Improving branch-and-price algorithms for solving 1D cutting stock problem

Soumitra Pal

under the guidance of

Prof. A. G. Ranade

January 17, 2007
Cutting stock problem

▶ Given larger raw paper rolls.
▶ Cut given number of final rolls of smaller widths
▶ Use minimum number of raw rolls.

▶ Input:
  ▶ Raw roll width $W$
  ▶ Set of finals $\{w_1, w_2, \ldots, w_m\}$
  ▶ Set of demands $\{b_1, b_2, \ldots, b_m\}$

▶ Output:
  ▶ Minimum no of raws and
  ▶ the cutting patterns
Outline of the presentation

Introduction

- Gilmore-Gomory formulation
- Column generation
- Branch-and-price
- Areas of improvements

Work done in the second stage

- GoodPrice: a new technique for accelerating column generation
- Dynamic prog solution to knapsack with forbidden list
- Random knapsack in polynomial time
- Other branching schemes

Work planned for the final stage
Gilmore-Gomory formulation

All possible valid cutting patterns are enumerated beforehand.

Let $J$ be the set of valid cutting patterns.

The pattern $j$ is represented by $(a_{1j}, \ldots, a_{ij}, \ldots, a_{mj})$ where $a_{ij}$ represents the number of rolls of width $w_i$ obtained from pattern $j$.

Let $\lambda_j$ denote the number of rolls to be cut according to pattern $j$.

Formulation:

$$\min \sum_{j \in J} \lambda_j$$

subject to

$$\sum_{j \in J} a_{ij} \lambda_j \geq b_i, \quad i = 1, 2, \ldots, m$$

$\lambda_j$ integer and $\geq 0$, $\forall j \in J$.

Solve the LP relaxation.
Gilmore-Gomory formulation

- All possible valid cutting patterns are enumerated beforehand
Gilmore-Gomory formulation

- All possible valid cutting patterns are enumerated beforehand
- Let $J$ be the set of valid cutting patterns

Formulation:

$$\min \sum_{j \in J} \lambda_j$$

subject to

$$\sum_{j \in J} a_{ij} \lambda_j \geq b_i, \quad i = 1, 2, \ldots, m$$

$\lambda_j$ integer and $\geq 0$, $\forall j \in J$

Solve the LP relaxation
Gilmore-Gomory formulation

- All possible valid cutting patterns are enumerated beforehand
- Let $J$ be the set of valid cutting patterns
- The pattern $j$ is represented by $(a_{1j}, \ldots, a_{ij}, \ldots, a_{mj})$ where $a_{ij}$ represents the number of rolls of width $w_i$ obtained from pattern $j$

Formulation:

$$\min \sum_{j \in J} \lambda_j$$
subject to

$$\sum_{j \in J} a_{ij} \lambda_j \geq b_i, \quad i = 1, 2, \ldots, m$$

$\lambda_j$ integer and $\geq 0, \forall j \in J$

Solve the LP relaxation
Gilmore-Gomory formulation

- All possible valid cutting patterns are enumerated beforehand
- Let \( J \) be the set of valid cutting patterns
- The pattern \( j \) is represented by \((a_{1j}, \cdots, a_{ij}, \cdots, a_{mj})\) where \( a_{ij} \) represents the number of rolls of width \( w_i \) obtained from pattern \( j \)
- Let \( \lambda_j \) denotes the number of rolls to be cut according to pattern \( j \)
Gilmore-Gomory formulation

- All possible valid cutting patterns are enumerated beforehand
- Let $J$ be the set of valid cutting patterns
- The pattern $j$ is represented by $(a_{1j}, \cdots, a_{ij}, \cdots, a_{mj})$ where $a_{ij}$ represents the number of rolls of width $w_i$ obtained from pattern $j$
- Let $\lambda_j$ denotes the number of rolls to be cut according to pattern $j$
- **Formulation:**

$$\begin{align*}
\min & \quad \sum_{j \in J} \lambda_j \\
\text{s.t.} & \quad \sum_{j \in J} a_{ij} \lambda_j \geq b_i, \quad i = 1, 2, \cdots, m \\
& \quad \lambda_j \text{ integer and } \geq 0, \quad \forall j \in J
\end{align*}$$
Gilmore-Gomory formulation

- All possible valid cutting patterns are enumerated beforehand
- Let \( J \) be the set of valid cutting patterns
- The pattern \( j \) is represented by \((a_{1j}, \ldots, a_{ij}, \ldots, a_{mj})\) where \( a_{ij} \) represents the number of rolls of width \( w_i \) obtained from pattern \( j \)
- Let \( \lambda_j \) denotes the number of rolls to be cut according to pattern \( j \)
- **Formulation:**
  
  \[
  \min \sum_{j \in J} \lambda_j \\
  \text{s.t.} \quad \sum_{j \in J} a_{ij} \lambda_j \geq b_i, \quad i = 1, 2, \ldots, m \\
  \lambda_j \text{ integer and } \geq 0, \quad \forall j \in J
  \]

- Solve the LP relaxation
Difficulties with Gilmore-Gomory formulation

- Huge number of variables or columns
Difficulties with Gilmore-Gomory formulation

- Huge number of variables or columns

- Though in most of practical instances the integer optimum differs from rounded up LP optimum by at most 1, the integer solution vectors differs widely from the fractional solution vector
Column generation to tackle huge no of variables

- In the final solution only a very few patterns are used

\[
\text{Let the dual solution to current restricted master problem (RMP) be } \pi_1, \pi_2, \ldots, \pi_m.
\]

\[
\text{Solve the following subproblem: } \max \sum_{i \in [1..m]} \pi_i a_i,
\]

s.t. \[
\sum_{i \in [1..m]} w_i a_i \leq W_a \quad \text{integer and } \geq 0, \forall i \in [1..m].
\]

- If solution to sub problem is > 1, the optimum of RMP can be reduced by adding the column and hence repeat.
- Otherwise, solution to RMP is the solution to the MP.
Column generation to tackle huge no of variables

- In the final solution only a very few patterns are used
- Idea: start with only a few columns ensuring feasibility and add new columns as and when necessary

Let the dual solution to current restricted master problem (RMP) be $\pi_1, \pi_2, \ldots, \pi_m$

Solve the following subproblem

\[
\max \sum_{i \in [1..m]} \pi_i a_i, \quad \text{s.t.} \quad \sum_{i \in [1..m]} w_i a_i \leq W a_i \text{ integer and } \geq 0, \quad \forall i \in [1..m]
\]

If solution to sub problem is $> 1$, the optimum of RMP can be reduced by adding the column $a$ and hence repeat

Otherwise, solution to RMP is the solution to the MP.
Column generation to tackle huge no of variables

- In the final solution only a very few patterns are used
- Idea: start with only a few columns ensuring feasibility and add new columns as and when necessary
- Let the dual solution to current restricted master problem (RMP) be $\pi_1, \pi_2, \ldots, \pi_m$
Column generation to tackle huge no of variables

- In the final solution only a very few patterns are used
- Idea: start with only a few columns ensuring feasibility and add new columns as and when necessary
- Let the dual solution to current restricted master problem (RMP) be $\pi_1, \pi_2, \ldots, \pi_m$
- Solve the following subproblem

$$\begin{align*}
\max & \quad \sum_{i \in [1..m]} \pi_i a_i, \\
\text{s.t.} & \quad \sum_{i \in [1..m]} w_i a_i \leq W \\
& \quad a_i \text{ integer and } \geq 0, \quad \forall i \in [1..m]
\end{align*}$$

- If solution to sub problem is $>1$, the optimum of RMP can be reduced by adding the column $a_i$ and hence repeat
- Otherwise, solution to RMP is the solution to the MP.
Column generation to tackle huge no of variables

- In the final solution only a very few patterns are used
- Idea: start with only a few columns ensuring feasibility and add new columns as and when necessary
- Let the dual solution to current restricted master problem (RMP) be $\pi_1, \pi_2, \ldots, \pi_m$
- Solve the following subproblem

$$\max \sum_{i \in [1..m]} \pi_i a_i, \quad \text{s.t.} \quad \sum_{i \in [1..m]} w_i a_i \leq W$$

$a_i$ integer and $\geq 0$, $\forall i \in [1..m]$

- If solution to sub problem is $> 1$, the optimum of RMP can be reduced by adding the column $a$ and hence repeat
In the final solution only a very few patterns are used.

