Algorithms for Stochastic Lot-Sizing Problems with Backlogging

Yongpei Guan

School of Industrial Engineering
University of Oklahoma

MIP 2008
Columbia University, New York City

This research is partially supported by NSF CMMI-0700868

Outline

1. Introduction
   - Motivation
   - Related Work
2. Our Results/Contribution
   - Stochastic Uncapacitated Lot-Sizing (SULS) with Backlogging
   - Stochastic Capacitated Lot-Sizing (SCLS) with Backlogging
   - Stochastic Constant Capacitated Lot-Sizing (SCCLS) with Backlogging
3. Current Research
4. Summary and Future Research

Outline

1. Introduction
   - Motivation
   - Related Work
2. Our Results/Contribution
   - Stochastic Uncapacitated Lot-Sizing (SULS) with Backlogging
   - Stochastic Capacitated Lot-Sizing (SCLS) with Backlogging
   - Stochastic Constant Capacitated Lot-Sizing (SCCLS) with Backlogging
3. Current Research
4. Summary and Future Research

Deterministic Inventory Planning Problems

- EOQ model
  - Demand is a constant number for each time period
  - Tradeoff between set up cost and inventory holding cost:

$$\min f(Q) = K\frac{D}{Q} + h\frac{Q}{2}.$$

- Economic lot-sizing model
  - Demand is deterministic and can be variant from period to period
  - Tradeoff between set up, production, and inventory holding costs.

Deterministic Inventory Planning Problems

- EOQ model
  - Demand is a constant number for each time period
  - Tradeoff between set up cost and inventory holding cost:

$$\min f(Q) = K\frac{D}{Q} + h\frac{Q}{2}.$$

- Economic lot-sizing model
  - Demand is deterministic and can be variant from period to period
  - Tradeoff between set up, production, and inventory holding costs.

Deterministic Uncapacitated Lot-Sizing Problem

Problem Description:

Decide when and how much to produce at each time period over a finite discrete horizon so as to satisfy demands while minimizing the total cost.

$$
\begin{aligned}
(\mathrm{LS}): \quad \min \quad & \sum_{i=0}^{T} (\alpha_i x_i + \beta_i y_i + h_i s_i) \\
\text{s.t.} \quad & s_{i-1} + x_i = d_i + s_i && i = 0, \dots, T, \\
& x_i \leq M y_i && i = 0, \dots, T, \\
& x_i, s_i \geq 0, \; y_i \in \{0, 1\} && i = 0, \dots, T,
\end{aligned}
$$

where x: production quantity; s: inventory; y: set-up indicator.

Algorithms for LS and Its Variants

- Uncapaciated lot-sizing problem (Wagner and Whitin 1958)
- Improved algorithm (Aggarwal and Park 1993, Federgruen and Tzur 1991, and Wagelmans et al. 1992)
- Uncapacitated problem with backlogging (Federgruen and Tzur 1993)
- Constant capacitated lot-sizing (Florian and Klein 1971, van Hoesel and Wagelmans 1996)
- Uncapacitated problem with demand time windows (Lee et al. 2001)
- Uncapacitated problem with inventory bounds (Atamtürk and Kücükyavuz 2007, Liu 2007)

Algorithms for LS and Its Variants

- Uncapaciated lot-sizing problem (Wagner and Whitin 1958)
- Improved algorithm (Aggarwal and Park 1993, Federgruen and Tzur 1991, and Wagelmans et al. 1992)
- Uncapacitated problem with backlogging (Federgruen and Tzur 1993)
- Constant capacitated lot-sizing (Florian and Klein 1971, van Hoesel and Wagelmans 1996)
- Uncapacitated problem with demand time windows (Lee et al. 2001)
- Uncapacitated problem with inventory bounds (Atamtürk and Kücükyavuz 2007, Liu 2007)

Algorithms for LS and Its Variants

- Uncapaciated lot-sizing problem (Wagner and Whitin 1958)
- Improved algorithm (Aggarwal and Park 1993, Federgruen and Tzur 1991, and Wagelmans et al. 1992)
- Uncapacitated problem with backlogging (Federgruen and Tzur 1993)
- Constant capacitated lot-sizing (Florian and Klein 1971, van Hoesel and Wagelmans 1996)
- Uncapacitated problem with demand time windows (Lee et al. 2001)
- Uncapacitated problem with inventory bounds (Atamtürk and Kücükyavuz 2007, Liu 2007)

Algorithms for LS and Its Variants

- Uncapaciated lot-sizing problem (Wagner and Whitin 1958)
- Improved algorithm (Aggarwal and Park 1993, Federgruen and Tzur 1991, and Wagelmans et al. 1992)
- Uncapacitated problem with backlogging (Federgruen and Tzur 1993)
- Constant capacitated lot-sizing (Florian and Klein 1971, van Hoesel and Wagelmans 1996)
- Uncapacitated problem with demand time windows (Lee et al. 2001)
- Uncapacitated problem with inventory bounds (Atamtürk and Kücükyavuz 2007, Liu 2007)

Algorithms for LS and Its Variants

- Uncapaciated lot-sizing problem (Wagner and Whitin 1958)
- Improved algorithm (Aggarwal and Park 1993, Federgruen and Tzur 1991, and Wagelmans et al. 1992)
- Uncapacitated problem with backlogging (Federgruen and Tzur 1993)
- Constant capacitated lot-sizing (Florian and Klein 1971, van Hoesel and Wagelmans 1996)
- Uncapacitated problem with demand time windows (Lee et al. 2001)
- Uncapacitated problem with inventory bounds (Atamtürk and Kücükyavuz 2007, Liu 2007)

Algorithms for LS and Its Variants

- Uncapaciated lot-sizing problem (Wagner and Whitin 1958)
- Improved algorithm (Aggarwal and Park 1993, Federgruen and Tzur 1991, and Wagelmans et al. 1992)
- Uncapacitated problem with backlogging (Federgruen and Tzur 1993)
- Constant capacitated lot-sizing (Florian and Klein 1971, van Hoesel and Wagelmans 1996)
- Uncapacitated problem with demand time windows (Lee et al. 2001)
- Uncapacitated problem with inventory bounds (Atamtürk and Kücükyavuz 2007, Liu 2007)

Stochastic Inventory Control Problem

- News vendor model

$$\min z(y) = cy - r\int_D \min(y, D)dF(D) - v\int_{D=0}^{y}(y-D)dF(D)\}.$$

- Base stock policies
- (Q, r) policies
- (s, S) policies (Scarf 1960, Zheng and Federgruen 1991)

$$Z_k(y_k) = \min_{y \geq y_k}\{K\delta(y-y_k) + G_k(y)\} - cy_k,$$

$$G_k(y) = c_k y + G(y) + \int_D Z_{k+1}(y-D)dF(D), \text{ and}$$

$$G(y) = h\int_D \max(y-D,0)dF(D) + b\int_D \max(D-y,0)dF(D),$$

where $\delta(y)$ is the indictor to show if y is positive and $G(y)$ is defined as *loss function*.

Stochastic Inventory Control Problem

- News vendor model

$$\min z(y) = cy - r \int_D \min(y, D) dF(D) - v \int_{D=0}^{y} (y - D) dF(D)\}.$$

- Base stock policies
- (Q, r) policies
- (s, S) policies (Scarf 1960, Zheng and Federgruen 1991)

$$Z_k(y_k) = \min_{y \geq y_k} \{K\delta(y - y_k) + G_k(y)\} - cy_k,$$

$$G_k(y) = c_k y + G(y) + \int_D Z_{k+1}(y - D) dF(D), \text{ and}$$

$$G(y) = h \int_D \max(y - D, 0) dF(D) + b \int_D \max(D - y, 0) dF(D),$$

where $\delta(y)$ is the indictor to show if y is positive and $G(y)$ is defined as *loss function*.

Stochastic Inventory Control Problem

- News vendor model

$$\min z(y) = cy - r\int_D \min(y, D)dF(D) - v\int_{D=0}^{y}(y - D)dF(D)\}.$$

- Base stock policies
- (Q, r) policies
- (s, S) policies (Scarf 1960, Zheng and Federgruen 1991)

$$Z_k(y_k) = \min_{y \geq y_k}\{K\delta(y - y_k) + G_k(y)\} - cy_k,$$

$$G_k(y) = c_k y + G(y) + \int_D Z_{k+1}(y - D)dF(D), \text{ and}$$

$$G(y) = h\int_D \max(y - D, 0)dF(D) + b\int_D \max(D - y, 0)dF(D),$$

where $\delta(y)$ is the indictor to show if y is positive and $G(y)$ is defined as *loss function*.

