The Integer Ray Projection Method in Column Generations Models for `Arc-Routing` and `Cutting-Stock`

Daniel Porumbel*

*Artois University (LGI2A), Lille-Nord de France

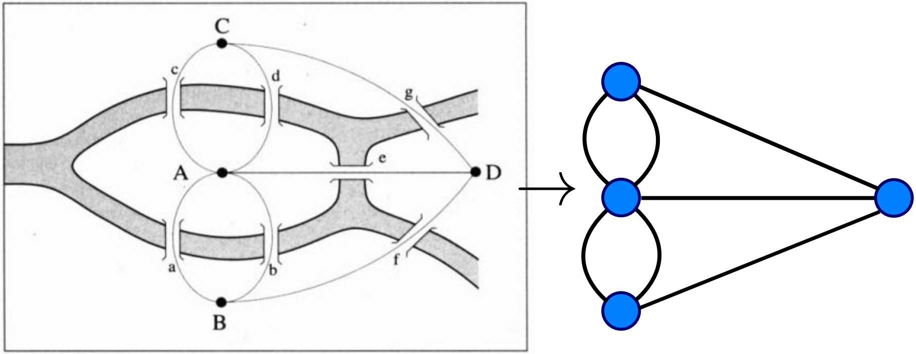
1. Set-Covering LPs: Cutting-Stock and Arc Routing
2. The Ray Approach: General Description
3. Ray SubProblem easier than Column Generation Subproblem?
4. Experiments and Conclusions

Introducing the Arc-Routing Problem

The Seven Bridges of Königsberg

This famous problem of Euler prefigured the idea of Arc-Routing

- find a walk through the city that would cross each bridge once and only once



Introducing the Arc-Routing Problem

The Seven Bridges of Königsberg

This famous problem of Euler prefigured the idea of Arc-Routing

- find a walk through the city that would cross each bridge once and only once

cross each bridge once and only once

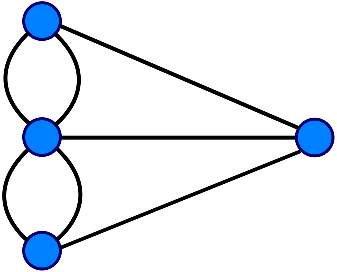
=

service each edge once but:

- never traverse it without service

or

- no "dead-heading"

→ 

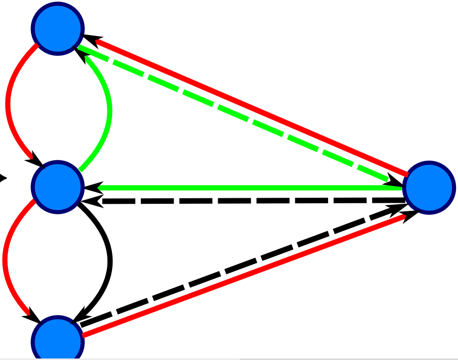
Formal Arc-Routing Definition

Find some routes of minimum total length **servicing each edge once**

capacitated Arc-Routing Assign weights w_i on edges: a feasible route has a total weight of maximum C ($\mathbf{w}^\top \mathbf{a} \leq C$)

- three routes (red, green, black) of total length 10
  - 7 serviced edges
  - 3 dead-headed edges (dashed lines)
- longest service in one route: 4

$\longrightarrow$



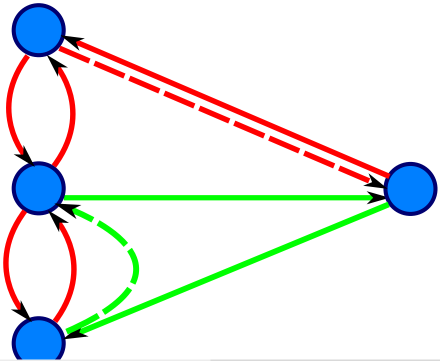
Formal Arc-Routing Definition

Find some routes of minimum total length **servicing each edge once**

capacitated Arc-Routing Assign weights w_i on edges: a feasible route has a total weight of maximum C ($\mathbf{w}^\top \mathbf{a} \leq C$)

- two routes (red, green) of total length 9
  - 7 serviced edges
  - 2 dead-headed edges (dashed lines)
- longest service in one route: 5
  - infeasible if one imposed a capacity of maximum 4

→



Applications 1

City Maintenance

Garbage Collection Garbage bins are placed on roads

Street Cleaning Costs of thousands of euros for major cities

Street Watering Capacity restrictions are very relevant

Ressources reported in in a study [(2002) Valencia]*

- An annual budget of > 100.000.000 euros
- > 1000 workers
- > 100 trucks

*[E. Benavent, Exact methods for Arc Routing Problems, Euro/Informs Congress, Rome, 2013]

Applications 2

rail link maintenance an important part in the budget of rail companies, major security interest

snow plowing Ressources reported in a [1987-1988 Indiana] study †:

- budget: $15.000.000
- 1000 vehicles
- 1140000 miles of roads and highways

meter reading savings of $874.000 reported in a paper [Wunderlich, Collette, Levy & Bodin: Scheduling Meter Readers for Southern. California Gas Company, 1992]

