DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 1
Introduction to Scheduling: Terminology, Classification

Marco Chiarandini

Outline

1. Course Introduction
2. Scheduling
   Problem Classification
3. Complexity Hierarchy

Outline

1. Course Introduction
2. Scheduling
   Problem Classification
3. Complexity Hierarchy

Course Presentation

Communication media

- Black Board (BB). What we use:
  - Mail
  - Announcements
  - Course Documents (for Photocopies)
  - Blog – Students' Lecture Diary
  - Electronic hand in of the exam project
- Web-site `http://www.imada.sdu.dk/~marco/DM204/`
  - Lecture plan and slides
  - Literature and Links
  - Exam documents

- Schedule

Third quarter 2008		Fourth quarter 2008	
Tuesday	10:15-12:00	Wednesday	12:15-14:00
Friday	8:15-10:00	Friday	10:15-12:00

- ∼ 27 lectures

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
- Schedule: Project hand in deadline + oral exam in June

Course Content

- General Optimization Methods
  - Mathematical Programming,
  - Constraint Programming,
  - Heuristics
  - Problem Specific Algorithms (Dynamic Programming, Branch and Bound)
- Scheduling
  - Single and Parallel Machine Models
  - Flow Shops and Flexible Flow Shops
  - Job Shops
  - Resource-Constrained Project Scheduling
- Timetabling
  - Interval Scheduling, Reservations
  - Educational Timetabling
  - Workforce and Employee Timetabling
  - Transportation Timetabling
- Vehicle Routing
  - Capacited Vehicle Routing
  - Vehicle Routing with Time Windows

Course Material

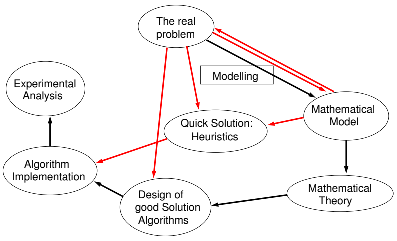
- Literature
  B1 Pinedo, M. Planning and Scheduling in Manufacturing and Services Springer Verlag, 2005
  B2 Pinedo, M. Scheduling: Theory, Algorithms, and Systems Springer New York, 2008
  B3 Toth, P. & Vigo, D. (ed.) The Vehicle Routing Problem SIAM Monographs on Discrete Mathematics and Applications, 2002
- Slides
- Class exercises (participatory)

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



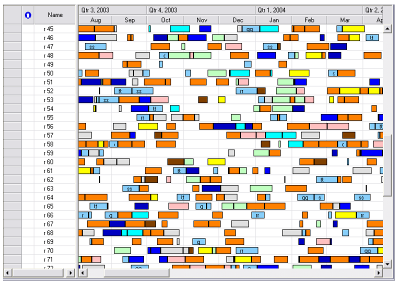
Outline

1. Course Introduction
2. Scheduling
   Problem Classification
3. Complexity Hierarchy

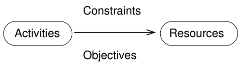
Scheduling

- 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



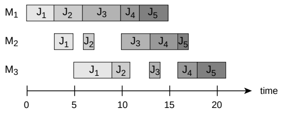
Problem Definition
Given: a set of **jobs** $\mathcal{J} = \{J_1, \dots, J_n\}$ that have to be processed by a set of **machines** $\mathcal{M} = \{M_1, \dots, M_m\}$
Find: a **schedule**,
i.e., a mapping of jobs to machines and processing times subject to feasibility and optimization constraints.

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 $O_{j1}, O_{j2}, \dots, O_{jn_j}$ 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)

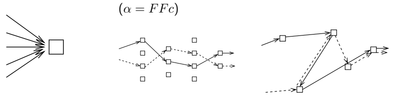
[R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1979): Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 4, 287-326.]

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

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

- 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



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

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

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

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

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

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

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

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

- 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 $U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & \text{otherwise} \end{cases}$

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

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

- Makespan: Maximum completion $C_{max} = \max\{C_1, \dots, C_n\}$ tends to max the use of machines
- Maximum lateness $L_{max} = \max\{L_1, \dots, L_n\}$
- 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 $C_1, \dots, C_n$) except E_i

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

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

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

Gate Assignment at an Airport

- 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.

Exercises

Scheduling Tasks in a Central Processing Unit (CPU)

- 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

Paper bag factory

- Basic raw material for such an operation are rolls of paper.
- Production process consists of three stages: (i) printing of the logo, (ii) gluing of the side of the bag, (iii) sewing of one end or both ends.
- Each stage consists of a number of machines which are not necessarily identical.
- Each production order indicates a given quantity of a specific bag that has to be produced and shipped by a committed shipping date or due date.
- Processing times for the different operations are proportional to the number of bags ordered.
- There are setup times when switching over different types of bags (colors, sizes) that depend on the similarities between the two consecutive orders
- A late delivery implies a penalty that depends on the importance of the order or the client and the tardiness of the delivery.

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 is feasible and it minimizes the given objective.

Classes of Schedules

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. (local shift)

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. (global shift without preemption)

Nondelay schedule

A feasible schedule is called nondelay if no machine is kept idle while an operation is waiting for processing. (global shift with preemption)

- There are optimal schedules that are nondelay for most models with regular objective function.
- There exists for $Jm||\gamma$ (γ regular) an optimal schedule that is active.
- nondelay ⇒ active but active ⇏ nondelay

Outline

1. Course Introduction
2. Scheduling
   Problem Classification
3. Complexity Hierarchy

Complexity Hierarchy

Reduction

A search problem Π is (polynomially) reducible to a search problem Π' ($\Pi \longrightarrow \Pi'$) if there exists an algorithm $\mathcal{A}$ that solves Π by using a hypothetical subroutine $\mathcal{S}$ for Π' and except for $\mathcal{S}$ everything runs in polynomial time. [Garey and Johnson, 1979]

NP-hard

A search problem Π' is NP-hard if

1. it is in NP
2. there exists some NP-complete problem Π that reduces to Π'

In scheduling, complexity hierarchies describe relationships between different problems.

Ex: $1||\sum C_j \longrightarrow 1||\sum w_j C_j$

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

Problems Involving Numbers

Partition

- **Input:** finite 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 disjoint sets $A_1, \dots, A_m$ 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)?

DMP204
SCHEDULING, TIMETABLING AND ROUTING

Lecture 2

Marco Chiarandini

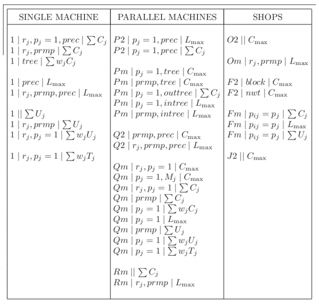
Outline

1. Complexity Hierarchies

Outline

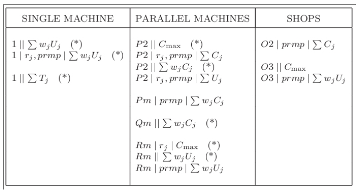
1. Complexity Hierarchies

Polynomial time solvable problems



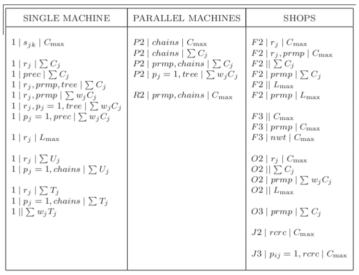
SINGLE MACHINE	PARALLEL MACHINES	SHOPS
$1 \mid r_j, p_j = 1, prec \mid \sum C_j$ $1 \mid r_j, prmp \mid \sum C_j$ $1 \mid tree \mid \sum w_j C_j$ $1 \mid prec \mid L_{\max}$ $1 \mid r_j, prmp, prec \mid L_{\max}$ $1 \mid\mid \sum U_j$ $1 \mid r_j, prmp \mid \sum U_j$ $1 \mid r_j, p_j = 1 \mid \sum w_j U_j$ $1 \mid r_j, p_j = 1 \mid \sum w_j T_j$	$P2 \mid p_j = 1, prec \mid L_{\max}$ $P2 \mid p_j = 1, prec \mid \sum C_j$ $Pm \mid p_j = 1, tree \mid C_{\max}$ $Pm \mid prmp, tree \mid C_{\max}$ $Pm \mid p_j = 1, outtree \mid \sum C_j$ $Pm \mid p_j = 1, intree \mid L_{\max}$ $Pm \mid prmp, intree \mid L_{\max}$ $Q2 \mid prmp, prec \mid C_{\max}$ $Q2 \mid r_j, prmp, prec \mid L_{\max}$ $Qm \mid r_j, p_j = 1 \mid C_{\max}$ $Qm \mid p_j = 1, M_j \mid C_{\max}$ $Qm \mid r_j, p_j = 1 \mid \sum C_j$ $Qm \mid prmp \mid \sum C_j$ $Qm \mid p_j = 1 \mid \sum w_j C_j$ $Qm \mid p_j = 1 \mid L_{\max}$ $Qm \mid prmp \mid \sum U_j$ $Qm \mid p_j = 1 \mid \sum w_j U_j$ $Qm \mid p_j = 1 \mid \sum w_j T_j$ $Rm \mid\mid \sum C_j$ $Rm \mid r_j, prmp \mid L_{\max}$	$O2 \mid\mid C_{\max}$ $Om \mid r_j, prmp \mid L_{\max}$ $F2 \mid block \mid C_{\max}$ $F2 \mid nwt \mid C_{\max}$ $Fm \mid p_{ij} = p_j \mid \sum C_j$ $Fm \mid p_{ij} = p_j \mid L_{\max}$ $Fm \mid p_{ij} = p_j \mid \sum U_j$ $J2 \mid\mid C_{\max}$

NP-hard problems in the ordinary sense



SINGLE MACHINE	PARALLEL MACHINES	SHOPS
$1 \mid\mid \sum w_j U_j$ (*) $1 \mid r_j, prmp \mid \sum w_j U_j$ (*) $1 \mid\mid \sum T_j$ (*)	$P2 \mid\mid C_{\max}$ (*) $P2 \mid r_j, prmp \mid \sum C_j$ $P2 \mid\mid \sum w_j C_j$ (*) $P2 \mid r_j, prmp \mid \sum U_j$ $Pm \mid prmp \mid \sum w_j C_j$ $Qm \mid\mid \sum w_j C_j$ (*) $Rm \mid r_j \mid C_{\max}$ (*) $Rm \mid\mid \sum w_j U_j$ (*) $Rm \mid prmp \mid \sum w_j U_j$	$O2 \mid prmp \mid \sum C_j$ $O3 \mid\mid C_{\max}$ $O3 \mid prmp \mid \sum w_j U_j$

Strongly NP-hard problems

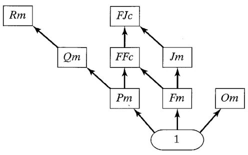


SINGLE MACHINE	PARALLEL MACHINES	SHOPS
$1 \mid s_{jk} \mid C_{\max}$ $1 \mid r_j \mid \sum C_j$ $1 \mid prec \mid \sum C_j$ $1 \mid r_j, prmp, tree \mid \sum C_j$ $1 \mid r_j, prmp \mid \sum w_j C_j$ $1 \mid r_j, p_j = 1, tree \mid \sum w_j C_j$ $1 \mid p_j = 1, prec \mid \sum w_j C_j$ $1 \mid r_j \mid L_{\max}$ $1 \mid r_j \mid \sum U_j$ $1 \mid p_j = 1, chains \mid \sum U_j$ $1 \mid r_j \mid \sum T_j$ $1 \mid p_j = 1, chains \mid \sum T_j$ $1 \mid\mid \sum w_j T_j$	$P2 \mid chains \mid C_{\max}$ $P2 \mid chains \mid \sum C_j$ $P2 \mid prmp, chains \mid \sum C_j$ $P2 \mid p_j = 1, tree \mid \sum w_j C_j$ $R2 \mid prmp, chains \mid C_{\max}$	$F2 \mid r_j \mid C_{\max}$ $F2 \mid r_j, prmp \mid C_{\max}$ $F2 \mid\mid \sum C_j$ $F2 \mid prmp \mid \sum C_j$ $F2 \mid\mid L_{\max}$ $F2 \mid prmp \mid L_{\max}$ $F3 \mid\mid C_{\max}$ $F3 \mid prmp \mid C_{\max}$ $F3 \mid nwt \mid C_{\max}$ $O2 \mid r_j \mid C_{\max}$ $O2 \mid\mid \sum C_j$ $O2 \mid prmp \mid \sum w_j C_j$ $O2 \mid\mid L_{\max}$ $O3 \mid prmp \mid \sum C_j$ $J2 \mid rcrc \mid C_{\max}$ $J3 \mid p_{ij} = 1, rcrc \mid C_{\max}$

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

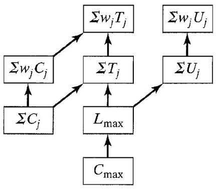
Complexity Hierarchy

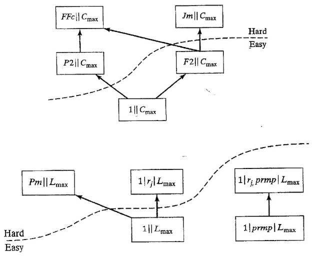
Elementary reductions for machine environment



Complexity Hierarchy

Elementary reductions for regular objective functions





DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 3

RCPSP and Mixed Integer Programming

Marco Chiarandini

Outline

1. Scheduling
   CPM/PERT
   Resource Constrained Project Scheduling Model

2. Mathematical Programming
   Introduction
   Solution Algorithms

Outline

1. Scheduling
   CPM/PERT
   Resource Constrained Project Scheduling Model

2. Mathematical Programming
   Introduction
   Solution Algorithms

Project Planning

Milwaukee General Hospital Project

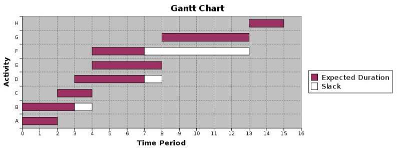
Activity	Description	Immediate Predecessor	*Duration*
A	Build internal components	–	2
B	Modify roof and floor	–	3
C	Construct collection stack	A	2
D	Pour concrete and install frame	A,B	4
E	Build high-temperature burner	C	4
F	Install pollution control system	C	3
G	Install air pollution device	D,E	5
H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

RCPSP

Resource Constrained Project Scheduling Model

Given:

- activities (jobs) $j = 1, \dots, n$
- renewable resources $i = 1, \dots, m$
- 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}.

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 $c_1, \dots, c_m$,
- h teachers $a_1, \dots, a_h$ and
- T teaching periods $t_1, \dots, t_T$.
- Furthermore, we have lectures $i = l_1, \dots, l_n$.
- 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 $J_1, \dots, J_g$ are to be processed by auditors $A_1, \dots, A_m$.
- Job J_l consists of n_l tasks $(l = 1, \dots, g)$.
- 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 $[s_k^\nu, l_k^\nu]$ ($\nu = 1, \dots, m)$ with $l_k^\nu < s_k^\nu$ for $\nu = 1, \dots, m_k - 1$.
- Furthermore, the total working time of A_k is bounded from below by H_k^- and from above by H_k^+ with $H_k^- \leq H_k^+$ $(k = 1, \dots, m)$.
- We have to find an assignment $\alpha(i)$ for each task $i = 1, \dots, n := \sum_{l=1}^{g} n_l$ 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 $k = 1, \dots, m$.
  - 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.

Outline

1. Scheduling
   CPM/PERT
   Resource Constrained Project Scheduling Model

2. Mathematical Programming
   Introduction
   Solution Algorithms

Mathematical Programming

Linear, Integer, Nonlinear

program = optimization problem

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

general (nonlinear) program (NLP)

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

linear program (LP)

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

integer (linear) program (IP, MIP)

Linear Programming

Linear Program in standard form

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

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

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.html`
- 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

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

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

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

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

Solution Algorithms

- 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

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

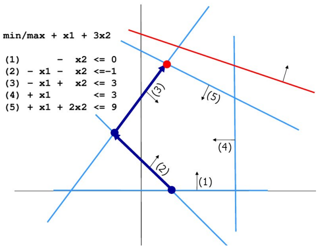
Linear programming

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.

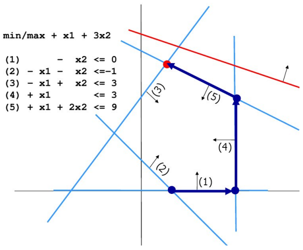
The simplex method

```
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).

Integer Programming (easy)

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

Integer Programming (hard)

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.

Integer Programming (hard)

- 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/]

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 4

Mixed Integer Programming (2)

Marco Chiarandini

Outline

Modeling

- Min cost flow
- Shortest path
- Max flow
- Assignment and Bipartite Matching
- Transportation
- Multicommmodies

Modeling

Set Covering

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

Set Partitioning

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

Set Packing

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

Traveling Salesman Problem

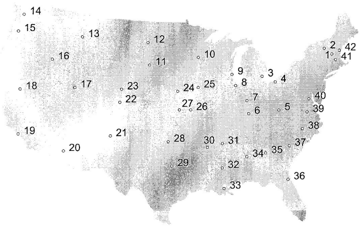


Figure 3.1 Locations of the 42 cities.

Traveling Salesman Problem

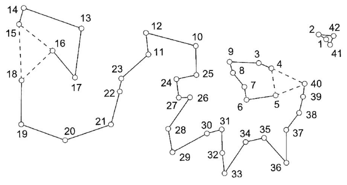


Figure 3.2 Solution of the initial LP relaxation.

Traveling Salesman Problem

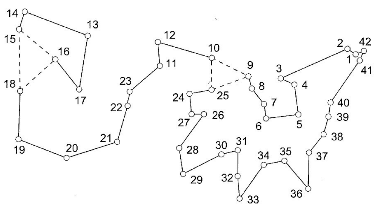


Figure 3.3 LP solution after three subtour constraints.

Traveling Salesman Problem

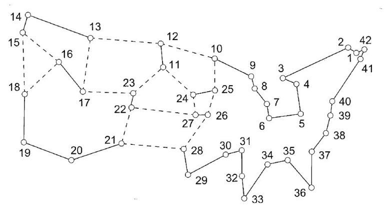


Figure 3.4 LP solution satisfying all subtour constraints.

Traveling Salesman Problem

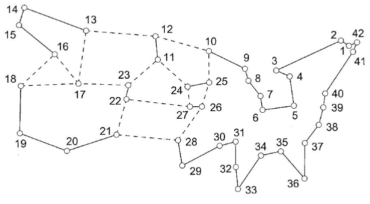


Figure 3.7 What is wrong with this vector?

Traveling Salesman Problem

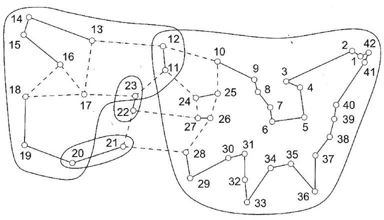


Figure 3.8 A violated comb.

Traveling Salesman Problem

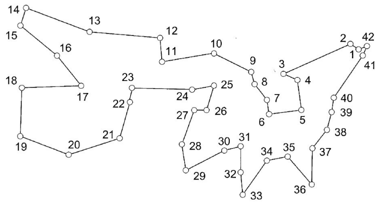


Figure 3.9 An optimal tour through 42 cities.

minimize $c^T x$ subject to

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

$\sum(x_e \,:\, v$ is an end of $e) = 2$ for all cities v,

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

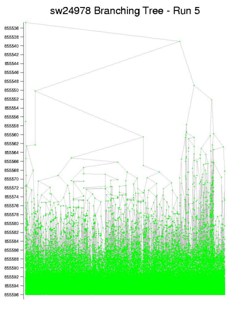
$\sum_{i=0}^{i=3}(\sum(x_e \,:\, e$ has one end in S_i and one end not in $S_i) \geq 10$,
for any comb

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`

MIP for Scheduling

- Formulation for $Qm|p_j = 1|\sum h_j(C_j)$ and relation with transportation problems
- Formulation of $1|prec|\sum w_j C_j$ and $Rm||\sum C_j$ as weighted bipartite matching and assignment problems.
- Formulation of $1|prec|\sum w_j C_j$ and how to deal with disjunctive constraints

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 5

Mixed Integer Programming
Models and Exercises

Marco Chiarandini

Outline

1. Models
2. An Overview of Software for MIP
3. ZIBOpt

Outline

1. Models
2. An Overview of Software for MIP
3. ZIBOpt

Modeling

- Min cost flow
- Shortest path
- Max flow
- Assignment and Bipartite Matching
- Transportation
- Multicommmodies

Modeling

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

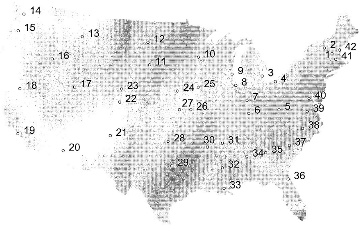


Figure 3.1 Locations of the 42 cities.

Traveling Salesman Problem

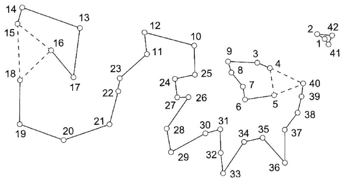


Figure 3.2 Solution of the initial LP relaxation.

Traveling Salesman Problem

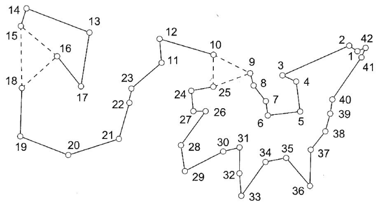


Figure 3.3 LP solution after three subtour constraints.

Traveling Salesman Problem

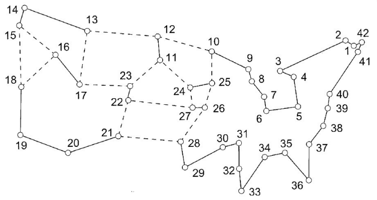


Figure 3.4 LP solution satisfying all subtour constraints.

Traveling Salesman Problem

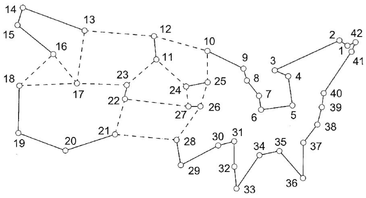


Figure 3.7 What is wrong with this vector?

Traveling Salesman Problem

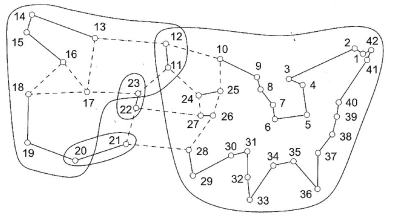


Figure 3.8 A violated comb.

Traveling Salesman Problem

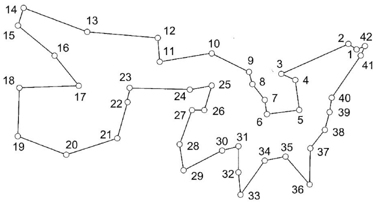


Figure 3.9 An optimal tour through 42 cities.

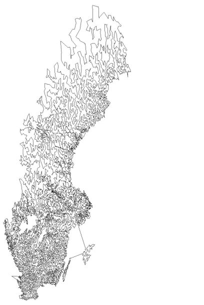
minimize $c^T x$ subject to

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

$\sum(x_e \,:\, v$ is an end of $e) = 2$ for all cities v,

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

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

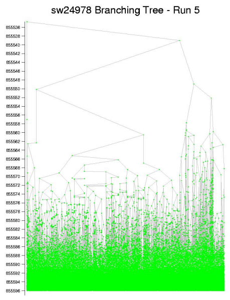


24,978 Cities

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`

MIP for Scheduling

- Formulation for $Qm|p_j = 1|\sum h_j(C_j)$ and relation with transportation problems
- Formulation of $1|prec|\sum w_j C_j$ and $Rm||\sum C_j$ as weighted bipartite matching and assignment problems.
- Formulation of $1|prec|\sum w_j C_j$ and how to deal with disjunctive constraints

Outline

1. Models
2. An Overview of Software for MIP
3. ZIBOpt

How to solve MIP 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 MIP 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 MIP 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 MIP 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 MIP 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

1. Models

2. An Overview of Software for MIP

3. ZIBOpt

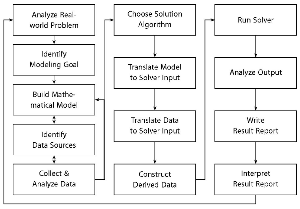
ZIBOpt

- 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.

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 6
Constraint Programming

Marco Chiarandini

Outline

1. Math Programming
   Scheduling Models
   Further issues

2. Constraint Programming
   Introduction

Outline

1. Math Programming
   Scheduling Models
   Further issues

2. Constraint Programming
   Introduction

Time indexed variables

$1|r_j|\sum w_j C_j$
Discretize time in $t = 0, \dots, l$, where l is upper bound

$$x_{jt} \in \{0,1\} \qquad j = 1, \dots, n;\ t = 0, \dots, l$$

Variables indicate if j starts at t

$$\sum_{t=1}^{l} x_{jt} = 1 \qquad \forall j = 1, \dots, n$$

Every job starts at one point in time

$$\sum_{j=1}^{n} \sum_{s=\max\{t-p_j,0\}}^{t-1} x_{js} \leq 1 \qquad \forall t = 0, \dots, l$$

At most one job can be processed in time

$$x_{jt} = 0 \forall j = 1, \dots, n,\ t = 0, \dots, \max\{r_j - 1, 0\}$$

Jobs cannot start before their release dates

$$\min \sum_{j=1}^{n} \sum_{t=0}^{l} w_j (t + p_j) x_{js}$$

Objective

Sequencing variables

$1|prec|\sum w_j C_j$

$$x_{jk} \in \{0,1\} \qquad j, k = 1, \dots, n$$

Variables indicate if j preceeds k

$$x_{jj} = 0 \qquad \forall j = 1, \dots, n$$

$$x_{kj} + x_{jk} = 1 \qquad \forall j, k = 1, \dots, n, j \neq k$$

Precedence constraints

$$x_{kj} + x_{lk} + x_{jl} \geq 1 \qquad j, k, l = 1, \dots, nj \neq k, k \neq l, j \neq l$$

Precedence constraints

$$\min \sum_{j=1}^{n} \sum_{k=1}^{n} w_j p_k x_{kj} + \sum_{j=1}^{n} w_j p_j$$

Objective

Real Variables
Disjunctive Programming

$1|prec|\sum w_j C_j$

Disjunctive graph model made of conjunctive arcs A and disjunctive arcs I.
Select disjunctive arcs such that the grph does not contain a cycle.

$$x_j \in \mathbf{R} \qquad j = 1, \dots, n$$

Variables denote completion of job j

$$x_k - x_j \geq p_k \qquad \forall j \rightarrow k \in A$$

precedence constraints conjunctive arcs

$$x_j \geq p_j \qquad \forall j = 1, \dots, n$$

min processing time

$$x_k - x_j \geq p_k \quad \text{or} \quad x_j - x_k \geq p_j \qquad \forall (i,j) \in I$$

Precedence constraints

$$\min \sum_{j=1}^{n} w_j x_j$$

Objective

Linearizations

How to linearize these non linear functions?

- Disjunctive constraints
- $\min |a - b|$
- $\min\{\max(a, b)\}$
- $\min \max_{i=1,\dots,m}(c_i^T x + d_i)$ piecewise-linear functions

Constraint types

Constraint type	Normalized representation
Set partitioning	$\sum x_i = 1$
Set packing	$\sum x_i \leqslant 1$
Set covering	$\sum x_i \geqslant 1$
Cardinality constraint	$\sum x_i = b$
Bin packing	$\sum a_i x_i + a_k x_k \leqslant a_k$
Invariant knapsack	$\sum x_i \leqslant b$
Knapsack	$\sum a_i x_i \leqslant b$
Integer knapsack	$\sum a_i y_i \leqslant b$
Variable lower bound	$p_k x_k - z_k \leqslant 0$ (or $p_k x_k - y_k \leqslant 0$)
Variable upper bound	$p_k x_k - z_k \geqslant 0$ (or $p_k x_k - y_k \geqslant 0$)
Mixed binary constraint	$\sum p_i x_i + \sum r_i z_i \leqslant t$ (or $=t$)
General constraint	$\sum p_i x_i + \sum q_i y_i + \sum r_i z_i \leqslant t$ (or $=t$)

x binary, y general integer, z a continous variable.
a and b integer numbers; p, q, r, s real numbers

- Specific domain propagation, preprocessing and cut generation exist for some of these constraints.
  [Achterberg, T. Constraint Integer Programming Department of Mathematics, Phd Thesis, Technical University of Berlin, Germany, 2007]

Outline

1. Math Programming
   Scheduling Models
   Further issues

2. Constraint Programming
   Introduction

Constraint Programming

Constraint Programming is about a fomrulation of the problem as a constraint satisfaction problem and about solving it by means of general or domain specific methods.

Constraint Satisfaction Problem

- **Input:**
  - a set of variables $X_1, X_2, \dots, X_n$
  - 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 specifies 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 $\{X_i = v_i, X_j = v_j, \dots\}$
  - 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.

Solution Process

Standard 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 previous assignments
- goal test: the current assignment is complete
- path cost: a constant cost for every step.

Two fundamental issues:

- exploration of search tree (of depth n)
- constraint propagation (filtering)
  - at every node of the search tree, remove domain values that do not belong to a solution
  - Repeat until nothing can be removed anymore

$\rightsquigarrow$ In CP, we mostly mean complete search but incomplete search is also included.

Constraint Propagation

Definition

Domain consistency A constraint C on the variables $x_1, \dots, x_k$ is called domain consistent if for each variable x_i and each value $d_i \in D(x_i)$ $(i = 1, \dots, k)$, there exist a value $d_j \in D(x_j)$ for all $j \neq i$ such that $(d_1, \dots, d_k) \in C$.

- domain consistency = hyper-arc consistency or generalized-arc consistency
- Establishing domain consistency for binary constraints is inexpensive.
- For higher arity constraints the naive approach requires time that is exponential in the number of variables.
- Exploiting underlying structure of a constraint can sometimes lead to establish domain consistency much more efficiently.

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.

  $$S_j + p_j \leq S_k$$

  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()`, `among()`, etc.

  Every higher order constraint can be reconduced to binary
  (you may need auxiliary constraints)
- Preference constraints
  cost on individual variable assignments

General Purpose Algorithms

Search algorithms

organize and explore the search tree

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

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 $\leftarrow$ 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 $\leftarrow$ RECURSIVE-BACKTRACKING(assignment, csp)
        if result $\neq$ 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

Implemnetation Refinements

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?

1) 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.

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

Propagating information through constraints

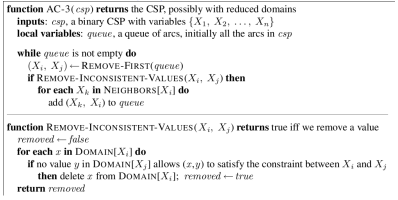
- Implicit in Select-Unassigned-Variable
- Forward checking (coupled with MRV)
- Constraint propagation (filtering)
  - arc consistency: force all (directed) arcs uv to be consistent: $\exists$ a value in $D(v)$: $\forall$ values in $D(u)$, otherwise detects inconsistency

    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.

Example: 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 $removed$
```

