Recoverable Robustness by Column Generation

Marjan van den Akker[1]
Paul Bouman[1,2]
Han Hoogeveen[1]

1 Department of Information and Computing Sciences, Utrecht University
2 Rotterdam School of Management, Erasmus University

Universiteit Utrecht

Overview

- Introduction to Recoverable Robustness
- Size Robust Knapsack Problems
- Decomposition Framework
- Experiments
- Demand Robust Shortest Path
- Conclusions

Recoverable robustness

- Method to cope with uncertainty
- Combination of
  - Robust optimization
  - Two-phase stochastic programming
- Reference:

  Liebchen, Lübbecke, Möhring, Stiller

  LNCS 5868, 2009
- Much work done within the ARRIVAL project

Example: European Soccer Championship 2012

Scheme of the tournament

1. Group phase.
   Four groups: numbers 1 and 2 advance
2. Knock-out phase:
   1. Quarter finals
   2. Semi finals
   3. Final

We want to support our team!

- We want to buy tickets
- Tickets must be bought ***months in advance***
- The schedule for the group is known
- The schedule for the quarter finals etc. depends on the results in the group phase
- Which tickets should we buy?

To make it more clear …

- Group B:
  - Netherlands
  - Germany
  - Denmark
  - Portugal

- Scenario 1: Netherlands B1, Germany B2
- Scenario 2: Germany B1, Netherlands B2
- …….alternative scenarios: other rankings

Robust Optimization

- Find the cheapest solution
  - You don’t know yet which scenario applies
  - It must be feasible for all scenarios (possibly excluding the very unlikely ones)
  - Recovery actions afterwards are not possible

- Here: buy tickets for all matches for B1 and B2

2-phase stochastic programming

- Assumption: The probability of each scenario is known
- A solution comprises of first stage and second stage decisions; its cost is the expected cost
- The first stage decisions render a solution that can be made feasible through the second stage decisions when the real scenario gets revealed

For our example:

- First stage: buy tickets for the group phase now
- Second stage: acquire the tickets for the knock-out phase you need in some way

Recoverable robustness

- Robust optimization: too conservative
- 2-Phase stochastic programming: difficult models, may require lots of replanning

- Recoverable robustness:
  - Find an initial solution (first stage decision)
  - **You must be able to make it feasible using a simple recovery algorithm for scenarios under consideration**
  - The cost includes the first stage cost and possibly second stage

Recoverable robustness: example

Initial plan

- Suppose you think that Netherlands and Germany will advance, i.e., scenarios 1 and 2 are the scenarios under consideration
- Solution: Dutch supporters buy tickets for all matches of B1 and German supporters for B2, i.e., plan for scenario 1

Recovery rule

- Exchange tickets with a supporter of the other team

Cheap and simple

Size Robust Knapsack Problem

- Knapsack items with deterministic weight a_i and profit c_i
- Select subset of the items:
  - Maximize profit while not exceeding knapsack volume

- The volume of the knapsack is b, but this value may be smaller than originally assumed
  - Set of scenarios: scenario *s* has volume b_s with probability p_s (scenario 0 is the original case)

- Initial knapsack must be recoverable according to given rules for each of the scenarios

- Büsing, Koster, Kutschka (2010). Recoverable robust knapsack , uncertainty in weights and profits of items.

Size Robust Knapsack Problem

- Possible objectives:
  - Initial profit and Feasibility recovery
  - Initial profit + Worst-case recovery
  - Expected profit
- Some possible recovery rules
  - Recovery by removal
  - Recovery by swapping
  - Cardinality constrained recovery
  - Greedy recovery (weight, profit, or profit per unit weight)

- Certain variants can be solved by Dynamic Programming (Bouman 2011)

