Definition 1 *A feasible solution* $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, *for which the objective function has an optimum (i.e. maximum or minimum) value is called an* **optimal solution.**

Definition 2 *A set* M *is called* **convex**, *if for any two vectors* $\mathbf{x^1}, \mathbf{x^2} \in M$, *any vector*

$$\lambda \mathbf{x^1} + (1 - \lambda)\mathbf{x^2}$$

with $0 \leq \lambda \leq 1$ *also belongs to set* M.

Definition 3 *A vector (point)* $\mathbf{x} \in M$ *is called an* **extreme point** *of the convex set* M, *if* $\mathbf{x}$ *cannot be written as*

$$\lambda \mathbf{x^1} + (1 - \lambda)\mathbf{x^2}$$

with $\mathbf{x^1}, \mathbf{x^2} \in M$ *and* $0 < \lambda < 1$.

Theorem 1 *The set* M *of feasible solutions of an LPP is either empty or a convex set with at most a finite number of extreme points.*

Theorem 2 *If the set M of feasible solutions of an LPP is bounded, it can be written as the set of all convex combinations of the extreme points $\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^s}$ of set M, i.e.:*

$$M = \Big\{ \mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} = \lambda_1 \mathbf{x^1} + \lambda_2 \mathbf{x^2} + \cdots + \lambda_s \mathbf{x^s};$$

$$0 \leq \lambda_i \leq 1, \quad i = 1, 2, \ldots, s, \quad \sum_{i=1}^{s} \lambda_i = 1 \Big\}$$

Theorem 3 *If an LPP has a (finite) optimal solution, then there exists at least one extreme point, where the objective function has an optimum value.*

Theorem 4 *Let $P_1, P_2, \ldots, P_r$ described by vectors*

$$\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^r}$$

be optimal extreme points. Then every convex combination

$$\mathbf{x^0} = \lambda_1 \mathbf{x^1} + \lambda_2 \mathbf{x^2} + \ldots + \lambda_r \mathbf{x^r}$$

$$\lambda_i \geq 0, \ i = 1, 2, \ldots, r, \quad \sum_{i=1}^{r} \lambda_i = 1$$

is also an optimal solution.

Definition 4 *A system* $A\mathbf{x} = \mathbf{b}$ *of* $p = r(A)$ *linear equations, where in each equation one variable occurs only in this equation and it has the coefficient +1, is called system of linear equations in* **canonical form**.

These eliminated variables are called **basic variables** *(bv), while the remaining variables are called* **nonbasic variables** *(nbv).*

Definition 5 *A solution* $\mathbf{x}$ *of a system of equations* $A\mathbf{x} = \mathbf{b}$ *in canonical form, where each nonbasic variable has the value zero, is called a* **basic solution**.

Definition 6 *An LPP of the form*

$$z = \mathbf{c}^T\mathbf{x} \longrightarrow \max!$$

$$s.t. \qquad A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0},$$

where $A = (A_N, I)$ *and* $\mathbf{b} \geq \mathbf{0}$, *is called the* **standard form** *of an LPP.*

The standard form of an LPP

- is a **maximization** problem

- the constraints are given as
  a **system of linear equations**
  in **canonical form**
  with **non-negative right-hand sides** and

- all variables have to be **non-negative**

BASIC IDEA OF THE SIMPLEX ALGORITHM

(Iterative procedure)

Starting with some initial extreme point (basic feasible solution), compute the value of the objective function and check whether the latter can be improved upon by moving to an adjacent extreme point (by applying the pivoting procedure). If so, we make the move and seek then whether further improvement is possible by a subsequent move.

When finally an extreme point is attained that does not admit of further improvement, it will constitute an optimal solution.

Theorem 5 (Optimality criterion)
If

$$g_j \geq 0, \; j = 1, 2, \ldots, n',$$

for all coefficients of the nonbasic variables in the objective row, the corresponding solution is optimal.

Theorem 6 *If we have* $g_l < 0$ *for a coefficient of a nonbasic variable in the objective row and* $\hat{a}_{il} \leq 0$ *for all coefficients in column* l*, then the LPP does not have a (finite) optimal solution.*

Theorem 7 *If there exists a coefficient* $g_l = 0$ *in the objective row of an optimal solution such that* $\hat{a}_{il} > 0$ *for at least one coefficient in column* l*, then there exists another optimal basic feasible solution, where* x_{Nl} *is a basic variable.*

2 Discrete Optimization

2.1 Preliminaries

Discrete Optimization Problem:

$$f(\mathbf{x}) \to \min! \quad (\max!)$$
$$\mathbf{x} \in S$$

Special case: S finite

$\to$ Often S is described by linear inequalities / equations.