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?

Backtracking-Search

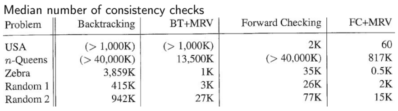
- chronological backtracking, the most recent decision point is revisited
- backjumping, backtracks to the most recent variable in the conflict set (set of previously assigned variables connected to X by constraints).

  every branch pruned by backjumping is also pruned by forward checking

  idea remains: backtrack to reasons of failure.

An Empirical Comparison

Median number of consistency checks



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

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

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

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 7

Constraint Programming (2)

Marco Chiarandini

Outline

1. Refinements on CP
   - Refinements: Modeling
   - Refinements: Search
   - Refinements: Constraints
   - Symmetry Breaking
   - Reification

2. Language and Systems

Outline

1. Refinements on CP
   - Refinements: Modeling
   - Refinements: Search
   - Refinements: Constraints
   - Symmetry Breaking
   - Reification

2. Language and Systems

A Puzzle Example

SEND +
MORE =
MONEY

Two representations

- The first yields initially a weaker constraint propagation. The tree has 23 nodes and the unique solution is found after visiting 19 nodes
- The second representation has a tree with 29 nodes and the unique solution is found after visiting 23 nodes

However for the puzzle GERALD + DONALD = ROBERT the situation is reverse. The first has 16651 nodes and 13795 visits while the second has 869 nodes and 791 visits

⇝ Finding the best model is an empirical science

Guidelines

Rules of thumbs for modelling (to take with a grain of salt):

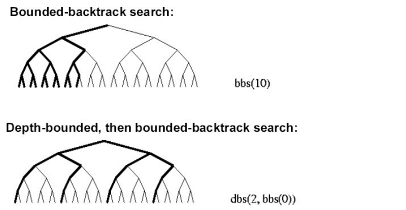
- use representations that involve less variables and simpler constraints for which constraint propagators are readily available
- use constraint propagation techniques that require less preprocessing (ie, the introduction of auxiliary variables) since they reduce the search space better.

  Disjunctive constraints may lead to an inefficient representation since they can generate a large search space.
- use global constraints (see below)

Randomization in Search Tree

- Dynamical selection of solution components in construction or choice points in backtracking.
- Randomization of construction method or selection of choice points in backtracking while still maintaining the method complete ⇝ *randomized systematic search.*
- Randomization can also be used in incomplete search

Incomplete Search

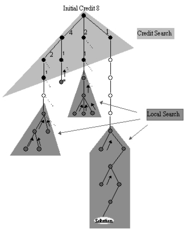

Bounded-backtrack search:
bbs(10)
Depth-bounded, then bounded-backtrack search:
dbs(2, bbs(0))

http://4c.ucc.ie/~hsimonis/visualization/techniques/partial_search/main.htm

Incomplete Search

Credit-based search

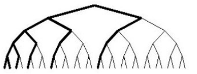
- Key idea: important decisions are at the top of the tree
- Credit = backtracking steps
- Credit distribution: one half at the best child the other divided among the other children.
- When credits run out follow deterministic best-search
- In addition: allow limited backtracking steps (eg, 5) at the bottom
- Control parameters: initial credit, distribution of credit among the children, amount of local backtracking at bottom.



Incomplete Search

Limited Discrepancy Search (LDS)

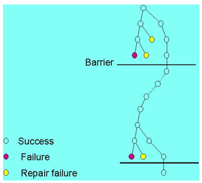
- Key observation that often the heuristic used in the search is nearly always correct with just a few exceptions.
- Explore the tree in increasing number of discrepancies, modifications from the heuristic choice.
- Eg: count one discrepancy if second best is chosen count two discrepancies either if third best is chosen or twice the second best is chosen
- Control parameter: the number of discrepancies



Incomplete Search

Barrier Search

- Extension of LDS
- Key idea: we may encounter several, independent problems in our heuristic choice. Each of these problems can be overcome locally with a limited amount of backtracking.
- At each barrier start LDS-based backtracking



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

Handling special constraints

Higher order constraints

Definition

Global constraints are complex constraints that are taken care of by means of a special purpose algorithm.

Modelling by means of global constraints is more efficient than relying on the general purpose constraint propagator.

Examples:

- alldiff
  - for m variables and n values cannot be satisfied if $m > n$,
  - consider first singleton variables
  - propagation based on bipartite matching considerations

- cumulative for RCPSP [Aggoun and Beldiceanu, 1993]
  - S_j starting times of jobs
  - P_j duration of job
  - R_j resource consumption
  - R limit not to be exceeded at any point in time

$$\texttt{cumulative}([S_j],[P_j],[R_j],R) := \{([s_j],[p_j],[r_j]R) \mid \forall t \sum_{i \mid s_i \le t \le s_i + p_i} r_i \le R\}$$

The special purpose algorithm employes the edge-finding technique (enforce precedences)

- atmost Resource Constraint
  - 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 $\texttt{Flight272} \in [255, 385]$

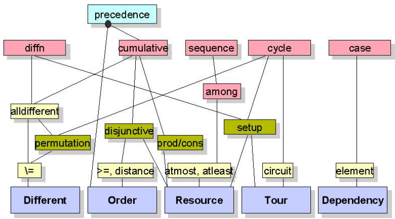
- sortedness for job shop [Older, Swinkels, and van Emden, 1995]

$$\texttt{sortedness}([X_1, \ldots, X_n], [Y_1, \ldots, Yn]) := \{([d_1, \ldots, d_n], [e_1, \ldots, e_n]) \mid [e_1, \ldots, e_n] \text{ is the sorted permutation of } [d_1, \ldots, d_n]\}$$

- $\texttt{among}(x|v, l, u)$ at least l and at most v variables take values in the set v.
- $\texttt{bin-packing}(x|w, u, k)$ pack items in k bins such that they do not exceed capacity u
- $\texttt{cardinality}(x|v, l, u)$ at least l_j and at most u_j of the variables take the value v_j
- $\texttt{cardinality-clause}(x|k)$ $\sum_{j=1}^{n} x_j \ge k$
- $\texttt{cardinality-conditional}(x, y|k, l)$ if $\sum_{j=1}^{n} x_j \ge k$ then $\sum_{j=1}^{m} y_j \ge l$
- $\texttt{change}(x|k, \text{rel})$ counts number of times a given change occur

- $\texttt{circuit}(x)$ imposes Hamiltonian cycle on digraph.
- $\texttt{clique}(x|G, k)$ requires that a given graph contain a clique
- $\texttt{conditional}(\mathcal{D}, \mathcal{C})$ between set of constrains $\mathcal{D} \Rightarrow \mathcal{C}$
- $\texttt{cutset}(x|G, k)$ requires that for the set of selected vertices V', the set $V \setminus V'$ induces a subgraph of G that contains no cycles.
- $\texttt{cycle}(x|y)$ select edges such that they form exactly y cycles. directed cycles in a graph.
- $\texttt{diffn}((x^1, \Delta x^1), \ldots, (x^m, \Delta x^m))$ arranges a given set of multidimensional boxes in n-space such that they do not overlap
- ...

Constraint Morphology



Kinds of symmetries

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

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?
  $$A + B = C \quad \vee \quad C = 0$$
  or soft constraints?
- Solution: reify the constraints:
  $$\begin{array}{lcll} (A + B = C & \Leftrightarrow & b_0) & \wedge \\ (C = 0 & \Leftrightarrow & b_1) & \wedge \\ (b_0 \quad \vee \quad b_1 & \Leftrightarrow & true) & \end{array}$$
- These kind of constraints are dealt with in efficient way by the systems
- Then if optimization problem (soft constraints) $\Rightarrow \min \sum_i b_i$

Outline

1. Refinements on CP
   Refinements: Modeling
   Refinements: Search
   Refinements: Constraints
   Symmetry Breaking
   Reification
2. Language and Systems

Prolog Approach

- Prolog II till Prolog IV [Colmerauer, 1990]
- CHIP V5 [Dincbas, 1988] `http://www.cosytec.com` (commercial)
- CLP [Van Hentenryck, 1989]
- Ciao Prolog (Free, GPL)
- GNU Prolog (Free, GPL)
- SICStus Prolog
- ECL^{i}PS^e[Wallace, Novello, Schimpf, 1997] `http://eclipse-clp.org/` (Open Source)
- Mozart programming system based on Oz language (incorporates concurrent constraint programming) `http://www.mozart-oz.org/` [Smolka, 1995]

Example

The puzzle `SEND+MORE = MONEY` in ECL^{i}PS^e

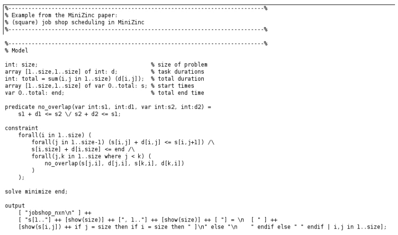
```
:- lib(ic).

sendmore(Digits) :-
    Digits = [S,E,N,D,M,O,R,Y],

% Assign a finite domain with each letter - S, E, N, D, M, O, R, Y -
% in the list Digits
    Digits :: [0..9],

% Constraints
    alldifferent(Digits),
    S #\= 0,
    M #\= 0,
                 1000*S + 100*E + 10*N + D
               + 1000*M + 100*O + 10*R + E
    #= 10000*M + 1000*O + 100*N + 10*E + Y,

% Search
    labeling(Digits).
```

Other Approaches

Modelling languages similar in concept to ZIMPL:

- OPL [Van Hentenryck, 1999] ILOG CP Optimizer `www.cpoptimizer.ilog.com` (ILOG, commercial)
- MiniZinc [] (open source, works for various systems, ECL^{i}PS^e, Geocode)

MiniZinc



```
%-----------------------------------------------------------------------------%
% Example from the MiniZinc paper:
% (square) job shop scheduling in MiniZinc
%-----------------------------------------------------------------------------%

%-----------------------------------------------------------------------------%
% Model

int: size;                                  % size of problem
array [1..size,1..size] of int: d;          % task durations
int: total = sum(i,j in 1..size) (d[i,j]);  % total duration
array [1..size,1..size] of var 0..total: s; % start times
var 0..total: end;                          % total end time

predicate no_overlap(var int:s1, int:d1, var int:s2, int:d2) =
    s1 + d1 <= s2 \/ s2 + d2 <= s1;

constraint
    forall(i in 1..size) (
        forall(j in 1..size-1) (s[i,j] + d[i,j] <= s[i,j+1]) /\
        s[i,size] + d[i,size] <= end /\
        forall(j,k in 1..size where j < k) (
            no_overlap(s[j,i], d[j,i], s[k,i], d[k,i])
        )
    );

solve minimize end;

output
    [ "jobshop_nxn\n" ] ++
    [ "s[1..", ++ [show(size)] ++ [", 1.."] ++ [show(size)] ++ [ "] = \n  [ " ] ++
    [show(s[i,j]) ++ if j = size then if i = size then " ]\n" else "\n    " endif else " " endif | i,j in 1..size];
```

Other Approaches

Libraries:
Constraints are modelled as objects and are manipulated by means of special methods provided by the given class.

- CHOCO (free) http://choco.sourceforge.net/
- Kaolog (commercial) http://www.koalog.com/php/index.php
- ECLiPSe (free) `www.eclipse-clp.org`
- ILOG CP Optimizer `www.cpoptimizer.ilog.com` (ILOG, commercial)
- Gecode (free) `www.gecode.org` C++, Programming interfaces Java and MiniZinc
- G12 Project
  `http://www.nicta.com.au/research/projects/constraint_programming_platform`

CP Languages

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 Languages

- 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

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 8

Constraint Programming (3)

Marco Chiarandini

Outline

Handling special constraints

Higher order constraints

Definition

Global constraints are complex constraints that are taken care of by means of a special purpose algorithm.

Modelling by means of global constraints is more efficient than relying on the general purpose constraint propagator.

Examples:

- alldiff
  - for m variables and n values cannot be satisfied if $m > n$,
  - consider first singleton variables
  - propagation based on bipartite matching considerations

- $\texttt{disjunctive}(s \,|\, p)$ $(s_i + p_i \leq s_j) \vee (s_j + p_j \leq s_i)$
- $\texttt{cumulative}(s \,|\, p, r, R)$ for RCPSP [Aggoun and Beldiceanu, 1993]
  - s_j starting times of jobs
  - p_j duration of job
  - r_j resource consumption
  - R limit not to be exceeded at any point in time

$$\texttt{cumulative}(s \,|\, p, r, R) := \{([s_j], [p_j], [r_j], R) \mid \forall t \sum_{i \,|\, s_i \leq t \leq s_i + p_i} r_i \leq R\}$$

edge-finding, not-first not-last rules

- sortedness for job shop [Older, Swinkels, and van Emden, 1995]

$$\texttt{sortedness}([X_1, \ldots, X_n], [Y_1, \ldots, Yn]) := \{([d_1, \ldots, d_n], [e_1, \ldots, e_n]) \mid [e_1, \ldots, e_n] \text{ is the sorted permutation of } [d_1, \ldots, d_n]\}$$

- $\texttt{atmost}(x|v, k)$
  - At most k variables of the x VARIABLES collection are assigned to value v.
    (1,<4,2,4,5>,2)
    The atmost constraint holds since at most 1 value of the collection <4,2,4,5> is equal to value 2.
  - $\texttt{among}(x|v, l, u)$ at least l and at most u variables take values in the set v.
  - $\texttt{nvalues}(x \,|\, l, u)$ requires that the variables x take at least l and at most u different values.

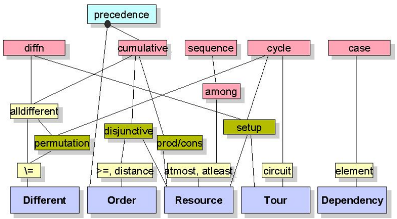
- $\texttt{bin-packing}(x|w, u, k)$ pack items in k bins such that they do not exceed capacity u
- $\texttt{cardinality}(x|v, l, u)$ at least l_j and at most u_j of the variables take the value v_j
- $\texttt{cardinality-clause}(x|k)$ $\sum_{j=1}^{n} x_j \geq k$
- $\texttt{cardinality-conditional}(x, y|k, l)$ if $\sum_{j=1}^{n} x_j \geq k$ then $\sum_{j=1}^{m} y_j \geq l$
- $\texttt{change}(x|k, \text{rel})$ counts number of times a given change occur

- $\texttt{circuit}(x)$ imposes Hamiltonian cycle on digraph.
- $\texttt{clique}(x|G, k)$ requires that a given graph contain a clique
- $\texttt{conditional}(\mathcal{D}, \mathcal{C})$ between set of constrains $\mathcal{D} \Rightarrow \mathcal{C}$
- $\texttt{cutset}(x|G, k)$ requires that for the set of selected vertices V', the set $V \setminus V'$ induces a subgraph of G that contains no cycles.
- $\texttt{cycle}(x|y)$ select edges such that they form exactly y cycles. directed cycles in a graph.
- $\texttt{diffn}((x^1, \Delta x^1), \ldots, (x^m, \Delta x^m))$ arranges a given set of multidimensional boxes in n-space such that they do not overlap

- $\texttt{element}(y, z \,|\, a)$ requires z to take the yth value in the tuple a. Useful with variable indices (variable subscripts), eg, a_y

  (3,2 | <6,9,2,9>)
  The element constraint holds since its third argument VALUE=2 is equal to the 3th (INDEX=3) item of the collection <6,9,2,9>

Constraint Morphology



Modelling in Gecode/J

- Implement model as a script
  - declare variables
  - post constraints (create propagators)
  - define branching
- Solve script
  - basic search strategy (DFS)
  - interactive, graphical search tool (Gist)

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 9
Heuristics

Marco Chiarandini

Outline

1. Construction Heuristics
   - General Principles
   - Metaheuristics
     - A* search
     - Rollout
     - Beam Search
     - Iterated Greedy
     - GRASP
2. Local Search
   - Beyond Local Optima
   - Search Space Properties
   - Neighborhood Representations
   - Distances
   - Efficient Local Search
     - Efficiency vs Effectiveness
     - Application Examples
   - Metaheuristics
     - Tabu Search
     - Iterated Local Search
3. Software Tools
   - The Code Delivered
   - Practical Exercise

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.

Outline

1. Construction Heuristics
   - General Principles
   - Metaheuristics
     - A* search
     - Rollout
     - Beam Search
     - Iterated Greedy
     - GRASP
2. Local Search
   - Beyond Local Optima
   - Search Space Properties
   - Neighborhood Representations
   - Distances
   - Efficient Local Search
     - Efficiency vs Effectiveness
     - Application Examples
   - Metaheuristics
     - Tabu Search
     - Iterated Local Search
3. Software Tools
   - The Code Delivered
   - Practical Exercise

Construction Heuristics

Heuristic: a common-sense rule (or set of rules) intended to increase the probability of solving some problem

Construction heuristics

(aka, single pass heuristics, dispatching rules, in scheduling)
They are closely related to tree search techniques but 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 do
    choose a solution component $c$
    add the solution component to $s$
```

Greedy best-first search

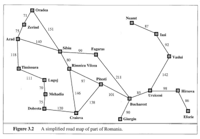


Figure 3.2 A simplified road map of part of Romania.

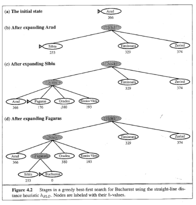


Figure 4.2 Stages in a greedy best-first search for Bucharest using the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

- Sometimes greedy heuristics can be proved to be optimal (Minimum Spanning Tree, Single Source Shortest Path, $1||\sum w_jC_j$, $1||L_{max}$)
- Other times an approximation ratio can be prooved

Designing heuristics

- Same idea of (variable, value) selection in CP without backtracking

Variable

```
* INT_VAR_NONE: First unassigned

* INT_VAR_MIN_MIN: With smallest min
* INT_VAR_MIN_MAX: With largest min
* INT_VAR_MAX_MIN: With smallest max
* INT_VAR_MAX_MAX: With largest max

* INT_VAR_SIZE_MIN: With smallest domain size
* INT_VAR_SIZE_MAX: With largest domain size

* INT_VAR_DEGREE_MIN: With smallest degree The degree of a variable is defined as the number of dependant
        propagators. In case of ties, choose the variable with smallest domain.
* INT_VAR_DEGREE_MAX: With largest degree The degree of a variable is defined as the number of dependant
       propagators. In case of ties, choose the variable with smallest domain.
* INT_VAR_SIZE_DEGREE_MIN: With smallest domain size divided by degree
* INT_VAR_SIZE_DEGREE_MAX: With largest domain size divided by degree

* INT_VAR_REGRET_MIN_MIN: With smallest min-regret The min-regret of a variable is the difference between
        the smallest and second-smallest value still in the domain.
* INT_VAR_REGRET_MIN_MAX: With largest min-regret The min-regret of a variable is the difference between
       the smallest and second-smallest value still in the domain.
* INT_VAR_REGRET_MAX_MIN: With smallest max-regret The max-regret of a variable is the difference between
        the largest and second-largest value still in the domain.
* INT_VAR_REGRET_MAX_MAX: With largest max-regret The max-regret of a variable is the difference between
       the largest and second-largest
value still in the domain.
```

Designing heuristics

- Same idea of (variable, value) selection in CP without backtracking

Value

```
* INT_VAL_MIN: Select smallest value
* INT_VAL_MED: Select median value
* INT_VAL_MAX: Select maximal value

* INT_VAL_SPLIT_MIN: Select lower half of domain
* INT_VAL_SPLIT_MAX: Select upper half of domain
```

- Static vs Dynamic (➨ quality time tradeoff)

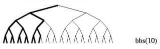
Dispatching Rules in Scheduling

	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

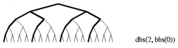
Truncated Search

They can be seen as form of Metaheuristics

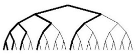
Bounded-backtrack search:



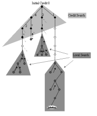
Depth-bounded, then bounded-backtrack search:



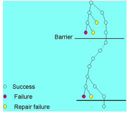
Limited Discrepancy Search (LDS)



Credit-based search



Barrier Search



A* best-first search

- The priority assigned to a node x is determined by the function
$$f(x) = g(x) + h(x)$$
  $g(x)$: cost of the path so far
  $h(x)$: heuristic estimate of the minimal cost to reach the goal from x.
- It is optimal if $h(x)$ is an
  - admissible heuristic: *never overestimates* the cost to reach the goal
  - consistent: $h(n) \leq c(n, a, n') + h(n')$

A* best-first search

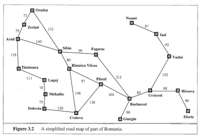


Figure 3.2 A simplified road map of part of Romania.

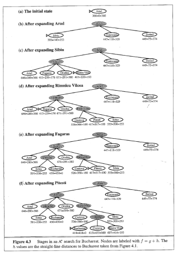

(a) The initial state
(b) After expanding Arad
(c) After expanding Sibiu
(d) After expanding Rimnicu Vilcea
(e) After expanding Fagaras
(f) After expanding Pitesti

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.

A* best-first search

Possible choices for admissible heuristic functions

- optimal solution to an easily solvable **relaxed problem**
- optimal solution to an easily solvable **subproblem**
- preferred heuristics functions with higher values (provided they do not overestimate)
- if several heuristics available $h_1, h_2, \ldots, h_m$ and not clear which is the best then:
$$h(x) = \max\{h_1(x), \ldots, h_m(x)\}$$

A* best-first search

Drawbacks

- Time complexity: In the worst case, the number of nodes expanded is exponential, but it is polynomial when the heuristic function h meets the following condition:
$$|h(x) - h^*(x)| \leq O(\log h^*(x))$$
  h^* is the optimal heuristic, the exact cost of getting from x to the goal.
- Memory usage: In the worst case, it must remember an exponential number of nodes.
  Several variants: including iterative deepening A* (IDA*), memory-bounded A* (MA*) and simplified memory bounded A* (SMA*) and recursive best-first search (RBFS)

Rollout Method

(aka, pilot method) [Bertsekas, Tsitsiklis, Cynara, JoH, 1997]
Derived from A*

- Each candidate solution is a collection of m components $S = (s_1, s_2, \ldots, s_m)$.
- Master process adds components sequentially to a partial solution $S_k = (s_1, s_2, \ldots s_k)$
- 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
  - minimal value

Rollout Method

Speed-ups:

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

Beam Search

[Lowerre, Complex System, 1976]

Derived from A* and branch and bound

- 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 $s := r$
```

GRASP

Greedy Randomized Adaptive Search Procedure (GRASP) []

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

Motivation:

- Candidate solutions obtained from construction heuristics can often be substantially improved by perturbative search.
- Perturbative search methods typically often require substantially fewer steps to reach high-quality solutions
when initialized using greedy constructive search rather than random picking.
- By iterating cycles of constructive + perturbative search, further performance improvements can be achieved.

