h_{st})d_t\}$. The wave stops when an order becomes tight.

(4) Open the order s that became tight. For each unserved demand point t contributing to s, serve t from s.

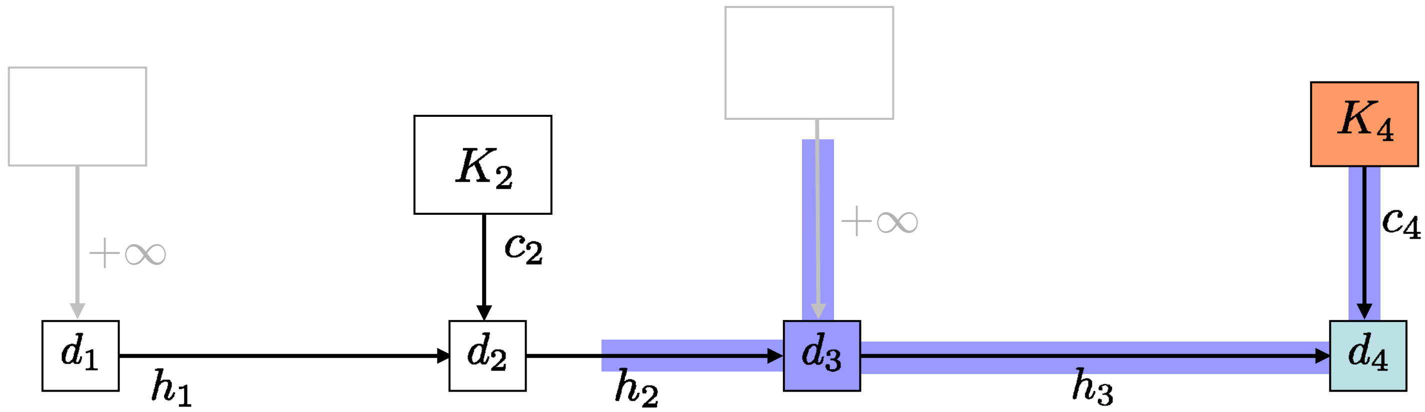
(5) **For** each open order s from 1 to n:

(6) If there is a demand point t that contributes to s and to another open order s' with $s' < s$, close s. Reassign all demand points previously served from s to s'.

(7) Return (x, y) and (v, w).

One iteration of (3):

- For each unserved order point t, compute position W_t when it becomes tight, assuming no other unfrozen order points become tight in the meantime.
- Compute W_3



ALGORITHM LSPD($n \in \mathbb{Z}_+$; $c, f, d \in \mathbb{Q}_+^n, h \in \mathbb{Q}_+^{n-1}$)

(1) Start with the wave at $W = h_{1n}$ and the dual solution $(v, w) = 0$. All orders are closed, and all demand points are unserved, i.e. $(x, y) = 0$.

(2) **While** there are unserved demand points:

(3) Decrease W continuously. At the same time increase v_t and w_{st} for unserved demand points t so as to maintain $v_t = \max\{0, d_t(h_{1t} - W)\}$ and $w_{st} = \max\{0, v_t - (c_s + h_{st})d_t\}$. The wave stops when an order becomes tight.

(4) Open the order s that became tight. For each unserved demand point t contributing to s, serve t from s.

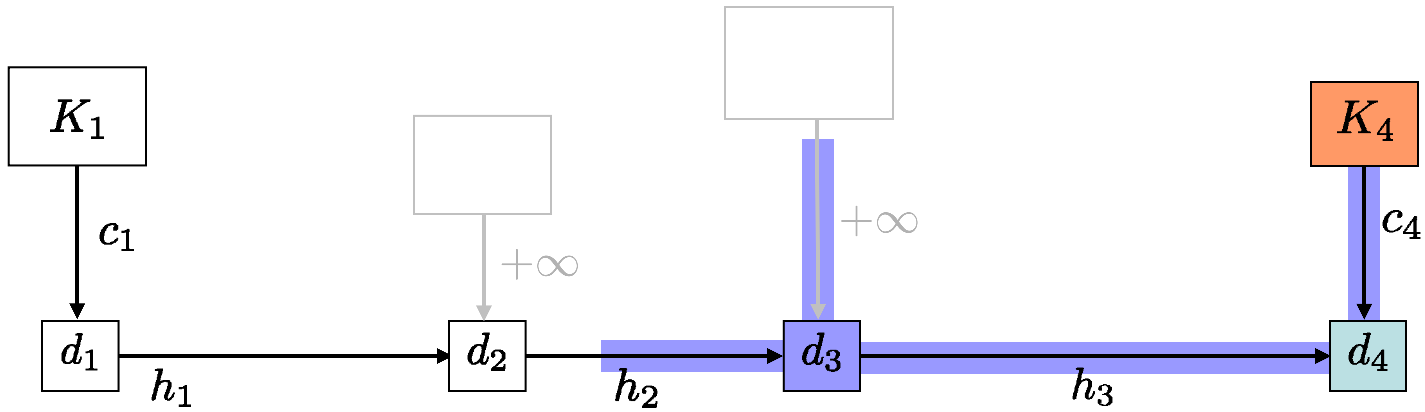
(5) **For** each open order s from 1 to n:

(6) If there is a demand point t that contributes to s and to another open order s' with $s' < s$, close s. Reassign all demand points previously served from s to s'.

(7) Return (x, y) and (v, w).

One iteration of (3):

- For each unfrozen order point t, compute position W_t when it becomes tight, assuming no other unfrozen order points become tight in the meantime.
- Compute W_3
- Compute W_2



ALGORITHM LSPD($n \in \mathbb{Z}_+$; $c, f, d \in \mathbb{Q}^n_+, h \in \mathbb{Q}^{n-1}_+$)

(1) Start with the wave at $W = h_{1n}$ and the dual solution $(v, w) = 0$. All orders are closed, and all demand points are unserved, i.e. $(x, y) = 0$.

(2) **While** there are unserved demand points:

(3) Decrease W continuously. At the same time increase v_t and w_{st} for unserved demand points t so as to maintain $v_t = \max\{0, d_t(h_{1t} - W)\}$ and $w_{st} = \max\{0, v_t - (c_s + h_{st})d_t\}$. The wave stops when an order becomes tight.

(4) Open the order s that became tight. For each unserved demand point t contributing to s, serve t from s.