Integer (Linear) Optimization Problem:

$$f(\mathbf{x}) = \mathbf{c}^T \cdot \mathbf{x} \to \min! \quad (\max!)$$

s.t.

$$A \cdot \mathbf{x} \leq \mathbf{b}$$
$$\mathbf{x} \in \mathbb{Z}_+^n$$

Parameters A, **b**, **c** integer

$\mathbb{Z}_+^n$ - Set of integer, non-negative, n-dimensional vectors

Mixed Integer (Linear) Optimization Problem:

replace $\mathbf{x} \in \mathbb{Z}_+^n$ by

$$x_1, x_2, \ldots, x_r \in \mathbb{Z}_+$$

$$x_{r+1}, x_{r+2}, \ldots, x_n \in \mathbb{R}_+$$

Binary Optimization Problem:

replace $\mathbf{x} \in \mathbb{Z}_+^n$ by

$$x_1, x_2, \ldots, x_n \in \{0, 1\}$$
$$\text{i.e.,} \quad \mathbf{x} \in \{0, 1\}^n$$

Mixed Binary Optimization Problem:

$$x_1, x_2, \ldots, x_r \in \{0, 1\}$$

$$x_{r+1}, x_{r+2}, \ldots, x_n \in \mathbb{R}_+$$

Combinatorial Optimization Problem (COP):

The set S is finite and non-empty.

Example 1 (Investment planning) *An enterprise may realize 5 projects with the following expenditures (in Mill. EUR) for the next three years.*

Project	*year 1*	*year 2*	*year 3*	*profit*
1	*5*	*1*	*8*	*20*
2	*4*	*7*	*10*	*40*
3	*3*	*9*	*2*	*20*
4	*7*	*4*	*1*	*15*
5	*8*	*6*	*10*	*30*
Available budgets	*25*	*25*	*25*	

Which projects should be realized in order to maximize the profit?

2.2 Branch and Bound Algorithms (B&B)

- Exact procedure
- Method of implicit enumeration: Exclude successively subsets of S which cannot contain an optimal solution.
- Basic idea for minimization problems:
  - BRANCHING: Partition the set of solutions at least into two (disjoint) subsets.
  - BOUNDING: Determine for each subset $S(i)$ a lower bound $LB(i)$
  - Let UB be a known upper bound and $LB(i) \geq UB$ for $S(i)$, then $S(i)$ does not need to be considered further.

First we consider a *binary optimization problem*:

$$f(\mathbf{x}) \to \min!$$

s.t.

$$\mathbf{x} \in S \subseteq \{0, 1\}^n$$

Remark: In the case of a complete enumeration for $n = 50$, we would already obtain

$$| \{0,1\}^{50} | = 2^{50} \approx 10^{15}$$

possible combinations.

States of variables

Variable u_j describes the state of x_j as follows:

State of x_j	Value of x_j	Value of u_j
fixed 'settled'	1	1
fixed 'locked'	0	0
free	$0 \vee 1$	-1

- Vector $\mathbf{u} \in U := \{-1, 0, 1\}^n$ is identified with node u in the branching tree. Node u restricts the set of solutions as follows:

  $$S(u) = \{\mathbf{x} \in S \mid x_j = u_j,\ x_j \text{ fixed}\}, \quad j \in \{1, \ldots, n\}$$

- To node u, there corresponds the following optimization problem:

  $$\left.\begin{array}{rr} & f(\mathbf{x}) \to \min! \\ \text{s.t.} & \\ & \mathbf{x} \in S(u) \end{array}\right\} \quad P(u)$$

  Let $f^*(u) := \min\{f(\mathbf{x}) \mid \mathbf{x} \in S(u)\}$.

Introduction of bound functions

Definition 7 *A function* $LB: U \to \mathbb{R} \cup \{\infty\}$ *is called a* lower bound function, *if*

(a) $LB(u) \ \leq \ \min\{f(\mathbf{x}) \mid \mathbf{x} \in S(u)\} = f^*(u)$

(b) $S(u) = \{\mathbf{x}\} \ \Rightarrow \ LB(u) = f(\mathbf{x})$

(c) $S(u) \subseteq S(v) \ \Rightarrow \ LB(u) \geq LB(v)$

Definition 8 $UB \in \mathbb{R}$ *is called an* upper bound *on the optimal objective function value, if* $UB \ \geq \ \min\{f(\mathbf{x}) \mid \mathbf{x} \in S\}$.

$\bar{\mathbf{x}}$ represents the best solution found so far.

At the beginning of a B&B procedure, we set $UB := f(\bar{\mathbf{x}})$, if $\bar{\mathbf{x}}$ a heuristic solution, or we set $UB := \infty$.

