DM87
SCHEDULING,
TIMETABLING AND ROUTING

DM87

Marco Chiarandini

Part I

Course Introduction. Scheduling: Terminology and Classification

---

Outline

Outline

Course Presentation

- Communication media
  - Blackboard (for private communications)
    Mail, Fora, Blog, Grades, Documents (photocopies),
  - Web-site `http://www.imada.sdu.dk/~marco/DM87/`
    Lecture plan, Syllabus, Links, Exam documents
- 40 hours of lectures + work at the exam project
- Schedule:
  1. Lectures:
     Mondays 12:00-13:45, Thursdays 8:15-10:00
     Weeks 5-10, 13-16
     Last lecture (preliminary date): Thursday, April 17
  2. Exam: June

Course Content

- Review of Optimization Methods:
  - Mathematical Programming,
  - Constraint Programming,
  - Heuristics
  - Problem Specific Algorithms (Dynamic Programming, Branch and Bound)
- Introduction to Scheduling, Terminology, Classification.
  - Single Machine Models
  - Parallel Machine Models
  - Flow Shops and Flexible Flow Shops
  - Job Shops, Open Shops
- Introduction to Timetabling, Terminology, Classification
  - Interval Scheduling, Reservations
  - Educational Timetabling
  - Workforce and Employee Timetabling
  - Transportation Timetabling
- Introduction to Vehicle Routing, Terminology, Classification
  - Capacited Vehicle Routing
  - Vehicle Routing with Time Windows

Evaluation

Final Assessment (10 ECTS)

- Oral exam: 30 minutes + 5 minutes defense project
  meant to assess the base knowledge
- Group project:
  free choice of a case study among few proposed ones
  Deliverables: program + report
  meant to assess the ability to apply

Course Material

- Literature
  - **Text book**: M.L. Pinedo, Planning and Scheduling in Manufacturing and Services; Springer Series in Operations Research and Financial Engineering, 2005. (388 DKK)
  - **Supplementary book**: M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems; 2nd ed., Prentice Hall, 2002.
  - **Supplementary book**: P. Toth, D. Vigo, eds. The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.
  - **Supplementary Articles**: will be indicated during the course
- Slides
- Class exercises (participatory)

Useful Previous Knowledge for this Course

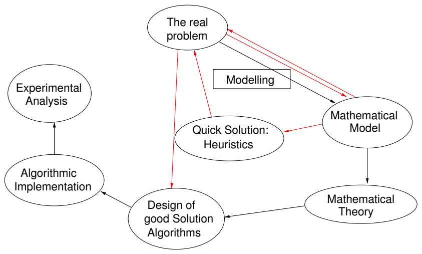
- Algorithms and data structures
- Programming A and B
- Networks and Integer Programming
- Heuristics for Optimization
- Software Methodologies and Engineering

Course Goals and Project Plan

How to Tackle Real-life Optimization Problems:

- Formulate (mathematically) the problem
- Model the problem and recognize possible similar problems
- Search in the literature (or in the Internet) for:
  - complexity results (is the problem NP-hard?)
  - solution algorithms for original problem
  - solution algorithms for simplified problem
- Design solution algorithms
- Test experimentally with the goals of:
  - configuring
  - tuning parameters
  - comparing
  - studying the behavior (prediction of scaling and deviation from optimum)

The problem Solving Cycle



Outline

Scheduling

- Manufacturing
  - Project planning
  - Single, parallel machine and job shop systems
  - Flexible assembly systems
    Automated material handling (conveyor system)
  - Lot sizing
  - Supply chain planning
- Services

⇒ different algorithms

Problem Definition

Constraints
Activities ⟶ Resources
Objectives

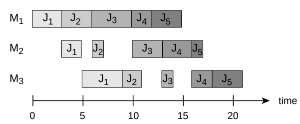
Problem Definition
Given: a set of **jobs** $\mathcal{J} = \{J_1, \ldots, J_n\}$ that have to be processed by a set of **machines** $\mathcal{M} = \{M_1, \ldots, M_m\}$
Find: a **schedule**,
i.e., a mapping of jobs to machines and processing times subject to feasibility and optimization constraints.

Notation:
n, j, k jobs
m, i, h machines

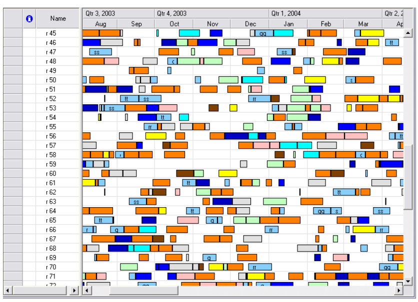
Visualization

Scheduling are represented by Gantt charts

- machine-oriented



- or job-oriented
  ...



Data Associated to Jobs

- Processing time p_{ij}
- Release date r_j
- Due date d_j (called deadline, if strict)
- Weight w_j

- A job J_j may also consist of a number n_j of operations $O_{j1}, O_{j2}, \ldots, O_{jn_j}$ and data for each operation.
- Associated to each operation a set of machines $\mu_{jl} \subseteq \mathcal{M}$

Data that depend on the schedule (dynamic)
- Starting times S_{ij}
- Completion time C_{ij}, C_j

Problem Classification

A scheduling problem is described by a triplet $\alpha \mid \beta \mid \gamma$.

- α machine environment (one or two entries)
- β job characteristics (none or multiple entry)
- γ objective to be minimized (one entry)

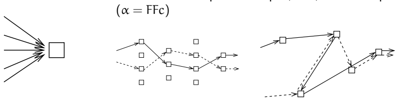
[R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1979): Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 4, 287-326.]

The $\alpha \mid \beta \mid \gamma$ Classification Scheme

Machine Environment $\quad \boldsymbol{\alpha_1\alpha_2} \mid \beta_1 \ldots \beta_{13} \mid \gamma$

- single machine/multi-machine ($\alpha_1 = \alpha_2 = 1$ or $\alpha_2 = m$)
- parallel machines: identical ($\alpha_1 = P$), uniform p_j/v_i ($\alpha_1 = Q$), unrelated p_j/v_{ij} ($\alpha_1 = R$)
- multi operations models: Flow Shop ($\alpha_1 = F$), Open Shop ($\alpha_1 = O$), Job Shop ($\alpha_1 = J$), Mixed (or Group) Shop ($\alpha_1 = X$)

Single Machine — Flexible Flow Shop ($\alpha = FFc$) — Open, Job, Mixed Shop



The $\alpha \mid \beta \mid \gamma$ Classification Scheme

Job Characteristics $\quad \alpha_1\alpha_2 \mid \boldsymbol{\beta_1 \ldots \beta_{13}} \mid \gamma$

- $\beta_1 = \mathrm{prmp}$ presence of preemption (resume or repeat)
- β_2 precedence constraints between jobs (with $\alpha = P, F$) acyclic digraph $G = (V, A)$
  - $\beta_2 = \mathrm{prec}$ if G is arbitrary
  - $\beta_2 = \{\mathrm{chains}, \mathrm{intree}, \mathrm{outtree}, \mathrm{tree}, \mathrm{sp\text{-}graph}\}$
- $\beta_3 = r_j$ presence of release dates
- $\beta_4 = p_j = p$ preprocessing times are equal
- ($\beta_5 = d_j$ presence of deadlines)
- $\beta_6 = \{\mathrm{s\text{-}batch}, \mathrm{p\text{-}batch}\}$ batching problem
- $\beta_7 = \{s_{jk}, s_{jik}\}$ sequence dependent setup times

The $\alpha \mid \beta \mid \gamma$ Classification Scheme

Job Characteristics (2) $\quad \alpha_1\alpha_2 \mid \boldsymbol{\beta_1 \ldots \beta_{13}} \mid \gamma$

- $\beta_8 = \mathrm{brkdwn}$ machines breakdowns
- $\beta_9 = M_j$ machine eligibility restrictions (if $\alpha = Pm$)
- $\beta_{10} = \mathrm{prmu}$ permutation flow shop
- $\beta_{11} = \mathrm{block}$ presence of blocking in flow shop (limited buffer)
- $\beta_{12} = \mathrm{nwt}$ no-wait in flow shop (limited buffer)
- $\beta_{13} = \mathrm{recrc}$ Recirculation in job shop

The $\alpha \mid \beta \mid \gamma$ Classification Scheme

Objective (always $f(C_j)$) $\quad \alpha_1\alpha_2 \mid \beta_1\beta_2\beta_3\beta_4 \mid \boldsymbol{\gamma}$

- Lateness $L_j = C_j - d_j$
- Tardiness $T_j = \max\{C_j - d_j, 0\} = \max\{L_j, 0\}$
- Earliness $E_j = \max\{d_j - C_j, 0\}$
- Unit penalty $U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & \text{otherwise} \end{cases}$

The $\alpha \mid \beta \mid \gamma$ Classification Scheme

Objective $\quad \alpha_1\alpha_2 \mid \beta_1\beta_2\beta_3\beta_4 \mid \boldsymbol{\gamma}$

- Makespan: Maximum completion $C_{max} = \max\{C_1, \ldots, C_n\}$ tends to max the use of machines
- Maximum lateness $L_{max} = \max\{L_1, \ldots, L_n\}$
- Total completion time $\sum C_j$ (flow time)
- Total weighted completion time $\sum w_j \cdot C_j$ tends to min the av. num. of jobs in the system, ie, work in progress, or also the throughput time
- Discounted total weighted completion time $\sum w_j(1 - e^{-rC_j})$
- Total weighted tardiness $\sum w_j \cdot T_j$
- Weighted number of tardy jobs $\sum w_j U_j$

All regular functions (nondecreasing in $C_1, \ldots, C_n$) except E_i

The $\alpha \mid \beta \mid \gamma$ Classification Scheme

Other Objectives $\quad \alpha_1\alpha_2 \mid \beta_1\beta_2\beta_3\beta_4 \mid \boldsymbol{\gamma}$

Non regular objectives

- Min $\sum w'_j E_j + \sum w''_j T_j$ (just in time)
- Min waiting times
- Min set up times/costs
- Min transportation costs

Exercises

Scheduling Tasks in a Central Processing Unit (CPU) [Ex. 1.1.3, textbook]

- Multitasking operating system
- Schedule time that the CPU devotes to the different programs
- Exact processing time unknown but an expected value might be known
- Each program has a certain priority level
- Minimize the time expected sum of the weighted completion times for all tasks
- Tasks are often sliced into little pieces. They are then rotated such that low priority tasks of short duration do not stay for ever in the system.

Exercises

Gate Assignment at an Airport [Ex. 1.1.2, textbook]

- Airline terminal at a airport with dozes of gates and hundreds of arrivals each day.
- Gates and Airplanes have different characteristics
- Airplanes follow a certain schedule
- During the time the plane occupies a gate, it must go through a series of operations
- There is a scheduled departure time (due date)
- Performance measured in terms of on time departures.

Solutions

Distinction between

- sequence
- schedule
- scheduling policy

Feasible schedule

A schedule is feasible if no two time intervals overlap on the same machine, and if it meets a number of problem specific constraints.

Optimal schedule

A schedule is optimal if it minimizes the given objective.

Classes of Schedules

Nondelay schedule

A feasible schedule is called nondelay if no machine is kept idle while an operation is waiting for processing.

There are optimal schedules that are nondelay for most models with regular objective function.

Active schedule

A feasible schedule is called active if it is not possible to construct another schedule by changing the order of processing on the machines and having at least one operation finishing earlier and no operation finishing later.

There exists for $Jm||\gamma$ (γ regular) an optimal schedule that is active.

nondelay $\Rightarrow$ active

active $\not\Rightarrow$ nondelay

Semi-active schedule

A feasible schedule is called semi-active if no operation can be completed earlier without changing the order of processing on any one of the machines.

Part II

Complexity hierarchies, PERT, Mathematical Programming

Outline

Outline

Outline

Complexity Hierarchy

A problem $\mathcal{A}$ is reducible to $\mathcal{B}$ if a procedure for $\mathcal{B}$ can be used also for $\mathcal{A}$.

Ex: $1||\sum C_j \;\propto\; 1||\sum w_j C_j$

Complexity hierarchy describes relationships between different scheduling problems.

Interest in characterizing the borderline: polynomial vs NP-hard problems

Problems Involving Numbers

Partition

- **Input:** finte set A and a size $s(a) \in \mathbf{Z}^+$ for each $a \in A$
- **Question:** is there a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)?$$

3-Partition

- **Input:** set A of $3m$ elements, a bound $B \in \mathbf{Z}^+$, and a size $s(a) \in \mathbf{Z}^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = mB$
- **Question:** can A be partitioned into m diskoint sets $A_1, \ldots, A_m$ such that for $1 \le i \le m$, $\sum_{a \in A_i} s(a) = B$ (note that each A_i must therefore contain exactly three elements from A)?

Complexity Hierarchy of Problems

TABLE D.1 POLYNOMIAL TIME SOLVABLE PROBLEMS

Single machine	Parallel machines	Shops
$1 \mid r_j, p_j = 1, prec \mid \sum C_j$ $1 \mid r_j, prmp \mid \sum C_j$ $1 \mid tree \mid \sum w_j C_j$ $1 \mid prec \mid L_{max}$ $1 \mid r_j, prmp, prec \mid L_{max}$ $1 \mid\mid \sum U_j$ $1 \mid r_j, prmp \mid \sum U_j$ $1 \mid r_j, p_j = 1 \mid \sum w_j U_j$ $1 \mid r_j, p_j = 1 \mid \sum w_j T_j$	$P2 \mid p_j = 1, prec \mid L_{max}$ $P2 \mid p_j = 1, prec \mid \sum C_j$ $Pm \mid p_j = 1, tree \mid C_{max}$ $Pm \mid prmp, tree \mid C_{max}$ $Pm \mid p_j = 1, outtree \mid \sum C_j$ $Pm \mid p_j = 1, intree \mid L_{max}$ $Pm \mid prmp, intree \mid L_{max}$ $Q2 \mid prmp, prec \mid C_{max}$ $Q2 \mid r_j, prmp, prec \mid L_{max}$ $Qm \mid r_j, p_j = 1 \mid C_{max}$ $Qm \mid r_j, p_j = 1 \mid \sum C_j$ $Qm \mid prmp \mid \sum C_j$ $Qm \mid p_j = 1 \mid \sum w_j C_j$ $Qm \mid p_j = 1 \mid L_{max}$ $Qm \mid prmp \mid \sum U_j$ $Qm \mid p_j = 1 \mid \sum w_j U_j$ $Qm \mid p_j = 1 \mid \sum w_j T_j$ $Rm \mid\mid \sum C_j$ $Rm \mid r_j, prmp \mid L_{max}$	$O2 \mid\mid C_{max}$ $Om \mid r_j, prmp \mid L_{max}$ $F2 \mid block \mid C_{max}$ $F2 \mid nwt \mid C_{max}$ $Fm \mid p_{ij} = p_j \mid \sum C_j$ $Fm \mid p_{ij} = p_j \mid L_{max}$ $Fm \mid p_{ij} = p_j \mid \sum U_j$ $J2 \mid\mid C_{max}$

TABLE D.2 NP-HARD PROBLEMS IN THE ORDINARY SENSE

Single machine	Parallel machines	Shops
$1 \mid\mid \sum w_j U_j$ (*) $1 \mid r_j, prmp \mid \sum w_j U_j$ (*) $1 \mid\mid \sum T_j$ (*)	$P2 \mid\mid C_{max}$ (*) $P2 \mid r_j, prmp \mid \sum C_j$ $P2 \mid\mid \sum w_j C_j$ (*) $P2 \mid r_j, prmp \mid \sum U_j$ $Pm \mid prmp \mid \sum w_j C_j$ $Qm \mid\mid \sum w_j C_j$ (*) $Rm \mid r_j \mid C_{max}$ (*) $Rm \mid\mid \sum w_j U_j$ (*) $Rm \mid prmp \mid \sum w_j U_j$	$O2 \mid prmp \mid \sum C_j$ $O3 \mid\mid C_{max}$ $O3 \mid prmp \mid \sum w_j U_j$

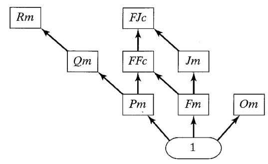
TABLE D.3 STRONGLY NP-HARD PROBLEMS

Single machine	Parallel machines	Shops
$1 \mid s_{jk} \mid C_{max}$	$P2 \mid chains \mid C_{max}$	$F2 \mid r_j \mid C_{max}$
	$P2 \mid chains \mid \sum C_j$	$F2 \mid r_j, prmp \mid C_{max}$
$1 \mid r_j \mid \sum C_j$	$P2 \mid prmp, chains \mid \sum C_j$	$F2 \parallel \sum C_j$
$1 \mid prec \mid \sum C_j$	$P2 \mid p_j = 1, tree \mid \sum w_j C_j$	$F2 \mid prmp \mid \sum C_j$
$1 \mid r_j, prmp, tree \mid \sum C_j$		$F2 \parallel L_{max}$
$1 \mid r_j, prmp \mid \sum w_j C_j$	$R2 \mid prmp, chains \mid C_{max}$	$F2 \mid prmp \mid L_{max}$
$1 \mid r_j, p_j = 1, tree \mid \sum w_j C_j$		
$1 \mid p_j = 1, prec \mid \sum w_j C_j$		$F3 \parallel C_{max}$
		$F3 \mid prmp \mid C_{max}$
$1 \mid r_j \mid L_{max}$		$F3 \mid nwt \mid C_{max}$
$1 \mid r_j \mid \sum U_j$		$O2 \mid r_j \mid C_{max}$
$1 \mid p_j = 1, chains \mid \sum U_j$		$O2 \parallel \sum C_j$
		$O2 \mid prmp \mid \sum w_j C_j$
$1 \mid r_j \mid \sum T_j$		$O2 \parallel L_{max}$
$1 \mid p_j = 1, chains \mid \sum T_j$		
$*1 \parallel \sum w_j T_j$		$O3 \mid prmp \mid \sum C_j$
		$J2 \mid recrc \mid C_{max}$
		$J3 \mid p_{ij} = 1, recrc \mid C_{max}$

http://www.mathematik.uni-osnabrueck.de/research/OR/class/

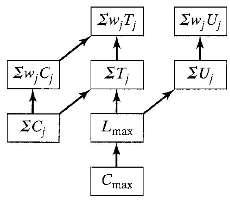
Complexity Hierarchy

Elementary reductions for machine environment

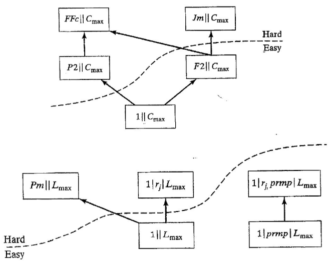


Complexity Hierarchy

Elementary reductions for regular objective functions



Complexity Hierarchy of Problems



Outline

3. Resume
4. Complexity Hierarchy
5. CPM/PERT
6. Mathematical Programming

Project Planning

Milwaukee General Hospital Project

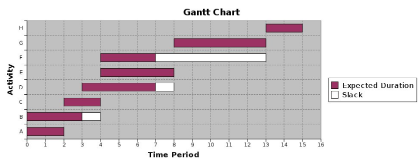
Activity	Description	Immediate Predecessor	*Duration*
A	Build internal components	–	2
B	Modify roof and floor	–	3
C	Construct collection stack	A	2
D	Pour concrete and install frame	A,B	4
E	Build high-temperature burner	C	4
F	Install pollution control system	C	3
G	Install air pollution device	D,E	5
H	Inspect and test	F,G	2

Project Planning

Milwaukee General Hospital Project

Activity	Description	Immediate Predecessor	*Duration*	EST	EFT	LST	LFT	Slack
A	Build internal components	–	2	0	2	0	2	0
B	Modify roof and floor	–	3	0	3	1	4	1
C	Construct collection stack	A	2	2	4	2	4	0
D	Pour concrete and install frame	A,B	4	3	7	6	10	3
E	Build high-temperature burner	C	4	4	8	6	10	2
F	Install pollution control system	C	3	4	7	10	13	6
G	Install air pollution device	D,E	5	8	13	8	13	0
H	Inspect and test	F,G	2	13	15	13	15	0
		Expected project duration			*15*			

Project Planning



Project Planning

Milwaukee General Hospital Projec			Expecte d						Time Estimates			Activity Varianc
Activity	**Description**	**Immediate Predecessor**	*(a+4m+b)/(*	**EST**	**EFT**	**LST**	**LFT**	**Slack**	*a*	*m*	*b*	*((b-a)/6)^2*
A	Build internal components	–	2	0	2	0	2	0	1	2	3	0.1111
B	Modify roof and floor	–	3	0	3	1	4	1	2	3	4	0.1111
C	Construct collection stack	A	2	2	4	2	4	0	1	2	3	0.1111
D	’our concrete and install fram	A,B	4	3	7	4	8	1	2	4	6	0.4444
E	3uild high-temperature burne	C	4	4	8	4	8	0	1	4	7	1.0000
F	nstall pollution control systen	C	3	4	7	10	13	6	1	2	9	1.7778
G	Install air pollution device	D,E	5	8	13	8	13	0	3	4	11	1.7778
H	Inspect and test	F,G	2	13	15	13	15	0	1	2	3	0.1111
		Expected project duration		*15*				*Variance of project duration*				*3.1111*

Outline

3. Resume
4. Complexity Hierarchy
5. CPM/PERT
6. Mathematical Programming

Linear, Integer, Nonlinear Programming

program = optimization problem

$$\begin{aligned} \min\ & f(x) \\ & g_i(x) = 0, \quad i = 1, 2, \ldots, k \\ & h_j(x) \leq 0, \quad j = 1, 2, \ldots, m \\ & x \in \mathbf{R}^n \end{aligned}$$

general (nonlinear) program (NLP)

$$\begin{aligned} \min\ & c^T x \\ & Ax = a \\ & Bx \leq b \\ & x \geq 0 \\ & (x \in \mathbf{R}^n) \end{aligned}$$

linear program (LP)

$$\begin{aligned} \min\ & c^T x \\ & Ax = a \\ & Bx \leq b \\ & x \geq 0 \\ & (x \in \mathbf{Z}^n) \\ & (x \in \{0,1\}^n) \end{aligned}$$

integer (linear) program (IP, MIP)

Linear Programming

Linear Program in standard form

$$\begin{aligned} \min\ & c_1 x_1 + c_2 x_2 + \ldots c_n x_n \\ \text{s.t.}\ & a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n = b_1 \\ & a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n = b_2 \\ & \vdots \\ & a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n = b_n \\ & x_1, x_2, \ldots, x_n \geq 0 \end{aligned}$$

$$\begin{aligned} \min\ & c^T x \\ & Ax = b \\ & x \geq 0 \end{aligned}$$

Historic Roots

- 1939 L. V. Kantorovitch: Foundations of linear programming (Nobel Prize 1975)
- George J. Stigler's 1945 (Nobel Prize 1982) "Diet Problem": "the first linear program"
  find the cheapest combination of foods that will
  satisfy the daily requirements of a person
  Army's problem had 77 unknowns and 9 constraints.
  http://www.mcs.anl.gov/home/otc/Guide/CaseStudies/diet/index.h
- 1947 G. B. Dantzig: Invention of the simplex algorithm
- Founding fathers:
  - 1950s Dantzig: Linear Programming 1954, the Beginning of IP G. Dantzig, D.R. Fulkerson, S. Johnson TSP with 49 cities
  - 1960s Gomory: Integer Programming

LP Theory

- Max-Flow Min-Cut Theorem
  The maximal (s,t)-flow in a capaciatetd network is equal to the minimal capacity of an (s,t)-cut
- The Duality Theorem of Linear Programming

$$\begin{aligned} \max\ & c^T x \\ & Ax \leq b \\ & x \geq 0 \end{aligned} \qquad\qquad \begin{aligned} \min\ & y^T b \\ & y^T A \geq c^T \\ & y \geq 0 \end{aligned}$$

  If feasible solutions to both the primal and the dual problem in a pair of dual LP problems exist, then there is an optimum solution to both systems and the optimal values are equal.

LP Theory

- Max-Flow Min-Cut Theorem
  does not hold if several source-sink relations are given (multicommodity flow)
- The Duality Theorem of Integer Programming

$$\begin{aligned} \max\ & c^T x \\ & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbf{Z}^n \end{aligned} \qquad \leq \qquad \begin{aligned} \min\ & y^T b \\ & y^T A \geq c^T \\ & y \geq 0 \\ & y \in \mathbf{Z}^n \end{aligned}$$

LP Solvability

- Linear programs can be solved in polynomial time with
  the Ellipsoid Method (Khachiyan, 1979)
  Interior Point Methods (Karmarkar, 1984, and others)
- Open: is there a strongly polynomial time algorithm for the solution of LPs?
- Certain variants of the Simplex Algorithm run - under certain conditions - in expected polynomial time (Borgwardt, 1977...)
- Open: Is there a polynomial time variant of the Simplex Algorithm?

IP Solvability

- Theorem
  Integer, 0/1, and mixed integer programming are NP-hard.
- Consequence
  - special cases
  - special purposes
  - heuristics

- Algorithms for the solution of nonlinear programs
- Algorithms for the solution of linear programs
  - 1) Fourier-Motzkin Elimination (hopeless)
  - 2) The Simplex Method (good, above all with duality)
  - 3) The Ellipsoid Method (total failure)
  - 4) Interior-Point/Barrier Methods (good)
- Algorithms for the solution of integer programs
  - 1) Branch & Bound
  - 2) Cutting Planes

Algorithms for nonlinear programming

- Iterative methods that solve the equation and inequality systems representing the necessary local optimality conditions.
- Steepest descent (Kuhn-Tucker sufficient conditions)
- Newton method
- Subgradient method

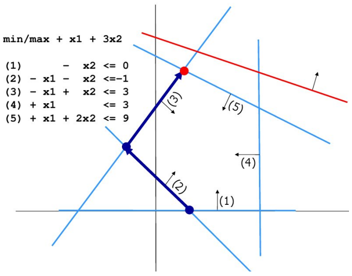
Algorithms for linear programming

The Simplex Method

- Dantzig, 1947: primal Simplex Method
- Lemke, 1954; Beale, 1954: dual Simplex Method
- Dantzig, 1953: revised Simplex Method
- ....
- Underlying Idea: Find a vertex of the set of feasible LP solutions (polyhedron) and move to a better neighbouring vertex, if possible.

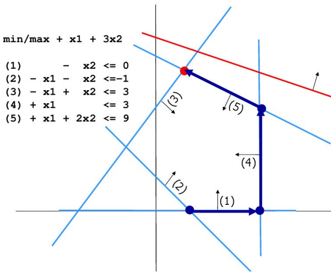
The simplex method

```
min/max + x1 + 3x2

(1)        -  x2 <= 0
(2) - x1 -   x2 <=-1
(3) - x1 +   x2 <= 3
(4) + x1        <= 3
(5) + x1 + 2x2 <= 9
```



The simplex method

```
min/max + x1 + 3x2

(1)        -  x2 <= 0
(2) - x1 -   x2 <=-1
(3) - x1 +   x2 <= 3
(4) + x1        <= 3
(5) + x1 + 2x2 <= 9
```



The simplex method

Hirsch Conjecture
If P is a polytope of dimension n with m facets then every vertex of P can be reached from any other vertex of P on a path of length at most m-n.

In the example before: m=5, n=2 and m-n=3, conjecture is true.

At present, not even a polynomial bound on the path length is known.
Best upper bound: Kalai, Kleitman (1992): The diameter of the graph of an n-dimensional polyhedron with m facets is at most m(log n+1).
Lower bound: Holt, Klee (1997): at least m-n (m, n large enough).

Algorithms for Integer Programming

special „simple" combinatorial optimization problems Finding a:

- minimum spanning tree
- shortest path
- maximum matching
- maximal flow through a network
- cost-minimal flow
- ...

solvable in polynomial time by special purpose algorithms

Algorithms for Integer Programs

special „hard" combinatorial optimization problems

- traveling salesman problem
- location and routing
- set-packing, partitioning, -covering
- max-cut
- linear ordering
- scheduling (with a few exceptions)
- node and edge colouring
- ...

NP-hard (in the sense of complexity theory)
The most successful solution techniques employ linear programming.

Algorithms for Integer Programs

- 1) Branch & Bound
- 2) Cutting Planes

Branch & cut, Branch & Price (column generation), Branch & Cut & Price

Summary

- We can solve today explicit LPs with
  - up to 500,000 of variables and
  - up to 5,000,000 of constraints routinely

  in relatively short running times.
- We can solve today structured implicit LPs (employing column generation and cutting plane techniques) in special cases with hundreds of million (and more) variables and almost infinitely many constraints in acceptable running times. (Examples: TSP, bus circulation in Berlin)

[Martin Grötschel, Block Course at TU Berlin, "Combinatorial Optimization at Work", 2005 http://co-at-work.zib.de/berlin/]

Part III

Mathematical Programming Formulations, Constraint Programming

Outline

Modeling: Mixed Integer Formulations

- Transportation Problem
- Weighted Bipartite Matching Problem (if $m = n \Rightarrow$ assignment)

Set Covering

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i \\ & x_j \in \{0,1\} \end{aligned}$$

Set Partitioning

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j = 1 \quad \forall i \\ & x_j \in \{0,1\} \end{aligned}$$

Set Packing

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \leq 1 \quad \forall i \\ & x_j \in \{0,1\} \end{aligned}$$

Traveling Salesman Problem

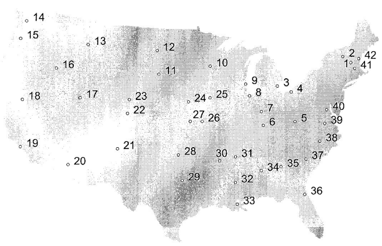


Figure 3.1 Locations of the 42 cities.

Traveling Salesman Problem

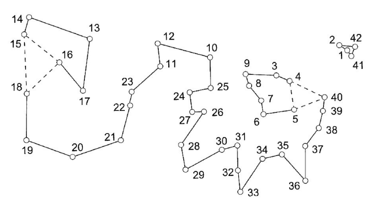


Figure 3.2 Solution of the initial LP relaxation.

Traveling Salesman Problem

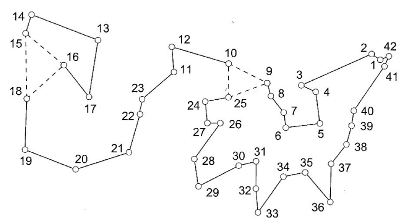


Figure 3.3 LP solution after three subtour constraints.

Traveling Salesman Problem

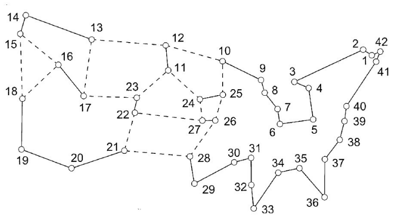


Figure 3.4 LP solution satisfying all subtour constraints.