Stochastic Inventory Control Problem

- News vendor model

$$\min z(y) = cy - r\int_D \min(y, D)dF(D) - v\int_{D=0}^{y}(y-D)dF(D)\}.$$

- Base stock policies
- (Q, r) policies
- (s, S) policies (Scarf 1960, Zheng and Federgruen 1991)

$$Z_k(y_k) = \min_{y \geq y_k}\{K\delta(y-y_k) + G_k(y)\} - cy_k,$$

$$G_k(y) = c_k y + G(y) + \int_D Z_{k+1}(y-D)dF(D), \text{ and}$$

$$G(y) = h\int_D \max(y-D, 0)dF(D) + b\int_D \max(D-y, 0)dF(D),$$

where $\delta(y)$ is the indictor to show if y is positive and $G(y)$ is defined as *loss function*.

Stochastic Integer Programming

- Two-Stage: Significant Progress
  - Benders Decomposition/L-Shaped method (Benders 1962, Van Slyke and Wets 1969)
  - Decomposition methods (Care and Tind 1998, Carøe and Schultz 1999, Laporte and Louveaux 1993, and Ahmed et al 2004)
  - Combination of disjunctive inequalities with decomposition methods (Ntaimo and Sen 2005, Sen and Sherali 2006)
- Multi-Stage: Study Is Limited
  - Approximation scheme for stochastic integer programs arising in capacity expansion (Ahmed and Sahinidis 2003)
  - A branch-and-price algorithm for multi-stage stochastic integer program (Lulli and Sen 2004)
  - Cutting planes for multi-stage stochastic integer programs (Guan, Ahmed and Nemhauser 2008)

Stochastic Integer Programming

- Two-Stage: Significant Progress
  - Benders Decomposition/L-Shaped method (Benders 1962, Van Slyke and Wets 1969)
  - Decomposition methods (Care and Tind 1998, Carøe and Schultz 1999, Laporte and Louveaux 1993, and Ahmed et al 2004)
  - Combination of disjunctive inequalities with decomposition methods (Ntaimo and Sen 2005, Sen and Sherali 2006)
- Multi-Stage: Study Is Limited
  - Approximation scheme for stochastic integer programs arising in capacity expansion (Ahmed and Sahinidis 2003)
  - A branch-and-price algorithm for multi-stage stochastic integer program (Lulli and Sen 2004)
  - Cutting planes for multi-stage stochastic integer programs (Guan, Ahmed and Nemhauser 2008)

Stochastic Integer Programming

- Two-Stage: Significant Progress
  - Benders Decomposition/L-Shaped method (Benders 1962, Van Slyke and Wets 1969)
  - Decomposition methods (Care and Tind 1998, Carøe and Schultz 1999, Laporte and Louveaux 1993, and Ahmed et al 2004)
  - Combination of disjunctive inequalities with decomposition methods (Ntaimo and Sen 2005, Sen and Sherali 2006)
- Multi-Stage: Study Is Limited
  - Approximation scheme for stochastic integer programs arising in capacity expansion (Ahmed and Sahinidis 2003)
  - A branch-and-price algorithm for multi-stage stochastic integer program (Lulli and Sen 2004)
  - Cutting planes for multi-stage stochastic integer programs (Guan, Ahmed and Nemhauser 2008)

Stochastic Integer Programming

- Two-Stage: Significant Progress
  - Benders Decomposition/L-Shaped method (Benders 1962, Van Slyke and Wets 1969)
  - Decomposition methods (Care and Tind 1998, Carøe and Schultz 1999, Laporte and Louveaux 1993, and Ahmed et al 2004)
  - Combination of disjunctive inequalities with decomposition methods (Ntaimo and Sen 2005, Sen and Sherali 2006)
- Multi-Stage: Study Is Limited
  - Approximation scheme for stochastic integer programs arising in capacity expansion (Ahmed and Sahinidis 2003)
  - A branch-and-price algorithm for multi-stage stochastic integer program (Lulli and Sen 2004)
  - Cutting planes for multi-stage stochastic integer programs (Guan, Ahmed and Nemhauser 2008)

Stochastic Integer Programming

- Two-Stage: Significant Progress
  - Benders Decomposition/L-Shaped method (Benders 1962, Van Slyke and Wets 1969)
  - Decomposition methods (Care and Tind 1998, Carøe and Schultz 1999, Laporte and Louveaux 1993, and Ahmed et al 2004)
  - Combination of disjunctive inequalities with decomposition methods (Ntaimo and Sen 2005, Sen and Sherali 2006)
- Multi-Stage: Study Is Limited
  - Approximation scheme for stochastic integer programs arising in capacity expansion (Ahmed and Sahinidis 2003)
  - A branch-and-price algorithm for multi-stage stochastic integer program (Lulli and Sen 2004)
  - Cutting planes for multi-stage stochastic integer programs (Guan, Ahmed and Nemhauser 2008)

Stochastic Integer Programming

- Two-Stage: Significant Progress
  - Benders Decomposition/L-Shaped method (Benders 1962, Van Slyke and Wets 1969)
  - Decomposition methods (Care and Tind 1998, Carøe and Schultz 1999, Laporte and Louveaux 1993, and Ahmed et al 2004)
  - Combination of disjunctive inequalities with decomposition methods (Ntaimo and Sen 2005, Sen and Sherali 2006)
- Multi-Stage: Study Is Limited
  - Approximation scheme for stochastic integer programs arising in capacity expansion (Ahmed and Sahinidis 2003)
  - A branch-and-price algorithm for multi-stage stochastic integer program (Lulli and Sen 2004)
  - Cutting planes for multi-stage stochastic integer programs (Guan, Ahmed and Nemhauser 2008)

Stochastic Integer Programming

- Two-Stage: Significant Progress
  - Benders Decomposition/L-Shaped method (Benders 1962, Van Slyke and Wets 1969)
  - Decomposition methods (Care and Tind 1998, Carøe and Schultz 1999, Laporte and Louveaux 1993, and Ahmed et al 2004)
  - Combination of disjunctive inequalities with decomposition methods (Ntaimo and Sen 2005, Sen and Sherali 2006)
- Multi-Stage: Study Is Limited
  - Approximation scheme for stochastic integer programs arising in capacity expansion (Ahmed and Sahinidis 2003)
  - A branch-and-price algorithm for multi-stage stochastic integer program (Lulli and Sen 2004)
  - Cutting planes for multi-stage stochastic integer programs (Guan, Ahmed and Nemhauser 2008)

What Are We Going To Address?

- How about the case that demands are mutually dependent for stochastic inventory control problem? (Sampling approach?)
- How to apply stochastic integer programming to formulate general production/inventory planning under uncertainty problems?
- What is the computational complexity for the problem in terms of input size?

What Are We Going To Address?

- How about the case that demands are mutually dependent for stochastic inventory control problem? (Sampling approach?)
- How to apply stochastic integer programming to formulate general production/inventory planning under uncertainty problems?
- What is the computational complexity for the problem in terms of input size?

What Are We Going To Address?

- How about the case that demands are mutually dependent for stochastic inventory control problem? (Sampling approach?)
- How to apply stochastic integer programming to formulate general production/inventory planning under uncertainty problems?
- What is the computational complexity for the problem in terms of input size?

Notation

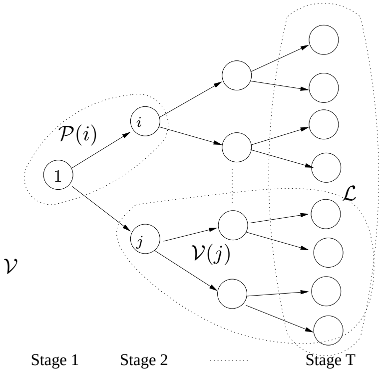


Figure: Multi-stage stochastic scenario tree

Formulation

General Stochastic Capacitated Lot-Sizing with Backlogging:

$$
\begin{aligned}
(\mathrm{SCLS}): \quad \min \quad & \sum_{i \in \mathcal{V}} (\alpha_i x_i + \beta_i y_i + h_i s_i^+ + b_i s_i^-) \\
\text{s.t.} \quad & s_{i^-}^+ + s_i^- + x_i = d_i + s_i^+ + s_{i^-}^- && \forall\, i \in \mathcal{V}, \\
& x_i \leq \mu_i y_i && \forall\, i \in \mathcal{V}, \\
& x_i, s_i^+, s_i^- \geq 0, \;\; y_i \in \{0,1\} && \forall\, i \in \mathcal{V}.
\end{aligned}
$$

x: Production; s^+: Inventory; s^-: Backorder; y: Set-up Indicator.

α: Production cost; β: Setup cost; h: Holding cost;

b: Backorder cost; d: Demands; μ: Capacity.

