Optimierung und Entscheidungsunterstützung

SS19

Georg Ringwelski

Ziele

- Optimierungsprobleme erkennen und modellieren können
- Grundwissen haben über Lösungsverfahren
  - Lineare Programmierung
  - Gemischtganzzahlige Programmierung
  - Constaintprogrammierung
- Erfahrung haben mit der Anwendung in integrierten Anwendungen auf der Basis von OPL
- Grundwissen haben über UI-Design in Entscheidungsunterstützungssystemen

Inhalt

Vorlesung

1. Optimierungsprobleme
2. LP und MIP, Grundkonzepte und Anwendung
3. Constraintprogrammierung
4. UI Design

Übungen

- Modellierung in OPL
- Modellbasierte und imperative Programmierung OPL/Java
- Interaktive Optimierungssysteme

2 Übungsblätter: OPL und GCP

Formative Übungsaufgabe „GCP_Aufgabe“

1. GCP in OPL modellieren → Installieren Sie OPL Studio von IBM
2. Benchmarks damit lösen (Ziel: Performanzverbesserung)
3. Schnittstelle imperative Sprache, modellbasierte Solver z.B. Java und OPL
4. OOD „Stundenplanung“ und bijektive Transformation in GCP, Implementierung
5. Interaktion in dieser Konstellation realisieren: Manuelle Entscheidungen, undo/redo,…

Organisatorisches

- Prüfung: PK120: Ein Projekt analog GCP-Übung
- VL und Übung integriert: Mi 8.00, GII 207; Do 10.00, GII 210
- Website zur LV
- Fragen?

Literatur aus HS-Bibo

Survey, Search (incl. CP)

- Burke, Edmund (eds.) „Search Methodologies“

LP, MIP

- Hillier, Lieberman „Operations Research“
- Domschke, Drexl „Einführung Operations-Research“

Literatur aus HS-Bibo

Constraint Programming

- Marriot, Stuckey „Programming with Constraints“
- K.R. Abt „Principles of Constraint Programming
- Hofstedt, Wolf „Einführung in die Constraint-Programmierung“

Literatur, Werkzeug

OPL and ILOG CPLEX Optimization Studio

- Download an install OPL IBM
  - Academic initiative
  - Free trial

→ Bitte installieren, so dass OPL-DIE läuft

1. Optimierungsprobleme

1. Introduction and Motivation
2. Examples
   1. puzzles
   2. scheduling
3. Definitions
4. Solving a Constraint Satisfaction Problem
5. Basic constraint solvers

1.1 Introduction and Motivation

Decision Support and Optimization

- Use computer to support human (management) work
- Use it for what it is good in today

1.1 Introduction and Motivation

Application Domains

- scheduling (production, transport etc)
- timetabling, rostering (schools, hospitals…)
- graphics systems (scene analysis, GUI layout)
- simulation and diagnostics
- natural language processing
- purchasing and configuration
- etc

1.1 Introduction and Motivation

Approaches

- Artificial Intelligence
  - Tree-search algorithms (BT..A*)
  - Heuristics, Metaheuristics, Local Search, GA
  - Constraint Programming (CP) extends Tree-Search
- Operations Research, Management Science
  - Linear Programming (LP)
  - Mixed Integer Programming (MIP)

1.1 Introduction and Motivation

Basic Concepts

- A (decision-)**variable** has a set of possible values, its **domain**
- A **constraint** on a sequence of variables is a requirement that states which combinations of values are admitted

1.1 Introduction and Motivation

Basic Concepts

- defining variables and constraints is problem **modelling**
- finding values that satisfy all constraints is problem **solving**
- In LP/MIP/CP: modelling is programming and solving is performed by a reused system

1.1 Introduction and Motivation

Examples

Analyse and execute OPL model for the
- N-Queens Problem

Exercise: Model and solve the
- Golomb Ruler Problem

1.1 Introduction and Motivation

SW-Engineering aspects

problem solving by coding algorithms
- can be tuned faster
- high programming and research effort
- hard to re-use

problem solving by modelling for **constraint solvers**
- quick prototype easy
- tuning of model for high performance
- model sometimes reusable, solver highly reusable

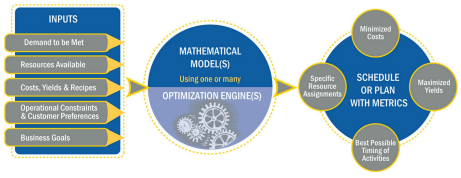
1.1 Introduction and Motivation

Why bother?

- combinatorial problems in practice are highly complex: hundreds or thousands of variables and constraints
- **The problems are usually NP-complete**
- There is no way to solve them without smart methodologies
- We don't want (are not able) to re-invent the methodologies for each application

Hochschule Zittau/Görlitz
UNIVERSITY OF APPLIED SCIENCES

Modellbasierte Verfahren (LP, MIP, CP)



[IBM Optimization and Decision Support Design Guide]

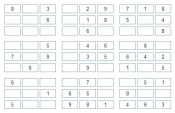
1. Optimierungsprobleme

1. Introduction and Motivation
2. Examples
   1. puzzles
   2. scheduling
3. Definitions

1. Optimierungsprobleme

1. Introduction and Motivation
2. Examples
   1. puzzles
   2. scheduling
3. Definitions
4. Solving a Constraint Satisfaction Problem
5. Basic constraint solvers