Idea: start with only a few columns ensuring feasibility and add new columns as and when necessary.

Let the dual solution to current restricted master problem (RMP) be $\pi_1, \pi_2, \ldots, \pi_m$.

Solve the following subproblem:

$$\max \sum_{i \in [1..m]} \pi_i a_i, \quad \text{s.t.} \quad \sum_{i \in [1..m]} w_i a_i \leq W$$

$a_i$ integer and $\geq 0$, $\forall i \in [1..m]$

If solution to sub problem is $> 1$, the optimum of RMP can be reduced by adding the column $a$ and hence repeat.

Otherwise, solution to RMP is the solution to the MP.
Consider a fractional variable, $\lambda_1 = 4.7$. Divide the solution space into:

1. One subspace containing $\lambda_1 \geq 5$ (left)
2. Other subspace containing $\lambda_1 \leq 4$ (right)

Solve two instances of the MP with these two extra constraints, pick the best.

Similarly, if required, divide the solution space further. We get a tree in which at each node we solve the constrained MP and branch into two children. However, a node is pruned if the LP solution is worse than some integer solution obtained already.

This method of getting integer solution is known as branch-and-bound. In addition, the LP at each node is solved using column generation. This combined method is known as branch-and-price.
Branch-and-bound to get integer solution

- Consider a fractional variable, $\lambda_1 = 4.7$. Divide the solution space into
  1. One subspace containing $\lambda_1 \geq 5$ (left)
  2. Other subspace containing $\lambda_1 \leq 4$ (right)
Branch-and-bound to get integer solution

- Consider a fractional variable, $\lambda_1 = 4.7$. Divide the solution space into
  1. One subspace containing $\lambda_1 \geq 5$ (left)
  2. Other subspace containing $\lambda_1 \leq 4$ (right)

- Solve two instances of the MP with these two extra constraints, pick the best
Branch-and-bound to get integer solution

- Consider a fractional variable, $\lambda_1 = 4.7$. Divide the solution space into
  1. One subspace containing $\lambda_1 \geq 5$ (left)
  2. Other subspace containing $\lambda_1 \leq 4$ (right)
- Solve two instances of the MP with these two extra constraints, pick the best
- Similarly, if required, divide the solution space further
Branch-and-bound to get integer solution

- Consider a fractional variable, $\lambda_1 = 4.7$. Divide the solution space into
  1. One subspace containing $\lambda_1 \geq 5$ (left)
  2. Other subspace containing $\lambda_1 \leq 4$ (right)
- Solve two instances of the MP with these two extra constraints, pick the best
- Similarly, if required, divide the solution space further
- We get a tree in which at each node we solve the constrained MP and branch into two children
Branch-and-bound to get integer solution

- Consider a fractional variable, \( \lambda_1 = 4.7 \). Divide the solution space into
  1. One subspace containing \( \lambda_1 \geq 5 \) (left)
  2. Other subspace containing \( \lambda_1 \leq 4 \) (right)
- Solve two instances of the MP with these two extra constraints, pick the best
- Similarly, if required, divide the solution space further
- We get a tree in which at each node we solve the constrained MP and branch into two children
- However, a node is pruned if the LP solution is worse than some integer solution obtained already
Branch-and-bound to get integer solution

- Consider a fractional variable, $\lambda_1 = 4.7$. Divide the solution space into
  1. One subspace containing $\lambda_1 \geq 5$ (left)
  2. Other subspace containing $\lambda_1 \leq 4$ (right)
- Solve two instances of the MP with these two extra constraints, pick the best
- Similarly, if required, divide the solution space further
- We get a tree in which at each node we solve the constrained MP and branch into two children
- However, a node is pruned if the LP solution is worse than some integer solution obtained already
- This method of getting integer solution is known as branch-and-bound
Branch-and-bound to get integer solution

- Consider a fractional variable, $\lambda_1 = 4.7$. Divide the solution space into
  1. One subspace containing $\lambda_1 \geq 5$ (left)
  2. Other subspace containing $\lambda_1 \leq 4$ (right)

- Solve two instances of the MP with these two extra constraints, pick the best

- Similarly, if required, divide the solution space further

- We get a tree in which at each node we solve the constrained MP and branch into two children

- However, a node is pruned if the LP solution is worse than some integer solution obtained already

- This method of getting integer solution is known as branch-and-bound

- In addition, the LP at each node is solved using column generation. This combined method is known as branch-and-price
Solving subproblem in branch-and-price

- On the left branch ($\lambda_1 \geq 5$), it is easy to handle
  - In this case, adding the constraint is equivalent to reducing the demand vector by the column $j$ multiplied by 5 and solving the residual problem
  - No need to modify the subproblem
  - If the same column $j$ appears in the solution of the residual problem with a value $z$, we make $\lambda_j = 5 + z$ in the final solution

- On the right branch ($\lambda_1 \leq 4$), it is not so easy
  - The number of times pattern $j$ is used, is upper bounded
  - However, it is quite possible that the column may be generated by subproblem
  - The subproblem should not regenerate this forbidden column
  - Hence, at any node, the subproblem takes the form of a knapsack problem with a list of forbidden set of items
  - The form of subproblem will change if the branching scheme changes. Some other branching scheme will be described later.
Solving subproblem in branch-and-price

- On the left branch ($\lambda_1 \geq 5$), it is easy to handle
  - In this case, adding the constraint is equivalent to reducing the demand vector by the column $j$ multiplied by 5 and solving the residual problem
  - No need to modify the subproblem
  - If the same column $j$ appears in the solution of the residual problem with a value $z$, we make $\lambda_j = 5 + z$ in the final solution
- On the right branch ($\lambda_1 \leq 4$), it is not so easy
  - The number of times pattern $j$ is used, is upper bounded
  - However, it is quite possible that the column may be generated by subproblem
  - The subproblem should not regenerate this forbidden column
Solving subproblem in branch-and-price

- On the left branch ($\lambda_1 \geq 5$), it is easy to handle
  - In this case, adding the constraint is equivalent to reducing the demand vector by the column $j$ multiplied by 5 and solving the residual problem
  - No need to modify the subproblem
  - If the same column $j$ appears in the solution of the residual problem with a value $z$, we make $\lambda_j = 5 + z$ in the final solution
- On the right branch ($\lambda_1 \leq 4$), it is not so easy
  - The number of times pattern $j$ is used, is upper bounded
  - However, it is quite possible that the column may be generated by subproblem
    - The subproblem should not regenerate this forbidden column
- Hence, at any node, the subproblem takes the form of a knapsack problem with a list of forbidden set of items
Solving subproblem in branch-and-price