Outline

1. Introduction
   - Motivation
   - Related Work
2. Our Results/Contribution
   - Stochastic Uncapacitated Lot-Sizing (SULS) with Backlogging
   - Stochastic Capacitated Lot-Sizing (SCLS) with Backlogging
   - Stochastic Constant Capacitated Lot-Sizing (SCCLS) with Backlogging
3. Current Research
4. Summary and Future Research

Approximation and Polynomial Time Algorithms

- A fully polynomial time approximation scheme for SULS (Halman, Klabjan, Mostagir, Orlin and Simchi-Levi 2006)
- SULS without setup cost (Huang and Ahmed 2006)
- SULS with and without setup cost (Guan and Miller 2007)
- SULS with random lead time (Huang and Kücükyavuz 2007)

Approximation and Polynomial Time Algorithms

- A fully polynomial time approximation scheme for SULS (Halman, Klabjan, Mostagir, Orlin and Simchi-Levi 2006)
- SULS without setup cost (Huang and Ahmed 2006)
- SULS with and without setup cost (Guan and Miller 2007)
- SULS with random lead time (Huang and Kücükyavuz 2007)

Approximation and Polynomial Time Algorithms

- A fully polynomial time approximation scheme for SULS (Halman, Klabjan, Mostagir, Orlin and Simchi-Levi 2006)
- SULS without setup cost (Huang and Ahmed 2006)
- SULS with and without setup cost (Guan and Miller 2007)
- SULS with random lead time (Huang and Kücükyavuz 2007)

Approximation and Polynomial Time Algorithms

- A fully polynomial time approximation scheme for SULS (Halman, Klabjan, Mostagir, Orlin and Simchi-Levi 2006)
- SULS without setup cost (Huang and Ahmed 2006)
- SULS with and without setup cost (Guan and Miller 2007)
- SULS with random lead time (Huang and Kücükyavuz 2007)

Outline

1 Introduction
- Motivation
- Related Work

2 Our Results/Contribution
- Stochastic Uncapacitated Lot-Sizing (SULS) with Backlogging
- Stochastic Capacitated Lot-Sizing (SCLS) with Backlogging
- Stochastic Constant Capacitated Lot-Sizing (SCCLS) with Backlogging

3 Current Research

4 Summary and Future Research

The Production Path Property

Production Path Property

For any instance of SULS with backlogging, there exists an optimal solution (x^*, y^*, s^*) such that for each node $i \in \mathcal{V}$,

$$\text{if } x_i^* > 0, \text{ then } x_i^* = d_{ik} - s_{i^-}^* \text{ for some } k \in \mathcal{V}(i).$$

In other words, there always exists an optimal solution such that if we produce at a node i, then we produce exactly enough to satisfy demand along the path from node i to some descendant of node i.

The Production Path Property

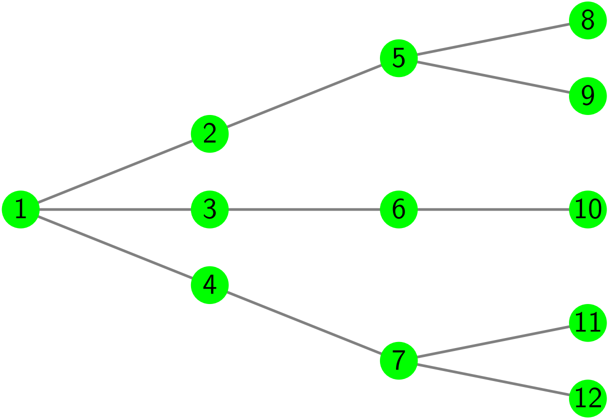
Production Path Property Corollary

For any instance of SULS with backlogging, there exists an optimal solution (x^*, y^*, s^*) such that the inventory left after node $i \in \mathcal{V}$

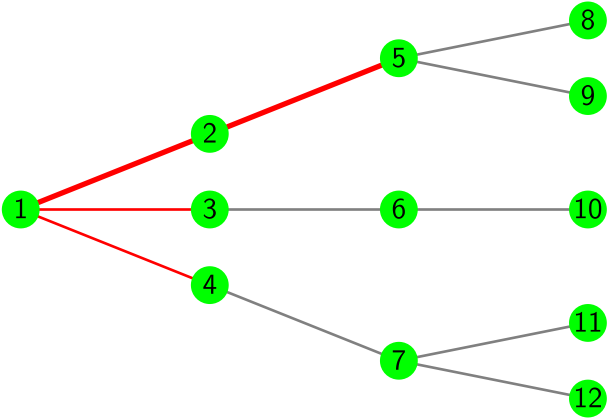
$$s_i^* = d_{1k} - d_{1i} \text{ for some node } k \in \mathcal{V}.$$

Thus, there are finite number of possible values for s_i^*.

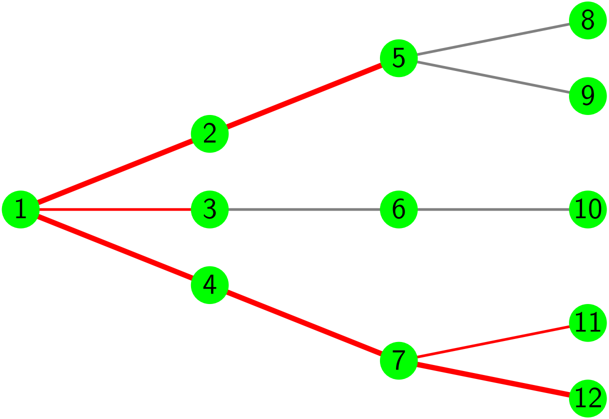
The Production Path Property



The Production Path Property



The Production Path Property



Dynamic Programming Recursion

- Let $\mathcal{H}(i, s)$ be the "cost to go" at node i, given that s units of inventory are carried into i from i's parent i^-
- We can use the Production Path Property and backward recursion to construct this value function at each node
- Thus $\mathcal{H}(1, 0)$ (the value function of the root node evaluated at 0) will yield the optimal objective function value
- Take advantage of the scenario tree structure to speed up the algorithm

Dynamic Programming Recursion

- Let $\mathcal{H}(i, s)$ be the "cost to go" at node i, given that s units of inventory are carried into i from i's parent i^-
- We can use the Production Path Property and backward recursion to construct this value function at each node
- Thus $\mathcal{H}(1, 0)$ (the value function of the root node evaluated at 0) will yield the optimal objective function value
- Take advantage of the scenario tree structure to speed up the algorithm

Dynamic Programming Recursion

- Let $\mathcal{H}(i, s)$ be the "cost to go" at node i, given that s units of inventory are carried into i from i's parent i^-
- We can use the Production Path Property and backward recursion to construct this value function at each node
- Thus $\mathcal{H}(1, 0)$ (the value function of the root node evaluated at 0) will yield the optimal objective function value
- Take advantage of the scenario tree structure to speed up the algorithm

Dynamic Programming Recursion

- Let $\mathcal{H}(i, s)$ be the "cost to go" at node i, given that s units of inventory are carried into i from i's parent i^-
- We can use the Production Path Property and backward recursion to construct this value function at each node
- Thus $\mathcal{H}(1, 0)$ (the value function of the root node evaluated at 0) will yield the optimal objective function value
- Take advantage of the scenario tree structure to speed up the algorithm

Value Functions

- Production Value Function:

$$\mathcal{H}_{\mathrm{P}}(i,s) = \beta_i + \min_{k \in \mathcal{V}(i): d_{ik} > s} \{\alpha_i(d_{ik} - s) + h_i(d_{ik} - d_i) + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, d_{ik} - d_i)\}.$$

- Non-Production Value Function:

$$\mathcal{H}_{\mathrm{NP}}(i,s) = \max\{h_i(s - d_i), -b_i(s - d_i)\} + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s - d_i).$$

- Value Function:

$$\mathcal{H}(i,s) = \min\{\mathcal{H}_{\mathrm{P}}(i,s), \mathcal{H}_{\mathrm{NP}}(i,s)\}.$$

- We can find an algorithm that runs in $\mathcal{O}(\mathcal{C}n^3)$ time, where $\mathcal{C}$ is the maximum number of children and n is the total number of nodes in the tree.

Value Functions

- Production Value Function:

$$\mathcal{H}_{\mathsf{P}}(i,s) = \beta_i + \min_{k \in \mathcal{V}(i): d_{ik} > s} \{\alpha_i (d_{ik} - s) + h_i (d_{ik} - d_i) + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, d_{ik} - d_i)\}.$$

- Non-Production Value Function:

$$\mathcal{H}_{\mathsf{NP}}(i,s) = \max\{h_i(s - d_i), -b_i(s - d_i)\} + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s - d_i).$$

- Value Function:

$$\mathcal{H}(i,s) = \min\left\{\mathcal{H}_{\mathsf{P}}(i,s), \mathcal{H}_{\mathsf{NP}}(i,s)\right\}.$$

- We can find an algorithm that runs in $\mathcal{O}(\mathcal{C}n^3)$ time, where $\mathcal{C}$ is the maximum number of children and n is the total number of nodes in the tree.