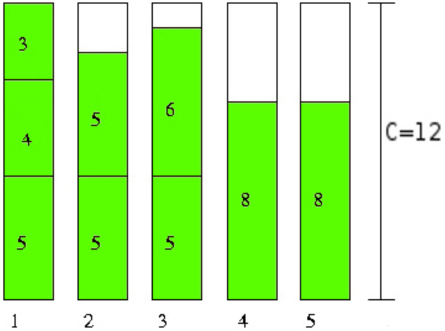
cattle feeding reported in [Dror Moshe, Livestock Feed Distribution and Arc Traversal Problems, 2000]

† [E. Benavent, Exact methods for Arc Routing Problems, Euro/Informs Congress, Rome, 2013]

Cutting-Stock: Introduction

- A fundamental problem in optimization
- Given number of (metal, paper) rolls of fixed length C
- We have n clients that each requires b_i items of length w_i

Goal: Minimize the number of rolls to produce all required items

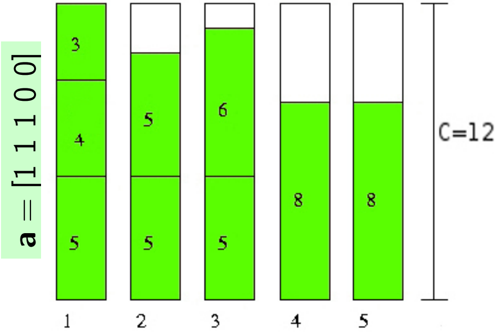


Solution example with 5 patterns

Cutting-Stock: Practical Interest

- Huge number of applications in the field of cutting and packing
- Capacitated Arc Routing can be seen as a form of Cutting-Stock if all routes have a cost of $c_a = 1$ and the patterns indicate serviced edges

$$\mathbf{w} = [3\ 4\ 5\ 6\ 8]$$
$$\mathbf{b} = [1\ 1\ 4\ 1\ 2]$$



Solution example with 5 patterns

1. Set-Covering LPs: Cutting-Stock and Arc Routing
   - Problem Defintions
   - Linear Program Modelling
2. The Ray Approach: General Description
3. Ray SubProblem easier than Column Generation Subproblem?
4. Experiments and Conclusions

Column Generation Model: the Primal

Defining columns/configurations (*or* routes *or* patterns):

Cutting-Stock	**Capacitated Arc Routing**
a_i: item i is cut a_i times	a_i: edge i is serviced a_i times
c_a: cost of the pattern a (> 1 in Elastic Cut-Stock)	c_a: total distance traversed by route a

Common Capacity constraint: $\mathbf{w}^\top \mathbf{a} \le C$, $\forall \mathbf{a} \in \mathcal{K}$

Goal: minimize total cost of selected columns

$$\min \sum_{a \in \mathcal{K}} c_a x_a$$

$\mathcal{K}$: Column set

$$\sum_{a \in \mathcal{K}} a_i x_a \ge b_i, \quad \forall i \in [1..n]$$

$$x \in \mathbb{Z}^{|\mathcal{K}|}$$

Set-covering constraints dualized to dual vector y

Column Generation Model: the Primal

Defining columns/configurations (*or* routes *or* patterns):

Cutting-Stock	Capacitated Arc Routing
a_i: item i is cut a_i times	a_i: edge i is serviced a_i times
c_a: cost of the pattern a (> 1 in Elastic Cut-Stock)	c_a: total distance traversed by route a

Common Capacity constraint: $\mathbf{w}^\top \mathbf{a} \leq C, \forall \mathbf{a} \in \mathcal{K}$

Goal: minimize total cost of selected columns

$$\min \sum_{a \in \mathcal{K}} c_a x_a$$

$\mathcal{K}$: Column set

$$\sum_{a \in \mathcal{K}} a_i x_a \geq b_i, \quad \forall i \in [1..n]$$

$$x \in \mathbb{Z}^{|\mathcal{K}|}$$

Set-covering constraints dualized to dual vector y

Set-Covering and Column Generation: the Dual

Primal→Dual

$$\left.\begin{array}{ll} & \max \mathbf{b}^\top \mathbf{y} \\ \mathbf{a}^\top \mathbf{y} \leq c_a, & \forall a \in \mathcal{K} \\ y_i \geq 0, & i \in [1..n] \end{array}\right\} \mathbf{P}$$

$$\min \sum_{a \in \mathcal{K}} c_a x_a$$

$$\sum_{a \in \mathcal{K}} a_i x_a \geq b_i, \quad \forall i \in [1..n]$$

$$x \in \mathbb{R}^{|\mathcal{K}|}$$

Set-Covering and Column Generation: the Dual

$$\max \mathbf{b}^\top \mathbf{y}$$
$$\left.\begin{array}{ll} \mathbf{a}^\top \mathbf{y} \leq c_a, & \forall a \in \mathcal{K} \\ y_i \geq 0, & i \in [1..n] \end{array}\right\} \mathbf{P}$$

Primal→Dual

$$\min \sum_{a \in \mathcal{K}} c_a x_a$$
$$\sum_{a \in \mathcal{K}} a_i x_a \geq b_i, \quad \forall i \in [1..n]$$
$$x \in \mathbb{R}^{|\mathcal{K}|}$$

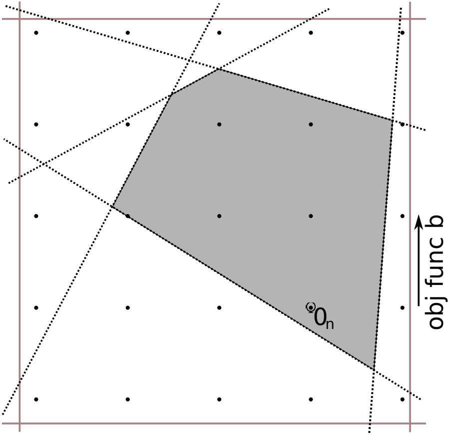
The Dual Polytope **P**

Main dual constraints:

- $\mathbf{a}^\top \mathbf{y} \le c_a, a \in \mathcal{K}$

Initial constraints:

- $y_i \in [l_i, u_i] \forall i \in [1..n]$



Column generation:

constraints (primal columns) generated one by one via the pricing problem

- pricing input: an (infeasible) dual solution that can be *anywhere* in the dual space

The Dual Polytope **P**

Main dual constraints:

- $\mathbf{a}^\top \mathbf{y} \leq c_a, a \in \mathcal{K}$

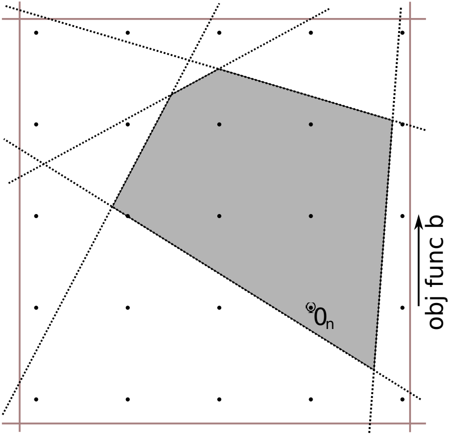
Initial constraints:

- $y_i \in [l_i, u_i] \forall i \in [1..n]$

Column generation:

constraints (primal columns) generated one by one via the pricing problem

- pricing input: an (infeasible) dual solution that can be *anywhere* in the dual space



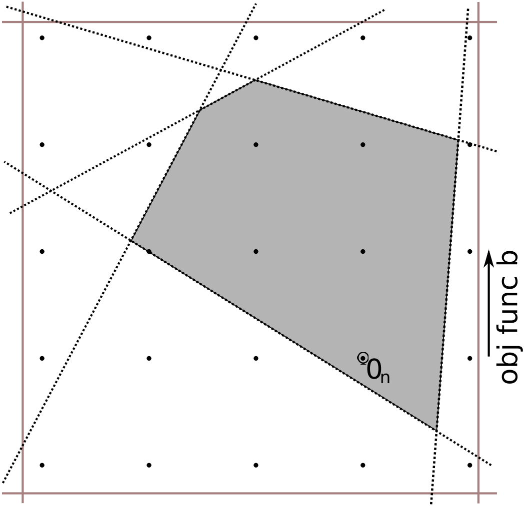
Optimizing Polytope P

$$\max_{\mathbf{y}\in\mathbf{P}} \mathbf{b}^\top \mathbf{y},$$

where **P** has prohibitively many constraints.

Constraints generated one by one (refine an "outer approximation" polytope):

- `Branch-and-Cut`, **P** is the primal
- `Column Generation`, **P** is the dual



Ray Projection in **P**

Init first ray: $\mathbf{r} \leftarrow \mathbf{b}$

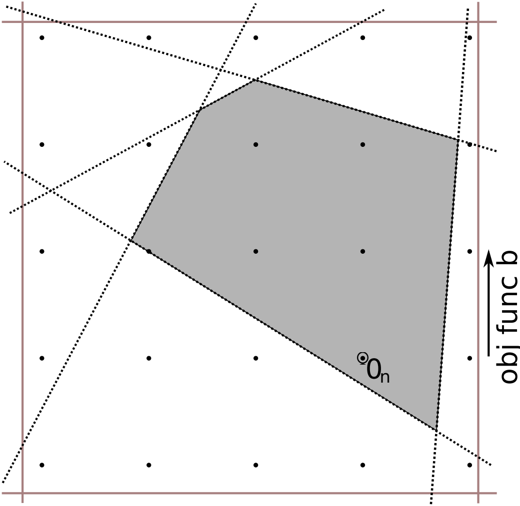
- fastest obj. improvement
- integer rays only

Intersection/Ray Subproblem

find the intersection point between ray $\mathbf{0_n} \rightarrow \mathbf{r}$ and **P**:

- lb$= t \cdot \mathbf{r}$ (contact point)
- a "first hit" facet

the generated facets form an "outer polytope": its optimum is ub



Ray Projection in **P**

Init first ray: $\mathbf{r} \leftarrow \mathbf{b}$

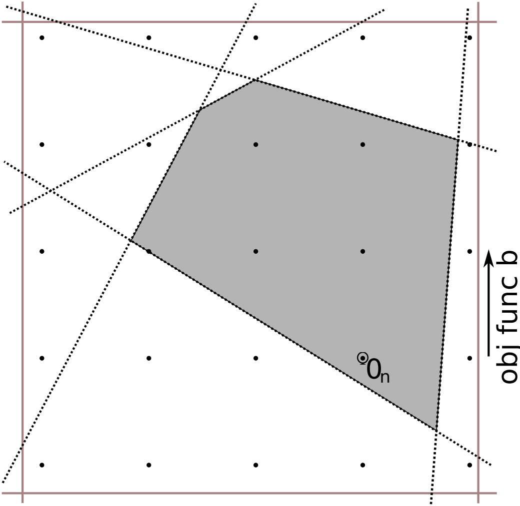
- fastest obj. improvement
- integer rays only