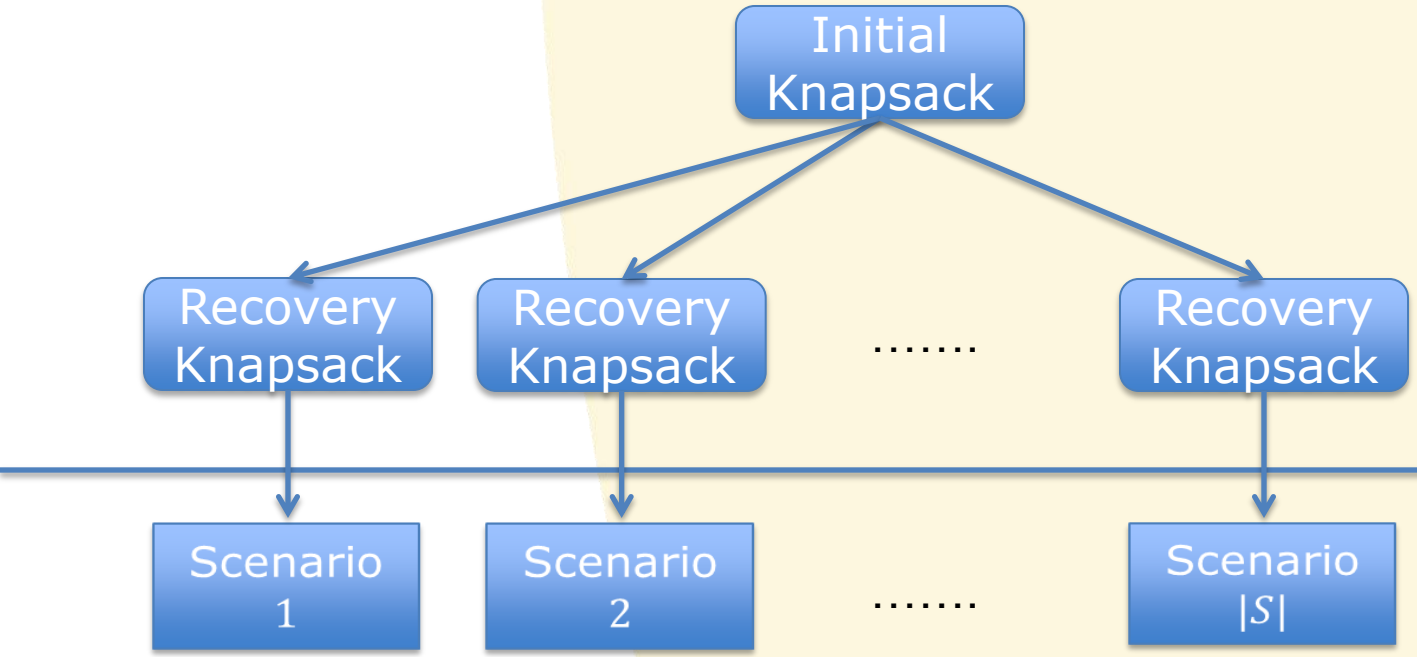
Decomposition framework

- Two variants:
  - Separate recovery decomposition
  - Combined recovery decomposition

- Shown for demand robust knapsack problem with
  - Maximize expected profit
  - Recovery by removing items

Decomposition Framework: separate recovery

- Idea: a solution to a recoverable robust optimization problem is a combination of a single initial solution and a recovery solution for each scenario



(Solution)

Contains all items in each

Does not exceed capacity of

(Problem)

Knapsack – Separate Recovery

- Objective: maximize expected profit
- Recovery: remove items

$$K(b) = \{k \subseteq \{1,2,\ldots,n\} \mid \sum_{i \in k} a_i \leq b\}$$

$$C_k = \sum_{i \in k} c_i$$

$$\text{For } k \in K(b): \quad a_{ik} = 1 \text{ if } i \in k$$

$$\text{For } k \in K(b_s): \quad a^s_{ik} = 1 \text{ if } i \in k$$

Variables :

$$x_k = 1 \text{ if knapsack } k \text{ is selected, and } 0 \text{ otherwise}$$

$$y^s_k = 1 \text{ if knapsack } k \text{ is selected for scenario } s, \text{ and } 0 \text{ otherwise}$$