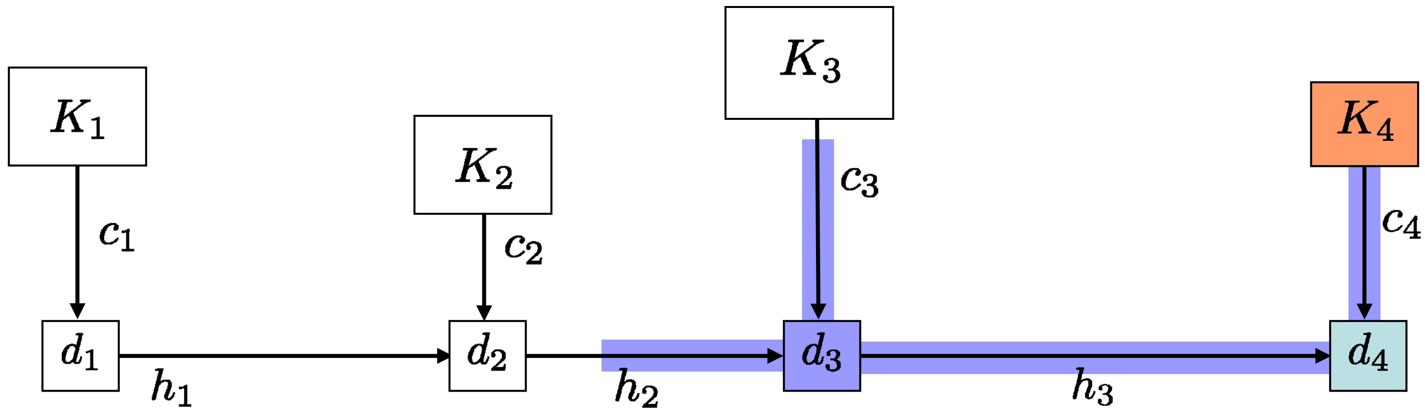
(5) **For** each open order s from 1 to n:

(6) If there is a demand point t that contributes to s and to another open order s' with $s' < s$, close s. Reassign all demand points previously served from s to s'.

(7) Return (x, y) and (v, w).

One iteration of (3):

- For each unfrozen order point t, compute position W_t when it becomes tight, assuming no other unfrozen order points become tight in the meantime.
- Compute W_3
- Compute W_2
- Compute W_1



ALGORITHM LSPD($n \in \mathbb{Z}_+$; $c, f, d \in \mathbb{Q}^n_+, h \in \mathbb{Q}^{n-1}_+$)

(1) Start with the wave at $W = h_{1n}$ and the dual solution $(v, w) = 0$. All orders are closed, and all demand points are unserved, i.e. $(x, y) = 0$.

(2) **While** there are unserved demand points:

(3) Decrease W continuously. At the same time increase v_t and w_{st} for unserved demand points t so as to maintain $v_t = \max\{0, d_t(h_{1t} - W)\}$ and $w_{st} = \max\{0, v_t - (c_s + h_{st})d_t\}$. The wave stops when an order becomes tight.

(4) Open the order s that became tight. For each unserved demand point t contributing to s, serve t from s.

(5) **For** each open order s from 1 to n:

(6) If there is a demand point t that contributes to s and to another open order s' with $s' < s$, close s. Reassign all demand points previously served from s to s'.

(7) Return (x, y) and (v, w).

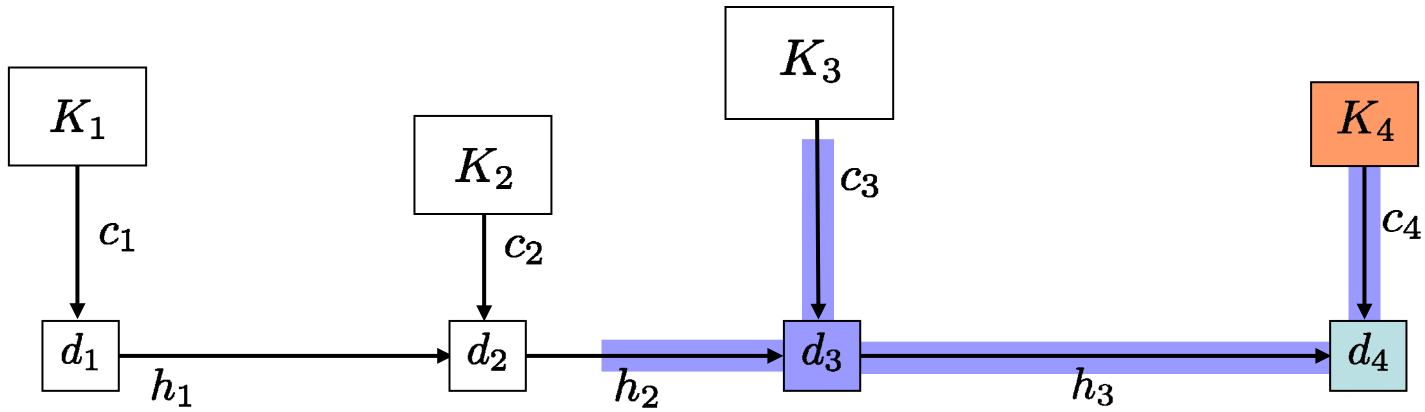
One iteration of (3):

- For each unfrozen order point t, compute position W_t when it becomes tight, assuming no other unfrozen order points become tight in the meantime.
- Set $W' = \min\{ W_t : t \text{ is unfrozen} \}$

Lemma: $W' = W^*$, the position when the next order point becomes tight. W_t that yields the minimum corresponds to the next order point that becomes tight.

Running time $O(n^3)$:

- One computation of W_t takes $O(n)$.
- $O(n)$ computations before an order point becomes tight.
- At most $O(n)$ order points become tight.



ALGORITHM LSPD($n \in \mathbb{Z}_+$; $c, f, d \in \mathbb{Q}_+^n, h \in \mathbb{Q}_+^{n-1}$)

(1) Start with the wave at $W = h_{1n}$ and the dual solution $(v, w) = 0$. All orders are closed, and all demand points are unserved, i.e. $(x, y) = 0$.

(2) **While** there are unserved demand points:

(3) Decrease W continuously. At the same time increase v_t and w_{st} for unserved demand points t so as to maintain $v_t = \max\{0, d_t(h_{1t} - W)\}$ and $w_{st} = \max\{0, v_t - (c_s + h_{st})d_t\}$. The wave stops when an order becomes tight.

(4) Open the order s that became tight. For each unserved demand point t contributing to s, serve t from s.

(5) **For** each open order s from 1 to n:

(6) If there is a demand point t that contributes to s and to another open order s' with $s' < s$, close s. Reassign all demand points previously served from s to s'.

(7) Return (x, y) and (v, w).