Greedy Randomized "Adaptive" Search Procedure (GRASP):

```
While termination criterion is not satisfied:
    generate candidate solution $s$ using
        subsidiary greedy randomized constructive search
    perform subsidiary perturbative search on $s$
```

Note:

- Randomization in *constructive search* ensures that a large number of good starting points for *subsidiary perturbative search* is obtained.
- Constructive search in GRASP is 'adaptive' (or dynamic):
Heuristic value of solution component to be added to given partial candidate solution r may depend on solution components present in r.
- Variants of GRASP without perturbative search phase (aka *semi-greedy heuristics*) typically do not reach the performance of GRASP with perturbative search.

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 $h(l) \leq h_{min} + \alpha \cdot (h_{max} - h_{min})$,
where h_{min} = minimal value of h and h_{max} = maximal value of h for any l. (α is a parameter of the algorithm.)

Example: GRASP for SAT [Resende and Feo, 1996]

- **Given:** CNF formula F over variables $x_1, \dots, x_n$
- **Subsidiary constructive search:**
  - start from empty variable assignment
  - in each step, add one atomic assignment (*i.e.*, assignment of a truth value to a currently unassigned variable)
  - heuristic function $h(i, v)$:= number of clauses that become satisfied as a consequence of assigning $x_i := v$
  - RCLs based on cardinality restriction (contain fixed number k of atomic assignments with largest heuristic values)
- **Subsidiary perturbative search:**
  - iterative best improvement using 1-flip neighborhood
  - terminates when model has been found or given number of steps has been exceeded

GRASP has been applied to many combinatorial problems, including:

- SAT, MAX-SAT
- various scheduling problems

Extensions and improvements of GRASP:

- reactive GRASP (*e.g.*, dynamic adaptation of α during search)

Outline

1. Construction Heuristics
   General Principles
   Metaheuristics
   A* search
   Rollout
   Beam Search
   Iterated Greedy
   GRASP
2. Local Search
   Beyond Local Optima
   Search Space Properties
   Neighborhood Representations
   Distances
   Efficient Local Search
   Efficiency vs Effectiveness
   Application Examples
   Metaheuristics
   Tabu Search
   Iterated Local Search
3. Software Tools
   The Code Delivered
   Practical Exercise

Local Search Paradigm

- search space = complete candidate solutions
- search step = modification of one or more solution components
- iteratively generate and evaluate candidate solutions
  - decision problems: evaluation = test if solution
  - optimization problems: evaluation = check objective function value
- evaluating candidate solutions is typically computationally much cheaper than finding (optimal) solutions

Iterative Improvement (II):

```
determine initial candidate solution $s$
while $s$ has better neighbors do
    choose a neighbor $s'$ of $s$ such that $f(s') < f(s)$
    $s := s'$
```

Local Search Algorithm (1)

Given a (combinatorial) optimization problem Π and one of its instances π:

- search space $S(\pi)$
specified by candidate solution representation:
discrete structures: sequences, permutations, graphs, partitions
(*e.g.*, for SAT: array (sequence of all truth assignments to propositional variables)

  Note: solution set $S'(\pi) \subseteq S(\pi)$
(*e.g.*, for SAT: models of given formula)
- evaluation function $f(\pi) : S(\pi) \mapsto \mathbf{R}$
(*e.g.*, for SAT: number of false clauses)
- neighborhood function, $\mathcal{N}(\pi) : S \mapsto 2^{S(\pi)}$
(*e.g.*, for SAT: neighboring variable assignments differ in the truth value of exactly one variable)

Local Search Algorithm (2)

- set of memory states $M(\pi)$
(may consist of a single state, for LS algorithms that do not use memory)
- initialization function $\texttt{init} : \emptyset \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
(specifies probability distribution over initial search positions and memory states)
- step function $\texttt{step} : S(\pi) \times M(\pi) \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
(maps each search position and memory state onto probability distribution over subsequent, neighboring search positions and memory states)
- termination predicate $\texttt{terminate} : S(\pi) \times M(\pi) \mapsto \mathcal{P}(\{\top, \bot\})$
(determines the termination probability for each search position and memory state)

Local Search Algorithm

For given problem instance π:

- search space (solution representation) $S(\pi)$
- neighborhood relation $\mathcal{N}(\pi) \subseteq S(\pi) \times S(\pi)$
- evaluation function $f(\pi) : S \mapsto \mathbf{R}$
- set of memory states $M(\pi)$
- initialization function $\texttt{init} : \emptyset \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
- step function $\texttt{step} : S(\pi) \times M(\pi) \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
- termination predicate $\texttt{terminate} : S(\pi) \times M(\pi) \mapsto \mathcal{P}(\{\top, \bot\})$

LS Algorithm Components

Search Space

Defined by the solution representation:

- permutations
  - linear (scheduling)
  - circular (TSP)
- arrays (assignment problems: GCP)
- sets or lists (partition problems: Knapsack)

LS Algorithm Components

Neighborhood function $\mathcal{N}(\pi) : S(\pi) \mapsto 2^{S(\pi)}$

Also defined as: $\mathcal{N} : S \times S \to \{T, F\}$ or $\mathcal{N} \subseteq S \times S$

- neighborhood (set) of candidate solution s: $N(s) := \{s' \in S \mid \mathcal{N}(s, s')\}$
- neighborhood size is $|N(s)|$
- neighborhood is symmetric if: $s' \in N(s) \Rightarrow s \in N(s')$
- neighborhood graph of (S, N, π) is a directed vertex-weighted graph:
$G_{\mathcal{N}}(\pi) := (V, A)$ with $V = S(\pi)$ and $(uv) \in A \Leftrightarrow v \in N(u)$
(if symmetric neighborhood $\Rightarrow$ undirected graph)

Note on notation: N when set, $\mathcal{N}$ when collection of sets or function

A neighborhood function is also defined by means of an operator.

An operator Δ is a collection of operator functions $\delta : S \to S$ such that

$$s' \in N(s) \quad \Longleftrightarrow \quad \exists\, \delta \in \Delta, \delta(s) = s'$$

Definition

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

Examples:

- 1-exchange (flip) neighborhood for SAT
(solution components = single variable assignments)
- 2-exchange neighborhood for TSP
(solution components = edges in given graph)

LS Algorithm Components

Note:

- Local search implements a walk through the neighborhood graph
- Procedural versions of `init`, `step` and `terminate` implement sampling from respective probability distributions.
- Memory state m can consist of multiple independent attributes, *i.e.*, $M(\pi) := M_1 \times M_2 \times \ldots \times M_{l(\pi)}$.
- Local search algorithms are Markov processes:
behavior in any search state $\{s, m\}$ depends only on current position s and (limited) memory m.

LS Algorithm Components

Search step (or move):
pair of search positions s, s' for which
s' can be reached from s in one step, *i.e.*, $\mathcal{N}(s, s')$ and
$\texttt{step}(\{s, m\}, \{s', m'\}) > 0$ for some memory states $m, m' \in M$.

- Search trajectory: finite sequence of search positions $< s_0, s_1, \dots, s_k >$ such that (s_{i-1}, s_i) is a *search step* for any $i \in \{1, \dots, k\}$ and the probability of initializing the search at s_0 is greater zero, *i.e.*, $\texttt{init}(\{s_0, m\}) > 0$ for some memory state $m \in M$.
- Search strategy: specified by `init` and `step` function; to some extent independent of problem instance and other components of LS algorithm.
  - random
  - based on evaluation function
  - based on memory

Uninformed Random Picking

- $\mathcal{N} := S \times S$
- does not use memory and evaluation function
- $\texttt{init}, \texttt{step}$: uniform random choice from S,
i.e., for all $s, s' \in S$, $\texttt{init}(s) := \texttt{step}(\{s\}, \{s'\}) := 1/|S|$

Uninformed Random Walk

- does not use memory and evaluation function
- `init`: uniform random choice from S
- `step`: uniform random choice from current neighborhood,
i.e., for all $s, s' \in S$, $\texttt{step}(\{s\}, \{s'\}) := \begin{cases} 1/|N(s)| & \text{if } s' \in N(s) \\ 0 & \text{otherwise} \end{cases}$

Note: These uninformed LS strategies are quite ineffective, but play a role in combination with more directed search strategies.

LS Algorithm Components

Evaluation (or cost) function:

- function $f(\pi) : S(\pi) \mapsto \mathbf{R}$ 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).

Iterative Improvement

- does not use memory
- `init`: uniform random choice from S
- `step`: uniform random choice from improving neighbors, *i.e.*, $\texttt{step}(\{s\},\{s'\}) := 1/|I(s)|$ if $s' \in I(s)$, and 0 otherwise, where $I(s) := \{s' \in S \mid \mathcal{N}(s,s') \text{ and } f(s') < f(s)\}$
- terminates when no improving neighbor available (to be revisited later)
- different variants through modifications of step function (to be revisited later)

Note: II is also known as *iterative descent* or *hill-climbing*.

Example: Iterative Improvement for SAT

- **search space** S: set of all truth assignments to variables in given formula F (**solution set** S': set of all models of F)
- **neighborhood function** $\mathcal{N}$: 1-flip neighborhood (as in Uninformed Random Walk for SAT)
- **memory:** not used, *i.e.*, $M := \{0\}$
- **initialization:** uniform random choice from S, *i.e.*, $\texttt{init}(\emptyset, \{a'\}) := 1/|S|$ for all assignments a'
- **evaluation function:** $f(a) :=$ number of clauses in F that are *unsatisfied* under assignment a (*Note:* $f(a) = 0$ iff a is a model of F.)
- **step function**: uniform random choice from improving neighbors, *i.e.*, $\texttt{step}(a, a') := 1/\#I(a)$ if $s' \in I(a)$, and 0 otherwise, where $I(a) := \{a' \mid \mathcal{N}(a,a') \wedge f(a') < f(a)\}$
- **termination**: when no improving neighbor is available *i.e.*, $\texttt{terminate}(a, \top) := 1$ if $I(a) = \emptyset$, and 0 otherwise.

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 $f(s) \leq f(s')$ 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.

There might be more than one neighbor that have better cost.

Pivoting rule decides which to choose:

- Best Improvement (aka *gradient descent*, *steepest descent*, *greedy hill-climbing*): Choose maximally improving neighbor, *i.e.*, randomly select from $I^*(s) := \{s' \in N(s) \mid f(s') = f^*\}$, where $f^* := \min\{f(s') \mid s' \in N(s)\}$.

  Note: Requires evaluation of all neighbors in each step.

- First Improvement: Evaluate neighbors in fixed order, choose first improving step encountered.

  Note: Can be much more efficient than Best Improvement; order of evaluation can have significant impact on performance.

Example: Iterative Improvement for TSP (2-opt)

```
procedure TSP-2opt-first($s$)
    input: an initial candidate tour $s \in S(\in)$
    output: a local optimum $s \in S(\pi)$
    $\Delta = 0$;
    do
        Improvement=FALSE;
        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==FALSE;
end TSP-2opt-first
```

➤ Are we in a local optimum when it terminates?

A note on terminology

Heuristic Methods ≡ Metaheuristics ≡ Local Search Methods ≡ Stochastic Local Search Methods ≡ Hybrid Metaheuristics

Method $\neq$ Algorithm

Stochastic Local Search (SLS) algorithms allude to:

- Local Search: informed search based on *local* or incomplete knowledge as opposed to systematic search
- Stochastic: use *randomized choices* in generating and modifying candidate solutions. They are introduced whenever it is unknown which deterministic rules are profitable for all the instances of interest.

Escaping from Local Optima

- Enlarge the neighborhood
- Restart: re-initialize search whenever a local optimum is encountered. (Often rather ineffective due to cost of initialization.)
- Non-improving steps: in local optima, allow selection of candidate solutions with equal or worse evaluation function value, *e.g.*, using minimally worsening steps. (Can lead to long walks in *plateaus*, *i.e.*, regions of search positions with identical evaluation function.)

Note: None of these mechanisms is guaranteed to always escape effectively from local optima.

Diversification *vs* Intensification

- Goal-directed and randomized components of LS strategy need to be balanced carefully.
- Intensification: aims to greedily increase solution quality or probability, *e.g.*, by exploiting the evaluation function.
- Diversification: aim to prevent search stagnation by preventing search process from getting trapped in confined regions.

Examples:

- Iterative Improvement (II): *intensification* strategy.
- Uninformed Random Walk/Picking (URW/P): *diversification* strategy.

Balanced combination of intensification and diversification mechanisms forms the basis for advanced LS methods.

Learning goals of this section

- Review basic theoretical concepts
- Learn about techniques and goals of experimental search space analysis.
- Develop intuition on which features of local search are adequate to contrast a specific situation.

Definitions

- Search space S
- Neighborhood function $\mathcal{N} : S \subseteq 2^S$
- Evaluation function $f(\pi) : S \mapsto \mathbf{R}$
- Problem instance π

Definition:

The **search landscape** L is the vertex-labeled neighborhood graph given by the triplet $\mathcal{L} = (S(\pi), N(\pi), f(\pi))$.

Fundamental Search Space Properties

The behavior and performance of an LS algorithm on a given problem instance crucially depends on properties of the respective search space.

Simple properties of search space S:

- search space size $|S|$
- reachability: solution j is reachable from solution i if neighborhood graph has a path from i to j.
  - strongly connected neighborhood graph
  - weakly optimally connected neighborhood graph
- search space diameter $\texttt{diam}(G_\mathcal{N})$ (= maximal distance between any two candidate solutions) **Note:** Diameter of $G_\mathcal{N}$ = worst-case lower bound for number of search steps required for reaching (optimal) solutions. Maximal shortest path between any two vertices in the neighborhood graph.

Solution Representations and Neighborhoods

Three different types of solution representations:

- Permutation
  - linear permutation: Single Machine Total Weighted Tardiness Problem
  - circular permutation: Traveling Salesman Problem
- Assignment: Graph Coloring Problem, SAT, CSP
- Set, Partition: Knapsack, Max Independent Set

A neighborhood function $\mathcal{N} : S \to S \times S$ is also defined through an operator. An operator Δ is a collection of operator functions $\delta : S \to S$ such that

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

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 $1 \leq i \leq n$ 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

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

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

Interchange operator

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

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

(≡ set of all transpositions)

Insert operator

$$\Delta_I = \{\delta_I^{ij} | 1 \leq i \leq n, 1 \leq j \leq n, j \neq i\}$$

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

Neighborhood Operators for Circular Permutations

Reversal (2-edge-exchange)

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

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

Block moves (3-edge-exchange)

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

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

Short block move (Or-edge-exchange)

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

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

Neighborhood Operators for Assignments

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

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

One-exchange operator

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

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

Two-exchange operator

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

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

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

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

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

One-deletion operator

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

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

Swap operator

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

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

Distances

Set of paths in $G_\mathcal{N}$ with $s, s' \in S$:

$$\Phi(s, s') = \{(s_1, \ldots, s_h) | s_1 = s, s_h = s'\ \forall i : 1 \leq i \leq h - 1, \langle s_i, s_{i+1} \rangle \in E_N\}$$

If $\phi = (s_1, \ldots, s_h) \in \Phi(s, s')$ let $|\phi| = h$ be the length of the path; then the distance between any two solutions s, s' is the length of shortest path between s and s' in $G_\mathcal{N}$:

$$d_\mathcal{N}(s, s') = \min_{\phi \in \Phi(s,s')} |\Phi|$$

$$\texttt{diam}(G_\mathcal{N}) = \max\{d_\mathcal{N}(s, s') \mid s, s' \in S\}$$

Note: with permutations it is easy to see that:

$$d_\mathcal{N}(\pi, \pi') = d_\mathcal{N}(\pi^{-1} \cdot \pi', \iota)$$

Distances for Linear Permutation Representations

- Swap neighborhood operator
  computable in $O(n^2)$ by the precedence based distance metric:
  $d_S(\pi,\pi') = \#\{\langle i,j\rangle | 1 \le i < j \le n, pos_{\pi'}(\pi_j) < pos_{\pi'}(\pi_i)\}$.
  $\mathtt{diam}(G_{\mathcal{N}}) = n(n-1)/2$
- Interchange neighborhood operator
  Computable in $O(n) + O(n)$ since
  $d_X(\pi,\pi') = d_X(\pi^{-1}\cdot\pi',\iota) = n - c(\pi^{-1}\cdot\pi')$
  where $c(\pi)$ is the number of disjoint cycles that decompose a permutation.
  $\mathtt{diam}(G_{\mathcal{N}_X}) = n-1$
- Insert neighborhood operator
  Computable in $O(n) + O(n\,log(n))$ since
  $d_I(\pi,\pi') = d_I(\pi^{-1}\cdot\pi',\iota) = n - |lis(\pi^{-1}\cdot\pi')|$ where $lis(\pi)$ denotes the length of the longest increasing subsequence.
  $\mathtt{diam}(G_{\mathcal{N}_I}) = n-1$

Distances for Circular Permutation Representations

- Reversal neighborhood operator
  sorting by reversal is known to be NP-hard
  surrogate in TSP: bond distance
- Block moves neighborhood operator
  unknown whether it is NP-hard but there does not exist a proved polynomial-time algorithm

Distances for Assignment Representations

- Hamming Distance
- An assignment can be seen as a partition of n in k mutually exclusive non-empty subsets

  One-exchange neighborhood operator
  The *partition-distance* $d_{1E}(\mathcal{P},\mathcal{P}')$ between two partitions $\mathcal{P}$ and $\mathcal{P}'$ is the minimum number of elements that must be moved between subsets in $\mathcal{P}$ so that the resulting partition equals $\mathcal{P}'$.

  The partition-distance can be computed in polynomial time by solving an assignment problem. Given the assignment matrix M where in each cell (i,j) it is $|S_i \cap S'_j|$ with $S_i \in \mathcal{P}$ and $S'_j \in \mathcal{P}'$ and defined $A(\mathcal{P},\mathcal{P}')$ the assignment of maximal sum then it is $d_{1E}(\mathcal{P},\mathcal{P}') = n - A(\mathcal{P},\mathcal{P}')$

Example: Search space size and diameter for the TSP

- Search space size $= (n-1)!/2$
- Insert neighborhood
  size $= (n-3)n$
  diameter $= n-2$
- 2-exchange neighborhood
  size $= \binom{n}{2} = n\cdot(n-1)/2$
  diameter in $[n/2, n-2]$
- 3-exchange neighborhood
  size $= \binom{n}{3} = n\cdot(n-1)\cdot(n-2)/6$
  diameter in $[n/3, n-1]$

Example: Search space size and diameter for SAT

SAT instance with n variables, 1-flip neighborhood:
$G_{\mathcal{N}} = n$-dimensional hypercube; diameter of $G_{\mathcal{N}} = n$.

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two different neighborhood functions for the same instance (S,f,π) of a combinatorial optimization problem.

If for all solutions $s \in S$ we have $N_1(s) \subseteq N_2(s')$ then we say that $\mathcal{N}_2$ dominates $\mathcal{N}_1$

Example:

In TSP, 1-insert is domnated by 3-exchange.
(1-insert corresponds to 3-exchange and there are 3-exchnages that are not 1-insert)

Efficiency vs Effectiveness

The performance of local search is determined by:

1. quality of local optima (effectiveness)
2. time to reach local optima (efficiency):
   A. time to move from one solution to the next
   B. number of solutions to reach local optima

Note:

- Local minima depend on g and neighborhood function $\mathcal{N}$.
- Larger neighborhoods $\mathcal{N}$ induce
  - neighborhood graphs with smaller diameter;
  - fewer local minima.

Ideal case: exact neighborhood, *i.e.*, neighborhood function for which any local optimum is also guaranteed to be a global optimum.

- Typically, exact neighborhoods are too large to be searched effectively (exponential in size of problem instance).
- *But:* exceptions exist, *e.g.*, polynomially searchable neighborhood in Simplex Algorithm for linear programming.

Trade-off (to be assessed experimentally):

- Using larger neighborhoods
  can improve performance of II (and other LS methods).
- *But:* time required for determining improving search steps increases with neighborhood size.

Speedups Techniques for Efficient Neighborhood Search

1) Incremental updates
2) Neighborhood pruning

Speedups in Neighborhood Examination

1) 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.
- Typically crucial for the efficient implementation of II algorithms (and other LS techniques).

Example: Incremental updates for TSP

- solution components = edges of given graph G
- standard 2-exchange neighborhood, *i.e.*, neighboring round trips p, p' differ in two edges
- $w(p') := w(p)$ − edges in p but not in p'
  + edges in p' but not in p

Note: Constant time (4 arithmetic operations), compared to linear time (n arithmetic operations for graph with n vertices) for computing $w(p')$ from scratch.

2) Neighborhood Pruning

- **Idea:** Reduce size of neighborhoods by excluding neighbors that are likely (or guaranteed) not to yield improvements in f.
- **Note:** Crucial for large neighborhoods, but can be also very useful for small neighborhoods (*e.g.*, linear in instance size).

Example: Heuristic candidate lists for the TSP

- *Intuition:* High-quality solutions likely include short edges.
- Candidate list of vertex v: list of v's nearest neighbors (limited number), sorted according to increasing edge weights.
- Search steps (*e.g.*, 2-exchange moves) always involve edges to elements of candidate lists.
- Significant impact on performance of LS algorithms for the TSP.

Overview

Delta evaluations and neighborhood examinations in:

- Permutations
  - TSP
  - SMTWTP
- Assignments
  - SAT
- Sets
  - Max Independent Set

Local Search for TSP

- k-exchange heuristics
  - 2-opt
  - 2.5-opt
  - Or-opt
  - 3-opt
- complex neighborhoods
  - Lin-Kernighan
  - Helsgaun's Lin-Kernighan
  - Dynasearch
  - ejection chains approach

Implementations exploit speed-up techniques

1 neighborhood pruning: fixed radius nearest neighborhood search
2 neighborhood lists: restrict exchanges to most interesting candidates
3 don't look bits: focus perturbative search to "interesting" part
4 sophisticated data structures

TSP data structures

Tour representation:

- determine pos of v in π
- determine succ and prec
- check whether u_k is visited between u_i and u_j
- execute a k-exchange (reversal)

Possible choices:

- $|V| < 1.000$ array for π and π^{-1}
- $|V| < 1.000.000$ two level tree
- $|V| > 1.000.000$ splay tree

Moreover static data structure:

- priority lists
- k-d trees

SMTWTP

- Interchange: size $\binom{n}{2}$ and $O(|i-j|)$ evaluation each
  - first-improvement: π_j, π_k
    $p_{\pi_j} \le p_{\pi_k}$ for improvements, $w_jT_j + w_kT_k$ must decrease because jobs in $\pi_j,\dots,\pi_k$ can only increase their tardiness.
    $p_{\pi_j} \ge p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
  - first-improvement: π_j, π_k
    $p_{\pi_j} \le p_{\pi_k}$ for improvements, $w_jT_j + w_kT_k$ must decrease at least as the best interchange found so far because jobs in $\pi_j,\dots,\pi_k$ can only increase their tardiness.
    $p_{\pi_j} \ge p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
- Swap: size $n-1$ and $O(1)$ evaluation each
- Insert: size $(n-1)^2$ and $O(|i-j|)$ evaluation each
  But possible to speed up with systematic examination by means of swaps: an interchange is equivalent to $|i-j|$ swaps hence overall examination takes $O(n^2)$

LS for GCP

- **search space** S: set of all k-colorings of G
- **solution set** S': set of all proper k-coloring of F
- **neighborhood function** $\mathcal{N}$: 1-exchange neighborhood (as in Uninformed Random Walk)
- **memory:** not used, *i.e.*, $M := \{0\}$
- **initialization:** uniform random choice from S, *i.e.*, $\mathtt{init}\{\emptyset,\varphi'\} := 1/|S|$ for all colorings φ'
- **step function**:
  - **evaluation function:** $g(\varphi) :=$ number of edges in G whose ending vertices are assigned the same color under assignment φ (*Note:* $g(\varphi) = 0$ iff φ is a proper coloring of G.)
  - **move mechanism:** uniform random choice from improving neighbors, *i.e.*, $\mathtt{step}\{\varphi,\varphi'\} := 1/|I(\varphi)|$ if $s' \in I(\varphi)$, and 0 otherwise, where $I(\varphi) := \{\varphi' \mid \mathcal{N}(\varphi,\varphi') \wedge g(\varphi') < g(\varphi)\}$
- **termination**: when no improving neighbor is available *i.e.*, $\mathtt{terminate}\{\varphi,\top\} := 1$ if $I(\varphi) = \emptyset$, and 0 otherwise.