Modelle kombinatorischer Probleme



Sudoku

Spielregeln:

Ziffern in jeder Reihe unterschiedlich

Ziffern in jeder Spalte unterschiedlich

Ziffern in jedem Feld unterschiedlich

Modellierung

Variablen:
K11, K12 .....K19
K21, K22 .....K29
.....
K91.....K99

Constraints:
Reihen:
K11 ≠ K12, K11 ≠ K13 …
K12 ≠ K13, K12 ≠ K14…
…
K98 ≠ K99
Spalten:
K11 ≠ K21 … K89 ≠ K99
Felder:
K11 ≠ K22, …K88 ≠K99

Modelle kombinatorischer Probleme

Golomb Lineal

Ein Golomb Lineal der Dimension n besteht aus n unterschiedlichen Markierungen zwischen 0 und n^2, so dass die Differenzen aller Markierungen unterschiedlich sind.

Modellierung

Variablen:
V = V1…Vn

Constraints:
V1 =0
V1 <V2 <…<Vn
f.a.(x1,y1),(x2,y2) in VxV gilt:
… x1-y1 ! = x2-y2 …

Ziel:
minimiere Vn

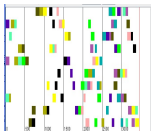
…vgl. OPL Lösung

Modelle kombinatorischer Probleme

Übung im Plenum: Wir definieren eine Modell für OPL Aufgabe Nr 3

- Welche Variablen gibt es?
- Welche Constraints gibt es?
- Was ist das Ziel der Optimierung?

Examples



scheduling

The job shop scheduling problem: in production n jobs have to be performed

- the plant uses m resources (e.g. machines)
- each job consists of a chain of m tasks which may not be interrupted
- each task is asigned to one machine

Goal: find an assignment of all tasks to machines and times such that everything is finished earliest

→ Analyse and solve a JSSP instance in OPL

Demo: Interaktives Problemlösen
hszg.de/eus → Demo

→ Welche features zur interaktiven Planung bietet das tool?

1. Optimierungsprobleme

1. Introduction and Motivation
2. Examples
   1. puzzles
   2. scheduling
3. Definitions

1. Optimierungsprobleme

1. Introduction and Motivation
2. Examples
   1. puzzles
   2. scheduling
3. Definitions
4. Solving a Constraint Satisfaction Problem
5. Basic constraint solvers

1.3 Definition

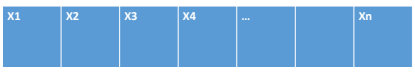
(constraint/decision) variable

- A **variable** is a placeholder for a value
- Each (constraint/decision) variable x has a **domain** D_x which is a set including all values the variable may take

1.3 Definitions

For instance: N queens

- a chessboard of dimension n
- Place n queens s.t. they cannot beat each other
- Domain: 1..n



1.3 Definition

OPL Syntax

dvar <type> <name> [<range>] [in <domain>]

- type: int, int+, float or float+
- range: [<index set>]
- domain: lower .. upper bound

Examples

- dvar int var in 1..4;
- dvar int queens[1..n] in 1..n

1.3 Definition

Constraint

- A constraint c has a (finite) scope S_c = <x1,x2,…xn> where xi is a decision variable
- A constraint C is a relation defined on the domains of its scope $C \subseteq D_{x1} \times D_{x2} \ldots \times D_{xn}$

(can be represented exlicitely or implicitely)

→on blackboard for instance: n-queens

1.3 Definition

OPL Syntax

- Constraints are defined in the „subject to“- block
- They are expressed implicitely by
  - equations or inequations
  - global constraint expressions (see ILOG docu)
- We can post multiple constraints using forall-expressions
- More to come

1.3 Definition

Übung im Plenum

Wir programmieren unser Modell zu OPL-Aufgabe 3 in OPL und lassen die Optimale Lösung berechnen

1.3 Definition

Constraint Satisfaction Problem (CSP)

- A Constraint Satisfaction Problem (V,D,C) is given by a set of variables V, their respective domains D and a set C of constraints over V

1.3 The CSOP

Definition

A (Constraint) Optimization Problem (CSOP/COP) is given by a CSP and an objective function o: S → M, where

•S is the set of all solutions to the CSP

•M is a metric

(typically M is defined as the set of real numbers)

1.3 The CSOP

OPL Syntax

- The objective function is defined by the keyword minimize/maximize followed a valid expression (which evaluations to a number)
- Before the „subject-to“-block
- Examples:

```
minimize ruler[n];

maximize sum(p in products) production[p]* profit[p];
```

1.3 The CSOP

Übung in Kleingruppen

Definieren Sie ein Modell für OPL-Aufgabe 4 als CSOP (auf Papier)

1.3 The CSOP

Solving tasks for CSOP/COP

- Find the best solution (if one exists)!
- Find the best solution you can get within a certain amount of time

1.3 The CSOP

Complexity

- Naturally the CSOP is NP-complete because it includes a CSP
- The problem is „harder“ such that performance can be expected to be worse

1. Optimierungsprobleme

1.4 Solving CSP

Definition

A **solution** to a CSP (V,D,C) is a variable assignment $\alpha : V \rightarrow D$, such that

- $\forall x \in V: \alpha(x) \in D_x$
- $\forall c \in C: S_c = \langle x1 \ldots xn \rangle \Rightarrow (\alpha(x1) \ldots \alpha(xn)) \in c$

1.4 Solving CSP

Solving tasks for CSP

- Does a solution exist?
- Give me one solution or tell me that none exists!
- Give me all or several solutions!