Value Functions

- Production Value Function:

$$\mathcal{H}_{\mathrm{P}}(i,s) = \beta_i + \min_{k\in\mathcal{V}(i):d_{ik}>s}\{\alpha_i(d_{ik}-s)+h_i(d_{ik}-d_i)+\sum_{\ell\in\mathcal{C}(i)}\mathcal{H}(\ell,d_{ik}-d_i)\}.$$

- Non-Production Value Function:

$$\mathcal{H}_{\mathrm{NP}}(i,s) = \max\{h_i(s-d_i), -b_i(s-d_i)\} + \sum_{\ell\in\mathcal{C}(i)}\mathcal{H}(\ell, s-d_i).$$

- Value Function:

$$\mathcal{H}(i,s) = \min\left\{\mathcal{H}_{\mathrm{P}}(i,s), \mathcal{H}_{\mathrm{NP}}(i,s)\right\}.$$

- We can find an algorithm that runs in $\mathcal{O}(\mathcal{C}n^3)$ time, where $\mathcal{C}$ is the maximum number of children and n is the total number of nodes in the tree.

Value Functions

- Production Value Function:

$$\mathcal{H}_{\text{P}}(i,s) = \beta_i + \min_{k \in \mathcal{V}(i): d_{ik} > s} \{\alpha_i(d_{ik} - s) + h_i(d_{ik} - d_i) + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, d_{ik} - d_i)\}.$$

- Non-Production Value Function:

$$\mathcal{H}_{\text{NP}}(i,s) = \max\{h_i(s - d_i), -b_i(s - d_i)\} + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s - d_i).$$

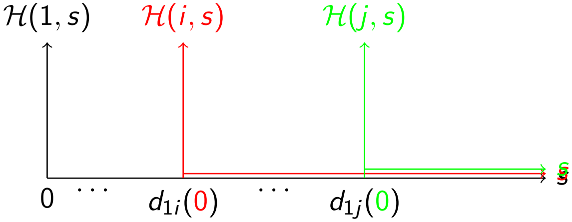
- Value Function:

$$\mathcal{H}(i,s) = \min\{\mathcal{H}_{\text{P}}(i,s), \mathcal{H}_{\text{NP}}(i,s)\}.$$

- We can find an algorithm that runs in $\mathcal{O}(\mathcal{C}n^3)$ time, where $\mathcal{C}$ is the maximum number of children and n is the total number of nodes in the tree.

Can We Do Better? The Algorithm

It is sufficient to calculate and store $\mathcal{H}(i,s)$ for $s = d_{1k} - d_{1i^-}$ for all $k \in \mathcal{V}$. Due to the relationship between nodes in the tree as shown in the following,



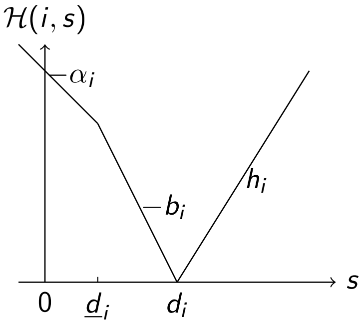
there are n inventory values to be stored for each node.

The Algorithm (Initial Step)

1. set relationship indicator $\delta(i,k)$ between each node $i \in \mathcal{V}$ and each node $k \in \mathcal{V}(i)$. For instance, $\delta(i,k) = 1$ if node $k \in \mathcal{V}(i)$ and $\delta(i,k) = 0$, otherwise;
2. use $\theta(i,s)$ to store $\sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s)$ for which $s = d_{1k} - d_{1i}$ for all $k \in \mathcal{V}$ and initialize them to zero.

This initial step can be completed in $\mathcal{O}(n^2)$ time.

The Algorithm (The Leaf Node)



1. $\mathcal{H}(i,s)$ where $s = d_{1k} - d_{1i^-}$ for each node $k \in \mathcal{V}$ can be calculated and stored in $\mathcal{O}(n)$ time.
2. Increase $\theta(i^-, d_{1k} - d_{1i^-})$ by $\mathcal{H}(i, d_{1k} - d_{1i^-})$ for each node $k \in \mathcal{V}$. This step takes $\mathcal{O}(n)$ time.

The Algorithm (The Induction Step)

Let

$$\phi(i,k) = \beta_i + \alpha_i d_{ik} + h_i(d_{ik} - d_i) + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, d_{ik} - d_i).$$

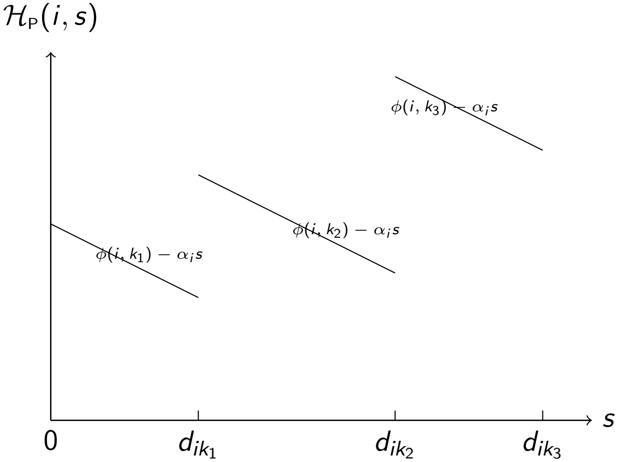
Assuming there are r linear pieces, each piece of the production value function can be described as follows:

$$\begin{aligned}
\mathcal{H}_{\mathrm{P}}(i,s) &= \phi(i,k_1) - \alpha_i s \text{ if } s \leq d_{ik_1}, \\
\mathcal{H}_{\mathrm{P}}(i,s) &= \phi(i,k_2) - \alpha_i s \text{ if } d_{ik_1} < s \leq d_{ik_2}, \\
&\vdots \\
\mathcal{H}_{\mathrm{P}}(i,s) &= \phi(i,k_r) - \alpha_i s \text{ if } d_{ik_{r-1}} < s \leq d_{ik_r},
\end{aligned}$$

where $k_j = \text{argmin}\{\phi(i,k) : k \in \mathcal{V}(i) \text{ and } d_{ik} > d_{ik_{j-1}}\}$.

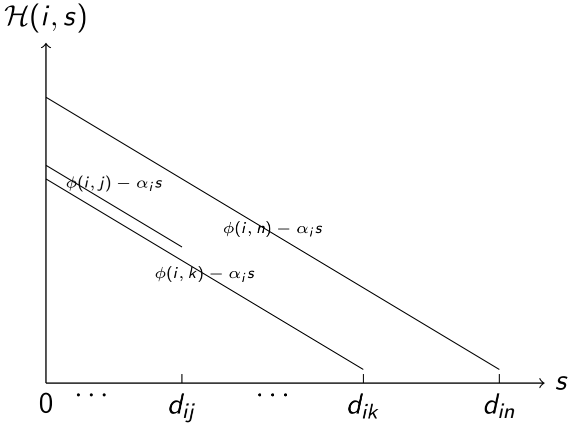
The Algorithm (The Induction Step)

The production value function:



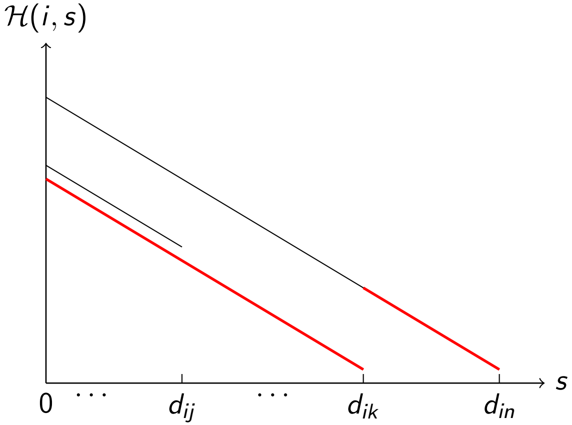
The Algorithm (The Induction Step)

Step 1: Calculate $\mathcal{H}_P(i, s)$ for $s = d_{1k} - d_{1i^-}$ for each $k \in \mathcal{V}$



The Algorithm (The Induction Step)

Step 1: Calculate $\mathcal{H}_{\text{P}}(i, s)$ for $s = d_{1k} - d_{1i^-}$ for each $k \in \mathcal{V}$



The Algorithm (The Induction Step)

Step 2: Calculate and store $\mathcal{H}_{\text{NP}}(i,s)$ for $s = d_{1k} - d_{1i^-}$ for each node $k \in \mathcal{V}$: since $\theta(i,s) = \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s)$ for $s = d_{1k} - d_{1i}$ for each node $k \in \mathcal{V}$ is obtained, this step can be obtained in $\mathcal{O}(n)$ time;

Step 3: Calculate and store $\mathcal{H}(i,s)$ for $s = d_{1k} - d_{1i^-}$ for each node $k \in \mathcal{V}$: this step can be completed in $\mathcal{O}(n)$ time;

Step 4: Update $\theta(i^-, s)$ for $s = d_{1k} - d_{1i^-}$ for each node $k \in \mathcal{V}$: increase $\theta(i^-, d_{1k} - d_{1i^-})$ by $\mathcal{H}(i, d_{1k} - d_{1i^-})$ for each node $k \in \mathcal{V}$. This step can be completed in $\mathcal{O}(n)$ time.