- On the left branch ($\lambda_1 \geq 5$), it is easy to handle
  - In this case, adding the constraint is equivalent to reducing the demand vector by the column $j$ multiplied by 5 and solving the residual problem
  - No need to modify the subproblem
  - If the same column $j$ appears in the solution of the residual problem with a value $z$, we make $\lambda_j = 5 + z$ in the final solution
- On the right branch ($\lambda_1 \leq 4$), it is not so easy
  - The number of times pattern $j$ is used, is upper bounded
  - However, it is quite possible that the column may be generated by subproblem
  - The subproblem should not regenerate this forbidden column
- Hence, at any node, the subproblem takes the form of a knapsack problem with a list of forbidden set of items
- The form of subproblem will change if the branching scheme changes. Some other branching scheme will be described later.
Conclusion from the survey on branch-and-price

- Accelerate column generation
- Solve the subproblem efficiently
- Devise efficient branching scheme
Work done in the second stage

- Accelerate Column Generation
  - A new technique named GoodPrice has been developed
Work done in the second stage

- Accelerate Column Generation
  - A new technique named GoodPrice has been developed
- Solve the subproblem efficiently
  - A dynamic programming solution to the knapsack problem with forbidden list has been proposed
  - A paper on random knapsack in polynomial time was surveyed
  - The proof of the same using the isolating lemma is outlined
Work done in the second stage

- Accelerate Column Generation
  - A new technique named **GoodPrice** has been developed
- Solve the subproblem efficiently
  - A dynamic programming solution to the knapsack problem with forbidden list has been proposed
  - A paper on random knapsack in polynomial time was surveyed
  - The proof of the same using the isolating lemma is outlined
- Devise efficient branching scheme
  - Two algorithms were studied
What happens in column generation? For solving the LP relaxation, initially, we take a very restricted or constrained form of the master problem with only a few columns enough to ensure the feasibility and gradually relax the formulation by adding new columns until we get an optimal solution.
What happens in column generation? For solving the LP relaxation, initially, we take a very restricted or constrained form of the master problem with only a few columns enough to ensure the feasibility and gradually relax the formulation by adding new columns until we get an optimal solution.

Any trickier formulation to accelerate this relaxation process should make the column generation process faster.
**GoodPrice: Idea**

**What happens in column generation?** For solving the LP relaxation, initially, we take a very restricted or constrained form of the master problem with only a few columns enough to ensure the feasibility and gradually relax the formulation by adding new columns until we get an optimal solution.

Any trickier formulation to accelerate this relaxation process should make the column generation process faster.

**Idea:** Relax by allowing the demand of an item of smaller width be met by an item of larger width.

Suppose the demands for $w_1$ and $w_2$ are $b_1$ and $b_2$ respectively ($w_1 \geq w_2$). We relax this requirement by saying that we require $b_1$ number of items of width $w_1$ and together $b_1 + b_2$ number of items of width either $w_2$ or $w_1$. 
Lemma 1 The relaxation in the MP is equivalent to adding the constraint $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$ in its dual.
Lemma 1  The relaxation in the MP is equivalent to adding the constraint \( \pi_1 \geq \pi_2 \geq \cdots \geq \pi_n \) in its dual.

Proof

- The constraints in the master LP are as follows

\[
\begin{align*}
A_1 \lambda & \geq b_1 \\
A_2 \lambda & \geq b_2 \\
& \quad \ldots \\
A_n \lambda & \geq b_n
\end{align*}
\]
Lemma 1 The relaxation in the MP is equivalent to adding the constraint $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$ in its dual.

Proof

- The constraints in the master LP are as follows
  
  \[ A_1 \lambda \geq b_1 \]
  
  \[ A_2 \lambda \geq b_2 \]
  
  \[ \cdots \]
  
  \[ A_n \lambda \geq b_n \]

- Applying the relaxation means
  
  \[ A_1 \lambda' \geq b_1 \]
  
  \[(A_1 + A_2)\lambda' \geq b_1 + b_2 \]
  
  \[ \cdots \]
  
  \[(A_1 + A_2 + \cdots + A_n)\lambda' \geq b_1 + b_2 + \cdots + b_n \]
GoodPrice: Proof of correctness (II)

- It is equivalent to multiplying both sides by $L$ where

\[
L = \begin{bmatrix}
1 & 0 & \ldots & 0 \\
1 & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
1 & 1 & \ldots & 1
\end{bmatrix}
\]

Thus, the changed master LP is

\[
\min 1^T \lambda' \\
s.t. \quad L\lambda' \geq Lb' \\
\geq 0
\]

The corresponding dual is

\[
\max b^T L^T \pi' \\
s.t. \quad A^T L^T \pi' \leq 1 \\
\pi' \geq 0
\]
GoodPrice: Proof of correctness (II)

- It is equivalent to multiplying both sides by $L$ where

$$L = \begin{bmatrix}
1 & 0 & \ldots & 0 \\
1 & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
1 & 1 & \ldots & 1
\end{bmatrix}$$

- Thus, the changed master LP is

$$\begin{align*}
\min & \quad \mathbf{1}^T \lambda' \\
\text{s.t.} & \quad LA\lambda' \geq Lb \\
& \quad \lambda' \geq 0
\end{align*}$$
GoodPrice: Proof of correctness (II)

- It is equivalent to multiplying both sides by $L$ where

$$L = \begin{bmatrix}
1 & 0 & \ldots & 0 \\
1 & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
1 & 1 & \ldots & 1
\end{bmatrix}$$

- Thus, the changed master LP is

$$\min \ 1^T \lambda' \\
\text{s.t.} \quad L A \lambda' \geq L b \\
\lambda' \geq 0$$

- The corresponding dual is

$$\max \ b^T L^T \pi' \\
\text{s.t.} \quad A^T L^T \pi' \leq 1 \\
\pi' \geq 0$$
If we substitute $L^T \pi'$ by $\pi$, i.e. $\pi' = (L^T)^{-1} \pi$

$$\begin{align*}
\text{max} & \quad b^T \pi \\
\text{s.t.} & \quad A^T \pi \leq 1 \\
& \quad (L^T)^{-1} \pi \geq 0
\end{align*}$$
GoodPrice: Proof of correctness (III)

- If we substitute $L^T \pi'$ by $\pi$, i.e. $\pi' = (L^T)^{-1} \pi$

$$\begin{align*}
\max & \quad b^T \pi \\
\text{s.t.} & \quad A^T \pi \leq 1 \\
& \quad (L^T)^{-1} \pi \geq 0
\end{align*}$$

- However, $(L^T)^{-1}$ is given by

$$(L^T)^{-1} = \begin{bmatrix}
1 & -1 & 0 & \ldots & 0 \\
0 & 1 & -1 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 1 & -1 \\
0 & 0 & \ldots & 0 & 1
\end{bmatrix}$$
If we substitute $L^T \pi'$ by $\pi$, i.e. $\pi' = (L^T)^{-1} \pi$

$$\max \ b^T \pi$$

s.t. $A^T \pi \leq 1$

$$(L^T)^{-1} \pi \geq 0$$

However, $(L^T)^{-1}$ is given by

$$(L^T)^{-1} = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & \cdots & 0 \\ \hdots & \hdots & \hdots & \hdots & \hdots \\ 0 & 0 & \cdots & 1 & -1 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

The extra constraint $(L^T)^{-1} \pi \geq 0$ implies

$$\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n \geq 0$$
Lemma 2 The dual solution to the original formulation implicitly satisfies the constraints $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$
Lemma 2 The dual solution to the original formulation implicitly satisfies the constraints $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$

Proof We prove by contradiction.