1.4 Solving CSP

CSP notions

- **Solving a CSP** usually refers to finding one solution or a proof that none exists
- A **partial solution** is a partial assignment which satisfies all constraints
- An **instantiation** is a variable assignment (which does not necessarily satisfy all constraints)
- A **solution** is an instantiation which is a partial solution

1.4 Solving CSP

Complexity

- The solution(s) of (V,D,C) are some of $|D|^{|V|}$ possible assignments
- There is no polynomial algorithm known to solve the (general) CSP
- The CSP is NP-complete

1. Optimierungsprobleme

1.5 Basic constraint solvers

Constraint solver

- A **constraint solver** contains an algorithm which can perform constraint solving tasks to CSPs
- Input: a (arbitrary) CSP
- Output: a solution OR all solutions OR proof that no solution exists
- Constraint solvers should be designed as reusable software components

1.5 Basic constraint solvers

Approach 1 generate-and-test solver:

- Algorithms: Local Search, GA, (greedy) Heuristics
- Efficiently ($O(n^c)$) create instantiation
- Try to improve by changing values (neighborhood-fct.)
- Usually Non-determinisitc
- Incomplete
- Hard to implement (performant) generic framework

1.5 Basic constraint solvers

Approach 2: Tree Search

- Algorithms: BT, BnB, A*, CP
- Incrementally extend partial solution
- Usually deterministic
- Exponential worst-time complexity ($O(c^n)$)
- complete

1.5 Basic constraint solvers

Approach 3: Linear Optimization

- Algorithms: simplex (LP) + BnB, CPLEX (MIP…)
- Compute analytically few solutions and BnB on them
- Deterministic
- Complexity: simplex: $O(n^c)$, BnB: $O(c^n)$

1.5 Basic constraint solvers

Optimization Programming Language

- Product of IBM
- Define CSOP and use CP or CPLEX to solve it „using CP“
- Reusable, highly optimized algorithms
- Alternatives: AMPL, GLPK (open source)

1.5 Basic constraint solvers

Differences between CP and CPLEX

- Completely different algorithms used in background
- LP Proplems can be solved much more efficiently in CPLEX
  - Float-variables only
  - Constraints: == and <= over linear terms only
- Solver automatically applies MIP, if int-variables are used
- Implicit constraints available in CP: alldifferent, count, …

1.5 Basic constraint solvers

Übung in Kleingruppen

- Analysieren Sie die Ergebnisse für Ihre bisherigen OPL-Programme (Aufg. 3 und 4) unter Verwendung
  - von CP vs. CPLEX
  - Von int-Variablen vs. Float-Variablen

1.5 Basic constraint solvers

Übung in Kleingruppen

- Implementieren Sie einen solver für das n-queens-Problem
- Gruppe 1: mit CP
- Gruppe 2: mit CPLEX

Inhalt

Vorlesung

1. Optimierungsprobleme
2. LP und MIP, Grundkonzepte und Anwendung
3. Constraintprogrammierung
4. UI Design

Übungen

- Modellierung in OPL
- Modellbasierte und imperative Programmierung OPL/Java
- Interaktive Optimierungssysteme

2. Lineare Optimierung

1. Lineare Optimierung , der Simplex Algorithmus
2. Integer Programming

Linear Programming (LP) general format

- maximize/minimize $\sum_{i=1}^{n} c_i * x_i$
- subject to $\sum_{i=1}^{n} a_{1i} * x_i \approx b_i$

  ...

  $\sum_{i=1}^{n} a_{mi} * x_i \approx b_i$
- where ≈ is one out of =, ≤, ≥

Diskussion im Plenum

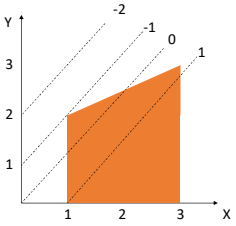
- **Sind Übungsaufgaben 1 und 2 LP-Probleme?**
- Lösen Sie Aufgabe 2a)
- Lassen Sie dann die ganzzahligkeitsbedingungen weg und lösen nochmal → was ist zu beobachten?

solving LP-type problems

- Use the simplex-Algorithm
- very widely used today
- roots in Mathematics (Dantzig) established in Operations Research
- Defined in Detail in for example in [HillierLieberman02]

a simplex intuition

minimize	X –Y
subject to	1 ≤ X
	X ≤ 3
	0 ≤ Y
	2Y – X ≤ 3



a simplex intuition

preprocessing steps:

- define objective as maximization (invert sign)
- replace variables x which may be negative (replace them by x^+ - x^- where x^+ and x^- are new non-negative variables and add „$x = x^+ - x^-$„)
- make all right-hand-sides positive (multiply by -1)
- replace inequations by equations with slack-variables : X ≤ Y → X + S = Y
- Write into tableau and solve

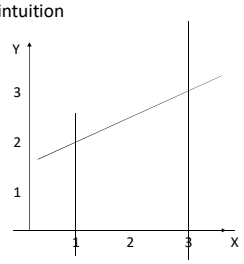
Hochschule Zittau/Görlitz

Lineare Optimierung

a simplex intuition

maximise -X +Y

subject to -S1 +X = 1
X + S2 = 3
2Y – X + S3 = 3
(0 ≤ X, 0 ≤ Y)