Intersection/Ray Subproblem

find the intersection point between ray $\mathbf{0_n} \rightarrow \mathbf{r}$ and $\mathbf{P}$:

- lb$= t \cdot \mathbf{r}$ (contact point)
- a "first hit" facet

the generated facets form an "outer polytope": its optimum is ub



Ray Projection in **P**

Init first ray: $\mathbf{r} \leftarrow \mathbf{b}$

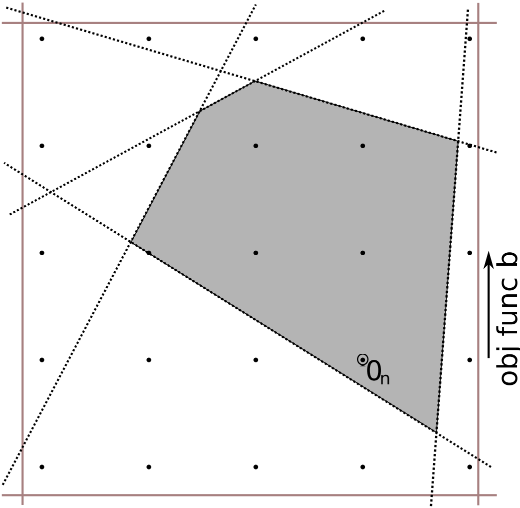
- fastest obj. improvement
- integer rays only

Intersection/Ray Subproblem

find the intersection point between ray $\mathbf{0_n} \rightarrow \mathbf{r}$ and **P**:

- lb$= t \cdot \mathbf{r}$ (contact point)
- a "first hit" facet

the generated facets form an "outer polytope": its optimum is ub



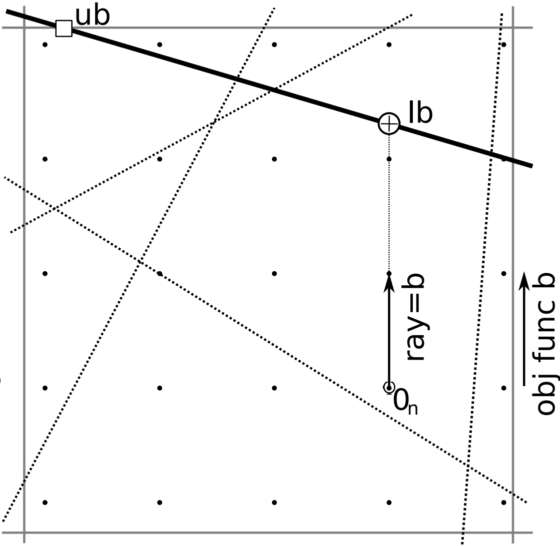
A Sequence of Integer Rays

One *Intersection Subproblem*

- one lower and one upper bound: *lb*, *ub*

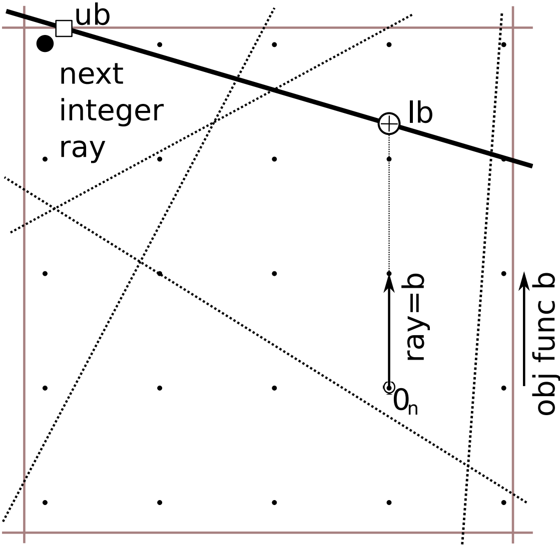
$\Longrightarrow$

Next integer ray need to be generated: search rays somewhere "in-between" *lb* and *ub*



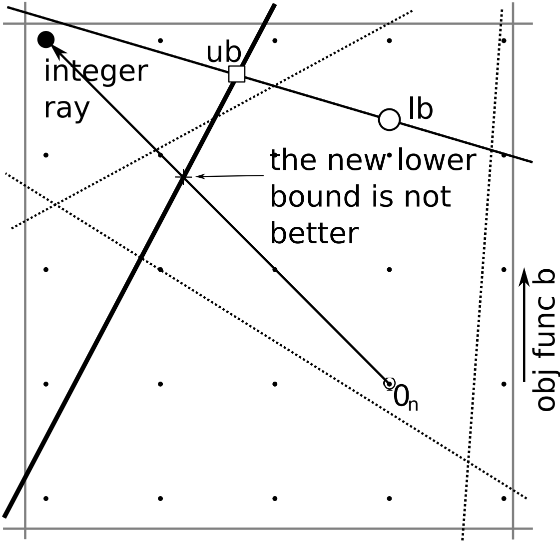
A Sequence of Integer Rays

New Ray Generator: find integer points close to the segment $lb - ub$



A Sequence of Integer Rays

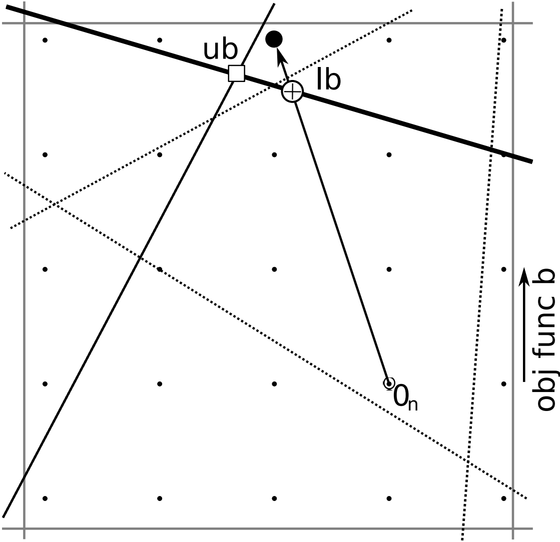
New constraint, but no better lower bound



A Sequence of Integer Rays

New ray: new lower bound, but no new upper bound.