► Suppose that the solution to the original formulation has two dual variables $\pi_l$ and $\pi_m$ such that $\pi_l < \pi_m$ and $w_l \geq w_m$. 
Lemma 2 The dual solution to the original formulation implicitly satisfies the constraints $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$

Proof We prove by contradiction.

- Suppose that the solution to the original formulation has two dual variables $\pi_l$ and $\pi_m$ such that $\pi_l < \pi_m$ and $w_l \geq w_m$.
- Suppose, in the corresponding primal solution, $\lambda_j > 0$, where the pattern $j$ contains the item $l$. Let the column for the pattern $j$ be given by $[a_1 \ldots a_l \ldots a_m \ldots a_n]^T$. 
Lemma 2 The dual solution to the original formulation implicitly satisfies the constraints $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_n$

Proof We prove by contradiction.

- Suppose that the solution to the original formulation has two dual variables $\pi_l$ and $\pi_m$ such that $\pi_l < \pi_m$ and $w_l \geq w_m$.
- Suppose, in the corresponding primal solution, $\lambda_j > 0$, where the pattern $j$ contains the item $l$. Let the column for the pattern $j$ be given by $[a_1 \ldots a_l \ldots a_m \ldots a_n]^T$.
- Using complementary slackness, in the dual, the corresponding constraint is tight. i.e.

$$a_1\pi_1 + \ldots + a_l\pi_l + \ldots + a_m\pi_m + \ldots + a_n\pi_n = 1$$
GoodPrice: Proof of correctness (V)

- However, since, $w_l \geq w_m$ if we replace each instance of item $l$ in pattern $j$ by an instance of item $m$, we get a new valid pattern.
GoodPrice: Proof of correctness (V)

- However, since, $w_i \geq w_m$ if we replace each instance of item $i$ in pattern $j$ by an instance of item $m$, we get a new valid pattern.
- The column for the new pattern is given by $[a_1 \ldots 0 \ldots a_i + a_m \ldots a_n]^T$. 

That means, the constraint in the dual is violated. That is not possible. Hence proved.

Combining Lemma 1 and Lemma 2, gives proof of correctness of GoodPrice.
GoodPrice: Proof of correctness (V)

- However, since, \( w_l \geq w_m \) if we replace each instance of item \( l \) in pattern \( j \) by an instance of item \( m \), we get a new valid pattern.

- The column for the new pattern is given by \[ [a_1 \ldots 0 \ldots a_l + a_m \ldots a_n]^T. \]

- The left hand of the corresponding constraint in the dual is

\[
 a_1 \pi_1 + \ldots + 0 \pi_l + \ldots + (a_l + a_m) \pi_m \ldots + a_n \pi_n \\
> a_1 \pi_1 + \ldots + a_l \pi_l + \ldots + a_m \pi_m \ldots + a_n \pi_n \quad \therefore \pi_m > \pi_l \\
= 1
\]
GoodPrice: Proof of correctness (V)

- However, since, \( w_l \geq w_m \) if we replace each instance of item \( l \) in pattern \( j \) by an instance of item \( m \), we get a new valid pattern.

- The column for the new pattern is given by
  \[
  [a_1 \ldots 0 \ldots a_l + a_m \ldots a_n]^T.
  \]

- The left hand of the corresponding constraint in the dual is
  \[
  a_1 \pi_1 + \ldots + 0 \pi_l + \ldots + (a_l + a_m) \pi_m \ldots + a_n \pi_n
  \]
  \[
  > a_1 \pi_1 + \ldots + a_l \pi_l + \ldots + a_m \pi_m \ldots + a_n \pi_n \quad \because \pi_m > \pi_l
  \]
  \[
  = 1
  \]

- That means, the constraint in the dual is violated. That is not possible. Hence proved.
GoodPrice: Proof of correctness (V)

- However, since, \( w_l \geq w_m \) if we replace each instance of item \( l \) in pattern \( j \) by an instance of item \( m \), we get a new valid pattern.
- The column for the new pattern is given by
  \[
  [a_1 \; \ldots \; 0 \; \ldots \; a_l + a_m \; \ldots \; a_n]^T.
  \]
- The left hand of the corresponding constraint in the dual is
  \[
  a_1\pi_1 + \ldots + 0\pi_l + \ldots + (a_l + a_m)\pi_m \ldots + a_n\pi_n
  \]
  \[
  > a_1\pi_1 + \ldots + a_l\pi_l + \ldots + a_m\pi_m \ldots + a_n\pi_n \quad \therefore \pi_m > \pi_l
  \]
  \[
  = 1
  \]
- That means, the constraint in the dual is violated. That is not possible. Hence proved.
- Combining Lemma 1 and Lemma 2, gives proof of correctness of GoodPrice.
Experimental result (I)