Hochschule Zittau/Görlitz

Lineare Optimierung

a simplex intuition

- consider vertices in polygon where all constraints are satisfied only(!)
- vertices in polygon are points where lines cross
- we can use Gauss-Jordan elimination to find crossing points

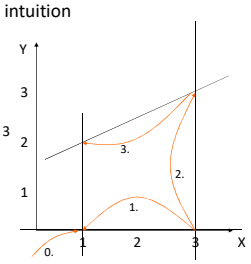
Hochschule Zittau/Görlitz

Lineare Optimierung

a simplex intuition

maximise -X +Y

subject to -S1 +X = 1
X + S2 = 3
2Y – X + S3 = 3
(0 ≤ X, 0 ≤ Y)



Hochschule Zittau/Görlitz

Lineare Optimierung

things to know about LP

- standard simplex has exponential worst case complexity
- variants step over points inside feasible region (interior point methods)
- some interior point methods have polynomial time complexity
- good implementations are available in LP solvers, appliers should not implement the algorithm
- Modelling may improve performance

2. Lineare Optimierung

1. Lineare Optimierung , der Simplex Algorithmus
2. Integer Programming

2.2 Integer Programming

What is Integer Programming (IP)

- Extends Linear Programming with **constraints requiring** variables to take **integer values**
- if all variables must be integers we IP, if some must be we talk **Mixed Integer Programming** (MIP)
- if we have constraints requiring boolean values we talk **Binary Integer Programming** (BIP)

2.2 Integer Programming

Why do we need IP, MIP and BIP?

- LP generally assumes dividability of variables
- however, many things are not dividable: humans, machines, yes/no-decisions …
- LP is not sufficient when we expect decisions

2.2 Integer Programming

IP example

a company owns 4 factories to produce a raw chemical the company wants to build refining plants at some of three possible sites which are associated to costs:

		site 1	site 2	site 3	production
transport cost	factory 1	25 $/t	20 $/t	15 $/t	1000 t/wk
	factory 2	15 $/t	25 $/t	20 $/t	1000 t/wk
	factory 3	20 $/t	15 $/t	25 $/t	500 t/wk
	factory 4	25 $/t	15 $/t	15 $/t	500 t/wk
fixed cost		500T$/yr	500T$/yr	500T$/yr	
capacity		1500 t/wk	1500 t/wk	1500 t/wk	

Where should they build refinaries to refine all chemicals?
How can they maximise their profit optaining 70$/t ?

2.2 Integer Programming

What is a linear relaxation?

if we drop from an MIP-type model P the integer-requirements we get its **linear relaxation** (LR) such that:

- if P is a maximization/minimization problem then the solution to LR is more/less than or equal to the solution to P
- if the solution to LR contains integers only it is the solution to P

2.2 Integer Programming

IP solving algorithm: B'n'B

1. create linear relaxation and solve it
2. do
   1. branch on non-integer-valued integer-variable and create new sub-problems by instantiating it to possible integers
   2. solve LR of new problems
   3. if solution is worse than best known, discard the new sub-problem
3. until all integer-variables have integer values

2.2 Integer Programming

complexity

- a problem with n binary variables may have up to 2^n sub-problems
- MIP has exponential worst case complexity
- no wonder: the problems subject to MIP are NP-complete

2.2 Integer Programming

things to know about IP/MIP/BIP

- it vastly extends the expressiveness of LP
- there are smart methods known to accelerate the algorithm in average performance
- out-of-the-box MIP systems use parameters to control the Branch and Bound
- MIP systems support specification languages
- Modelling has vast influence on performance

2.2 Integer Programming

controlling B'n'B in MIP

- select „important" variables first for branching
- use depth-first for selecting sub-problem to solve next when you are looking for a quick (but maybe not so good) solution, otherwise use best-bound

Inhalt

Vorlesung

1. Optimierungsprobleme
2. LP und MIP, Grundkonzepte und Anwendung
3. Constraintprogrammierung
4. UI Design

Übungen

- Modellierung in OPL
- Modellbasierte und imperative Programmierung OPL/Java
- Interaktive Optimierungssysteme

3. Constraint Programming

1. Foundations
2. Constraint Domains
3. Consistency in finite domains (FD)
4. Propagation in FD
5. Labelling in FD
6. Modelling issues in FD
7. Properties of CP

3.1 Foundations of CP

reconsidering CSP

- A Constraint Satisfaction Problem (V,D,C) is given by a set of variables V, their respective domains D and a set C of constraints over V
- in Constraint Programming (CP) we use this as models

3.1 Foundations of CP

represenatation of CSP

- a CSP is a model that serves as input for CP-solvers (declarative programming)
- CSPs can be modelled explicitely or implicitely
- Popular representation as constraint network
  - variables are nodes
  - constraints are (hyper-)arcs

3.1 Foundations of CP

inference in CP

- CP solvers use domain-knowledge of the variable-domains for inference
  ($\Rightarrow$ CP solvers differ with different constraint domains)
- inference helps exclude infeasible solutions quickly
- inference boosts complete search

3.1 Foundations of CP

OPL Syntax

- To use CP as a solver we write „using CP;“ as the first command in an OPL file
- The default is CPLEX, which does not allow as many constraints, but can be faster…