Traveling Salesman Problem

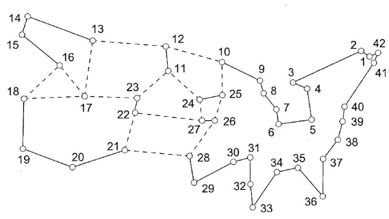


Figure 3.7 What is wrong with this vector?

Traveling Salesman Problem

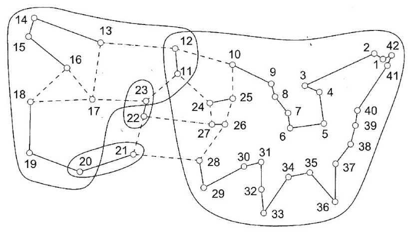


Figure 3.8 A violated comb.

Traveling Salesman Problem

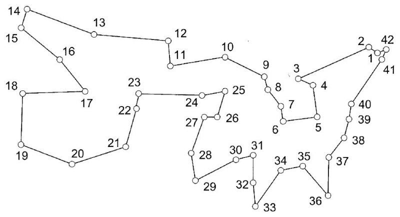


Figure 3.9 An optimal tour through 42 cities.

minimize $c^T x$ subject to

$0 \le x_e \le 1$ for all edges e,

$\sum (x_e : v \text{ is am emd of } e) = 2$ for all cities v,

$\sum (x_e : e \text{ has one end in } S \text{ and one end not in } S) \ge 2$
for all nonempty proper subsets S of cities,

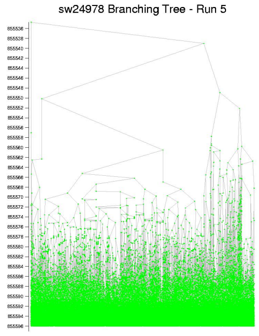
$\sum_{i=0}^{i=3} (\sum (x_e : e \text{ has one end in } S_i \text{ and one end not in } S_i) \ge 10$,
for any comb

24,978 Cities

solved by LK-heuristic and prooved optimal by branch and cut

10 months of computation on a cluster of 96 dual processor Intel Xeon 2.8 GHz workstations

`http://www.tsp.gatech.edu`



24,978 Cities

solved by LK-heuristic and prooved optimal by branch and cut

10 months of computation on a cluster of 96 dual processor Intel Xeon 2.8 GHz workstations

`http://www.tsp.gatech.edu`

Modeling: Mixed Integer Formulations

- Formulation for $Qm|p_j = 1|\sum h_j(C_j)$ and relation with transportation problems
- Totally unimodular matrices and sufficient conditions for total unimodularity i) two ones per column and ii) consecutive 1's property
- Formulation of $1|prec|\sum w_j C_j$ and $Rm||\sum C_j$ as weighted bipartite matching and assignment problems.
- Formulation of set covering, set partitioning and set packing
- Formulation of Traveling Salesman Problem
- Formulation of $1|prec|\sum w_j C_j$ and how to deal with disjunctive constraints
- Graph coloring

Outline

Special Purpose Algorithms

Dynamic programming
procedure based on divide and conquer

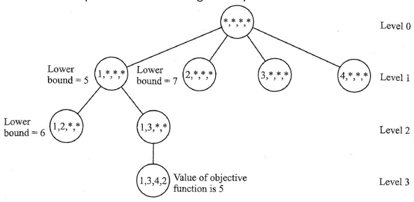
Based on principle of optimality the completion of an optimal sequence of decisions must be optimal

- Break down the problem in stages at which the decisions take place
- Find a recurrence relation that takes us backward (forward) from one stage to the previous (next)

In scheduling, this can be typically done only for objectives that are sequence independent (eg, the makespan).

Special Purpose Algorithms

Branch and Bound
divide and conquer + lower bounding technique



[Jens Clausen. (2003)]

Outline

Constraint Satisfaction Problem

- **Input:**
  - a set of variables $X_1, X_2, \ldots, X_n$
  - each variable has a non-empty domain D_i of possible values
  - a set of constraints. Each constraint C_i involves some subset of the variables and specify the allowed combination of values for that subset.
    [A constraint C on variables X_i and X_j, $C(X_i, X_j)$, defines the subset of the Cartesian product of variable domains $D_i \times D_j$ of the consistent assignments of values to variables. A constraint C on variables X_i, X_j is satisfied by a pair of values v_i, v_j if $(v_i, v_j) \in C(X_i, X_j)$.]
- **Task:**
  - find an assignment of values to all the variables $\{X_i = v_i, X_j = v_j, \ldots\}$
  - such that it is consistent, that is, it does not violate any constraint

If assignments are not all equally good, but some are preferable this is reflected in an objective function.

Search Problem

- initial state: the empty assignment {} in which all variables are unassigned
- successor function: a value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables
- goal test: the current assignment is complete
- path cost: a constant cost

Two search paradigms:

- search tree of depth n
- complete state formulation: local search

Types of Variables and Values

- Discrete variables with finite domain:
  complete enumeration is $O(d^n)$
- Discrete variables with infinite domains:
  Impossible by complete enumeration.
  Instead a constraint language (constraint logic programming and constraint reasoning)
  Eg, project planning.

  $$S_j + p_j \le S_k$$

  NB: if only linear constraints, then integer linear programming
- variables with continuous domains
  NB: if only linear constraints or convex functions then mathematical programming

Types of constraints

- Unary constraints
- Binary constraints (constraint graph)
- Higher order (constraint hypergraph)
  Eg, `Alldiff()`
  Every higher order constraint can be reconduced to binary
  (you may need auxiliary constraints)
- Preference constraints
  cost on individual variable assignments

General Purpose Solution Algorithms

Search algorithms
tree with branching factor at the top level nd
at the next level $(n-1)d$.
The tree has $n! \cdot d^n$ even if only d^n possible complete assignments.

- CSP is commutative in the order of application of any given set of action. (the order of the assignment does not influence)
- Hence we can consider search algs that generate successors by considering possible assignments for only a single variable at each node in the search tree.

Backtracking search
depth first search that chooses one variable at a time and backtracks when a variable has no legal values left to assign.

Backtrack Search

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment according to CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure
```


Backtrack Search

- No need to copy solutions all the times but rather extensions and undo extensions
- Since CSP is standard then the alg is also standard and can use general purpose algorithms for initial state, successor function and goal test.
- Backtracking is uninformed and complete. Other search algorithms may use information in form of heuristics

General Purpose backtracking methods

1) Which variable should we assign next, and in what order should its values be tried?
2) What are the implications of the current variable assignments for the other unassigned variables?
3) When a path fails – that is, a state is reached in which a variable has no legal values can the search avoid repeating this failure in subsequent paths?

Which variable should we assign next, and in what order should its values be tried?

- Select-Initial-Unassigned-Variable
  degree heuristic (reduces the branching factor) also used as tied breaker
- Select-Unassigned-Variable
  Most constrained variable (DSATUR) = fail-first heuristic
  = Minimum remaining values (MRV) heuristic (speeds up pruning)
- Order-Domain-Values
  least-constraining-value heuristic (leaves maximum flexibility for subsequent variable assignments)

NB: If we search for all the solutions or a solution does not exists, then the ordering does not matter.

What are the implications of the current variable assignments for the other unassigned variables?

Propagating information through constraints

- Implicit in Select-Unassigned-Variable
- Forward checking (coupled with MRV)
- Constraint propagation
  - arc consistency: force all (directed) arcs uv to be consistent: $\exists$ a value in $D(v)$: $\forall$ values in $D(u)$, otherwise detects inconsistency

    can be applied as preprocessing or as propagation step after each assignment (MAC, Maintaining Arc Consistency)

    Applied repeatedly
  - k-consistency: if for any set of $k-1$ variables, and for any consistent assignment to those variables, a consistent value can always be assigned to any k-th variable.

    determining the appropriate level of consistency checking is mostly an empirical science.

Arc Consistency Algorithm: AC-3

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	R	B	G	R B	R G B	B	R G B
After *V=blue*	R	B	G	R	B		R G B

Figure 5.6 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.

Arc Consistency Algorithm: AC-3

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

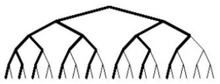
  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS[X_i] **do**
        add (X_k, X_i) to *queue*

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff we remove a value
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN[X_i] **do**
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint between X_i and X_j
      then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  return *removed*

Incomplete Search

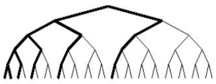
General purpose algorithms:

Credit-based search:



credit(16)

Limited Discrepancy Search:



lds(1)

Limited Discrepancy Search

- A discrepancy is a branch against the value of an heuristic
- Ex: count one discrepancy if second best is chosen
  count two discrepancies either if third best is chosen or twice the second best is chosen
- Explore the tree in order of an increasing number of discrepancies

Handling special constraints (higher order constraints)

Special purpose algorithms

- `Alldiff`
  - for m variables and n values cannot be satisfied if $m > n$,
  - consider first singleton variables
  - propagation based on bipartite matching considerations
- Resource Constraint `atmost`
  - check the sum of minimum values of single domains
    delete maximum values if not consistent with minimum values of others.
  - for large integer values not possible to represent the domain as a set of integers but rather as bounds.
    Then bounds propagation: Eg,
    $\texttt{Flight271} \in [0, 165]$ and $\texttt{Flight272} \in [0, 385]$
    $\texttt{Flight271} + \texttt{Flight272} \in [420, 420]$
    $\texttt{Flight271} \in [35, 165]$ and $\texttt{Flight272} \in [255, 385]$

When a path fails – that is, a state is reached in which a variable has no legal values can the search avoid repeating this failure in subsequent paths?

Backtracking-Search

- chronological backtracking, the most recent decision point is revisited
- backjumping, backtracks to the most recent variable in the conflict set (set of previously assigned variables connected to X by constraints).

  every branch pruned by backjumping is also pruned by forward checking

  idea remains: backtrack to reasons of failure.

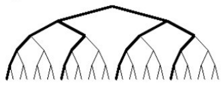
Incomplete Search

General purpose algorithms:

Bounded-backtrack search:

bbs(10)

Depth-bounded, then bounded-backtrack search:



dbs(2, bbs(0))

An Empirical Comparison

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV
USA	(> 1,000K)	(> 1,000K)	2K	60
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	817K
Zebra	3,859K	1K	35K	0.5K
Random 1	415K	3K	26K	2K
Random 2	942K	27K	77K	15K

Mendian number of consistency checks

The structure of problems

- Decomposition in subproblems:
  - connected components in the constraint graph
  - $O(d^c n/c)$ vs $O(d^n)$
- Constraint graphs that are tree are solvable in poly time by reverse arc-consistency checks.
- Reduce constraint graph to tree:
  - removing nodes (cutset conditioning: find the smallest cycle cutset. It is NP-hard but good approximations exist)
  - collapsing nodes (tree decomposition)
    divide-and-conquer works well with small subproblems

Optimization Problems

Objective function $F(X_1, X_2, \ldots, X_n)$

- Solve a modified Constraint Satisfaction Problem by setting a (lower) bound z^* in the objective function
- Dichotomic search: U upper bound, L lower bound

$$M = \frac{U + L}{2}$$

Constraint Logic Programming

Language is first-order logic.

- Syntax – Language
  - Alphabet
  - Well-formed Expressions
    E.g., 4X + 3Y = 10; 2X - Y = 0
- Semantics – Meaning
  - Interpretation
  - Logical Consequence
- Calculi – Derivation
  - Inference Rule
  - Transition System

Logic Programming

A logic program is a set of axioms, or rules, defining relationships between objects.

A computation of a logic program is a deduction of consequences of the program.

A program defines a set of consequences, which is its meaning.

The art of logic programming is constructing concise and elegant programs that have desired meaning.

Sterling and Shapiro: The Art of Prolog, Page 1.

Local Search for CSP

- Uses a complete-state formulation: a value assigned to each variable (randomly)
- Changes the value of one variable at a time
- Min-conflicts heuristic is effective particularly when given a good initial state.
- Run-time independent from problem size
- Possible use in online settings in personal assignment: repair the schedule with a minimum number of changes

Part IV

Constraint Programming, Heuristic Methods

Outline

Outline

9. Heuristic Methods
  Construction Heuristics and Local Search
  Solution Representations and Neighborhood Structures in LS
  Metaheuristics
    Metaheuristics for Construction Heuristics
    Metaheuristics for Local Search and Hybrids

Introduction

Heuristic methods make use of two search paradigms:

- construction rules (extends partial solutions)
- local search (modifies complete solutions)

These components are problem specific and implement informed search.

They can be improved by use of metaheuristics which are general-purpose guidance criteria for underlying problem specific components.

Final heuristic algorithms are often hybridization of several components.

Construction Heuristics

(aka Dispatching Rules, in scheduling)
Closely related to search tree techniques
Correspond to a single path from root to leaf

- search space = partial candidate solutions
- search step = extension with one or more solution components

Construction Heuristic (CH):

$s := \emptyset$
While s is not a complete solution:
 choose a solution component c
 add the solution component to s

Greedy best-first search

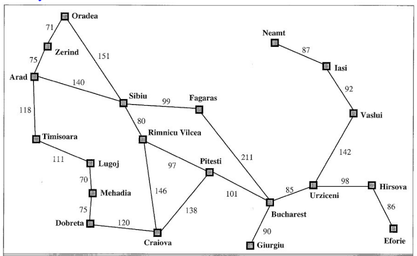


Figure 3.2 A simplified road map of part of Romania.

Greedy best-first search

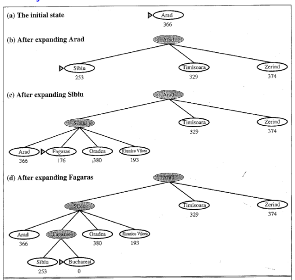


Figure 4.2 Stages in a greedy best-first search for Bucharest using the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

- An important class of Construction Heuristics are greedy algorithms
  Always make the choice which is the best at the moment.
  - Sometime it can be proved that they are optimal
    (Minimum Spanning Tree, Single Source Shortest Path, $1||\sum w_j C_j$, $1||L_{max}$)
  - Other times it can be proved an approximation ratio
- Another class can be derived by the (variable, value) selection rules in CP and removing backtracking (ex, MRV, least-constraining-values).

Examples of Dispatching Rules in Scheduling

Table C.1. Summary of Dispatching Rules

	RULE	DATA	OBJECTIVES
Rules Dependent on Release Dates and Due Dates	*ERD*	r_j	Variance in Throughput Times
	EDD	d_j	Maximum Lateness
	MS	d_j	Maximum Lateness
Rules Dependent on Processing Times	*LPT*	p_j	Load Balancing over Parallel Machines
	SPT	p_j	Sum of Completion Times, WIP
	WSPT	p_j, w_j	Weighted Sum of Completion Times, WIP
	CP	p_j, *prec*	Makespan
	LNS	p_j, *prec*	Makespan
Miscellaneous	*SIRO*	–	Ease of Implementation
	SST	s_{jk}	Makespan and Throughput
	LFJ	M_j	Makespan and Throughput
	SQNO	–	Machine Idleness

Local Search

Example: Local Search for CSP

function MIN-CONFLICTS(*csp*, *max_steps*) **returns** a solution or failure
 inputs: *csp*, a constraint satisfaction problem
  max_steps, the number of steps allowed before giving up

 current ← an initial complete assignment for *csp*
 for i = 1 to *max_steps* **do**
  if *current* is a solution for *csp* **then return** *current*
  var ← a randomly chosen, conflicted variable from VARIABLES[*csp*]
  value ← the value v for *var* that minimizes CONFLICTS(*var*, *v*, *current*, *csp*)
  set *var* = *value* in *current*
 return *failure*

Local Search

Components

- solution representation
- initial solution
- neighborhood structure
- acceptance criterion

Solution Representation

The solution representation determines the search space S

- permutations
  - linear (scheduling)
  - circular (routing)
- assignment arrays (timetabling)
- sets or lists (timetabling)

Initial Solution

- Random
- Construction heuristic

Neighborhood Structure

- Neighborhood structure (relation): equivalent definitions:
  - $\mathcal{N} : S \times S \to \{T, F\}$
  - $\mathcal{N} \subseteq S \times S$
  - $\mathcal{N} : S \to 2^S$
- Neighborhood (set) of a candidate solution s: $N(s) := \{s' \in S \mid \mathcal{N}(s, s')\}$
- A neighborhood structure is also defined by an operator.
  An operator Δ is a collection of operator functions $\delta : S \to S$ such that

$$s' \in N(s) \iff \exists \delta \in \Delta \mid \delta(s) = s'$$

Example

k-exchange neighborhood: candidate solutions s, s' are neighbors iff s differs from s' in at most k solution components

Acceptance Criterion

The acceptance criterion defines how the neighborhood is searched and which neighbor is selected.
Examples:

- uninformed random walk
- iterative improvement (hill climbing)
  - best improvement
  - first improvement

Evaluation function

- function $f(\pi) : S(\pi) \mapsto \mathbb{R}$ that maps candidate solutions of a given problem instance π onto real numbers, such that global optima correspond to solutions of π;
- used for ranking or assessing neighbors of current search position to provide guidance to search process.

Evaluation *vs* objective functions:

- *Evaluation function*: part of LS algorithm.
- *Objective function*: integral part of optimization problem.
- Some LS methods use evaluation functions different from given objective function (*e.g.*, dynamic local search).

Implementation Issues

At each iteration, the examination of the neighborhood must be fast!!

- Incremental updates (aka delta evaluations)
  - **Key idea:** calculate *effects of differences* between current search position s and neighbors s' on evaluation function value.
  - Evaluation function values often consist of *independent contributions of solution components*; hence, $f(s)$ can be efficiently calculated from $f(s')$ by differences between s and s' in terms of solution components.
- Special algorithms for solving efficiently the neighborhood search problem

Local Optima

Definition:

- Local minimum: search position without improving neighbors w.r.t. given evaluation function f and neighborhood $\mathcal{N}$, *i.e.*, position $s \in S$ such that $f(s) \le f(s')$ for all $s' \in N(s)$.
- Strict local minimum: search position $s \in S$ such that $f(s) < f(s')$ for all $s' \in N(s)$.
- *Local maxima* and *strict local maxima*: defined analogously.

Example: Iterative Improvement

First improvement for TSP

procedure *TSP-2opt-first*(s)
 input: *an initial candidate tour* $s \in S(\in)$
 $\Delta = 0$;
 Improvement=FALSE;
 do
  for $i = 1$ **to** $n - 2$ **do**
  if $i = 1$ **then** $n' = n - 1$ **else** $n' = n$
   for $j = i + 2$ **to** n' **do**
    $\Delta_{ij} = d(c_i, c_j) + d(c_{i+1}, c_{j+1}) - d(c_i, c_{i+1}) - d(c_j, c_{j+1})$
    if $\Delta_{ij} < 0$ **then**
     UpdateTour(s,i,j);
     Improvement=TRUE;
   end
  end
 until Improvement==TRUE;
 return: *a local optimum* $s \in S(\pi)$
end *TSP-2opt-first*

Permutations

$\Pi(n)$ indicates the set all permutations of the numbers $\{1, 2, \ldots, n\}$

$(1, 2 \ldots, n)$ is the identity permutation ι.

If $\pi \in \Pi(n)$ and $1 \le i \le n$ then:

- π_i is the element at position i
- $pos_\pi(i)$ is the position of element i

Alternatively, a permutation is a bijective function $\pi(i) = \pi_i$

The permutation product $\pi \cdot \pi'$ is the composition $(\pi \cdot \pi')_i = \pi'(\pi(i))$

For each π there exists a permutation such that $\pi^{-1} \cdot \pi = \iota$

$$\Delta_N \subset \Pi$$

Neighborhood Operators for Linear Permutations

Swap operator

$$\Delta_S = \{\delta_S^i | 1 \le i \le n\}$$

$$\delta_S^i(\pi_1 \ldots \pi_i \pi_{i+1} \ldots \pi_n) = (\pi_1 \ldots \pi_{i+1} \pi_i \ldots \pi_n)$$

Interchange operator

$$\Delta_X = \{\delta_X^{ij} | 1 \le i < j \le n\}$$

$$\delta_X^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{i+1} \ldots \pi_{j-1} \pi_i \pi_{j+1} \ldots \pi_n)$$

Insert operator

$$\Delta_I = \{\delta_I^{ij} | 1 \le i \le n, 1 \le j \le n, j \ne i\}$$

$$\delta_I^{ij}(\pi) = \begin{cases} (\pi_1 \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_n) & i < j \\ (\pi_1 \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_n) & i > j \end{cases}$$

Neighborhood Operators for Circular Permutations

Reversal (2-edge-exchange)

$$\Delta_R = \{\delta_R^{ij} | 1 \le i < j \le n\}$$

$$\delta_R^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \ldots \pi_i \pi_{j+1} \ldots \pi_n)$$

Block moves (3-edge-exchange)

$$\Delta_B = \{\delta_B^{ijk} | 1 \le i < j < k \le n\}$$

$$\delta_B^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \ldots \pi_k \pi_i \ldots \pi_{j-1} \pi_{k+1} \ldots \pi_n)$$

Short block move (Or-edge-exchange)

$$\Delta_{SB} = \{\delta_{SB}^{ij} | 1 \le i < j \le n\}$$

$$\delta_{SB}^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{j+1} \pi_{j+2} \pi_i \ldots \pi_{j-1} \pi_{j+3} \ldots \pi_n)$$

Neighborhood Operators for Assignments

An assignment can be represented as a mapping
$\sigma : \{X_1 \dots X_n\} \to \{v : v \in D, |D| = k\}$:

$$\sigma = \{X_i = v_i, X_j = v_j, \dots\}$$

One exchange operator

$$\Delta_{1E} = \{\delta_{1E}^{il} | 1 \le i \le n, 1 \le l \le k\}$$

$$\delta_{1E}^{il}(\sigma) = \left\{\sigma : \sigma'(X_i) = v_l \text{ and } \sigma'(X_j) = \sigma(X_j)\ \forall j \ne i\right\}$$

Two exchange operator

$$\Delta_{2E} = \{\delta_{2E}^{ij} | 1 \le i < j \le n\}$$

$$\delta_{2E}^{ij}\left\{\sigma : \sigma'(X_i) = \sigma(X_j),\ \sigma'(X_j) = \sigma(X_i) \text{ and } \sigma'(X_l) = \sigma(X_l)\ \forall l \ne i, j\right\}$$

Neighborhood Operators for Partitions or Sets

An assignment can be represented as a partition of objects selected and not selected $s : \{X\} \to \{C, \overline{C}\}$
(it can also be represented by a bit string)

One addition operator

$$\Delta_{1E} = \{\delta_{1E}^{v} | v \in \overline{C}\}$$

$$\delta_{1E}^{v}(s) = \left\{s : C' = C \cup v \text{ and } \overline{C}' = \overline{C} \setminus v\right\}$$

One deletion operator

$$\Delta_{1E} = \{\delta_{1E}^{v} | v \in C\}$$

$$\delta_{1E}^{v}(s) = \left\{s : C' = C \setminus v \text{ and } \overline{C}' = \overline{C} \cup v\right\}$$

Swap operator

$$\Delta_{1E} = \{\delta_{1E}^{v} | v \in C, u \in \overline{C}\}$$

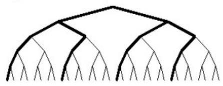
$$\delta_{1E}^{v}(s) = \left\{s : C' = C \cup u \setminus v \text{ and } \overline{C}' = \overline{C} \cup v \setminus u\right\}$$

Construction Heuristics (Extensions)

Bounded-backtrack search:

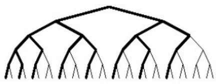
bbs(10)

Depth-bounded, then bounded-backtrack search:



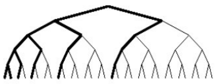
dbs(2, bbs(0))

Credit-based search:



credit(16)

Limited Discrepancy Search:



lds(1)

Rollout/Pilot Method

Derived from A*

- Each candidate solution is a collection of m components $s = (s_1, s_2, \dots, s_m)$.
- Master process add components sequentially to a partial solution $S_k = (s_1, s_2, \dots s_k)$
- At the k-th iteration the master process evaluates seemly feasible components to add based on a look-ahead strategy based on heuristic algorithms.
- The evaluation function $H(S_{k+1})$ is determined by sub-heuristics that complete the solution starting from S_k
- Sub-heuristics are combined in $H(S_{k+1})$ by
  - weighted sum
  - maximal value

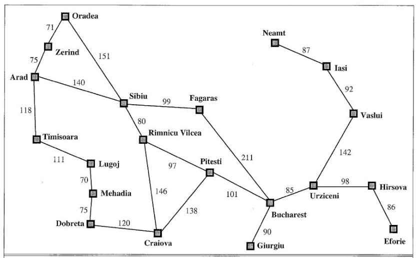


Figure 3.2 A simplified road map of part of Romania.

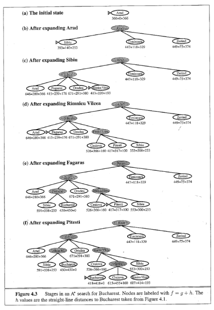


Figure 4.3 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 4.1.

Speed-ups:

- halt whenever cost of current partial solution exceeds current upper bound
- evaluate only a fraction of possible components

It is optimal if $H(S_k)$ is an

- admissible heuristic: *never overestimates* the cost to reach the goal
- consistent: $h(n) \le c(n, a, n') + h(n')$; $c(n, a, n')$ cost to go from node n to n' with action a

Possible choices for admissible heuristic functions

- optimal solution to an easily solvable **relaxed problem**
- optimal solution to an easily solvable **subproblem**
- learning from experience by gathering statistics on state features
- preferred heuristics functions with higher values (provided they do not overestimate)
- if several heuristics available $h_1, h_2, \dots, h_m$ and not clear which is the best then:

$$h(x) = \max\{h_1(x), \dots, h_m(x)\}$$

Beam Search

Possible extension of tree based construction heuristics:

- maintains a set B of bw (beam width) partial candidate solutions
- at each iteration extend each solution from B in fw (filter width) possible ways
- rank each $bw \times fw$ candidate solutions and take the best bw partial solutions
- complete candidate solutions obtained by B are maintained in B_f
- stop when no partial solution in B is to be extended

Iterated Greedy

Key idea: use greedy construction

- alternation of Construction and Deconstruction phases
- an acceptance criterion decides whether the search continues from the new or from the old solution.

Iterated Greedy (IG):
determine initial candidate solution s
while termination criterion is not satisfied **do**
$r := s$
greedily destruct part of s
greedily reconstruct the missing part of s
based on acceptance criterion,
keep s or revert to $s := r$

Greedy Randomized Adaptive Search Procedure (GRASP)

Key Idea: Combine randomized constructive search with subsequent local search.

Greedy Randomized Adaptive Search Procedure (GRASP):
While *termination criterion* is not satisfied:
generate candidate solution s using
subsidiary greedy randomized constructive search
perform subsidiary local search on s

Restricted candidate lists (RCLs)

- Each step of *constructive search* adds a solution component selected uniformly at random from a restricted candidate list (RCL).
- RCLs are constructed in each step using a *heuristic function* h.
  - RCLs based on cardinality restriction comprise the k best-ranked solution components. (k is a parameter of the algorithm.)
  - RCLs based on value restriction comprise all solution components l for which $h(l) \le h_{min} + \alpha \cdot (h_{max} - h_{min})$, where h_{min} = minimal value of h and h_{max} = maximal value of h for any l. (α is a parameter of the algorithm.)

Simulated Annealing

Key idea: Vary temperature parameter, *i.e.*, probability of accepting worsening moves, in Probabilistic Iterative Improvement according to *annealing schedule* (aka *cooling schedule*).

Simulated Annealing (SA):
determine initial candidate solution s
set initial temperature T according to *annealing schedule*
While termination condition is not satisfied:
While maintain same temperature T according to *annealing schedule*:
probabilistically choose a neighbor s' of s
using *proposal mechanism*
If s' satisfies probabilistic *acceptance criterion* (depending on T):
$s := s'$
update T according to *annealing schedule*

Note:

- 2-stage neighbor selection procedure
  - proposal mechanism (often uniform random choice from $N(s)$)
  - acceptance criterion (often *Metropolis condition*)

$$p(T, s, s') := \begin{cases} 1 & \text{if } g(s') \le f(s) \\ \exp \frac{f(s) - f(s')}{T} & \text{otherwise} \end{cases}$$

- Annealing schedule (function mapping run-time t onto temperature $T(t)$):
  - initial temperature T_0 (may depend on properties of given problem instance)
  - temperature update scheme (*e.g.*, linear cooling: $T_{i+1} = T_0(1 - i/I_{max})$, geometric cooling: $T_{i+1} = \alpha \cdot T_i$)
  - number of search steps to be performed at each temperature (often multiple of neighborhood size)
- Termination predicate: often based on *acceptance ratio*, *i.e.*, ratio of proposed *vs* accepted steps *or* number of idle iterations

Example: Simulated Annealing for the TSP

Extension of previous PII algorithm for the TSP, with

- *proposal mechanism:* uniform random choice from 2-exchange neighborhood;
- *acceptance criterion:* Metropolis condition (always accept improving steps, accept worsening steps with probability $\exp[(f(s) - f(s'))/T]$);
- *annealing schedule*: geometric cooling $T := 0.95 \cdot T$ with $n \cdot (n-1)$ steps at each temperature (n = number of vertices in given graph), T_0 chosen such that 97% of proposed steps are accepted;
- *termination:* when for five successive temperature values no improvement in solution quality and acceptance ratio $< 2\%$.

Improvements:

- neighborhood pruning (*e.g.*, candidate lists for TSP)
- greedy initialization (*e.g.*, by using NNH for the TSP)
- *low temperature starts* (to prevent good initial candidate solutions from being too easily destroyed by worsening steps)

Tabu Search

Key idea: Use aspects of search history (memory) to escape from local minima.

- Associate *tabu attributes* with candidate solutions or solution components.
- Forbid steps to search positions recently visited by underlying iterative best improvement procedure based on tabu attributes.

Tabu Search (TS):
determine initial candidate solution s
While *termination criterion* is not satisfied:
determine set N' of non-tabu neighbors of s
choose a best improving candidate solution s' in N'
update tabu attributes based on s'
$s := s'$

Note:

- Non-tabu search positions in $N(s)$ are called *admissible neighbors of s*.
- After a search step, the current search position or the solution components just added/removed from it are declared *tabu* for a fixed number of subsequent search steps (*tabu tenure*).
- Often, an additional *aspiration criterion* is used: this specifies conditions under which tabu status may be overridden (*e.g.*, if considered step leads to improvement in incumbent solution).
- Crucial for efficient implementation:
  - keep time complexity of search steps minimal by using special data structures, incremental updating and caching mechanism for evaluation function values;
  - efficient determination of tabu status: store for each variable x the number of the search step when its value was last changed it_x; x is tabu if $it - it_x < tt$, where it = current search step number.

Note: Performance of Tabu Search depends crucially on setting of tabu tenure *tt*:

- *tt* too low ⇒ search stagnates due to inability to escape from local minima;
- *tt* too high ⇒ search becomes ineffective due to overly restricted search path (admissible neighborhoods too small)

Iterated Local Search

Key Idea: Use two types of LS steps:

- *subsidiary local search* steps for reaching local optima as efficiently as possible (intensification)
- *perturbation steps* for effectively escaping from local optima (diversification).

Also: Use *acceptance criterion* to control diversification *vs* intensification behavior.

Iterated Local Search (ILS):

determine initial candidate solution s
perform *subsidiary local search* on s
While termination criterion is not satisfied:
  $r := s$
  perform *perturbation* on s
  perform *subsidiary local search* on s
  based on *acceptance criterion*,
    keep s or revert to $s := r$

Memetic Algorithm

Population based method inspired by evolution

determine initial population sp
perform subsidiary local search on sp
While *termination criterion* is not satisfied:
  generate set spr of new candidate solutions
    by recombination
  perform subsidiary local search on spr
  generate set spm of new candidate solutions
    from spr and sp by mutation
  perform subsidiary local search on spm
  select new population sp from
    candidate solutions in sp, spr, and spm

Selection

Main idea: selection should be related to fitness

- Fitness proportionate selection (Roulette-wheel method)

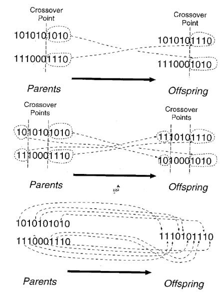
$$p_i = \frac{f_i}{\sum_j f_j}$$

- Tournament selection: a set of chromosomes is chosen and compared and the best chromosome chosen.
- Rank based and selection pressure

Recombination (Crossover)

- Binary or assignment representations
  - one-point, two-point, m-point (preference to positional bias w.r.t. distributional bias
  - uniform cross over (through a mask controlled by a Bernoulli parameter p)
- Non-linear representations
  - (Permutations) Partially mapped crossover
  - (Permutations) mask based
- More commonly ad hoc crossovers are used as this appears to be a crucial feature of success
- Two off-springs are generally generated
- Crossover rate controls the application of the crossover. May be adaptive: high at the start and low when convergence

Example: crossovers for binary representations



Mutation

- *Goal:* Introduce relatively small perturbations in candidate solutions in current population + offspring obtained from *recombination*.
- Typically, perturbations are applied stochastically and independently to each candidate solution; amount of perturbation is controlled by *mutation rate*.
- Mutation rate controls the application of bit-wise mutations. May be adaptive: low at the start and high when convergence
- Possible implementation through Poisson variable which determines the m genes which are likely to change allele.
- Can also use *subsidiary selection function* to determine subset of candidate solutions to which mutation is applied.
- The role of mutation (as compared to recombination) in high-performance evolutionary algorithms has been often underestimated

New Population

- Determines population for next cycle (*generation*) of the algorithm by selecting individual candidate solutions from current population + new candidate solutions obtained from *recombination*, *mutation* (+ *subsidiary local search*). (λ, μ) $(\lambda + \mu)$
- *Goal:* Obtain population of high-quality solutions while maintaining *population diversity*.
- Selection is based on evaluation function (*fitness*) of candidate solutions such that better candidate solutions have a higher chance of 'surviving' the selection process.
- It is often beneficial to use *elitist selection strategies*, which ensure that the best candidate solutions are always selected.
- Most commonly used: *steady state* in which only one new chromosome is generated at each iteration
- Diversity is checked and duplicates avoided

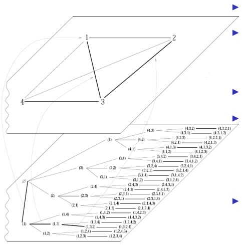
Ant Colony Optimization

The Metaheuristic

- The optimization problem is transformed into the problem of finding the best path on a weighted graph $G(V, E)$ called construction graph
- The artificial ants incrementally build solutions by moving on the graph.
- The solution construction process is
  - stochastic
  - biased by a pheromone model, that is, a set of parameters associated with graph components (either nodes or edges) whose values are modified at runtime by the ants.
- All *pheromone trails* are initialized to the same value, τ_0.
- At each iteration, *pheromone trails* are updated by decreasing (*evaporation*) or increasing (*reinforcement*) some trail levels on the basis of the solutions produced by the ants

Ant Colony Optimization

Example: A simple ACO algorithm for the TSP



- *Construction graph*
- To each edge ij in G associate
  - pheromone trails τ_{ij}
  - heuristic values $\eta_{ij} := \frac{1}{c_{ij}}$
- Initialize pheromones
- *Constructive search:*

$$p_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta},$$

- *Update pheromone trail levels*

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \rho \cdot \text{Reward}$$

Example: A simple ACO algorithm for the TSP (1)

- Search space and solution set as usual (all Hamiltonian cycles in given graph G).
- Associate pheromone trails τ_{ij} with each edge (i, j) in G.
- Use heuristic values $\eta_{ij} := \frac{1}{c_{ij}}$
- Initialize all weights to a small value τ_0 $(\tau_0 = 1)$.
- *Constructive search:* Each ant starts with randomly chosen vertex and iteratively extends partial round trip π^k by selecting vertex not contained in π^k with probability

$$p_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}$$

α and β are parameters.

Example: A simple ACO algorithm for the TSP (2)

- *Subsidiary local search:* Perform iterative improvement based on standard 2-exchange neighborhood on each candidate solution in population (until local minimum is reached).
- *Update pheromone trail levels* according to

$$\tau_{ij} := (1 - \rho) \cdot \tau_{ij} + \sum_{s \in sp'} \Delta_{ij}(s)$$

where $\Delta_{ij}(s) := 1/C^s$
if edge (i, j) is contained in the cycle represented by s', and 0 otherwise.

Motivation: Edges belonging to highest-quality candidate solutions and/or that have been used by many ants should be preferably used in subsequent constructions.

- *Termination:* After fixed number of cycles (= construction + local search phases).

Part V

Mathematical Programming, Exercises

Outline

Outline

How to solve mathematical programs

- Use a mathematical workbench like MATLAB, MATHEMATICA, MAPLE, R.
- Use a modeling language to convert the theoretical model to a computer usable representation and employ an out-of-the-box general solver to find solutions.
- Use a framework that already has many general algorithms available and only implement problem specific parts, e. g., separators or upper bounding.
- Develop everything yourself, maybe making use of libraries that provide high-performance implementations of specific algorithms.

Thorsten Koch
"Rapid Mathematical Programming"
Technische Universität, Berlin, Dissertation, 2004

How to solve mathematical programs

- Use a mathematical workbench like MATLAB, MATHEMATICA, MAPLE, R.

Advantages: easy if familiar with the workbench

Disadvantages: restricted, not state-of-the-art

How to solve mathematical programs

- Use a modeling language to convert the theoretical model to a computer usable representation and employ an out-of-the-box general solver to find solutions.

Advantages: flexible on modeling side, easy to use, immediate results, easy to test different models, possible to switch between different state-of-the-art solvers

Disadvantages: algoritmical restrictions in the solution process, no upper bounding possible

How to solve mathematical programs

- Use a framework that already has many general algorithms available and only implement problem specific parts, e.g., separators or upper bounding.

Advantages: allow to implement sophisticated solvers, high performance bricks are available, flexible

Disadvantages: view imposed by designers, vendor specific hence no transferability,

How to solve mathematical programs

- Develop everything yourself, maybe making use of libraries that provide high-performance implementations of specific algorithms.

Advantages: specific implementations and max flexibility

Disadvantages: for extremely large problems, bounding procedures are more crucial than branching

Modeling Languages

Name		URL	Solver	State
AIMMS	Advanced Integrated Multi-dimensional Modeling Software	www.aimms.com	open	commercial
AMPL	A Modeling Language for Mathematical Programming	www.ampl.com	open	commercial
GAMS	General Algebraic Modeling System	www.gams.com	open	commercial
LINGO	Lingo	www.lindo.com	fixed	commercial
LPL	(Linear|Logical|Literate) Programming Language	www.virtual-optima.com	open	commercial
MINOPT	Mixed Integer Non-linear Optimizer	titan.princeton.edu/MINOPT	open	mixed
MOSEL	Mosel	www.dashoptimization.com	fixed	commercial
MPL	Mathematical Programming Language	www.maximalsoftware.com	open	commercial
OMNI	Omni	www.haverly.com	open	commercial
OPL	Optimization Programming Language	www.ilog.com	fixed	commercial
GNU-MP	GNU Mathematical Programming Language	www.gnu.org/software/glpk	fixed	free
ZIMPL	Zuse Institute Mathematical Programming Language	www.zib.de/koch/zimpl	open	free

Thorsten Koch
"Rapid Mathematical Programming"
Technische Universität, Berlin, Dissertation, 2004

LP-Solvers

CPLEX	`http://www.ilog.com/products/cplex`
XPRESS-MP	`http://www.dashoptimization.com`
SOPLEX	`http://www.zib.de/Optimization/Software/Soplex`
COIN CLP	`http://www.coin-or.org`
GLPK	`http://www.gnu.org/software/glpk`
LP_SOLVE	`http://lpsolve.sourceforge.net/`

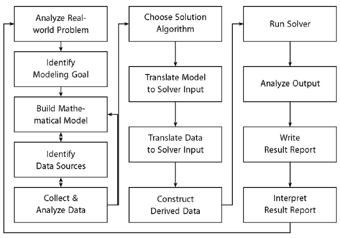
"Software Survey: Linear Programming" by Robert Fourer
`http://www.lionhrtpub.com/orms/orms-6-05/frsurvey.html`

Outline

ZIBOpt

- Zimpl is a little algebraic Modeling language to translate the mathematical model of a problem into a linear or (mixed-) integer mathematical program expressed in .lp or .mps file format which can be read and (hopefully) solved by a LP or MIP solver.
- Scip is an IP-Solver. It solves Integer Programs and Constraint Programs: the problem is successively divided into smaller subproblems (branching) that are solved recursively. Integer Programming uses LP relaxations and cutting planes to provide strong dual bounds, while Constraint Programming can handle arbitrary (non-linear) constraints and uses propagation to tighten domains of variables.
- SoPlex is an LP-Solver. It implements the revised simplex algorithm. It features primal and dual solving routines for linear programs and is implemented as a C++ class library that can be used with other programs (like SCIP). It can solve standalone linear programs given in MPS or LP-Format.

Modeling Cycle



H. Schichl. "Models and the history of modeling".
In Kallrath, ed., Modeling Languages in Mathematical Optimization, Kluwer, 2004.

Some commands