Generation of the branching tree

active node: a node, which has *not* been investigated yet

At the beginning, the branching tree contains only the root $u = (-1, -1, \ldots, -1)^T$ as active node.

Investigation of an active node u

- *Case 1:* $LB(u) \geq UB$

  Node u is removed from the branching tree, since according to Definition 5 (a)

$$\min\{f(\mathbf{x}) \mid \mathbf{x} \in S(u)\} \geq LB(u) \geq UB$$

  holds.

  $\rightarrow$ Problem can be excluded.

- *Case 2a:* $LB(u) < UB$ with $u_j \in \{0,1\}$ for $j = 1, 2, \ldots, n$

  solution $x = u$ is uniquely determined

  If $\mathbf{x} \in S \quad \Rightarrow \quad$ due to

  $$f(\mathbf{x}) = LB(u) < UB = f(\bar{\mathbf{x}}),$$

  we have found a new best solution. Set $\bar{\mathbf{x}} := \mathbf{x}$ and $UB := f(\bar{\mathbf{x}})$. (Node u is no longer active.)

  $\rightarrow$ Problem can be excluded.

- *Case 2b:* $LB(u) < UB$ with $u_j = -1$ for (at least) one $j \in \{1, 2, \ldots, n\}$

  Generate the successor nodes w^i of node u by fixing one (or several) free variables. (Node u is no longer active, but the successors w^i of u are active.)

  $\rightarrow$ Problem is branched.

Search strategies - Selection of the next active node to be selected for investigation

(a) *FIFO strategy* (first in, first out)

Newly generated nodes are added to the end of the queue and the node at the beginning of the queue is investigated next.

$\rightarrow$ Breadth first search

(b) *LIFO strategy* (last in, first out)

Newly generated nodes are added to the end of the queue and the node at the end of the queue is investigated first.

$\rightarrow$ Depth first search

(c) *LLB strategy* (least lower bound)

The node with the smallest $LB(u)$ is investigated next. (If $LB(u) \geq UB$, then stop.)

The LIFO strategy delivers often quickly feasible solutions. During the course of the search, it is often recommendable to switch to the FIFO or LLB strategy.

On the bound function $LB(u)$

One or several constraints of $P(u)$ are "'relaxed"' or removed.

$\Rightarrow$ One obtains an easier problem $P^*(u)$ with $S^*(u) \supseteq S(u)$.

$P^*(u) \;\rightarrow\;$ *Relaxation* of $P(u)$

Set $LB(u) := f(\mathbf{x}^*(u))$, where $\mathbf{x}^*(u)$ is an optimal solution of $P^*(u)$.

Binary problem: For the free variables, replace $x_i \in \{0,1\}$ by $0 \leq x_i \leq 1$.

(*LP relaxation*)

B&B algorithm for binary optimization problems (minimization)

Step 1:

- If a feasible solution $\mathbf{x} \in S$ is known, set

$$\overline{\mathbf{x}} := \mathbf{x} \quad \text{and} \quad UB := f(\mathbf{x}),$$

  otherwise set

$$UB := \infty.$$

- Set $u^0 := (-1, -1, \dots, -1)^T$ and $U_a := \{u^0\}$ (u^0 is the root).

Step 2:

- If $U_a = \emptyset$, go to Step 4.

  Otherwise, select by means of a search strategy a node $u \in U_a$, remove u from U_a and calculate $LB(u)$.

Step 3:

- If $LB(u) \geq UB$, go to Step 4 in the case of the LLB strategy.

  Otherwise, eliminate u from the branching tree.

- If $LB(u) < UB$ and all variables are fixed, set in the case of $\mathbf{x} \in S$:

$$\overline{\mathbf{x}} := \mathbf{x} \quad \text{and} \quad UB := f(\mathbf{x}).$$

- If $LB(u) < UB$ and at least one variable is free, generate by fixing one (or several) free variables the successors of node u. Add the successors of u to U_a and to the branching tree.
- Go to Step 2.

Step 4: (Stop)

- If $UB < \infty$, then $\overline{\mathbf{x}}$ is an optimal solution with $f(\mathbf{x}) = UB$. Otherwise, the problem has no feasible solution.

This procedure can be generalized to <u>mixed binary problems</u> of the form

$$f(\mathbf{x}, \mathbf{y}) \to \min!$$

s.t.

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \in S$$

$$\mathbf{x} \in \mathbb{R}^n, \ \mathbf{y} \in \{0, 1\}^k.$$

If all binary variables are fixed, we have an LPP in the variables $x_1, x_2, \ldots, x_n$.

Modifications for integer programming problems

Use as relaxation the resulting LPP, where $x_i \in \mathbb{Z}_+$ is replaced by $x_i \geq 0$ (*LP Relaxation*).