5. Constraint Programming

1. Foundations
2. Constraint Domains
3. Consistency in finite domains
4. Propagation in FD
5. Labelling in finite domains
6. Modelling issues in fd
7. Properties of CP

5.2 Constraint Domains

symbolic finite domains

- direct implementation of CSP: variable domains are arbitrary finite sets
- any symbolic values in domains
- few implicit constraints (=, ≠)
- explicit constraints expressed in tables or relations

5.2 Constraint Domains

finite integer domains

- variable domains are finite sets of integers
- most popular domain called finite domain (FD)
- implicit arithmetic constraints over arbitrary expressions
- implicit global constraints
- explicit constraints rarely used
- In OPL: dvar int <name>

5.2 Constraint Domains

boolean domains

- variable domains are {0,1}
- implicit logical constraints: AND, OR, …
- explicit constraints as tables
- usually implemented as regular FD
- In OPL: dvar boolean <name>;

5.2 Constraint Domains

real domains

- variable domains are intervals of reals (infinite)
- implicit arithmetic constraints
- solutions are given as disjunctions of intervals
- In OPL: dvar float <name>; (which is more recommendable when using CPLEX as solver)

5. Constraint Programming

1. Foundations
2. Constraint Domains
3. Consistency in finite domains
4. Propagation in FD
5. Labelling in FD
6. Modelling issues in FD
7. Properties of CP

5.3 Consistency in FD

Consistency of CSP

- „consistency“ stands for a property *and* for algorithms to achieve this property
- algorithms exclude domain values and thus reduce search space in polynomial complexity
- consistency algorithm transforms $(V,D_1 \ldots D_n, C)$ into $(V,D_1' \ldots D_n', C)$ such that $D`_i \subseteq D_i$ while retaining all solutions

5.3 Consistency in FD

Consistency and constraint solvers

- consistency algorithms are incomplete solvers
- consistency may (rarely) lead to singleton domains representing a solution
- if consistency leads to an empty domain D_i then there is no solution to the CSP
- in CP consistency is used in combination with search (e.g. backtracking) to achieve completeness

5.3 Consistency in FD

Node Consistency

(V,D,C) is node consistent iff

$\forall c \in C$: $|S_c| > 1$ OR

(S_c = <x> AND $\forall d \in D_x$: d satisfies c)

```
foreach Dx in D do
  foreach c in C do
    if(|scope(c)| == 1) then
      Dx := {d in D | d satisfies c}
```


5.3 Consistency in FD

Arc Consistency

(V,D,C) is arc consistent iff

$\forall c \in C$: $|S_c| \neq 2$ OR

(S_c = <x,y> AND $\forall d_x \in D_x \exists d_y \in D_y$: (d_x,d_y) satisfies c AND vice versa)

```
repeat
 W := D;
 foreach Dx in D do
  remove Dx from D
  foreach c in C do
   if(|scope(c)| == 2) then
    Dx := {dx in Dx | for some dy in Dy (dx,dy)sats c}
    Dy := {dy in Dy | for some dx in Dx (dx,dy)sats c}
until W == D
```


5.3 Consistency in FD

Scope of Node and Arc Consistency

- Applicable for any finite domains
- Work well for unary and binary constaints
- extensions of arc-consistency (e.g. hyper-arc-conistency) are NP-complete
- pragmatic solution for FD: bounds consistency, range consistency, „propagation“

5.3 Consistency in FD

Bounds/Range Consistency

- In FD a range [l..u] represents the set {l,l+1…u-1,u} of integer values betwen l and u
- Roughly: „A CSP is bounds/range consistent, if each variable domain is a range such that for the min and max/all values between u and l there is support in neighboring constraints“
- Exact definition of bounds consistency see e.g. [MarriotStuckey99]

5. Constraint Programming

1. Foundations
2. Constraint Domains
3. Consistency in finite domains
4. Propagation in FD
5. Labelling in FD
6. Modelling issues in FD
7. Properties of CP

5.4 Propagation in FD

Consistency in FD

- We consider ranges of values as domain
- We do **not** look at each value individually
- Objective: Best balance between search and inference
- Algorithm: Chaotic Iteration over Propagators

5.4 Propagation in FD

What do propagators in FD do?

- implement a constraint
- prune domains of variables in scope
- implemented as procedure or rule
- for arbitrary number of variables
- constant or linear complexity for arithmetric constraints

5.4 Propagation in FD

Example

- What could a propagater for „x < y +2“ look like?

(assume you get access to domains l..u by l := x.min(), u := x.max())

- Apply this propagator to x in 1..5, y in 1..5

5.4 Propagation in FD

Propagators of Global Constraints in FD

- A Global Constraint usually has a generic scope
- usually less complex than expressing the same thing with simple constraints
- can often prune more values because of more global view

5.4 Propagation in FD

The Global Constraints AllDiff

```
forall v in V with Dv = [1..1]
  V = V \ {v}
  forall v' in V
    Dv' = Dv \ 1
nv := |V|
R = emptyset
foreach v in V do
  R := R ∪ Dv
if nv > |R| then inconsistency detected
```


5.4 Propagation in FD

Example

Find out the difference between

a) x != y, y != z, x!= z

b) AllDifferent([x,y,z])

…by choosing domains für x,y,z such that propabagtion of a) and b) yield different results

5.4 Propagation in FD

Depending constraints

- propagators may prune values from an initial CSP
- propagators may prune further when other propagators have pruned values

thus constraints should be triggered whenever the domain of a variable in its scope was reduced.