Computational Complexity for SULS with Backlogging

Theorem: The general SULS with backlogging can be solved in $\mathcal{O}(n^2)$ time, regardless of the scenario tree structure.

Remark: For the case that initial inventory level is a decision variable, it can be transformed into the case with zero initial inventory by adding a dummy root node 0 as the parent node of node 1.

Remark: In this paper, a more efficient algorithm is developed comparing to the one studied in Guan and Miller (2007) in which backlogging is not allowed and the computational complexity is $\mathcal{O}(n^2 \max\{\mathcal{C}, \log n\})$.

Outline

1 Introduction
- Motivation
- Related Work

2 Our Results/Contribution
- Stochastic Uncapacitated Lot-Sizing (SULS) with Backlogging
- Stochastic Capacitated Lot-Sizing (SCLS) with Backlogging
- Stochastic Constant Capacitated Lot-Sizing (SCCLS) with Backlogging

3 Current Research

4 Summary and Future Research

The Production Path Property

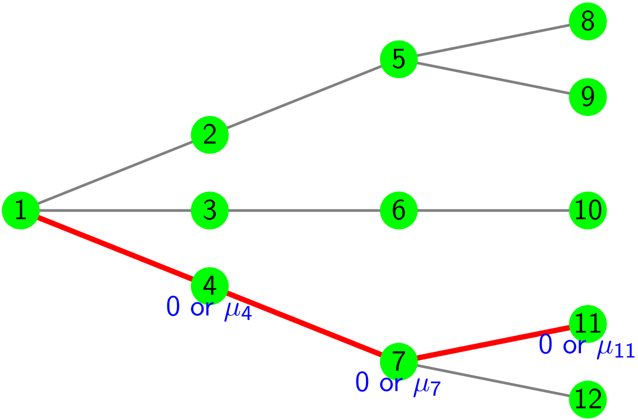
SCLS Production Path Property

For any instance of SCLS with backlogging, there exists an optimal solution (x^*, y^*, s^*) such that for each node $i \in \mathcal{V}$,

$$\text{if } 0 < x_i^* < \mu_i, \text{ then } x_i^* + s_{i^-}^* = d_{ik} - \sum_{j \in S} \mu_j \text{ for some node } k \in \mathcal{V}(i)$$
$$\text{and } S \subseteq \mathcal{P}(k) \setminus \mathcal{P}(i) \text{and } x_j^* = 0 \text{ or } \mu_j \text{ for each } j \in \mathcal{P}(k) \setminus \mathcal{P}(i).$$

In other words, there always exists an optimal solution such that if we produce at a node i, then we produce enough to satisfy demands along the path from node i to some descendant of node i besides some nodes along this path producing at their capacities.

The Production Path Property



Production Value Function:

$$\mathcal{H}_{\mathrm{P}}^{\tilde{\mu}}(i,s) = \min_{k\in\mathcal{V}(i): d_{ik}-\sum_{j\in S}\mu_j-\mu_i\le s<d_{ik}-\sum_{j\in S}\mu_j \text{ and } S\subseteq\mathcal{P}(k)\setminus\mathcal{P}(i)} \mathcal{H}_{\mathrm{P}}^{k,S}(i,s),$$

where

$$\begin{aligned}\mathcal{H}_{\mathrm{P}}^{k,S}(i,s) &= \beta_i + \alpha_i(d_{ik} - \sum_{j\in S}\mu_j - s) \\ &+ \max\left\{h_i(d_{ik} - \sum_{j\in S}\mu_j - d_i), -b_i(d_{ik} - \sum_{j\in S}\mu_j - d_i)\right\} \\ &+ \sum_{\ell\in\mathcal{C}(i)} \mathcal{H}(\ell, d_{ik} - \sum_{j\in S}\mu_j - d_i).\end{aligned}$$

Value Functions

- Production at Capacity

$$\mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s) = \beta_i + \alpha_i\mu_i + \max\{h_i(\mu_i + s - d_i), -b_i(\mu_i + s - d_i)\} + \sum_{\ell\in\mathcal{C}(i)} \mathcal{H}(\ell, \mu_i + s - d_i)$$

- Non-Production Value Function

$$\mathcal{H}_{\mathrm{NP}}(i,s) = \max\{h_i(s-d_i), -b_i(s-d_i)\} + \sum_{\ell\in\mathcal{C}(i)} \mathcal{H}(\ell, s-d_i)$$

- Value Function

$$\mathcal{H}(i,s) = \min\left\{\mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s), \mathcal{H}_{\mathrm{P}}^{\tilde{\mu}}(i,s), \mathcal{H}_{\mathrm{NP}}(i,s)\right\}$$

Value Functions

- Production at Capacity

$$\mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s) = \beta_i + \alpha_i\mu_i + \max\{h_i(\mu_i + s - d_i), -b_i(\mu_i + s - d_i)\} + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, \mu_i + s - d_i)$$

- Non-Production Value Function

$$\mathcal{H}_{\mathrm{NP}}(i,s) = \max\{h_i(s - d_i), -b_i(s - d_i)\} + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s - d_i)$$

- Value Function

$$\mathcal{H}(i,s) = \min\left\{\mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s), \mathcal{H}_{\mathrm{P}}^{\tilde{\mu}}(i,s), \mathcal{H}_{\mathrm{NP}}(i,s)\right\}$$

Value Functions

- Production at Capacity

$$\mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s) = \beta_i + \alpha_i\mu_i + \max\{h_i(\mu_i + s - d_i), -b_i(\mu_i + s - d_i)\} + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, \mu_i + s - d_i)$$

- Non-Production Value Function

$$\mathcal{H}_{\mathrm{NP}}(i,s) = \max\{h_i(s - d_i), -b_i(s - d_i)\} + \sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s - d_i)$$

- Value Function

$$\mathcal{H}(i,s) = \min\left\{\mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s), \mathcal{H}_{\mathrm{P}}^{\tilde{\mu}}(i,s), \mathcal{H}_{\mathrm{NP}}(i,s)\right\}$$

Observations

Observation: The summation of piecewise linear and continuous functions is still a piecewise linear and continuous function

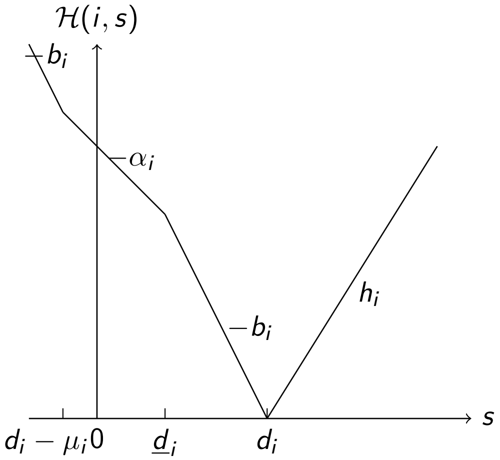
Observation: The minimum of two piecewise linear and continuous functions is still a piecewise linear and continuous function

Observations

Observation: The summation of piecewise linear and continuous functions is still a piecewise linear and continuous function

Observation: The minimum of two piecewise linear and continuous functions is still a piecewise linear and continuous function