The optimal solution (OS) gives a lower bound $LB(u)$ for node u (we have $S^*(u) \supseteq S(u)$).

<u>Algorithm by Dakin:</u> (branching strategy)

If in the OS of the LPP at least one variable x_i^* is not integer, generate *two* successor nodes v^k und v^l by adding the following constraints:

$$x_i \leq [x_i^*] \text{ in } S(v^k) \qquad \text{and}$$

$$x_i \geq [x_i^*] + 1 \text{ in } S(v^l)$$

3 Metaheuristics

3.1 Local Search, Preliminaries

Introduce a neighborhood structure as follows:

$$N \ : \ S \rightarrow 2^S$$
$$\mathbf{x} \in S \ \Rightarrow \ N(\mathbf{x}) \subseteq 2^S$$

S - Set of feasible solutions

$N(\mathbf{x})$ - Set of neighbors of a feasible solution $\mathbf{x} \in S$

Algorithm *ITERATIVE IMPROVEMENT*

1. determine an initial solution $\mathbf{x} \in S$;

   REPEAT

2. determine the best solution $\mathbf{x}' \in N(\mathbf{x})$;
3. **IF** $f(\mathbf{x}') < f(\mathbf{x})$ **THEN** $\mathbf{x} := \mathbf{x}'$;

   UNTIL $f(\mathbf{x}') \geq f(\mathbf{x})$ for all $\mathbf{x}' \in N(\mathbf{x})$.

$\mathbf{x}'$ - local minimal point w.r.t. neighborhood N

$\rightarrow$ The algorithm works with "largest improvement" *(best-fit)*.

Modification:

Use "first improvement" *(first-fit)*, i.e., search the neighborhood in a systematic way and accept a neighbor with a better objective function value than the current starting solution immediately for the next iteration.

(Stop, if a complete cycle with all neighbors has been checked without getting a better objective function value.)

$|\,\mathbf{N(x)}\,|$ **very large** $\Rightarrow$ Generate the neighbors randomly.

$\Rightarrow$ Replace row 2 in algorithm "Iterative Improvement" by

$$2^*\text{: Determine a solution } \mathbf{x}' \in N(\mathbf{x})$$

Stop, if

- a settled time limit is elapsed or
- a settled number of feasible solutions has been generated or
- a settled number of solutions after the last objective function value improvement has been generated without improving the objective function value further.

We consider

$$\boxed{\begin{array}{l} f(\mathbf{x}) \to \min! \quad (\max!) \\ \text{s.t.} \\ \quad \mathbf{x} \in S \subseteq \{0,1\}^n \end{array}}$$

Neighborhood $N_k(\mathbf{x})$:

$$N_k(\mathbf{x}) = \{\mathbf{x}' \in S \mid \sum_{i=1}^{n} \mid x_i - x_i' \mid \leq k\}$$

($\mathbf{x}' \in N_k(\mathbf{x}') \;\Leftrightarrow\; \mathbf{x}'$ is feasible and differs in at most k components from $\mathbf{x}$)

$$\Rightarrow \mid N_1(\mathbf{x}) \mid \quad \leq n$$

$$\mid N_2(\mathbf{x}) \mid \quad \leq n + \binom{n}{2} = n + \frac{n(n-1)}{2} = \frac{n(n+1)}{2}$$

For the systematic generation of neighbors, change component 1,2,...etc.

3.2 Simulated Annealing

randomized procedure, since

- $\mathbf{x}' \in N(\mathbf{x})$ is randomly selected
- in the i-th iteration, $\mathbf{x}'$ is accepted with probability

$$\min\left\{1, exp\Big(-\frac{f(\mathbf{x}') - f(\mathbf{x})}{t_i}\Big)\right\}$$

  as new starting solution

  ($\{t_i\}$ is a sequence of positive control parameters known as the temperature).

Algorithm *SIMULATED ANNEALING*

1. $i := 0$; choose t_0;
2. determine an initial solution $\mathbf{x} \in S$;
3. $best := f(\mathbf{x})$;
4. $\mathbf{x}^* := \mathbf{x}$;

   REPEAT

5. generate randomly a solution $\mathbf{x}' \in N(\mathbf{x})$;
6. **IF** $rand[0,1] < \min\left\{1, exp\left(-\frac{f(\mathbf{x}')-f(\mathbf{x})}{t_i}\right)\right\}$

   THEN $\mathbf{x} := \mathbf{x}'$;

7. **IF** $f(\mathbf{x}') < best$ **THEN**

   BEGIN $\mathbf{x}^* := \mathbf{x}'$; $best := f(\mathbf{x}')$ **END**;

8. $t_{i+1} := g(t_i)$;
9. $i := i + 1$;

   UNTIL stopping criterion is satisfied.