```
$ zimpl -t lp sudoku.zpl
$ scip -f sudoku.lp
```

```
scip> help
scip> read sudoku.lp
scip> display display
scip> display problem
scip> set display width 120
scip> display statistics
scip> display parameters
scip> set default
scip> set load settings/*/*.set
scip> set load /home/marco/ZIBopt/ziboptsuite-1.00/scip-1.00/settings/
    emphasis/cpsolver.set
```


Callable libraries
How to construct a problem instance in SCIP

```
SCIPcreate(), // create a SCIP object
SCIPcreateProb() // build the problem
SCIPcreateVar() // create variables
SCIPaddVar() // add them to the problem
// Constraints: For example, if you want to
// fill in the rows of a general MIP, you have to call
SCIPcreateConsLinear(),
SCIPaddConsLinear()
SCIPreleaseCons() // after finishing.
SCIPsolve()
SCIPreleaseVar() releas variable poiinters
```

```
SCIP_CALL() // exception handling
```

```
SCIPsetIntParam(scip, "display/memused/status", 0) == set display \
memused status 0
SCIPprintStatistics() == display statistics
```


Sudoku into Exact Hitting Set

Exact Covering: Set partitioning with $\vec{c} = \vec{1}$

- A = 1, 4, 7;
- B = 1, 4;
- C = 4, 5, 7;
- D = 3, 5, 6;
- E = 2, 3, 6, 7; and
- F = 2, 7.

$$\min \sum_{j=1}^{n} y_j$$
$$\sum_{j=1}^{n} a_{ij} y_j = 1 \quad \forall i$$
$$y_j \in \{0,1\}$$

$$\begin{array}{c c} & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{array}{c} 1\\2\\3\\4\\5\\6\\7 \end{array} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

The dual of Exact Covering is the Exact Hitting Set

- A = 1, 2
- B = 5, 6
- C = 4, 5
- D = 1, 2, 3
- E = 3, 4
- F = 4, 5
- G = 1, 3, 5, 6

$$\max \sum_{j=1}^{n} x_j$$
$$\sum_{j=1}^{n} a_{ij} x_j = 1 \quad \forall i$$
$$x_j \in \{0,1\}$$

$$\begin{array}{c c} & \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{array}{c} 1\\2\\3\\4\\5\\6 \end{array} & \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Part VI

Constraint Programming in Practice

Outline

Outline

Constraint Programming Systems

CP systems must provide reusable services for:

- Variable domains
  finite domain integer, finite sets, multisets, intervals, ...
- Constraints
  distinct, arithmetic, scheduling, graphs, ...
- Solving
  propagation, branching, exploration, ...
- Modelling
  variables, values, constraints, heuristics, symmetries, ...

CP modelling

Greater expressive power than mathematical programming

- constraints involving disjunction can be represented directly
- constraints can be encapsulated (as predicates) and used in the definition of further constrains

However, CP models can often be translated into MIP model by

- eliminating disjunctions in favor of auxiliary Boolean variables
- unfolding predicates into their definitions

CP System Interfaces

Two possible interfaces:

- host language
- libraries

Modelling Language

- Fundamental difference to LP
  - language has structure (global constraints)
  - different solvers support different constraints
- In its infancy
- Key questions:
  - what level of abstraction?
    - solving approach independent: LP, CP, ...?
    - how to map to different systems?
  - Modelling is very difficult for CP
    - requires lots of knowledge and tinkering

Modelling Languages

- Prolog
  - B-Prolog (Prolog based, proprietary)
  - CHIP V5 (Prolog based, also includes C++ and C libraries, proprietary)
  - Ciao Prolog (Prolog based, Free software: GPL/LGPL)
  - ECLiPSe (Prolog based, open source)
  - SICStus (Prolog based, proprietary)
  - GNU Prolog
- OPL
- Zinc, MiniZinc, FlatZinc

CP Systems

- Library-based
  - CHOCO (free) `http://choco.sourceforge.net/`
  - Kaolog (commercial) `http://www.koalog.com/php/index.php`
  - Gecode (free) `www.gecode.org`
    Programming interfaces Java and MiniZinc, library C++

CP Systems

- Language-based
  - SICStus Prolog (commercial) `www.sics.se/sicstus`
    Prolog language, library
  - ECLiPSe (free) `www.eclipse-clp.org`
    Prolog language, library
  - Mozart (free) `http://www.mozart-oz.org`
    Oz language
  - ILOG CP Optimizer `http://www.ilog.com/products/`
    OPL Language, libraries C/C++/
  - CHIP (commercial) `http://www.cosytec.com`
    Prolog language, library C/C++
  - G12 Project `http://www.g12.cs.mu.oz.au/`

Outline

Solving CP

- Compute with possible values
  rather than enumerating assignments
- Prune inconsistent values
  constraint propagation
- Search
  branch: define search tree
  explore: explore search tree for solution
  branching heuristics
  best solution search (in optimization)

Propagators

CP Systems do not compute constraints extensionally (as a collection of assignments):

- impractical (space)
- would make difficult to take advantage of structure

A Constraint c is implemented by a set of propagators (also known as filtering algorithms and narrowing operators).

A propagator p is a function that maps domains to domains. They are decreasing and monotonic.

A set of propagators implements a constraint c if all $p \in P$ are correct for c and P is checking for c. Notation: $P = \text{prop}(c)$

Execution of Propagators

- Execution of propagator p
  - narrows domains of variables in var(p)
  - signals failure
- Execution computes largest simultaneous fixpoint
  - fixpoint: propagators cannot narrow any further
  - largest: no solutions lost
- Propagator is either
  fix: has reached fixpoint
  runnable: not known to have reached fixpoint
- Propagation execution maintains propagator sets
- Propagators know their variables
  - to perform domain modifications
  - passed as parameters to propagator creation
- Variables know dependent propagators
  - to perform efficient computation of dependent propagators

Global Constraints

- Classic example: $x, y, z \in \{1, 2\}, \quad x \neq y, x \neq z, y \neq z$
- No solution!
- But: each individual constraint still satisfiable!
  no propagation possible!
- Solution: look at several constraints at once
  distinct(x,y,z)
- Specialization

Kinds of symmetries

- Variable symmetry:
  permuting variables keeps solutions invariant
  $\{x_i \to v_i\} \in sol(P) \Leftrightarrow \{x_{\pi(i)} \to v_i\} \in sol(P)$
- Value symmetry: permuting values keeps solutions invariant
  $\{x_i \to v_i\} \in sol(P) \Leftrightarrow \{x_i \to \pi(v_i)\} \in sol(P)$
- Variable/value symmetry:
  permute both variables and values
  $\{x_i \to v_i\} \in sol(P) \Leftrightarrow \{x_{\pi(i)} \to \pi(v_i)\} \in sol(P)$

Symmetry

- inherent in the problem (sudoku, queens)
- artefact of the model (order of groups)

How can we avoid it?

- ... by model reformulation (eg, use set variables,
- ... by adding constraints to the model
  (ruling out symmetric solutions)
- ... during search
- ... by dominance detection

Reified constraints

- Constraints are in a big conjunction
- How about disjunctive constraints?

$$A + B = C \quad \vee \quad C = 0$$

- Solution: reify the constraints:

$$\begin{array}{lll} (A + B = C & \Leftrightarrow & b_0) \quad \wedge \\ (C = 0 & \Leftrightarrow & b_1) \quad \wedge \\ (b_0 \quad \vee \quad b_1 & \Leftrightarrow & \text{true}) \end{array}$$

Scheduling Models

- Variable for start-time of task a start(a)
- Precedence constraint:
  $\text{start}(a) + \text{dur}(a) \leq \text{start}(b)$ (a before b)
- Disjunctive constraint:
  $\text{start}(a) + \text{dur}(a) \leq \text{start}(b)$ (a before b)
  or
  $\text{start}(b) + \text{dur}(b) \leq \text{start}(a)$ (b before a)
  Solved by reification
- Cumulative Constraints (renewable resources)
  For tasks a and b on resource R
  $\text{use}(a) + \text{use}(b) \leq \text{cap}(R)$
  or $\text{start}(a) + \text{dur}(a) \leq \text{start}(b)$
  or $\text{start}(b) + \text{dur}(b) \leq \text{start}(a)$

Propagators for Scheduling

Serialization: ordering of tasks on one machine

- Consider all tasks on one resource
- Deduce their order as much as possible
- Propagators:
  - Timetabling: look at free/used time slots
  - Edge-finding: which task first/last?
  - Not-first / not-last

Job Shop Problem

- Hard problem!
- 6x6 instance solvable using Gecode
  - disjunction by reification
  - normal branching
- Classic 10x10 instance not solvable using Gecode!
  - specialized propagators (edge-finding) and branchings needed

References

- Lecture notes by Christian Schulte for courses at KTH, Sweden
- Lecture notes by Marco Kuhlmann and Guido Tack for courses at Saarland University

Outline

Exercise

Write a MiniZinc model for the instance of Resource Constraint Project Scheduling Problem and solve the instance made available.

An installation of `minizinc-0.7` might be sufficient (uses G12 to solve the problem)

```
> mzn2fzn --data rcpsp.data rcpsp.mzn
> flatzinc jobshop.fzn
```

Otherwise, it is possible to use the interface `gecode-flatzinc-1.1` for `gecode-2.0.1`

```
> mzn2fzn --data rcpsp.data rcpsp.mzn
> fz jobshop.fzn
```


Part VII

Local Search Heuristics, Exercises

Outline

2.1. Consider the instance of $1 \mid\mid \sum w_j C_j$ with the following processing times and weights.

jobs	1	2	3	4
w_j	6	11	9	5
p_j	3	5	7	4

(a) Find the optimal sequence and compute the value of the objective.
(b) Give an argument for positioning jobs with larger weight more toward the beginning of the sequence and jobs with smaller weight more toward the end of the sequence.
(c) Give an argument for positioning jobs with smaller processing time more toward the beginning of the sequence and jobs with larger processing time more toward the end of the sequence.
(d) Determine which one of the following two generic rules is the most suitable for the problem:
i. sequence the jobs in decreasing order of $w_j - p_j$;
ii. sequence the jobs in decreasing order of w_j/p_j.

2.2. Consider the instance of $1 \mid\mid L_{\max}$ with the following processing times and due dates.

jobs	1	2	3	4
p_j	5	4	3	6
d_j	3	5	11	12

(a) Find the optimal sequence and compute the value of the objective.
(b) Give an argument for positioning jobs with earlier due dates more toward the beginning of the sequence and jobs with later due dates more toward the end of the sequence.
(c) Give an argument for positioning jobs with smaller processing time more toward the beginning of the sequence and jobs with larger processing time more toward the end of the sequence.
(d) Determine which one of the following four rules is the most suitable generic rule for the problem:
i. sequence the jobs in increasing order of $d_j + p_j$;
ii. sequence the jobs in increasing order of $d_j p_j$;
iii. sequence the jobs in increasing order of d_j;
iv. sequence the jobs in increasing order of p_j.

Outline

Software Tools

- Modeling languages
  interpreted languages with a precise syntax and semantics
- Software libraries
  collections of subprograms used to develop software
- Software frameworks
  set of abstract classes and their interactions
  - *frozen spots* (remain unchanged in any instantiation of the framework)
  - *hot spots* (parts where programmers add their own code)

No well established software tool for Local Search:

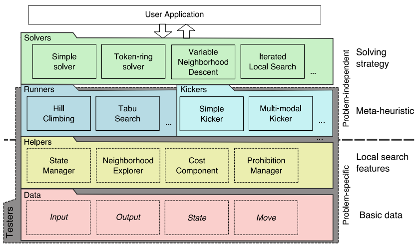
- the apparent simplicity of Local Search induces to build applications from scratch.
- crucial roles played by delta/incremental updates which is problem dependent
- the development of Local Search is in part a craft, beside engineering and science.
- lack of a unified view of Local Search.

Software tools for Local Search and Metaheuristics

Tool	Reference	Language	Type
ILOG	[?]	C++, Java, .NET	LS
GAlib	[?]	C++	GA
GAUL	[?]	C	GA
Localizer++	[?]	C++	Modeling
HotFrame	[?]	C++	LS
EasyLocal++	[?]	C++, Java	LS
HSF	[?]	Java	LS, GA
ParadisEO	[?]	C++	EA, LS
OpenTS	[?]	Java	TS
MDF	[?]	C++	LS
TMF	[?]	C++	LS
SALSA	[?]	—	Language
Comet	[?]	—	Language

table prepared by L. Di Gaspero

Separation of Concepts in Local Search Algorithms



implemented in EasyLocal++

Outline

Input (util.h, util.c)

```
typedef struct {
  long int number_jobs; /* number of jobs in instance */
  long int release_date[MAX_JOBS]; /*there is no release date for these instances*/
  long int proc_time[MAX_JOBS];
  long int weight[MAX_JOBS];
  long int due_date[MAX_JOBS];
} instance_type;

instance_type instance;

void read_problem_size (char name[100])
void read_instances (char input_file_name[100])
```


State/Solution (util.h)

```
typedef struct {
  long int job_at_pos[MAX_JOBS]; /* Gives the job at a certain pos */
  long int pos_of_job[MAX_JOBS]; /* Gives the position of a specific job */
  long int completion_time_job[MAX_JOBS]; /* Gives C_j of job j */
  long int start_time_job[MAX_JOBS]; /* Gives start time of job j */
  long int tardiness_job[MAX_JOBS]; /* Gives T_j of job j */
  long int value; /* Objective function value */
} sol_representation;

sol_representation sequence;
```

Output (util.c)

```
void print_sequence (long int k)
void print_completion_times ()
```

State Manager (util.c)

```
void construct_sequence_random ()
void construct_sequence_canonical ()
long int evaluate ()
```


Random Generator (random.h, random.c)

```
void set_seed (double arg)
double MRG32k3a (void)
double ranU01 (void)
int ranUint (int i, int j)
void shuffle (int *X, int size)
```

Timer (timer.c)

```
double getCurrentTime ()
```


Outline

Your Task on $1 \| \sum_j w_j T_j$

1. Implement two basic local search procedures that return a local optimum:

```
void ls_swap_first( ) {};
void ls_interchange_first( ) {};
```

2. Implement the other neighborhood for permutation representation mentioned at the lecture from one of the two previous neighborhoods.
3. Provide computational analysis of the LS implemented. Consider:
   - size of the neighborhood
   - diameter of neighborhood
   - complete neighborhood examination
   - local optima attainment
4. Devise speed ups to reduce the computational complexity of the LS implemented
5. Improve your heuristic in order to find solutions of better quality. (Hint: use a construction heuristic and/or a metaheuristic)

Part VIII

Single Machine Models

Outline

Outline

Dispatching rules

Distinguish static and dynamic rules.

- Service in random order (SIRO)
- Earliest release date first (ERD=FIFO)
  - tends to min variations in waiting time
- Earliest due date (EDD)
- Minimal slack first (MS)
  - $j^* = \arg\min_j \{\max(d_j - p_j - t, 0)\}$.
  - tends to min due date objectives (T,L)

- (Weighted) shortest processing time first (WSPT)
  - $j^* = \arg\max_j \{w_j/p_j\}$.
  - tends to min $\sum w_j C_j$ and max work in progress and
- Loongest processing time first (LPT)
  - balance work load over parallel machines
- Shortest setup time first (SST)
  - tends to min C_{max} and max throughput
- Least flexible job first (LFJ)
  - eligibility constraints

- Critical path (CP)
  - first job in the CP
  - tends to min C_{max}
- Largest number of successors (LNS)
- Shortest queue at the next operation (SQNO)
  - tends to min idleness of machines

Table C.1. Summary of Dispatching Rules

	RULE	DATA	OBJECTIVES
Rules Dependent on Release Dates and Due Dates	*ERD*	r_j	Variance in Throughput Times
	EDD	d_j	Maximum Lateness
	MS	d_j	Maximum Lateness
Rules Dependent on Processing Times	*LPT*	p_j	Load Balancing over Parallel Machines
	SPT	p_j	Sum of Completion Times, WIP
	WSPT	p_j, w_j	Weighted Sum of Completion Times, WIP
	CP	p_j, *prec*	Makespan
	LNS	p_j, *prec*	Makespan
Miscellaneous	*SIRO*	–	Ease of Implementation
	SST	s_{jk}	Makespan and Throughput
	LFJ	M_j	Makespan and Throughput
	SQNO	–	Machine Idleness

When dispatching rules are optimal?

	RULE	DATA	ENVIRONMENT
1	SIRO	—	—
2	ERD	r_j	$1 \mid r_j \mid \mathrm{Var}(\sum(C_j - r_j)/n)$
3	EDD	d_j	$1 \mid\mid L_{\max}$
4	MS	d_j	$1 \mid\mid L_{\max}$
5	SPT	p_j	$Pm \mid\mid \sum C_j; Fm \mid p_{ij} = p_j \mid \sum C_j$
6	WSPT	w_j, p_j	$Pm \mid\mid \sum w_j C_j$
7	LPT	p_j	$Pm \mid\mid C_{\max}$
8	SPT-LPT	p_j	$Fm \mid block, p_{ij} = p_j \mid C_{\max}$
9	CP	$p_j, prec$	$Pm \mid prec \mid C_{\max}$
10	LNS	$p_j, prec$	$Pm \mid prec \mid C_{\max}$
11	SST	s_{jk}	$1 \mid s_{jk} \mid C_{\max}$
12	LFJ	M_j	$Pm \mid M_j \mid C_{\max}$
13	LAPT	p_{ij}	$O2 \mid\mid C_{\max}$
14	SQ	—	$Pm \mid\mid \sum C_j$
15	SQNO	—	$Jm \mid\mid \gamma$

Composite dispatching rules

Why composite rules?

- Example: $1 \mid\mid \sum w_j T_j$:
  - WSPT, optimal if due dates are zero
  - EDD, optimal if due dates are loose
  - MS, tends to minimize T

➤ The efficacy of the rules depends on instance factors

Instance characterization

- Job attributes: {weight, processing time, due date, release date}
- Machine attributes: {speed, num. of jobs waiting, num. of jobs eligible }

Possible instance factors:

$$\theta_1 = 1 - \frac{\bar{d}}{C_{max}} \qquad \text{(due date tightness)}$$

$$\theta_2 = \frac{d_{max} - d_{min}}{C_{max}} \qquad \text{(due date range)}$$

$$\theta_3 = \frac{\bar{s}}{\bar{p}} \qquad \text{(set up time severity)}$$

(estimated $\hat{C}_{max} = \sum_{j=1}^{n} p_j + n\bar{s}$)

- Dynamic apparent tardiness cost (ATC)

$$I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{K\bar{p}}\right)$$

- Dynamic apparent tardiness cost with setups (ATCS)

$$I_j(t, l) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{K_1\bar{p}}\right) \exp\left(\frac{-s_{jk}}{K_2\bar{s}}\right)$$

after job l has finished.

Summary

- Scheduling classification
- Solution methods
- Practice with general solution methods
  - Mathematical Programming
  - Constraint Programming
  - Heuristic methods

Remainder on Scheduling

Objectives:
Look closer into scheduling models and learn:

- special algorithms
- application of general methods

Cases:

- Single Machine
- Parallel Machine
- Permutation Flow Shop
- Job Shop
- Resource Constrained Project Scheduling

Outline

Summary

Single Machine Models:

- C_{max} is sequence independent
- if $r_j = 0$ and h_j is monotone in C_j then optimal schedule is nondelay and has no preemption.

$1 \mid\mid \sum w_j C_j$

[Total weighted completion time]

- **Theorem:** The weighted shortest processing time first (WSPT) rule is optimal.

Extensions to $1 \mid prec \mid \sum w_j C_j$

- in the general case strongly NP-hard
- chain precedences:
  process first chain with highest ρ-factor up to, and included, job with highest ρ-factor.
- poly also for tree and sp-graph precedences

Extensions to $1 \mid r_j, prmp \mid \sum w_j C_j$

- in the general case strongly NP-hard
- preemptive version of the WSPT if equal weights
- however, $1 \mid r_j \mid \sum w_j C_j$ is strongly NP-hard

$1 \mid prec \mid L_{max}$

[maximum lateness]

- generalization: $h_{max} = \max\{h(C_1), h(C_2), \ldots, h(C_n)\}$
- Solved by backward dynamic programming in $O(n^2)$:

  J set of jobs already scheduled;
  J^c set of jobs still to schedule;
  $J' \subseteq J^c$ set of schedulable jobs

  Step 1: Set $J = \emptyset$, $J^c = \{1, \ldots, n\}$ and J' the set of all jobs with no successor

  Step 2: Select j^* such that $j^* = \arg\min_{j \in J'}\{h_j\left(\sum_{k \in J^c} p_k\right)\}$;
  add j^* to J; remove j^* from J^c; update J'.

  Step 3: If J^c is empty then stop, otherwise go to Step 2.

- For $1 \mid\mid L_{max}$ Earliest Due Date first
- $1|r_j|L_{max}$ is instead strongly NP-hard

$1 \mid\mid \sum h_j(C_j)$

- generalization of $\sum w_j T_j$ hence strongly NP-hard
- efficient (forward) dynamic programming algorithm $O(2^n)$

  J set of job already scheduled;

  $V(J) = \sum_{j \in J} h_j(C_j)$

  Step 1: Set $J = \emptyset$, $V(j) = h_j(p_j)$, $\quad j = 1, \ldots, n$

  Step 2: $V(J) = \min_{j \in J}\left(V(J - \{j\}) + h_j\left(\sum_{k \in J} p_k\right)\right)$

  Step 3: If $J = \{1, 2, \ldots, n\}$ then $V(\{1, 2, \ldots, n\})$ is optimum, otherwise go to Step 2.

$1 \mid s_{jk} \mid C_{max}$

[Makespan with sequence-dependent setup times]

- general case is NP-hard (traveling salesman reduction).
- special case:

  parameters a_j, b_j for job j with

  $$s_{jk} \propto |a_k - b_j|$$

  [Gilmore and Gomory, 1964] give a $O(n^2)$ algorithm

- assume $b_0 \leq b_1 \leq \ldots \leq b_n$ ($k > j$ and $b_k \geq b_j$)
- one-to-one correspondence with solution of TSP with $n+1$ cities
  city 0 has a_0, b_0
  start at b_0 finish at a_0
- tour representation $\phi : \{0, 1, \ldots, n\} \mapsto \{0, 1, \ldots, n\}$
  (permutation map, single linked array)
- Hence,

$$\min \; c(\phi) = \sum_{i=1}^{n} c_{i,\phi(i)} \tag{1}$$

$$\phi(S) \neq S \qquad \forall S \subset V \tag{2}$$

- find ϕ^* by ignoring (2)
  make ϕ^* a tour through swaps
  (swap chosen solving a min spanning tree and applied in a certain order)

- Interchange δ^{jk}

$$\delta^{jk}(\phi) = \{\phi' \quad | \quad \phi'(j) = \phi(k), \quad \phi(k) = \phi(j), \quad \phi'(l) = \phi(l), \quad \forall l \neq j, k\}$$

- Cost

$$c_\phi(\delta^{jk}) = c(\delta^{jk}(\phi)) - c(\phi)$$
$$= \| [b_j, b_k] \cap [a_{\phi(j)}, a_{\phi(k)}] \|$$

- **Theorem:** Let ϕ^* be a permutation that ranks the a that is $k > j$ implies $a_{\phi(k)} \geq a_{\phi(j)}$ then

$$c(\phi^*) = \min_\phi c(\phi)$$

.

- **Lemma:** If ϕ is a permutation consisting of cycles $C_1, \ldots, C_p$ and δ^{jk} is an interchange with $j \in C_r$ and $k \in C_s$, $r \neq s$, then $\delta^{jk}(\phi)$ contains the same cycles except that C_r and C_s have been replaced by a single cycle containing all their nodes.

- **Theorem:** Let $\delta^{j_1 k_1}, \delta^{j_2 k_2}, \ldots, \delta^{j_p k_p}$ be the interchanges corresponding to the arcs of a spanning tree of G_{ϕ^*}. The arcs may be taken in any order. Then ϕ',

$$\phi' = \delta^{j_1 k_1} \circ \delta^{j_2 k_2} \circ \ldots \circ \delta^{j_p k_p}(\phi^*)$$

is a tour.

- The $p - 1$ interchanges can be found by greedy algorithm (similarity to Kruskal for min spanning tree)
- **Lemma:** There is a minimum spanning tree in G_{ϕ^*} that contains only arcs $\delta^{j,j+1}$.
- Generally, $c(\phi') \neq c(\delta^{j_1 k_1}) + c(\delta^{j_2 k_2}) + \ldots + c(\delta^{j_p k_p})$.

- node j in ϕ is of $\begin{cases} \text{Type I}, & \text{if } b_j \le a_{\phi(j)} \\ \text{Type II}, & \text{otherwise} \end{cases}$

  interchange jk is of $\begin{cases} \text{Type I}, & \text{if lower node of type I} \\ \text{Type II}, & \text{if lower node of type II} \end{cases}$
- Order:
  interchanges in Type I in decreasing order
  interchanges in Type II in increasing order
- Apply to ϕ^* interchanges of Type I and Type II in that order.
- **Theorem:** The tour found is a minimal cost tour.

Resuming the final algorithm [Gilmore and Gomory, 1964]:

Step 1: Arrange b_j in order of size and renumber jobs so that $b_j \le b_{j+1}$, $j = 1, \ldots, n$.

Step 2: Arrange a_j in order of size.

Step 3: Define ϕ by $\phi(j) = k$ where k is the $j+1$-smallest of the a_j.

Step 4: Compute the interchange costs $c_{\delta^{j,j+1}}$, $j = 0, \ldots, n-1$

$$c_{\delta^{j,j+1}} = \| [b_j, b_{j+1}] \cap [a_{\phi(j)}, a_{\phi(i)}] \|$$

Step 5: While G has not one single component, Add to G_ϕ the arc of minimum cost $c(\delta^{j,j+1})$ such that j and $j+1$ are in two different components.

Step 6: Divide the arcs selected in Step 5 in Type I and II.
Sort Type I in decreasing and Type II increasing order of index.
Apply the relative interchanges in the order.

Summary

Single Machine Models:

$1 \,||\, \sum w_j C_j$: weighted shortest processing time first is optimal

$1 \,|\, prec \,|\, L_{max}$: dynamic programming in $O(n^2)$

$1 \,||\, \sum h_j(C_j)$: dynamic programming in $O(2^n)$

$1 \,|\, s_{jk} \,|\, C_{max}$: in the special case, Gilmore and Gomory algorithm optimal in $O(n^2)$

Part IX

Single and Parallel Machine Models

Outline

$1 \,||\, \sum w_j C_j$: weighted shortest processing time first is optimal

$1 \,|\, prec \,|\, L_{max}$: backward dynamic programming in $O(n^2)$ [Lawler, 1973]

$1 \,||\, \sum h_j(C_j)$: dynamic programming in $O(2^n)$

$1 \,|\, s_{jk} \,|\, C_{max}$: in the special case, Gilmore and Gomory algorithm optimal in $O(n^2)$

$1 \,||\, \sum w_j C_j$: weighted shortest processing time first is optimal

$1 \,|\, prec \,|\, L_{max}$: backward dynamic programming in $O(n^2)$ [Lawler, 1973]

$1 \,|\, r_j, (prec) \,|\, L_{max}$ branch and bound

$1 \,||\, \sum_j U_j$ Moore's algorithm

$1 \,||\, \sum w_j T_j$ branch and Bound, Dynasearch

$1 \,||\, \sum h_j(C_j)$: dynamic programming in $O(2^n)$

$1 \,|\, s_{jk} \,|\, C_{max}$: in the special case, Gilmore and Gomory algorithm optimal in $O(n^2)$

$Pm \,|\, prmp \,|\, C_{max}$ Linear Programming, dispatching rules

Outline

$1 \,|\, s_{jk} \,|\, C_{max}$

[Makespan with sequence-dependent setup]
Resuming the final algorithm [Gilmore and Gomory, 1964]:

Step 1: Arrange b_j in order of size and renumber jobs so that $b_j \le b_{j+1}$, $j = 1, \ldots, n$.

Step 2: Arrange a_j in order of size.

Step 3: Define ϕ by $\phi(j) = k$ where k is the $j+1$-smallest of the a_j.

Step 4: Compute the interchange costs $c_{\delta^{j,j+1}}$, $j = 0, \ldots, n-1$

$$c_{\delta^{j,j+1}} = \| [b_j, b_{j+1}] \cap [a_{\phi(j)}, a_{\phi(i)}] \|$$

Step 5: While G has not one single component, Add to G_ϕ the arc of minimum cost $c(\delta^{j,j+1})$ such that j and $j+1$ are in two different components.

Step 6: Divide the arcs selected in Step 5 in Type I and II.
Sort Type I in decreasing and Type II increasing order of index.
Apply the relative interchanges in the order.

$1 \,|\, r_j \,|\, L_{max}$

[Maximum lateness with release dates]

- Strongly NP-hard (reduction from 3-partition)
- might have optimal schedule which is not non-delay
- Branch and bound algorithm (valid also for $1 \,|\, r_j, prec \,|\, L_{max}$)
  - **Branching**:
    schedule from the beginning (level k, $n!/(k-1)!$ nodes)
    elimination criterion: do not consider job j_k if:

    $$r_j > \min_{l \in J} \{\max(t, r_l) + p_l\} \qquad J \text{ jobs to schedule, } t \text{ current time}$$
  - **Lower bounding**: relaxation to preemptive case for which EDD is optimal

Branch and Bound

S root of the branching tree