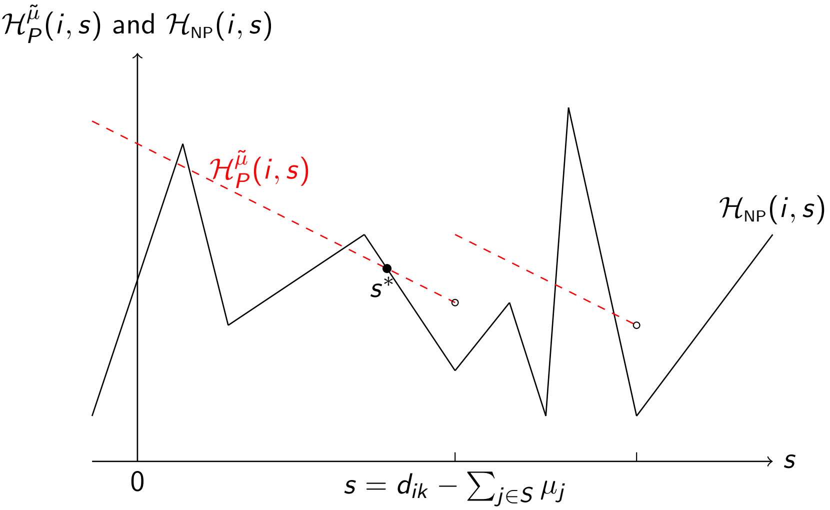
Value Function for Leaf Nodes



A Lemma

Lemma: The value function $\mathcal{H}_P^{\tilde{\mu}}(i, s)$ for each node $i \in \mathcal{V}$ is right continuous and piecewise linear with the same slope $-\alpha_i$. Corresponding to each piece of the production value function ending with $s = d_{ik} - \sum_{j \in S} \mu_j$, if it intercrosses with $\mathcal{H}_{\text{NP}}(i, s)$ and both $\mathcal{H}_{\text{NP}}(i, s)$ and $\sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, S)$ are piecewise linear and continuous, then there exists a point s^* such that $\mathcal{H}_{\text{NP}}(i, s^*) = \mathcal{H}_{\text{P}}^{\tilde{\mu}}(i, s^*)$ and $\mathcal{H}_{\text{NP}}(i, s) < \mathcal{H}_{\text{P}}^{\tilde{\mu}}(i, s)$ for each $s > s^*$ within the piece.

Value Functions $\mathcal{H}_{\text{P}}^{\tilde{\mu}}(i,s)$ vs $\mathcal{H}_{\text{NP}}(i,s)$



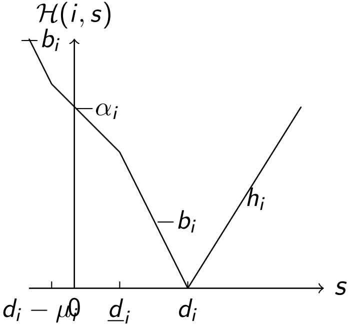
Value Function $\mathcal{H}(i, s)$

Proposition: The value functions $\mathcal{H}(i, s), \mathcal{H}_P^{\mu_i}(i, s)$, and $\mathcal{H}_{\text{NP}}(i, s)$ for each node $i \in \mathcal{V}$ are piecewise linear and continuous.

Proof sketch.

1. $\mathcal{H}'(i, s) = \min\{\mathcal{H}_{\text{NP}}(i, s), \mathcal{H}_{\text{P}}^{\mu_i}(i, s)\}$ is piecewise linear and continuous.
2. If the jump point is in the form of $s = d_{ik} - \sum_{j \in S} \mu_j$, then according to Lemma,
$$\mathcal{H}'(i, s) \leq \mathcal{H}_{\text{NP}}(i, s) \leq \mathcal{H}_{\text{P}}^{\tilde{\mu}}(i, s).$$
3. If the jump point is in the form of $s = d_{ik} - \sum_{j \in S} \mu_j - \mu_i$, then the production reaches its capacity and
$$\mathcal{H}'(i, s) \leq \mathcal{H}_{\text{P}}^{\mu_i}(i, s) = \mathcal{H}_{\text{P}}^{\tilde{\mu}}(i, s).$$

Dynamic Programming Recursion: Leaf Nodes



$$
\mathcal{H}(i,s) = \begin{cases} \beta_i + \alpha_i \mu_i \\ +(d_i - \mu_i - s) & \text{if } s < d_i - \mu_i, \\ \alpha_i(d_i - s) + \beta_i & \text{if } d_i - \mu_i \le s < \underline{d}_i, \\ b_i(d_i - s) & \text{if } \underline{d}_i \le s < d_i, \\ h_i(s - d_i) & \text{if } s \ge d_i, \end{cases}
$$

Number of Breakpoints

The breakpoint $s = s'$ is a "convex" breakpoint of $\mathcal{H}(i,s)$ if

$$\lim_{s \to s'^-} \partial \mathcal{H}(i,s)/\partial s < \lim_{s \to s'^+} \partial \mathcal{H}(i,s)/\partial s.$$

Proposition

All "convex" breakpoints in $\mathcal{H}(i,s)$, $\mathcal{H}_{\text{NP}}(i,s)$, and $\mathcal{H}_{\text{P}}^{\mu_i}(i,s)$ are in the form $s = d_{ik} - \sum_{j \in S} \mu_j$ for some node $k \in \mathcal{V}(i)$ and $S \subseteq \mathcal{P}(k) \setminus \mathcal{P}(i^-)$.

Number of Breakpoints

Lemma: The number of breakpoints of the non-production value function $|B_{\text{NP}}(i)| \leq \sum_{\ell \in \mathcal{C}(i)} |B(\ell)|$ if $|\mathcal{C}(i)| \geq 2$ for each $i \in \mathcal{V} \setminus \mathcal{L}$.

Lemma: The breakpoints for $\mathcal{H}_{\text{P}}^{\mu_i}(i,s)$ belong to the node set in which all the "old" breakpoints in $\mathcal{H}_{\text{NP}}(i,s)$ are shifted to the left by μ_i.

Lemma: The number of breakpoints generated by $\mathcal{H}(i,s)$ is at most $4|B_{\text{NP}}(i)|$.

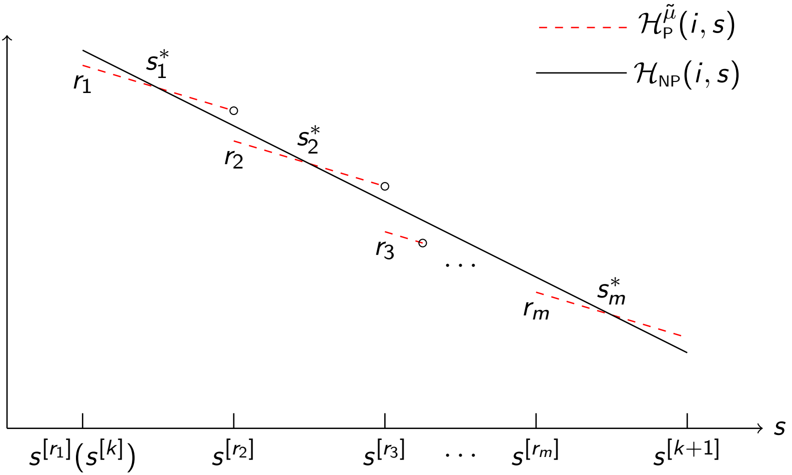
More on $|B(i)| \leq 4|B_{\text{NP}}(i)|$

1. The number of breakpoints of $\mathcal{H}_{\text{P}}^{\mu_i}(i,s)$ is the same as the number of breakpoints of $\mathcal{H}_{\text{NP}}(i,s)$.
2. Denote the breakpoints for $\mathcal{H}_{\text{NP}}(i,s)$ as $s = s^{[1]}, s^{[2]}, \ldots, s^{[m]}$ such that $-\infty = s^{[1]} \leq s^{[2]} \leq \ldots \leq s^{[m]}$, and the interval $[s^{[k]}, s^{[k+1]})$ as the k^{th} interval in $\mathcal{H}_{\text{NP}}(i,s)$.
3. For each interval, we want to prove

$$|B(i)|_k \leq 2(|B_{\text{NP}}(i)|_k + |B_{\text{P}}^{\mu_i}(i)|_k).$$

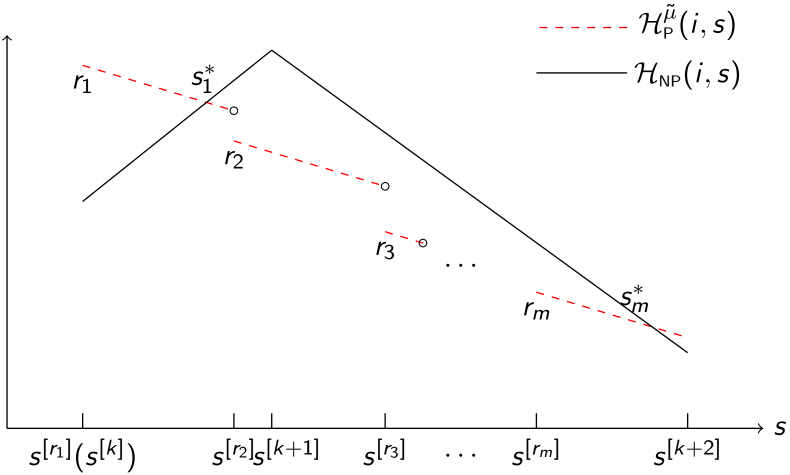
Two Cases

CASE 1: $\mathcal{H}_{\text{NP}}(i, s^{[k]}) > \mathcal{H}_{\text{P}}^{\tilde{\mu}}(i, s^{[k]})$.