Modification:

Threshold Accepting (deterministic variant of Simulated Annealing)

- accept $\mathbf{x}' \in N(\mathbf{x})$ if

$$f(\mathbf{x}') - f(\mathbf{x}) \leq t_i$$

  t_i – *Threshold* in the i-th iteration

3.2 Tabu Search

Goal: Avoidance of 'short cycles'

$\Rightarrow$ use attributes to characterize the solutions attended recently and forbid the returnal to such solutions for a specified number of iterations

Notations:

$Cand(\mathbf{x})$	–	contains all neighbors $\mathbf{x}' \in N(\mathbf{x})$, to which a transition ('move') is allowed
TL	–	tabu list
t	–	length of the tabu list

Algorithm *TABU SEARCH*

1. determine an initial solution $\mathbf{x} \in S$;
2. $best := f(\mathbf{x})$;
3. $\mathbf{x}^* := \mathbf{x}$;
4. $TL := \emptyset$;

   REPEAT

5. determine $\quad Cand(\mathbf{x}) = \{ \ \mathbf{x}' \in N(\mathbf{x}) \mid$ the move from $\mathbf{x}$ to $\mathbf{x}'$ is not tabu $\}$;
6. select a solution $\overline{\mathbf{x}} \in Cand(\mathbf{x})$;
7. update TL (such that maximal t attributes are contained in TL);
8. $\mathbf{x} := \overline{\mathbf{x}}$;
9. **IF** $f(\overline{\mathbf{x}}) < best$ **THEN**

   BEGIN $\mathbf{x}^* := \overline{\mathbf{x}}$; $best := f(\overline{\mathbf{x}})$ **END**;

   UNTIL stopping criterion is satisfied.

3.3 Genetic Algorithms

- Use of **Darwin's evolution theory** (survival of the fittest)

- Genetic algorithms work with a **population of individuals** (chromosomes), which are characterized by their fitness

- Generation of offspring by **genetic operators** (crossover, mutation)

Fitness and Encoding of an Individual

e.g. fitness(ch)=$f(\mathbf{x})$ for $f \to$ max!

fitness(ch)=$\frac{1}{f(\mathbf{x})}$ for $f \to$ min! and $f(\mathbf{x}) > 0$,

where ch denotes the encoding of solution $\mathbf{x} \in S$

$$\mathbf{x} = (0, 1, 1, 1, 0, 1, 0, 1)^T \in \{0, 1\}^8$$

ch: | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

Genetic Operators for Generating Offspring

Mutation: 'Mutate' the genes of an individual.

parent chromosome	0 1 1 1 0 1 0 1
(3,5)-Inversion	0 1 | **0 1 1** | 1 0 1
2-Mutation	0 **0** 1 1 0 1 0 1
(1,4,7)-Mutation	**1** 1 1 **0** 0 1 **1** 1

Crossover: Combine the genetic structures of two individuals and generate two offspring.

1-Point-Crossover e.g. (4,8)-Crossover

P_1	1 0 1 | 0 0 1 0 1		O_1	1 0 1 | **1 0 0 1 1**
		$\rightarrow$		
P_2	0 1 1 | 1 0 0 1 1		O_2	0 1 1 | **0 0 1 0 1**

2-Point-Crossover e.g. (3,5)-Crossover

P_1	1 0 | 1 0 0 | 1 0 1		O_1	1 0 | **1 1 0** | 1 0 1
		$\rightarrow$		
P_2	0 1 | 1 1 0 | 0 1 1		O_2	0 1 | **1 0 0** | 0 1 1

Algorithm *GEN-ALG*

1. set the parameters population size $POPSIZE$, maximal number of generations $MAXGEN$, probability P_{CO} for the application of a crossover and probability P_{MU} for the application of a mutation;
2. generate the initial population POP_0 with $POPSIZE$ individuals (chromosomes);
3. determine the fitness of all individuals;
4. $k := 0$;

   WHILE $k < MAXGEN$ **DO**

      BEGIN

5. $h := 0$;

      WHILE $h < POPSIZE$ **DO**

         BEGIN

6. select two parents from POP_k (e.g. randomly proportional to their fitness values or according to roulette wheel selection);
7. apply with probability P_{CO} a crossover to the selected parents;
8. apply with probability P_{MU} a mutation to each of the individuals;
9. $h := h + 2$;

         END;

10. $k := k + 1$;
11. select from the generated offspring (and possibly also from the parents) $POPSIZE$ individuals of the k-th generation POP_k (e.g. proportional to their fitness values);

      END

4 Dynamic Programming

→ Problems are considered, which can be partitioned into particular 'stages' so that the overall optimization can be replaced by a 'stepwise optimization' over the stages.