```
LIST := {S};
U:=value of some heuristic solution;
current_best := heuristic solution;
while LIST ≠ ∅
   Choose a branching node k from LIST;
   Remove k from LIST;
   Generate children child(i), i = 1,...,n_k, and calculate corresponding
      lower bounds LB_i;
   for i:=1 to n_k
     if LB_i < U then
       if child(i) consists of a single solution then
         U:=LB_i;
         current_best:=solution corresponding to child(i)
       else add child(i) to LIST
```


$1 \,||\, \sum_j U_j$

[Number of tardy jobs]

- [Moore, 1968] algorithm in $O(n \log n)$
  - Add jobs in increasing order of due dates
  - If inclusion of job j^* results in this job being completed late discard the scheduled job k^* with the longest processing time
- $1 \,||\, \sum_j w_j U_j$ is a knapsack problem hence NP-hard

$1 \,||\, \sum w_j T_j$

[single-machine total weighted tardiness]

- $1 \,||\, \sum T_j$ is hard in ordinary sense, hence admits a pseudo polynomial algorithm (dynamic programming)
- $1 \,||\, \sum w_j T_j$ strongly NP-hard
  - branch and bound
  - time indexed integer program
  - dynaserach

Branch and bound

- **Branching:**
  - work backward in time
  - elimination criterion:
    if $p_j \le p_k$ and $d_j \le d_k$ and $w_j \ge w_k$ then there is an optimal schedule with j before k
- **Lower Bounding:**
  relaxation to preemptive case
  transportation problem

$$\min \sum_{j=1}^{n} \sum_{t=1}^{C_{max}} c_{jt} x_{jt}$$
$$\text{s.t.} \sum_{t=1}^{C_{max}} x_{jt} = p_j, \qquad \forall j = 1, \ldots, n$$
$$\sum_{j=1}^{n} x_{jt} \le 1, \qquad \forall t = 1, \ldots, C_{max}$$
$$x_{jt} \ge 0 \qquad \forall j = 1, \ldots, n; \; t = 1, \ldots, C_{max}$$

[Pan and Shi, 2007]'s lower bounding through time indexed
Stronger but computationally more expensive

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j} h_j(t + p_j) y_{jt}$$
$$\text{s.t.} \sum_{t=1}^{T-p_j} y_{jt} = 1, \qquad \forall j = 1, \ldots, n$$
$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} y_{jt} \le 1, \qquad \forall t = 1, \ldots, C_{max}$$
$$y_{jt} \ge 0 \qquad \forall j = 1, \ldots, n; \; t = 1, \ldots, C_{max}$$

Dynasearch

- Two interchanges δ_{jk} and δ_{lm} are independent if $\max\{j,k\} < \min\{l,m\}$ or $\min\{l,k\} > \max\{l,m\}$.
- The dynasearch neighborhood is obtained by a series of independent interchanges
- It has size $2^{n-1} - 1$ but a best move can be found in $O(n^3)$.
- It yields in average better results than the interchange neighborhood alone.
- Searched by dynamic programming

- state (k, π)
- π_k is the partial sequence at state (k, π) that has min $\sum wT$
- π_k is obtained from state (i, π)

$$\begin{cases} \text{appending job } \pi(k) \text{ after } \pi(i) & i = k-1 \\ \text{appending job } \pi(k) \text{ and interchanging } \pi(i+1) \text{ and } \pi(k) & 0 \le i < k-1 \end{cases}$$

- $F(\pi_0) = 0; \qquad F(\pi_1) = w_{\pi(1)} \left(p_{\pi(1)} - d_{\pi(1)}\right)^+;$

$$F(\pi_k) = \min \begin{cases} F(\pi_{k-1}) + w_{\pi(k)} \left(C_{\pi(k)} - d_{\pi(k)}\right)^+, \\ \min\limits_{1 \le i < k-1} \{F(\pi_i) + w_{\pi(k)} \left(C_{\pi(i)} + p_{\pi(k)} - d_{\pi(k)}\right)^+ + \\ \quad + \sum_{j=i+2}^{k-1} w_{\pi(j)} \left(C_{\pi(j)} + p_{\pi(k)} - p_{\pi(i+1)} - d_{\pi(j)}\right)^+ + \\ \quad + w_{\pi(i+1)} \left(C_{\pi(k)} - d_{\pi(i+1)}\right)^+ \} \end{cases}$$

- The best choice is computed by recursion in $O(n^3)$ and the optimal series of interchanges for $F(\pi_n)$ is found by backtrack.
- Local search with dynasearch neighborhood starts from an initial sequence, generated by ATC, and at each iteration applies the best dynasearch move, until no improvement is possible (that is, $F(\pi_n^t) = F(\pi_n^{(t-1)})$, for iteration t).
- Speedups:
  - pruning with considerations on $p_{\pi(k)}$ and $p_{\pi(i+1)}$
  - maintainig a string of late, no late jobs
  - h_t largest index s.t. $\pi^{(t-1)}(k) = \pi^{(t-2)}(k)$ for $k = 1, \ldots, h_t$ then $F(\pi_k^{(t-1)}) = F(\pi_k^{(t-2)})$ for $k = 1, \ldots, h_t$ and at iter t no need to consider $i < h_t$.

Dynasearch, refinements:

- [Grosso et al. 2004] add insertion moves to interchanges.
- [Ergun and Orlin 2006] show that dynasearch neighborhood can be searched in $O(n^2)$.

Performance:

- exact solution via branch and bound feasible up to 40 jobs [Potts and Wassenhove, Oper. Res., 1985]
- exact solution via time-indexed integer programming formulation used to lower bound in branch and bound solves instances of 100 jobs in 4-9 hours [Pan and Shi, Math. Progm., 2007]
- dynasearch: results reported for 100 jobs within a 0.005% gap from optimum in less than 3 seconds [Grosso et al., Oper. Res. Lett., 2004]

Complexity resume

Single machine, single criterion problems $1 \,||\, \gamma$:

C_{max}	$\mathcal{P}$
T_{max}	$\mathcal{P}$
L_{max}	$\mathcal{P}$
h_{max}	$\mathcal{P}$
$\sum C_j$	$\mathcal{P}$
$\sum w_j C_j$	$\mathcal{P}$
$\sum U$	$\mathcal{P}$
$\sum w_j U_j$	weakly $\mathcal{NP}$-hard
$\sum T$	weakly $\mathcal{NP}$-hard
$\sum w_j T_j$	strongly $\mathcal{NP}$-hard
$\sum h_j(C_j)$	strongly $\mathcal{NP}$-hard

Extensions

Non regular objectives

- $1 \,|\, d_j = d \,|\, \sum E_j + \sum T_j$
- In an optimal schedule,
  - early jobs are scheduled according to LPT
  - late jobs are scheduled according to SPT

Multicriteria scheduling

Resolution process and decision maker intervention:

- a priori methods (definition of weights, importance)
  - goal programming
  - weighted sum
  - ...
- interactive methods
- a posteriori methods (Pareto optima)
  - lexicographic with goals
  - ...

Outline

$Pm \,||\, C_{max}$ (without Preemption)

$Pm \,||\, C_{max}$ LPT heuristic, approximation ratio: $\frac{4}{3} - \frac{1}{3m}$

$P\infty \,||\, C_{max}$ CPM

$Pm \,|\, prec \,|\, C_{max}$ strongly NP-hard, LNS heuristic (non optimal)

$Pm \,|\, p_j = 1, M_j \,|\, C_{max}$ LFJ-LFM heuristic (if M_j are nested, then LFJ is optimal)

$Pm \,|\, prmp \,|\, C_{max}$

Not NP hard:

- Linear Programming, x_{ij}: time job j in machine i
- Construction based on lower bound

$$LWB = \max\left\{p_1, \sum_{j=1}^{n} \frac{p_j}{m}\right\}$$

- Dispatching rule: longest remaining processing time (LRPT) optimal in discrete time

$Qm \,|\, prmp \,|\, C_{max}$

- Construction based on

$$LWB = \max\left\{\frac{p_1}{v_1}, \frac{p_1+p_2}{v_1+v_2}, \dots, \frac{\sum_{j=1}^{n} p_j}{\sum_{j=1}^{m} v_j}\right\}$$

- Dispatching rule: longest remaining processing time on the fastest machine first (processor sharing) optimal in discrete time

Part X

Parallel Machine and Flow Shop Models

Outline

Outline

Complexity resume

Single machine models

$1 \,||\, C_{max}$	$\mathcal{P}$	
$1 \,|\, s_{jk} \,|\, C_{max}$	$\mathcal{P}$	Gilmore and Gomory's alg. in $O(n^2)$
$1 \,||\, T_{max}$	$\mathcal{P}$	
$1 \,||\, L_{max}$	$\mathcal{P}$	
$1 \,|\, prec \,|\, L_{max}$	$\mathcal{P}$	Lawler's alg. (Backward dyn. progr.)
$1 \,|\, r_j, (prec) \,|\, L_{max}$	strongly $\mathcal{NP}$-hard	Branch and Bound
$1 \,||\, h_{max}$	$\mathcal{P}$	
$1 \,||\, \sum C_j$	$\mathcal{P}$	
$1 \,||\, \sum w_j C_j$	$\mathcal{P}$	WSPT
$1 \,||\, \sum U$	$\mathcal{P}$	Moore's algorithm
$1 \,||\, \sum w_j U_j$	weakly $\mathcal{NP}$-hard	
$1 \,||\, \sum T$	weakly $\mathcal{NP}$-hard	
$1 \,||\, \sum w_j T_j$	strongly $\mathcal{NP}$-hard	Branch and Bound, Dynasearch
$1 \,||\, \sum h_j(C_j)$	strongly $\mathcal{NP}$-hard	Dynamic programming in $O(2^n)$

Branch and Bound

[Jens Clausen (1999). Branch and Bound Algorithms - Principles and Examples.]

- Eager Strategy:
  based on the bound value of the subproblems
  1. select a node
  2. branch
  3. for each subproblem compute bounds and compare with current best solution
  4. discard or store nodes together with their bounds

  (Bounds are calculated as soon as nodes are available)
- Lazy Strategy:
  often used when selection criterion for next node is max depth
  1. select a node
  2. compute bound
  3. branch
  4. store the new nodes together with the bound of the processed node

Components

- Initial good feasible solution (heuristic) – might be crucial!
- Bounding function
- Strategy for selecting
- Branching

Bounding

$$\min_{s \in P} g(s) \le \left\{ \begin{array}{l} \min_{s\in P} f(s) \\ \min_{s\in S} g(s) \end{array} \right\} \le \min_{s \in S} f(s)$$

P: candidate solutions; $S \subseteq P$ feasible solutions

- relaxation: $\min_{s\in P} f(s)$
- solve (to optimality) in P but with g

Strategy for selecting next subproblem

- best first
  (combined with eager strategy)
- breadth first
  (memory problems)
- depth first
  works on recursive updates (hence good for memory)
  but might compute a large part of the tree which is far from optimal
  (enhanced by alternating search in lowest and largest bounds combined with branching on the node with the largest difference in bound between the children)
  (it seems to perform best)

Branch and bound *vs* backtracking

- $=$ a state space tree is used to solve a problem.
- $\neq$ branch and bound does not limit us to any particular way of traversing the tree (backtracking is depth-first)
- $\neq$ branch and bound is used only for optimization problems.

Branch and bound *vs* A*

- $=$ In A* the admissible heuristic mimics bounding
- $\neq$ In A* there is no branching. It is a search algorithm.
- $\neq$ A* is best first

Dynasearch

- Two interchanges δ_{jk} and δ_{lm} are independent if $\max\{j,k\} < \min\{l,m\}$ or $\min\{l,k\} > \max\{l,m\}$.
- The dynasearch neighborhood is obtained by a series of independent interchanges
- It has size $2^{n-1} - 1$ but a best move can be found in $O(n^3)$.
- It yields in average better results than the interchange neighborhood alone.
- Searched by dynamic programming

- state (k, π)
- π_k is the partial sequence at state (k, π) that has min $\sum wT$
- π_k is obtained from state (i, π)

$$\begin{cases} \text{appending job } \pi(k) & i = k-1 \\ \text{appending job } \pi(k) \text{ and interchanging } \pi(i+1) \text{ and } \pi(k) & 0 \leq i < k-1 \end{cases}$$

- $F(\pi_0) = 0; \qquad F(\pi_1) = w_{\pi(1)} \left(p_{\pi(1)} - d_{\pi(1)}\right)^+;$

$$F(\pi_k) = \min \begin{cases} F(\pi_{k-1}) + w_{\pi(k)} \left(C_{\pi(k)} - d_{\pi(k)}\right)^+, \\ \min\limits_{1 \leq i < k-1} \{F(\pi_i) + w_{\pi(k)} \left(C_{\pi(i)} + p_{\pi(k)} - d_{\pi(k)}\right)^+ + \\ \quad + \sum_{j=i+2}^{k-1} w_{\pi(j)} \left(C_{\pi(j)} + p_{\pi(k)} - p_{\pi(i+1)} - d_{\pi(k)}\right)^+ + \\ \quad + w_{\pi(i+1)} \left(C_{\pi(k-1)} - p_{\pi(i+1)} + p_{\pi(k)} - d_{\pi(k)}\right)^+\} \end{cases}$$

- The best choice is computed by recursion in $O(n^3)$ and the optimal series of interchanges for $F(\pi_n)$ is found by backtrack.
- Local search with dynasearch neighborhood starts from an initial sequence, generated by ATC, and at each iteration applies the best dynasearch move, until no improvement is possible (that is, $F(\pi_n^t) = F(\pi_n^{(t-1)})$, for iteration t).
- Speedups:
  - pruning with considerations on $p_{\pi(k)}$ and $p_{\pi(i+1)}$
  - maintaining a string of late, no late jobs
  - h_t largest index s.t. $\pi^{(t-1)}(k) = \pi^{(t-2)}(k)$ for $k = 1, \ldots, h_t$ then $F(\pi_k^{(t-1)}) = F(\pi_k^{(t-2)})$ for $k = 1, \ldots, h_t$ and at iter t no need to consider $i < h_t$.

Dynasearch, refinements:

- [Grosso et al. 2004] add insertion moves to interchanges.
- [Ergun and Orlin 2006] show that dynasearch neighborhood can be searched in $O(n^2)$.

Performance:

- exact solution via branch and bound feasible up to 40 jobs [Potts and Wassenhove, Oper. Res., 1985]
- exact solution via time-indexed integer programming formulation used to lower bound in branch and bound solves instances of 100 jobs in 4-9 hours [Pan and Shi, Math. Progm., 2007]
- dynasearch: results reported for 100 jobs within a 0.005% gap from optimum in less than 3 seconds [Grosso et al., Oper. Res. Lett., 2004]

Extensions

Non regular objectives

- $1 \mid d_j = d \mid \sum E_j + \sum T_j$
- In an optimal schedule,
  - early jobs are scheduled according to LPT
  - tardy jobs are scheduled according to SPT

Multicriteria scheduling

Resolution process and decision maker intervention:

- a priori methods (definition of weights, importance)
  - goal programming
  - weighted sum
  - ...
- interactive methods
- a posteriori methods (Pareto optima)
  - lexicographic with goals
  - ...

Outline

$Pm \mid\mid C_{max}$ (without Preemption)

$Pm \mid\mid C_{max}$ LPT heuristic, approximation ratio: $\frac{4}{3} - \frac{1}{3m}$

$P\infty \mid prec \mid C_{max}$ CPM

$Pm \mid prec \mid C_{max}$ strongly NP-hard, LNS heuristic (non optimal)

$Pm \mid p_j = 1, M_j \mid C_{max}$ LFJ-LFM (optimal if M_j are nested)

$Pm \mid prmp \mid C_{max}$

Not NP hard:

- Linear Programming, x_{ij}: time job j in machine i
- Construction based on $LWB = \max \left\{p_1, \sum_{j=1}^n \frac{p_j}{m}\right\}$
- Dispatching rule: longest remaining processing time (LRPT) optimal in discrete time

$Qm \mid prmp \mid C_{max}$

- Construction based on

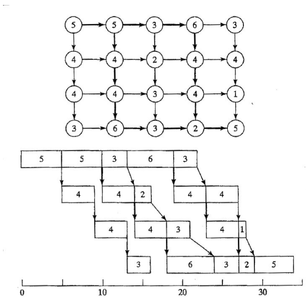
$$LWB = \max \left\{ \frac{p_1}{v_1}, \frac{p_1 + p_2}{v_1 + v_2}, \ldots, \frac{\sum_{j=1}^n p_j}{\sum_{j=1}^m v_j} \right\}$$

- Dispatching rule: longest remaining processing time on the fastest machine first (processor sharing) optimal in discrete time

Outline

Flow Shop

- Buffer limited, unlimited
- Permutation Flow Shop
- Directed graph representation
- C_{max} computation (critical path length)



Exact Solutions

- **Theorem:** There always exist an optimum without sequence change in the first two and last two machines. (hence $F2 \mid\mid C_{max}$ and $F3 \mid\mid C_{max}$ are permutation flow shop)
- $F2 \mid\mid C_{max}$: Johnson's rule (1954)
  - Set I: $p_{1j} < p_{2j}$, order in increasing p_{1j}, SPT(1)
  - Set II: $p_{2j} < p_{1j}$, order in decreasing p_{2j}, LPT(2)
- $F3 \mid\mid C_{max}$ is strongly NP-hard

$Fm \mid prmu, p_{ij} = p_j \mid C_{max}$

[Proportionate permutation flow shop]

- **Theorem:** $C_{max} = \sum_{j=1}^n p_j + (m-1)\max(p_1, \ldots, p_n)$ and is sequence independent
- Generalization to include machines with different speed: $p_{ij} = p_j / v_i$

  Theorem:
  if the first machine is the bottleneck then LPT is optimal.
  if the last machine is the bottleneck then SPT is optimal.

Construction Heuristics for $Fm \mid prmu \mid C_{max}$

Slope heuristic

- schedule in decreasing order of $A_j = -\sum_{i=1}^m (m - (2i-1)) p_{ij}$

Campbell, Dudek and Smith's heuristic (1970)

extension of Johnson's rule to when permutation is not dominant

- recursively create 2 machines 1 and $m-1$

$$p'_{ij} = \sum_{k=1}^{i} p_{kj} \qquad p''_{ij} = \sum_{k=m-i+1}^{m} p_{kj}$$

  and use Johnson's rule
- repeat for all $m-1$ possible pairings
- return the best for the overall m machine problem

Nawasz, Enscore, Ham's heuristic (1983)

- **Step 1**: order in decreasing $\sum_{j=1}^m p_{ij}$
- **Step 2**: schedule the first 2 jobs at best
- **Step 3**: insert all others in best position

Implementation in $O(n^2 m)$

Framinan, Gupta, Leisten (2004) examined 177 different arrangements of jobs in Step 1 and concluded that the NEH arrangement is the best one for C_{max}.

Metaheuristics for $Fm \mid prmu \mid C_{max}$

Iterated Greedy [Ruiz, Stützle, 2007]

- Destruction: remove d jobs at random
- Construction: reinsert them with NEH heuristic in the order of removal
- Local Search: insertion neighborhood
  (first improvement, whole evaluation $O(n^2m)$)
- Acceptance Criterion: random walk, best, SA-like

Performance on up to $n = 500 \times m = 20$:

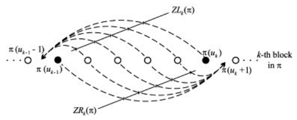
- NEH average gap 3.35% in less than 1 sec.
- IG average gap 0.44% in about 360 sec.

Tabu Search

[Novicki, Smutnicki, 1994, Grabowski, Wodecki, 2004]

- C_{max} expression through critical path
- Block B_k, definition
- Internal block B_k^{Int}, definition
- **Theorem:** Let $\pi, \pi' \in \Pi$, if π' has been obtained from π by an interchange of jobs so that $C_{max}(\pi') < C_{max}(\pi)$ then in π':
  - a) at least one job $j \in B_k$ precedes job $\pi(u_{k-1}), k = 1, \ldots, m$
  - b) at least one job $j \in B_k$ succeeds job $\pi(u_k), k = 1, \ldots, m$

- Insert neighborhood
- Tabu search requires a best strategy. How to search efficiently?
- **Theorem:** (Elimination Criterion) If π' is obtained by π by a "block insertion" then $C_{max}(\pi') \leq C_{max}(\pi)$.
- Define good moves:



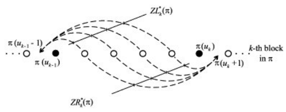
- Use of lower bounds in delta evaluations:

$$D_{ka}(x) = \begin{cases} p_{\pi(x),k+1} - p_{\pi(u_k),k+1} & x \neq u_{k-1} \\ p_{\pi(x),k+1} - p_{\pi(u_k),k+1} + p_{\pi(u_{k-1}+1,k} - p_{\pi(x),k} & x = u_{k-1} \end{cases}$$

$$C_{max}(\delta_x(\pi)) \geq C_{max}(\pi) + D_{ka}(x)$$

- Prohibition criterion:
  an insertion δ_{x,u_k} is tabu if it restores the realtive order of $\pi(x)$ and $\pi(x+1)$.
- Tabu length: $TL = 6 + \left[\frac{n}{10m}\right]$

- Perturbation



  - perform all interchanges among all the blocks that have $D < 0$
  - activated after `MaxIdleIter` idle iterations

Tabu Search: the final algorithm:

Initialization	:	$\pi = \pi_0$, $C^* = C_{max}(\pi)$, set iteration counter to zero.
Searching	:	Create UR_k and UL_k (set of non tabu moves)
Selection	:	Find the best move according to lower bound D. Compute $C_{max}(\delta(\pi))$. Apply move. If improving compare with C^* and in case update. Else increase number of idle iterations.
Stop criterion	:	Exit if `MaxIter` iterations are done.
Perturbation	:	Apply perturbation if `MaxIdleIter` done.

Part XI

Flow Shop and Job Shop Models

Outline

Outline

Resume

Permutation Flow Shop:

- Directed graph representation and C_{max} computation
- Johnson's rule for $F2 \mid \mid C_{max}$
- Construction heuristics:
  - Slope heuristic
  - Campbell, Dudeck and Smith's heuristic
  - Nawasz, Enscore and Ham's heuristic

Outline

$Jm \mid \mid C_{max}$

[Job shop makespan]
Given:

- $J = \{1, \ldots, N\}$ set of jobs
- $M = \{1, \ldots, m\}$ set of machines
- $J_j = \{O_{ij} \mid i = 1, \ldots, n_j\}$ set of operations for each job
- $O_{1j} \to O_{2j} \to \ldots \to O_{n_j,j}$ precedences (without loss of generality)
- p_{ij} processing times of operations O_{ij}
- $\mu_{ij} \in \{M_1, \ldots, M_m\}$ with $\mu_{ij} \neq \mu_{i+1,j}$ eligibility for each operations (one machine per operation)
- without repetition and with unlimited buffers

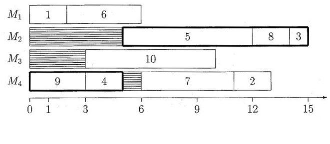
Task:

- Find a schedule $S = (S_{ij})$, indicating the starting times of O_{ij}, such that:
  it is feasible, that is,
  - $S_{ij} + p_{ij} \leq S_{i+1,j}$ for all $O_{ij} \to O_{i+1,j}$
  - $S_{ij} + p_{ij} \leq S_{uv}$ or $S_{uv} + p_{uv} \leq S_{ij}$ for all operations with $\mu_{ij} = \mu_{uv}$.

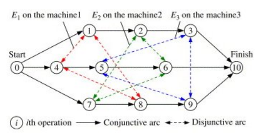
  and has minimum makespan.

A schedule can be also represented by an m-tuple $\pi = (\pi^1, \pi^2, \ldots, \pi^m)$ where π^i defines the processing order on machine i.

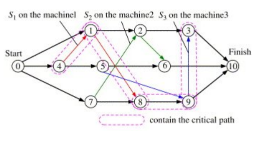
Then a semi-active schedule is found by computing the feasible earliest start time for each operation in π.



- Often simplified notation: $N = \{1, \ldots, n\}$ denotes the set of operations
- Disjunctive graph representation: $G = (N, A, E)$
  - vertices N: operations with two dummy operations 0 and $n+1$ denoting "start" and "finish".
  - directed arcs A, conjunctions
  - undirected arcs E, disjunctions
  - length of (i, j) in A is p_i



- A complete selection corresponds to choosing one direction for each arc of E.
- A complete selection that makes D acyclic corresponds to a feasible schedule and is called consistent.
- Complete, consistent selection $\Leftrightarrow$ semi-active schedule (feasible earliest start schedule).
- Length of longest path 0–$(n+1)$ in D corresponds to the makespan



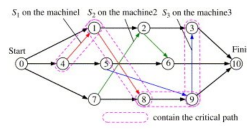
Longest path computation

In an acyclic digraph:

- construct topological ordering ($i < j$ forall $i \to j \in A$)
- recursion:

$$r_0 = 0$$
$$r_l = \max_{\{j \mid j \to l \in A\}} \{r_j + p_j\} \qquad \text{for } l = 1, \ldots, n+1$$

- A block is a maximal sequence of adjacent critical operations processed on the same machine.
- In the Fig. below: $B_1 = \{4, 1, 8\}$ and $B_2 = \{9, 3\}$



- Any operation, u, has two immediate predecessors and successors:
  - its job predecessor $JP(u)$ and successor $JS(u)$
  - its machine predecessor $MP(u)$ and successor $MS(u)$

Exact methods

- Disjunctive programming

$$
\begin{array}{lll}
\min & C_{max} & \\
\text{s.t.} & x_{ij} + p_{ij} \leq C_{max} & \forall\, O_{ij} \in N \\
 & x_{ij} + p_{ij} \leq x_{lj} & \forall\, (O_{ij}, O_{lj}) \in A \\
 & x_{ij} + p_{ij} \leq x_{ik} \vee x_{ij} + p_{ij} \leq x_{ik} & \forall\, (O_{ij}, O_{ik}) \in E \\
 & x_{ij} \leq 0 & \forall\, i = 1, \ldots, m \;\; j = 1, \ldots, N
\end{array}
$$

- Constraint Programming
- Branch and Bound [Carlier and Pinson, 1983]

Typically unable to schedule optimally more than 10 jobs on 10 machines.
Best result is around 250 operations.

Shifting Bottleneck Heuristic

- A complete selection is made by the union of selections S_k for each clique E_k that corresponds to machines.
- **Idea**: use a priority rule for ordering the machines.
chose each time the bottleneck machine and schedule jobs on that machine.
- Measure bottleneck quality of a machine k by finding optimal schedule to a certain single machine problem.
- Critical machine, if at least one of its arcs is on the critical path.

- $M_0 \subset M$ set of machines already sequenced.
- $k \in M \setminus M_0$
- $P(k, M_0)$ is problem $1 \mid r_j \mid L_{max}$ obtained by:
  - the selections in M_0
  - removing any disjunctive arc in $p \in M \setminus M_0$
- $v(k, M_0)$ is the optimum of $P(k, M_0)$
- bottleneck $m = \arg \max_{k \in M \setminus M_0} \{v(k, M_0)\}$
- $M_0 = \emptyset$

Step 1: Identify bottleneck m among $k \in M \setminus M_0$ and sequence it optimally. Set $M_0 \leftarrow M_0 \cup \{m\}$

Step 2: Reoptimize the sequence of each critical machine $k \in M_0$ in turn: set $M'_o = M_0 - \{k\}$ and solve $P(k, M'_0)$.
Stop if $M_0 = M$ otherwise Step 1.

- Local Reoptimization Procedure

Construction of $P(k, M_0)$

$1 \mid r_j \mid L_{max}$:

- $r_j = L(0, j)$
- $d_j = L(0, n) - L(j, n) + p_j$

$L(i, j)$ length of longest path in G: Computable in $O(n)$

An acyclic complete directed graph is the transitive closure of its unique directed Hamilton path.

Hence, only predecessors and successor are to be checked.
The graph is not constructed explicitly, but by maintaining a list of jobs per machines and a list machines per jobs.

$1 \mid r_j \mid L_{max}$ can be solved optimally very efficiently.
Results reported up to 1000 jobs.

$1 \mid r_j \mid L_{max}$ — From Lecture 9

[Maximum lateness with release dates]

- Strongly NP-hard (reduction from 3-partition)
- might have optimal schedule which is not non-delay
- Branch and bound algorithm (valid also for $1 \mid r_j, prec \mid L_{max}$)
  - **Branching**:
schedule from the beginning (level k, $n!/(k-1)!$ nodes)
elimination criterion: do not consider job j_k if:

$$r_j > \min_{l \in J} \{\max(t, r_l) + p_l\} \qquad J \text{ jobs to schedule, } t \text{ current time}$$

  - **Lower bounding**: relaxation to preemptive case for which EDD is optimal

Tabu Search for Job Shop

Neighborhoods

Change the orientation of certain disjunctive arcs of the current complete selection

Issues:

1. Can it be decided easily if the new disjunctive graph $G(S')$ is acyclic?
2. Can the neighborhood selection S' improve the makespan?
3. Is the neighborhood connected?

Swap Neighborhood [Novicki, Smutnicki]

Reverse one oriented disjunctive arc (i, j) on some critical path.

Theorem: All neighbors are consistent selections.

Note: If the neighborhood is empty then there are no disjunctive arcs, nothing can be improved and the schedule is already optimal.

Theorem: The swap neighborhood is connected.

Insertion Neighborhood [Balas, Vazacopoulos, 1998]

For some nodes u, v in the critical path:

- move u right after v (forward insert)
- move v right before u (backward insert)

Theorem: If a critical path containing u and v also contain $JS(v)$ and

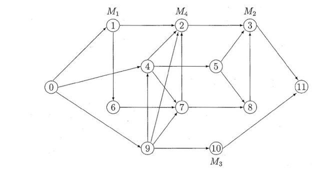
$$L(v, n) \geq L(JS(u), n)$$

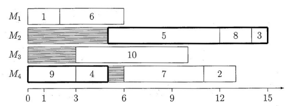
then a forward insert of u after v yields an acyclic complete selection.

Theorem: If a critical path containing u and v also contain $JS(v)$ and

$$L(0, u) + p_u \geq L(0, JP(v)) + p_{JP(v)}$$

then a backward insert of v before v yields an acyclic complete selection.





Theorem: (Elimination criterion) If $C_{max}(S') < C_{max}(S)$ then at least one operation of a machine block B on the critical path has to be processed before the first or after the last operation of B.

- Swap neighborhood can be restricted to first and last operations in the block
- Insert neighborhood can be restricted to moves similar to those saw for the flow shop. [Grabowski, Wodecki]

Tabu Search requires a best improvement strategy hence the neighborhood must be search very fast.

Neighbor evaluation:

- exact recomputation of the makespan $O(n)$
- approximate evaluation (rather involved procedure but much faster and effective in practice)

The implementation of Tabu Search follows the one saw for flow shop.

Part XII

Job Shop and Resource Constrained Project Scheduling

Outline

Resume

Flow Shop

- Iterated Greedy
- Tabu Search (block representation and neighborhood pruning)

Job Shop:

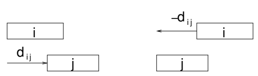
- Definition
- Starting times and m-tuple permutation representation
- Disjunctive graph representation [Roy and Sussman, 1964]
- Shifting Bottleneck Heuristic [Adams, Balas and Zawack, 1988]

Outline

27. Job Shop Generalizations

28. Resource Constrained Project Scheduling Model

Generalizations: Time Lags



Generalized time constraints

They can be used to model:

- Release time:

$$S_0 + r_i \leq S_i \qquad \Longleftrightarrow \qquad d_{0i} = r_i$$

- Deadlines:

$$S_i + p_i - d_i \leq S_0 \qquad \Longleftrightarrow \qquad d_{i0} = p_i - d_i$$

- Modelling

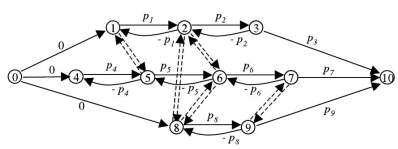
$$
\begin{array}{lll}
\min & C_{max} & \\
\text{s.t.} & x_{ij} + d_{ij} \leq C_{max} & \forall\, O_{ij} \in N \\
 & x_{ij} + d_{ij} \leq x_{lj} & \forall\, (O_{ij}, O_{lj}) \in A \\
 & x_{ij} + d_{ij} \leq x_{ik} \vee x_{ij} + d_{ij} \leq x_{ik} & \forall\, (O_{ij}, O_{ik}) \in E \\
 & x_{ij} \geq 0 & \forall\, i = 1, \ldots, m \;\; j = 1, \ldots, N
\end{array}
$$

- In the disjunctive graph, d_{ij} become the lengths of arcs

- Exact relative timing (perishability constraints):
if operation j must start l_{ij} after operation i:

$$S_i + p_i + l_{ij} \leq S_j \qquad \text{and} \qquad S_j - (p_i + l_{ij}) \leq S_i$$

($l_{ij} = 0$ if no-wait constraint)



- Set up times:

$$S_i + p_i + s_{ij} \leq S_j \quad \text{or} \quad S_j + p_j + s_{ji} \leq S_i$$

- Machine unavailabilities:
  - Machine M_k unavailable in $[a_1, b_1]$, $[a_2, b_2]$, ... , $[a_v, b_v]$
  - Introduce v artificial operations with $\lambda = 1, \ldots, v$ with $\mu_\lambda = M_k$ and:
    $p_\lambda = b_\lambda - a_\lambda$
    $r_\lambda = a_\lambda$
    $d_\lambda = b_\lambda$

- Minimum lateness objectives:

$$L_{max} = \max_{j=1}^{N}\{C_j - d_j\} \quad \Longleftrightarrow \quad d_{n_j, n+1} = p_{n_j} - d_j$$

Blocking

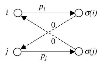
Arises with limited buffers:
after processing, a job remains on the machine until the next machine is freed

- Needed a generalization of the disjunctive graph model
  $\Longrightarrow$ Alternative graph model $G = (N, E, A)$ [Mascis, Pacciarelli, 2002]

1. two non-blocking operations to be processed on the same machine

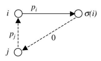
$$S_i + p_i \leq S_j \quad \text{or} \quad S_j + p_j \leq S_i$$

2. Two blocking operations i, j to be processed on the same machine $\mu(i) = \mu(j)$



$$S_{MS(j)} \leq S_i \quad \text{or} \quad S_{MS(i)} \leq S_j$$

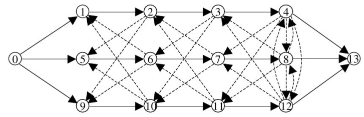
3. i is blocking, j is non-blocking (ideal) and i, j to be processed on the same machine $\mu(i) = \mu(j)$.



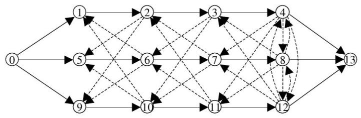
$$S_i + p_i \leq S_j \quad \text{or} \quad S_{MS(j)} \leq S_i$$

Example

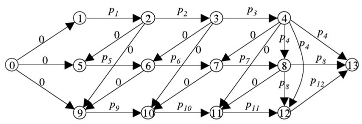
- $O_0, O_1, \ldots, O_{13}$
- $M(O_1) = M(O_5) = M(O_9)$
  $M(O_2) = M(O_6) = M(O_{10})$
  $M(O_3) = M(O_7) = M(O_{11})$



- Length of arcs can be negative
- Multiple occurrences possible: $((i, j), (u, v)) \in A$ and $((i, j), (h, k)) \in A$
- The last operation of a job j is always non-blocking.



- A complete selection S is consistent if it chooses alternatives from each pair such that the resulting graph does not contain positive cycles.



Example:

- $p_a = 4$
- $p_b = 2$
- $p_c = 1$

- b must start at least 9 days after a has started
- c must start at least 8 days after b is finished
- c must finish within 16 days after a has started

$$\begin{aligned} S_a + 9 &\leq S_b \\ S_b + 10 &\leq S_c \\ S_c - 15 &\leq S_a \end{aligned}$$

This leads to an absurd.
In the alternative graph the cycle is positive.

- The Makespan still corresponds to the longest path in the graph with the arc selection $G(S)$.

- If there are no cycles of length strictly positive it can still be computed efficiently in $O(|N||E \cup A|)$ by Bellman-Ford (1958) algorithm.

- The algorithm iteratively considers all edges in a certain order and updates an array of longest path lengths for each vertex. It stops if a loop over all edges does not yield any update or after $|N|$ iterations over all edges (in which case we know there is a positive cycle).

- Possible to maintain incremental updates when changing the selection [Demetrescu Frangioni, Marchetti-Spaccamela, Nanni, 2000].

Heuristics for the Alternative Graph Model

- The search space is highly constrained + detecting positive cycles is costly

- Hence local search methods not very successful

- Rely on the construction paradigm

- Rollout algorithm [Meloni, Pacciarelli, Pranzo, 2004]

Rollout

- Master process: grows a partial selection S^k:
  decides the next element to fix based on an heuristic function (selects the one with minimal value)

- Slave process: evaluates heuristically the alternative choices.
  Completes the selection by keeping fixed what passed by the master process and fixing one alternative at a time.

- Slave heuristics
  - *Avoid Maximum Current Completion time*
    find an arc (h, k) that if selected would increase most the length of the longest path in $G(S^k)$ and select its alternative

$$\max_{(uv) \in A} \{l(0, u) + a_{uv} + l(u, n)\}$$

  - *Select Most Critical Pair*
    find the pair that, in the worst case, would increase least the length of the longest path in $G(S^k)$ and select the best alternative

$$\max_{((ij),(hk)) \in A} \min\{l(0, u) + a_{hk} + l(k, n), l(0, i) + a_{ij} + l(j, n)\}$$

  - *Select Max Sum Pair*
    finds the pair with greatest potential effect on the length of the longest path in $G(S^k)$ and select the best alternative

$$\max_{((ij),(hk)) \in A} |l(0, u) + a_{hk} + l(k, n) + l(0, i) + a_{ij} + l(j, n)|$$

Trade off quality *vs* keeping feasibility
Results depend on the characteristics of the instance.

Outline

Resource Constrained Project Scheduling Model

Given:

- activities (jobs) $j = 1, \ldots, n$
- renewable resources $i = 1, \ldots, m$
- amount of resources available R_i
- processing times p_j
- amount of resource used r_{ij}
- precedence constraints $j \rightarrow k$

Further generalizations

- Time dependent resource profile $R_i(t)$
  given by (t_i^μ, R_i^μ) where $0 = t_i^1 < t_i^2 < \ldots < t_i^{m_i} = T$
  Disjunctive resource, if $R_k(t) = \{0, 1\}$; cumulative resource, otherwise
- Multiple modes for an activity j
  processing time and use of resource depends on its mode m: p_{jm}, r_{jkm}.

Solutions

Task: Find a schedule indicating the starting time of each activity

- All solution methods restrict the search to feasible schedules, S, S'

- Types of schedules
  - Local left shift (LLS): $S \rightarrow S'$ with $S'_j < S_j$ and $S'_l = S_l$ for all $l \neq j$.
  - Global left shift (GLS): LLS passing through infeasible schedule
  - Semi active schedule: no LLS possible
  - Active schedule: no GLS possible
  - Non-delay schedule: no GLS and LLS possible even with preemption

- If regular objectives $\Longrightarrow$ exists an optimum which is active

Hence:

- Schedule not given by start times S_i
  - space too large $O(T^n)$
  - difficult to check feasibility

- Sequence (list, permutation) of activities $\pi = (j_1, \ldots, j_n)$

- π determines the order of activities to be passed to a schedule generation scheme

Modeling

Assignment 1

- A contractor has to complete n activities.
- The duration of activity j is p_j
- each activity requires a crew of size W_j.
- The activities are not subject to precedence constraints.
- The contractor has W workers at his disposal
- his objective is to complete all n activities in minimum time.

Assignment 2

- Exams in a college may have different duration.
- The exams have to be held in a gym with W seats.
- The enrollment in course j is W_j and
- all W_j students have to take the exam at the same time.
- The goal is to develop a timetable that schedules all n exams in minimum time.
- Consider both the cases in which each student has to attend a single exam as well as the situation in which a student can attend more than one exam.

Assignment 3

- In a basic high-school timetabling problem we are given m classes $c_1, \ldots, c_m$,
- h teachers $a_1, \ldots, a_h$ and
- T teaching periods $t_1, \ldots, t_T$.
- Furthermore, we have lectures $i = l_1, \ldots, l_n$.
- Associated with each lecture is a unique teacher and a unique class.
- A teacher a_j may be available only in certain teaching periods.
- The corresponding timetabling problem is to assign the lectures to the teaching periods such that
  - each class has at most one lecture in any time period
  - each teacher has at most one lecture in any time period,
  - each teacher has only to teach in time periods where he is available.

Assignment 4

- A set of jobs $J_1, \ldots, J_g$ are to be processed by auditors $A_1, \ldots, A_m$.
- Job J_l consists of n_l tasks $(l = 1, \ldots, g)$.
- There are precedence constraints $i_1 \rightarrow i_2$ between tasks i_1, i_2 of the same job.
- Each job J_l has a release time r_l, a due date d_l and a weight w_l.
- Each task must be processed by exactly one auditor. If task i is processed by auditor A_k, then its processing time is p_{ik}.
- Auditor A_k is available during disjoint time intervals $[s_k^v, l_k^v]$ ($v = 1, \ldots, m$) with $l_k^v < s_k^v$ for $v = 1, \ldots, m_k - 1$.
- Furthermore, the total working time of A_k is bounded from below by H_k^- and from above by H_k^+ with $H_k^- \leq H_k^+$ $(k = 1, \ldots, m)$.
- We have to find an assignment $\alpha(i)$ for each task $i = 1, \ldots, n := \sum_{l=1}^{g} n_l$ to an auditor $A_{\alpha(i)}$ such that
  - each task is processed without preemption in a time window of the assigned auditor
  - the total workload of A_k is bounded by H_k^- and H_k^k for $k = 1, \ldots, m$.
  - the precedence constraints are satisfied,
  - all tasks of J_l do not start before time r_l, and
  - the total weighted tardiness $\sum_{l=1}^{g} w_l T_l$ is minimized.

Part XIII

Resource Constrained Project Scheduling. Reservations and Timetabling

Outline

Outline

Preprocessing: Temporal Analysis

- Precedence network must be acyclic
- Heads r_j and Tails q_j ⇐ Longest paths ⇐ Topological ordering (deadlines d_j can be obtained as $UB - q_j$)

Preprocessing: constraint propagation

1. conjunctions $i \to j$ $\quad S_i + p_i \leq S_j$
   [precedence constrains]
2. parallelity constraints $i \| j$ $\quad S_i + p_i \geq S_j$ and $S_j + p_j \geq S_i$
   [time windows $[r_j, d_j]$,$[r_l, d_l]$ and $p_l + p_j > \max\{d_l, d_j\} - \min\{r_l, r_j\}$]
3. disjunctions $i - j$ $\quad S_i + p_i \leq S_j$ or $S_j + p_j \leq S_i$
   [resource constraints: $r_{jk} + r_{lk} > R_k$]

N. Strengthenings: symmetric triples, etc.

Schedule Generation Schemes

Given a sequence of activity, SGS determine the starting times of each activity

Serial schedule generation scheme (SSGS)

n stages, S_λ scheduled jobs, E_λ eligible jobs

Step 1 Select next from E_λ and schedule at earliest.

Step 2 Update E_λ and $R_k(\tau)$.
If E_λ is empty then STOP,
else go to Step 1.

Parallel schedule generation scheme (PSGS)

(Time sweep)

stage λ at time t_λ

S_λ (finished activities), A_λ (activities not yet finished), E_λ (eligible activities)

Step 1 In each stage select maximal resource-feasible subset of eligible activities in E_λ and schedule it at T_λ.

Step 2 Update E_λ, A_λ and $R_k(\tau)$.
If E_λ is empty then STOP,
else move to $t_{\lambda+1} = \min \left\{ \min_{j \in A_\lambda} C_j, \min_{i \in M} t_i^\mu \right\}$
and go to Step 1.

- If constant resource, it generates non-delay schedules
- Search space of PSGS is smaller than SSGS

Dispatching Rules

Determines the sequence of activities to pass to the schedule generation scheme

- activity based
- network based
- path based
- resource based

Static *vs* Dynamic

Local Search

All typical neighborhood operators can be used:

- Swap
- Interchange
- Insert

Genetic Algorithms

Recombination operator:

- One point crossover
- Two point crossover
- Uniform crossover

Outline

Reservations without slack – Interval Scheduling

Given:

- m parallel machines (resources)
- n activities
- r_j starting times (integers),
  d_j termination (integers),
  w_j or w_{ij} weight,
  M_j eligibility
- without slack $p_j = d_j - r_j$

Task: Maximize weight of assigned activities

Examples: Hotel room reservation, Car rental

Polynomially solvable cases

1. $p_j = 1$

Solve an assignment problem at each time slot

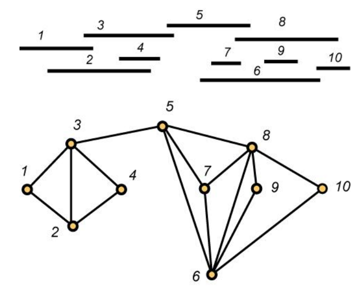
2. $w_j = 1$, $M_j = M$, Obj. minimize resources used

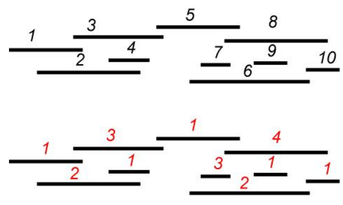
- Corresponds to coloring interval graphs with minimal number of colors
- Optimal greedy algorithm (First Fit):

order $r_1 \leq r_2 \leq \ldots \leq r_n$

Step 1 assign resource 1 to activity 1

Step 2 **for** j from 2 to n **do**
Assume k resources have been used.
Assign activity j to the resource with minimum feasible value from $\{1, \ldots, k+1\}$





3. $w_j = 1$, $M_j = M$, Obj. maximize activities assigned

- Corresponds to coloring max # of vertices in interval graphs with k colors
- Optimal k-coloring of interval graphs:

order $r_1 \leq r_2 \leq \ldots \leq r_n$
$J = \emptyset$, $j = 1$

Step 1 if a resource is available at time r_j then assign activity j to that resource;
include j in J; go to Step 3

Step 2 Else, select j^* such that $C_{j^*} = \max_{j \in J} C_j$
if $C_j = r_j + p_j > C_{j^*}$ go to Step 3
else remove j^* from J, assign j in J

Step 3 **if** $j = n$ STOP **else** $j = j + 1$ go to Step 1

Outline

Reservations with Slack

Given:

- m parallel machines (resources)
- n activities
- r_j starting times (integers),
  d_j termination (integers),
  w_j or w_{ij} weight,
  M_j eligibility
- with slack $p_j \leq d_j - r_j$

Task: Maximize weight of assigned activities

Most constrained variable, least constraining value heuristic

$|M_j|$ indicates how much constrained an activity is
ν_{it}: # activities that can be assigned to i in $[t-1, t]$

Select activity j with smallest $I_j = f\left(\frac{w_j}{p_j}, |M_j|\right)$

Select resource i with smallest $g(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j})$ (or discard j if no place free for j)

Examples for f and g:

$$f\left(\frac{w_j}{p_j}, |M_j|\right) = \frac{|M_j|}{w_j/p_j}$$

$$g(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j}) = \max(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j})$$

$$g(\nu_{i,t+1}, \ldots, \nu_{i,t+p_j}) = \sum_{l=1}^{p_j} \frac{\nu_{i,t+l}}{p_j}$$

Outline

Timetabling with Workforce or Personnel Constrains

There is only one type of operator that processes all the activities

Example:

- A contractor has to complete n activities.
- The duration of activity j is p_j
- Each activity requires a crew of size W_j.
- The activities are not subject to precedence constraints.
- The contractor has W workers at his disposal
- His objective is to complete all n activities in minimum time.

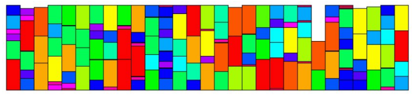
- RCPSP Model
- If p_j all the same ➜ Bin Packing Problem (still NP-hard)

Example: Exam scheduling

- Exams in a college with same duration.
- The exams have to be held in a gym with W seats.
- The enrollment in course j is W_j and
- all W_j students have to take the exam at the same time.
- The goal is to develop a timetable that schedules all n exams in minimum time.
- Each student has to attend a single exam.

- Bin Packing model
- In the more general (and realistic) case it is a RCPSP

Heuristics for Bin Packing



- Construction Heuristics
  - Best Fit Decreasing (BFD)
  - First Fit Decreasing (FFD) $C_{max}(FFD) \leq \frac{11}{9} C_{max}(OPT) + \frac{6}{9}$
- Local Search: [Alvim and Aloise and Glover and Ribeiro, 1999]
  - Step 1: remove one bin and redistribute items by BFD
  - Step 2: if infeasible, re-make feasible by redistributing items for pairs of bins, such that their total weights becomes equal (number partitioning problem)

[Levine and Ducatelle, 2004]

The solution before local search (the bin capacity is 10):
The bins: | 3 3 3 | 6 2 1 | 5 2 | 4 3 | 7 2 | 5 4 |

Open the two smallest bins:
Remaining: | 3 3 3 | 6 2 1 | 7 2 | 5 4 |
Free items: 5, 4, 3, 2

Try to replace 2 current items by 2 free items, 2 current by 1 free or 1 current by 1 free:
First bin: 3 3 3 → 3 5 2 new free: 4, 3, 3, 3
Second bin: 6 2 1 → 6 4 new free: 3, 3, 3, 2, 1
Third bin: 7 2 → 7 3 new free: 3, 3, 2, 2, 1
Fourth bin: 5 4 stays the same

Reinsert the free items using FFD:
Fourth bin: 5 4 → 5 4 1
Make new bin: 3 3 2 2
Final solution: | 3 5 2 | 6 4 | 7 3 | 5 4 1 | 3 3 2 2 |

Repeat the procedure: no further improvement possible

Outline

Timetabling with Different Operator or Tools

- There are several operators and activities can be done by an operator only if he is available
- Two activities that share an operator cannot be scheduled at the same time

Examples:

- aircraft repairs
- scheduling of meetings (people ➜ operators; resources ➜ rooms)
- exam scheduling (students may attend more than one exam ➜ operators)

If $p_j = 1$ ➜ Graph-Vertex Coloring (still NP-hard)

Mapping to Graph-Vertex Coloring

- activities ➜ vertices
- if 2 activities require the same operators ➜ edges
- time slots ➜ colors
- feasibility problem (if # time slots is fixed)
- optimization problem

DSATUR heuristic for Graph-Vertex Coloring

saturation degree: number of differently colored adjacent vertices

set of empty color classes $\{C_1, \ldots, C_k\}$, where $k = |V|$

Sort vertices in decreasing order of their degrees

Step 1 A vertex of maximal degree is inserted into C_1.

Step 2 The vertex with the maximal saturation degree is chosen and inserted according to the greedy heuristic (first feasible color). Ties are broken preferring vertices with the maximal number of adjacent, still uncolored vertices; if further ties remain, they are broken randomly.

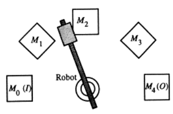
Outline

Resume: Job Shop

- Disjunctive graph representation [Roy and Sussman, 1964]
- Shifting Bottleneck Heuristic [Adams, Balas and Zawack, 1988]
- Local Search
- Generalizations:
  - Time lags d_{ij} to model:
    - set up times
    - synchronizations
    - deadlines
    - perishability (no-wait)
  - Blocking (alternative graph) ➜ Rollout

Exercise 1

Robotic Cell



Search for periodic pattern of moves (cycle)
one-unit cycle: the robot load (or unload) each machine exactly once
k-unit cycle: each activity is carried out exactly k times

Given:

- m machines $M_1, M_2, \ldots M_m$
- $c_{i,i+1}$ times of part transfer (unload+travel+load=activity) from M_i to M_{i+1}
- $d_{i,j}$ times of the empty robot from M_i to M_j ($c_{i,i+1} \geq d_{i,i+1}$)
- p_{ij} processing time of part j on machine i (identical vs different parts)

Task:

- Determine input time for each part t_j
- Minimize throughput $\rightsquigarrow$ minimize period

Alternative graph model with intermediate robot operations

Part XIV

Educational Timetabling

Outline

Outline

The Timetabling Activity

Assignment of events to a limited number of time periods and locations subject to constraints

Two categories of constraints:

Hard constraints $H = \{H_1, \ldots, H_n\}$: must be strictly satisfied, no violation is allowed

Soft constraints $\Sigma = \{S_1, \ldots, S_m\}$: their violation should be minimized (determine quality)

Each institution may have some unique combination of hard constraints and take different views on what constitute the quality of a timetable.

Types of Timetabling

- Educational Timetabling
  - Class timetabling
  - Exam timetabling
  - Course timetabling
- Employee Timetabling
  - Crew scheduling
  - Crew rostering
- Transport Timetabling,
- Sports Timetabling,
- Communication Timetabling

Educational timetabling process

Phase:	Planning	Scheduling	Dispatching
Horizon:	Long Term	Timetable Period	Day of Operation
Objective:	Service Level	Feasibility	Get it Done
Steps:	Curricula Manpower, Equipment	Weekly Timetabling	Repair, find rooms

We will concentrate on simple models that admit IP formulations or graph and network algorithms. These simple problems might:

- occur at various stages
- be instructive to derive heuristics for more complex cases

Outline

School Timetabling

[aka, teacher-class model]

The daily or weekly scheduling for all the classes of a high school, avoiding teachers meeting two classes in the same time, and vice versa.

Input:

- a set of classes $\mathcal{C} = \{C_1, \ldots, C_m\}$
  A class is a set of students who follow exactly the same program. Each class has a dedicated room.
- a set of teachers $\mathcal{P} = \{P_1, \ldots, P_n\}$
- a requirement matrix $\mathcal{R}_{m \times n}$ where R_{ij} is the number of lectures given by teacher R_j to class C_i.
- all lectures have the same duration (say one period)
- a set of time slots $\mathcal{T} = \{T_1, \ldots, T_p\}$ (the available periods in a day).

Output: An assignment of lectures to time slots such that no teacher or class is involved in more than one lecture at a time

IP formulation:

Binary variables: assignment of teacher P_j to class C_i in T_k

$$x_{ijk} = \{0, 1\} \quad \forall i = 1, \ldots, m;\ j = 1, \ldots, n;\ k = 1, \ldots, p$$

Constraints:

$$\sum_{k=1}^{p} x_{ijk} = R_{ij} \quad \forall i = 1, \ldots, m;\ j = 1, \ldots, n$$
$$\sum_{j=1}^{n} x_{ijk} \leq 1 \quad \forall i = 1, \ldots, m;\ k = 1, \ldots, p$$
$$\sum_{i=1}^{m} x_{ijk} \leq 1 \quad \forall j = 1, \ldots, n;\ k = 1, \ldots, p$$

Graph model

Bipartite multigraph $G = (\mathcal{C}, \mathcal{T}, \mathcal{R})$:

- nodes $\mathcal{C}$ and $\mathcal{T}$: classes and teachers
- R_{ij} parallel edges

Time slots are colors ➜ Graph-Edge Coloring problem

Theorem: [König] There exists a solution to (1) iff:

$$\sum_{i=1}^{m} R_{ij} \leq p \quad \forall j = 1, \ldots, n$$
$$\sum_{i=1}^{n} R_{ij} \leq p \quad \forall i = 1, \ldots, m$$

Extension

From daily to weekly schedule
(timeslots represent days)

- a_i max number of lectures for a class in a day
- b_j max number of lectures for a teacher in a day

IP formulation:

Variables: number of lectures to a class in a day

$$\sum_{i=1}^{m} x_{ijk} \leq b_j \quad \forall j = 1, \ldots, n;\ k = 1, \ldots, p$$
$$x_{ijk} \in N \quad \forall i = 1, \ldots, m;\ j = 1, \ldots, n;\ k = 1, \ldots, p$$

Constraints:

$$\sum_{k=1}^{p} x_{ijk} = R_{ij} \quad \forall i = 1, \ldots, m;\ j = 1, \ldots, n$$
$$\sum_{j=1}^{n} x_{ijk} \leq a_i \quad \forall i = 1, \ldots, m;\ k = 1, \ldots, p$$

Graph model

Edge coloring model still valid but with

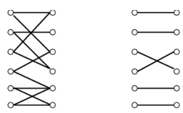
- no more than a_i edges adjacent to C_i have same colors and
- and more than b_j edges adjacent to T_j have same colors

Theorem: [König] There exists a solution to (2) iff:

$$\sum_{i=1}^{m} R_{ij} \leq b_j p \quad \forall j = 1, \ldots, n$$
$$\sum_{i=1}^{n} R_{ij} \leq a_i p \quad \forall i = 1, \ldots, m$$

A recurrent sub-problem in Timetabling is Matching

Input: A (weighted) bipartite graph $G = (V, E)$ with bipartition $\{A, B\}$.
Task: Find the largest size set of edges $M \in E$ such that each vertex in V is incident to at most one edge of M.



Efficient algorithms for constructing matchings are based on augmenting paths in graphs. An implementation is available at:
`http://www.cs.sunysb.edu/~algorith/implement/bipm/implement.shtml`

Theorem [Hall, 1935]: G contains a matching of A if and only if $|N(U)| \geq |U|$ for all $U \subseteq A$.

- The edge coloring problem in the multigraph is solvable in polynomial time by solving a sequence of network flows problems p.
  Possible approach: solve the weekly timetable first and then the daily timetable

Further constraints that may arise:

- Preassignments
- Unavailabilities
  (can be expressed as preassignments with dummy class or teachers)

They make the problem NP-complete.

- Bipartite matchings can still help in developing heuristics, for example, for solving x_{ijk} keeping any index fixed.

Further complications:

- Simultaneous lectures (eg, gymnastic)
- Subject issues (more teachers for a subject and more subject for a teacher)
- Room issues (use of special rooms)

So far feasibility problem.

Preferences (soft constraints) may be introduced

- Desirability of assignment p_j to class c_i in t_k

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{p} d_{ijk} x_{ijk}$$

- Organizational costs: having a teacher available for possible temporary teaching posts
- Specific day off for a teacher

Introducing soft constraints the problem becomes a multiobjective problem.

Possible ways of dealing with multiple objectives:

- weighted sum
- lexicographic order
- minimize maximal cost
- distance from optimal or nadir point
- Pareto-frontier
- ...

Heuristic Methods

Construction heuristic

Based on principles:

- most-constrained lecture on first (earliest) feasible timeslot
- most-constrained lecture on least constraining timeslot

Enhancements:

- limited backtracking
- local search optimization step after each assignment

More later

Local Search Methods and Metaheuristics

High level strategy:

- Single stage (hard and soft constraints minimized simultaneously)
- Two stages (feasibility first and quality second)

Dealing with feasibility issue:

- partial assignment: do not permit violations of H but allow some lectures to remain unscheduled
- complete assignment: schedule all the lectures and seek to minimize H violations

More later

University Course Timetabling

The weekly scheduling of the lectures of courses avoiding students, teachers and room conflicts.

Input:

- A set of courses $\mathcal{C} = \{C_1, \ldots, C_n\}$ each consisting of a set of lectures $C_i = \{L_{i1}, \ldots, L_{il_i}\}$. Alternatively,
  A set of lectures $\mathcal{L} = \{L_1, \ldots, L_l\}$).
- A set of curricula $\mathcal{S} = \{S_1, \ldots, S_r\}$ that are groups of courses with common students (curriculum based model). Alternatively,
  A set of enrollments $\mathcal{S} = \{S_1, \ldots, S_s\}$ that are groups of courses that a student wants to attend (Post enrollment model).
- a set of time slots $\mathcal{T} = \{T_1, \ldots, T_p\}$ (the available periods in the scheduling horizon, one week).
- All lectures have the same duration (say one period)

Output:

An assignment of each lecture L_i to some period in such a way that no student is required to take more than one lecture at a time.

IP formulation

m_t rooms ⇒ maximum number of lectures in time slot t

Variables

$$x_{it} \in \{0, 1\} \quad i = 1, \ldots, n;\ t = 1, \ldots, p$$

Number of lectures per course

$$\sum_{t=1}^{p} x_{it} = l_i \quad \forall i = 1, \ldots, n$$

Number of lectures per time slot

$$\sum_{i=1}^{n} x_{it} \leq m_t \quad \forall t = 1, \ldots, p$$

Number of lectures per time slot (students' perspective)

$$\sum_{C_i \in S_j}^{n} x_{it} \leq 1 \quad \forall i = 1, \ldots, n;\ t = 1, \ldots, p$$

If some preferences are added:

$$\max \quad \textstyle\sum_{i=1}^{p} \sum_{i=1}^{n} d_{it} x_{it}$$

Corresponds to a bounded coloring.
It can be solved up for 70 lectures, 25 courses and 40 curricula. [de Werra, 1985]

Graph model

Graph $G = (V, E)$:

- V correspond to lectures L_i
- E correspond to conflicts between lectures due to curricula or enrollments

Time slots are colors ➜ Graph-Vertex Coloring problem ➜ NP-complete
(exact solvers max 100 vertices)

Typical further constraints:

- Unavailabilities
- Preassignments

The overall problem can still be modeled as Graph-Vertex Coloring. How?

Further complications:

- Teachers that teach more than one course (treated similarly to students' enrollment)
- A set of rooms $\mathcal{R} = \{R_1, \ldots, R_n\}$ with suitability and availability constraints (this can be modeled as Hypergraph Coloring!)

Moreover,

- Logistic constraints: not two adjacent lectures if at different campus
- Max number of lectures in a single day and changes of campuses.
- Periods of variable length

IP formulation to include room eligibility [Lach and Lübbecke, 2008]

Decomposition of the problem in two stages:

1. assign courses to timeslots
2. match courses with rooms within each timeslot

In stage 1
Let $R(C_i) \subseteq \mathcal{R}$ be the rooms eligible for course C_i
Let $G_{conf} = (V_{conf}, E_{conf})$ be the conflict graph (vertices are pairs (C_i, T_t))

Variables: course C_i assigned to time slot T_t

$$x_{it} \in \{0,1\} \qquad i = 1, \ldots, n;\ t = 1, \ldots, p$$

Edge constraints
(forbids that C_{i_1} is assigned to T_{t_1} and C_{i_2} to T_{t_2} simultaneously)

$$x_{i_1,t_1} + x_{j_2,t_2} \leq 1 \qquad \forall\, ((i_1,t_1),(i_2,t_2)) \in E_{conf}$$

Hall's constraints
(guarantee that in stage 1 we find only solutions that are feasible for stage 2)

$$\sum_{C_i \in U}^{n} x_{it} \leq |N(U)| \qquad \forall\, U \in, t \in \mathcal{T}$$

If some preferences are added:

$$\max \sum_{i=1}^{p} \sum_{i=1}^{n} d_{it} x_{it}$$

So far feasibility.

Preferences (soft constraints) may be introduced

- Compactness or distribution
- Minimum working days
- Room stability
- Student min max load per day
- Travel distance
- Room suitability
- Double lectures
- Professors' preferences for time slots

For most of these different way to model them exist.

Exam Timetabling

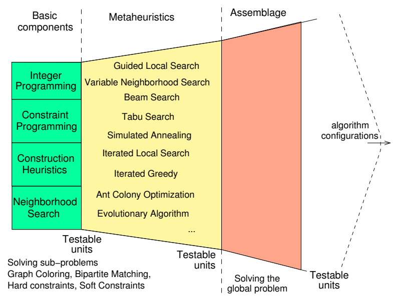
By substituting lecture with exam we have the same problem!
However:

Course Timetabling	Exam Timetabling
limited number of time slots	unlimited number of time slots, seek to minimize
conflicts in single slots, seek to compact	conflicts may involve entire days and consecutive days,seek to spread
one single course per room	possibility to set more than one exam in a room with capacity constraints
lectures have fixed duration	exams have different duration

Solution Methods

Hybrid Heuristic Methods

- Some metaheuristic solve the general problem while others or exact algorithms solve the special problem
- Replace a component of a metaheuristic with one of another or of an exact method (ILS+ SA, VLSN)
- Treat algorithmic procedures (heuristics and exact) as black boxes and serialize
- Let metaheuristics cooperate (evolutionary + tabu search)
- Use different metaheuristics to solve the same solution space or a partitioned solution space



Configuration Problem

Algorithms must be configured and tuned and the best selected.

This has to be done anew every time because constraints and their density are specific of the institution.

Appropriate techniques exist to aid in the experimental assessment of algorithms.

Outline

A Solution Example on Course Timetabling

Course Timetabling Problem

Find an assignment of lectures to time slots and rooms which is

Feasible

rooms are only used by one lecture at a time,
each lecture is assigned to a suitable room,
no student has to attend more than one lecture at once,
lectures are assigned only time slots where they are available;
} Hard Constraints

and Good

no more than two lectures in a row for a student,
unpopular time slots avoided (last in a day),
students do not have one single lecture in a day.
} Soft Constraints

A look at the instances

instance	events	students	rooms	events/students	students/event	rooms/event	degree	av_slot_event
comp-2007-2-10.tim	400	500	10	20.984	26.23	3.2025	0.3843	25.4675
comp-2007-2-2.tim	400	500	10	21.03	26.2875	3.9475	0.3744	25.69
comp-2007-2-1.tim	400	500	10	21.02	26.275	4.0775	0.3418	25.3425
comp-2007-2-9.tim	400	500	10	21.428	26.785	2.9125	0.3409	25.4175
comp-2007-2-13.tim	400	300	20	21.1933	15.895	8.6825	0.3236	25.7525
comp-2007-2-14.tim	400	300	20	20.8567	15.6425	7.5625	0.3209	25.44
comp-2007-2-5.tim	400	300	20	20.9167	15.6875	6.805	0.3081	25.425
comp-2007-2-6.tim	400	300	20	20.7267	15.545	5.065	0.3029	25.3925
comp-2007-2-12.tim	200	1000	10	13.607	68.035	3.355	0.5842	25.67
comp-2007-2-15.tim	200	500	10	13.054	32.635	2.23	0.5365	17.375
comp-2007-2-7.tim	200	500	20	13.466	33.665	1.575	0.5336	17.86
comp-2007-2-4.tim	200	1000	20	13.396	66.98	6.4	0.5199	25.665
comp-2007-2-8.tim	200	500	20	13.832	34.58	1.915	0.5175	17.17
comp-2007-2-11.tim	200	1000	10	13.608	68.04	3.375	0.5006	25.32
comp-2007-2-3.tim	200	1000	20	13.383	66.915	5.045	0.4748	25.535
comp-2007-2-16.tim	200	500	10	13.638	34.095	1.74	0.4558	17.57

These are large scale instances.

A look at the evaluation of a timetable can help in understanding the solution strategy

High level solution strategy:

- Single phase strategy (not well suited here due to soft constraints)
- ➜ Two phase strategy: Feasibility first, quality second

Searching a feasible solution:

- Suitability of rooms complicate the use of IP and CP.
- Heuristics:
  1. Complete assignment of lectures
  2. Partial assignment of lectures
- Room assignment:
  A. Left to matching algorithm
  B. Carried out heuristically

Solution Representation

A. Room assignment left to matching algorithm:

Array of Lectures and Time-slots and/or
Collection of sets Lectures, one for each Time-slot

B. Room assignment included

Assignment Matrix

$$\text{Rooms}\quad \begin{array}{c|cccccccccc} & \multicolumn{10}{c}{\text{Time-slots}} \\ & T_1 & T_2 & & T_i & & T_j & & T_{45} \\ \hline R_1 & -1 & L_4 & \cdots & \mathbf{L_{10}} & \cdots & L_{14} & \cdots & -1 \\ R_2 & L_1 & L_5 & \cdots & L_{11} & \cdots & \mathbf{L_{15}} & \cdots & -1 \\ R_3 & L_2 & L_6 & \cdots & \mathbf{L_{12}} & \cdots & -1 & \cdots & -1 \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & \\ R_r & L_3 & L_7 & \cdots & L_{13} & & L_{16} & \cdots & -1 \end{array}$$

Construction Heuristic

most-constrained lecture on least constraining time slot

Step 1. Initialize the set $\hat{L}$ of all unscheduled lectures with $\hat{L} = L$.
Step 2. Choose a lecture $L_i \in \hat{L}$ according to a *heuristic rule.*
Step 3. Let $\hat{X}$ be the set of all positions for L_i in the assignment matrix with minimal violations of the hard constraints H.
Step 4. Let $\bar{X} \subseteq \hat{X}$ be the subset of positions of $\hat{X}$ with minimal violations of the soft constraints Σ.
Step 5. Choose an assignment for L_i in $\bar{X}$ according to a *heuristic rule.* Update information.
Step 6. Remove L_i from $\hat{L}$, and go to step 2 until $\hat{L}$ is not empty.

Local Search Algorithms

Neighborhood Operators:

A. Room assignment left to matching algorithm

The problem becomes a bounded graph coloring
➜ Apply well known algorithms for GCP with few adaptations

Ex:

1. complete assignment representation: TabuCol with one exchange
2. partial assignment representation: PartialCol with i-swaps

See [Blöchliger and N. Zufferey, 2008] for a description

B. Room assignment included

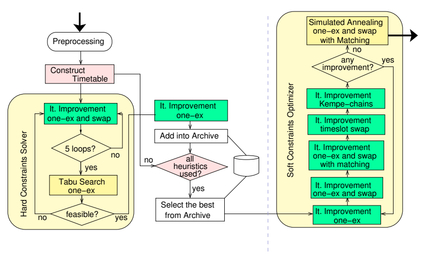
	Monday									Tuesday									Wednesday								
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23	T24	T25	T26	T27
R1	187	239	378	66	380	53	208	279		300	350	211	375	254	366	369	223	163	298		118	368	234	97	329	274	58
R2	360	345	2	153		354	91	61	319	349	278	86	204	316	220	323	176		314	7	108		50	312	235	330	
R3	263	71	186	67	222	288	99	24		237		232	253	117		195	203	102	207	287	290	146	286	358	303	277	
R4	181	160		90	82			193		206	156	152		341	179	171	226		4	348	127			365	213	80	
R5	324	291	309	339	267	283				269	170	299	311	34		65	216		275	199	26		27	327	33	39	285
R6	322	225	352	28	168	72	49	69	12	92	38	373	390	164	135	121	268	115	75	87	140	165	104	137	133	385	346
R7	228	31	107	371	30	355	46	227	246	271	182	313	224	128		89	258	356	343	280	35	109	306	43	83	11	154
R8	256	32	147	270	289	130	48	282		0	116	251	307	44	260	79	296		242	150	81	353	158	293	338	218	161
R9	396	144	173	78	25	183	387	337	240	132	328	212	370	308	336	244	126	14	231	51	342	136	93	129	266	393	155
R10	382	1	56	362	45	247	392	85	389	384	17	394	200		294	273	391	180	42	157	388	397	331	131	363	383	

- N_1: One Exchange
- N_2: Swap
- N_3: Period Swap
- N_4: Kempe Chain Interchange

Example of stochastic local search for case 1. with A.