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 neighbors 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 $s := r$
```

Outline

1. Construction Heuristics
   General Principles
   Metaheuristics
   A* search
   Rollout
   Beam Search
   Iterated Greedy
   GRASP
2. Local Search
   Beyond Local Optima
   Search Space Properties
   Neighborhood Representations
   Distances
   Efficient Local Search
   Efficiency vs Effectiveness
   Application Examples
   Metaheuristics
   Tabu Search
   Iterated Local Search
3. Software Tools
   The Code Delivered
   Practical Exercise

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:

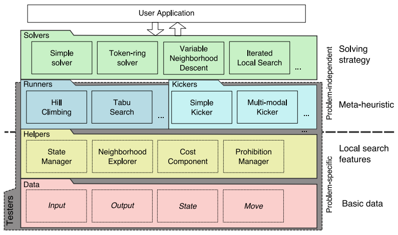
- the apparent simplicity of Local Search induces to build applications from scratch.
- crucial roles played by delta/incremental updates which is problem dependent
- the development of Local Search is in part a craft, beside engineering and science.
- lack of a unified view of Local Search.

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++

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 ()
```

- Implement two basic local search procedures that return a local optimum:

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

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

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 9
Heuristics

Marco Chiarandini

Outline

1. Ants
   Adaptive Iterated Construction Search

Outline

1. Ants
   Adaptive Iterated Construction Search

Adaptive Iterated Construction Search

Key Idea: Alternate construction and perturbative local search phases as in GRASP, exploiting experience gained during the search process.

Realisation:

- Associate *weights* with possible decisions made during constructive search.
- Initialize all weights to some small value τ_0 at beginning of search process.
- After every cycle (= constructive + perturbative local search phase), update weights based on solution quality and solution components of current candidate solution.

Adaptive Iterated Construction Search (AICS):

```
initialise weights
While termination criterion is not satisfied:
    generate candidate solution $s$ using
        subsidiary randomized constructive search
    perform subsidiary local search on $s$
    adapt weights based on $s$
```

Subsidiary constructive search:

- The solution component to be added in each step of *constructive search* is based on *weights* and heuristic function h.
- h can be standard heuristic function as, *e.g.*, used by greedy construction heuristics, GRASP or tree search.
- It is often useful to design solution component selection in constructive search such that any solution component may be chosen (at least with some small probability) irrespective of its weight and heuristic value.

Subsidiary perturbative local search:

- As in GRASP, perturbative local search phase is typically important for achieving good performance.
- Can be based on Iterative Improvement or more advanced LS method (the latter often results in better performance).
- Tradeoff between computation time used in construction phase *vs* local search phase (typically optimized empirically, depends on problem domain).

Weight updating mechanism:

- Typical mechanism: increase weights of all solution components contained in candidate solution obtained from local search.
- Can also use aspects of search history; *e.g.*, current *incumbent candidate solution* can be used as basis for weight update for additional intensification.

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

(Based on Ant System for the TSP [Dorigo et al., 1991])

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

$$[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta$$

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

- *Subsidiary local search* = iterative improvement based on standard 2-exchange neighborhood (until local minimum is reached).
- *Weight update* according to

  $$\tau_{ij} := (1 - \rho) \cdot \tau_{ij} + \Delta(i, j, s')$$

  where $\Delta(i, j, s') := 1/f(s')$, if edge (i, j) is contained in the cycle represented by s', and 0 otherwise.
- Criterion for weight increase is based on intuition that edges contained in short round trips should be preferably used in subsequent constructions.

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 10

Single Machine Models, Dynamic Programming

Marco Chiarandini

Outline

1. Dispatching Rules

2. Single Machine Models

Outline

1. Dispatching Rules

2. Single Machine Models

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)

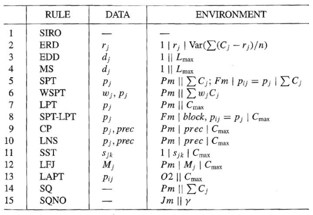
- (Weighted) shortest processing time first (WSPT)
  - $j^* = \arg\max_j \{w_j/pj\}$.
  - tends to min $\sum w_j C_j$ and max work in progress and
- Loongest processing time first (LPT)
  - balance work load over parallel machines
- Shortest setup time first (SST)
  - tends to min C_{max} and max throughput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

Dispatching Rules in Scheduling

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

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

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

  - $1 \,|\, s_{jk} |\, \sum w_j T_j$

$$(\theta_1,\, \theta_2 \text{ with estimated } \hat{C}_{max} = \sum_{j=1}^{n} p_j + n\bar{s})$$

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

- $1 \,|\,|\, \sum w_j T_j$, dynamic apparent tardiness cost (ATC)

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

- $1 \,|\, s_{jk} |\, \sum w_j T_j$, 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

Outlook 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

1. Dispatching Rules

2. Single Machine Models

Outlook

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

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

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

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

$1 \,|\,|\, \sum w_j T_j$: local search and dynasearch

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

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

$1 \,|\,|\, \sum w_j T_j$: column generation approaches

Multicriteria

Summary

Single Machine Models:

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

$1 \,|\,|\, \sum w_j C_j$

[Total weighted completion time]

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

Extensions to $1 \,|\, prec \,|\, \sum w_j C_j$

- 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.
- polytime algorithm also for tree and sp-graph precedences

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

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

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

[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

Dynamic programming

Procedure based on divide and conquer

Principle of optimality the completion of an optimal sequence of decisions must be optimal

- Break down the problem into 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, backward procedure feasible only if the makespan is schedule, eg, single machine problems without setups, multiple machines problems with identical processing times.)

$1\,|\,prec|\; h_{max}$

- $h_{max} = \max\{h_1(C_1), h_2(C_2), \dots, h_n(C_n)\}$, h_j regular
- special case: $1\,|\,prec|\; h_{max}$ [maximum lateness]
- solved by backward dynamic programming in $O(n^2)$ [Lawler, 1978]

  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 $J = \emptyset$, $J^c = \{1, \dots, n\}$ 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 $1\,|\,|\,L_{max}$ Earliest Due Date first
- $1|r_j|L_{max}$ is instead strongly NP-hard

$1\,|\,|\,\sum h_j(C_j)$

- generalization of $\sum w_j T_j$ hence strongly NP-hard
- (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 $J = \emptyset$, $V(j) = h_j(p_j)$, $\quad j = 1, \dots, n$

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

$1\,|\,|\,\sum h_j(C_j)$

A lot of work done on $1\,|\,|\,\sum w_j T_j$
[single-machine total weighted tardiness]

- $1\,|\,|\,\sum T_j$ is hard in ordinary sense, hence admits a pseudo polynomial algorithm (dynamic programming in $O(n^4 \sum p_j)$)
- $1\,|\,|\,\sum w_j T_j$ strongly NP-hard (reduction from 3-partition)
  - 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]

$1\,|\,|\,\sum h_j(C_j)$

Local search

- Interchange: size $\binom{n}{2}$ and $O(|i - j|)$ evaluation each
  - first-improvement: π_j, π_k

    $p_{\pi_j} \le p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease because jobs in $\pi_j, \dots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \ge p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
  - best-improvement: π_j, π_k

    $p_{\pi_j} \le p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease at least as the best interchange found so far because jobs in $\pi_j, \dots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \ge p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
- Swap: size $n - 1$ and $O(1)$ evaluation each
- Insert: size $(n-1)^2$ and $O(|i - j|)$ evaluation each
  But possible to speed up with systematic examination by means of swaps: an interchange is equivalent to $|i - j|$ swaps hence overall examination takes $O(n^2)$

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 $O(n^3)$ searched by dynamic programming;
- it yields in average better results than the interchange neighborhood alone.

- 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 $k = 1, \dots, h_t$ then $F(\pi_k^{(t-1)}) = F(\pi_k^{(t-2)})$ for $k = 1, \dots, h_t$ 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 $O(n^2)$.

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

[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($i$), $i = 1, \dots, n_k$, 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
```

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 11

Single Machine Models, Branch and Bound

Marco Chiarandini

Outline

1. Single Machine Models
   - Branch and Bound
   - $1\,|\,s_{jk}\,|\,C_{max}$

Outline

1. Single Machine Models
   - Branch and Bound
   - $1\,|\,s_{jk}\,|\,C_{max}$

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

[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
```

Branch and Bound

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

- Eager Strategy:
  1. select a node
  2. branch
  3. for each subproblem compute bounds and compare with incumbent solution
  4. discard or store nodes together with their bounds

  (Bounds are calculated as soon as nodes are available)
- Lazy Strategy:
  1. select a node
  2. compute bound
  3. branch
  4. store the new nodes together with the bound of the processed node

(often used when selection criterion for next node is max depth)

Components

- Initial feasible solution (heuristic) – might be crucial!
1. Bounding function
2. Strategy for selecting
3. Branching
- Fathmoing (dominance test)

Bounding

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

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

- relaxation: $\min_{s \in P} f(s)$
- solve (to optimality) in P but with g
- Lagrangian relaxation combines the two
- should be polytime and strong (trade off)

Strategy for selecting next subproblem

- best first
  (combined with eager strategy but also with lazy)
- 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)

Branching

- dichotomic
- polytomic

Overall guidelines

- finding good initial solutions is important
- if initial solution is close to optimum then the selection strategy makes little difference
- Parallel B&B: distributed control or a combination are better than centralized control
- parallelization might be used also to compute bounds if few nodes alive
- parallelization with static work load distribution is appealing with large search trees

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

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

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

$$\min \sum_{j=1}^{n} \sum_{t=1}^{C_{max}} c_{jt} x_{jt}$$

$$\text{s.t.} \sum_{t=1}^{C_{max}} x_{jt} = p_j, \qquad \forall j = 1, \ldots, n$$

$$\sum_{j=1}^{n} x_{jt} \leq 1, \qquad \forall t = 1, \ldots, C_{max}$$

$$x_{jt} \geq 0 \qquad \forall j = 1, \ldots, n; \ t = 1, \ldots, C_{max}$$

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

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-1} c_{jt} y_{jt}$$

s.t.

$$\sum_{t=1}^{T-p_j} c_{jt} \leq h_j(t + p_j)$$

$$\sum_{t=1}^{T-p_j} y_{jt} = 1, \qquad \forall j = 1, \ldots, n$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} y_{jt} \leq 1, \qquad \forall t = 1, \ldots, C_{max}$$

$$y_{jt} \geq 0 \qquad \forall j = 1, \ldots, n; \ t = 1, \ldots, C_{max}$$

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

[Makespan with sequence-dependent setup times]

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

  parameters for job j:
  - a_j initial state
  - b_j final state

  such that:

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

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

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

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

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

- Interchange δ^{jk}

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

- Cost

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

- **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',

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

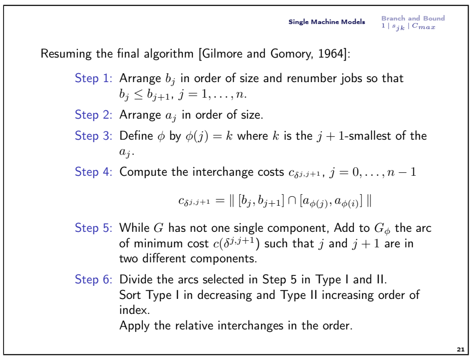
  is a tour.
- The $p-1$ interchanges can be found by greedy algorithm (similarity to Kruskal for min spanning tree)
- **Lemma:** There is a minimum spanning tree in G_{ϕ^*} that contains only arcs $\delta^{j,j+1}$.
- Generally, $c(\phi') \neq c(\delta^{j_1 k_1}) + c(\delta^{j_2 k_2}) + \ldots + c(\delta^{j_p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

-

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

$$\text{interchange } jk \text{ is of} \begin{cases} \text{Type I,} & \text{if lower node of type I} \\ \text{Type II,} & \text{if lower node of type II} \end{cases}$$

- 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.

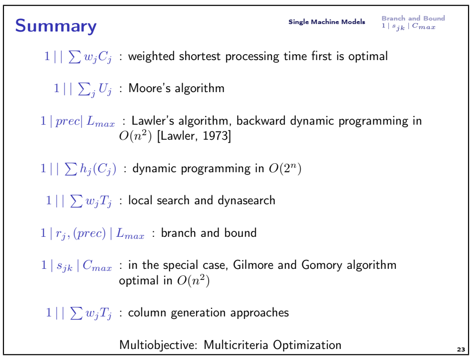
Step 3: Define ϕ by $\phi(j) = k$ where k is the $j+1$-smallest of the a_j.

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

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

$1 \mid \mid \sum_j U_j$: Moore's algorithm

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

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

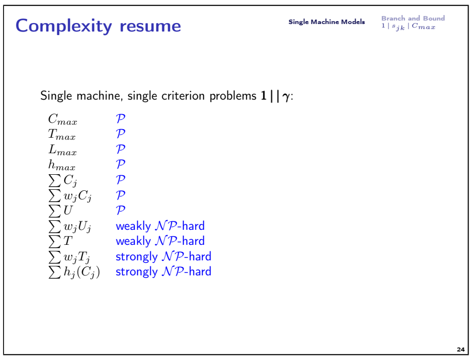
$1 \mid \mid \sum w_j T_j$: local search and dynasearch

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

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

$1 \mid \mid \sum w_j T_j$: column generation approaches

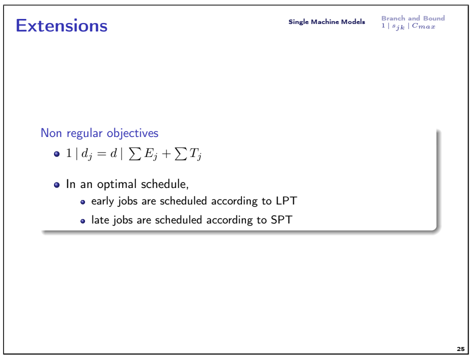
Multiobjective: Multicriteria Optimization



Complexity resume

Single machine, single criterion problems **1 | | γ**:

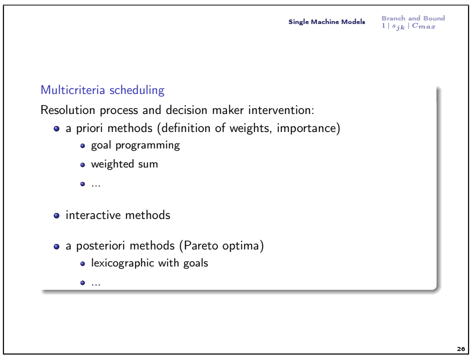
C_{max}	$\mathcal{P}$
T_{max}	$\mathcal{P}$
L_{max}	$\mathcal{P}$
h_{max}	$\mathcal{P}$
$\sum C_j$	$\mathcal{P}$
$\sum w_j C_j$	$\mathcal{P}$
$\sum U$	$\mathcal{P}$
$\sum w_j U_j$	weakly $\mathcal{NP}$-hard
$\sum T$	weakly $\mathcal{NP}$-hard
$\sum w_j T_j$	strongly $\mathcal{NP}$-hard
$\sum h_j(C_j)$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



Extensions

Non regular objectives

- $1 \mid d_j = d \mid \sum E_j + \sum T_j$
- 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
  - ...

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 12

Single Machine Models, Column Generation

Marco Chiarandini
Slides from David Pisinger's lectures at DIKU

Outline

1. Lagrangian Relaxation
2. Dantzig-Wolfe Decomposition
   Dantzig-Wolfe Decomposition
   Delayed Column Generation
3. Single Machine Models

Outline

1. Lagrangian Relaxation
2. Dantzig-Wolfe Decomposition
   Dantzig-Wolfe Decomposition
   Delayed Column Generation
3. Single Machine Models

Relaxation

In branch and bound we find upper bounds by relaxing the problem

Relaxation

$$\max_{s \in P} g(s) \geq \left\{ \begin{array}{c} \max_{s\in P} f(s) \\ \max_{s\in S} g(s) \end{array} \right\} \geq \max_{s \in S} f(s)$$

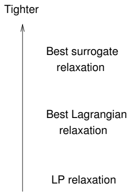
- P: candidate solutions;
- $S \subseteq P$ feasible solutions;
- $g(x) \geq f(x)$

Which constraints should be relaxed?

- Quality of bound (tightness of relaxation)
- Remaining problem can be solved efficiently
- Proper multipliers can be found efficiently
- Constraints difficult to formulate mathematically
- Constraints which are too expensive to write up

Different relaxations

- LP-relaxation
- Deleting constraint
- Lagrange relaxation
- Surrogate relaxation
- Semidefinite relaxation



Relaxations are often used in combination.

Tightness of relaxation

$$\begin{aligned} &\max cx \\ &\text{s.t. } Ax \leq b \\ &\quad Dx \leq d \\ &\quad x \in \mathbb{Z}^n_+ \end{aligned}$$

LP-relaxation:

$$\max \{cx : x \in \text{conv}(Ax \leq b, Dx \leq d, x \in \mathbb{Z}_+)\}$$

$\rightsquigarrow$ Lagrangian Relaxation:

$$\begin{aligned} &\max z_{LR}(\lambda) = cx - \lambda(Dx - d) \\ &\text{s.t. } Ax \leq b \\ &\quad x \in \mathbb{Z}^n_+ \end{aligned}$$

LP-relaxation:

$$\max \{cx : Dx \leq d, x \in \text{conv}(Ax \leq b, x \in \mathbb{Z}_+)\}$$

Relaxation strategies

Which constraints should be relaxed

- "the complicating ones"
- remaining problem is polynomially solvable
  (e.g. min spanning tree, assignment problem, linear programming)
- remaining problem is totally unimodular
  (e.g. network problems)
- remaining problem is NP-hard but good techniques exist
  (e.g. knapsack)
- constraints which cannot be expressed in MIP terms
  (e.g. cutting)
- constraints which are too extensive to express
  (e.g. subtour elimination in TSP)

Subgradient optimization Lagrange multipliers

$$\begin{aligned} &\max\ z = cx \\ &\text{s. t. } Ax \leq b \\ &\quad Dx \leq d \\ &\quad x \in \mathbb{Z}^n_+ \end{aligned}$$

Lagrange Relaxation, multipliers $\lambda \geq 0$

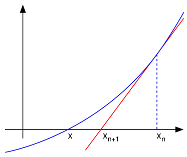
$$\begin{aligned} &\max\ z_{LR}(\lambda) = cx - \lambda(Dx - d) \\ &\text{s. t. } Ax \leq b \\ &\quad x \in \mathbb{Z}^n_+ \end{aligned}$$

Lagrange Dual Problem

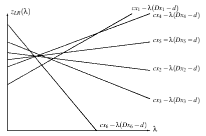
$$z_{LD} = \min_{\lambda \geq 0} z_{LR}(\lambda)$$

- We do not need best multipliers in B&B algorithm
- Subgradient optimization fast method
- Works well due to convexity
- Roots in nonlinear programming, Held and Karp (1971)

Subgradient optimization, motivation



Netwon-like method to minimize a function in one variable



Lagrange function $z_{LR}(\lambda)$ is piecewise linear and convex

Subgradient
Generalization of gradients to non-differentiable functions.

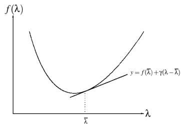
Definition
An m-vector γ is subgradient of $f(\lambda)$ at $\lambda = \bar{\lambda}$ if

$$f(\lambda) \geq f(\bar{\lambda}) + \gamma(\lambda - \bar{\lambda})$$

The inequality says that the hyperplane

$$y = f(\bar{\lambda}) + \gamma(\lambda - \bar{\lambda})$$

is tangent to $y = f(\lambda)$ at $\lambda = \bar{\lambda}$ and supports $f(\lambda)$ from below



Proposition Given a choice of nonnegative multipliers $\bar{\lambda}$. If x' is an optimal solution to $z_{LR}(\lambda)$ then

$$\gamma = d - Dx'$$

is a subgradient of $z_{LR}(\lambda)$ at $\lambda = \bar{\lambda}$.

Proof We wish to prove that from the subgradient definition:

$$\max_{Ax \leq b}\left(cx = \lambda(Dx - d)\right) \geq \gamma(\lambda - \bar{\lambda}) + \max_{Ax \leq b}\left(cx - \bar{\lambda}(Dx - d)\right)$$

where x' is an opt. solution to the right-most subproblem.
Inserting γ we get:

$$\begin{aligned} \max_{Ax \leq b}\left(cx - \lambda(Dx - d)\right) &\geq (d - Dx')(\lambda - \bar{\lambda}) + (cx' - \bar{\lambda}(Dx' - d)) \\ &= cx' - \lambda(Dx' - d) \end{aligned}$$

Intuition
Lagrange relaxation

$$\begin{aligned} &\max z_{LR}(\lambda) = cx - \lambda(Dx - d) \\ &\text{s.t. } Ax \leq b \\ &\quad x \in \mathbb{Z}^n_+ \end{aligned}$$

Gradient in x' is

$$\gamma = d - Dx'$$

Subgradient Iteration
Recursion

$$\lambda^{k+1} = \max\left\{\lambda^k - \theta\gamma^k, 0\right\}$$

where $\theta > 0$ is step-size

If $\gamma > 0$ and θ is sufficiently small $z_{LR}(\lambda)$ will decrease.

- Small θ slow convergence
- Large θ unstable

Held and Karp

Initially

$$\lambda^{(0)} = \{0, \ldots, 0\}$$

compute the new multipliers by recursion

$$\lambda_i^{(k+1)} := \begin{cases} \lambda_i^{(k)} & \text{if } |\gamma_i| \leq \varepsilon \\ \max(\lambda_i^{(k)} - \theta\gamma_i, 0) & \text{if } |\gamma_i| > \varepsilon \end{cases}$$

where γ is subgradient.

The step size θ is defined by

$$\theta = \mu \frac{\overline{z} - \underline{z}}{\sum_i \gamma_i^2}$$

where μ is an appropriate constant.
E.g. $\mu = 1$ and halved if upper bound not decreased in 20 iterations

Lagrange relaxation and LP
For an LP-problem where we Lagrange relax all constraints

- Dual variables are best choice of Lagrange multipliers
- Lagrange relaxation and LP "relaxation" give same bound

Gives a clue to solve LP-problems without Simplex

- Iterative algorithms
- Polynomial algorithms

Outline

1. Lagrangian Relaxation
2. Dantzig-Wolfe Decomposition
   Dantzig-Wolfe Decomposition
   Delayed Column Generation
3. Single Machine Models

Dantzig-Wolfe Decomposition

Motivation

- split it up into smaller pieces a large or difficult problem

Applications

- Cutting Stock problems
- Multicommodity Flow problems
- Facility Location problems
- Capacitated Multi-item Lot-sizing problem
- Air-crew and Manpower Scheduling
- Vehicle Routing Problems
- Scheduling (current research)

Two currently most promising directions for MIP:

- Branch-and-price
- Branch-and-cut

Dantzig-Wolfe Decomposition
The problem is split into a master problem and a subproblem

+ Tighter bounds
+ Better control of subproblem
− Model may become (very) large

Delayed column generation
Write up the decomposed model gradually as needed

- Generate a few solutions to the subproblems
- Solve the master problem to LP-optimality
- Use the dual information to find most promising solutions to the subproblem
- Extend the master problem with the new subproblem solutions.

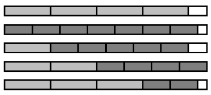
Motivation: Cutting stock problem

- Infinite number of raw stocks, having length L.
- Cut m piece types i, each having width w_i and demand b_i.
- Satisfy demands using least possible raw stocks.

Example:

- $w_1 = 5, b_1 = 7$ ▨
- $w_2 = 3, b_2 = 3$ ■
- Raw length $L = 22$ □

Some possible cuts



Formulation 1

$$\begin{array}{ll}\text{minimize} & u_1+u_2+u_3+u_4+u_5\\ \text{subject to} & 5x_{11}+3x_{12}\le 22u_1\\ & 5x_{21}+3x_{22}\le 22u_2\\ & 5x_{31}+3x_{32}\le 22u_3\\ & 5x_{41}+3x_{42}\le 22u_4\\ & 5x_{51}+3x_{52}\le 22u_5\\ & x_{11}+x_{21}+x_{31}+x_{41}+x_{51}\ge 7\\ & x_{12}+x_{22}+x_{32}+x_{42}+x_{52}\ge 3\\ & u_j\in\{0,1\}\\ & x_{ij}\in\mathbb{Z}_+\end{array}$$

LP-relaxation gives solution value $z=2$ with

$$u_1=u_2=1, x_{11}=2.6, x_{12}=3, x_{21}=4.4$$

Block structure

$$\boxed{\begin{array}{lcccccccccccccccc}\text{min} & & u_1 & & +u_2 & & +u_3 & & +u_4 & & +u_5 & \\ \text{s.t} & x_{11} & & +x_{21} & & +x_{31} & & +x_{41} & & +x_{51} & & \ge 7\\ & x_{12} & & +x_{22} & & +x_{32} & & +x_{42} & & +x_{51} & & \ge 3\\ & \multicolumn{2}{c}{5x_{11}+3x_{12}-22u_1} & & & & & & & & & \le 0\\ & & & \multicolumn{2}{c}{5x_{21}+3x_{22}-22u_2} & & & & & & & \le 0\\ & & & & & \multicolumn{2}{c}{5x_{31}+3x_{32}-22u_3} & & & & & \le 0\\ & & & & & & & \multicolumn{2}{c}{5x_{41}+3x_{42}-22u_4} & & & \le 0\\ & & & & & & & & & \multicolumn{2}{c}{5x_{51}+3x_{52}-22u_5} & \le 0\end{array}}$$

Formulation 2

The matrix A contains all different cutting patterns
All (undominated) patterns:

$$A=\begin{pmatrix}4&0&1&2&3\\0&7&5&4&2\end{pmatrix}$$

Problem

$$\begin{array}{ll}\text{minimize} & \lambda_1+\lambda_2+\lambda_3+\lambda_4+\lambda_5\\ \text{subject to} & 4\lambda_1+0\lambda_2+1\lambda_3+2\lambda_4+3\lambda_5\ge 7\\ & 0\lambda_1+7\lambda_2+5\lambda_3+4\lambda_4+2\lambda_5\ge 3\\ & \lambda_j\in\mathbb{Z}_+\end{array}$$

LP-relaxation gives solution value $z=2.125$ with

$$\lambda_1=1.375, \lambda_4=0.75$$

Due to integer property a lower bound is $\lceil 2.125\rceil=3$.
Optimal solution value is $z^*=3$.

Round up LP-solution getting heuristic solution $z_H=3$.