One iteration of (3):

- For each unfrozen order point t, compute position W_t when it becomes tight, assuming no other unfrozen order points become tight in the meantime.
- Set $W' = \min\{ W_t : t \text{ is unfrozen} \}$

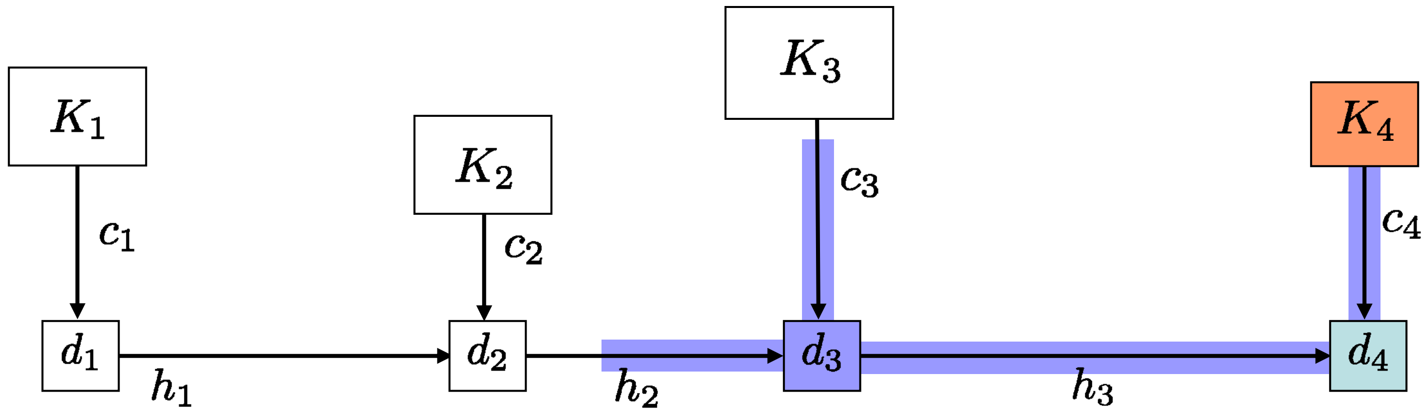
Lemma: $W' = W^*$, the position when the next order point becomes tight. W_t that yields the minimum corresponds to the next order point that becomes tight.

Running time $O(n^2)$:

- With preprocessing W_t takes $O(1)$ amortized.
- $O(n)$ computations before an order point becomes tight.
- At most $O(n)$ order points become tight.

Outline

1. Models:
   a. The economic lot-sizing problem.
   b. An important special case: the non-speculative condition.
2. Overview of previous results.
3. Review of "wave" primal-dual algorithm of Levi et al (2006).
4. Main result: a faster algorithm for the lot-sizing problem.
5. Conclusions:
   a. Connection with basic algorithms problems. O(n)?
   b. Wave primal-dual applicable to other inventory problems.



ALGORITHM LSPD($n \in \mathbb{Z}_+$; $c, f, d \in \mathbb{Q}_+^n, h \in \mathbb{Q}_+^{n-1}$)

(1) Start with the wave at $W = h_{1n}$ and the dual solution $(v, w) = 0$. All orders are closed, and all demand points are unserved, i.e. $(x, y) = 0$.

(2) **While** there are unserved demand points:

(3) Decrease W continuously. At the same time increase v_t and w_{st} for unserved demand points t so as to maintain $v_t = \max\{0, d_t(h_{1t} - W)\}$ and $w_{st} = \max\{0, v_t - (c_s + h_{st})d_t\}$. The wave stops when an order becomes tight.

(4) Open the order s that became tight. For each unserved demand point t contributing to s, serve t from s.

(5) **For** each open order s from 1 to n:

(6) If there is a demand point t that contributes to s and to another open order s' with $s' < s$, close s. Reassign all demand points previously served from s to s'.

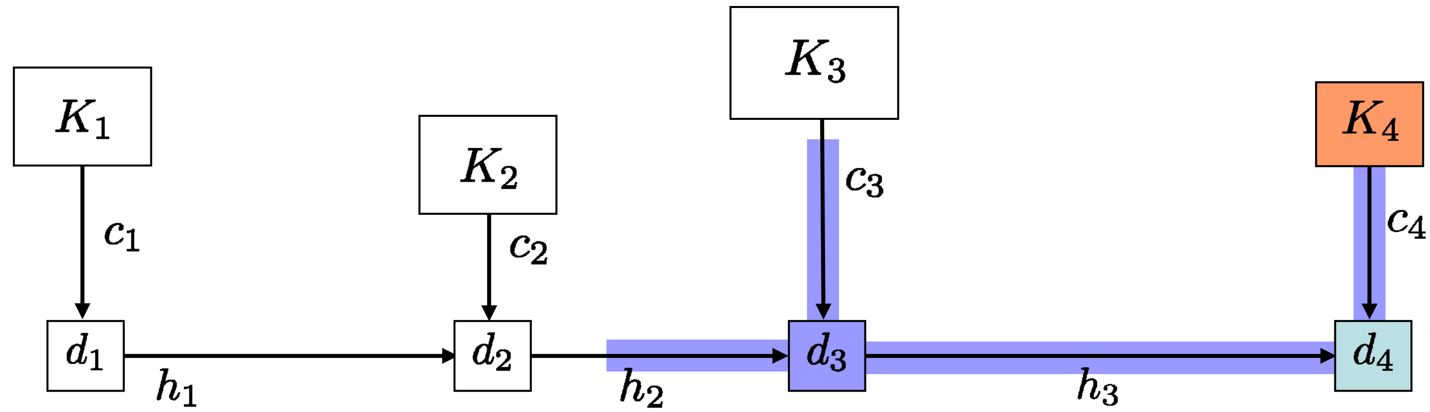
(7) Return (x, y) and (v, w).

Events:

- Demand point becomes active
- Budget begins contributing to K_t
- Order point becomes tight & assoc

Algorithms for digital computers:

- Goal at end of execution to obtain primal *y* and dual *v*.
- We will have “tentative” executions of (3), which may be incorrect.
- When we realize an execution is incorrect, we go back and delete it.
- Algorithm terminates => remaining executions guaranteed correct.



ALGORITHM LSPD$(n \in \mathbb{Z}_+;\ c, f, d \in \mathbb{Q}_+^n, h \in \mathbb{Q}_+^{n-1})$

(1) Start with the wave at $W = h_{1n}$ and the dual solution $(v, w) = 0$. All orders are closed, and all demand points are unserved, i.e. $(x, y) = 0$.

(2) **While** there are unserved demand points:

(3) Decrease W continuously. At the same time increase v_t and w_{st} for unserved demand points t so as to maintain $v_t = \max\{0, d_t(h_{1t} - W)\}$ and $w_{st} = \max\{0, v_t - (c_s + h_{st})d_t\}$. The wave stops when an order becomes tight.

(4) Open the order s that became tight. For each unserved demand point t contributing to s, serve t from s.

(5) **For** each open order s from 1 to n:

(6) If there is a demand point t that contributes to s and to another open order s' with $s' < s$, close s. Reassign all demand points previously served from s to s'.