```
initialize data (fast updates, dont look bit, etc.)
while (hcv!=0 && stillTime && idle iterations < PARAMETER)
  shuffle the time slots
  for each lecture L causing a conflict
    for each time slot T
      if not dont look bit
        if lecture is available in T
          if lectures in T < number of rooms
            try to insert L in T
            compute delta
            if delta < 0 || with a PARAMETER probability if delta==0
              if there exists a feasible matching room-lectures
                implement change
                update data
                if (delta==0) idle_iterations++ else idle_iterations=0;
                break
          for all lectures in time slot
            try to swap time slots
            compute delta
            if delta < 0 || with a PARAMETER probability if delta==0
              implement change
              update data
              if (delta==0) idle_iterations++ else idle_iterations=0;
              break
```

Algorithm Flowchart



Outline

In Practice

A timetabling system consists of:

- Information Management
- Solver (written in a fast language, *i.e.*, C, C++)
- Input and Output management (various interfaces to handle input and output)
- Interactivity: Declaration of constraints (professors' preferences may be inserted directly through a web interface and stored in the information system of the University)

See examples `http://www.easystaff.it`
`http://www.eventmap-uk.com`

The timetabling process

1. Collect data from the information system
2. Execute a few runs of the Solver starting from different solutions selecting the timetable of minimal cost. The whole computation time should not be longer than say one night. This becomes a "draft" timetable.
3. The draft is shown to the professors who can require adjustments. The adjustments are obtained by defining new constraints to pass to the Solver.
4. Post-optimization of the "draft" timetable using the new constraints
5. The timetable can be further modified manually by using the Solver to validate the new timetables.

Current Research Directions

1. Attempt to formulate standard timetabling problems with super sets of constraints where portable programs can be developed and compared
2. Development of general frameworks that leave the user the final instantiation of the program
3. Methodology for choosing automatically and intelligently the appropriate algorithm for the problem at hand (*hyper-heuristics case-based reasoning systems* and *racing for algorithm configuration*).
4. Robust timetabling

For latest developments see results of International Timetabling Competition 2007: `http://www.cs.qub.ac.uk/itc2007/`

Part XV

Sport Timetabling

Outline

Problems we treat:

- single and double round-robin tournaments
- balanced tournaments
- bipartite tournaments

Solutions:

- general results
- graph algorithms
- integer programming
- constraint programming
- metaheuristics

Outline

Terminology:

- A schedule is a mapping of games to slots or time periods, such that each team plays at most once in each slot.
- A schedule is compact if it has the minimum number of slots.
- Mirrored schedule: games in the first half of the schedule are repeated in the same order in the second half (with venues reversed)
- Partially mirrored schedule: all slots in the schedule are paired such that one is the mirror of the other
- A pattern is a vector of home (H) away (A) or bye (B) for a single team over the slots
- Two patterns are complementary if in every slot one pattern has a home and the other has an away.
- A pattern set is a collection of patterns, one for each team
- A tour is the schedule for a single team, a trip a series of consecutive away games and a home stand a series of consecutive home games

Round Robin Tournaments

(round-robin principle known from other fields, where each person takes an equal share of something in turn)

- Single round robin tournament (SRRT) each team meets each other team once
- Double round robin tournament (DRRT) each meets each other team twice

Definition SRRT Problem

Input: A set of n teams $T = \{1, \dots, n\}$

Output: A mapping of the games in the set $G = \{g_{ij} : i, j \in T, i < j\}$, to the slots in the set $S = \{s_k, k = 1, \dots, n-1$ if n is even and $k = 1, \dots, n$ if n is odd$\}$ such that no more than one game including i is mapped to any given slot for all $i \in T$.

Circle method

Label teams and play:

```
Round 1. (1 plays 14, 2 plays 13, ... )

 1  2  3  4  5  6  7
 14 13 12 11 10 9  8
```

Fix one team (number one in this example) and rotate the others clockwise:

```
Round 2. (1 plays 13, 14 plays 12, ... )

 1  14 2  3  4  5  6
 13 12 11 10 9  8  7

Round 3. (1 plays 12, 13 plays 11, ... )

 1  13 14 2  3  4  5
 12 11 10 9  8  7  6
```

Repeat until almost back at the initial position

```
Round 13. (1 plays 2, 3 plays 14, ... )

 1  3  4  5  6  7  8
 2  14  13 12 11 10 9
```


Definition DRRT Problem

Input: A set of n teams $T = \{1, \dots, n\}$.

Output: A mapping of the games in the set $G = \{g_{ij} : i, j \in T, i \neq j\}$, to the slots in the set $S = \{s_k, k = 1, \dots, 2(n-1)$ if n is even and $k = 1, \dots, 2n$ if n is odd$\}$ such that no more than one game including i is mapped to any given slot for all $i \in T$.

The schedule can be obtained by the circle method plus mirroring

Venue assignment can also be done through the circle method

Latin square

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 5 & 1 & 4 \\ 3 & 5 & 4 & 2 & 1 \\ 4 & 1 & 2 & 5 & 3 \\ 5 & 4 & 1 & 3 & 2 \end{bmatrix}$$

Even, symmetric Latin square $\Leftrightarrow$ SRRT

Example: 4 Teams

```
round 1: 1 plays 2, 3 plays 4
round 2: 2 plays 3, 1 plays 4
round 3: 3 plays 1, 2 plays 4
```

Rewrite the schedule as a multiplication table: "a plays b in round c".

```
    1  2  3  4
   -----------
1 |    1  3  2
2 | 1     2  3
3 | 3  2     1
4 | 2  3  1
```

If the blank entries were filled with the symbol 4, then we have an even, symmetric latin square.

Round robin tournaments with preassignments correspond to complete partial latin squares ➜ NP-complete

Extension:

- determining the venue for each game
- assigning actual teams to slots (so far where just place holders)

Decomposition:

1. First generate a pattern set
2. Then find a compatible pairing team-games (this yeilds a timetable)
3. Then assign actual teams in the timetable

Generation of feasible pattern sets

- In SRRT:
  - every pair of patterns must differ in at least one slot. $\Rightarrow$ no two patterns are equal in the pattern set
  - if at most one break per team, then a feasible pattern must have the complementary property ($m/2$ complementary pairs)
- In DRRT,
  - for every pair of patterns i, j such that $1 \leq i < j \leq n$ there must be at least one slot in which i is home and j is away and at least one slot in which j is at home and i is away.
  - every slot in the pattern set includes an equal number of home and away games.

Definition Balanced Tournament Designs (BTDP)

Input: A set of n teams $T = \{1, \dots, n\}$ and a set of facilities F.

Output: A mapping of the games in the set $G = \{g_{ij} : i, j \in T, i < j\}$, to the slots available at each facility described by the set
$S = \{s_{fk}, f = 1, \dots, |F|, k = 1, \dots, n-1$ if n is even and $k = 1, \dots, n$ if n is odd$\}$ such that no more than one game involving team i is assigned to a particular slot and the difference between the number of appearances of team i at two separate facilities is no more than 1.

- BTDP($2m$,m): $2m$ teams and m facilities. There exists a solution for every $m \neq 2$.
- BTDP($2m+1$, m): extension of the circle method:
  - Step 1: arrange the teams $1, \dots, 2m+1$ in an elongated pentagon. Indicate a facility associated with each row containing two teams.
  - Step 2: For each slot $k = 1, \dots, 2m+1$, give the team at the top of the pentagon the bye. For each row with two teams i, j associated with facility f assign g_{ij} to s_{kf}. Then shift the teams around the pentagon one position in a clockwise direction.

Bipartite Tournament

Input: Two teams with n players $T_1 = \{x_1, \dots, x_2\}$ and $T_2 = \{y_1, \dots, y_n\}$.

Output: A mapping of the games in the set $G = \{g_{ij}\ i \in T_1, j \in T_2\}$, to the slots in the set $S = \{s_k,\ k = 1, \dots, n\}$ such that exactly one game including t is mapped to any given slot for all $t \in T_1 \cup T_2$

Latin square $\Leftrightarrow$ bipartite tournament ($l[i,j]$ if player x_i meets player y_j in l_{ij})

Extensions:

- n facilities and seek for a balanced BT in which each player plays exactly once in each facility
  $\iff$ two mutually orthogonal Latin squares
  (rows are slots and columns facilities)

  A pair of Latin squares $A = [a_{ij}]$ and $B = [b_{ij}]$ are orthogonal iff the the ordered pairs (a_{ij}, b_{ij}) are distinct for all i and j.