Decomposition

If model has "block" structure

$$\begin{array}{lccccccc}\max & c^1x^1 & + & c^2x^2 & +\ldots+ & c^Kx^K & & \\ \text{s.t.} & A^1x^1 & + & A^2x^2 & +\ldots+ & A^Kx^K & = & b\\ & D^1x^1 & + & & & & \le & d_1\\ & & + & D^2x^2 & & & \le & d_2\\ & & & & \ldots & & \le & \vdots\\ & & & & & D^Kx^K & \le & d_K\\ & x^1\in\mathbb{Z}_+^{n_1} & & x^2\in\mathbb{Z}_+^{n_2} & \ldots & x^K\in\mathbb{Z}_+^{n_K} & & \end{array}$$

Lagrangian relaxation

Objective becomes

$$\begin{array}{l}c^1x^1+c^2x^2+\ldots+c^Kx^K\\ -\lambda\left(A^1x^1+A^2x^2+\ldots+A^Kx^K-b\right)\end{array}$$

Decomposed into

$$\begin{array}{lccccc}\max c^1x^1-\lambda A^1x^1+c^2x^2-\lambda A^2x^2+\ldots+c^Kx^K-\lambda A^Kx^K+b & & & & & \\ \text{s.t.}\quad D^1x^1 & + & & & & \le d_1\\ & + & D^2x^2 & & & \le d_2\\ & & & \ldots & & \le \vdots\\ & & & & D^Kx^K & \le d_K\\ x^1\in\mathbb{Z}_+^{n_1} & & x^2\in\mathbb{Z}_+^{n_2} & \ldots & x^K\in\mathbb{Z}_+^{n_K} & \end{array}$$

Model is separable

Dantzig-Wolfe decomposition

If model has "block" structure

$$\begin{array}{lccccccc}\max & c^1x^1 & + & c^2x^2 & +\ldots+ & c^Kx^K & & \\ \text{s.t.} & A^1x^1 & + & A^2x^2 & +\ldots+ & A^Kx^K & = & b\\ & D^1x^1 & + & & & & \le & d_1\\ & & + & D^2x^2 & & & \le & d_2\\ & & & & \ldots & & \le & \vdots\\ & & & & & D^Kx^K & \le & d_K\\ & x^1\in\mathbb{Z}_+^{n_1} & & x^2\in\mathbb{Z}_+^{n_2} & \ldots & x^K\in\mathbb{Z}_+^{n_K} & & \end{array}$$

Describe each set X^k, $k=1,\ldots,K$

$$\begin{array}{lccccccc}\max & c^1x^1 & + & c^2x^2 & +\ldots+ & c^Kx^K & & \\ \text{s.t.} & A^1x^1 & + & A^2x^2 & +\ldots+ & A^Kx^K & = & b\\ & x^1\in X^1 & & x^2\in X^2 & \ldots & x^K\in X^K & & \end{array}$$

where $X^k=\{x^k\in\mathbb{Z}_+^{n_k}: D^kx^k\le d_k\}$

Assuming that X^k has finite number of points $\{x^{k,t}\}$ $t\in T_k$

$$X^k=\left\{\begin{array}{ll}x^k\in\mathbb{R}^{n_k}: & x^k=\sum_{t\in T_k}\lambda_{k,t}x^{k,t},\\ & \sum_{t\in T_k}\lambda_{k,t}=1,\\ & \lambda_{k,t}\in\{0,1\}, t\in T_k\end{array}\right\}$$

Dantzig-Wolfe decomposition

Substituting X^k in original model getting *Master Problem*

$$\max c^1\Big(\sum_{t\in T_1}\lambda_{1,t}x^{1,t}\Big)+c^2\Big(\sum_{t\in T_2}\lambda_{2,t}x^{2,t}\Big)+\ldots+c^K\Big(\sum_{t\in T_K}\lambda_{K,t}x^{K,t}\Big)$$

$$\text{s.t. } A^1\Big(\sum_{t\in T_1}\lambda_{1,t}x^{1,t}\Big)+A^2\Big(\sum_{t\in T_2}\lambda_{2,t}x^{2,t}\Big)+\ldots+A^K\Big(\sum_{t\in T_K}\lambda_{K,t}x^{K,t}\Big)=b$$

$$\begin{array}{lll}\sum_{t\in T_k}\lambda_{k,t}=1 & & k=1,\ldots,K\\ \lambda_{k,t}\in\{0,1\}, & t\in T_k & k=1,\ldots,K\end{array}$$

Strength of linear master model

Solving LP-relaxation of master problem, is equivalent to
(Wolsey Prop 11.1)

$$\begin{array}{lcccccc}\max & c^1x^1 & + & c^2x^2 & +\ldots+ & c^kx^k & \\ \text{s.t.} & A^1x^1 & + & A^2x^2 & +\ldots+ & A^kx^k & =b\\ & x^1\in\text{conv}(X^1) & & x^2\in\text{conv}(X^2) & \ldots & x^k\in\text{conv}(X^k) & \end{array}$$

Proof: Consider LP-relaxation

$$\max c^1\Big(\sum_{t\in T_1}\lambda_{1,t}x^{1,t}\Big)+c^2\Big(\sum_{t\in T_2}\lambda_{2,t}x^{2,t}\Big)+\ldots+c^K\Big(\sum_{t\in T_K}\lambda_{K,t}x^{K,t}\Big)$$

$$\text{s.t. } A^1\Big(\sum_{t\in T_1}\lambda_{1,t}x^{1,t}\Big)+A^2\Big(\sum_{t\in T_2}\lambda_{2,t}x^{2,t}\Big)+\ldots+A^K\Big(\sum_{t\in T_K}\lambda_{K,t}x^{K,t}\Big)=b$$

$$\begin{array}{lll}\sum_{t\in T_k}\lambda_{k,t}=1 & & k=1,\ldots,K\\ \lambda_{k,t}\ge 0, & t\in T_k & k=1,\ldots,K\end{array}$$

Informally speaking we have

- joint constraint is solved to LP-optimality
- block constraints are solved to IP-optimality

Strength of Lagrangian relaxation

- z^{LPM} be LP-solution value of master problem
- z^{LD} be solution value of lagrangian dual problem

(Theorem 11.2)

$$z^{LPM}=z^{LD}$$

Proof: Lagrangian relaxing joint constraint in

$$\begin{array}{lccccccc}\max & c^1x^1 & + & c^2x^2 & +\ldots+ & c^Kx^K & & \\ \text{s.t.} & A^1x^1 & + & A^2x^2 & +\ldots+ & A^Kx^K & = & b\\ & D^1x^1 & + & & & & \le & d_1\\ & & + & D^2x^2 & & & \le & d_2\\ & & & & \ldots & & \le & \vdots\\ & & & & & D^Kx^K & \le & d_K\\ & x^1\in\mathbb{Z}_+^{n_1} & & x^2\in\mathbb{Z}_+^{n_2} & \ldots & x^K\in\mathbb{Z}_+^{n_K} & & \end{array}$$

Using result next page

$$\begin{array}{lcccccc}\max & c^1x^1 & + & c^2x^2 & +\ldots+ & c^kx^k & \\ \text{s.t.} & A^1x^1 & + & A^2x^2 & +\ldots+ & A^kx^k & =b\\ & x^1\in\text{conv}(X^1) & & x^2\in\text{conv}(X^2) & \ldots & x^k\in\text{conv}(X^k) & \end{array}$$

Strength of Lagrangian Relaxation (section 10.2)

Integer Programming Problem

$$\begin{array}{ll}\text{maximize} & cx\\ \text{subject to} & Ax\le b\\ & Dx\le d\\ & x_j\in\mathbb{Z}_+,\quad j=1,\ldots,n\end{array}$$

Lagrange Relaxation, multipliers $\lambda\ge 0$

$$\begin{array}{ll}\text{maximize} & z_{LR}(\lambda)=cx-\lambda(Dx-d)\\ \text{subject to} & Ax\le b\\ & x_j\in\mathbb{Z}_+,\quad j=1,\ldots,n\end{array}$$

for best multiplier $\lambda\ge 0$

$$\boxed{\max\left\{cx: Dx\le d, x\in\text{conv}(Ax\le b, x\in\mathbb{Z}_+)\right\}}$$

Delayed Column Generation

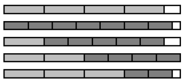
Delayed column generation, linear master

- Master problem can (and will) contain many columns
- To find bound, solve LP-relaxation of master
- Delayed column generation gradually writes up master

Delayed column generation, linear master

- $w_1=5, b_1=7$ ▭
- $w_2=3, b_2=3$ ▬
- Raw length $L=22$

Some possible cuts



In matrix form

$$A=\begin{pmatrix}4&0&1&2&3&\cdots\\0&7&5&4&2&\cdots\end{pmatrix}$$

LP-problem

$$\begin{array}{ll}\min & cx\\ \text{s.t.} & Ax=b\\ & x\ge 0\end{array}$$

where

- $b=(7,3)$,
- $x=(x_1,x_2,x_3,x_4,x_5,\cdots)$
- $c=(1,1,1,1,1,\cdots)$.

Reduced Costs

Simplex in matrix form

$$\min\{cx \mid Ax=b, x\ge\}$$

In matrix form:

$$\begin{bmatrix}0&A\\-1&c\end{bmatrix}\begin{bmatrix}z\\x\end{bmatrix}=\begin{bmatrix}b\\0\end{bmatrix}$$

- $\mathcal{B}=\{1,2,\ldots,p\}$ basic variables
- $\mathcal{L}=\{1,2,\ldots,q\}$ non-basis variables (will be set to lower bound $=0$)
- $(\mathcal{B},\mathcal{L})$ basis structure
- $x_{\mathcal{B}}, x_{\mathcal{L}}, c_{\mathcal{B}}, c_{\mathcal{L}}$,
- $B=[A_1,A_2,\ldots,A_p]$, $L=[A_{p+1},A_{p+2},\ldots,A_{p+q}]$

$$\begin{bmatrix}0&B&L\\-1&c_{\mathcal{B}}&c_{\mathcal{L}}\end{bmatrix}\begin{bmatrix}z\\x_{\mathcal{B}}\\x_{\mathcal{L}}\end{bmatrix}=\begin{bmatrix}b\\0\end{bmatrix}$$

$$Bx_{\mathcal{B}}+Lx_{\mathcal{L}}=b\quad\Rightarrow\quad x_{\mathcal{B}}+B^{-1}Lx_{\mathcal{L}}=B^{-1}b\quad\Rightarrow\quad\left[\begin{array}{l}x_{\mathcal{L}}=0\\x_{\mathcal{B}}=B^{-1}b\end{array}\right.$$

$$\begin{bmatrix}0&B&L\\-1&c_{\mathcal{B}}&c_{\mathcal{L}}\end{bmatrix}\begin{bmatrix}z\\x_{\mathcal{B}}\\x_{\mathcal{L}}\end{bmatrix}=\begin{bmatrix}b\\0\end{bmatrix}$$

Simplex algorithm sets $x_{\mathcal{L}}=0$ and $x_{\mathcal{B}}=B^{-1}b$
B invertible, hence rows linearly independent

The objective function is obtained by multiplying and subtracting constraints by means of multipliers π (the dual variables)

$$z=\sum_{j=1}^{p}\left[c_j-\sum_{i=1}^{p}\pi_ia_{ij}\right]+\sum_{j=1}^{q}\left[c_j-\sum_{i=1}^{p}\pi_ia_{ij}\right]+\sum_{i=1}^{p}\pi_ib_i$$

Each basic variable has cost null in the objective function

$$c_j-\sum_{i=1}^{p}\pi_ia_{ij}=0\qquad\Longrightarrow\qquad\pi=B^{-1}c_{\mathcal{B}}$$

Reduced costs of non-basic variables:

$$c_j-\sum_{i=1}^{p}\pi_ia_{ij}$$

Delayed column generation (example)

- $w_1=5, b_1=7$ ▭
- $w_2=3, b_2=3$ ▬
- Raw length $L=22$

Initially we choose only the trivial cutting patterns

$$A=\begin{pmatrix}4&0\\0&7\end{pmatrix}$$

Solve LP-problem

$$\begin{array}{ll}\min & cx\\ \text{s.t.} & Ax=b\\ & x\ge 0\end{array}$$

i.e.

$$\begin{pmatrix}4&0\\0&7\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix}=\begin{pmatrix}7\\3\end{pmatrix}$$

with solution $x_1=\frac{7}{4}$ and $x_2=\frac{3}{7}$.
The dual variables are $y=c_BA_B^{-1}$ i.e.

$$\begin{pmatrix}1&1\end{pmatrix}\begin{pmatrix}\frac{1}{4}&0\\0&\frac{1}{7}\end{pmatrix}=\begin{pmatrix}\frac{1}{4}\\\frac{1}{7}\end{pmatrix}$$

Small example (continued)

Find entering variable

$$\begin{array}{lcl}A & = & \begin{pmatrix}1&2&3&\cdots\\5&4&2&\cdots\end{pmatrix}\qquad\begin{array}{l}\frac{1}{4}\leftarrow y_1\\\frac{1}{7}\leftarrow y_2\end{array}\\ c_N-yA_N & = & \begin{pmatrix}1-\frac{27}{28}&1-\frac{30}{28}&1-\frac{29}{28}&\cdots\end{pmatrix}\end{array}$$

We could also solve optimization problem

$$\begin{array}{ll}\min & 1-\frac{1}{4}x_1-\frac{1}{7}x_2\\ \text{s.t.} & 5x_1+3x_2\le 22\\ & x\ge 0,\text{integer}\end{array}$$

which is equivalent to knapsack problem

$$\begin{array}{ll}\max & \frac{1}{4}x_1+\frac{1}{7}x_2\\ \text{s.t.} & 5x_1+3x_2\le 22\\ & x\ge 0,\text{integer}\end{array}$$

This problem has optimal solution $x_1=2, x_2=4$.
Reduced cost of entering variable

$$1-2\frac{1}{4}-4\frac{1}{7}=1-\frac{30}{28}=-\frac{1}{14}<0$$

Small example (continued)

Add new cutting pattern to A getting

$$A=\begin{pmatrix}4&0&3\\0&7&2\end{pmatrix}$$

Solve problem to LP-optimality, getting primal solution

$$x_1=\frac{5}{8}, x_3=\frac{3}{2}$$

and dual variables

$$y_1=\frac{1}{4}, y_2=\frac{1}{8}$$

Note, we do not need to care about "leaving variable"
To find entering variable, solve

$$\begin{array}{ll}\max & \frac{1}{4}x_1+\frac{1}{8}x_2\\ \text{s.t.} & 5x_1+3x_2\le 22\\ & x\ge 0,\text{integer}\end{array}$$

This problem has optimal solution $x_1=4, x_2=0$.
Reduced cost of entering variable

$$1-4\frac{1}{4}-0\frac{1}{7}=0$$

Terminate with $x_1=\frac{5}{8}$, $x_3=\frac{3}{2}$, and $z_{LP}=\frac{17}{8}=2.125$.

Questions

- Will the process terminate?

  Always improving objective value. Only a finite number of basis solutions.

- Can we repeat the same pattern?

  No, since the objective functions is improved. We know the best solution among existing columns. If we generate an already existing column, then we will not improve the objective.

Outline

1. Lagrangian Relaxation

2. Dantzig-Wolfe Decomposition
   Dantzig-Wolfe Decomposition
   Delayed Column Generation

3. Single Machine Models

Scheduling

$1|prec|\sum w_jC_j$

Sequencing (linear ordering) variables

$$\min\sum_{j=1}^{n}\sum_{k=1}^{n}w_jp_kx_{kj}+\sum_{j=1}^{n}w_jp_j$$

$$\begin{array}{lll}\text{s.t.} & x_{kj}+x_{lk}+x_{jl}\ge 1 & j,k,l=1,\ldots,nj\ne k,k\ne l\\ & x_{kj}+x_{jk}=1 & \forall j,k=1,\ldots,n,j\ne k\\ & x_{jk}\in\{0,1\} & j,k=1,\ldots,n\\ & x_{jj}=0 & \forall j=1,\ldots,n\end{array}$$

Scheduling

$1|prec|C_{max}$

Completion time variables

$$\begin{aligned}
\min \sum_{j=1}^{n} w_j z_j & \\
\text{s.t. } z_k - z_j \geq p_k & \quad \text{for } j \rightarrow k \in A \\
z_j \geq p_j, & \quad \text{for } j = 1, \dots, n \\
z_k - z_j \geq p_k & \quad \text{or} \quad z_j - z_k \geq p_j, \text{ for } (i,j) \in I \\
z_j \in \mathbf{R}, & \quad j = 1, \dots, n
\end{aligned}$$

Scheduling

$1||\sum h_j(C_j)$

Time indexed variables

$$\begin{aligned}
\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)x_{jt} & \\
\text{s.t.} \sum_{t=1}^{T-p_j+1} x_{jt} = 1, & \quad \text{for all } j = 1, \dots, n \\
\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1, & \quad \text{for each } t = 1, \dots, T \\
x_{jt} \in \{0,1\}, & \quad \text{for each } j = 1, \dots, n;\ t = 1, \dots, T - p_j + 1
\end{aligned}$$

+ This formulation gives better bounds than the two preceding

– pseudo-polynomial number of variables

Dantzig-Wolfe decomposition
Reformulation:

$$\begin{aligned}
\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)x_{jt} & \\
\text{s.t.} \sum_{t=1}^{T-p_j+1} x_{jt} = 1, & \quad \text{for all } j = 1, \dots, n \\
x_{jt} \in X & \quad \text{for each } j = 1, \dots, n;\ t = 1, \dots, T - p_j + 1
\end{aligned}$$

where $X = \left\{ x \in \{0,1\} : \sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1, \text{ for each } t = 1, \dots, T \right\}$

$x^l, l = 1, \dots, L$ extreme points of X.

$$X = \left\{ \begin{array}{lll} x \in \{0,1\} & : & x = \sum_{l=1}^{L} \lambda_l x^l \\ & & \sum_{l=1}^{L} \lambda_l = 1, \\ & & \lambda_l \in \{0,1\} \end{array} \right\}$$

matrix of X is interval matrix

extreme points are integral

they are pseudo-schedules

Dantzig-Wolfe decomposition
Substituting X in original model getting master problem

$$\begin{aligned}
& \min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)(\sum_{l=1}^{L} \lambda_l x^l) \\
\pi \quad & \text{s.t.} \sum_{t=1}^{T-p_j+1} \sum_{l=1}^{L} \lambda_l x_{jt}^l = 1, \quad \text{for all } j = 1, \dots, n \Longleftarrow \sum_{l=1}^{L} \lambda_l n_j^l = 1 \\
\alpha \quad & \sum_{l=1}^{L} \lambda_l = 1, \\
& \lambda_l \in \{0,1\} \Longleftarrow \lambda_l \geq 0 \text{ LP-relaxation}
\end{aligned}$$

- solve LP-relaxation by column generation on pseudo-schedules x^l
- reduced cost of λ_k is $\bar{c}_k = \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (c_{jt} - \pi_j)x_{jt}^k - \alpha$

- The subproblem can be solved by finding shortest path in a network N with
  - $1, 2, \dots, T+1$ nodes corresponding to time periods
  - process arcs, for all j, t, $t \rightarrow t + p_j$ and cost $c_{jt} - \pi_j$
  - idle time arcs, for all t, $t \rightarrow t + 1$ and cost 0
- a path in this network corrsponds to a pseudo-schedule in which a job may be started more than once or not processed.
- the lower bound on the master problem produced by the LP-relaxation of the restricted master problem can be tighten by inequalities

[Pessoa, Uchoa, Poggi de Aragão, Rodrigues, 2008], propose another time index formulation that dominates this one.
They can solve consistently instances up to 100 jobs.

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 13

Parallel Machine Models

Marco Chiarandini

Outline

1. Parallel Machine Models

Outline

1. Parallel Machine Models

$Pm \mid \mid C_{max}$
(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)

$Pm \mid prmp \mid C_{max}$
(with preemption)

Not NP-hard:

- Linear Programming (exercise)
- 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

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 14
Flow Shop Models

Marco Chiarandini

Outline

1. Parallel Machine Models
2. Flow Shop
   - Introduction
   - Makespan calculation
   - Johnson's algorithm
   - Construction heuristics
   - Iterated Greedy
   - Efficient Local Search and Tabu Search

Outline

1. Parallel Machine Models
2. Flow Shop
   - Introduction
   - Makespan calculation
   - Johnson's algorithm
   - Construction heuristics
   - Iterated Greedy
   - Efficient Local Search and Tabu Search

Identical machines

Min makespan, without preemption

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

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

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

$Pm\,|\,p_j=1, M_j\,|\,C_{max}$: least flexible job (LFJ) -
least flexible machine (LFM)
(optimal if M_j are nested)

Identical machines

Min makespan, with preemption

$Pm\,|\,|\,C_{max}$: Not NP-hard:

- Linear Programming (exercise)
- 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

Uniform machines

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

- 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

Unrelated machines

$R\,|\,|\sum_j C_j$ is NP-hard

Solved by local search methods.

- Solution representation
  - a collection of m sequences, one for each job

    recall that $1\,|\,|\sum w_jC_j$ is solvable in $O(n\log n)$
  - indirect representation
    assignment of jobs to machines
    the sequencing is left to the optimal SWPT rule
- Neighborhood: one exchange, swap
- Evaluation function. How costly is the computation?

Outline

1. Parallel Machine Models
2. Flow Shop
   - Introduction
   - Makespan calculation
   - Johnson's algorithm
   - Construction heuristics
   - Iterated Greedy
   - Efficient Local Search and Tabu Search

Flow Shop

General Shop Scheduling:

- $J=\{1,\dots,N\}$ set of jobs; $M=\{1,2,\dots,m\}$ set of machines
- $J_j=\{O_{ij}\,|\,i=1,\dots,n_j\}$ set of operations for each job
- p_{ij} processing times of operations O_{ij}
- $\mu_{ij}\subseteq M$ machine eligibilities for each operation
- precedence constraints among the operations
- one job processed per machine at a time,
  one machine processing each job at a time
- C_j completion time of job j

➨ Find feasible schedule that minimize some regular function of C_j

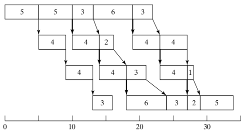
Flow Shop Scheduling:

- $\mu_{ij}=l$, $l=1,2,\dots,m$
- precedence constraints: $O_{ij}\rightarrow O_{i+1,j}$, $i=1,2,\dots,n$ for all jobs

Example

$jobs$	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	5	5	3	6	3
p_{2,j_k}	4	4	2	4	4
p_{3,j_k}	4	4	3	4	1
p_{4,j_k}	3	6	3	2	5

Gantt chart

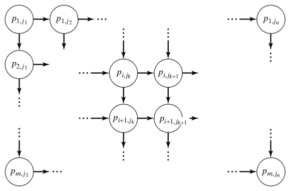


schedule representation
π_1,π_2,π_3,π_4:
$\pi_1: O_{11},O_{12},O_{13},O_{14}$
$\pi_2: O_{21},O_{22},O_{23},O_{24}$
$\pi_3: O_{31},O_{32},O_{33},O_{34}$
$\pi_4: O_{41},O_{42},O_{43},O_{44}$

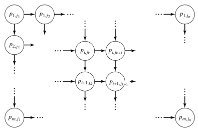
- we assume unlimited buffer
- if same job sequence on each machine ➨ permutation flow shop

Directed Graph Representation

Given a sequence: operation-on-node network,
jobs on columns, and machines on rows



Directed Graph Representation



Recursion for C_{max}

$$\begin{aligned}C_{i,\pi(1)}&=\sum_{l=1}^{i}p_{l,\pi(1)}\\C_{1,\pi(j)}&=\sum_{l=1}^{j}p_{l,\pi(l)}\\C_{i,\pi(j)}&=\max\{C_{i-1,\pi(j)},C_{i,\pi(j-1)}\}+p_{i,\pi(j)}\end{aligned}$$

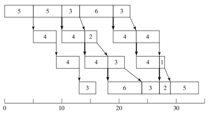
Computation cost?

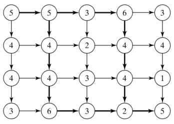
Example

$jobs$	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	5	5	3	6	3
p_{2,j_k}	4	4	2	4	4
p_{3,j_k}	4	4	3	4	1
p_{4,j_k}	3	6	3	2	5

$C_{max}=34$

corresponds to longest path



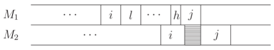


$Fm\,|\,|\,C_{max}$

Theorem

There always exist an optimum sequence without change in the first two and last two machines.

Proof: By contradiction.



Corollary

$F2\,|\,|\,C_{max}$ *and* $F3\,|\,|\,C_{max}$ *are permutation flow shop*

Note: $F3\,|\,|\,C_{max}$ is strongly NP-hard

$F2\,|\,|\,C_{max}$

Intuition: give something short to process to 1 such that 2 becomes operative and give something long to process to 2 such that its buffer has time to fill.

Constructs a sequence $T: T(1),\dots,T(n)$ to process in the same order on both machines by concatenating two sequences:
a left sequence $L: L(1),\dots,L(t)$, and a right sequence
$R: R(t+1),\dots,R(n)$, that is, $T=L\circ R$

[Selmer Johnson, 1954, Naval Research Logistic Quarterly]

```
        Let $J$ be the set of jobs to process
        Let $T, L, R = \emptyset$
Step 1  Find $(i^*, j^*)$ such that $p_{i^*,j^*} = \min\{p_{ij} \,|\, i \in 1,2, j \in J\}$
Step 2  If $i^* = 1$ then $L = L \circ \{i^*\}$
        else if $i^* = 2$ then $R = R \circ \{i^*\}$
Step 3  $J := J \setminus \{j^*\}$
Step 4  If $J \neq \emptyset$ go to Step 1 else $T = L \circ R$
```

Theorem

The sequence $T: T(1),\dots,\dots,T(n)$ *is optimal.*

Proof

- Assume at one iteration of the algorithm that job k has the min processing time on machine 1. Show that in this case job k has to go first on machine 1 than any other job selected later.
- By contradiction, show that if in a schedule S a job j precedes k on machine 1 and has larger processing time on 1, then S is a worse schedule than S'.
  There are three cases to consider.
- Iterate the prove for all jobs in L.
- Prove symmetrically for all jobs in R.