- Implementation using COIN-OR framework

<table>
<thead>
<tr>
<th>Testcase set</th>
<th>Heuristic</th>
<th>Num</th>
<th>Gain</th>
<th>Solver</th>
<th>Gain</th>
<th>Master</th>
<th>Gain</th>
<th>Time</th>
<th>ST %</th>
<th>Time</th>
<th>PT %</th>
<th>Time</th>
<th>KT %</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic</td>
<td>163.00</td>
<td>0.00</td>
<td>35.32</td>
<td>0.00</td>
<td>0.24</td>
<td>0.00</td>
<td>35.08</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodp</td>
<td>67.00</td>
<td>58.90</td>
<td>25.91</td>
<td>26.66</td>
<td>0.11</td>
<td>53.33</td>
<td>25.79</td>
<td>26.48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>basic</td>
<td>160.00</td>
<td>0.00</td>
<td>47.74</td>
<td>0.00</td>
<td>0.28</td>
<td>0.00</td>
<td>47.46</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodp</td>
<td>65.00</td>
<td>59.38</td>
<td>36.04</td>
<td>24.52</td>
<td>0.10</td>
<td>64.79</td>
<td>35.94</td>
<td>24.28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>basic</td>
<td>144.00</td>
<td>0.00</td>
<td>30.87</td>
<td>0.00</td>
<td>0.19</td>
<td>0.00</td>
<td>30.68</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodp</td>
<td>62.00</td>
<td>56.94</td>
<td>39.84</td>
<td>-29.07</td>
<td>0.08</td>
<td>56.25</td>
<td>39.76</td>
<td>-29.60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>basic</td>
<td>119.00</td>
<td>0.00</td>
<td>39.07</td>
<td>0.00</td>
<td>0.10</td>
<td>0.00</td>
<td>38.98</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodp</td>
<td>61.00</td>
<td>48.74</td>
<td>36.41</td>
<td>6.82</td>
<td>0.06</td>
<td>37.50</td>
<td>36.35</td>
<td>6.74</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>basic</td>
<td>158.00</td>
<td>0.00</td>
<td>26.08</td>
<td>0.00</td>
<td>0.26</td>
<td>0.00</td>
<td>25.82</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodp</td>
<td>70.00</td>
<td>55.70</td>
<td>23.47</td>
<td>10.00</td>
<td>0.11</td>
<td>57.58</td>
<td>23.36</td>
<td>9.51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>basic</td>
<td>163.00</td>
<td>0.00</td>
<td>48.52</td>
<td>0.00</td>
<td>0.29</td>
<td>0.00</td>
<td>48.22</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodp</td>
<td>70.00</td>
<td>57.06</td>
<td>60.81</td>
<td>-25.34</td>
<td>0.12</td>
<td>60.27</td>
<td>60.69</td>
<td>-25.86</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>basic</td>
<td>142.00</td>
<td>0.00</td>
<td>42.06</td>
<td>0.00</td>
<td>0.19</td>
<td>0.00</td>
<td>41.87</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodp</td>
<td>71.00</td>
<td>50.00</td>
<td>49.15</td>
<td>-16.85</td>
<td>0.10</td>
<td>48.93</td>
<td>49.06</td>
<td>-17.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>basic</td>
<td>133.00</td>
<td>0.00</td>
<td>24.27</td>
<td>0.00</td>
<td>0.16</td>
<td>0.00</td>
<td>24.11</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodp</td>
<td>64.00</td>
<td>51.88</td>
<td>44.24</td>
<td>-82.28</td>
<td>0.10</td>
<td>40.00</td>
<td>44.14</td>
<td>-83.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>basic</td>
<td>164.00</td>
<td>0.00</td>
<td>27.21</td>
<td>0.00</td>
<td>0.30</td>
<td>0.00</td>
<td>26.91</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodp</td>
<td>69.00</td>
<td>57.93</td>
<td>23.13</td>
<td>15.02</td>
<td>0.12</td>
<td>60.00</td>
<td>23.01</td>
<td>14.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>basic</td>
<td>161.00</td>
<td>0.00</td>
<td>43.71</td>
<td>0.00</td>
<td>0.24</td>
<td>0.00</td>
<td>43.47</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodp</td>
<td>63.00</td>
<td>60.87</td>
<td>49.88</td>
<td>-14.11</td>
<td>0.10</td>
<td>57.38</td>
<td>49.78</td>
<td>-14.51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average: 55.74 -8.46 53.6 -8.87
Experimental result (I)

- Implementation using COIN-OR framework
- Testcase set 1: Easier examples

<table>
<thead>
<tr>
<th>Test No</th>
<th>Heuristic</th>
<th>Num Iteration</th>
<th>Gain NT %</th>
<th>Solver Time</th>
<th>Gain ST %</th>
<th>Master Time</th>
<th>Gain PT %</th>
<th>Knapsack Time</th>
<th>Gain KT %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic</td>
<td>163.00</td>
<td>0.00</td>
<td>35.32</td>
<td>0.00</td>
<td>0.24</td>
<td>0.00</td>
<td>35.08</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>goodp</td>
<td>67.00</td>
<td>58.90</td>
<td>25.91</td>
<td>26.66</td>
<td>0.11</td>
<td>53.33</td>
<td>25.79</td>
<td>26.48</td>
</tr>
<tr>
<td>2</td>
<td>basic</td>
<td>160.00</td>
<td>0.00</td>
<td>47.74</td>
<td>0.00</td>
<td>0.28</td>
<td>0.00</td>
<td>47.46</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>goodp</td>
<td>65.00</td>
<td>59.38</td>
<td>36.04</td>
<td>24.52</td>
<td>0.10</td>
<td>64.79</td>
<td>35.94</td>
<td>24.28</td>
</tr>
<tr>
<td>3</td>
<td>basic</td>
<td>144.00</td>
<td>0.00</td>
<td>30.87</td>
<td>0.00</td>
<td>0.19</td>
<td>0.00</td>
<td>30.68</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>goodp</td>
<td>62.00</td>
<td>56.94</td>
<td>39.84</td>
<td>-29.07</td>
<td>0.08</td>
<td>56.25</td>
<td>39.76</td>
<td>-29.60</td>
</tr>
<tr>
<td>4</td>
<td>basic</td>
<td>119.00</td>
<td>0.00</td>
<td>39.07</td>
<td>0.00</td>
<td>0.10</td>
<td>0.00</td>
<td>38.98</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>goodp</td>
<td>61.00</td>
<td>48.74</td>
<td>36.41</td>
<td>6.82</td>
<td>0.06</td>
<td>37.50</td>
<td>36.35</td>
<td>6.74</td>
</tr>
<tr>
<td>5</td>
<td>basic</td>
<td>158.00</td>
<td>0.00</td>
<td>26.08</td>
<td>0.00</td>
<td>0.26</td>
<td>0.00</td>
<td>25.82</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>goodp</td>
<td>70.00</td>
<td>55.70</td>
<td>23.47</td>
<td>10.00</td>
<td>0.11</td>
<td>57.58</td>
<td>23.36</td>
<td>9.51</td>
</tr>
<tr>
<td>6</td>
<td>basic</td>
<td>163.00</td>
<td>0.00</td>
<td>48.52</td>
<td>0.00</td>
<td>0.29</td>
<td>0.00</td>
<td>48.22</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>goodp</td>
<td>70.00</td>
<td>57.06</td>
<td>60.81</td>
<td>-25.34</td>
<td>0.12</td>
<td>60.27</td>
<td>60.69</td>
<td>-25.86</td>
</tr>
<tr>
<td>7</td>
<td>basic</td>
<td>142.00</td>
<td>0.00</td>
<td>42.06</td>
<td>0.00</td>
<td>0.19</td>
<td>0.00</td>
<td>41.87</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>goodp</td>
<td>71.00</td>
<td>50.00</td>
<td>49.15</td>
<td>-16.85</td>
<td>0.10</td>
<td>48.93</td>
<td>49.06</td>
<td>-17.15</td>
</tr>
<tr>
<td>8</td>
<td>basic</td>
<td>133.00</td>
<td>0.00</td>
<td>24.27</td>
<td>0.00</td>
<td>0.16</td>
<td>0.00</td>
<td>24.11</td>
<td>0.00</td>
</tr>
<tr>
<td>8</td>
<td>goodp</td>
<td>64.00</td>
<td>51.88</td>
<td>44.24</td>
<td>-82.28</td>
<td>0.10</td>
<td>40.00</td>
<td>44.14</td>
<td>-83.09</td>
</tr>
<tr>
<td>9</td>
<td>basic</td>
<td>164.00</td>
<td>0.00</td>
<td>27.21</td>
<td>0.00</td>
<td>0.30</td>
<td>0.00</td>
<td>26.91</td>
<td>0.00</td>
</tr>
<tr>
<td>9</td>
<td>goodp</td>
<td>69.00</td>
<td>57.93</td>
<td>23.13</td>
<td>15.02</td>
<td>0.12</td>
<td>60.00</td>
<td>23.01</td>
<td>14.52</td>
</tr>
<tr>
<td>10</td>
<td>basic</td>
<td>161.00</td>
<td>0.00</td>
<td>43.71</td>
<td>0.00</td>
<td>0.24</td>
<td>0.00</td>
<td>43.47</td>
<td>0.00</td>
</tr>
<tr>
<td>10</td>
<td>goodp</td>
<td>63.00</td>
<td>60.87</td>
<td>49.88</td>
<td>-14.11</td>
<td>0.10</td>
<td>57.38</td>
<td>49.78</td>
<td>-14.51</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>55.74</td>
<td>-8.46</td>
<td>53.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-8.87</td>
</tr>
</tbody>
</table>
Experimental result (II)