→ Dynamic programming is often applied to an optimal control of economic processes, where the stages correspond to time periods.

4.1 Introductory Examples

(a) Inventory Problem

<u>Problem Formulation:</u>

- A good is stored during a finite planning horizon consisting of n periods.
- In each period, a delivery to the inventory is possible at the beginning.
- There is a demand in each period, which has to be satisfied after a potential delivery.

Notations:

$u_j \geq 0$ - amount delivered at the beginning of period j

$r_j \geq 0$ - demand in period j

x_j - stock immediately before the delivery in period j $(j = 1, 2, \ldots, n)$

Optimization problem:

$$\boxed{\begin{aligned} &\sum_{j=1}^{n} \left[K\delta(u_j) + hx_{j+1}\right] \to \min! \\ &\text{s.t.} \\ &x_{j+1} = x_j + u_j - r_j, \quad j = 1, 2, \ldots, n \\ &x_1 = x_{n+1} = 0 \\ &x_j \geq 0, \qquad j = 2, 3, \ldots, n \\ &u_j \geq 0, \qquad j = 1, 2, \ldots, n \end{aligned}} \tag{12}$$

Remark:

$$x_1 = x_{n+1} = 0 \quad \text{and} \quad (12)$$

$\Rightarrow$ Replace in the objective function hx_{j+1} by hx_j such that each term in the sum has the form $g_j(x_j, u_j)$.

$$x_j = x_{j+1} - u_j + r_j \geq 0 \quad \Rightarrow \quad u_j \leq x_{j+1} + r_j$$

The constraints can be formulated as follows:

$$\begin{aligned} x_1 &= x_{n+1} = 0 & \\ x_j &= x_{j+1} - u_j + r_j, & j = 1, 2, \ldots, n \\ x_j &\geq 0, & j = 1, 2, \ldots, n \\ 0 \leq u_j &\leq x_{j+1} + r_j, & j = 1, 2, \ldots, n \end{aligned}$$

(b) Knapsack Problem

$$u_j := \begin{cases} 1, & \text{if item } j \text{ is put into the knapsack} \\ 0, & \text{otherwise} \end{cases}$$

<u>Optimization problem:</u>

$$\sum_{j=1}^{n} c_j u_j \ \rightarrow \max!$$

s.t.

$$\sum_{j=1}^{n} a_j u_j \qquad \leq V$$

$$u_1, u_2, \ldots, u_n \in \{0, 1\}$$

$\rightarrow$ Here the states are no time periods. The decisions which of the items $1, 2, \ldots, n$ are put into the knapsack is interpreted as decisions in n successive stages.

x_j - remaining volume of the knapsack for the items $j,\ j+1,\ldots,n$

$\Rightarrow x_1 = V \quad \text{and} \quad x_{j+1} = x_j - a_j u_j \quad \text{for all } j = 1,2,\ldots,n$

<u>Reformulated optimization problem:</u>

$$\sum_{j=1}^{n} c_j u_j \ \rightarrow \text{max!}$$

u.d.N.

$$\begin{array}{lll}
x_{j+1} = x_j - a_j u_j, & & j = 1,2,\ldots,n \\
x_1 = V & & \\
0 \le x_{j+1} \le V, & & j = 1,2,\ldots,n \\
u_j \in \{0,1\}, & \text{if } x_j \ge a_j, & j = 1,2,\ldots,n \\
u_j = 0, & \text{if } x_j < a_j, & j = 1,2,\ldots,n
\end{array}$$

4.2 Problem Formulation

Dynamic programming problems consider a finite planning horizon, which is partitioned into n periods or stages.

State variable x_j:

$\rightarrow$ describes the state of the system at the beginning of period j (and at the end of period $j-1$, respectively)

$\rightarrow$ $x_1 := x_a$ - given initial state of the system

Decision variable u_j:

$\rightarrow$ In period 1 the decision u_1 is made, which transforms die system into the state x_2, i.e.,

$$x_2 = f_1(x_1, u_1),$$

where, from the decision u_1, the cost $g_1(x_1, u_1)$ results.

<u>in general:</u>

$x_{j+1} = f_j(x_j, u_j)$	resultant *state*
$g_j(x_j, u_j)$	*stage cost*
$X_{j+1} \neq \emptyset$	*State region*, which contains possible states at the end of period j, where $X_1 = \{x_1\}$
$U_j(x_j) \neq \emptyset$	*Control region*, which contains possible decisions in period j (depends on state x_j at the beginning of period j)

<u>Optimization problem:</u>

$$\boxed{\begin{array}{ll} \displaystyle\sum_{j=1}^{n} g_j(x_j, u_j) \ \rightarrow \min! & \\ \text{u.d.N.} & \\ x_{j+1} = f_j(x_j, u_j), & j = 1, 2, \ldots, n \\ x_1 = x_a, & \\ x_{j+1} \in X_{j+1}, & j = 1, 2, \ldots, n \\ u_j \in U_j(x_j), & j = 1, 2, \ldots, n \end{array}} \tag{13}$$