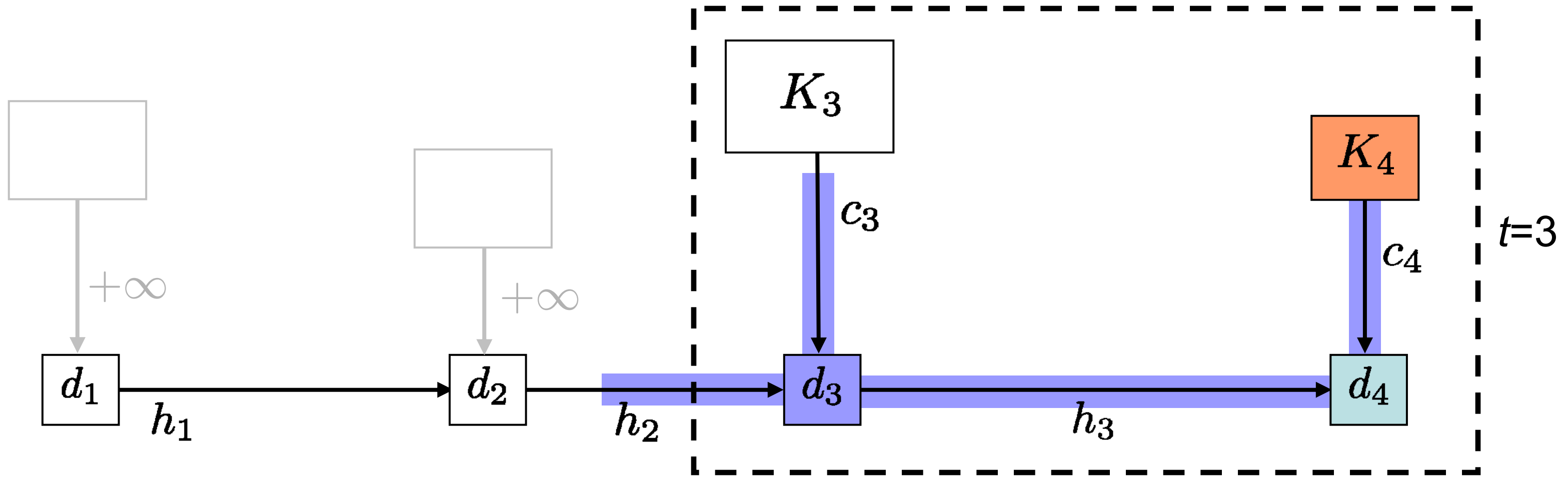
(7) Return (x, y) and (v, w).

Additional data structure:

- Stack $D = (o_k, o_{k-1}, \ldots, o_1)$ of provisionally tight order points.
- Stack initially empty, at end of loop (2) will contain the correct order points.

Algorithm:

1) Start with D=(), t=n.
2) While $t \geq 1$:
   2.1) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t$-1.
   2.2) If t becomes tight before o_k, delete o_k from stack D, and go to 2.1.
   2.3) Add t to stack D, set t:=t-1.



ALGORITHM LSPD($n \in \mathbb{Z}_+$; $c, f, d \in \mathbb{Q}_+^n, h \in \mathbb{Q}_+^{n-1}$)

(1) Start with the wave at $W = h_{1n}$ and the dual solution $(v, w) = 0$. All orders are closed, and all demand points are unserved, i.e. $(x, y) = 0$.

(2) **While** there are unserved demand points:

(3) Decrease W continuously. At the same time increase v_t and w_{st} for unserved demand points t so as to maintain $v_t = \max\{0, d_t(h_{1t} - W)\}$ and $w_{st} = \max\{0, v_t - (c_s + h_{st})d_t\}$. The wave stops when an order becomes tight.

(4) Open the order s that became tight. For each unserved demand point t contributing to s, serve t from s.

(5) **For** each open order s from 1 to n:

(6) If there is a demand point t that contributes to s and to another open order s' with $s' < s$, close s. Reassign all demand points previously served from s to s'.

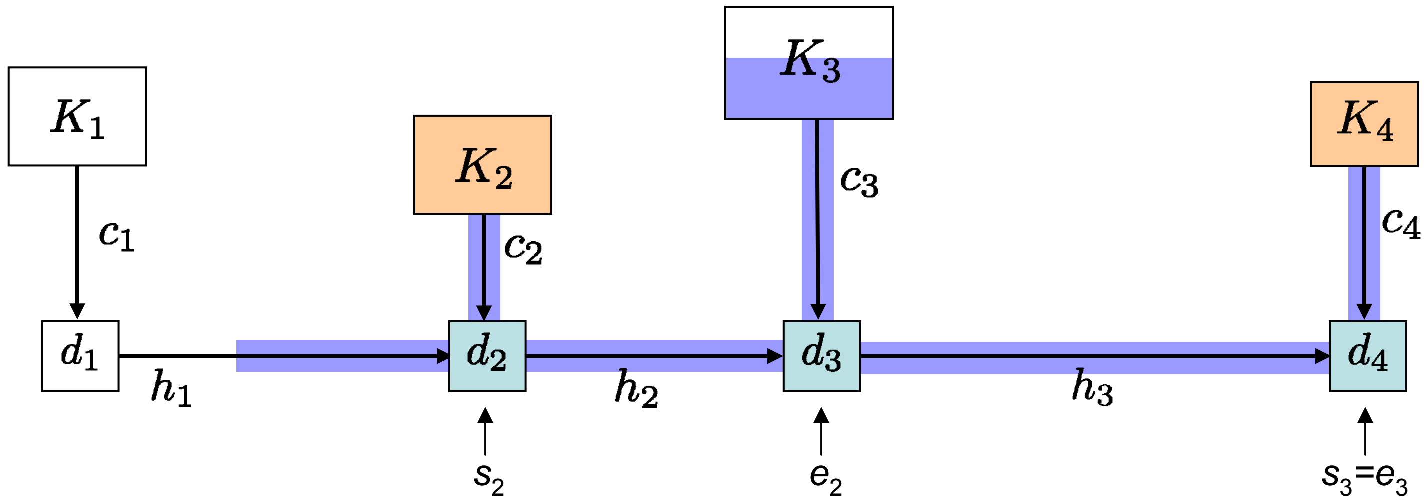
(7) Return (x, y) and (v, w).

Additional data structure:

- Stack $D = (o_k, o_{k-1}, \ldots, o_1)$ of provisionally tight order points.
- Stack initially empty, at end of loop (2) will contain the correct order points.

Algorithm:

1) Start with D=(), t=n.
2) While t≥1:
   - 2.1) Compute time when order point t becomes tight, taking into account periods t, …, n but ignoring periods 1, …, t-1.
   - 2.2) If t becomes tight before o_k, delete o_k from stack D, and go to 2.1.
   - 2.3) Add t to stack D, set t:=t-1.