- The infeasible *ub* is not cut by the new constraint
- No better rays available nearby *lb*, *ub*



A Sequence of Integer Rays

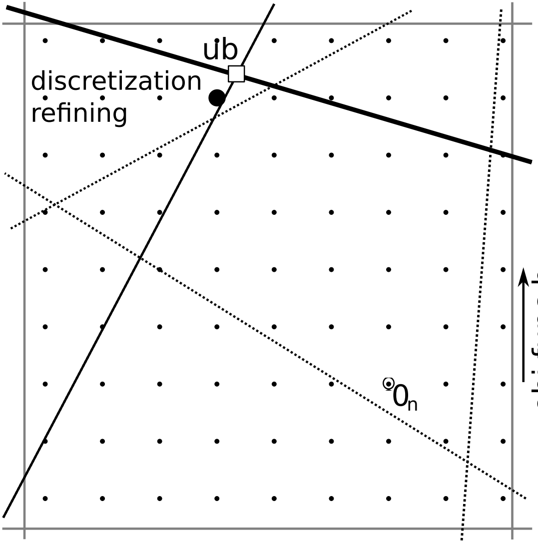
No potential integer ray improves the gap

$\Longrightarrow$

Discretization Refining

fractional rays are scaled ($\times 2$) to larger integer rays

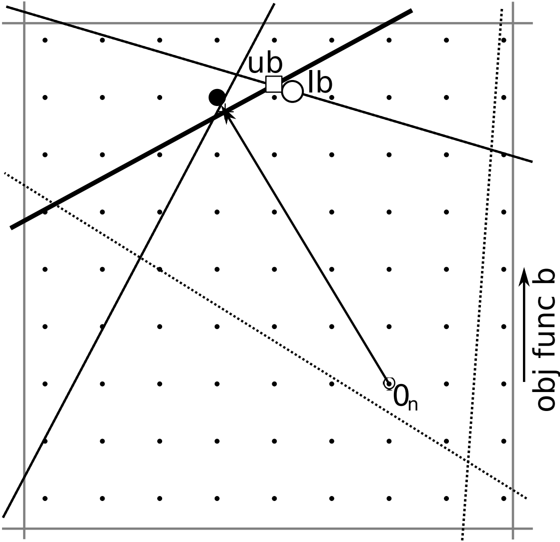
- ex.: $[3.5, 4.5] \rightarrow [7, 9]$



A Sequence of Integer Rays

Rays with larger integers:

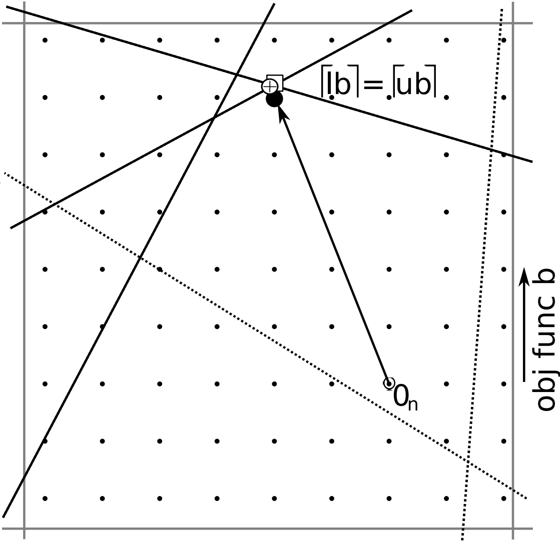
- more precision, new constraint discovered
- more calculations in the subproblem



A Sequence of Integer Rays

Stopping Condition: integer equality of *lb* and *ub*

- enough precision
- such polytopes are often relaxations of IPs

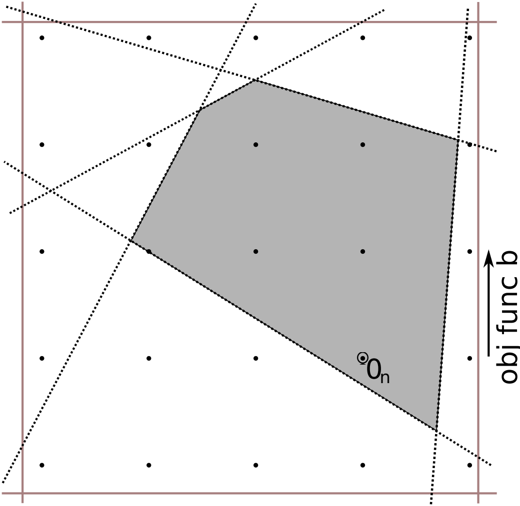


Solving the Intersection Subproblem: Intuition

Intersection Subproblem between ray $\mathbf{0_n} \to \mathbf{r}$ and $\mathbf{P}$