Remark:

In general, the time complexity increases exponentially with the dimension of the state and decision variables

Definition 9 *A sequence of decisions* $(u_1, u_2, \ldots, u_n)$ *is called* policy *or* control. *The sequence of decisions* $(x_1, x_2, \ldots, x_n, x_{n+1})$ *corresponding to a given policy* $(u_1, u_2, \ldots, u_n)$ *according to*

$$x_1 = x_a \text{ and } x_{j+1} = f_j(x_j, u_j) \qquad \text{for all } j = 1, 2, \ldots, n$$

is called the corresponding **state sequence**.

A policy or state sequence satisfying the constraints (13) is called **feasible**.

4.3 Bellman Equations and Bellman's Principle of Optimality

Given are g_j, f_j, X_{j+1} and U_j for all $j = 1, 2, \ldots, n$.

$\Rightarrow$ Optimization problem depends on x_1, i.e., $P_1(x_1)$.

analogously: $P_j(x_j)$ - problem for the periods $j, j+1, \ldots, n$ with the initial state x_j

Theorem 8 (Bellman's Principle of Optimality)

Let $(u_1^*, \ldots, u_j^*, \ldots, u_n^*)$ be an optimal policy for the problem $P_1(x_1)$ and x_j^* be the state at the beginning of period j, then $(u_j^*, \ldots, u_n^*)$ is an optimal policy for the problem $P_j(x_j^*)$, i.e.:

The decisions in the periods $j, \ldots, n$ of the n-period problem $P_1(x_1)$ are (for a given state x_j^*) independent of the decisions in the periods $1, \ldots, j-1$.

Bellman Equations:

1. Let $v_j^*(x_j)$ be the minimal cost for the problem $P_j(x_j)$. For $j = 1, 2, \ldots, n$, the relationships

$$\boxed{\begin{aligned} v_j^*(x_j) &= g_j(x_j, u_j^*) + v_{j+1}^*(x_{j+1}^*) \\ &= \min_{u_j \in U_j(x_j)} \left\{ g_j(x_j, u_j) + v_{j+1}^*[f_j(x_j, u_j)] \right\} \\ & x_j \in X_j \end{aligned}} \tag{14}$$

are called the *Bellman equations* (BE), where

$$v_{n+1}^*(x_{n+1}) = 0$$

for $x_{n+1} \in X_{n+1}$.

$\Rightarrow$ Function v_j^* can be determined provided that v_{j+1}^* is known.

2. BE can also be determined for the following cases:

(a) $\sum_{i=1}^{n} g_j(x_j, u_j) \;\to \max!$

$\Rightarrow$ Replace in (14) min! by max!

(b) $\prod_{i=1}^{n} g_j(x_j, u_j) \;\to \min!$

$\Rightarrow$ BE:

$$v_j^*(x_j) = \min_{u_j \in U_j(x_j)} \left\{ g_j(x_j, u_j) \cdot v_{j+1}^* \big[f_j(x_j, u_j) \big] \right\}$$

where $v_{n+1}^*(x_{n+1}) := 1$ and $g_j(x_j, u_j) > 0$

for all $x_j \in X_j, u_j \in U_j(x_j), j = 1, 2, \ldots, n$

(c) $\max_{1 \le j \le n} \big\{ g_j(x_j, u_j) \big\} \;\to \min!$

$\Rightarrow$ BE:

$$v_j^*(x_j) = \min_{u_j \in U_j(x_j)} \left\{ \max\big\{ g_j(x_j, u_j); v_{j+1}^* \big[f_j(x_j, u_j) \big] \big\} \right\}$$

where $v_{n+1}^*(x_{n+1}) = 0$

4.4 Bellman Method

$\Rightarrow$ successive evaluation of (14) for $j = n, n-1, \ldots, 1$ to determine $v_j^*(x_j)$

Algorithm DO

Phase 1: Backward Calculation

(a) Set $v_{n+1}^*(x_{n+1}) := 0$ for all $x_{n+1} \in X_{n+1}$.

(b) For $j = n, n-1, \ldots, 1$ do:
For all $x_j \in X_j$, determine $z_j^*(x_j)$ as the minimum point of function

$$w_j(x_j, u_j) := g_j(x_j, u_j) + v_{j+1}^*\big[f_j(x_j, u_j)\big]$$

on $U_j(x_j)$, i.e.,

$$w_j(x_j, z_j^*(x_j)) = \min_{u_j \in U_j(x_j)} w_j(x_j, u_j) = v_j^*(x_j) \text{ for } x_j \in X_j$$

Phase 2: Forward Calculation

(a) Set $x_1^* := x_a$.