ILP-Model: branch-and

$$\max \sum_{k \in K(b)} p_0 C_k x_k + \sum_{s \in S} p_s \sum_{k}$$

s.t.

$$\sum_{k \in K(b)} x_k = 1$$

$$\sum_{k \in K(b_s)} y_k^s = 1 \qquad \forall s \in S$$

$$\sum_{k \in K(b)} a_{ik} x_k - \sum_{k \in K(b_s)} a_{ik}^s y_k^s \geq 0 \qquad \forall i \forall s \in S$$

$$x_k, y_k^s \in \{0,1\}$$

Profit of the initial

Profit of the recovery solution

Select one valid initial knapsack

Select one valid initial knapsack

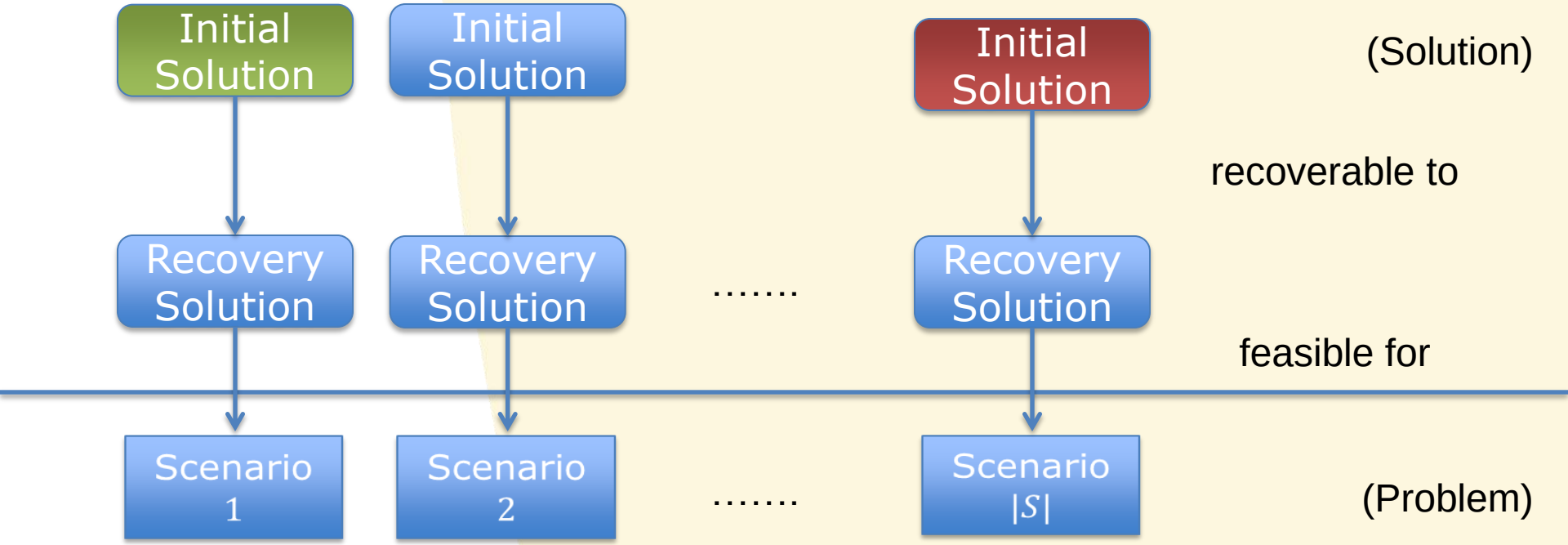
An item can only be in a recovery knapsack, if it is in the initial knapsack

Pricing problem

- Π_{is} duals of recovery constraints
- Pricing for initial problem variables (x_k):
  - Find optimal knapsack with total weight at most b and profit $(p_0\, c_i - \pi_{is})$
- Pricing for scenario s variables (y_k^s):
  - Find optimal knapsack with total weight b_s and profit $(p_s\, c_i + \pi_{is})$