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

[Proportionate permutation flow shop]

- **Theorem:** $C_{max}=\sum_{j=1}^{n}p_j+(m-1)\max(p_1,\dots,p_n)$ 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 (1)

$Fm\,|\,prmu\,|\,C_{max}$

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

Construction Heuristics (2)

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

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 $O(n^2m)$

[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}.

Iterated Greedy

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

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 $n = 500 \times m = 20$:

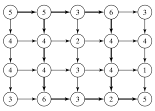
- NEH average gap 3.35% in less than 1 sec.
- IG average gap 0.44% in about 360 sec.

Efficient local search for $Fm \mid prmu \mid C_{max}$

Tabu search (TS) with insert neighborhood.

TS uses best strategy. ➨ need to search efficiently!

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



A sequence $t = (t_1, t_2, \ldots, t_{m-1})$ defines a path in π:

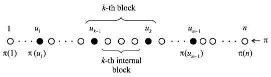
$$C(\pi, t) = \sum_{j=1}^{t_1} p_{\pi(j)1} + \sum_{j=t_1}^{t_2} p_{\pi(j)2} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m}.$$

C_{max} expression through critical path:

$$C_{\max}(\pi) = \max_{1 \leqslant t_1 \leqslant t_2 \leqslant \cdots \leqslant t_{m-1} \leqslant n} \left(\sum_{j=1}^{t_1} p_{\pi(j)1} + \sum_{j=t_1}^{t_2} p_{\pi(j)2} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m} \right)$$

critical path: $\vec{u} = (u_1, u_2, \ldots, u_m) : C_{max}(\pi) = C(\pi, u)$

Block B_k and Internal Block B_k^{Int}



Theorem (Werner, 1992)

Let $\pi, \pi' \in \Pi$, if π' has been obtained from π by an job insert so that $C_{max}(\pi') < C_{max}(\pi)$ then in π':

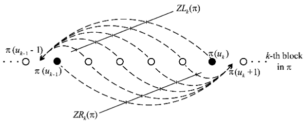
a) *at least one job $j \in B_k$ precedes job $\pi(u_{k-1})$, $k = 1, \ldots, m$*

b) *at least one job $j \in B_k$ succeeds job $\pi(u_k)$, $k = 1, \ldots, m$*

Corollary (Elimination Criterion)

If π' is obtained by π by an "internal block insertion" then $C_{max}(\pi') \geq C_{max}(\pi)$.

Hence we can restrict the search to where the good moves can be:



Further speedup: Use of lower bounds in delta evaluations:
Let δ^r_{x,u_k} indicate insertion of x after u_k (move of type $ZR_k(\pi)$)

$$\Delta(\delta^r_{x,u_k}) = \begin{cases} p_{\pi(x),k+1} - p_{\pi(u_k),k+1} & x \neq u_{k-1} \\ p_{\pi(x),k+1} - p_{\pi(u_k),k+1} + p_{\pi(u_{k-1}+1),k-1} - p_{\pi(x),k-1} & x = u_{k-1} \end{cases}$$

That is, add and remove from the adjacent blocks
It can be shown that:

$$C_{max}(\delta^r_{x,u_k}(\pi)) \geq C_{max}(\pi) + \Delta(\delta^r_{x,u_k})$$

Theorem (Nowicki and Smutnicki, 1996, EJOR)

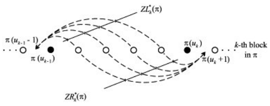
The neighborhood thus defined is connected.

Metaheuristic details:

Prohibition criterion:
an insertion δ_{x,u_k} is tabu if it restores the relative order of $\pi(x)$ and $\pi(x+1)$.

Tabu length: $TL = 6 + \left[\frac{n}{10m}\right]$

Perturbation



- perform all *inserts* among all the blocks that have $\Delta < 0$
- 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 Δ.
Apply move. Compute true $C_{max}(\delta(\pi))$.
If improving compare with C^* and in case update.
Else increase number of idle iterations.
Perturbation : Apply perturbation if `MaxIdleIter` done.
Stop criterion : Exit if `MaxIter` iterations are done.

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 15

Flow Shop and Job Shop Models

Marco Chiarandini

Outline

1. Job Shop
   - Modelling
   - Exact Methods
   - Local Search Methods
   - Shifting Bottleneck Heuristic

Outline

1. Job Shop
   - Modelling
   - Exact Methods
   - Local Search Methods
   - Shifting Bottleneck Heuristic

Job Shop

General Shop Scheduling:

- $J = \{1, \dots, N\}$ set of jobs; $M = \{1, 2, \dots, m\}$ set of machines
- $J_j = \{O_{ij} \mid i = 1, \dots, n_j\}$ set of operations for each job
- p_{ij} processing times of operations O_{ij}
- $\mu_{ij} \subseteq M$ machine eligibilities for each operation
- precedence constraints among the operations
- one job processed per machine at a time, one machine processing each job at a time
- C_j completion time of job j

➨ Find feasible schedule that minimize some regular function of C_j

Job shop

- $\mu_{ij} = l, l = 1, \dots, n_j$ and $\mu_{ij} \neq \mu_{i+1,j}$ (one machine per operation)
- $O_{1j} \to O_{2j} \to \dots \to O_{n_j,j}$ precedences (without loss of generality)
- 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,

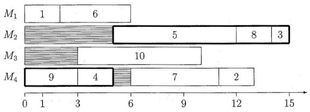
  - $S_{ij} + p_{ij} \leq S_{i+1,j}$ for all $O_{ij} \to O_{i+1,j}$
  - $S_{ij} + p_{ij} \leq S_{uv}$ or $S_{uv} + p_{uv} \leq S_{ij}$ for all operations with $\mu_{ij} = \mu_{uv}$.

  and has minimum makespan: $\min\{\max_{j \in J}(S_{n_j,j} + p_{n_j,j})\}$.

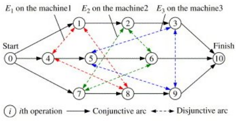
A schedule can also be represented by an m-tuple $\pi = (\pi^1, \pi^2, \dots, \pi^m)$ where π^i defines the processing order on machine i.

There is always an optimal schedule that is semi-active.

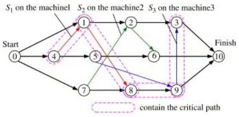
(semi-active schedule: for each machine, start each operation at the earliest feasible time.)



- Often simplified notation: $N = \{1, \dots, n\}$ 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 ⇔ semi-active schedule (feasible earliest start schedule).
- Length of longest path 0–$(n+1)$ in D corresponds to the makespan



Longest path computation

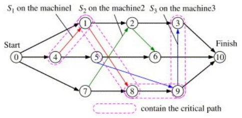
In an acyclic digraph:

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

$$r_0 = 0$$

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

- 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

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

- 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.

Branch and Bound [Carlier and Pinson, 1983] [B2, p. 179]

Let Ω contain the first operation of each job;
Let $r_{ij} = 0$ for all $O_{ij} \in \Omega$

Machine Selection Compute for the current partial schedule

$$t(\Omega) = \min_{ij \in \Omega}\{r_{ij} + p_{ij}\}$$

and let i^* denote the machine on which the minimum is achieved

Branching Let Ω' denote the set of all operations O_{i^*j} on machine i^* such that

$$r_{i^*j} < t(\Omega) \qquad \text{(i.e. eliminate } r_{i^*j} \geq t(\Omega)\text{)}$$

For each operation in Ω', consider an (extended)partial schedule with that operation as the next one on machine i^*.
For each such (extended) partial schedule, delete the operations from Ω, include its immediate follower in Ω and return to Machine Selection.

Lower Bounding:

- longest path in partially selected disjunctive digraph
- solve $1|r_{ij}|L_{max}$ on each machine i like if all other machines could process at the same time (see later shifting bottleneck heuristic) + longest path.

Efficient local search for job shop

Solution representation:
m-tuple $\pi = (\pi^1, \pi^2, \dots, \pi^m) \Longleftrightarrow$ oriented digraph $D_\pi = (N, A, E_\pi)$

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

Issues:

1. Can it be decided easily if the new digraph $D_{\pi'}$ 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 weakly optimal 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 contains $JS(v)$ and

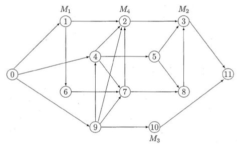
$$L(v, n) \geq L(JS(u), n)$$

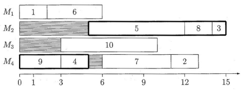
then a forward insert of u after v yields an acyclic complete selection.

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

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 $O(n)$

acyclic complete directed graph $\iff$ transitive closure of its unique directed Hamiltonian 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 one of the past lectures

[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

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 16

Job Shop
The Alternative Graph Model

Marco Chiarandini

Outline

1. Job Shop Generalizations

Resume

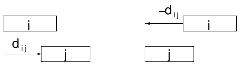
Job Shop:

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

Outline

1. Job Shop Generalizations

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

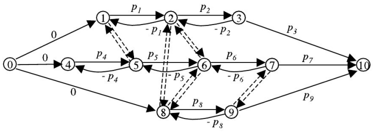
$$\begin{array}{lll} \min & C_{max} & \\ s.t. & x_{ij} + d_{ij} \leq C_{max} & \forall\, O_{ij} \in N \\ & x_{ij} + d_{ij} \leq x_{lj} & \forall\, (O_{ij}, O_{lj}) \in A \\ & x_{ij} + d_{ij} \leq x_{ik} \vee x_{ij} + d_{ij} \leq x_{ik} & \forall\, (O_{ij}, O_{ik}) \in E \\ & x_{ij} \geq 0 & \forall\,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

- 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:

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

- Machine unavailabilities:
  - Machine M_k unavailable in $[a_1, b_1],\ [a_2, b_2],\ \ldots\ ,\ [a_v, b_v]$
  - 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:

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

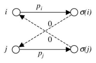
Arises with limited buffers:
after processing, a job remains on the machine until the next machine is freed

- 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

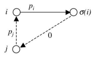
$$S_i + p_i \leq S_j \qquad \text{or} \qquad S_j + p_j \leq S_i$$

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



$$S_{\sigma(j)} \leq S_i \qquad \text{or} \qquad S_{\sigma(i)} \leq S_j$$

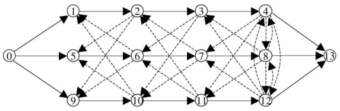
3. i is blocking, j is non-blocking (ideal) and i, j to be processed on the same machine $\mu(i) = \mu(j)$.



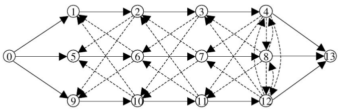
$$S_i + p_i \leq S_j \qquad \text{or} \qquad S_{\sigma(j)} \leq S_i$$

Example

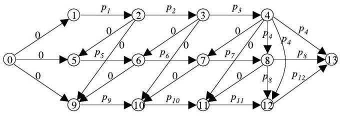
- $O_0, O_1, \ldots, O_{13}$
- $M(O_1) = M(O_5) = M(O_9)$
  $M(O_2) = M(O_6) = M(O_{10})$
  $M(O_3) = M(O_7) = M(O_{11})$



- 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

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

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 $G(S)$.
- Problem: now the digraph may contain cycles. Longest path with simple cyclic paths is NP-complete. However, here we have to care only of non-positive cycles.
- 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].

Heuristic Methods

- 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
    $$\max_{(uv) \in A} \{l(0, u) + a_{uv} + l(v, n)\}$$
  - *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*
    find 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.

Implementation details of the slave heuristics

- Once an arc is added we need to update all $L(0, u)$ and $L(u, n)$. Backward and forward visit $O(|F| + |A|)$
- When adding arc a_{ij}, we detect positive cycles if $L(i, j) + a_{ij} > 0$. This happens only if we updated $L(0, i)$ or $L(j, n)$ in the previous point and hence it comes for free.
- Overall complexity $O(|A|(|F| + |A|))$

Speed up of Rollout:

- Stop if partial solution overtakes upper bound
- limit evaluation to say 20% of arcs in A

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 17

Resource Constrained Project Scheduling
Reservations

Marco Chiarandini

Outline

1. Exercises

2. RCPS Model
   Preliminaries
   Heuristics for RCPSP

Outline

1. Exercises

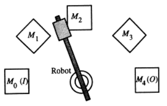
2. RCPS Model
   Preliminaries
   Heuristics for RCPSP

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 $M_1, M_2, \dots M_m$
- $c_{i,i+1}$ 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 ⇝ minimize period

Alternative graph model with intermediate robot operations

Outline

1. Exercises

2. RCPS Model
   Preliminaries
   Heuristics for RCPSP

insert figures that you find in diku.pdf

RCPS Model

Resource Constrained Project Scheduling Model

Given:

- activities (jobs) $j = 1, \dots, n$
- renewable resources $i = 1, \dots, m$
- 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 $0 = t_i^1 < t_i^2 < \dots < t_i^{m_i} = T$
- Multiple modes for an activity j
  processing time and use of resource depends on its mode m: p_{jm}, r_{jkm}.

Modeling

Case 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.

Modeling

Case 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.

Modeling

Case 3

- A set of jobs $J_1, \dots, J_g$ are to be processed by auditors $A_1, \dots, A_m$.
- Job J_l consists of n_l tasks $(l = 1, \dots, g)$.
- 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 $[s_k^\nu, l_k^\nu]$ ($\nu = 1, \dots, m$) with $l_k^\nu < s_k^\nu$ for $\nu = 1, \dots, m_k - 1$.
- Furthermore, the total working time of A_k is bounded from below by H_k^- and from above by H_k^+ with $H_k^- \leq H_k^+$ $(k = 1, \dots, m)$.
- We have to find an assignment $\alpha(i)$ for each task $i = 1, \dots, n := \sum_{l=1}^g n_l$ 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 $k = 1, \dots, m$.
  - 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.

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 $i \rightarrow j$ $\qquad S_i + p_i \leq S_j$
   [precedence constrains]
2. parallelity constraints $i \,||\, j$ $\qquad S_i + p_i \geq S_j$ 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 $i - j$ $\qquad S_i + p_i \leq S_j$ or $S_j + p_j \leq S_i$
   [resource constraints: $r_{jk} + r_{lk} > R_k$]

N. Strengthenings: symmetric triples, etc.

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 $\pi = (j_1, \dots, j_n)$
- π determines the order of activities to be passed to a schedule generation scheme

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 $t_{\lambda+1} = \min \left\{ \min_{j \in A_\lambda} C_j, \min_{\substack{k=1,\dots,r \\ i \in m_k}} t_i^\mu \right\}$
and go to Step 1.

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

Possible uses:

- Forward
- Backward
- Bidirectional
- Forward-backward improvement (justification techniques)
  [V. Valls, F. Ballestín and S. Quintanill, EJOR, 2005]

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

reduced to only those moves compatible with precedence constraints

Genetic Algorithms

Recombination operator:

- One point crossover
- Two point crossover
- Uniform crossover

Implementations compatible with precedence constraints

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 18

Reservations and Educational Timetabling

Marco Chiarandini

Outline

1. Reservations without slack
2. Reservations with slack
3. Timetabling with one Operator
4. Timetabling with Operators
5. Educational Timetabling
   Introduction
   School Timetabling

Timetabling

- Educational Timetabling
  - School/Class timetabling
  - University timetabling
- Personnel/Employee timetabling
  - Crew scheduling
  - Crew rostering
- Transport Timetabling
- Sports Timetabling
- Communication Timetabling

Outline

1. Reservations without slack
2. Reservations with slack
3. Timetabling with one Operator
4. Timetabling with Operators
5. Educational Timetabling
   Introduction
   School Timetabling

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

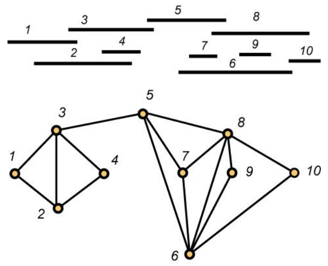
1. $p_j = 1$

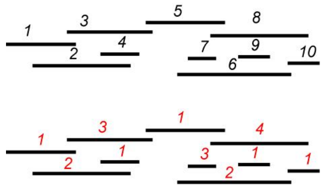
Solve an assignment problem at each time slot

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

1. Reservations without slack
2. Reservations with slack
3. Timetabling with one Operator
4. Timetabling with Operators
5. Educational Timetabling
   Introduction
   School Timetabling

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 $p_j \le d_j - r_j$

Task: Maximize weight of assigned activities

Heuristics

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

1. Reservations without slack
2. Reservations with slack
3. Timetabling with one Operator
4. Timetabling with Operators
5. Educational Timetabling
   Introduction
   School Timetabling

Timetabling with one Operator

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.

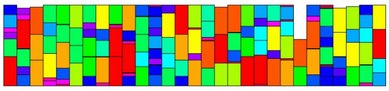
- RCPSP Model
- If p_j all the same ➜ Bin Packing Problem (still NP-hard)

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) $C_{max}(FFD) \le \frac{11}{9} C_{max}(OPT) + \frac{6}{9}$
- 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

1. Reservations without slack
2. Reservations with slack
3. Timetabling with one Operator
4. Timetabling with Operators
5. Educational Timetabling
   Introduction
   School Timetabling

Timetabling with Operators

- 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

1. Reservations without slack
2. Reservations with slack
3. Timetabling with one Operator
4. Timetabling with Operators
5. Educational Timetabling
   Introduction
   School Timetabling

Educational timetabling process

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

The Timetabling Activity

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

Two categories of constraints:

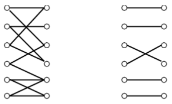
Hard constraints $\mathsf{H} = \{\mathsf{H}_1, \ldots, \mathsf{H}_n\}$: must be strictly satisfied, no violation is allowed

Soft constraints $\Sigma = \{\mathsf{S}_1, \ldots, \mathsf{S}_m\}$: 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.

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

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

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.

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 $G = (\mathcal{C}, \mathcal{P}, \mathcal{R})$:

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

Time slots are colors ➜ Graph-Edge Coloring problem

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

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

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

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

Constraints:

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

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:

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

- 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

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 19
University Timetabling

Marco Chiarandini

Outline

1. University Timetabling
   - Formalization and Modelling
   - An Example
   - Timetabling in Practice

Outline

1. University Timetabling
   - Formalization and Modelling
   - An Example
   - Timetabling in Practice

Course Timetabling

The weekly scheduling of the lectures/events/courses 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 $\mathcal{C}_i = \{L_{i1}, \ldots, L_{il_i}\}$. 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.

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?

IP model

Including the assignment of indistinguishable rooms
m_t rooms ⇒ maximum number of lectures in time slot t

Variables

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

Number of lectures per course

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

Number of lectures per time slot

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

Number of lectures per time slot (students' perspective)

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

If some preferences are added:

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

Corresponds to a bounded coloring. [de Werra, 1985]

Further complications:

- Teachers that teach more than one course
  (not really a complication: treated similarly to students' enrollment)
- A set of rooms $\mathcal{R} = \{R_1, \ldots, R_n\}$
  with eligibility constraints
  (this can be modeled as Hypergraph Coloring [de Werra, 1985]:
  - introduce an (hyper)edge for events that can be scheduled in the same room
  - the edge cannot have more colors than the rooms available of that type)

Moreover,

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

IP approach

3D IP model including room eligibility [Lach and Lübbecke, 2008]

$R(c) \subseteq \mathcal{R}$: rooms eligible for course c
$G_{conf} = (V_{conf}, E_{conf})$: conflict graph (vertices are pairs (c, t))

$$\begin{aligned}
&\min \sum_{ctr} d(c,t) x_{ctr} && \forall c \in \mathcal{C}\\
&\sum_{\substack{t \in T\\ r \in R(c)}} x_{ctr} = l(c) && \forall c \in \mathcal{C}\\
&\sum_{c \in R^{-1}(r)} x_{ctr} \leq 1 && \forall t \in T, r \in \mathcal{R}\\
&\sum_{r \in R(c_1)} x_{c_1 t_1 r} + \sum_{r \in R(c_2)} x_{c_2 t_2 r} \leq 1 && \forall ((c_1,t_1)(c_2,t_2)) \in E_{conf}\\
&x_{ctr} \in \{1, 0\} && \forall (c,t) \in V_{conf}, r \in \mathcal{R}
\end{aligned}$$

This 3D model is too large in size and computationally hard to solve

2D IP model including room eligibility [Lach and Lübbecke, 2008]

Decomposition of the problem in two stages:

Stage 1 assign courses to timeslots
Stage 2 match courses with rooms within each timeslot solved by bipartite matching

Model in stage 1

Variables: course c assigned to time slot t

$$x_{ct} \in \{0,1\} \qquad c \in \mathcal{C}, t \in \mathcal{T}$$

Edge constraints
(forbids that c_1 is assigned to t_1 and c_2 to t_2 simultaneously)

$$x_{c_1,t_1} + x_{c_2,t_2} \leq 1 \qquad \forall\, ((c_1,t_1),(c_2,t_2)) \in E_{conf}$$

Hall's constraints
(guarantee that in stage 1 we find only solutions that are feasibile for stage 2)
$G_t = (\mathcal{C}_t \cup \mathcal{R}_t, E_t)$ bipartite graph for each t
$G = \cup_t G_t$

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

If some preferences are added:

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

- Hall's constraints are exponentially many
- [Lach and Lübbecke] study the polytope of the bipartite matching and find strengthening conditions

  (polytope: convex hull of all incidence vectros defining subsets of $\mathcal{C}$ perfectly matched)
- Algorithm for generating all facets not given but claimed efficient
- Could solve the overall problem by branch and cut (separation problem is easy).
  However the the number of facet inducing Hall inequalities is in practice rather small hence they can be generated all at once

So far feasibility.

Preferences (soft constraints) may be introduced [Lach and Lübbecke, 2008b]

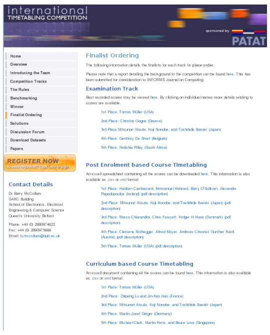
- Compactness or distribution
- Minimum working days
- Room stability
- Student min max load per day
- Travel distance
- Room eligibility
- Double lectures
- Professors' preferences for time slots

Different ways to model them exist.
Often the auxiliary variables have to be introduced

Examination 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



Contact Details

Dr Barry McCollum
SARC Building
School of Electronics, Electrical
Engineering & Computer Science
Queen's University Belfast

Phone: +44 (0) 2890974622
Fax: +44 (0) 2890975666
Email: b.mccollum@qub.ac.uk

Finalist Ordering

The following information details the finalists for each track in place order.

Please note that a report detailing the background to the competition can be found here. This has been submitted for consideration to INFORMS Journal on Computing.

Examination Track

Best recorded scores may be viewed here. By clicking on individual names more details relating to scores are available.

1st Place: Tomas Müller (USA)
2nd Place: Christos Gogos (Greece)
3rd Place:Mitsunori Atsuta, Koji Nonobe, and Toshihide Ibaraki (Japan)
4th Place: Geoffrey De Smet (Belgium)
5th Place: Nelishia Pillay (South Africa)

Post Enrolment based Course Timetabling

An excell spreadsheet containing all the scores can be downloaded here. This information is also available as .csv or.xml format.

1st Place: Hadrien Cambazard, Emmanuel Hebrard, Barry O'Sullivan, Alexandre Papadopoulos (Ireland) (pdf description)
2nd Place: Mitsunori Atsuta, Koji Nonobe, and Toshihide Ibaraki (Japan) (pdf description)
3rd Place: Marco Chiarandini, Chris Fawcett, Holger H Hoos (Denmark) (pdf description)
4th Place: Clemens Nothegger, Alfred Mayer, Andreas Chwatal, Gunther Raidl (Austria) (pdf description)
5th Place: Tomas Müller (USA) (pdf description)

Curriculum based Course Timetabling

An excell document containing all the scores can be found here. This information is also available as .csv or.xml format.

1st Place: Tomas Müller (USA)
2nd Place: Zhipeng Lu and Jin-Kao Hao (France)
3rd Place: Mitsunori Atsuta, Koji Nonobe, and Toshihide Ibaraki (Japan)
4th Place: Martin Josef Geiger (Germany)
5th Place: Michael Clark, Martin Henz, and Bruce Love (Singapore)

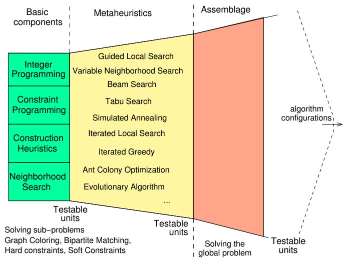
2007 Competition

- Constraint Programming is shown by [Cambazard et al. (PATAT 2008)] to be not yet competitive
- Integer programming is promising [Lach and Lübbecke] and under active development (see J.Marecek `http://www.cs.nott.ac.uk/~jxm/timetabling/`) however it was not possible to submit solvers that make use of IP commercial programs
- Two teams submitted to all three tracks:
  - [Ibaraki, 2008] models everything in terms of CSP in its optimization counterpart. The CSP solver is relatively very simple, binary variables + tabu search
  - [Tomas Mueller, 2008] developed an open source Constraint Solver Library based on local search to tackle University course timetabling problems (`http://www.unitime.org`)
  - All methods ranked in the first positions are heuristic methods based on local search

Heuristic 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 (problem instance) are specific of the institution.

Appropriate techniques exist to aid in the experimental assessment of algorithms. Example: F-race [Birattari et al. 2002]
(see: `http://www.imada.sdu.dk/~marco/exp/` for a full list of references)

Post Enrollment Timetabling

Definition

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;
precedences are satisfied;

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

Graph models

We define:

- precedence digraph $D = (V, A)$: directed graph having a vertex for each lecture in the vertex set V and an arc from u to v, $u, v \in V$, if the corresponding lecture u must be scheduled before v.
- Transitive closure of D: $D' = (V, A')$
- conflict graph $G = (V, E)$: edges connecting pairs of lectures if:
  - the two lectures share students;
  - the two lectures can only be scheduled in a room that is the same for both;
  - there is an arc between the lectures in the digraph D'.