- Testcase set 2: Hard examples

<table>
<thead>
<tr>
<th>Test No</th>
<th>Heuristic</th>
<th>Num Iteration</th>
<th>Gain NT %</th>
<th>Solv Time</th>
<th>Gain ST %</th>
<th>Master Time</th>
<th>Gain PT %</th>
<th>Knapsack Time</th>
<th>Gain KT %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>basic</td>
<td>649</td>
<td>0.00</td>
<td>845.89645</td>
<td>0.00</td>
<td>7.22445</td>
<td>0.00</td>
<td>838.67200</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>goodp</td>
<td>428</td>
<td>34.05</td>
<td>717.66129</td>
<td>15.16</td>
<td>4.82829</td>
<td>33.17</td>
<td>712.83300</td>
<td>15.00</td>
</tr>
<tr>
<td>2</td>
<td>basic</td>
<td>721</td>
<td>0.00</td>
<td>2023.45445</td>
<td>0.00</td>
<td>7.08445</td>
<td>0.00</td>
<td>2016.37000</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>basic</td>
<td>587</td>
<td>0.00</td>
<td>1236.87830</td>
<td>0.00</td>
<td>4.92830</td>
<td>0.00</td>
<td>1231.95000</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>goodp</td>
<td>402</td>
<td>31.52</td>
<td>1017.40823</td>
<td>17.74</td>
<td>3.52823</td>
<td>28.41</td>
<td>1013.88000</td>
<td>17.70</td>
</tr>
<tr>
<td>4</td>
<td>basic</td>
<td>558</td>
<td>0.00</td>
<td>1516.71623</td>
<td>0.00</td>
<td>3.61623</td>
<td>0.00</td>
<td>1513.10000</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>goodp</td>
<td>325</td>
<td>41.76</td>
<td>1416.91014</td>
<td>6.58</td>
<td>2.08014</td>
<td>42.48</td>
<td>1414.83000</td>
<td>6.49</td>
</tr>
<tr>
<td>5</td>
<td>basic</td>
<td>510</td>
<td>0.00</td>
<td>1464.36618</td>
<td>0.00</td>
<td>2.85618</td>
<td>0.00</td>
<td>1461.51000</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>goodp</td>
<td>283</td>
<td>44.51</td>
<td>1193.44008</td>
<td>18.50</td>
<td>1.30008</td>
<td>54.48</td>
<td>1192.14000</td>
<td>18.43</td>
</tr>
<tr>
<td>6</td>
<td>basic</td>
<td>538</td>
<td>0.00</td>
<td>1396.68420</td>
<td>0.00</td>
<td>3.50420</td>
<td>0.00</td>
<td>1393.18000</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>goodp</td>
<td>340</td>
<td>36.80</td>
<td>1462.53213</td>
<td>-4.71</td>
<td>1.99212</td>
<td>43.15</td>
<td>1460.54000</td>
<td>-4.83</td>
</tr>
<tr>
<td>7</td>
<td>basic</td>
<td>762</td>
<td>0.00</td>
<td>2105.62447</td>
<td>0.00</td>
<td>7.52447</td>
<td>0.00</td>
<td>2098.10000</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>goodp</td>
<td>435</td>
<td>42.91</td>
<td>1265.77022</td>
<td>39.89</td>
<td>3.40022</td>
<td>54.81</td>
<td>1262.37000</td>
<td>39.83</td>
</tr>
<tr>
<td>8</td>
<td>basic</td>
<td>415</td>
<td>0.00</td>
<td>131.56416</td>
<td>0.00</td>
<td>2.40416</td>
<td>0.00</td>
<td>129.16000</td>
<td>0.00</td>
</tr>
<tr>
<td>8</td>
<td>goodp</td>
<td>158</td>
<td>61.93</td>
<td>139.09304</td>
<td>-5.72</td>
<td>0.76404</td>
<td>68.22</td>
<td>138.32900</td>
<td>-7.10</td>
</tr>
<tr>
<td>9</td>
<td>basic</td>
<td>525</td>
<td>0.00</td>
<td>1030.76818</td>
<td>0.00</td>
<td>2.82818</td>
<td>0.00</td>
<td>1027.94000</td>
<td>0.00</td>
</tr>
<tr>
<td>9</td>
<td>goodp</td>
<td>383</td>
<td>27.05</td>
<td>1091.64613</td>
<td>-5.91</td>
<td>1.89613</td>
<td>32.96</td>
<td>1089.75000</td>
<td>-6.01</td>
</tr>
<tr>
<td>10</td>
<td>basic</td>
<td>466</td>
<td>0.00</td>
<td>1275.49813</td>
<td>0.00</td>
<td>1.94813</td>
<td>0.00</td>
<td>1273.55000</td>
<td>0.00</td>
</tr>
<tr>
<td>10</td>
<td>goodp</td>
<td>232</td>
<td>50.21</td>
<td>1175.82606</td>
<td>7.81</td>
<td>0.91606</td>
<td>52.98</td>
<td>1174.91000</td>
<td>7.75</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>40.11</td>
<td>9.14</td>
<td>44.77</td>
<td></td>
<td></td>
<td></td>
<td>8.92</td>
<td></td>
</tr>
</tbody>
</table>
Experimental result (III)

- Number of iteration is reduced by more than 40%

Total solver time has not improved much. This is because the knapsack problem solved in the subproblem becomes harder to solve. However, we need to see this in the context of complete solution. We are hopeful that this will be useful because most of the time spent in solving knapsack problem is at the end of column generation. We do not need to solve the LP to optimality always. This also indicates that we need to look for better method for solving the subproblem.
Experimental result (III)

- Number of iteration is reduced by more than 40%
- Total solver time has not improved much. This is because the knapsack problem solved in the subproblem becomes harder to solve

However, we need to see this in the context of complete solution. We are hopeful that this will be useful because most of the time spent in solving knapsack problem is at the fag end of column generation. We do not need to solve the LP to optimality always.

This also indicates that we need to look for better method for solving the subproblem.
Experimental result (III)

- Number of iteration is reduced by more than 40%
- Total solver time has not improved much. This is because the knapsack problem solved in the subproblem becomes harder to solve
- However, we need to see this in the context of complete solution. We are hopeful that this will be useful because most of the time spent in solving knapsack problem is at the far end of column generation. We do not need to solve the LP to optimality always.

This also indicates that we need to look for better method for solving the subproblem.
Experimental result (III)

- Number of iteration is reduced by more than 40%
- Total solver time has not improved much. This is because the knapsack problem solved in the subproblem becomes harder to solve.
- However, we need to see this in the context of complete solution. We are hopeful that this will be useful because most of the time spent in solving knapsack problem is at the end of column generation. We do not need to solve the LP to optimality always.
- This also indicates that we need to look for better method for solving the subproblem.
Dynamic prog solution to knapsack with forbidden list

\[
\text{KPFL}(i, w, p, W, L, S) = \max \begin{cases}
\text{KPFL}(i - 1, w, p, W - w[i], \{\text{patterns in } L \text{ having } i\}) \cup S,
\text{KPFL}(i - 1, w, p, W, \{\text{patterns in } L \text{ not having } i\}) \cup S
\end{cases}
\]

Worstcase time complexity

Parameters that vary \(i, W\) and \(L\). However, for the particular values of \(i\) and \(W\) there can be at most 2 possible values of \(L\).