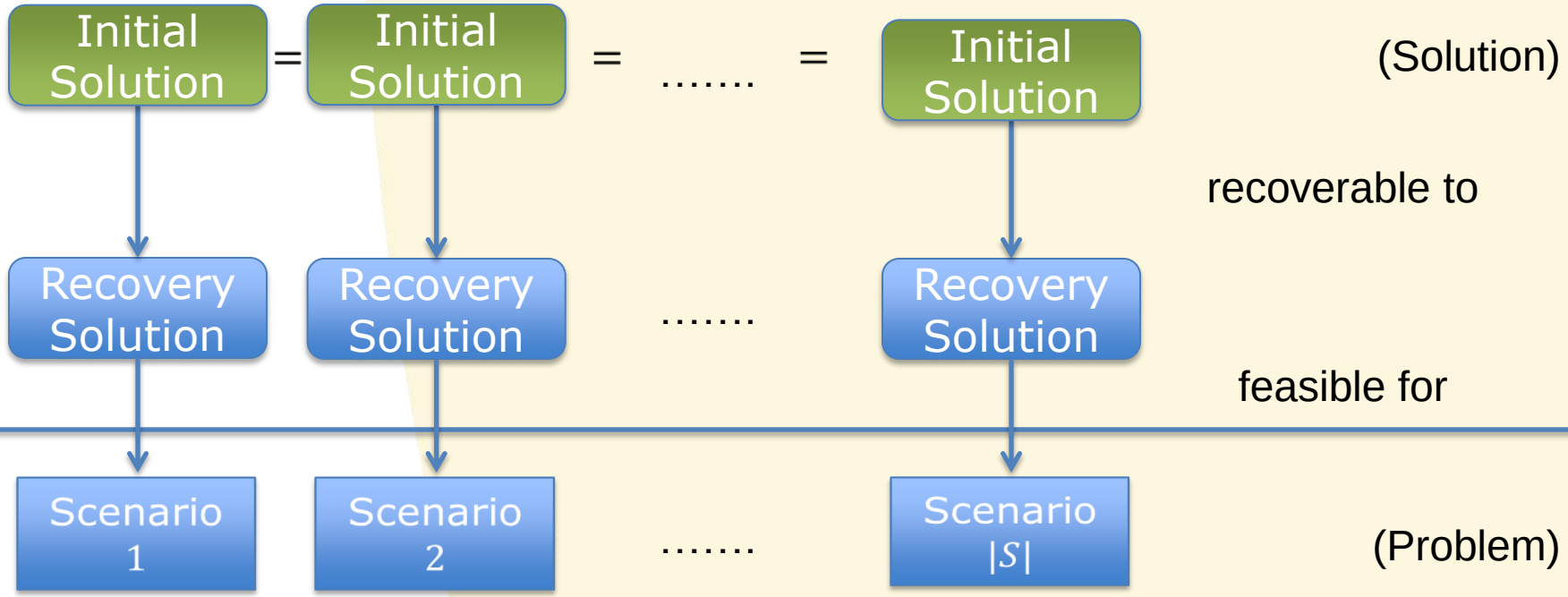
Decomposition Framework: Combined Recovery

- The decomposition considers combinations of an initial solution and a recovery solution



Decomposition Framework: Combined Recovery

- This may lead to a situation where we have a different initial solution for each scenario…
  - So we add the constraint that they must be equal



ILP-Model: branch-and-price

Variables :

x_i if item i is selected in initial solution

z_{kq}^s if initial solution k and recovery solution q are selected for scenario s.

$$\max \sum_{i=1}^{n} c_i x_i + \sum_{s \in S} p_s \sum_{(k,q) \in K(b) \times K(b_s)} C_q z_{kq}^s$$

$$s.t.$$

$$\sum_{k \in K(b)} z_{kq}^s = 1 \qquad \forall s \in S$$

$$x_i = \sum_{(k,q) \in K(b) \times K(b_s)} a_{ik} z_{kq}^s \qquad \forall i \forall s \in S$$

$$x_i, z_{kq}^s \in \{0,1\}$$

Select one combination of initial and recovery knapsack per scenario

All initial knapsacks are identical

Pricing problem

- Find an optimal combination of an initial and recovery knapsack for a scenarios *s*
  - The profit of the initial knapsack items are determined by duals of the ´equal initial´ constraints
  - The profit of the recovery knapsack items are equal to original profits.
- Solve by dynamic programming with state variables