5.4 Propagation in FD

Propagation in FD solvers

- store all constraints to be considered centrally (constraint store) CS:

```
while CS is not empty
  execute propagator of any c in CS
```

- store the constraints DC with $x \in S_c$, $c \in DC$ in each variable and trigger them:

```
if myDomain was reduced
  CS = CS ∪ DC;
```


5.4 Propagation in FD

Chaotic Iteration

- Propagation in FD using constraint store and triggers implements a Chaotic Iteration
  - All propagators must not extend domains
  - We can use arbitrary execution orders
  - There exists a fixed point which is reached
- Implementation has (bad) polynomial complexity: O(|D| * |C|)
- average complexity can be very good

5.4 Propagation in FD

Example

- Perform the chaotic iteration of

„x < y +2; y < z+2“ , where x,y,z in 1..6

5. Constraint Programming

1. Foundations
2. Constraint Domains
3. Consistency in finite domains
4. Propagation in FD
5. Labelling in FD
6. Modelling issues in FD
7. Properties of CP

5.5 Labelling in FD

Consistency and Propagation

- reduces domains
- may detect inconsistencies

but are not complete
We need search in addition

5.5 Labelling in FD

Search in FD

- in FD we have finite but exponential search spaces, size: $O(|D|^{|V|})$
- We use backtracking as basic (and complete) search
- ***during search*** we use propagation for inference and prune the search space

5.5 Labelling in FD

Backtracking in FD

```
assignment solveBT(csp,a)
  if(is_solution(a)) return a;
  else choose new variable x in csp;
    for all d in Dx do
      a = a ⊕ x ← d; // instantiate x
      if( is_partial_solution(a,csp))
        return solveBT(csp,a)
      a = a – X ← d  //undo instantiation
```


5.5 Labelling in FD

Backtracking in FD with Propagation

```
assignment solveBTwithProp(csp,a)
  if(is_solution(a)) return a;
  else choose new variable x in csp;
    for all d in Dx do
      s = currentStateOfVariableDomains;
      a = a ⊕ x ← d; // instantiate x
      ok = propagate(); // use inference
      if( ok & is_partial_solution(a,csp))
        return solveBTwithProp(csp,a)
      currentState = s;  // undo instantiation
                            and propagation
```


5.5 Labelling in FD

Übung im Plenum

- Führen Sie labelling mit und ohne Propagierung beim 6-queens-Problem durch.
- Wie viele backtracks sind jeweils notwendig?

5.5 Labelling in FD

Labelling

- Backtracking with Propagation is called Labelling
- Labelling is sound and complete
- can be accelerated with heuristics:
  - variable selection
  - value selection

5.5 Labelling in FD

Variable Selection

```
assignment solveBTwithProp(csp,a)
  if(is_solution(a)) return a;
  else choose new variable x in csp;
    for all d in Dx do
      s = currentStateOfVariableDomains;
      a = a ⊕ x ← d; // instantiate x
      ok = propagate(); // use inference
      if( ok & is_partial_solution(a,csp))
        return solveBTwithProp(csp,a)
      currentState = s;  // undo instantiation
                            and propagation
```


5.5 Labelling in FD

Variable Selection

- Defines during search which variable to instantiate next
  - lexicographic/random
  - first-fail
  - domain-specific
- Usually more crucial to performance than Value-Selection

5.5 Labelling in FD

Übung im Plenum

- Vergleichen Sie bei 8-queens Problem zwei Variablenauswahlheuristiken:
  - Von links nach rechts
  - Von der mitte nach außen
  - First fail
- (Wählen sie dabei immer den kleinsten Wert zuerst)

5.5 Labelling in FD

Value Selection

```
assignment solveBTwithProp(csp,a)
  if(is_solution(a)) return a;
  else choose new variable x in csp;
    for all d in Dx do
      s = currentStateOfVariableDomains;
      a = a ⊕ x ← d; // instantiate x
      ok = propagate(); // use inference
      if( ok & is_partial_solution(a,csp))
        return solveBTwithProp(csp,a)
      currentState = s;  // undo instantiation
                            and propagation
```


5.5 Labelling in FD

Value Selection

- Defines during search which value to instantiate the variable with next
  - ascendning, decending, mid
  - Random (each value once)
  - domain-specific

5.5 Labelling in FD

Übung im Plenum

- Vergleichen Sie bei 4-queesn Problem zwei Wertauswahlheuristiken:
  - Von min nach max
  - Von der Mitte alternierend nach oben/unten
- (Wählen Sie dabei die Variablen von links nach rechts)

5.5 Labelling in FD

Übung im teams (10 min)

- Definieren Sie Variablen- und Wertauswahlheuristiken für das 10-queens Problem, so dass sie möglichst ohne backtracking ein Lösung finden.
- (Im Internet finden Sie vielleicht gute Ideen)

5.5 Labelling in FD

Optimization in FD

- most commonly used: Branch and Bound
- implement objective function f as constraint f = Y with new variable Y
- gradually approximate optimal solution:

```
while(success)
    Z := Y;
    instantiate Y with better value;
    success := findSolution();
end while
Z is optimal
```


5.5 Labelling in FD

Optimization in FD

- linear descending (=monotonic) BnB

```
upperBound = MAX_VALUE;
success = solve(upperBound);
while(success)
  upperBound = upperBound – 1;
  success = solve(upperBound)
end while
optimalSolution = solve(upperBound + 1)
```