Stack data structure:

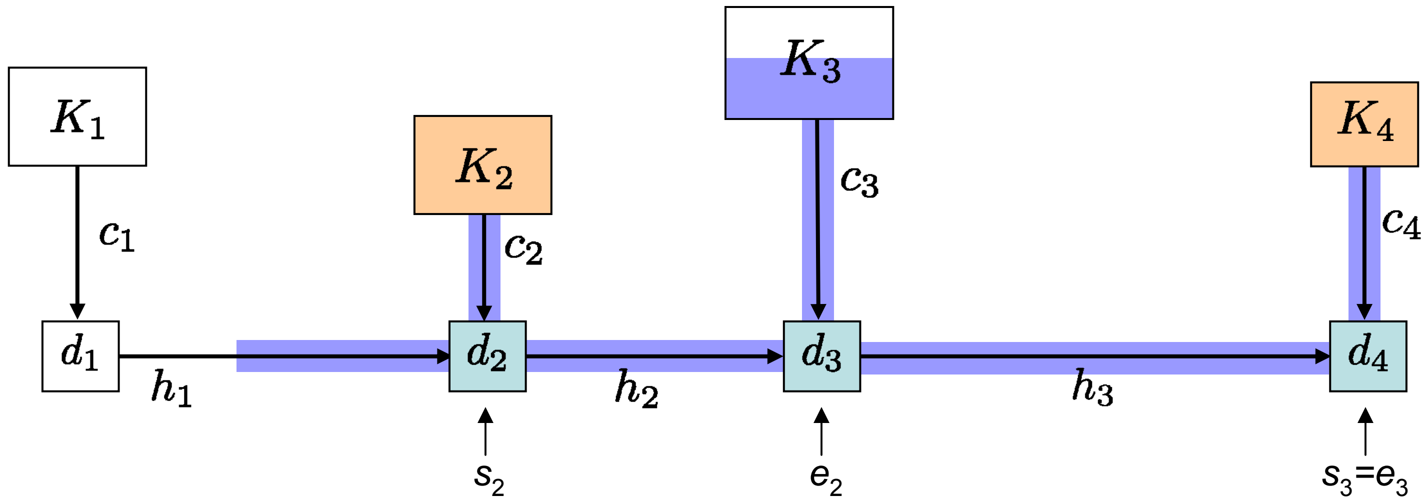
- Stack $D = (o_k, o_{k-1}, \ldots, o_1)$ of provisionally tight order points.
- Stack initially empty, at end of loop (2) will contain the correct order points.

Algorithm:

1) Start with D=(), t=n.
2) While $t \geq 1$:
   2.1) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods 1, …, t-1.
   2.2) If t becomes tight before o_k, delete o_k from stack D, and go to 2.1.
   2.3) Add t to stack D, *set* t:=t-1.

Add information to stack:

- $E = ((o_k, s_k, e_k), (o_{k-1}, s_{k-1}, e_{k-1}), (o_1, s_1, e_1)$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k become tight.



Stack data structure:

- Stack $D = (o_k, o_{k-1}, \ldots, o_1)$ of provisionally tight order points.
- Stack initially empty, at end of loop (2) will contain the correct order points.

Algorithm:

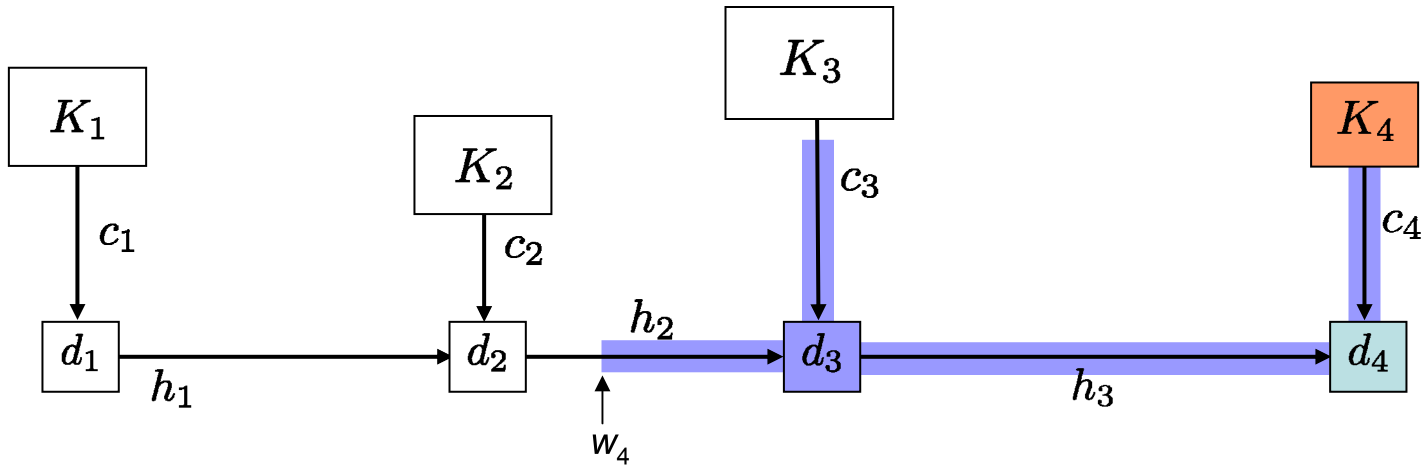
1) Start with D=(), t=n.
2) While $t \geq 1$:
   - 2.1) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t$-1.
   - 2.2) If t becomes tight before o_k, delete o_k from stack D, and go to 2.1.
   - 2.3) Add t to stack D, set t:=t-1.

Add information to stack:

- $E = ((o_k, s_k, e_k), (o_{k-1}, s_{k-1}, e_{k-1}), \ldots, (o_1, s_1, e_1))$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k became tight.

Add further information to stack:

- $E = ((o_k, s_k, e_k, w_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k), \ldots, (o_1,s_1,e_1,w_1))$ of provisionally tight order points.
- w_k is the wave position when o_k became tight.