```
 1  2  3        1  2  3        1 1   2 2   3 3
 2  3  1        3  1  2        2 3   3 1   1 2
 3  1  2        2  3  1        3 2   1 3   2 1
                                   A and B
    A              B             superimposed
```

  Mutually orthogonal Latin squares do not exist if $m = 2, 6$.

- Chess tournaments (assigning white and black)
- avoid carry-over effects, no two players x_i and y_j may play the same sequence of opponents y_p and followed immediately by y_q. $\Rightarrow$ complete latin square.

Graph Algorithms

A spanning subgraph of $G = (V, E)$ with all vertices of degree k is called a k-factor (A subgraph $H \subseteq G$ is a 1-factor $\Leftrightarrow E(H)$ is a matching of V)

A 1-factorization of $K_n \equiv$ decomposition of K_n in perfect matchings $\equiv$ edge coloring of K_n

A SRRT among $2m$ teams is modeled by a complete graph K_{2m} with edge (i, j) representing the game between i and j and the schedule correspond to an edge coloring.

To include venues, the graph K_{2m} is oriented
(arc (ij) represents the game team i at team j)
and the edge coloring is said an oriented coloring.

A DRRT is modeled by the oriented graph G_{2m} with two arcs a_{ij} and a_{ji} for each ij and the schedule correspond to a decomposition of the arc set that is equivalent to the union of two oriented colorings of K_{2m}.

Assigning venues with minimal number of breaks:

- SRRT: there are at least $2m - 2$ breaks. Extension of circle method.
- DRRT: Any decomposition of G_{2m-2} has at least $6m - 6$ breaks.
- SRRT for n odd: the complete graph on an odd number of nodes k_{2m+1} has an oriented factorization with no breaks.

Three phase approach by IP

1. Find pattern sets (basic SRRT)

   Variable

   $$x_{ijk} \in \{0, 1\} \qquad \forall i, j = 1, \ldots, n;\ i < j,\ k = 1, \ldots, n - 1$$

   Every team plays exactly once in each slot

   $$\sum_{j:j>i} x_{ijk} = 1 \qquad \forall i = 1, \ldots, n;\ k = 1, \ldots, n - 1$$

   Each team plays every opponent exactly once.

   $$\sum_{k} x_{ijk} = 1 \qquad \forall i, j = 1, \ldots, n;\ i < j$$

Branch and cut algorithm
Adds odd-set constrains that strengthen the one-factor constraint, that is, exactly one game for each team in each slot

$$\sum_{i \in S, j \notin S} x_{ijk} \leq 1 \qquad \forall S \subseteq T, |S| \text{ is odd},\ k = 1, \ldots, n - 1$$

2. Find the timetable selecting the patterns and assining the games.

   Variable denoting that pattern i plays at j in slot k. It is defined only if the ith pattern has an A in its kth position, and the jth has an H in its kth position. (S pattern set; T round set; F set of feasible triples (ijk))

   $$x_{ijk} = \{0, 1\} \qquad \forall i, j \in S;\ k \in T, :\ (ijk) \in F$$

   i and j meet at most once:

   $$\sum_{t} x_{ijt} + \sum_{t} x_{jit} = 1 \qquad \forall i, j \in S, i \neq j$$

   j plays at most once in a round

   $$\sum_{j:(ijk) \in F} x_{ijk} + \sum_{j:(jik) \in F} x_{jik} \leq 1 \qquad \forall i \in S;\ k \in T$$

3. Assign teams to selected patterns (assignment problem)

CP formulation

- CP for phase 1 (games and patterns)

```
int n = ...;
range Teams [1..n];
range Slots [1..n-1];
var Teams opponent[Teams,Slots];

solve {

   forall (i in Teams, k in Slots) opponent[i,t]<>i;
   forall (i in Teams) alldifferent(all (k in Slots) opponent[i,k]);
   forall (k in Slots) onefactor(all (i in Teams) opponent[i,k]);

};
```

- CP for phase 2: assign actual teams to position in timetable

Constraints to be included in practice:

- Pattern set constraints
  - feasible pattern sequences: avoid three consecutive home or away games
  - equally distributed home and away games
- Team-specific constraints
  - fixed home and away patterns
  - fixed games and opponent constraints
  - stadium availability
  - forbidden patterns for sets of teams
  - constraints on the positioning of top games

Objective: maximize the number of good slots, that is, slots with popular match-ups later in the season or other TV broadcasting preferences.

Application Examples

- Dutch Professional Football League [Schreuder, 1992]
  1. SRRT canonical schedule with minimum breaks and mirroring to make a DRRT
  2. assign actual teams to the patterns
- European Soccer League [Bartsch, Drexl, Kroger (BDK), 2002]
  1. DRRT schedule made of two separate SRRT with complementary patterns (Germany)
     four SRRTs the (2nd,3rd) and (1st,4th) complementary (Austria)
  2. teams assigned to patterns with truncated branch and bound
  3. games in each round are assigned to days of the week by greedy and local search algorithms
- Italian Football League [Della Croce, Olivieri, 2006]
  1. Search for feasible pattern sets appealing to TV requirements
  2. Search for feasible calendars
  3. Matching teams to patterns

Reference

Kelly Easton and George Nemhauser and Michael Trick, Sport Scheduling, in Handbook of Scheduling: Algorithms, Models, and Performance Analysis, J.Y-T. Leung (Ed.), Computer & Information Science Series, Chapman & Hall/CRC, 2004.

Part XVI

Transportation Timetabling

Outline

Outline

Traveling tournament problem

Input: A set of teams $T = \{1, \ldots, n\}$; D an $n \times n$ integer distance matrix with elements d_{ij}; l, u integer parameters.

Output: A double round robin tournament on the teams in T such that

1. the length of every home stand and road trip is between l and u inclusive
2. the total distance traveled by the teams is minimized

A metaheuristic approach: Simulated Annealing

Constraints:

- DRRT constraints always satisfied (enforced)
- constraints on repeaters (i may not play at j and host j at home in consecutive slots) are relaxed in soft constraints

Objective made of:

- total distance
- a component to penalize violation of constraints on repeaters

➜ Penalties are dynamically adjusted to prevent the algorithm from spending too much time in a space where the soft constraints are not satisfied.

Neighborhood operators:

- Swap the positions of two slots of games
- Swap the schedules of two teams (except for the games when they play against)
- Swap venues for a particular pair of games (i at j in slot s and j at i in slot s' becomes i at j in slot s' and j at i in slot s)

Use reheating in SA.

Outline

Outline

Problems

- Tanker Scheduling
- Aircraft Routing and Scheduling
- Train Timetabling

MIP Models using complicated variables: Let a variable represent a road trip, a schedule section, or a whole schedule for a crew.

- Set packing
- Set partitioning

Solution techniques

- Branch and bound
- Local branching
- Branch and price (column generation)
- Subgradient optimization of Lagrangian multipliers (solution without Simplex)

Planning problems in public transport

Phase:	Planning	Scheduling	Dispatching
Horizon:	Long Term	Timetable Period	Day of Operation
Objective:	Service Level	Cost Reduction	Get it Done
Steps:	Network Design Line Planning Timetabling Fare Planning	Vehicle Scheduling Duty Scheduling Duty Rostering	Crew Assignment Delay Management Failure Management Depot Management

Master Schedule $\xrightarrow{\text{Dynamic Management}}$ Conflict resolution

[Borndörfer, Grötschel, Pfetsch, 2005, ZIB-Report 05-22]

Tanker Scheduling

Input:

- p ports
  limits on the physical characteristics of the ships
- n cargoes:
  type, quantity, load port, delivery port, time window constraints on the load and delivery times
- ships (tanker): s company-owned plus others chartered
  Each ship has a capacity, draught, speed, fuel consumption, starting location and times
  These determine the costs of a shipment: c_i^l (company-owned) c_j^* (chartered)

Output: A schedule for each ship, that is, an itinerary listing the ports visited and the time of entry in each port within the rolling horizon such that the total cost of transportation is minimized

Two phase approach:

1. determine for each ship i the set S_i of all possible itineraries
2. select the itineraries for the ships by solving an IP problem

Phase 1 can be solved by some ad-hoc enumeration or heuristic algorithm that checks the feasibility of the itinerary and its cost.

For each itinerary l of ship i compute the profit with respect to charter:

$$\pi_i^l = \sum_{j=1}^{n} a_{ij}^l c_j^* - c_i^l$$

where $a_{ij}^l = 1$ if cargo j is shipped by ship i in itinerary l and 0 otherwise.

Phase 2:

A set packing model with additional constraints

Variables

$$x_i^l \in \{0, 1\} \qquad \forall i = 1, \dots, s;\ l \in S_i$$

Each cargo is assigned to at most one ship:

$$\sum_{i=1}^{s} \sum_{l \in S_i} a_{ij}^l x_i^l \leq 1 \qquad \forall j = 1, \dots, n$$

Each tanker can be assigned at most one itinerary

$$\sum_{l \in S_i} x_i^l \leq 1 \qquad \forall i = 1, \dots, s$$

Objective: maximize profit

$$\max \sum_{i=1}^{s} \sum_{l \in S_i} \pi_i^l x_i^l$$

Branch and bound (Variable fixing)

Solve LP relaxation (this provides an upper bound) and branch by:

- select a fractional variable with value closest to 0.5
  (keep tree balanced)
  set a branch $x_i^l = 0$ and
  the other $x_i^l = 1$ (this rules out the other itineraries of ship i)
- select one ship and branch on its itineraries
  select the ship that may lead to largest profit or largest cargo or with largest number of fractional variables.

Local Branching

- The procedure is in the spirit of heuristic local search paradigm.
- The neighborhoods are obtained through the introduction in the MIP model of (invalid) linear inequalities called local branching cuts.
- Takes advantage of black box efficient MIP solvers.

In the previous branch and bound, unclear how to fix variables
➜ Idea: soft fixing

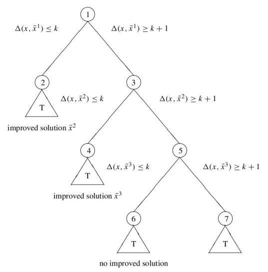
Given a feasible solution $\bar{x}$ let $\bar{O} := \{i \in B : \bar{x}_i = 1\}$.
Define the k-opt neighborhood $\mathcal{N}(\bar{x}, k)$ as the set of feasible solutions satisfying the additional local branching constraint:

$$\Delta(x, \bar{x}) := \sum_{i \in \bar{O}} (1 - x_i) + \sum_{i \in B \setminus \bar{O}} x_i \leq k$$

(Δ counts the number of flips)

Partition at the branching node:

$$\Delta(x, \bar{x}) \leq k \text{ (left branching)} \quad \textbf{or} \quad \Delta(x, \bar{x}) \geq k + 1 \text{ (right branching)}$$



- The idea is that the neighborhood $N(\bar{x}, k)$ corresponding to the left branch must be "sufficiently small" to be optimized within short computing time, but still "large enough" to likely contain better solutions than x.
- According to computational experience, good values for k are in $[10, 20]$

This procedure coupled with an efficient MIP solver (subgradient optimization of Lagrangian multipliers) was shown able to solve very large problems with more than 8000 variables.

OR in Air Transport Industry

- Aircraft and Crew Schedule Planning
  - Schedule Design (specifies legs and times)
  - Fleet Assignment
  - Aircraft Maintenance Routing
  - Crew Scheduling
    - crew pairing problem
    - crew assignment problem (bidlines)
- Airline Revenue Management
  - number of seats available at fare level
  - overbooking
  - fare class mix (nested booking limits)
- Aviation Infrastructure
  - airports
    - runaways scheduling (queue models, simulation; dispatching, optimization)
    - gate assignments
  - air traffic management

Daily Aircraft Routing and Scheduling (DARS)

Input:

- L set of flight legs with airport of origin and arrival, departure time windows $[e_i, l_i]$, $i \in L$, duration, cost/revenue
- Heterogeneous aircraft fleet T, with m_t aircrafts of type $t \in T$

Output: For each aircraft, a sequence of operational flight legs and departure times such that operational constraints are satisfied:

- number of planes for each type
- restrictions on certain aircraft types at certain times and certain airports
- required connections between flight legs (thrus)
- limits on daily traffic at certain airports
- balance of airplane types at each airport

and the total profits are maximized.

- L_t denotes the set of flights that can be flown by aircraft of type t
- S_t the set of feasible schedules for an aircraft of type t (inclusive of the empty set)
- $a_{ti}^l = \{0, 1\}$ indicates if leg i is covered by $l \in S_t$
- π_{ti} profit of covering leg i with aircraft of type i

$$\pi_t^l = \sum_{i \in L_t} \pi_{ti} a_{ti}^l \qquad \text{for } l \in S_t$$

- P set of airports, P_t set of airports that can accommodate type t
- o_{tp}^l and d_{tp}^l equal to 1 if schedule l, $l \in S_t$ starts and ends, resp., at airport p

A set partitioning model with additional constraints

Variables

$$x_t^l \in \{0, 1\} \qquad \forall t \in T;\ l \in S_t \qquad \text{and} \qquad x_t^0 \in \mathbf{N} \qquad \forall t \in T$$

Maximum number of aircraft of each type:

$$\sum_{l \in S_t} x_t^l = m_t \qquad \forall t \in T$$

Each flight leg is covered exactly once:

$$\sum_{t \in T} \sum_{l \in S_t} a_{ti}^l x_t^l = 1 \qquad \forall i \in L$$

Flow conservation at the beginning and end of day for each aircraft type

$$\sum_{l \in S_t} (o_{tp}^l - d_{tp}^l) x_t^l = 0 \qquad \forall t \in T;\ p \in P$$

Maximize total anticipate profit

$$\max \sum_{t \in T} \sum_{l \in S_t} \pi_t^l x_t^l$$

Solution Strategy: branch-and-price (branch-and-bound + column generation)

- At the high level branch-and-bound similar to the Tanker Scheduling case
- Upper bounds obtained solving linear relaxations by column generation.
  - Decomposition into
    - Restricted Master problem, defined over a restricted number of schedules
    - Subproblem, used to test the optimality or to find a new feasible schedule to add to the master problem (column generation)
  - Each restricted master problem solved by LP.
    It finds current optimal solution and dual variables
  - Subproblem (or pricing problem) corresponds to finding longest path with time windows in a network defined by using dual variables of the current optimal solution of the master problem. Solve by dynamic programming.

Train Timetabling

Input:

- Corridors made up of two independent one-way tracks
- L links between $L + 1$ stations.
- T set of trains and T_j, $T_j \subseteq T$, subset of trains that pass through link j

Output: We want to find a periodic (eg, one day) timetable for the trains on one track (the other can be mirrored) that specifies:

- y_{ij} = time train i enters link j
- z_{ij} = time train i exists link j

such that specific constraints are satisfied and costs minimized.

Constraints:

- Minimal time to traverse one link
- Minimum stopping times at stations to allow boarding
- Minimum headways between consecutive trains on each link for safety reasons
- Trains can overtake only at train stations
- There are some "*predetermined*" upper and lower bounds on arrival and departure times for certain trains at certain stations

Costs due to:

- deviations from some "*preferred*" arrival and departure times for certain trains at certain stations
- deviations of the travel time of train i on link j
- deviations of the dwelling time of train i at station j

Solution Approach

- All constraints and costs can be modeled in a MIP with the variables: y_{ij}, z_{ij} and $x_{ihj} = \{0, 1\}$ indicating if train i precedes train h
- Two dummy trains T' and T'' with fixed times are included to compact and make periodic
- Large model solved heuristically by decomposition.
- Key Idea: insert one train at a time and solve a simplified MIP.
- In the simplified MIP the order in each link of trains already scheduled is maintained fixed while times are recomputed. The only order not fixed is the one of the new train inserted k (x_{ihj} simplifies to x_{ij} which is 1 if k is inserted in j after train i)

Overall Algorithm

Step 1 (Initialization)
Introduce two "dummy trains" as the first and last trains in T_0

Step 2 (Select an Unscheduled Train) Problem) Select the next train k through the train selection priority rule

Step 3 (Set up and preprocess the MIP) Include train k in the set T_0
Set up MIP(K) for the selected train k
Preprocess MIP(K) to reduce number of 0–1 variables and constraints

Step 4 (Solve the MIP) Solve MIP(K). If algorithm does not yield feasible solution STOP.
Otherwise, ass train k to the list of already scheduled trains and fix for each link the sequences of all trains in T_0.

Step 5 (Reschedule all trains scheduled earlier) Consider the current partial schedule that includes train k.
For each train $i \in \{T_0 - k\}$ delete it and reschedule it

Step 6 (Stopping criterion) If T_0 consists of all train, then STOP otherwise go to Step 2.

Further References

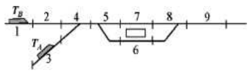
M. Fischetti and A. Lodi, Local Branching, Mathematical Programming, 98(1-3), pp 23-47, 2003.

C. Barnhart, P. Belobaba, A. Odoni, Applications of Operations Research in the Air Transport Industry, Transportation Science, 2003, vol. 37, issue 4, p 368.

Exercise

Short-term Railway Traffic Optimization

Conflict resolution problem (CRP) with two trains traveling at different speed:



Block sections: track segment with signals (fixed NS54)

At time $t = 0$ there are two trains in the network.

Train T_A is a slow train running from block section 3 to block section 9, and stopping at platform 6. It can enter a block section only if the signal aspect is yellow or green.

Train T_B is a fast train running from block section 1 to block section 9 through platform 7 without stopping. It can enter a block section at high speed only if the signal aspect is green.

A blocking job shop model:

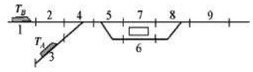
Given:

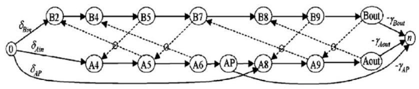
- Passing of trains in a block ➜ Operation
- Traverse (running) times ➜ Processing times
- Itinerary of the train ➜ Precedences
- Safety standards between blocks ➜ Setup times

Task:

- Find the starting times $t_1, t_2, \ldots, t_n$, (or the precedences) such that:
  - No conflict (two trains on the same track segment at the same time)
  - Minimize maximum delay (or disrupt least possible the original plan)

- Signals and train speed constraints can be modeled as blocking constraints ➜ Alternative graph
- Speed and times goals can be modeled with time lags





- δ_{AP} scheduled departing time from platform P
- $-\gamma_{AP}$ planned due dates

Part XVII

Workforce Scheduling

Outline

Outline

Workforce Scheduling

Workforce Scheduling:

- Crew Scheduling and Rostering
- Employee Timetabling

Shift: consecutive working hours
Roster: shift and rest day patterns over a fixed period of time (a week or a month)

Two main approaches:

- coordinate the design of the rosters and the assignment of the shifts to the employees, and solve it as a single problem.
- consider the scheduling of the actual employees only after the rosters are designed, solve two problems in series.

Features to consider: rest periods, days off, preferences, availabilities, skills.

Crew Scheduling and Rostering

Workforce scheduling applied in the transportation and logistics sector for enterprises such as airlines, railways, mass transit companies and bus companies (pilots, attendants, ground staff, guards, drivers, etc.)

Employee Timetabling

Employee timetabling (aka labor scheduling) is the operation of assigning employees to tasks in a set of shifts during a fixed period of time, typically a week.
Days off, shifts, tours (set of shifts)

Examples of employee timetabling problems include:

- assignment of nurses to shifts in a hospital,
- assignment of workers to cash registers in a large store
- assignment of phone operators to shifts and stations in a service-oriented call-center

Differences with Crew scheduling:

- no need to travel to perform tasks in locations
- start and finish time not predetermined

Outline

Crew Scheduling

Input:

- Flight leg (departure, arrival, duration)
- A set of feasible combinations of flights for a crew

Output: A subset of flights feasible for a crew

Set partitioning problem!

Often treated as set covering because:

- its linear programming relaxation is numerically more stable and thus easier to solve
- it is trivial to construct a feasible integer solution from a solution to the linear programming relaxation
- it makes possible to restrict to only rosters of maximal length

Extension: a set of crews

Subgradient Optimization Lagrange Multipliers

$$\begin{aligned}
\max\quad & c^T x \\
\text{st}\quad & Ax \le b \\
& Dx \le d \\
& x_j \in \mathbf{Z}^+, \quad j = 1, \ldots, n
\end{aligned}$$

Lagrange Relaxation, multipliers $\lambda \ge 0$

$$\begin{aligned}
\max\quad & z_{LR}(\lambda) = c^T x - \lambda(Dx - d) \\
\text{st}\quad & Ax \le b \\
& x_j \in \mathbf{Z}^+, \quad j = 1, \ldots, n
\end{aligned}$$

Lagrange Dual Problem

$$z_{LD} = \min_{\lambda \ge 0} z_{LR}(\lambda)$$

Lagrangian dual solved by Subgradient optimization

- Works well due to convexity
- Roots in nonlinear programming

Outline

Shift Scheduling

Creating daily shifts:

- cycle made of m time intervals not necessarily identical
- during each period, b_i personnel is required
- n different shift patterns (columns of matrix A)

$$\begin{aligned}
\min\quad & c^T x \\
\text{st}\quad & Ax \ge b \\
& x \ge 0 \text{ and integer}
\end{aligned}$$

(k, m)-cyclic Staffing Problem

Assign persons to an m-period cyclic schedule so that:

- requirements b_i are met
- each person works a shift of k consecutive periods and is free for the other $m - k$ periods. (periods 1 and m are consecutive)

and the cost of the assignment is minimized.

$$\begin{aligned}
\min\quad & cx \\
\text{st}\quad & \begin{bmatrix}
1 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1
\end{bmatrix} x \ge b \qquad \text{(P)} \\
& x \ge 0 \text{ and integer}
\end{aligned}$$

Recall: Totally Unimodular Matrices

Definition: A matrix A is totally unimodular (TU) if every square submatrix of A has determinant +1, -1 or 0.

Proposition 1: The linear program $\max\{cx : Ax \le b, x \in \mathbf{R}^m_+\}$ has an integral optimal solution for all integer vectors b for which it has a finite optimal value if and only if A is totally unimodular

Efficient algorithms to recognize if a matrix is totally unimodular are nontrivial.

Proposition 2: A matrix A is TU if

(i) $a_{ij} \in \{+1, -1, 0\}$ for all i, j

(ii) each column contains at most two nonzero coefficients ($\sum_{i=1}^m |a_{ij}| \le 2$)

(iii) there exists a partition (M_1, M_2) of the set M of rows such that each column j containing two nonzero coefficients satisfies $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$

Proposition 3: A matrix A is TU if

(i) $a_{ij} \in \{+1, -1, 0\}$ for all i, j

(ii) for any subset M of the rows, there exists a partition (M_1, M_2) of M such that each column j satisfies

$$\left| \sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} \right| \le 1$$

Definition: A $(0,1)$–matrix B has the consecutive 1's property if for any column j, $b_{ij} = b_{i'j} = 1$ with $i < i'$ implies $b_{lj} = 1$ for $i < l < i'$. That is, if there is a permutation of the rows such that the 1's in each column appear consecutively.

Whether a matrix has the consecutive 1's property can be determined in polynomial time [D. R. Fulkerson and O. A. Gross; Incidence matrices and interval graphs. 1965 Pacific J. Math. 15(3) 835-855.]

A matrix with consecutive 1's property satisfies Proposition 3 and is therefore TU.

What about this matrix?

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Definition A $(0,1)$-matrix B has the circular 1's property for rows (resp. for columns) if the columns of B can be permuted so that the 1's in each row are circular, that is, appear in a circularly consecutive fashion

The circular 1's property for columns does not imply circular 1's property for rows.

Whether a matrix has the circular 1's property for rows (resp. columns) can be determined in $O(m^2 n)$ time [A. Tucker, Matrix characterizations of circular-arc graphs. (1971) Pacific J. Math. 39(2) 535-545]

Integer programs where the constraint matrix A have the circular 1's property for **rows** can be solved efficiently as follows:

Step 1 Solve the linear relaxation of (P) to obtain $x'_1, \ldots, x'_n$. If $x'_1, \ldots, x'_n$ are integer, then it is optimal for (P) and STOP. Otherwise go to Step 2.

Step 2 Form two linear programs LP1 and LP2 from the relaxation of the original problem by adding respectively the constraints

$$x_1 + \ldots + x_n = \lfloor x'_1 + \ldots + x'_n \rfloor \qquad \text{(LP1)}$$

and

$$x_1 + \ldots + x_n = \lceil x'_1 + \ldots + x'_n \rceil \qquad \text{(LP2)}$$

The solutions to LP1 and LP2 can be taken to be integral and the best of the two solutions is an optimal solution to the staffing problem (P)

Cyclic Staffing with Overtime

- Hourly requirements b_i
- Basic work shift 8 hours
- Overtime of up to additional 8 hours possible

minimize cx

subject to

$$\begin{array}{c} 07\\08\\09\\10\\11\\12\\13\\14\\15\\16\\17\\18\\19\\20\\21\\22\\23\\24\\01\\02\\03\\04\\05\\06 \end{array}
\left[\begin{array}{ccccccccccccccccccccccccccc}
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1
\end{array}\right] x \geq b$$

Days-Off Scheduling

- Guarantee two days-off each week, including every other weekend.

IP with matrix A:

$$\begin{array}{c} \\ \\ \\ \text{first week} \\ \\ \\ \\ \\ \\ \\ \text{second week} \\ \\ \\ \\ \end{array}
\left[\begin{array}{cccccc|cccccc}
1&1&1&1&1&1&1&1&1&1&1&0\\
1&1&1&1&1&1&1&1&1&1&0&0\\
1&1&1&1&1&1&1&1&1&0&0&1\\
1&1&1&1&1&1&1&1&0&0&1&1\\
1&1&1&1&1&1&1&0&0&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&1&1&1&1&1\\
\hline
0&1&1&1&1&1&1&1&1&1&1&1\\
0&0&1&1&1&1&1&1&1&1&1&1\\
1&0&0&1&1&1&1&1&1&1&1&1\\
1&1&0&0&1&1&1&1&1&1&1&1\\
1&1&1&0&0&1&1&1&1&1&1&1\\
1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0&0&0
\end{array}\right]$$

Cyclic Staffing with Part-Time Workers

- Columns of A describe the work-shifts
- Part-time employees can be hired for each time period i at cost c'_i per worker

$$\begin{aligned} \min \quad & cx + c'x' \\ \text{st} \quad & Ax + Ix' \geq b \\ & x, x' \geq 0 \text{ and integer} \end{aligned}$$

Cyclic Staffing with Linear Penalties for Understaffing and Overstaffing

- demands are not rigid
- a cost c'_i for understaffing and a cost c''_i for overstaffing

$$\begin{aligned} \min \quad & cx + c'x' + c''(b - Ax - x') \\ \text{st} \quad & Ax + Ix' \geq b \\ & x, x' \geq 0 \text{ and integer} \end{aligned}$$

Once rosters (set of shifts) are designed, people can be assigned to them according to availabilities, preferences, skills.

Alternatively one can take care of these two phases at the same time:

Nurse Scheduling

- Hospital: head nurses on duty seven days a week 24 hours a day
- Three 8 hours shifts per day (1: daytime, 2: evening, 3: night)
- In a day each shift must be staffed by a different nurse
- The schedule must be the same every week
- Four nurses are available (A,B,C,D) and must work at least 5 days a week.
- No shift should be staffed by more than two different nurses during the week
- No employee is asked to work different shifts on two consecutive days
- An employee that works shifts 2 and 3 must do so at least two days in a row.

Mainly a feasibility problem

A CP approach

Two solution representations

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Shift 1	A	B	A	A	A	A	A
Shift 2	C	C	C	B	B	B	B
Shift 3	D	D	D	D	C	C	D

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Worker A	1	0	1	1	1	1	1
Worker B	0	1	0	2	2	2	2
Worker C	2	2	2	0	3	3	0
Worker D	3	3	3	3	0	0	3

Variables w_{sd} nurse assigned to shift s on day d and y_{id} the shift assigned for each day

$$w_{sd} \in \{A, B, C, D\} \qquad y_{id} \in \{0, 1, 2, 3\}$$

Three different nurses are scheduled each day

$$\texttt{alldiff}(w_{\cdot d}) \qquad \forall d$$

Every nurse is assigned to at least 5 days of work

$$\texttt{cardinality}(w_{\cdot\cdot} \mid (A, B, C, D), (5, 5, 5, 5), (6, 6, 6, 6))$$

At most two nurses work any given shift

$$\texttt{nvalues}(w_{s\cdot} \mid 1, 2) \qquad \forall s$$

All shifts assigned for each day

$$\texttt{alldiff}(y_{\cdot d}) \qquad \forall d$$

Maximal sequence of consecutive variables that take the same values

$$\texttt{stretch-cycle}(y_{i\cdot} | (2,3), (2,2), (6,6), P) \qquad \forall i, P = \{(s,0),(0,s) | s = 1,2,3$$

Channeling constraints between the two representations:
on any day, the nurse assigned to the shift to which nurse i is assigned must be nurse i

$$w_{y_{id}, d} = i \qquad \forall i, d$$

$$y_{w_{sd}, d} = s \qquad \forall s, d$$

Global Constraint Catalog
`http://www.emn.fr/x-info/sdemasse/gccat/`

Solved by

- Constraint Propagation (Edge filtering)
- Search: branch on domains (first fail)
- Symmetry breaking

Local search methods and metaheuristics are used if the problem has large scale. Procedures very similar to what we saw for timetabling.

Part XVIII

Vehicle Routing

Outline

Outline

Problem Definition

Vehicle Routing: distribution of goods between depots and customers.

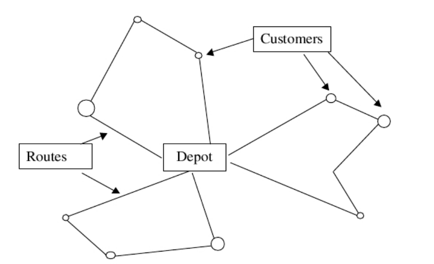
Delivery, collection, transportation.

Examples: solid waste collection, street cleaning, school bus routing, dial-a-ride systems, transportation of handicapped persons, routing of salespeople and maintenance unit.

Vehicle Routing Problems

Input: Vehicles, depots, road network, costs and customers requirements.
Output: Set of routes such that:

- requirement of customers are fulfilled,
- operational constraints are satisfied and
- a global transportation cost is minimized.



Refinement

Road Network

- represented by a (directed or undirected) complete graph
- travel costs and travel times on the arcs obtained by shortest paths

Customers

- vertices of the graph
- collection or delivery demands
- time windows for service
- service time
- subset of vehicles that can serve them
- priority (if not obligatory visit)

Vehicles

- capacity
- types of goods
- subsets of arcs traversable
- fix costs associated to the use of a vehicle
- distance dependent costs
- a-priori partition of customers
- home depot in multi-depot systems
- drivers with union contracts

Operational Constraints

- vehicle capacity
- delivery or collection
- time windows
- working periods of the vehicle drivers
- precedence constraints on the customers

Objectives

- minimization of global transportation cost (variable + fixed costs)
- minimization of the number of vehicles
- balancing of the routes
- minimization of penalties for un-served customers

History:
Dantzig, Ramser "The truck dispatching problem", Management Science, 1959
Clark, Wright, "Scheduling of vehicles from a central depot to a number of delivery points". Operation Research. 1964

Vehicle Routing Problems

- Capacited (and Distance Constrained) VRP (CVRP and DCVRP)
- VRP with Time Windows (VRPTW)
- VRP with Backhauls (VRPB)
- VRP with Pickup and Delivery (VRPPD)
- Periodic VRP (PVRP)
- Multiple Depot VRP (MDVRP)
- Split Delivery VRP (SDVRP)
- VRP with Satellite Facilities (VRPSF)
- Site Dependent VRP
- Open VRP
- Stochastic VRP (SVRP)
- ...

Capacited Vehicle Routing (CVRP)

Input: (common to all VRPs)

- (di)graph (strongly connected, typically complete) $G(V, A)$, where $V = \{0, \ldots, n\}$ is a vertex set:
  - 0 is the depot.
  - $V' = V \setminus \{0\}$ is the set of n customers
  - $A = \{(i, j) : i, j \in V\}$ is a set of arcs
- C a matrix of non-negative costs or distances c_{ij} between customers i and j (shortest path or Euclidean distance) $(c_{ik} + c_{kj} \geq c_{ij} \quad \forall\, i, j \in V)$
- a non-negative vector of costumer demands d_i
- a set of K (identical!) vehicles with capacity Q, $d_i \leq Q$

Task:
Find collection of K circuits with minimum cost, defined as the sum of the costs of the arcs of the circuits and such that:

- each circuit visits the depot vertex
- each customer vertex is visited by exactly one circuit; and
- the sum of the demands of the vertices visited by a circuit does not exceed the vehicle capacity Q.

Note: lower bound on K

- $\lceil d(V')/Q \rceil$
- number of bins in the associated *Bin Packing Problem*

A feasible solution is composed of:

- a partition $R_1, \ldots, R_m$ of V;
- a permutation π^i of $R_i \bigcup 0$ specifying the order of the customers on route i.

A route R_i is feasible if $\sum_{i=\pi_1}^{\pi_m} d_i \leq Q$.

The cost of a given route (R_i) is given by: $F(R_i) = \sum_{i=\pi_0^i}^{\pi_m^i} c_{i,i+1}$

The cost of the problem solution is: $F_{VRP} = \sum_{i=1}^{m} F(R_i)$.

Relation with TSP

- VRP with $K = 1$, no limits, no (any) depot, customers with no demand ➜ TSP
- VRP is a generalization of the Traveling Salesman Problem (TSP) ➜ is NP-Hard.
- VRP with a depot, K vehicles with no limits, customers with no demand ➜ Multiple TSP = one origin and K salesman
- Multiple TSP is transformable in a TSP by adding K identical copies of the origin and making costs between copies infinite.

Variants of CVRP:

- minimize number of vehicles
- different vehicles Q_k, $k = 1, \ldots, K$
- Distance-Constrained VRP: length t_{ij} on arcs and total duration of a route cannot exceed T associated with each vehicle
  Generally $c_{ij} = t_{ij}$
  (Service times s_i can be added to the travel times of the arcs: $t'_{ij} = t_{ij} + s_i/2 + s_j/2$)
- Distance constrained CVRP

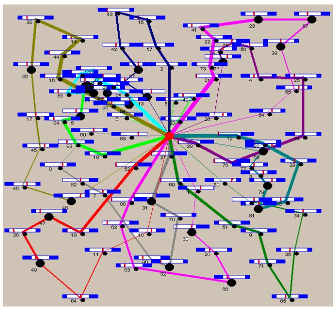
Vehicle Routing with Time Windows (VRPTW)

Further Input:

- each vertex is also associated with a time interval $[a_i, b_j]$.
- each arc is associated with a travel time t_{ij}
- each vertex is associated with a service time s_i

Task:
Find a collection of K simple circuits with minimum cost, such that:

- each circuit visit the depot vertex
- each customer vertex is visited by exactly one circuit; and
- the sum of the demands of the vertices visited by a circuit does not exceed the vehicle capacity Q.
- for each customer i, the service starts within the time windows $[a_i, b_i]$ (it is allowed to wait until a_i if early arrive)



Time windows induce an orientation of the routes.

Variants

- Minimize number of routes
- Minimize hierarchical objective function
- Makespan VRP with Time Windows (MPTW)
  minimizing the completion time
- Delivery Man Problem with Time Windows (DMPTW)
  minimizing the sum of customers waiting times

Solution Techniques for CVRP

- Integer Programming (only formulations)
- Construction Heuristics
- Local Search
- Metaheuristics
- Hybridization with Constraint Programming

Outline

Basic Models

- vehicle flow formulation
  integer variables on the edges counting the number of time it is traversed
  two or three index variables
- commodity flow formulation
  additional integer variables representing the flow of commodities along the paths traveled bu the vehicles
- set partitioning formulation

VRPTW

Pre-processing

- Time windows reduction
  - Increase earliest allowed departure time, a_k
  - Decrease latest allowed arrival time b_k
- Arc elimination
  - $a_i + t_{ij} > b_j$ ➜ arc (i, j) cannot exist
  - $d_i + d_j > C$ ➜ arcs (i, j) and (j, i) cannot exist

Outline

Construction Heuristics for CVRP

- TSP based heuristics
- Savings heuristics (Clarke and Wright)
- Insertion heuristics
- Cluster-first route-second
  - Sweep algorithm
  - Generalized assignment
  - Location based heuristic
  - Petal algorithm
- Route-first cluster-second

Cluster-first route-second seems to perform better
(Note: Distinction Construction Heuristic / Iterative Improvement is often blurred)

Construction heuristics for TSP

They can be used for route-first cluster-second or for growing multiple tours subject to capacity constraint.

- Heuristics that Grow Fragments
  - Nearest neighborhood heuristics
  - Double-Ended Nearest Neighbor heuristic
  - Multiple Fragment heuristic (aka, greedy heuristic)
- Heuristics that Grow Tours
  - Nearest Addition
  - Farthest Addition
  - Random Addition
  - Nearest Insertion
  - Farthest Insertion
  - Random Insertion
  - Clarke-Wright savings heuristic
- Heuristics based on Trees
  - Minimum span tree heuristic
  - Christofides' heuristics

(But recall! Concorde: http://www.tsp.gatech.edu/)

[Bentley, 1992]

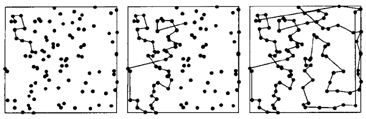


Figure 1. The Nearest Neighbor heuristic.

NN (Flood, 1956)

1. Randomly select a starting node
2. Add to the last node the closest node until no more node is available
3. Connect the last node with the first node

Running time $O(N^2)$

[Bentley, 1992]

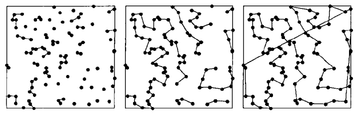


Figure 5. The Multiple Fragment heuristic.

Add the cheapest edge provided it does not create a cycle.

[Bentley, 1992]

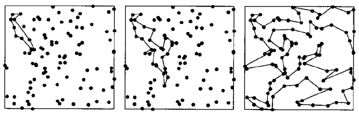


Figure 8. The Nearest Addition heuristic.

NA

1. Select a node and its closest node and build a tour of two nodes
2. Insert in the tour the closest node v until no more node is available

Running time $O(N^3)$

[Bentley, 1992]

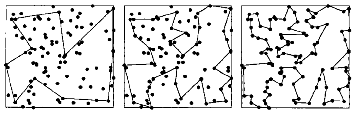


Figure 11. The Farthest Addition heuristic.

FA

1. Select a node and its farthest and build a tour of two nodes
2. Insert in the tour the farthest node v until no more node is available

FA is more efficient than NA because the first few farthest points sketch a broad outline of the tour that is refined after.

Running time $O(N^3)$

[Bentley, 1992]

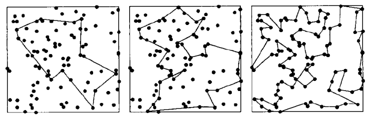


Figure 14. The Random Addition heuristic.

[Bentley, 1992]

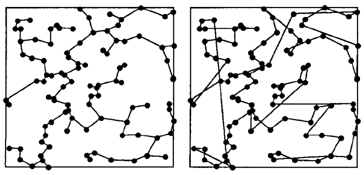


Figure 18. The Minimum Spanning Tree heuristic.

1. Find a minimum spanning tree $O(N^2)$
2. Append the nodes in the tour in a depth-first, inorder traversal

Running time $O(N^2)$ $\quad$ $A = MST(I)/OPT(I) \leq 2$

[Bentley, 1992]

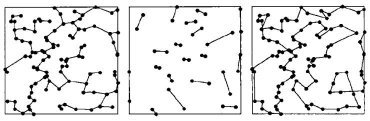
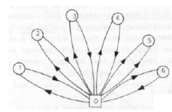


Figure 19. Christofides' heuristic.

1. Find the minimum spanning tree T. $O(N^2)$
2. Find nodes in T with odd degree and find the cheapest perfect matching M in the complete graph consisting of these nodes only. Let G be the multigraph all nodes and edges in T and M. $O(N^3)$
3. Find an Eulerian walk (each node appears at least once and each edge exactly once) on G and an embedded tour. $O(N)$

Running time $O(N^3)$ $\quad$ $A = CH(I)/OPT(I) \leq 3/2$

Construction Heuristics Specific for VRP



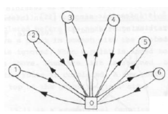
Clarke-Wright Saving Heuristic (1964)

1. Start with an initial allocation of one vehicle to each customer (0 is the depot for VRP or any chosen city for TSP)

Sequential:

2. consider in turn route $(0, i, \ldots, j, 0)$ determine s_{ki} and s_{jl}
3. merge with $(k, 0)$ or $(0, l)$

Construction Heuristics Specific for VRP

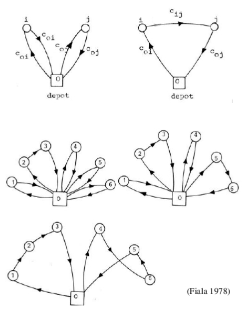


Clarke-Wright Saving Heuristic (1964)

1. Start with an initial allocation of one vehicle to each customer (0 is the depot for VRP or any chosen city for TSP)

Parallel:

2. Calculate saving $s_{ij} = c_{0i} + c_{0j} - c_{ij}$ and order the saving in non-increasing order
3. scan s_{ij}
   merge routes if i) i and j are not in the same tour ii) neither i and j are interior to an existing route iii) vehicle and time capacity are not exceeded



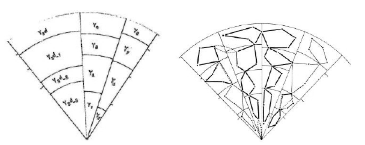
Matching Based Saving Heuristic

1. Start with an initial allocation of one vehicle to each customer (0 is the depot for VRP or any chosen city for TSP)
2. Compute $s_{pq} = t(S_p) + t(S_q) - t(S_p \cup S_q)$ where $t(\cdot)$ is the TSP solution
3. solve a max weighted matching on the S_k with weights s_{pq} on edges. A connection between a route p and q exists only if the merging is feasible.

Insertion Heuristic

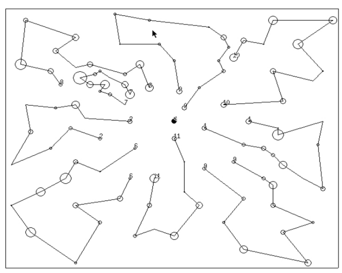
$$\alpha(i, k, j) = c_{ik} + c_{kj} - \lambda c_{ij}$$
$$\beta(i, k, j) = \mu c_{0k} - \alpha(i, k, j)$$

1. construct emerging route $(0, k, 0)$
2. compute for all k unrouted the feasible insertion cost:
   $$\alpha^*(i_k, k, j_k) = \min\{\alpha(i, k, j)\}$$
   if no feasible insertion go to 1 otherwise choose k^* such that
   $$\beta^*(i_k^*, k^*, j_k^*) = \max\{\beta(i_k, k, j_k)\}$$



Cluster-first route-second: Sweep algorithm [Wren & Holliday (1971)]

1. Choose i^* and set $\theta(i^*) = 0$ for the rotating ray
2. Compute and rank the polar coordinates (θ, ρ) of each point
3. Assign customers to vehicles until capacity not exceeded. If needed start a new route. Repeat until all customers scheduled.



Cluster-first route-second: Generalized-assignment-based algorithm [Fisher & Jaikumur (1981)]

1. Choose a j_k at random for each route k
2. For each point compute
   $$d_{ik} = \min\{c_{0,i} + c_{i,j_k} + c_{j_k,0}, c_{0j_k} + c_{j_k,i} + c_{i,0}\} - (c_{0,j_k} + c_{j_k,0})$$
3. Solve GAP with d_{ik}, Q and q_i

Cluster-first route-second: Location based heuristic [Bramel & Simchi-Levi (1995)]

1. Determine seeds by solving a capacited location problem (k-median)
2. Assign customers to closest seed

(better performance than insertion and saving heuristics)

Cluster-first route-second: Petal Algorithm

1. Construct a subset of feasible routes
2. Solve a set partitioning problem

Route-first cluster-second [Beasley]

1. Construct a TSP tour over all customers
2. Choose an arbitrary orientation of the TSP;
   partition the tour according to capacity constraint;
   repeat for several orientations and select the best
   Alternatively, solve a shortest path in an acyclic digraph with cots on arcs: $d_{ij} = c_{0i} + c_{0j} + l_{ij}$ where l_{ij} is the cost of traveling from i to j in the TSP tour.

(not very competitive)

Exercise

Which heuristics can be used to minimize K
and which one need to have K fixed a priori?

Part XIX

Vehicle Routing Heuristic Methods

Outline

Outline

Construction Heuristics for VRPTW

Extensions of those for CVRP [Solomon (1987)]

- Savings heuristics (Clarke and Wright)
- Time-oriented nearest neighbors
- Insertion heuristics
- Time-oriented sweep heuristic

Time-Oriented Nearest-Neighbor

- Add the unrouted node "closest" to the depot or the last node added without violating feasibility
- Metric for "closest":

$$c_{ij} = \delta_1 d_{ij} + \delta_2 T_{ij} + \delta_3 v_{ij}$$

d_{ij} geographical distance
T_{ij} time distance
v_{ij} urgency to serve j

Insertion Heuristics

Step 1: Compute for each unrouted costumer u the *best feasible position* in the route:

$$c_1(i(u), u, j(u)) = \min_{p=1,\ldots,m} \{c_1(i_{p-1}, u, i_p)\}$$

(c_1 is a composition of increased time and increase route length due to the insertion of u)
(use push forward rule to check feasibility efficiently)

Step 2: Compute for each unrouted customer u which can be feasibly inserted:

$$c_2(i(u^*), u^*, j(u^*)) = \max_u \{\lambda d_{0u} - c_1(i(u), u, j(u))\}$$

(max the benefit of servicing a node on a partial route rather than on a direct route)

Step 3: Insert the customer u^* from Step 2

Outline

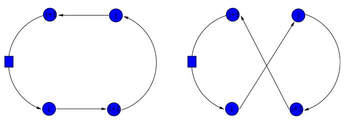
Local Search for CVRP and VRPTW

- Neighborhoods structures:
  - Intra-route: 2-opt, 3-opt, Lin-Kernighan (not very well suited) 2H-opt, Or-opt
  - Inter-routes: λ-interchange, relocate, exchange, cross, 2-opt*, ejection chains, GENI
- Solution representation and data structures
  - They depend on the neighborhood.
  - It can be advantageous to change them from one stage to another of the heuristic

Intra-route Neighborhoods

2-opt

$$\{i, i+1\}\{j, j+1\} \longrightarrow \{i, j\}\{i+1, j+1\}$$

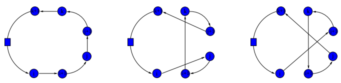


$O(n^2)$ possible exchanges
One path is reversed

Intra-route Neighborhoods

3-opt

$$\{i, i+1\}\{j, j+1\}\{k, k+1\} \longrightarrow \ldots$$

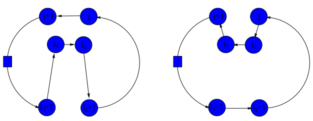


$O(n^3)$ possible exchanges
Paths can be reversed

Intra-route Neighborhoods

Or-opt [Or (1976)]

$$\{i_1 - 1, i_1\}\{i_2, i_2 + 1\}\{j, j+1\} \longrightarrow \{i_1 - 1, i_2 + 1\}\{j, i_1\}\{i_2, j+1\}$$



sequences of one, two, three consecutive vertices relocated
$O(n^2)$ possible exchanges — No paths reversed

Time windows: Feasibility check

In TSP verifying k-optimality requires $O(n^k)$ time
In TSPTW feasibility has to be tested then $O(n^{k+1})$ time

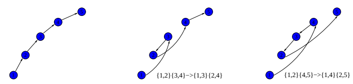
(Savelsbergh 1985) shows how to verify constraints in constant time
Search strategy + Global variables

$\Downarrow$

$O(n^k)$ for k-optimality in TSPTW

Search Strategy

- Lexicographic search, for 2-exchange:
  - $i = 1, 2, \ldots, n-2$ (outer loop)
  - $j = i+2, i+3, \ldots, n$ (inner loop)



Previous path is expanded by the edge $\{j-1, j\}$

Global variables (auxiliary data structure)

- Maintain auxiliary data such that it is possible to:
  - handle single move in constant time
  - update their values in constant time

Ex.: in case of time windows:

- total travel time of a path
- earliest departure time of a path
- latest arrival time of a path

Inter-route Neighborhoods

[Savelsbergh, ORSA (1992)]

Figure 6. The exchange neighborhood.

Inter-route Neighborhoods

[Savelsbergh, ORSA (1992)]

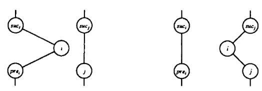


Figure 5. The relocate neighborhood.

Inter-route Neighborhoods

[Savelsbergh, ORSA (1992)]

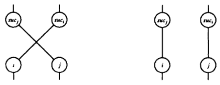


Figure 7. The cross neighborhood.

Inter-route Neighborhoods

GENI: generalized insertion [Gendreau, Hertz, Laporte, Oper. Res. (1992)]

- select the insertion restricted to the neighborhood of the vertex to be added (not necessarily between consecutive vertices)
- perform the best 3- or 4-opt restricted to reconnecting arc links that are close to one another.

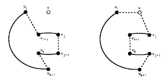


Figure 1. Type I insertion of vertex v between v_i and v_j.

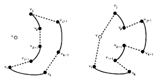


Figure 2. Type II insertion of vertex v between v_i and v_j.

General recommendation: use a combination of 2-opt* + or-opt [Potvin, Rousseau, (1995)]

However,

- Designing a local search algorithm is an engineering process in which learnings from other courses in CS might become important.
- It is important to make such algorithms as much efficient as possible.
- Many choices are to be taken (search strategy, order, auxiliary data structures, etc.) and they may interact with instance features. Often a trade-off between examination cost and solution quality must be decided.
- The assessment is conducted through:
  - analytical analysis (computational complexity)
  - experimental analysis

Table 5.6. *The effect of 3-opt on the Clarke and Wright algorithm.*

	Sequential				Parallel			
Problem	No 3-opt[1]	+ 3-opt FI[2]	+ 3-opt BI[3]	K[4]	No 3-opt[5]	+ 3-opt FI[6]	+ 3-opt BI[7]	K[8]
E051-05e	625.56	624.20	624.20	5	584.64	578.56	578.56	6
E076-10e	1005.25	991.94	991.94	10	900.26	888.04	888.04	10
E101-08e	982.48	980.93	980.93	8	886.83	878.70	878.70	8
E101-10c	939.99	930.78	928.64	10	833.51	824.42	824.42	10
E121-07c	1291.33	1232.90	1237.26	7	1071.07	1049.43	1048.53	7
E151-12c	1299.39	1270.34	1270.34	12	1133.43	1128.24	1128.24	12
E200-17c	1708.00	1667.65	1669.74	16	1395.74	1386.84	1386.84	17
D051-06c	670.01	663.59	663.59	6	618.40	616.66	616.66	6
D076-11c	989.42	988.74	988.74	12	975.46	974.79	974.79	12
D101-09c	1054.70	1046.69	1046.69	10	973.94	968.73	968.73	9
D101-11c	952.53	943.79	943.79	11	875.75	868.50	868.50	11
D121-11c	1646.60	1638.39	1637.07	11	1596.72	1587.93	1587.93	11
D151-14c	1383.87	1374.15	1374.15	15	1287.64	1284.63	1284.63	15
D200-18c	1671.29	1652.58	1652.58	20	1538.66	1523.24	1521.94	19

[1] Sequential savings.
[2] Sequential savings + 3-opt and first improvement.
[3] Sequential savings + 3-opt and best improvement.
[4] Sequential savings: number of vehicles in solution.
[5] Parallel savings.
[6] Parallel savings + 3-opt and first improvement.
[7] Parallel savings + 3-opt and best improvement.
[8] Parallel savings: number of vehicles in solution.

What is best?

Outline

Tabu Search for VRPTW [Potvin (1996)]

Initial solution: Solomon's insertion heuristic

Neighborhood: or-opt and 2-opt* (in VNS fashion or neighborhood union)
speed up in or-opt: i is moved between j and $j+q$ if i is one of the h nearest neighbors

Step : best improvement

Tabu criterion: forbidden to reinsert edges which were recently removed

Tabu length: fixed

Aspiration criterion: tabu move is overridden if an overall best is reached

End criterion: number of iterations without improvements

Taburoute

[Gendreau, Hertz, Laporte, 1994]

Neighborhood: remove one vertex from one route and insert with GENI in another that contains one of its p nearest neighbors
Re-optimization of routes at different stages

Tabu criterion: forbidden to reinsert vertex in route

Tabu length: random from $[5, 10]$

Evaluation function: possible to examine infeasible routes + diversification component:

- penalty term measuring overcapacity (every 10 iteration multiplied or divided by 2)
- penalty term measuring overduration
- frequency of movement of a vertex currently considered

Overall strategy: false restart (initially several solutions, limited search for each of them, selection of the best)

False restart:

Step 1: (Initialization) Generate $\lceil\sqrt{n}/2\rceil$ initial solutions and perform tabu search on $W' \subset W = V \setminus \{0\}$ ($|W'| \approx 0.9|W|$) up to 50 idle iterations.

Step 2: (Improvement) Starting with the best solution observed in Step 1 perform tabu search on $W' \subset W = V \setminus \{0\}$ ($|W'| \approx 0.9|W|$) up to $50n$ idle iterations.

Step 3: (Intensification) Starting with the best solution observed in Step 2, perform tabu search up to 50 idle iterations.
Here W' is the set of the $\lceil|V|/2\rceil$ vertices that have been most often moved in Steps 1 and 2.

Adaptive Memory Procedure

[Rochart and Taillard, 1995]

1. Keep an adaptive memory as a pool of good solutions
2. Some element (single tour) of these solutions are combined together to form new solution (more weight is given to best solutions)
3. Partial solutions are completed by an insertion procedure.
4. Tabu search is applied at the tour level

Granular Tabu Search

[Toth and Vigo, 1995]

Long edges are unlikely to be in the optimal solution

⇓

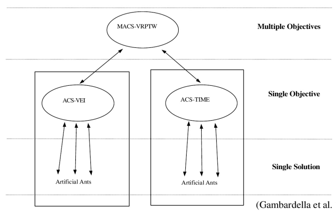
Remove those edges that exceed a granularity threshold ν

$$\nu = \beta\bar{c}$$

- β sparsification parameter
- $\bar{c}$ average length for a solution from a construction heuristic
- adjust β after a number of idle iterations

Ant Colony System [Gambardella et al. 1999]

VRP-TW: in case of vehicle and distance minimization two ant colonies are working in parallel on the two objective functions (colonies exchange pheromone information)



(Gambardella et al. 1999

Constraints: A constructed solution must satisfy i) each customer visited once ii) capacity not exceeded iii) Time windows not violated

Pheromone trails: associated with connections (desirability of order)

Heuristic information: savings + time considerations

Solution construction:

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}^\alpha \eta_{il}^\beta} \qquad j \in N_i^k$$

if no feasible, open a new route
or decide routes to merge
if customers left out use an insertion procedure

Pheromone update:

Global $\quad \tau_{ij} \leftarrow \tau_{ij} + \rho\Delta\tau_{ij}^{bs} \quad \forall(i,j) \in T^{bs}$

Local $\quad \tau_{ij} \leftarrow (1-\epsilon)\tau_{ij} + \epsilon\tau_o^{bs} \quad \forall(i,j) \in T^{bs}$

Outline

Vehicle Routing with Backhauls (VRPB)

Further Input from CVRP:

- a partition of customers:
  $L = \{1, \ldots, n\}$ Lineahaul customers (deliveries)
  $B = \{n+1, \ldots, n+m\}$ Backhaul customers (collections)
- precedence constraint:
  in a route, customers from L must be served before customers from B

Task: Find a collection of K simple circuits with minimum costs, such that:

- each circuit visit the depot vertex
- each customer vertex is visited by exactly one circuit; and
- the sum of the demands of the vertices visited by a circuit does not exceed the vehicle capacity Q.
- in any circuit all the linehaul customers precede the backhaul customers, if any.

Vehicle Routing with Pickup and Delivery (VRPPD)

Further Input from CVRP:

- each customer i is associated with quantities d_i and p_i to be delivered and picked up, resp.
- for each customer i, O_i denotes the vertex that is the origin of the delivery demand and D_i denotes the vertex that is the destination of the pickup demand

Task:
Find a collection of K simple circuits with minimum costs, such that:

- each circuit visit the depot vertex
- each customer vertex is visited by exactly one circuit; and
- the current load of the vehicle along the circuit must be non-negative and may never exceed Q
- for each customer i, the customer O_i when different from the depot, must be served in the same circuit and before customer i
- for each customer i, the customer D_i when different from the depot, must be served in the same circuit and after customer i

Multiple Depots VRP

Further Input from CVRP:

- multiple depots to which customers can be assigned
- a fleet of vehicles at each depot

Task:
Find a collection of K simple circuits for each depot with minimum costs, such that:

- each circuit visit the depot vertex
- each customer vertex is visited by exactly one circuit; and
- the current load of the vehicle along the circuit must be non-negative and may never exceed Q
- vehicles start and return to the depots they belong

Vertex set $V = \{1, 2, \ldots, n\}$ and $V_0 = \{n+1, \ldots, n+m\}$
Route i defined by $R_i = \{l, 1, \ldots, l\}$

Periodic VRP

Further Input from CVRP:

- planning period of M days

Task:
Find a collection of K simple circuits with minimum costs, such that:

- each circuit visit the depot vertex
- each customer vertex is visited by exactly one circuit; and
- the current load of the vehicle along the circuit must be non-negative and may never exceed Q
- A vehicle may not return to the depot in the same day it departs.
- Over the M-day period, each customer must be visited l times, where $1 \leq l \leq M$.

Three phase approach:

1. Generate feasible alternatives for each customer.
   Example, $M = 3$ days $\{d1, d2, d3\}$ then the possible combinations are: $0 \rightarrow 000$; $1 \rightarrow 001$; $2 \rightarrow 010$; $3 \rightarrow 011$; $4 \rightarrow 100$; $5 \rightarrow 101$; $6 \rightarrow 110$; $7 \rightarrow 111$.

Customer	Diary Demand	Number of Visits	Number of Combinations	Possible Combinations
1	30	1	3	1,2,4
2	20	2	3	3,4,6
3	20	2	3	3,4,6
4	30	2	3	1,2,4
5	10	3	1	7

2. Select one of the alternatives for each customer, so that the daily constraints are satisfied. Thus, select the customers to be visited in each day.
3. Solve the vehicle routing problem for each day.

Split Delivery VRP

Constraint Relaxation: it is allowed to serve the same customer by different vehicles. (necessary if $d_i > Q$)

Task:
Find a collection of K simple circuits with minimum costs, such that:
- each circuit visit the depot vertex
- the current load of the vehicle along the circuit must be non-negative and may never exceed Q

Note: a SDVRP can be transformed into a VRP by splitting each customer order into a number of smaller indivisible orders [Burrows 1988].

Inventory VRP

Input:
- a facility, a set of customers and a planning horizon T
- r_i product consumption rate of customer i (volume per day)
- C_i maximum local inventory of the product for customer i
- a fleet of M homogeneous vehicles with capacity Q

Task:
Find a collection of K daily circuits to run over the planing horizon with minimum costs and such that:
- each circuit visit the depot vertex
- no customer goes in stock-out during the planning horizon
- the current load of the vehicle along the circuit must be non-negative and may never exceed Q

Other VRPs

VRP with Satellite Facilities (VRPSF)

Possible use of satellite facilities to replenish vehicles during a route.

Open VRP (OVRP)

The vehicles do not need to return at the depot, hence routes are not circuits but paths

Dial-a-ride VRP (DARP)

- It generalizes the VRPTW and VRP with Pick-up and Delivery by incorporating time windows and maximum ride time constraints
- It has a human perspective
- Vehicle capacity is normally constraining in the DARP whereas it is often redundant in PDVRP applications (collection and delivery of letters and small parcels)

Part XX

Vehicle Routing, Rich Models

Outline

Outline

Performance of exact methods

Current limits of exact methods [Ropke, Pisinger (2007)]:

CVRP: up to 135 customers by branch and cut and price

VRPTW: 50 customers (but 1000 customers can be solved if the instance has some structure)

CP can handle easily side constraints but hardly solve VRPs with more than 30 customers.

Large Neighborhood Search

Other approach with CP: [Shaw, 1998]

- Use an over all local search scheme
- Moves change a large portion of the solution
- CP system is used in the exploration of such moves.
- CP used to check the validity of moves and determine the values of constrained variables
- As a part of checking, constraint propagation takes place. Later, iterative improvement can take advantage of the reduced domains to speed up search by performing fast legality checks.

Solution representation:

- Handled by local search:
  Next pointers: A variable n_i for every customer i representing the next visit performed by the same vehicle
  $$n_i \in N \cup S \cup E$$
  where $S = \bigcup S_k$ and $E = \bigcup E_k$ are additional visits for each vehicle k marking the start and the end of the route for vehicle k
- Handled by the CP system: time and capacity variables.

In the literature, the overall heuristic idea received different names:

- Removal and reinsertion
- Ruin and repair
- Iterated greedy
- Fix and re-optimize

Remove

Remove some related customers
(their re-insertion is likely to change something)

Relatedness measure r_{ij}

- geographical
  $$r_{ij} = \frac{1}{D}(d'(i,j) + d'(i,j+n) + d'(i+n,j) + d'(i+n,j+n))$$
- temporal and load based
  $$d'(u,v) = |T_{p_i} - T_{p_j}| + |T_{d_i} - T_{d_j}| + |l_i - l_j|$$
- cluster removal
- history based: neighborhood graph removal

Dispersion sub-problem:
choose q customers to remove with minimal r_{ij}

Heuristic stochastic procedure:

- choose a pair randomly;
- select an already removed i and find j that minimizes r_{ij}

Insertion

by CP:

- constraint propagation rules: time windows, load and bound considerations
- search heuristic most constrained variable + least constrained valued (for each v find cheapest insertion and choose v with largest such cost)
- Complete search: ok for 15 visits (25 for VRPTW) but with heavy tails
- Limited discrepancy search

[Shaw, 1998]