A look at the instances

ID	year	lecs	studs	rooms	lecs/stud	studs/lec	rooms/le(	degree	slots/lec	slots/lec	slots/lec	Prec.	Rel. Prec.
1	2007	400	500	10	21.02	26.27	4.08	0.34	16	25.34	34	40	14
2	2007	400	500	10	21.03	26.29	3.95	0.37	17	25.69	33	36	14
3	2007	200	1000	20	13.38	66.92	5.04	0.47	19	25.54	33	20	11
4	2007	200	1000	20	13.40	66.98	6.40	0.52	15	25.66	33	20	9
5	2007	400	300	20	20.92	15.69	6.80	0.31	16	25.43	34	120	43
6	2007	400	300	20	20.73	15.54	5.07	0.30	13	25.39	36	119	32
7	2007	200	500	20	13.47	33.66	1.57	0.53	9	17.86	26	20	10
8	2007	200	500	20	13.83	34.58	1.92	0.52	11	17.17	26	21	13
9	2007	400	500	10	21.43	26.79	2.91	0.34	17	25.42	34	41	18
10	2007	400	500	10	20.98	26.23	3.20	0.38	14	25.47	34	40	13
11	2007	200	1000	10	13.61	68.04	3.38	0.50	17	25.32	35	21	17
12	2007	200	1000	10	13.61	68.03	3.35	0.58	15	25.67	35	20	13
13	2007	400	300	20	21.19	15.89	8.68	0.32	17	25.75	34	116	34
14	2007	400	300	20	20.86	15.64	7.56	0.32	17	25.44	36	118	46
15	2007	200	500	10	13.05	32.63	2.23	0.54	11	17.38	24	21	13
16	2007	200	500	10	13.64	34.09	1.74	0.46	10	17.57	25	19	10

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:

- Room eligibility complicate the use of IP and CP.
- Heuristics:

  Complete assignment of lectures
  Partial assignment of lectures
- Room assignment:
  A. Left to matching algorithm
  B. Carried out heuristically (matrix representation of solutions)

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

Time-slots

Rooms

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

Construction Heuristic

most-constrained lecture on least constraining time slot

Step 1. Initialize the set $\widehat{L}$ of all unscheduled lectures with $\widehat{L} = L$.
Step 2. Choose a lecture $L_i \in \widehat{L}$ according to a *heuristic rule*.
Step 3. Let $\widehat{X}$ be the set of all positions for L_i in the assignment matrix with minimal violations of the hard constraints H.
Step 4. Let $\bar{X} \subseteq \widehat{X}$ be the subset of positions of $\widehat{X}$ 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 $\widehat{L}$, and go to step 2 until $\widehat{L}$ 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:

complete assignment representation: TabuCol with one exchange

partial assignment representation: PartialCol with i-swaps

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

B. Room assignment included

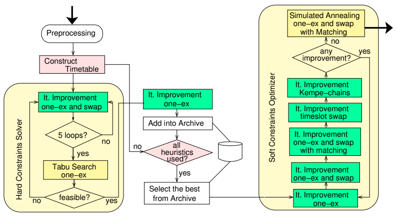
	Monday									Tuesday									Wednesday								
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23	T24	T25	T26	T27
R1	187	239	378	66	380	53	208	279		300	350	211	375	254	366	369	223	163	298		118	368	234	97	329	274	58
R2	360	345	2	153		354	91	61	319	349	278	86	204	316	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_5: Insert + Rematch
- N_3: Period Swap
- N_4: Kempe Chain Interchange
- N_6: Swap + Rematch

Example of stochastic local search for Hard Constraints, representation 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


Preprocessing
Construct Timetable
Hard Constraints Solver
It. Improvement one–ex and swap
5 loops?
no
yes
Tabu Search one–ex
feasible?
no
yes
It. Improvement one–ex
Add into Archive
no
all heuristics used?
yes
Select the best from Archive
Soft Constraints Optimizer
It. Improvement one–ex
It. Improvement one–ex and swap
It. Improvement one–ex and swap with matching
It. Improvement timeslot swap
It. Improvement Kempe–chains
any improvement?
yes
no
Simulated Annealing one–ex and swap with Matching

In Practice

A timetabling system consists of:

- Information management (database maintenance)
- 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.

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 20

Timetabling in Transportation

Marco Chiarandini

Outline

Outline

Outline

Problems

- Tanker Scheduling
- Aircraft Routing and Scheduling
- Public Transports

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
- Lagrangian relaxation (solution without Simplex)
- Branch and price (column generation)

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

Network Flow

Network representation of the tanker scheduling problem:

- a node for each shipment
- an arc from i to j if possible to accomplish j after completing i
- a directed path corresponds to a feasible schedule for the tank

Model as minimum value problem solvable by maximum flow algorithm in the following network:

- split each node i into i' and i''
- introduce shipment arcs (i', i'') of flow lower bound 1
- introduce source and sink
- set all flow upper bounds to 1

Finds minimum number of ships required to cover the cargos. Does not include costs.

IP model

Two phase approach:

determine for each ship i the set S_i of all possible itineraries

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

$$x_i^l \in \{0,1\} \qquad \forall i = 1, \ldots, s;\ l \in S_i$$

Each cargo is assigned to at most one ship:

$$\sum_{i=1}^s \sum_{l \in S_i} a_{ij}^l x_i^l \leq 1 \qquad \forall j = 1, \ldots, n$$

Each tanker can be assigned at most one itinerary

$$\sum_{l \in S_i} x_i^l \leq 1 \qquad \forall i = 1, \ldots, s$$

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:

- selecting 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 and of other ships covering the same cargo)
- selecting one ship and branching on its itineraries
  select the ship that may lead to largest profit or largest cargo or with largest number of fractional variables.

Primal heuristics

- Improve the formulation: the goal of improving the lower bounds or solutions whose real variables are closer to be integer
- Use heuristics within the IP framework. Goal: finding good feasible solutions
  - construction heuristics
  - improvement heuristics

The following heuristics can be applied at each node of a branch-and-cut/bound tree

Truncated MIP

Run branch-and-cut/bound for a fixed amount of time and return the best solution when time exceeds.

Diving

Carry out a depth-first search in branch-and-cut/bound tree.
At each node, fix variables that take integer values in the LP relaxation and branch on the others

- LP-driven dives: fix the variable that is closest to integer
- IP-driven or guided dives: given an incumbent solution, choose the variable to be fixed next and assign it the value it has in the incumbent

These are typically already implemented in MIP systems
LP or incumbent solutions are the guide.

LP-and-fix or Cut-and-Fix

Fix everything that is integer and solve the resulting MIP^{LP-FIX}

Either the new problem is infeasible or it provides and LP-and-fix heuristic solution

(best solutions if formulation is tight and has few fractional variables)

Relax-and-fix

Partition the variables into R disjoint sets and solve sequentially R MIPs, MIP^r with $1 \leq r \leq R$.
(For example partitions correspond to variables of a tank, machine, product family, location, most often time periods)

- In the first MIP^1 impose integrality in the first partition and relax all the others
- Fix the variables in the first partition at the values found in MIP^1
- In the subsequent MIP^r, for $2 \leq r \leq R$ additionally fix the values of the variables of the $r-1$-th partition at the optimal value from MIP^{r-1} and add integrality restriction for the variables in the r-th partition.
- Either MIP^r is infeasible for some r and the heuristic has failed or else the solution found at $r = R$ is a relax-and-fix heuristic solution

(allow overlap between the partitions may be a good idea)
(Note: only MIP^1 is a valid lower bound to the MIP)

Exchange

Improvement version of the relax-and-fix heuristic

At each step r with $1 \leq r \leq R$ the MIP solved is obtained by fixing at their value in the best solution all the variables in the set $r-1$ partitions and imposing integrality to the variables in the r partition

Relaxation Induced Neighborhood Search

Explore neighborhood between LP solution $\hat{s}$ and best known feasible solution $\bar{s}$

Fix a variable that has same value in $\hat{s}$ and $\bar{s}$ and solve the IP problem

Either the solution found is infeasible or it is not found within a time limit so the heuristic has failed or the solution found is an heuristic solution

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 branch and bound most often unclear how to fix variables
➜ Idea: soft fixing

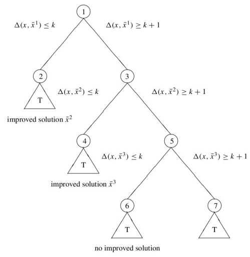
Given a feasible solution $\bar{x}$ let $\bar{O} := \{i \in B\ :\ \bar{x}_i = 1\}$.
Define the k-opt neighborhood $\mathcal{N}(\bar{x}, k)$ as the set of feasible solutions satisfying the additional local branching constraint:

$$\Delta(x, \bar{x}) := \sum_{i \in \bar{O}} (1 - x_i) + \sum_{i \in B \setminus \bar{O}} x_i \leq k$$

Δ counts
number of flips

Partition at the branching node:

$\Delta(x, \bar{x}) \leq k$ (left branching) **or** $\Delta(x, \bar{x}) \geq k + 1$ (right branching)


1
$\Delta(x, \bar{x}^1) \leq k$
$\Delta(x, \bar{x}^1) \geq k+1$
2
3
T
$\Delta(x, \bar{x}^2) \leq k$
$\Delta(x, \bar{x}^2) \geq k+1$
improved solution $\bar{x}^2$
4
5
T
$\Delta(x, \bar{x}^3) \leq k$
$\Delta(x, \bar{x}^3) \geq k+1$
improved solution $\bar{x}^3$
6
7
T
T
no improved solution

- 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

[Desaulniers, Desrosiers, Dumas, Solomon and Soumis, 1997]

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

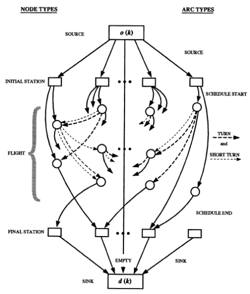
$$\sum_{l \in S_t} (o_{tp}^l - d_{tp}^l) x_t^l = 0 \qquad \forall t \in T;\ p \in P$$

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

- 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.



$$\text{Maximize} \sum_{k \in K} \sum_{(i,j) \in A^k} c_{ij}^k X_{ij}^k \tag{8}$$

subject to:

$$\sum_{k \in K} \sum_{j:(i,j) \in A^k} X_{ij}^k = 1 \quad \forall i \in N, \tag{9}$$

$$\sum_{i:(i,s) \in NS_2^k} X_{is}^k - \sum_{j:(s,j) \in S_1N^k} X_{sj}^k = 0 \quad \forall k \in K, \forall s \in S^k, \tag{10}$$

$$\sum_{s \in S_1^k} X_{o(k),s}^k + X_{o(k),d(k)}^k = n^k \quad \forall k \in K, \tag{11}$$

$$\sum_{i:(i,j) \in A^k} X_{ij}^k - \sum_{i:(j,i) \in A^k} X_{ji}^k = 0$$

$$\forall k \in K, \forall j \in V^k \backslash \{o(k), d(k)\}, \tag{12}$$

$$\sum_{s \in S_2^k} X_{s,d(k)}^k + X_{o(k),d(k)}^k = n^k \quad \forall k \in K, \tag{13}$$

$$X_{ij}^k \geq 0 \quad \forall k \in K, \forall (i,j) \in A^k, \tag{14}$$

$$a_i^k \leq T_i^k \leq b_i^k \quad \forall k \in K, \forall i \in V^k, \tag{15}$$

$$X_{ij}^k (T_i^k + d_{ij}^k - T_j^k) \leq 0 \quad \forall k \in K, \forall (i,j) \in A^k, \tag{16}$$

$$X_{ij}^k \text{ integer} \quad \forall k \in K, \forall (i,j) \in A^k. \tag{17}$$

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 21

Timetabling in Transportation

Marco Chiarandini

Outline

1. Transportation Timetabling
   Train Timetabling

Outline

1. Transportation Timetabling
   Train Timetabling

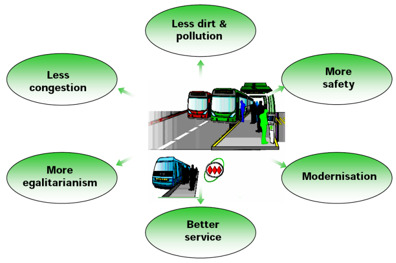
Planning problems in public transport

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

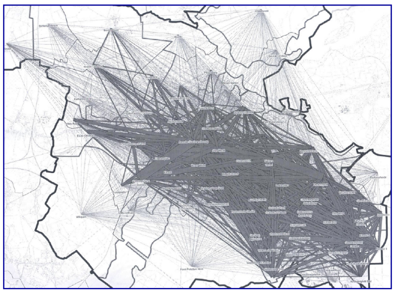
↑ ↑

Master Schedule $\xrightarrow{\text{Dynamic Management}}$ Conflict resolution

[Borndörfer, Grötschel, Pfetsch, 2005, ZIB-Report 05-22]

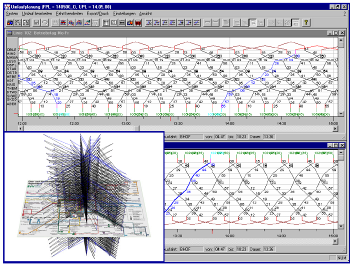


[Borndörfer, Liebchen, Pfetsch, course 2006, TU Berlin]



[Borndörfer, Liebchen, Pfetsch, course 2006, TU Berlin]

Time-space diagram



[Borndörfer, Liebchen, Pfetsch, course 2006, TU Berlin]

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 in T_0 two "dummy trains" as first and last trains

Step 2 (Select an Unscheduled Train) Select the next train k through the train selection priority rule

Step 3 (Set up and preprocess the MIP) Include train k in 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, add 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.

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 23

Workforce Scheduling

Marco Chiarandini

Outline

1. Transportation Timetabling

2. Workforce Scheduling
   - Crew Scheduling and Rostering
   - Employee Timetabling
     - Shift Scheduling
     - Nurse Scheduling

Outline

1. Transportation Timetabling

2. Workforce Scheduling
   - Crew Scheduling and Rostering
   - Employee Timetabling
     - Shift Scheduling
     - Nurse Scheduling

Periodic Event Scheduling Problem

Blackboard

Outline

1. Transportation Timetabling

2. Workforce Scheduling
   - Crew Scheduling and Rostering
   - Employee Timetabling
     - Shift Scheduling
     - Nurse Scheduling

Workforce Scheduling

Overview

A note on terminology
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.

Workforce Scheduling

Overview

Workforce Scheduling:

1. Crew Scheduling and Rostering
2. Employee Timetabling

1. Crew Scheduling and Rostering is 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.)

   The peculiarity is finding logistically feasible assignments.

Workforce Scheduling

Overview

2. 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.

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

Crew Scheduling

Input:

- A set of flight legs (departure, arrival, duration)
- A set of crews

Output: A subset of flights feasible for each crew

How do we solve it?

Set partitioning or set covering??

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

Shift Scheduling

Creating daily shifts:

- roster made of m time intervals not necessarily identical
- during each period, b_i personnel is required
- n different shift patterns (columns of matrix A)

$$\begin{aligned} \min\quad & c^T x \\ st\quad & Ax \geq b \\ & x \geq 0 \text{ and integer} \end{aligned}$$

(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.

$$\begin{aligned} \min\quad & cx \\ st\quad & \begin{bmatrix} 1&0&0&1&1&1&1\\ 1&1&0&0&1&1&1\\ 1&1&1&0&0&1&1\\ 1&1&1&1&0&0&1\\ 1&1&1&1&1&0&0\\ 0&1&1&1&1&1&0\\ 0&0&1&1&1&1&1 \end{bmatrix} x \geq b \qquad \text{(P)} \\ & x \geq 0 \text{ and integer} \end{aligned}$$

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 $\max\{cx : Ax \leq b, x \in \mathbf{R}^m_+\}$ 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

Recognizing total unimodularity can be done in polynomial time (see [Schrijver, 1986])

Total Unimodular Matrices

Resume'

Basic examples:

Theorem

The $V \times E$-incidence matrix of a graph $G = (V, E)$ is totally unimodular if and only if G is bipartite

Theorem

The $V \times A$-incidence matrix of a directed graph $D = (V, A)$ is totally unimodular

Theorem

Let $D = (V, A)$ be a directed graph and let $T = (V, A_0)$ be a directed tree on V. Let M be the $A_0 \times A$ matrix defined by, for $a = (v, w) \in A$ and $a' \in A_0$

$$M_{a',a} := \begin{array}{rl} +1 & \textit{if the unique } v-w\textit{-path in } T \textit{ passes through } a' \textit{ forwardly;} \\ -1 & \textit{if the unique } v-w\textit{-path in } T \textit{ passes through } a' \textit{ backwardly;} \\ 0 & \textit{if the unique } v-w\textit{-path in } T \textit{ does not pass through } a' \end{array}$$

M is called network matrix and is totally unimodular.

Total Unimodular Matrices

Resume'

All totally unimodular matrices arise by certain compositions from network matrices and from certain 5×5 matrices [Seymour, 1980]. This decomposition can be tested in polynomial time.

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 is called an interval matrix and they can be shown to be network matrices by taking a directed path for the directed tree T

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 $O(m^2n)$ 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

$$x_1 + \ldots + x_n = \lfloor x'_1 + \ldots + x'_n \rfloor \qquad \text{(LP1)}$$

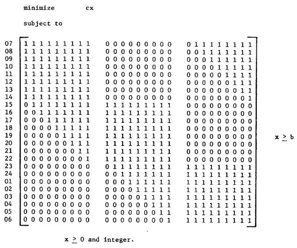
and

$$x_1 + \ldots + x_n = \lceil x'_1 + \ldots + x'_n \rceil \qquad \text{(LP2)}$$

From LP1 and LP2 an integral solution certainly arises (P)

Cyclic Staffing with Overtime

- Hourly requirements b_i
- Basic work shift 8 hours
- Overtime of up to additional 8 hours possible

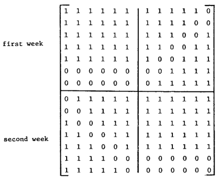


```
minimize        cx

subject to

07  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
08  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
09  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
10  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
11  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
12  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
13  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
14  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
15  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
16  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
17  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
18  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
19  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     x > b
20  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       -
21  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
22  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
23  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
24  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
01  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
02  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
03  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
04  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
05  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
06  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

                x > 0 and integer.
                  -
```

Days-Off Scheduling

- Guarantee two days-off each week, including every other weekend.

IP with matrix A:



$$\begin{array}{l} \\ \\ \\ \texttt{first week} \\ \\ \\ \\ \\ \\ \\ \texttt{second week} \\ \\ \\ \\ \end{array} \left[\begin{array}{cccccc|cccccc} 1&1&1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&1&0&0\\ 1&1&1&1&1&1&1&1&1&0&0&1\\ 1&1&1&1&1&1&1&1&0&0&1&1\\ 1&1&1&1&1&1&1&0&0&1&1&1\\ 0&0&0&0&0&0&0&0&1&1&1&1\\ 0&0&0&0&0&0&0&1&1&1&1&1\\ \hline 0&1&1&1&1&1&1&1&1&1&1&1\\ 0&0&1&1&1&1&1&1&1&1&1&1\\ 1&0&0&1&1&1&1&1&1&1&1&1\\ 1&1&0&0&1&1&1&1&1&1&1&1\\ 1&1&1&0&0&1&1&1&1&1&1&1\\ 1&1&1&1&0&0&0&0&0&0&0&0\\ 1&1&1&1&1&0&0&0&0&0&0&0 \end{array}\right]$$

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

$$\begin{array}{ll} \min & cx + c'x' \\ st & Ax + Ix' \geq b \\ & x, x' \geq 0 \text{ and integer} \end{array}$$

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

$$\begin{array}{ll} \min & cx + c'x' + c''(b - Ax - x') \\ st & Ax + Ix' \geq b \\ & x, x' \geq 0 \text{ and integer} \end{array}$$

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

$$\texttt{stretch-cycle}(y_{i\cdot} \mid (2, 3), (2, 2), (6, 6), P)$$
$$\forall i,\ P = \{(s, 0), (0, s) \mid s = 1, 2, 3\}$$

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

$$w_{y_{id},d} = i \qquad \forall i, d$$

$$y_{w_{sd},d} = s \qquad \forall s, d$$

The complete CP model

$$\text{Alldiff:} \left\{ \begin{array}{c} (w_{\cdot d}) \\ (y_{\cdot d}) \end{array} \right\},\ \text{all } d$$

$$\text{Cardinality: } (w_{\cdot\cdot} \mid (A, B, C, D), (5, 5, 5, 5), (6, 6, 6, 6))$$

$$\text{Nvalues: } (w_{s\cdot} \mid 1, 2),\ \text{all } s$$

$$\text{Stretch-cycle: } (y_{i\cdot} \mid (2, 3), (2, 2), (6, 6), P),\ \text{all } i$$

$$\text{Linear:} \left\{ \begin{array}{l} w_{y_{id}d} = i,\ \text{all } i \\ y_{w_{sd}d} = s,\ \text{all } s \end{array} \right\},\ \text{all } d$$

$$\text{Domains:} \left\{ \begin{array}{l} w_{sd} \in \{A, B, C, D\},\ s = 1, 2, 3 \\ y_{id} \in \{0, 1, 2, 3\},\ i = A, B, C, D \end{array} \right\},\ \text{all } d$$

Constraint Propagation:

- alldiff: matching
- nvalues: max flow
- stretch: poly-time dynamic programming
- index expressions $w_{y_{id}d}$ replaced by z and constraint: $\texttt{element}(y, x, z)$: z be equal to y-th variable in list $x_1, \ldots, x_m$

Search:

- branching by splitting domanins with more than one element
- first fail branching
- symmetry breaking:
  - employees are indistinguishable
  - shifts 2 and 3 are indistingushable
  - days can be rotated

  Eg: fix A, B, C to work $1, 2, 3$ resp. on sunday

Local search methods and metaheuristics are used if the problem has large scale. Procedures very similar to what we saw for employee timetabling.

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 24
Vehicle Routing

Marco Chiarandini

Outline

1. Vehicle Routing

2. Integer Programming

Outline

1. Vehicle Routing

2. Integer Programming

Problem Definition

Vehicle Routing: distribution of goods between depots and customers.

Delivery, collection, transportation.

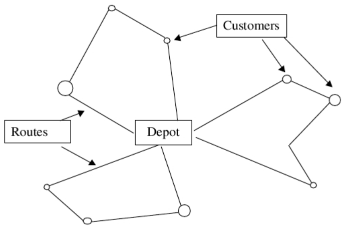
Examples: solid waste collection, street cleaning, school bus routing, dial-a-ride systems, transportation of handicapped persons, routing of salespeople and maintenance unit.

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 $V = \{0, \dots, n\}$ is a vertex set:
  - 0 is the depot.
  - $V' = V\backslash\{0\}$ 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) $(c_{ik} + c_{kj} \geq c_{ij} \qquad \forall\, i, j \in V)$
- 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 $R_1, \dots, R_m$ 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, $k = 1, \dots, K$
- 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

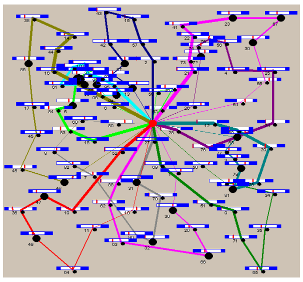
Vehicle Routing with Time Windows (VRPTW)

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
- Construction Heuristics
- Local Search
- Metaheuristics
- Hybridization with Constraint Programming

Outline

1. Vehicle Routing

2. Integer Programming

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

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 25

Vehicle Routing
Mathematical Programming

Marco Chiarandini

Outline

1. Integer Programming

Outline

1. Integer Programming

Basic Models

- arc flow formulation

  integer variables on the edges counting the number of time it is traversed
  one, two or three index variables

- set partitioning formulation

- multi-commodity network flow formulation for VRPTW

  integer variables representing the flow of commodities along the paths traveled by the vehicles and
  integer variables representing times

Two index arc flow formulation

$$
\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} && && (1)\\
\text{s.t.} \quad & \sum_{i \in V} x_{ij} = 1 && \forall j \in V \setminus \{0\} && (2)\\
& \sum_{j \in V} x_{ij} = 1 && \forall i \in V \setminus \{0\} && (3)\\
& \sum_{i \in V} x_{i0} = K && && (4)\\
& \sum_{j \in V} x_{0j} = K && && (5)\\
& \sum_{i \in S} \sum_{i \notin S} x_{ij} \geq r(S) && \forall S \subseteq V \setminus \{0\}, S \neq \emptyset && (6)\\
& x_{ij} \in \{0,1\} && \forall i,j \in V && (7)
\end{aligned}
$$

One index arc flow formulation

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e && && (8)\\
\text{s.t.} \quad & \sum_{e \in \delta(i)} x_e = 2 && \forall i \in V \setminus \{0\} && (9)\\
& \sum_{e \in \delta(0)} x_e = 2K && && (10)\\
& \sum_{e \in \delta S} x_e \geq 2r(S) && \forall S \subseteq V \setminus \{0\}, S \neq \emptyset && (11)\\
& x_e \in \{0,1\} && \forall e \notin \delta(0) && (12)\\
& x_e \in \{0,1,2\} && \forall e \in \delta(0) && (13)
\end{aligned}
$$

Three index arc flow formulation

$$
\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{j \in V} c_{ij} \sum_{k=1}^{K} x_{ijk} && && (14)\\
\text{s.t.} \quad & \sum_{k=1}^{K} y_{ik} = 1 && \forall i \in V \setminus \{0\} && (15)\\
& \sum_{k=1}^{K} y_{0k} = 1 && && (16)\\
& \sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} = y_{ik} && \forall i \in V, k = 1, \ldots, K && (17)\\
& \sum_{i \in V} d_i y_{ik} \leq C && \forall k = 1, \ldots, K && (18)\\
& \sum_{i \in S} \sum_{i \notin S} x_{ijk} \geq y_{hk} && \forall S \subseteq V \setminus \{0\}, h \in S, k = 1, 1, \ldots, K && (19)\\
& y_{ik} \in \{0,1\} && \forall i \in V, k = 1, \ldots, K && (20)\\
& x_{ijk} \in \{0,1\} && \forall i,j \in V, k = 1, \ldots, K && (21)
\end{aligned}
$$

What can we do with these integer programs?