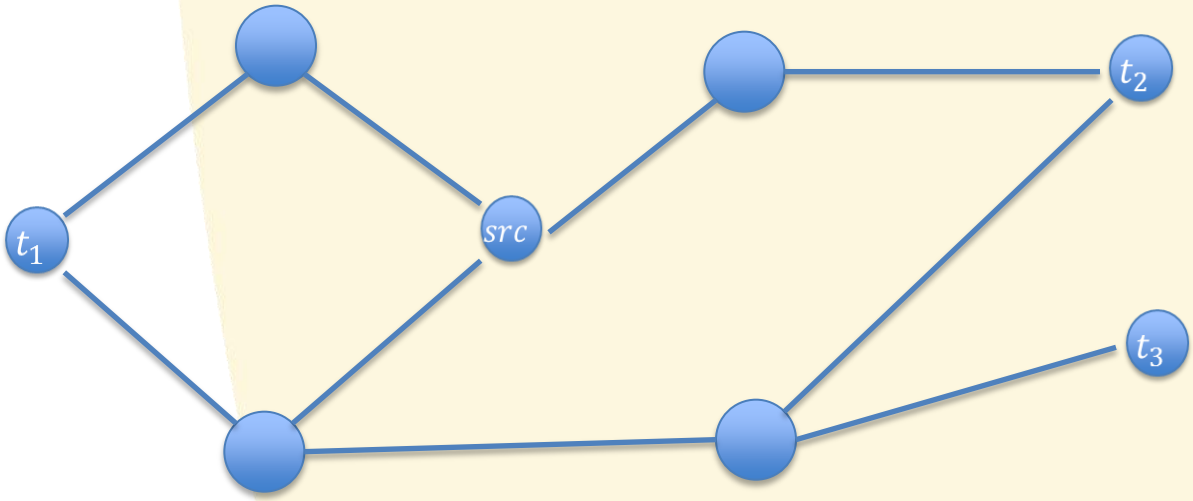
$D(i, w_0, w_s)$= Best value for a *combination* of an initial and recovery knapsack where the initial knapsack is a subset of $\{1,2,\ldots,i\}$ and the knapsacks have weights w_0 and w_s

Computational experiments

- 4 types of Exact Algorithms
  - Separate Recovery Branch & Price
  - Combined Recovery Branch & Price
  - Branch & Bound
  - Exact Dynamic Programming
- Local search
- Separate Recovery Branch&Price performs better than the Combined Recovery Branch&Price
- Hillclimbing seems to perform quite good, but gives no guarantees
- Separate Recovery Branch&Price best exact algorithm

Demand Robust Shortest Path

- Shortest path problem
- Single source src, scenarios define sinks
- Buy edges during a cheaper initial phase
- When sink is known, edges are more expensive



Demand Robust Shortest Path

- Separate Recovery Decomposition seems quite complex wrt modeling
- Use Combined Recovery Decomposition!
- Minimize worst-case cost
- Model similar to knapsack
- Additional constraint for worst-case cost

Demand Robust Shortest Path

- Pricing problem becomes a shortest path problem:
  - Two prices for each edge: initial prices and recovery prices
  - Just take the cheapest
  - All prices include duals
  - Only initial prices can be negative due to duals. We take edges with negative price no matter what and consider their costs to be 0
  - Find the shortest path in a graph with non-negative edge weights

Solve using Dijkstra's Shortest Path Algorithm!

Conclusions

- The type of recovery has a lot of impact on finding an algorithm to the problem
- The column generation framework is a great way to reduce recoverable robustness problems into regular problems (while customization remains possible)
- We believe that it can be applied to many different problems (we already have some preliminary work on Demand robust shortest path and Network flows)