5.5 Labelling in FD

Übung im Plenum

- Definieren Sie Variablen- und Wertauswahlheuristiken für das Golomb Ruler 8 Problem, so dass sie schnell eine Lösung finden
- Simulieren Sie monotones BnB auf der Suche nach einer Optimalen Lösung

5.5 Labelling in FD

Optimization in FD

- binary approximating (dichotomic) BnB

```
upperBound = MAX_VALUE;
lowerBound = MIN_VALUE;
lastSolved = upperBound;
success = solve(upperBound)
while(success)
  success = solve((upperBound+lowerBound)/2);
  if(success)
    lastSolved = upperBound;
    upperBound = (upperBound+lowerBound)/2
  else
    lowerBound = (upperBound+lowerBound)/2
end while
optimalSolution = solve(lastSolved);
```

5.5 Labelling in FD

Übung im Plenum

- Simulieren Sie dichotomes BnB auf der Suche nach einer Optimalen Lösung von Golomb Ruler 8 in OPL

5.5 Labelling in FD

Dichotomic B'n'B

- often faster
- same anytime properties
- applicable with other solvers
- (killer feature) provides information about the absolute quality of a non-optimal solution

5. Constraint Programming

1. Foundations
2. Constraint Domains
3. Consistency in finite domains
4. Propagation in FD
5. Labelling in FD
6. Modelling issues in FD
7. Properties of CP

5.6 Modelling in FD

Why bother about models?

- The model is most crucial to solving performance
- often vast performance gap between different models of same problem:
  - ideally prune the most possible with each execution of a propagator
  - focus the solutions we are really interested in

5.6 Modelling in FD

modelling methodology

- implement one model and test it
- try equivalent modifications and test again
- try more restricitive models that concentrate on most intended solutions and test again

always try with different labelling heuristics

5.6 Modelling in FD

global constraints

- usually more efficient
- usually more effective

- guideline: when possible use global constraints!

5.6 Modelling in FD

Experiment

- Vergleichen Sie die Performanz von n-queens oder golomb ruler mit und ohne globales AllDifferent Constraint

5.6 Modelling in FD

Projekt

- Wir vergleichen die Performanz von cumulative mit unterschiedlichen Cumulative.Filter Varianten

5.6 Modelling in FD

redundant constraints

- are additional constraints that are implied by other constraints
- they may speedup propagation
- guideline: connect variables when possible!

5.6 Modelling in FD

Example: redundant constraints in golomb ruler

// basic problem definition
X1 = 0
X1<X2<…<XM
∀i<j Di,j = Xj – Xi
all_different({D1,2,D1,3,…,DM,M-1})

//symmetry breaking
D1,2 < DM-1,M
// redundant constraints
∀i<j Di,j ≥ (j-i)*(j-i+1)/2
∀i<j Di,j ≤ XM–(M-1-j+i)*(M-j+i)/2

size	Base model	Base model + symmetry	Base model + symmetry + redundant constraints
7	220	80	30
8	1462	611	190
9	13690	5438	1001
10	120363	49971	7011
11	2480216	985237	170495

[Roman Bartak INAP´05]

5.6 Modelling in FD

Experiment

- Vergleiechen Sie die performanz von MagicSquare und Golomb Ruler mit und ohne redundante Constraints (Magische Summe festlegen bzw. Grenzen für Differenzen s.o.)

5.6 Modelling in FD

Projekt

- Wir verwenden mehrere cumulative-Constraints mit unterschiedlichen Filtering-Algorithmen
- Wo könnte man redundante Constraints einsetzen? Wo wäre ein positiver Effekt möglich?

5.6 Modelling in FD

symmetry breaking

- prunes some solutions that may be constructed afterwards
- prunes symmetrical parts of search tree
- applied by ordering variables (lex, <, ≤)

- guideline: always break symmetries!

5.6 Modelling in FD

Experiment

- Vergleiechen Sie die performanz von MagicSquare, nQueens und Golomb Ruler mit und ohne Symmetry Breaking

5.6 Modelling in FD

Dual models

- Redefine problems by turning variables to values an vice versa
- E.g. instead of assigning workers to tasks we could assign tasks to workers
- Choose the more performant model or…
- Use both models with channeling constraints.

5.6 Modelling in FD

Example dual model of assignment problem

- Assign workers w1..w4 to tasks p1…p4 s.t. profit is maximised. Profit is given by

	P1	***P2***	***P3***	***P4***
W1	**7**	**1**	**3**	**4**
W2	**8**	**2**	**5**	**1**
W3	**4**	**3**	**7**	**2**
W4	**3**	**1**	**6**	**3**

5. Constraint Programming

1. Foundations
2. Constraint Domains
3. Consistency in finite domains
4. Propagation in FD
5. Labelling in FD
6. Modelling issues in FD

Inhalt

Vorlesung

1. Optimierungsprobleme
2. LP und MIP, Grundkonzepte und Anwendung
3. Constraintprogrammierung
4. UI Design

Übungen

- Modellierung in OPL
- Modellbasierte und imperative Programmierung OPL/Java
- Interaktive Optimierungssysteme

HCI in DSS

1. Generelle Usability Richtlinien (nach Nielson)
2. Richtlinien zur Usability in DSS (nach Frayman)
3. Werkzeuge in DSS (nach Prenzel)

Usability Guidelines (Nielson95)

1. Visibility of system status

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

→ Was ist zu vermeiden?