No of table entries accessed \(O(2^n W)\)

Each access takes time \(O(L)\). Time = \(O(2^n WL)\)

Worstcase time complexity of naive B&B algorithm

The tree size \(O(2^n)\).

At each level the list of \(L\) forbidden columns is distributed among the nodes, the list processing time summed up for a level is \(O(L)\).

There are \(O(n)\) levels. Time = \(O(2^n n + nL)\)
Dynamic prog solution to knapsack with forbidden list

\[ \text{KPFL}(i, w, p, W, L, S) = \]

\[ \max \quad \{ \text{KPFL}(i - 1, w, p, W - w[i], \{\text{patterns in } L \text{ having } i\}, S \cup \{i\}) \]
\[ \text{KPFL}(i - 1, w, p, W, \{\text{patterns in } L \text{ not having } i\}, S) \]
Dynamic prog solution to knapsack with forbidden list

$\text{KPFL}(i, w, p, W, L, S) =$

$$\max \ \{ \text{KPFL}(i - 1, w, p, W - w[i], \{\text{patterns in } L \text{ having } i\}, S \cup \{i\}) \quad \text{KPFL}(i - 1, w, p, W, \{\text{patterns in } L \text{ not having } i\}, S) \}$$

- **Worstcase time complexity**
  - Parameters that vary $i$, $W$ and $L$. However, for the particular values of $i$ and $W$ there can be at most 2 possible values of $L$.
  - No of table entries accessed $O(2nW)$
  - Each access takes time $O(L)$. Time $= O(2nWL)$
Dynamic prog solution to knapsack with forbidden list

\[
KPFL(i, w, p, W, L, S) = \\
\max \{ \begin{align*}
& KPFL(i - 1, w, p, W - w[i], \{\text{patterns in } L \text{ having } i\}, S \cup \{i\}) \\
& KPFL(i - 1, w, p, W, \{\text{patterns in } L \text{ not having } i\}, S) 
\end{align*} \}
\]

- **Worstcase time complexity**
  - Parameters that vary \( i, W \) and \( L \). However, for the particular values of \( i \) and \( W \) there can be at most 2 possible values of \( L \).
  - No of table entries accessed \( O(2nW) \)
  - Each access takes time \( O(L) \). Time = \( O(2nWL) \)

- **Worstcase time complexity of naive B&B algorithm**
  - The tree size \( O(2^n) \).
  - At each level the list of \( L \) forbidden columns is distributed among the nodes, the list processing time summed up for a level is \( O(L) \)
  - There are \( O(n) \) levels. Time = \( O(2^n + nL) \)
Core algorithms

Item widths and profits are drawn at random from \([0,1]\)

Break item: greedy algorithm stops at this item

Break solution: greedy solution without taking break item

Dantzig ray: position of ray at the break item

Core items: items that are included in either an integer solution or greedy solution but not both

Core for opt int solution must be within \(\Gamma\) from Dantzig ray

\(\Gamma = O\left(\log_2 N\right)\) whp
Core algorithms

- Item widths and profits are drawn at random from [0,1]
Core algorithms

- Item widths and profits are drawn at random from $[0,1]$
- Break item: greedy algorithm stops at this item
- Dantzig ray: position of ray at the break item
- Core items: items that are included in either an integer solution or greedy solution but not both
- Core for opt int solution must be within $\Gamma$ from Dantzig ray
  $\Gamma = O\left(\log \frac{1}{N}\right)$ whp
Core algorithms

- Item widths and profits are drawn at random from $[0, 1]$
- Break item: greedy algorithm stops at this item
- Break solution: greedy solution without taking break item

![Diagram showing item selection and break points](image)
Core algorithms

- Item widths and profits are drawn at random from $[0,1]$
- Break item: greedy algorithm stops at this item
- Break solution: greedy solution without taking break item
- Dantzig ray: position of ray at the break item
Core algorithms

- Item widths and profits are drawn at random from \([0,1]\).
- Break item: greedy algorithm stops at this item.
- Break solution: greedy solution without taking break item.
- Dantzig ray: position of ray at the break item.
- Core items: items that are included in either an integer solution or greedy solution but not both.
Core algorithms

- Item widths and profits are drawn at random from \([0,1]\).
- Break item: greedy algorithm stops at this item.
- Break solution: greedy solution without taking break item.
- Dantzig ray: position of ray at the break item.
- Core items: items that are included in either an integer solution or greedy solution but not both.
- Core for opt int solution must be within \(\Gamma\) from Dantzig ray.

\(\Gamma = O(\log_2 N)\) whp.
Core algorithms

- Item widths and profits are drawn at random from [0,1]
- Break item: greedy algorithm stops at this item
- Break solution: greedy solution without taking break item
- Dantzig ray: position of ray at the break item
- Core items: items that are included in either an integer solution or greedy solution but not both
- Core for opt int solution must be within $\Gamma$ from Dantzig ray
- $\Gamma = O\left(\frac{\log^2 N}{N}\right)$ whp
Core algorithms (II)

- Profit of a subset of items = sum of profits of items in it

Opt set will be from dominating subsets

Nemhauser/Ullman gave an algorithm that solves knapsack in $O(nE[q])$, $E[q]$ is the no of dominating sets
Core algorithms (II)

- Profit of a subset of items = sum of profits of items in it
- Weight of a subset of items = sum of weights of items in it
Core algorithms (II)

- Profit of a subset of items = sum of profits of items in it
- Weight of a subset of items = sum of weights of items in it
- $S \text{ dom } T$ if $W(S) \leq W(T)$, $P(S) \geq P(T)$
Core algorithms (II)

- Profit of a subset of items = sum of profits of items in it
- Weight of a subset of items = sum of weights of items in it
- $S \text{ dom } T$ if $W(S) \leq W(T), \ P(S) \geq P(T)$
- Opt set will be from dominating subsets

Nemhauser/Ullman gave an algorithm that solves knapsack in $O(n E[q])$, $E[q]$ is the no of dominating sets
Core algorithms (II)

- Profit of a subset of items = sum of profits of items in it
- Weight of a subset of items = sum of weights of items in it
- $S \text{ dom } T$ if $W(S) \leq W(T)$, $P(S) \geq P(T)$
- Opt set will be from dominating subsets
- Nemhauser/Ullman gave an algorithm that solves knapsack in $O(nE[q])$, $E[q]$ is the no of dominating sets
Core algorithms (III)

- [Beier and Vöcking (2004)] combined all these
  1. Generate a static core with loss at most \( d = c_d N^{-1} \log^3 N \), with a suitable constant \( c_d \).
  2. Use Nemhauser/Ullman algorithm to generate the dominating sets over these core items.
  3. Among all the dominating sets satisfying the capacity bound, the most profitable one is selected.

\[ \text{Expected number of core items} = N \times \text{area covered by the } d \text{-region around the Dantzig ray} \approx 2 d N. \]

\[ \text{Expected run time} = O(N^2 d N) = O(N^2 c_d N^{-1} \log^3 N) = O(N \text{polylog } N). \]
Core algorithms (III)

- [Beier and Vöcking(2004)] combined all these
  1. Generate a static core with loss at most $d = c_d N^{-1} \log^3 N$, with a suitable constant $c_d$.
  2. Use Nemhauser/Ullman algorithm to generate the dominating sets over these core items.
  3. Among all the dominating sets satisfying the capacity bound, the most profitable one is selected.