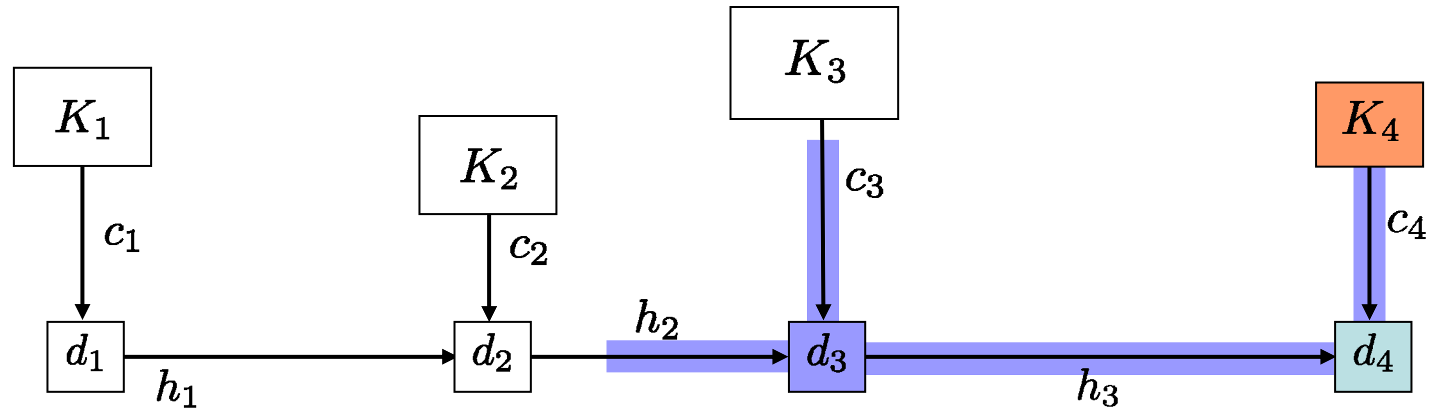


Stack data structure:
- Stack $D = (o_k, o_{k-1}, \ldots, o_1)$ of provisionally tight order points.
- Stack initially empty, at end of loop (2) will contain the correct order points.

Algorithm:
1) Start with D=(), t=n.
2) While $t \geq 1$:
   - 2.1) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t$-1.
   - 2.2) If t becomes tight before o_k, delete o_k from stack D, and go to 2.1.
   - 2.3) Add t to stack D, set t:=t-1.

Add information to stack:
- $E = ((o_k, s_k, e_k), (o_{k-1}, s_{k-1}, e_{k-1}), \ldots, (o_1, s_1, e_1))$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k became tight.

Add further information to stack:
- $E = ((o_k, s_k, e_k, w_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k), \ldots, (o_1, s_1, e_1, w_1))$ of provisionally tight order points.
- w_k is the wave position when o_k became tight.



Stack data structure:

- $E = ((o_k, s_k, e_k, w_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k), \ldots, (o_1,s_1,e_1,w_1))$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k became tight.
- w_k is the wave position when o_k became tight.

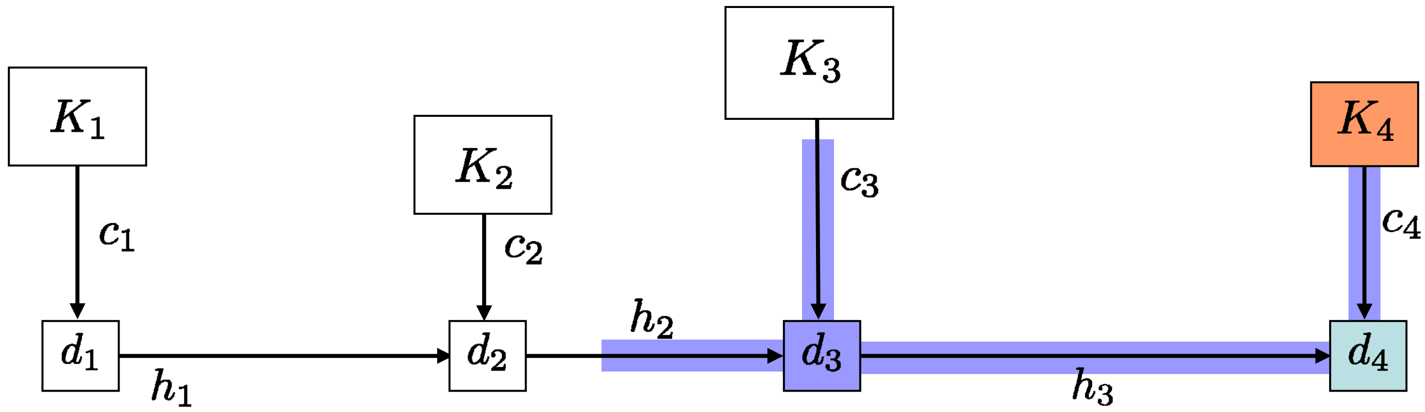
Algorithm:

1) Start with E=(), t=n.
2) While $t \geq 1$:
   - 2.1) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t$-1.
   - 2.2) If t becomes tight before o_k, delete o_k from stack E, and go to 2.1.
   - 2.3) Add t to stack E, set t:=t-1.

Algorithm details:

- Once the wave position when t becomes tight is computed, inserting the record into the stack takes $O(1)$.
- Deleting a record from the stack also takes $O(1)$.
- Every record is deleted at most once, and we make at most n insertions => at most $O(n)$ deletions / computations / insertions.

Remains to do computation, in $O(?)$



Stack data structure:

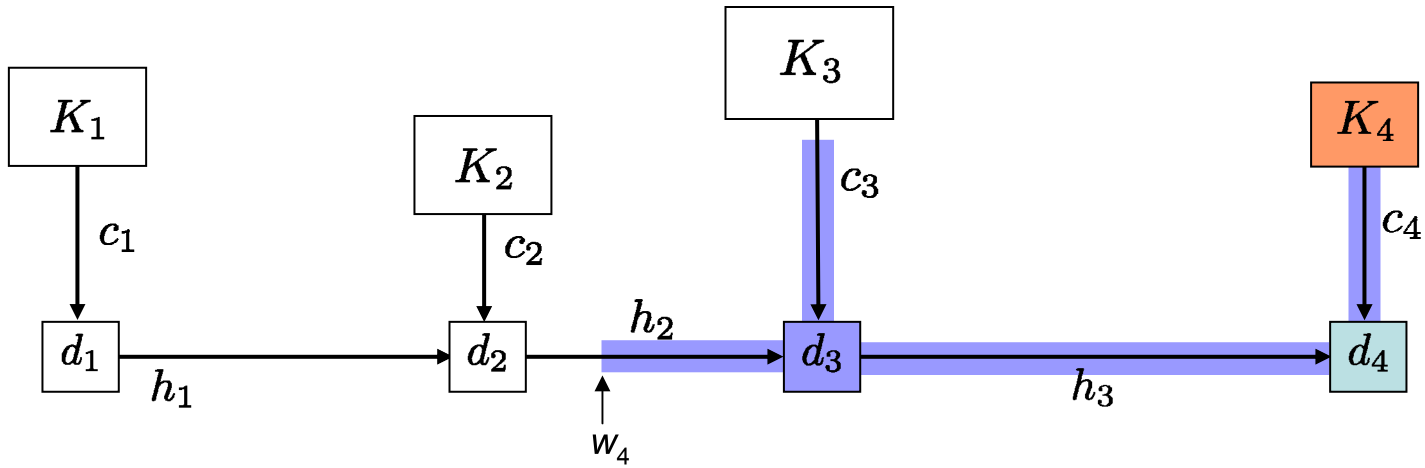
- $E = ((o_k, s_k, e_k, w_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k), \ldots, (o_1,s_1,e_1,w_1))$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k became tight.
- w_k is the wave position when o_k became tight.