→ Definieren Sie Anforderungen für einen interaktiven GCP-Solver?

Usability Guidelines (Nielson95)

2. Match between system and the real world

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

→ Was ist zu vermeiden?

→ Anforderungen für GCP?

Usability Guidelines (Nielson95)

3. User control and freedom

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

→ Was ist zu vermeiden?

→ Anforderungen für GCP?

Usability Guidelines (Nielson95)

4. Consistency and standards

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

→ Was ist zu vermeiden?

→ Anforderungen für GCP?

Usability Guidelines (Nielson95)

5. Error prevention

Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

→ Was ist zu vermeiden?

→ Anforderungen für GCP?

Usability Guidelines (Nielson95)

6. Recognition rather than recall

Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

→ Was ist zu vermeiden?

→ Anforderungen für GCP?

Usability Guidelines (Nielson95)

7. Flexibility and efficiency of use

Accelerators — unseen by the novice user — may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

→ Was ist zu vermeiden?

→ Anforderungen für GCP?

Usability Guidelines (Nielson95)

8. Aesthetic and minimalist design

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

→ Was ist zu vermeiden?

→ Anforderungen für GCP?

Usability Guidelines (Nielson95)

9. Help users recognize, diagnose, and recover from errors

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

→ Was ist zu vermeiden?

→ Anforderungen für GCP?

Usability Guidelines (Nielson95)

10. Help and documentation

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

→ Was ist zu vermeiden?

→ Anforderungen für GCP?

Usability Guidelines (Nielson95)

Markplatz

(15 min Vorbereitung, je 1-5 min Besuche)

Formulieren Sie für Ihren GCP-Solver Use-Cases auf der Basis der Guidelines von Nielson.

Skizzieren Sie auf Papier, wie sich die Use-Cases und ggf Dialoge konkret darstellen

HCI in DSS

1. Generelle Usability Richtlinien (nach Nielson)
2. Richtlinien zur Usability in DSS (nach Frayman)
3. Werkzeuge in DSS (nach Prenzel)

HCI in DSS (Frayman)

1. Wenn die Entscheidungsunterstützungssaufgabe abgeschlossen ist, muss die resultierende Instanziierung konsistent sein.

→Wo sind beim modellbasierten, vollständigen Solver hier die Fehlerquellen?

HCI in DSS (Frayman)

2. Der Nutzer sollte ausdrücken können, dass

"diese Variable nicht diese Werte besitzen sollte". Oft ist nicht ein bestimmter Wert einer Variablen interessant, sondern es ist wichtig, dass ihr bestimmte Werte nicht zugewiesen

werden.

→Welcher Use-Cse ergibt sich hieraus?

→Formulieren Sie Vor-/Nachbedingung, Standardszenario, Abweichungen

HCI in DSS (Frayman)

3. Wenn der Nutzer eine Auswahl treffen möchte, die mit den vorherigen Zuweisungen inkonsistent ist, sollte das System dem Nutzer eine Liste mit den vorherigen Zuweisungen anbieten, die zurückgenommen werden müssen, um die neue Auswahl konsistent zu machen.

→beschrieben Sie den Use-Case als Szenario

→Wie könnte man das realisieren? Skizzieren Sie ein Sequenzdiagramm

HCI in DSS (Frayman)

4. Der Nutzer sollte in der Lage sein, eine Auswahl zu treffen und später wieder zurückzunehmen. Es ist ein fundamentaler Aspekt einer guten Benutzeroberfläche dass der Nutzer Aktionen rückgängig machen kann, denn dies erlaubt es, ohne Risiko mit Aktionen zu experimentieren.

→Welche Anforderung ergibt sich hieraus konkret?

→Entwerfen Sie das in einem Sequenzdiagramm und Klassendiagramm

HCI in DSS (Frayman)

5. Der Nutzer muss eine Auswahl in beliebiger Reihenfolge treffen können. Auch dies ist ein fundamentaler Aspekt einer guten Benutzerschnittstelle, denn unterschiedliche Nutzer können eine unterschiedliche Reihenfolge von Aktionenbevorzugen.

→ Was ist also zu vermeiden?

HCI in DSS (Frayman)

6. Der Nutzer sollte keine Auswahl treffen können, die in eine Sackgasse führt, d.h. in eine Situation, in der eine Lösung aufgrund der bisherigen Zuweisungen nicht gefunden werden kann.

→Wie kann man das im GCP-Solver realisieren? Formulieren Sie das Szenario und skizzieren Sie ein entspr. Sequenzdiagramm

HCI in DSS (Frayman)

7. Die Antwortzeit sollte für alle Funktionen kurz sein. Eine Funktion sollte weniger als 1 Sekunde benötigen, um den Gedankenfluss des Nutzers nicht zu unterbrechen. Um die Aufmerksamkeit des Nutzers zu beizubehalten, sollte eine Wartezeit von 10 Sekunden nicht überschritten werden.

→ Was kann der Programmierer tun, um das zu erreichen?

HCI in DSS

1. Generelle Usability Richtlinien (nach Nielson)
2. Richtlinien zur Usability in DSS (nach Frayman)
3. Werkzeuge in DSS (nach Prenzel)

HCI in DSS

1. TODO