```
Reinsert(RoutingPlan plan, VisitSet visits, integer discrep)
    if |visits| = 0 then
        if Cost(plan) < Cost(bestplan) then
            bestplan := plan
        end if
    else
        Visit v := ChooseFarthestVisit(visits)
        integer i := 0
        for p in rankedPositions(v) and i ≤ discrep do
            Store(plan) // Preserve plan on stack
            InsertVisit(plan, v, p)
            Reinsert(plan, visits - v, discrep - i)
            Restore(plan) // Restore plan from stack
            i := i + 1
        end for
    end if
end Reinsert
```


Other insertion procedures:

- Greedy (cheapest insertion)
- Max regret:
  Δf_i^q due to insert i into its best position in its q^{th} best route
  $i = \arg\max(\Delta f_i^2 - \Delta f_i^1)$

Advantages of removal-reinsert procedure with many side constraints:

- the search space in local search may become disconnected
- it is easier to implement feasibility checks
- no need of computing delta functions in the objective function

Further ideas

- Adaptive removal: start by removing 1 pair and increase after a certain number of iterations
- use of roulette wheel to decide which removal and reinsertion heuristic to use
  $$p_i = \frac{\pi_i}{\sum \pi_i} \quad \text{for each heuristic } i$$
- SA as accepting criterion after each reconstruction

Outline

Stochastic VRP (SVRP)

Stochastic VRP (SVRP) are VRPs where one or several components of the problem are random.
Three different kinds of SVRP are:

- Stochastic customers: Each customer i is present with probability p_i and absent with probability $1 - p_i$.
- Stochastic demands: The demand d_i of each customer is a random variable.
- Stochastic times: Service times δ_i and travel times t_{ij} are random variables.

Current Research Directions

Optimization under uncertainty: some problem parameters are unknown.

- Stochastic optimization: If probability distributions governing the data are known or can be estimated

  In stochastic optimization the goal is to find some policy that is feasible for all (or almost all) the possible data instances and maximizes the expectation of some function of the decisions and the random variables.

- Robust optimization: If the parameters are known only within certain bounds

  In robust optimization the goal is to find a solution which is feasible for all such data and optimal in some sense.

Current Research Directions

Multistage stochastic optimization:

- Requests arrive dynamically
- Decisions on which requests to serve and how must be taken with a certain frequency
- Previous decisions can be changed to accommodate the new requests at best.
- large scale instances

➜ needed fast solvers that account for possible incoming data