Algorithm:

1) Start with E=(), t=n.
2) While $t \geq 1$:
   - 2.1) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t$-1.
   - 2.2) If t becomes tight before o_k, delete o_k from stack E, and go to 2.1.
   - 2.3) Add t to stack E, set t:=t-1.

Computation:

- Define new set of numbers $a_1, \ldots, a_n$ with $a_t = c_t + h_{tn}$.
- Only frozen demand points in segments $s_j,\ldots,e_j$ with $a_t \leq h_{1n} - w_j$ contribute to make order point t tight.
- All unfrozen demand points in $t, \ldots, o_k$ contribute.



Stack data structure:

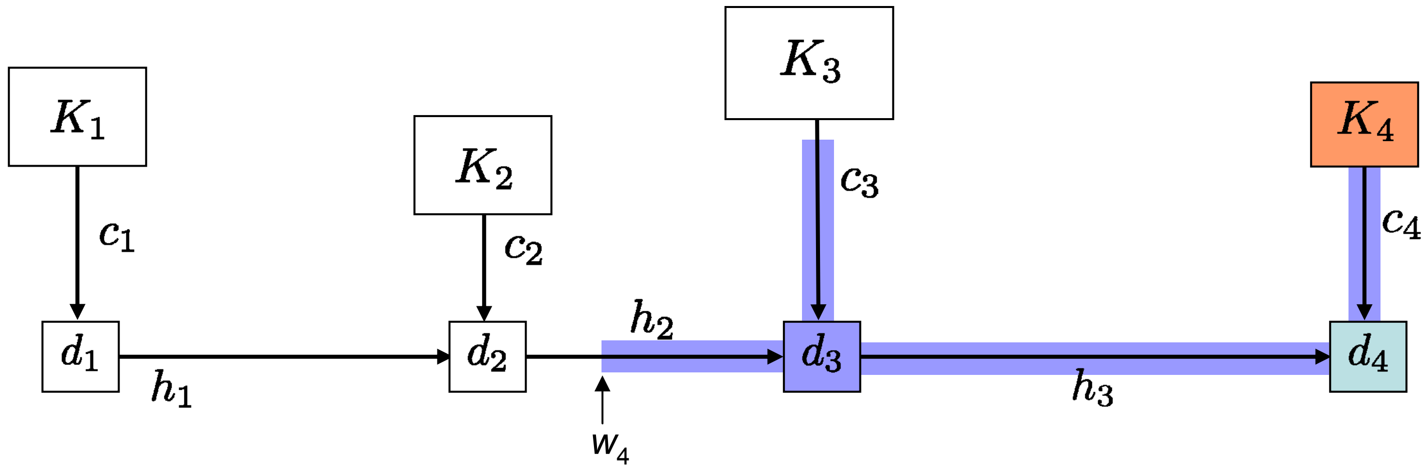
- $E = ((o_k, s_k, e_k, w_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k), \ldots, (o_1, s_1, e_1, w_1))$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k became tight.
- w_k is the wave position when o_k became tight.

Algorithm:

1) Start with $E=()$, $t=n$.
2) While $t \geq 1$:
   2.1) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t-1$.
   2.2) If t becomes tight before o_k, delete o_k from stack E, and go to 2.1.
   2.3) Add t to stack E, set $t := t-1$.

Computation:

- Define new set of numbers $a_1, \ldots, a_n$ with $a_t = c_t + h_{tn}$.
- Only frozen demand points in segments $s_j, \ldots, e_j$ with $a_t \leq h_{1n} - w_j$ contribute to make order point t tight.
- All unfrozen demand points in $t, \ldots, o_k$ contribute.



Stack data structure:

- $E = ((o_k, s_k, e_k, w_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k), \ldots, (o_1, s_1, e_1, w_1))$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k became tight.
- w_k is the wave position when o_k became tight.

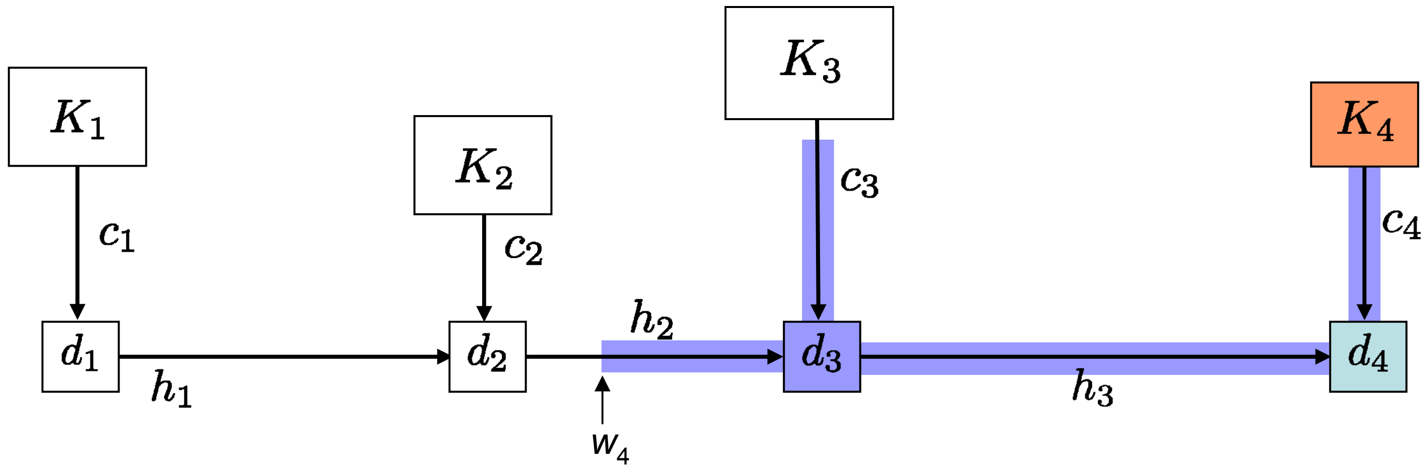
Algorithm:

1) Start with $E=()$, $t=n$.
2) While $t \geq 1$:
   2.1) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t-1$.
   2.2) If t becomes tight before o_k, delete o_k from stack E, and go to 2.1.
   2.3) Add t to stack E, set $t:=t-1$.