For $\mathbf{r} \in \mathbb{Z}^n$, find maximum t such that $t\mathbf{r} \in \mathbf{P}$

- lb$= t \cdot \mathbf{r}$ (contact point)
- a "first hit" constraint

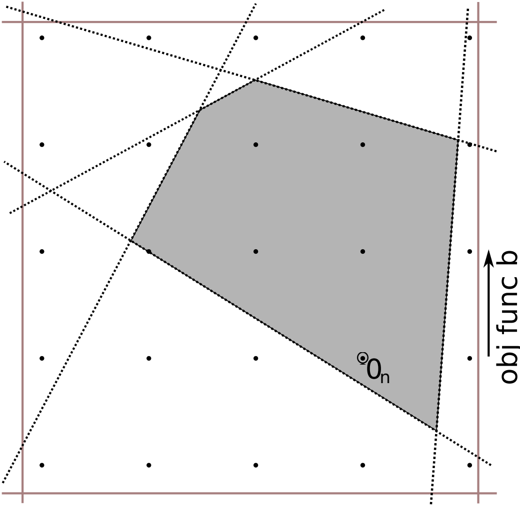


Solving the Intersection Subproblem: Intuition

Intersection Subproblem between ray $\mathbf{0_n} \to \mathbf{r}$ and $\mathbf{P}$

For $\mathbf{r} \in \mathbb{Z}^n$, find maximum t such that $t\mathbf{r} \in \mathbf{P}$

- lb$= t \cdot \mathbf{r}$ (contact point)
- a "first hit" constraint



The Intersection SubProblem Formalized

The maximum t such that $\mathbf{a}^\top(t\mathbf{r}) \leq c_a \; \forall \; \mathbf{a} \in \mathcal{K}$

"first hit" constraint $\mathbf{a}^\top \cdot (t\mathbf{r}) = c_a$

"Loose" constraint $\mathbf{a}^\top \cdot (t\mathbf{r}) < c_a$

Maximum t is associated to a first-hit constraint

$$t = \min \frac{c_a}{\mathbf{a}^\top \mathbf{r}}$$

The Column Generation sub-problem is different: minimize $c_a - \mathbf{a}^\top \mathbf{y}$, where $\mathbf{y}$ a dual-solution that is non-integer (or uncontrollable).

Column Generation and Intersection Subproblems

Column Gen (Separation) Subproblem: $\min c_a - \mathbf{a}^\top \mathbf{y}$, over all valid configurations $\mathbf{a} \in \mathcal{K}$

if $c_a = 1 \rightarrow$ this is equivalent to $\max \mathbf{a}^\top \mathbf{y}$

Ray (Intersection) Subproblem minimize cost/profit ratio $\frac{c_a}{\mathbf{a}^\top \mathbf{r}}$ over all valid configurations $\mathbf{a} \in \mathcal{K}$

if $c_a = 1 \rightarrow$ this is equivalent to $\max \mathbf{a}^\top \mathbf{r}$

$\Longrightarrow$ If the Column Generation Subproblem can be solved by Dynamic Programming, so can be the Intersection Subproblem

Important Advantage for the Intersection Subproblem: the input $\mathbf{r}$ can be selected (to become integer profits)

Column Generation and Intersection Subproblems

Column Gen (Separation) Subproblem: $\min c_a - \mathbf{a}^\top \mathbf{y}$, over all valid configurations $\mathbf{a} \in \mathcal{K}$

if $c_a = 1 \rightarrow$ this is equivalent to $\max \mathbf{a}^\top \mathbf{y}$

Ray (Intersection) Subproblem minimize cost/profit ratio $\frac{c_a}{\mathbf{a}^\top \mathbf{r}}$ over all valid configurations $\mathbf{a} \in \mathcal{K}$

if $c_a = 1 \rightarrow$ this is equivalent to $\max \mathbf{a}^\top \mathbf{r}$

$\Longrightarrow$ If the Column Generation Subproblem can be solved by Dynamic Programming, so can be the Intersection Subproblem

Important Advantage for the Intersection Subproblem: the input $\mathbf{r}$ can be selected (to become integer profits)

Dynamic program indexing: weights or profits?

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ brings a profit 1
- $w_2 = 4$ brings a profit 1
- $w_2 = 3$ brings a profit 1
- $w_2 = 2$ brings a profit 1

$maxP(Q)$	0	0	0	0	0	0	0	0	0	0	0
tot weight Q	0	1	2	3	4	5	6	7	8	9	10

Dynamic program indexing: weights or profits?

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ brings a profit 1
- $w_2 = 4$ brings a profit 1
- $w_2 = 3$ brings a profit 1
- $w_2 = 2$ brings a profit 1

$maxP(Q)$	0	0	0	0	0	1	0	0	0	0	0
tot weight Q	0	1	2	3	4	5	6	7	8	9	10

Dynamic program indexing: weights or profits?

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ brings a profit 1
- $w_2 = 4$ brings a profit 1
- $w_2 = 3$ brings a profit 1
- $w_2 = 2$ brings a profit 1

$maxP(Q)$	0	0	0	0	1	1	0	0	0	2	0
tot weight Q	0	1	2	3	4	5	6	7	8	9	10

Dynamic program indexing: weights or profits?