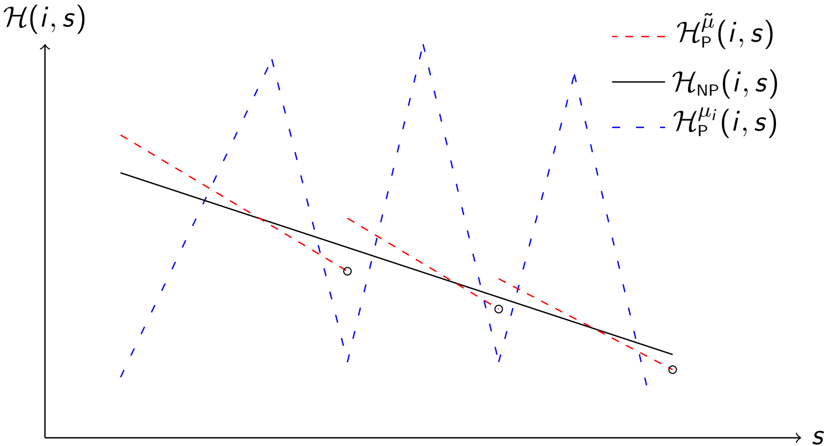


Two Cases

CASE 2: $\mathcal{H}_{\text{NP}}(i, s^{[k]}) \leq \mathcal{H}_{\text{P}}^{\tilde{\mu}}(i, s^{[k]})$.



An Example



$\mathcal{H}(i,s)$
$\mathcal{H}_P^{\tilde{\mu}}(i,s)$
$\mathcal{H}_{NP}(i,s)$
$\mathcal{H}_P^{\mu_i}(i,s)$
s

Number of Breakpoints

Proposition

The total number of breakpoints $|B(i)|$ is bounded by $\mathcal{O}(|\mathcal{V}(i)|^3)$.

Proof sketch.

1. For each leaf node i, $|B(i)| \leq 4 = 4|\mathcal{V}(i)|$.
2. If $|B(\ell)| \leq 4^{T-t(\ell)+1}|\mathcal{V}(\ell)|$, then

$$\begin{aligned} |B(i)| &\leq 4 \sum_{\ell \in \mathcal{C}(i)} |B(\ell)| \leq 4 \sum_{\ell \in \mathcal{C}(i)} 4^{T-t(\ell)+1}|\mathcal{V}(\ell)| \\ &= 4^{T-t(i)+1} \sum_{\ell \in \mathcal{C}(i)} |\mathcal{V}(\ell)| \leq 4^{T-t(i)+1}|\mathcal{V}(i)|. \end{aligned}$$

3. The number of breakpoints

$$|B(i)| \leq 4^{T-t(i)+1}|\mathcal{V}(i)| \leq (|\mathcal{V}(i)|+1)^2|\mathcal{V}(i)|,$$

since

$$|\mathcal{V}(i)| \geq 1 + 2^1 + \ldots + 2^{T-t(i)} = 2^{T-t(i)+1} - 1.$$

Computational Complexity for SCLS

Theorem: If $\mathcal{C}(i) \geq 2$ for each node $i \in \mathcal{V} \setminus \mathcal{L}$, then the optimal value function of SCLS with backlogging can be obtained in $\mathcal{O}(n^4)$ time.

Algorithm sketch

1. For the leaf node, we need to calculate and store 4 breakpoints.
2. Calculate and store $\sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s)$: Since $\theta(i, s)$ is obtained when we finish calculating all children of node i, this step can be completed in $\mathcal{O}(n^3)$ time.
3. Obtain and store $\mathcal{H}_{\text{NP}}(i, s)$: This step can be obtained by moving $\sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s)$ to the right by d_i units plus the piecewise linear function $\max\{h_i(s - d_i), -b_i(s - d_i)\}$. It can be completed in $\mathcal{O}(n^3)$ time.

Computational Complexity for SCLS

Theorem: If $\mathcal{C}(i) \geq 2$ for each node $i \in \mathcal{V} \setminus \mathcal{L}$, then the optimal value function of SCLS with backlogging can be obtained in $\mathcal{O}(n^4)$ time.

Algorithm sketch

1. For the leaf node, we need to calculate and store 4 breakpoints.
2. Calculate and store $\sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s)$: Since $\theta(i, s)$ is obtained when we finish calculating all children of node i, this step can be completed in $\mathcal{O}(n^3)$ time.
3. Obtain and store $\mathcal{H}_{\text{NP}}(i, s)$: This step can be obtained by moving $\sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s)$ to the right by d_i units plus the piecewise linear function $\max\{h_i(s - d_i), -b_i(s - d_i)\}$. It can be completed in $\mathcal{O}(n^3)$ time.

Computational Complexity for SCLS

Theorem: If $\mathcal{C}(i) \geq 2$ for each node $i \in \mathcal{V} \setminus \mathcal{L}$, then the optimal value function of SCLS with backlogging can be obtained in $\mathcal{O}(n^4)$ time.

Algorithm sketch

1. For the leaf node, we need to calculate and store 4 breakpoints.
2. Calculate and store $\sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s)$: Since $\theta(i, s)$ is obtained when we finish calculating all children of node i, this step can be completed in $\mathcal{O}(n^3)$ time.
3. Obtain and store $\mathcal{H}_{\text{NP}}(i, s)$: This step can be obtained by moving $\sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s)$ to the right by d_i units plus the piecewise linear function $\max\{h_i(s - d_i), -b_i(s - d_i)\}$. It can be completed in $\mathcal{O}(n^3)$ time.

Computational Complexity for SCLS

4. Obtain and store $\mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s)$: It can be obtained by moving $\mathcal{H}_{\mathrm{NP}}(i,s)$ to the left by μ_i plus a constant number $\beta_i + \alpha_i \mu_i$. This step can be completed in $\mathcal{O}(n^3)$ time.
5. Calculate and store $\mathcal{H}'(i,s) = \min\{\mathcal{H}_{\mathrm{NP}}(i,s), \mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s)\}$: Take the minimum of the two functions between any two consecutive breakpoints in $\mathcal{H}_{\mathrm{NP}}(i,s)$ and $\mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s)$.

Computational Complexity for SCLS

4. Obtain and store $\mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s)$: It can be obtained by moving $\mathcal{H}_{\mathrm{NP}}(i,s)$ to the left by μ_i plus a constant number $\beta_i + \alpha_i \mu_i$. This step can be completed in $\mathcal{O}(n^3)$ time.
5. Calculate and store $\mathcal{H}'(i,s) = \min\{\mathcal{H}_{\mathrm{NP}}(i,s), \mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s)\}$: Take the minimum of the two functions between any two consecutive breakpoints in $\mathcal{H}_{\mathrm{NP}}(i,s)$ and $\mathcal{H}_{\mathrm{P}}^{\mu_i}(i,s)$.

Computational Complexity for SCLS

6. Among the breakpoints generated by $\mathcal{H}_{\text{NP}}(i,s)$ and $\mathcal{H}_{\text{P}}^{\mu_i}(i,s)$, calculate $\phi(k,S) =$
$\beta_i + \alpha_i(d_{ik} - \sum_{j\in S}\mu_j) + \max\{h_i\Delta_{k,S}, -b_i\Delta_{k,S}\} + \sum_{\ell\in\mathcal{C}(i)}\mathcal{H}(\ell,\Delta_{k,S})$
where $\Delta_{k,S} = d_{ik} - \sum_{j\in S}\mu_j - d_i$ for each node $k \in \mathcal{V}(i)$ and
$S \subseteq \mathcal{P}(k) \setminus \mathcal{P}(i)$.

   For each combination of a node $k \in \mathcal{V}(i)$ and a set $S \subseteq \mathcal{P}(k) \setminus \mathcal{P}(i)$, based on the result in Step 2, the value of the function $\sum_{\ell\in\mathcal{C}(i)}\mathcal{H}(\ell,\Delta_{k,S})$ can be obtained by binary search in $\mathcal{O}(\log n^3) = \mathcal{O}(\log n)$ time. Therefore, this entire step can be completed in $\mathcal{O}(n^2 \log n)$ time.

Computational Complexity for SCLS

7. Calculate and store $\mathcal{H}(i,s) = \min\{\mathcal{H}_{\mathrm{P}}(i,s), \mathcal{H}'(i,s)\}$:

   Sort $\phi(k,S)$ in a non-decreasing order and build a corresponding list ξ_1, which takes $\mathcal{O}(n^2 \log n)$ time.

   Build a list ξ_2 to store the start and end breakpoints for all linear pieces of $\mathcal{H}_{\mathrm{P}}(i,s)$. All these linear pieces are stored according to increasing sequence of their start breakpoint inventory values. Initially $\xi_2 = \emptyset$.