Computation:

- Define new set of numbers $a_1, \ldots, a_n$ with $a_t = c_t + h_{tn}$.
- Only frozen demand points in segments $s_j, \ldots, e_j$ with $a_t \leq h_{1n} - w_j$ contribute to make order point t tight.
- All unfrozen demand points in $t, \ldots, o_k$ contribute.

Key Lemma. Given $\min\{ j : a_t \leq h_{1n} - w_j \}$, we can determine in $O(1)$ if order point t becomes tight before o_k. If t becomes tight after o_k, we can compute in $O(1)$ the wave position when t becomes tight.



Stack data structure:

- $E = ((o_k, s_k, e_k, w_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k), \ldots, (o_1, s_1, e_1, w_1))$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k became tight.
- w_k is the wave position when o_k became tight.

Algorithm:

1) Start with $E=()$, $t=n$.
2) While $t \geq 1$:
   2.1) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t-1$.
   2.2) If t becomes tight before o_k, delete o_k from stack E, and go to 2.1.
   2.3) Add t to stack E, set $t := t-1$.

Key Lemma. Given $j^* = \min\{ j : a_t \leq h_{1n} - w_j \}$, we Can determine in $O(1)$ if order point t becomes tight before o_k. If t becomes tight after o_k, we can compute in $O(1)$ the wave position when t becomes tight.

Proof Idea. Can determine and perform the computation by inspecting each demand point in $t, \ldots, o_k$, and each record in $k, \ldots, j^*$.

This would take $O(n)$.

Can perform in O(1), by computing the running sums $d_1, d_1+d_2, \ldots, d_1+d_2+\ldots+d_n$ at start of algorithm, as well as certain running sums in the stack.

Stack becomes $E = ((o_k, s_k, e_k, w_k, R_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k, R_{k-1}), \ldots, (o_1, s_1, e_1, w_1, R_1))$.

Time for Data Structures

Stack data structure:

- $E = ((o_k, s_k, e_k, w_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k), \ldots, (o_1, s_1, e_1, w_1))$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k became tight.
- w_k is the wave position when o_k became tight.

Algorithm:

1) Start with E=(), t=n.
2) While $t \geq 1$:
   2.1) Compute $j^* = \min\{ j : a_t \leq h_{1n} - w_j \}$.
   2.2) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t$-1.
   2.2) If t becomes tight before o_k, delete o_k from stack E, and go to 2.1.
   2.3) Add t to stack E, set t:=t-1.

Key Lemma. Given $j^* = \min\{ j : a_t \leq h_{1n} - w_j \}$, we Can determine in $O(1)$ if order point t becomes tight before o_k. If t becomes tight after o_k, we can compute in $O(1)$ the wave position when t becomes tight.

Running time.
$O(n)$ iterations.
Find j^* using binary search in $O(\log(n))$
Do the remaining computations in $O(1)$
Total: $O(n \log n)$

Running time when non-speculative.
$O(n)$ iterations.
Find j^* in $O(1)$, since $a_1, \ldots, a_n$ are monotonic.
Do the remaining computations in $O(1)$
Total: $O(n)$

Matches best times so far

Time for Data Structures

Stack data structure:

- $E = ((o_k, s_k, e_k, w_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k), \ldots, (o_1, s_1, e_1, w_1))$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k became tight.
- w_k is the wave position when o_k became tight.

Algorithm:

1) Start with E=(), t=n.
2) While $t \geq 1$:
   - 2.1) Compute $j^* = \min\{ j : a_t \leq h_{1n} - w_j \}$.
   - 2.2) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t$-1.
   - 2.2) If t becomes tight before o_k, delete o_k from stack E, and go to 2.1.
   - 2.3) Add t to stack E, set t:=t-1.

Key Lemma. Given $j^* = \min\{ j : a_t \leq h_{1n} - w_j \}$, we Can determine in $O(1)$ if order point t becomes tight before o_k. If t becomes tight after o_k, we can compute in $O(1)$ the wave position when t becomes tight.

Improved running time.
Machine: Word RAM

1) Sort $a_1, \ldots, a_n$ in $O(n \log \log n)$ before the start of the algorithm.
2) Create additional stack E' that contains only $n/\log(n)$ entries out of the stack E. It divides E into buckets of size $\log(n)$.
3) Whenever a record is inserted into E', look up the position of w_k in $a_1, \ldots, a_k$ and place it in the record. $E' = ((o_k, s_k, e_k, w_k, p_k))$.
4) When we have to find out j^*, first look up in E' using predecessor search, then look up in bucket in E using binary search.

Time for Data Structures

Stack data structure:

- $E = ((o_k, s_k, e_k, w_k), (o_{k-1}, s_{k-1}, e_{k-1}, w_k), \ldots, (o_1,s_1,e_1,w_1))$ of provisionally tight order points.
- $s_k, \ldots, e_k$ are the demand points that were frozen when o_k became tight.
- w_k is the wave position when o_k became tight.

Algorithm:

1) Start with E=(), t=n.
2) While $t \geq 1$:
   2.1) Compute $j^* = \min\{ j : a_t \leq h_{1n} - w_j \}$.
   2.2) Compute time when order point t becomes tight, taking into account periods $t, \ldots, n$ but ignoring periods $1, \ldots, t$-1.
   2.2) If t becomes tight before o_k, delete o_k from stack E, and go to 2.1.
   2.3) Add t to stack E, set t:=t-1.

Total: O(n log log n)

Improved running time.
Machine: Word RAM

1) Sort $a_1, \ldots, a_n$ in $O(n \log \log n)$ before the start of the algorithm.
2) Create additional stack E' that contains only $n/\log(n)$ entries out of the stack E. It divides E into buckets of size $\log(n)$.
3) Whenever a record is inserted into E', look up the position of w_k in $a_1, \ldots, a_k$ and place it in the record. $E' = ((o_k, s_k, e_k, w_k, p_k))$.
4) When we have to find out j^*, first look up in E' using predecessor search, then look up in bucket in E using binary search.

Running time:

1) O(n log log n).
3) Each addition takes O(log(n)), there are O(n/log(n)) additions, for a total of O(n). [*]
4) Predecessor search takes O(log log n), O(n) lookups for a total of O(n log log n).
4.1) Bucket size is O(log n), lookup takes O(log log n), total O(n log log n)

Conclusions

1. We obtain a $O(n \log \log n)$ algorithm for lot-sizing, improving upon the $O(n \log n)$ results from 1991–1993.

2. We connect the lot-sizing problem to basic computing primitives—sorting and predecessor search.
   a. Opportunity for further improvement in running time.
   b. Opportunity for other insights.

3. Wave primal-dual algorithms run on other inventory problems (JRP, multi-stage assembly).

4. $O(n+\mathrm{sort}(n))$? $O(n)$?