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ brings a profit 1
- $w_2 = 4$ brings a profit 1
- $w_2 = 3$ brings a profit 1
- $w_2 = 2$ brings a profit 1

$maxP(Q)$	0	0	0	1	1	1	0	2	2	2	0
tot weight Q	0	1	2	3	4	5	6	7	8	9	10

Dynamic program indexing: weights or profits?

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ brings a profit 1
- $w_2 = 4$ brings a profit 1
- $w_2 = 3$ brings a profit 1
- $w_2 = 2$ brings a profit 1

$maxP(Q)$	0	0	1	1	1	2	2	2	2	3	3
tot weight Q	0	1	2	3	4	5	6	7	8	9	10

Profit-indexed Dynamic Programming (DP)

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ required to gain a profit 1
- $w_2 = 4$ required to gain a profit 1
- $w_2 = 3$ required to gain a profit 1
- $w_2 = 2$ required to gain a profit 1

$minW(p)$	0	∞	∞	∞
tot profit p	0	1	2	3

Profit-indexed Dynamic Programming (DP)

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ required to gain a profit 1
- $w_2 = 4$ required to gain a profit 1
- $w_2 = 3$ required to gain a profit 1
- $w_2 = 2$ required to gain a profit 1

$minW(p)$	0	5	∞	∞
tot profit p	0	1	2	3

Profit-indexed Dynamic Programming (DP)

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ required to gain a profit 1
- $w_2 = 4$ required to gain a profit 1
- $w_2 = 3$ required to gain a profit 1
- $w_2 = 2$ required to gain a profit 1

$minW(p)$	0	5	∞	∞
tot profit p	0	1	2	3

Profit-indexed Dynamic Programming (DP)

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ required to gain a profit 1
- $w_2 = 4$ required to gain a profit 1
- $w_2 = 3$ required to gain a profit 1
- $w_2 = 2$ required to gain a profit 1

$minW(p)$	0	4	9	∞
tot profit p	0	1	2	3

Profit-indexed Dynamic Programming (DP)

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ required to gain a profit 1
- $w_2 = 4$ required to gain a profit 1
- $w_2 = 3$ required to gain a profit 1
- $w_2 = 2$ required to gain a profit 1

$minW(p)$	0	3	7	∞
tot profit p	0	1	2	3

Profit-indexed Dynamic Programming (DP)

Basic knapsack Example: $C = 10$, $\mathbf{w} = [5\ 4\ 3\ 2]$, all profits are 1

- $w_1 = 5$ required to gain a profit 1
- $w_2 = 4$ required to gain a profit 1
- $w_2 = 3$ required to gain a profit 1
- $w_2 = 2$ required to gain a profit 1

$minW(p)$	0	2	5	9
tot profit p	0	1	2	3

Weight-Indexed DP in Column Generation

Recall goal: $\max \mathbf{a}^\top \mathbf{y}$ over all $\mathbf{a} \in \mathcal{K}$

Cutting-Stock and Knapsack-like sub-problems

- calc. maximum profit $maxP(Q)$ for all feasible total weights Q
- a state for each $Q = \sum a_i w_i$ with $\mathbf{a} \in \mathcal{K}$

  classical knapsack : $Q = \sum a_i w_i \in [1..C]$, pattern cost 1

  elastic knapsack : $\sum a_i w_i$ can slightly exceed C

  - cost $c_a \leftarrow$ penalty for any capacity excess

Route subproblems in Arc-Routing

for each value $Q = \sum a_i w_i$: minCost(v, Q) defines the min red. cost of reaching vertex v with quantity Q

Weight-Indexed DP in Column Generation

Recall goal: $\max \mathbf{a}^\top \mathbf{y}$ over all $\mathbf{a} \in \mathcal{K}$

Cutting-Stock and Knapsack-like sub-problems

- calc. maximum profit $maxP(Q)$ for all feasible total weights Q
- a state for each $Q = \sum a_i w_i$ with $\mathbf{a} \in \mathcal{K}$

  classical knapsack : $Q = \sum a_i w_i \in [1..C]$, pattern cost 1
  elastic knapsack : $\sum a_i w_i$ can slightly exceed C
    - cost $c_a \leftarrow$ penalty for any capacity excess

Route subproblems in Arc-Routing

for each value $Q = \sum a_i w_i$: minCost(v, Q) defines the min red. cost of reaching vertex v with quantity Q

Profit-Indexed States for Intersection prob.

Recall goal: minimize cost/profit ratio $\boxed{\frac{c_{\mathbf{a}}}{\mathbf{r}^\top \mathbf{a}}}$ over all $\mathbf{a} \in \mathcal{K}$

We reverse the role of profits and weights

- integer rays $\rightarrow$ integer profits $\mathbf{r} = [r_1 \ r_2 \ldots r_n]$
- states defined by profit values $p = \sum r_i a_i$
- $minW(p)$: minimum *required weight* to yield profit p

c_p the minimum required cost to yield profit p is often determined from $minW(p)$

$$\text{return } t = \min_p \frac{c_p}{p}$$

Knapsack Subproblems in Cutting-Stock: Elastic Versions

Elastic Versions: (base) capacity C can be (slightly) exceeded

$$\text{configuration cost } c_a = \begin{cases} 1 & weight \leq C \\ f\left(\frac{weight}{C}\right) & weight > C \end{cases}$$

- f can be x^2 or x^3 or a stair-case function (`Var Sized BP`)

Dynamic Programming:

- Profit-Indexed: OK
  - the same profit-indexed scheme as for `Pure Knapsack`
  - each state has a minimum *weight* yielding minimum cost c_a
- Weight-indexed: TIME-CONSUMING if $C >> n$

$\Longrightarrow$

Intersection/RAY Subproblem OK

Column Generation Subproblem TIME-CONSUMING for $C >> n$

- Adapt other `Pure Knapsack` methods: DIFFICULT

Knapsack Subproblems in Cutting-Stock: Elastic Versions

Elastic Versions: (base) capacity C can be (slightly) exceeded

$$\text{configuration cost } c_a = \begin{cases} 1 & \textit{weight} \leq C \\ f\left(\frac{\textit{weight}}{C}\right) & \textit{weight} > C \end{cases}$$

- f can be x^2 or x^3 or a stair-case function (`Var Sized BP`)

Dynamic Programming:

- Profit-Indexed: OK
  - the same profit-indexed scheme as for `Pure Knapsack`
  - each state has a minimum *weight* yielding minimum cost c_a
- Weight-indexed: TIME-CONSUMING if $C >> n$

$\Longrightarrow$

Intersection/RAY Subproblem OK

Column Generation Subproblem TIME-CONSUMING for $C >> n$

- Adapt other `Pure Knapsack` methods: DIFFICULT

Experiments: Scaled and Non-Scaled Instances

Scaled large capacity `Cutting-Stock` and `Arc-Routing`:

- $C^* = C \times 1000,\ w_i^* = w_i \times 1000 - i\ Mod\ 10$

Inst		Ray Method		Column Gen With Pricing=		
Class	Name	Iters/Time		Minknap	Cplex	Class Dyn Prog
$f(x) = x^3$	vb10*-scaled	21	0.05	—	—	tm. out
	m01* -scaled	272	1.1	—	—	tm. out
	Hard*-scaled	578	16.2^{-1}	—	—	tm. out
	vb10	21	0.04	—	—	20 / 18.7
	m01	277	0.8	—	—	199 / 3.7
	Hard	568	19.3^{-1}	—	—	tm. out

Arc Routing Inst.			Ray Method			IP
Name	n	$\|V\|$	iters/time		final value	optimum
gdb1* -scaled	22	12	133	2.5	284	316
kshs1* -scaled	15	8	103	6.6	13553	14661
val1c* -scaled	39	24	204	152	225	319
gdb1	22	12	125	1.7	284	316
kshs1	15	8	103	2.7	13553	14661
val1c	39	24	193	205	225	319

Experiments: Scaled and Non-Scaled Instances

Scaled large capacity `Cutting-Stock` and `Arc-Routing`:

- $C^* = C \times 1000$, $w_i^* = w_i \times 1000 - i \; Mod \; 10$

Inst		Ray Method		Column Gen With Pricing=		
Class	Name	Iters	Time	Minknap	Cplex	Class Dyn Prog
$f(x) = x^3$	vb10*-scaled	21	0.05	—	—	tm. out
	m01* -scaled	272	1.1	—	—	tm. out
	Hard*-scaled	578	16.2^{-1}	—	—	tm. out
	vb10	21	0.04	—	—	20 / 18.7
	m01	277	0.8	—	—	199 / 3.7
	Hard	568	19.3^{-1}	—	—	tm. out

Arc Routing Inst. Name	n	$\lvert V \rvert$	Ray Method iters	time	final value	IP optimum
gdb1* -scaled	22	12	133	2.5	284	316
kshs1* -scaled	15	8	103	6.6	13553	14661
val1c* -scaled	39	24	204	152	225	319
gdb1	22	12	125	1.7	284	316
kshs1	15	8	103	2.7	13553	14661
val1c	39	24	193	205	225	319

Conclusions: Advantages of the Ray Method

- The computing effort stays in the same order of magnitude for scaled and unscaled instances
  - solved `Cutting-Stock` and `Arc-Routing` instances with weight magnitudes 1000 times larger than usual
- Lower bounds are provided *before* completely converging:
  - this is not a built-in feature in Column Gen.
- The rays (subproblem profits) can be controlled

$\mathbf{r} \in \mathbb{Z}^n \rightarrow$ profit-indexed Dynamic Programming can work even if weight-indexed Dynamic Programming fails in Col. Gen.

Conclusions: Advantages of the Ray Method

- The computing effort stays in the same order of magnitude for scaled and unscaled instances
  - solved `Cutting-Stock` and `Arc-Routing` instances with weight magnitudes 1000 times larger than usual
- Lower bounds are provided *before* completely converging:
  - this is not a built-in feature in Column Gen.
- The rays (subproblem profits) can be controlled

$\mathbf{r} \in \mathbb{Z}^n \rightarrow$ profit-indexed Dynamic Programming can work even if weight-indexed Dynamic Programming fails in Col. Gen.

Conclusions: Advantages of the Ray Method

- The computing effort stays in the same order of magnitude for scaled and unscaled instances
  - solved `Cutting-Stock` and `Arc-Routing` instances with weight magnitudes 1000 times larger than usual
- Lower bounds are provided *before* completely converging:
  - this is not a built-in feature in Column Gen.
- The rays (subproblem profits) can be controlled

$\mathbf{r} \in \mathbb{Z}^n \rightarrow$ profit-indexed Dynamic Programming can work even if weight-indexed Dynamic Programming fails in Col. Gen.