(b) For $j = 1, 2, \ldots, n$ do:

$$u_j^* := z_j^*(x_j),\ x_{j+1}^* := f_j(x_j^*, u_j^*)$$

$\Rightarrow$ $(u_1^*, u_2^*, \ldots, u_n^*)$ optimal policy

$\Rightarrow$ $(x_1^*, x_2^*, \ldots, x_{n+1}^*)$ optimal state sequence for problem $P_1(x_1^* = x_a)$

Summary: DP (Dynamic Programming)

Phase 1: *Decomposition*

Phase 2: *Backward calculation*

Phase 3: *Forward calculation*

<u>Remark:</u> If all equations

$$x_{j+1} = f_j(x_j, u_j), \quad j = 1, 2, \ldots, n$$

can be uniquely solved for x_j, one can also execute first a forward calculation and then a backward calculation (e.g. for the inventory problem from 4.1).

4.5 Examples and Applications

(a) Knapsack Problem

Assumption: V, a_j, c_j - integer

$$g_j(x_j, u_j) = c_j u_j, \qquad j = 1, 2, \ldots, n$$

$$f_j(x_j, u_j) = x_j - a_j u_j, \qquad j = 1, 2, \ldots, n$$

$$X_{j+1} = \{0, 1, \ldots, V\}$$

$$U_j(x_j) = \begin{cases} \{0,1\} & \text{for } x_j \geq a_j \\ 0 & \text{for } x_j < a_j \end{cases}, j = 1, 2, \ldots, n$$

BE:

$$v_j^*(x_j) = \max_{u_j \in U_j(x_j)} \big\{ c_j u_j + v_{j+1}^*(x_j - a_j u_j) \big\}, \quad 1 \leq j \leq n$$

Backward Calculation:

$$v_n^*(x_n) = \begin{cases} c_n, & \text{if } x_n \geq a_n \\ 0, & \text{otherwise} \end{cases}$$

$$z_n^*(x_n) = \begin{cases} 1, & \text{if } x_n \geq a_n \\ 0, & \text{otherwise} \end{cases}$$

For $j = n-1, n-2, \ldots, 1$:

$$v_j^*(x_j) = \begin{cases} \max\bigl\{v_{j+1}^*(x_j);\ c_j + v_{j+1}^*(x_j - a_j)\bigr\}, & \text{if } x_j \geq a_j \\ v_{j+1}^*(x_j), & \text{otherwise} \end{cases}$$

$$z_j^*(x_j) = \begin{cases} 1, & \text{if } v_j^*(x_j) > v_{j+1}^*(x_j) \\ 0, & \text{otherwise} \end{cases}$$

$\Rightarrow v_1^*(V)$ - maximal value of the knapsack filling

Forward Calculation:

$$\begin{aligned} x_1^* &:= V \\ u_j^* &:= z_j^*(x_j^*), & j = 1, 2, \ldots, n \\ x_{j+1}^* &:= x_j^* - a_j u_j^*, & j = 1, 2, \ldots, n \end{aligned}$$

(b) Determination of a Shortest (Longest) Path in a Graph

Goal: Determine a shortest path from vertex (city) x_1 to vertex (city) x_{n+1}.

Let:

$X_j = \{x_j^1, x_j^2, \ldots, x_j^k\}$ - set of all vertices of stage j, $\quad 2 \leq j \leq n$

$X_1 = \{x_1\}, \quad X_{n+1} = \{x_{n+1}\}$

$U_j(x_j) = \{x_{j+1} \in X_{j+1} \mid \exists$ a vertex from x_j to $x_{j+1}\}, \quad j = 1, 2, \ldots, n$

$v_j^*(x_j)$ - length of a shortest path from vertex $x_j \in X_j$ to vertex x_{n+1}

$g_j(x_j, u_j) = c_{x_j,u_j}$

$f_{j+1}(x_j, u_j) = u_j = x_{j+1}$

$z_j^*(x_j) = u_j = x_{j+1}$ if x_{j+1} is the next vertex after vertex x_j on a shortest path from vertex x_j to vertex x_{n+1}

BE:

$$v_n^*(x_n) = c_{x_n,x_{n+1}} \quad \text{for} \quad x_n \in X_n$$

For $j = n-1, n-2, \ldots, 1$:

$$v_j^*(x_j) = \min\Big\{ c_{x_j,x_{j+1}} + v_{j+1}^*(x_{j+1}) \mid x_{j+1} \in X_{j+1} \\ \text{such that the arc } (x_j, x_{j+1}) \text{ exists} \Big\}$$