- The expected number of the core items corresponds to $N$ times the area covered by the $d$-region around the Dantzig ray $\approx 2dN$. 

Expected run time $= O(N^2 d N) = O(N^{polylog N})$. 

Core algorithms (III)

- [Beier and Vöcking(2004)] combined all these
  1. Generate a static core with loss at most \( d = c_d N^{-1} \log^3 N \), with a suitable constant \( c_d \).
  2. Use Nemhauser/Ullman algorithm to generate the dominating sets over these core items.
  3. Among all the dominating sets satisfying the capacity bound, the most profitable one is selected.

- The expected number of the core items corresponds to \( N \) times the area covered by the \( d \)-region around the Dantzig ray \( \approx 2dN \).

- Expected run time = 
  \( O(N \cdot 2dN) = O(N^2 c_d N^{-1} \log^3 N) = O(N \text{polylog} N) \).
Proof using the isolating lemma

Lemma
Let $A = \{a_1, a_2, \ldots, a_m\}$ be a set of elements and let $S = \{S_1, S_2, \ldots, S_k\}$ be a collection of subsets of $A$. If we choose integer weights $w_1, w_2, \ldots, w_m$ to the elements of $A$ at random from the domain $\{1, \ldots, n\}$, then irrespective of the subsets in $S$, $\Pr(\text{the maximal weight subset in } S \text{ is not unique}) \leq \frac{m}{n}$. 
Proof using the isolating lemma

Lemma
Let \( A = \{a_1, a_2, \ldots, a_m\} \) be a set of elements and let \( S = \{S_1, S_2, \ldots, S_k\} \) be a collection of subsets of \( A \). If we choose integer weights \( w_1, w_2, \ldots, w_m \) to the elements of \( A \) at random from the domain \( \{1, \ldots, n\} \), then irrespective of the subsets in \( S \), \( \Pr(\text{the maximal weight subset in } S \text{ is not unique}) \leq \frac{m}{n} \).

- The dynamic programming algorithm used in the polynomial time approximation scheme runs in time polynomial in \( n \) and max profit \( V \).
Proof using the isolating lemma

Lemma
Let \( A = \{a_1, a_2, \ldots, a_m\} \) be a set of elements and let \( S = \{S_1, S_2, \ldots, S_k\} \) be a collection of subsets of \( A \). If we choose integer weights \( w_1, w_2, \ldots, w_m \) to the elements of \( A \) at random from the domain \( \{1, \ldots, n\} \), then irrespective of the subsets in \( S \), \( \Pr(\text{the maximal weight subset in } S \text{ is not unique}) \leq \frac{m}{n} \).

- The dynamic programming algorithm used in the polynomial time approximation scheme runs in time polynomial in \( n \) and max profit \( V \).
- The algorithm is polynomial time if \( V \) is restricted to be \( O(\log n) \) bits long.
Proof using the isolating lemma

**Lemma**

Let $A = \{a_1, a_2, \ldots, a_m\}$ be a set of elements and let $S = \{S_1, S_2, \ldots, S_k\}$ be a collection of subsets of $A$. If we choose integer weights $w_1, w_2, \ldots, w_m$ to the elements of $A$ at random from the domain $\{1, \ldots, n\}$, then irrespective of the subsets in $S$, \( \Pr(\text{the maximal weight subset in } S \text{ is not unique}) \leq \frac{m}{n} \).

- The dynamic programming algorithm used in the polynomial time approximation scheme runs in time polynomial in $n$ and $\max \text{ profit } V$.
- The algorithm is polynomial time if $V$ is restricted to be $O(\log n)$ bits long.
- If we consider the $(k + 1) \log n$ MSBs of profits, $\Pr(\text{maximal weight subset in } S \text{ is not unique}) \leq \frac{n}{n^{k+1}} = n^{-k}$, where the family of subsets $S$ are sets with only one element.
Other branching techniques

- We explored the two different branching scheme. One by [Vance(1998)] and other by [Vanderbeck(1999)].

\[
\sum_{q \in \hat{Q}} \lambda_q \geq \lceil \beta \rceil \quad \text{and} \quad \sum_{q \in \hat{Q}} \lambda_q \leq \lfloor \beta \rfloor
\]

New variables and constraints are added in the subproblem; problem is solved as a general IP. [Vance(1998)] solves the special case when only maximal columns are used. In that case, the subproblem is solved using the Horowitz-Sahni algorithm for solving the knapsack problems.
Other branching techniques

▶ We explored the two different branching scheme. One by [Vance(1998)] and other by [Vanderbeck(1999)].
▶ The key idea they used is to branch on a set of columns

\[
\sum_{q \in \hat{Q}} \lambda_q \geq \lceil \beta \rceil \quad \text{and} \quad \sum_{q \in \hat{Q}} \lambda_q \leq \lfloor \beta \rfloor
\]  

(1)
Other branching techniques

- We explored the two different branching scheme. One by [Vance(1998)] and other by [Vanderbeck(1999)].
- The key idea they used is to branch on a set of columns

\[
\sum_{q \in \hat{Q}} \lambda_q \geq \lceil \beta \rceil \text{ and } \sum_{q \in \hat{Q}} \lambda_q \leq \lfloor \beta \rfloor
\]  

(1)

- New variables and constraints are added in the subproblem; problem is solved as a general IP.
Other branching techniques

- We explored the two different branching scheme. One by [Vance(1998)] and other by [Vanderbeck(1999)].
- The key idea they used is to branch on a set of columns
  \[
  \sum_{q \in \hat{Q}} \lambda_q \geq \lceil \beta \rceil \quad \text{and} \quad \sum_{q \in \hat{Q}} \lambda_q \leq \lfloor \beta \rfloor
  \] (1)
- New variables and constraints are added in the subproblem; problem is solved as a general IP.
- [Vance(1998)] solves the special case when only maximal columns are used. In that case, the subproblem is solved using the Horowitz-Sahni algorithm for solving the knapsack problems.
Work to be done in the final stage

- Work on getting the integer solution. See how much beneficial is GoodPrice.
Work to be done in the final stage

- Work on getting the integer solution. See how much beneficial is GoodPrice.
- Comparative study of the branching schemes
  - Branching on single variable, solve subproblem as knapsack with forbidden list
  - Branching on a set of variables solve subproblem as a general integer programming solution
Work to be done in the final stage

- Work on getting the integer solution. See how much beneficial is GoodPrice.

- Comparative study of the branching schemes
  - Branching on single variable, solve subproblem as knapsack with forbidden list
  - Branching on a set of variables solve subproblem as a general integer programming solution

- Find better branching scheme
  - The dynamic programmic solution to knapsack with forbidden list
  - Modification to the method using general integer programming solution to subproblem
  - Combination of both
Thanks!
R. Beier and B. Vöcking.
Probabilistic analysis of knapsack core algorithms.

P.H. Vance.
Branch-and-Price Algorithms for the One-Dimensional Cutting Stock Problem.

F. Vanderbeck.
Computational study of a column generation algorithm for bin packing and cutting stock problems.