   Starting from the first one in ξ_1, for each pair k and S, we generate the value functions $\mathcal{H}_{\mathrm{P}}(i,s)$ and $\mathcal{H}(i,s)$ for the interval $[d_{ik} - \sum_{j \in S} \mu_j - \mu_i, d_{ik} - \sum_{j \in S} \mu_j)$ in the following steps:

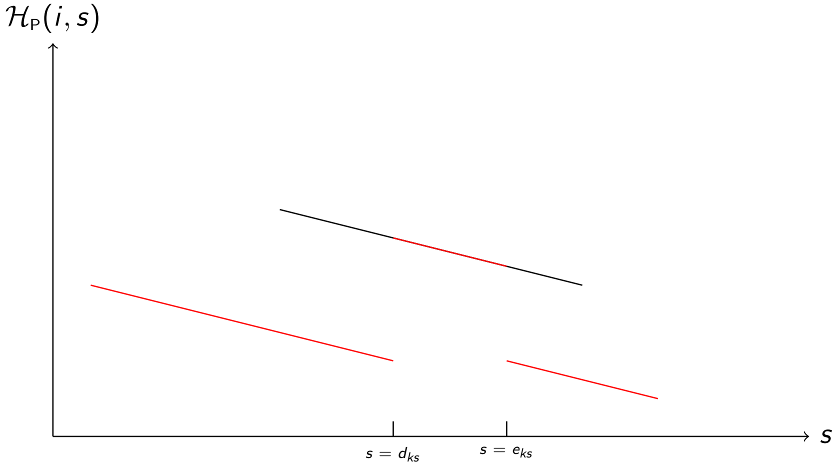
Computational Complexity for SCLS

7.

1. Find a linear piece in ξ_2 whose end breakpoint is the largest and in the interval $[d_{ik} - \sum_{j \in S} \mu_j - \mu_i, d_{ik} - \sum_{j \in S} \mu_j)$. Denote it as $s = d_{ks}$;
2. Find a linear piece in ξ_2 whose start breakpoint is the smallest and in the interval $[d_{ik} - \sum_{j \in S} \mu_j - \mu_i, d_{ik} - \sum_{j \in S} \mu_j)$. Denote it as $s = e_{ks}$;
3. Due to fixed interval length μ_i for each pair, the value of $\mathcal{H}_{\mathrm{P}}(i, s)$ in the interval $[d_{ks}, e_{ks})$ should be equal to $\phi(k, S) - \alpha_i s$ and the corresponding values for other parts of the interval $[d_{ik} - \sum_{j \in S} \mu_j - \mu_i, d_{ik} - \sum_{j \in S} \mu_j)$ are decided by other pairs;
4. Obtain $\mathcal{H}(i, s) = \min\{\mathcal{H}_{\mathrm{P}}(i, s), \mathcal{H}'(i, s)\}$ for the interval $[d_{ks}, e_{ks})$ and insert the linear piece of $\mathcal{H}_{\mathrm{P}}(i, s)$ for the interval $[d_{ks}, e_{ks})$ to the right position in ξ_2 to maintain the increasing sequence.

Note that there are at most $\mathcal{O}(n^2)$ pairs. We can use the binary search to find the start and end breakpoints, which takes $\mathcal{O}(\log n)$ time. The number of breakpoints in $\mathcal{H}'(i, s)$ is bounded by $\mathcal{O}(n^3)$. Thus, this step can be finished in $\mathcal{O}(n^3)$ time.

Computational Complexity for SCLS



8. Update $\theta(i^-, s)$ by adding $\mathcal{H}(i, s)$: This step can be completed in $\mathcal{O}(n^3)$ time since the number of breakpoints is bounded by $\mathcal{O}(n^3)$.

Outline

1. Introduction
   - Motivation
   - Related Work
2. Our Results/Contribution
   - Stochastic Uncapacitated Lot-Sizing (SULS) with Backlogging
   - Stochastic Capacitated Lot-Sizing (SCLS) with Backlogging
   - Stochastic Constant Capacitated Lot-Sizing (SCCLS) with Backlogging
3. Current Research
4. Summary and Future Research

Production Path Property

SCCLS Production Path Property

There exists an optimal solution (x^*, y^*, s^*) such that for each node $i \in \mathcal{V}$,

$$\text{if } 0 < x_i^* < \mu, \text{ then } x_i^* + s_{i^-}^* = d_{ik} - m\mu \text{ for some } m$$
$$\text{such that } d_{ik} - m\mu \geq 0 \text{ and } m < T,$$
$$\text{and } x_j^* = 0 \text{ or } \mu \text{ for each } j \in \mathcal{P}(i,k) \setminus \{i\}.$$

Proposition

For the stochastic constant capacitated lot-sizing problem with zero initial inventory, we have $s_i^* = d_{1j} - d_{1i} + m\mu$ with $-T \leq m \leq t(i)$ in an optimal solution.

Complexity for SCCLS

Theorem: The general stochastic constant capacitated lot-sizing problem with backlogging can be solved in $\mathcal{O}(n^2 T \log n)$ time.

Algorithm sketch.

1. Breakpoints of $\sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s)$: $\mathcal{O}(nT)$.
2. Store $\sum_{\ell \in \mathcal{C}(i)} \mathcal{H}(\ell, s)$: $\mathcal{O}(nT)$.
3. Calculate and store $\mathcal{H}_{\mathrm{NP}}(i, s)$ and $\mathcal{H}_{\mathrm{P}}^{\mu}(i, s)$: $\mathcal{O}(nT)$.
4. Calculate and store $\phi(k, m)$: $\mathcal{O}(nT \log n)$.
5. Calculate and store $\mathcal{H}_{\mathrm{P}}(i, s)$: $\mathcal{O}(nT \log n)$.
6. Calculate and store $\mathcal{H}(i, s)$: $\mathcal{O}(nT)$.

Stochastic Lot-Sizing with Inventory Bounds

- Equivalence among four models (de Heuvel and Wagelmans 2007): lot-sizing with inventory bounds, remanufacturing option, production time-window constraints, and cumulative capacity constraints.
- For the uncapacitated case, we have an $\mathcal{O}(n^2)$ time algorithm, which generalizes the deterministic case studied by Atamtürk and Kücükyavuz 2007 and Liu 2007 to the stochastic setting with the same computational complexity.

  For the constant capacitated case, we have an $\mathcal{O}(n^2 T \log n)$ time algorithm, which generalizes the deterministic case studied by Wolsey 2007 with complexity $\mathcal{O}(T^4)$ to the stochastic setting with a better computational complexity. (Joint work with Tieming Liu)

Stochastic Lot-Sizing with Inventory Bounds

- Equivalence among four models (de Heuvel and Wagelmans 2007): lot-sizing with inventory bounds, remanufacturing option, production time-window constraints, and cumulative capacity constraints.
- For the uncapacitated case, we have an $\mathcal{O}(n^2)$ time algorithm, which generalizes the deterministic case studied by Atamtürk and Kücükyavuz 2007 and Liu 2007 to the stochastic setting with the same computational complexity.
- For the constant capacitated case, we have an $\mathcal{O}(n^2 T \log n)$ time algorithm, which generalizes the deterministic case studied by Wolsey 2007 with complexity $\mathcal{O}(T^4)$ to the stochastic setting with a better computational complexity. (Joint work with Tieming Liu)

Stochastic Lot-Sizing with Inventory Bounds

- Equivalence among four models (de Heuvel and Wagelmans 2007): lot-sizing with inventory bounds, remanufacturing option, production time-window constraints, and cumulative capacity constraints.
- For the uncapacitated case, we have an $\mathcal{O}(n^2)$ time algorithm, which generalizes the deterministic case studied by Atamtürk and Kücükyavuz 2007 and Liu 2007 to the stochastic setting with the same computational complexity.

  For the constant capacitated case, we have an $\mathcal{O}(n^2 T \log n)$ time algorithm, which generalizes the deterministic case studied by Wolsey 2007 with complexity $\mathcal{O}(T^4)$ to the stochastic setting with a better computational complexity. (Joint work with Tieming Liu)

Summary and Future Research

- Generalized the work by Guan and Miller 2007 to include backlogging and variant capacities.
- Show that the value function for SCLS with backlogging can be achieved in polynomial time $\mathcal{O}(n^4)$ if each non-leaf node contains at least two children.
- More efficient algorithms found for SULS (i.e., $\mathcal{O}(n^2)$) and SCCLS (i.e., $\mathcal{O}(n^2 T \log n)$), regardless of the scenario tree structure.

Future Research

- Study the cases with nonlinear objective value functions (joint work with Andrew Miller).
- How can these results help find the convex hull description of SULS as well as SCLS polytopes? We are working on lifted valid inequalities for stochastic dynamic knapsack problems (joint work with Bo Zeng).