- plug them into a commercial solver and try to solve them
- preprocess them
- determine lower bounds
  - solve the linear relaxation
  - combinatorial relaxations
    relax some constraints and get an easy solvable problem
  - Lagrangian relaxation
  - polyhedral study to tighten the formulations
- upper bounds via heuristics
- branch and bound
- cutting plane
- branch and cut
- column generation (via reformulation)
- branch and price
- Dantzig Wolfe decomposition
- upper bounds via heuristics

Combinatorial Relaxations

Lower bounding via combinatorial relaxations

Relax: capacity cut constraints (CCC)
or generalized subtour elimination constraints (GSEC) Consider both

ACVRP and SCVRP

- Relax CCC in 2-index formulation
  obtain a transportation problem
  Solution may contain isolated circuits and exceed vertex capacity

- Relax CCC in 1-index formulation
  obtain a b-matching problem

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e && \\
\text{s.t.} \quad & \sum_{e \in \delta(i)} x_e = b_i && \forall i \in V\{0\}\\
& x_e \in \{0,1\} && \forall e \notin \delta(0)\\
& x_e \in \{0,1,2\} && \forall e \in \delta(0)
\end{aligned}
$$

Solution has same problems as above

- relax in two index flow formulation:

$$
\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} && \\
\text{s.t.} \quad & \sum_{i \in V} x_{ij} = 1 && \forall j \in V \setminus \{0\}\\
& \sum_{j \in V} x_{ij} = 1 && \forall i \in V \setminus \{0\}\\
& \sum_{i \in V} x_{i0} = K && \\
& \sum_{j \in V} x_{0j} = K && \\
& \sum_{i \in S} \sum_{i \notin S} x_{ij} \geq r(S) 1 && \forall S \subseteq V \setminus \{0\}, S \neq \emptyset\\
& x_{ij} \in \{0,1\} && \forall i,j \in V
\end{aligned}
$$

K-shortest spanning arborescence problem

- relax in two index formulation

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e && \\
\text{s.t.} \quad & \sum_{e \in \delta(i)} x_e = 2 && \forall i \in V \setminus \{0\}\\
& \sum_{e \in \delta(0)} x_e = 2K && \\
& \sum_{e \in \delta S} x_e \geq 2r(S) && \forall S \subseteq V \setminus \{0\}, S \neq \emptyset\\
& x_e \in \{0,1\} && \forall e \notin \delta(0)
\end{aligned}
$$

  K-tree: min cost set of $n + K$ edges spanning the graph with degree 2K at the depot.

- Lagrangian relaxation of the sub tour constraints iteratively added after violation recognized by separation procedure.
  Subgradient optimization for the multipliers.

Branch and Cut

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e && && (22)\\
\text{s.t.} \quad & \sum_{e \in \delta(i)} x_e = 2 && \forall i \in V \setminus \{0\} && (23)\\
& \sum_{e \in \delta(0)} x_e = 2K && && (24)\\
& \sum_{e \in \delta S} x_e \geq 2 \lceil \frac{d(S)}{C} \rceil && \forall S \subseteq V \setminus \{0\}, S \neq \emptyset && (25)\\
& x_e \in \{0,1\} && \forall e \notin \delta(0) && (26)\\
& x_e \in \{0,1,2\} && \forall e \in \delta(0) && (27)
\end{aligned}
$$

Branch and Cut

- Let $LP(\infty)$ be linear relaxation of IP

- $z_{LP(\infty)} \leq z_{IP}$

- Problems if many constraints

- Solve $LP(h)$ instead and add constraints later

- If $LP(h)$ has integer solution then we are done, that is optimal
  If not, then $z_{LP(h)} \leq z_{LP(h+1)} \leq z_{LP(\infty)} \leq z_{IP}$

- Crucial step: separation algorithm given a solution to $LP(h)$ it tell us if some constraints are violated.

If the procedure does not return an integer solution we proceed by branch and bound

Problems with B&C:

i) no good algorithm for the separation phase
   it may be not exact in which case bounds relations still hold and we can go on with branching

ii) number of iterations for cutting phase is too high

iii) program unsolvable because of size

iv) the tree explodes

The main problem is (iv).
Worth trying to strengthen the linear relaxation by inequalities that although unnecessary in the integer formulation force the optimal solution of LP and IP to get closer. ➨ Polyhedral studies

Set Covering Formulation

$\mathcal{R} = \{1, 2, \ldots, R\}$ index set of routes

$$
a_{ir} = \begin{cases} 1 & \text{if costumer } i \text{ is served by } r \\ 0 & \text{otherwise} \end{cases}
$$

$$
x_r = \begin{cases} 1 & \text{if route } r \text{ is selected} \\ 0 & \text{otherwise} \end{cases}
$$

$$
\begin{aligned}
\min \quad & \sum_{r \in \mathcal{R}} c_r x_r && && (28)\\
\text{s.t.} \quad & \sum_{r \in \mathcal{R}} a_{ir} x_r \geq 1 && \forall i \in V && (29)\\
& \sum_{r \in \mathcal{R}} x_r \leq K && && (30)\\
& x_r \in \{0,1\} && \forall r \in \mathcal{R} && (31)\\
& && && (32)
\end{aligned}
$$

Solving the SCP integer program

Branch and bound

- Generate routes such that:
  - they are good in terms of cost
  - they reduce the potential for fractional solutions

- constraint branching [Ryan, Foster, 1981]

$$
\exists \text{ constraints } r_1, r_2 : 0 < \sum_{j \in J(r_1, r_2)} x_j < 1
$$

$J(r_1, r_2)$ all columns covering r_1, r_2 simultaneously. Branch on:

$$
\begin{array}{cc}
/ & \backslash \\
\sum_{j \in J(r_1, r_2)} x_j \leq 0 & \sum_{j \in J(r_1, r_2)} x_j \geq 1
\end{array}
$$

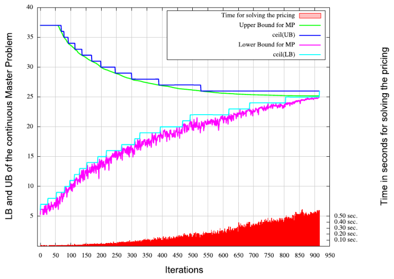
Solving the SCP linear relaxation

Column Generation Algorithm

```
Step 1  Generate an initial set of columns $\mathcal{R}'$
Step 2  Solve problem $P'$ and get optimal primal variables, $\bar{x}$, and
        optimal dual variables, $(\bar{\pi}, \bar{\theta})$
Step 3  Solve problem CG, or identify routes $r \in \mathcal{R}$ satisfying
        $\bar{c}_r < 0$
Step 4  For every $r \in \mathcal{R}$ with $\bar{c}_r < 0$ add the column $r$ to $\mathcal{R}'$ and
        go to Step 2
Step 5  If no routes $r$ have $\bar{c}_r < 0$, i.e., $\bar{c}_{min} \geq 0$ then stop.
```

In most of the cases we are left with a fractional solution

Convergence in CG



[plot by Stefano Gualandi, Milan Un.]

Solving the SCP integer program:

- cutting plane
- branch and price

Cutting Plane Algorithm

Step 1 Generate an initial set $\mathcal{R}'$ of columns

Step 2 Solve, using column generation, the problem P' (linear programming relaxation of P)

Step 3 If the optimal solution to P' is integer stop.
Else, generate cutting plane separating this fractional solution.
Add these cutting planes to the linear program P'

Step 4 Solve the linear program p'. Goto Step 3.

Is the solution to this procedure overall optimal?

Cuts

Intersection graph $G = (V, E)$ where V are the routes and E is made by the links between routes that intercept
Independence set in G is a collection of routes where each customer is visited only once.

Clique constraints

$$\sum_{r \in K} \bar{x}_r \leq 1 \qquad \forall \text{ cliques } K \text{ of } G$$

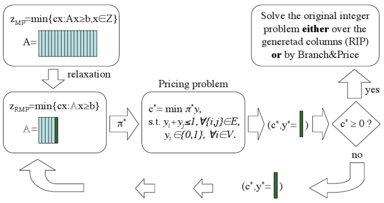
Cliques searched heuristically

Odd holes

Odd hole: odd cycle with no chord

$$\sum_{r \in H} \bar{x}_r \leq \frac{|H| - 1}{2} \qquad \forall \text{ odd holes } H$$

Generation via layered graphs



[illustration by Stefano Gualandi, Milan Un.]
(the pricing problem is for a GCP)

Branch and price

- it must be possible to incorporate information on the node in the column generation procedure
- easy to incorporate $x_r = 1$, just omit nodes in S_r from generation; but not clear how to impose $x_r = 0$.
- different branching: on the edges: $x_{ij} = 1$ then in column generation set $c_{ij} = -\infty$; if $x_{ij} = 0$ then $c_{ij} = \infty$

Implementation details

- throw out from LP columns that have not been basic for a long time
- good procedures to generate good columns
- generate columns that are disjoint, collectively exhaustive and of minimal cost

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 26
Vehicle Routing
Mathematical Programming

Marco Chiarandini

Outline

1. Integer Programming

Outline

1. Integer Programming

VRPTW

$$\begin{aligned}
\min \quad & \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}x_{ijk} && && (1)\\
\text{s.t.} \quad & \sum_{k\in K}\sum_{(i,j)\in\delta^+(i)} x_{ijk} = 1 && \forall i\in V && (2)\\
& \sum_{(i,j)\in\delta^+(0)} x_{ijk} = \sum_{(i,j)\in\delta^-(0)} x_{ijk} = 1 && \forall k\in K && (3)\\
& \sum_{(i,j)\in\delta^-(i)} x_{jik} - \sum_{(i,j)\in\delta^+(i)} x_{ijk} = 0 && i\in V, k\in K && (4)\\
& \sum_{(i,j)\in A} d_i x_{ijk} \le C && \forall k\in K && (5)\\
& x_{ijk}(w_{ik}+t_{ij}) \le w_{jk} && \forall k\in K, (i,j)\in A && (6)\\
& a_i \le w_{ik} \le b_i && \forall k\in K, i\in V && (7)\\
& x_{ijk}\in\{0,1\} && && (8)
\end{aligned}$$

Pre-processing

- 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
- Time windows reduction
  - $[a_i, b_i] \leftarrow [\max\{a_0 + t_{0i}, a_i\}, \min\{b_{n+1} - t_{i,n+1}, b_i\}]$ why?

- Time windows reduction:
  - Iterate over the following rules until no one applies anymore:
    1) Minimal arrival time from predecessors:
    $$a_l = \max\left\{a_l, \min\left\{b_l, \min_{(i,l)}\{a_i + t_{il}\}\right\}\right\}.$$
    2) Minimal arrival time to successors:
    $$a_l = \max\left\{a_l, \min\left\{b_l, \min_{(l,j)}\{a_j - t_{lj}\}\right\}\right\}.$$
    3) Maximal departure time from predecessors:
    $$b_l = \min\left\{b_l, \max\left\{a_l, \max_{(i,l)}\{b_i + t_{il}\}\right\}\right\}.$$
    4) Maximal departure time to successors:
    $$b_l = \min\left\{b_l, \max\left\{a_l, \max_{(l,j)}\{b_j - t_{lj}\}\right\}\right\}.$$

Lower Bounds

- Combinatorial relaxation
  reduce to network flow problem
- Lagrangian relaxation
  not very good because easy to not satisfy the capacity and time windows constraints

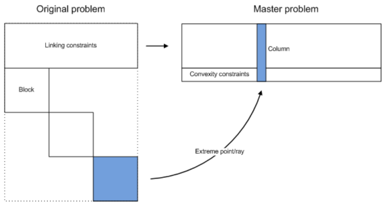
Dantzig Wolfe Decomposition

The VRPTW has the structure:

$$\begin{aligned}
\min \quad & c^k x^k \\
& \sum_{k\in K} A^k x^k \le b \\
& D^k x^k \le d^k && \forall k\in K\\
& x^k \in \mathbb{Z} && \forall k\in K
\end{aligned}$$

Dantzig Wolfe Decomposition

Illustrated with matrix blocks



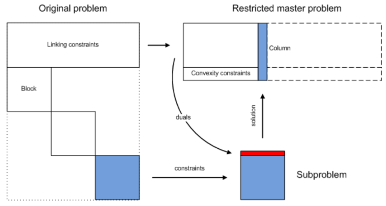
[illustration by Simon Spoorendonk, DIKU]

Linking constraint in VRPTW is $\sum_{k\in K}\sum_{(i,j)\in\delta^+(i)} x_{ijk} = 1, \quad \forall i$. The description of the block $D^k x^k \le d^k$ is all the rest:

$$\begin{aligned}
& \sum_{(i,j)\in A} d_i x_{ij} \le C && && (9)\\
& \sum_{j\in V} x_{0j} = \sum_{i\in V} x_{i,n+1} = 1 && && (10)\\
& \sum_{i\in V} x_{ih} - \sum_{j\in V} x_{hj} = 0 && \forall h\in V && (11)\\
& w_i + t_{ij} - M_{ij}(1 - x_{ij}) \le w_j && \forall (i,j)\in A && (12)\\
& a_i \le w_i \le b_i && \forall i\in V && (13)\\
& x_{ij}\in\{0,1\} && && (14)
\end{aligned}$$

where we omitted the index k because, by the assumption of homogeneous fleet, all blocks are equal.

Dantzig Wolfe Decomposition



[illustration by Simon Spoorendonk, DIKU]

Master problem

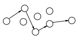
A Set Partitioning Problem

$$\begin{aligned}
\min \quad & \sum_{p\in\mathcal{P}} c_{ij}\alpha_{ijp}\lambda_p && && (15)\\
& \sum_{p\in\mathcal{P}}\sum_{(i,j)\in\delta^+(i)} \alpha_{ijp}\lambda_p = 1 && \forall i\in V && (16)\\
& \lambda_p = \{0,1\} && \forall p\in\mathcal{P} && (17)
\end{aligned}$$

where $\mathcal{P}$ is the set of valid paths and $\alpha_{ijp} = \begin{cases} 0 & \text{if } (i,j)\notin p\\ 1 & \text{otherwise}\end{cases}$

Subproblem

Elementary Shortest Path Problem with Resource Constraints (ESPPRC)

- arcs modified with duals (possible negative costs), NP-hard
- find shortest path without violating resource limits 

Subproblem

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in A} \hat{c}_{ij}x_{ij} && && (18)\\
\text{s.t.} \quad & \sum_{(i,j)\in A} d_i x_{ij} \le C && && (19)\\
& \sum_{j\in V} x_{0j} = \sum_{i\in V} x_{i,n+1} = 1 && && (20)\\
& \sum_{i\in V} x_{ih} - \sum_{j\in V} x_{hj} = 0 && \forall h\in V && (21)\\
& w_i + t_{ij} - M_{ij}(1 - x_{ij}) \le w_j && \forall (i,j)\in A && (22)\\
& a_i \le w_i \le b_i && \forall i\in V && (23)\\
& x_{ij}\in\{0,1\} && && (24)
\end{aligned}$$

Subproblem

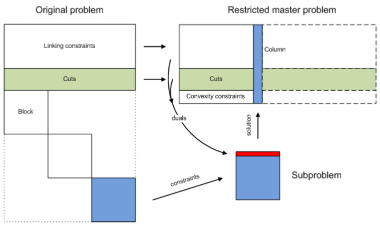
Solution Approach:

- Solved by dynamic programming. Algorithms maintain labels at vertices and remove dominated labels. Domination rules are crucial.
- relaxing and allowing cycles the problem can be sovled in pseudo-polynomial time.
  Negative cycles are however limited by the resource constraints
- optimal solution has only elementary routes if triangle inequality holds.
  Otherwise post-process by cycle elimination procedures

For details see chp. 2 of [B11]

Branch and Bound

Cuts in the original three index problem formulation (before DWD)



[illustration by Simon Spoorendonk, DIKU]

Branching

- branch on $\sum_k x_{ijk}$
  choose a candidate not close to 0 or 1
  $\max c_{ij} \min\{x_{ijk}, 1 - x_{ijk})$
- branch on time windows
  split time windows s.t. at least one route becomes infeasible
  compute $[l_i^r, u_i^r]$ (earliest latest) for the current fractional flow
  $L^i = \max_{\text{fract. routes } r} \{l_i^r\} \qquad \forall i\in V$
  $U^i = \max_{\text{fract. routes } r} \{u_i^r\} \qquad \forall i\in V$
  if $L_i > U_i$ ➨ at least two routes have disjoint feasibility intervals

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 27

Vehicle Routing Heuristics

Marco Chiarandini

Outline

1. Construction Heuristics
   - Construction Heuristics for CVRP
   - Construction Heuristics for VRPTW
2. Improvement Heuristics
3. Metaheuristics
4. Constraint Programming for VRP

Outline

1. Construction Heuristics
   - Construction Heuristics for CVRP
   - Construction Heuristics for VRPTW
2. Improvement Heuristics
3. Metaheuristics
4. Constraint Programming for VRP

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 spanning tree heuristic
  - Christofides' heuristics

(But recall! Concorde: http://www.tsp.gatech.edu/)

[Bentley, 1992]

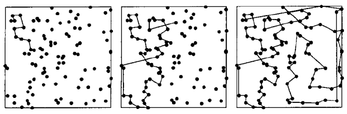


Figure 1. The Nearest Neighbor heuristic.

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]

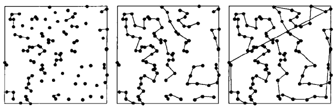


Figure 5. The Multiple Fragment heuristic.

Add the cheapest edge provided it does not create a cycle.

[Bentley, 1992]

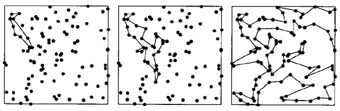


Figure 8. The Nearest Addition heuristic.

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]

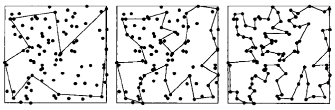


Figure 11. The Farthest Addition heuristic.

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 effective 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]

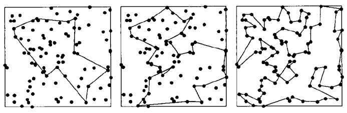


Figure 14. The Random Addition heuristic.

[Bentley, 1992]

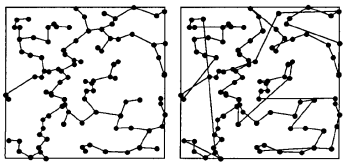


Figure 18. The Minimum Spanning Tree heuristic.

1. Find a minimum spanning tree $O(N^2)$
2. Append the nodes in the tour in a depth-first, inorder traversal

Running time $O(N^2)$ $A = MST(I)/OPT(I) \leq 2$

[Bentley, 1992]

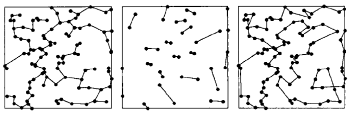
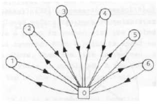


Figure 19. Christofides' heuristic.

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 $O(N^3)$ $A = CH(I)/OPT(I) \leq 3/2$

Construction Heuristics Specific for VRP



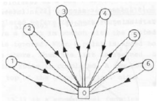
Clarke-Wright Saving Heuristic (1964)

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

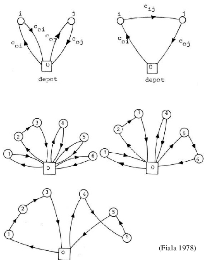


Clarke-Wright Saving Heuristic (1964)

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



(Fiala 1978)

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

$$\alpha(i,k,j) = c_{ik} + c_{kj} - \lambda c_{ij}$$

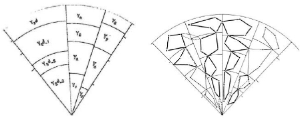
$$\beta(i,k,j) = \mu c_{0k} - \alpha(i,k,j)$$

1. construct emerging route $(0, k, 0)$
2. compute for all k unrouted the feasible insertion cost:

$$\alpha^*(i_k, k, j_k) = \min_p \{\alpha(i_p, k, i_{p+1})\}$$

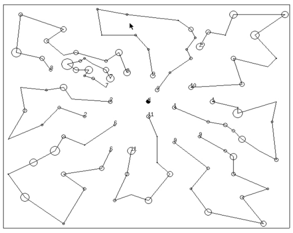
if no feasible insertion go to 1 otherwise choose k^* such that

$$\beta^*(i^*_k, k^*, j^*_k) = \max_k \{\beta(i_k, k, j_k)\}$$



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, 1983]

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 ones need to have K fixed a priori?

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

- Let's assume waiting is allowed and s_i indicates service times
- $b_i = \max\{e_i, b_j + s_j + t_{ji}\}$ begin of service
- insertion of u: $(i_0, i_1, \ldots, i_p, \mathbf{u}, i_{p+1}, \ldots, i_m)$
- $PF_{i_{p+1}} = b^{new}_{i_{p+1}} - b_{i_{p+1}} \geq 0$ push forward
- $PF_{i_{r+1}} = \max\{0, PF_{i_r} - w_{i_{r+1}}\}, \quad p \leq r \leq m-1$

Theorem

The insertion is feasible if and only if:

$$b_u \leq l_u \quad \textit{and} \quad PF_{i_r} + b_{i_r} \leq l_{i_r} \quad \forall p < r \leq m$$

Check vertices k, $u \leq k \leq m$ sequentially.

- if $b_k + PF_k > l_k$ then stop: the insertion is infeasible
- if $PF_k = 0$ then stop: the insertion is feasible

Outline

1. Construction Heuristics
   Construction Heuristics for CVRP
   Construction Heuristics for VRPTW
2. Improvement Heuristics
3. Metaheuristics
4. Constraint Programming for VRP

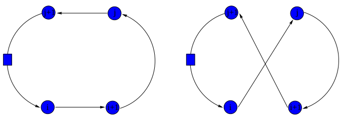
Local Search for CVRP and VRPTW

- 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

$$\{i, i+1\}\{j, j+1\} \longrightarrow \{i, j\}\{i+1, j+1\}$$

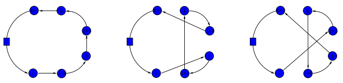


$O(n^2)$ possible exchanges
One path is reversed

Intra-route Neighborhoods

3-opt

$$\{i, i+1\}\{j, j+1\}\{k, k+1\} \longrightarrow \ldots$$

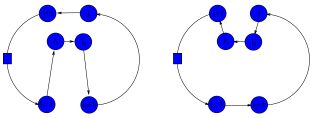


$O(n^3)$ possible exchanges
Paths can be reversed

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

Inter-route Neighborhoods

[Savelsbergh, ORSA (1992)]

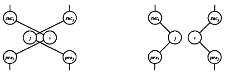


Figure 6. The exchange neighborhood.

Inter-route Neighborhoods

[Savelsbergh, ORSA (1992)]

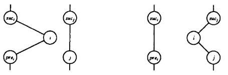


Figure 5. The relocate neighborhood.

Inter-route Neighborhoods

[Savelsbergh, ORSA (1992)]

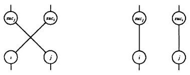


Figure 7. The cross neighborhood.

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.

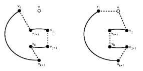


Figure 1. Type I insertion of vertex v between v_i and v_j.

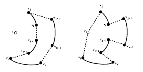


Figure 2. Type II insertion of vertex v between v_i and v_j.

Efficient Implementation

Intra-route

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

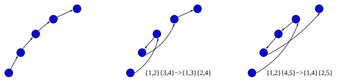
(Savelsbergh 1985) shows how to verify constraints in constant time
Search strategy + Global variables

⇓

$O(n^k)$ 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

Outline

1. Construction Heuristics
   Construction Heuristics for CVRP
   Construction Heuristics for VRPTW
2. Improvement Heuristics
3. Metaheuristics
4. Constraint Programming for VRP

Metaheuristics

Many and fancy examples, but first thing to try:

- Variable Neighborhood Search + Iterated greedy

Basic Variable Neighborhood Descent (BVND)

```
Procedure VND
input : $\mathcal{N}_k$, $k = 1, 2, \ldots, k_{max}$, and an initial solution $s$
output: a local optimum $s$ for $\mathcal{N}_k$, $k = 1, 2, \ldots, k_{max}$
$k \leftarrow 1$
repeat
    $s' \leftarrow$ FindBestNeighbor($s$,$\mathcal{N}_k$)
    if $g(s') < g(s)$ then
        $s \leftarrow s'$
        $(k \leftarrow 1)$
    else
        $k \leftarrow k+1$
until $k = k_{max}$ ;
```

Variable Neighborhood Descent (VND)

```
Procedure VND
input : $\mathcal{N}_k$, $k = 1, 2, \ldots, k_{max}$, and an initial solution $s$
output: a local optimum $s$ for $\mathcal{N}_k$, $k = 1, 2, \ldots, k_{max}$
$k \leftarrow 1$
repeat
    $s' \leftarrow$ IterativeImprovement($s$,$\mathcal{N}_k$)
    if $g(s') < g(s)$ then
        $s \leftarrow s'$
        $(k \leftarrow 1)$
    else
        $k \leftarrow k+1$
until $k = k_{max}$ ;
```

- Final solution is locally optimal w.r.t. all neighborhoods
- First improvement may be applied instead of best improvement
- Typically, order neighborhoods from smallest to largest
- If iterative improvement algorithms II_k, $k = 1, \ldots, k_{max}$ are available as black-box procedures:
  - order black-boxes
  - apply them in the given order
  - possibly iterate starting from the first one
  - order chosen by: *solution quality* and *speed*

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

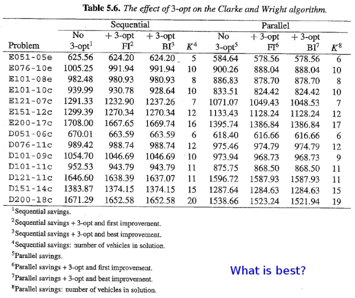


Table 5.6. *The effect of 3-opt on the Clarke and Wright algorithm.*

	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?

Iterated Greedy

Key idea: use the VRP cosntruction heuristics

- 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 $s$
    apply subsidiary iterative improvement procedure (eg, VNS)
    based on acceptance criterion,
        keep $s$ or revert to $s := r$
```

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

$$r_{ij} = \frac{1}{D}\left(d'(i,j) + d'(i,j+n) + d'(i+n,j) + d'(i+n,j+n)\right)$$

- 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 procedures:

- Greedy (cheapest insertion)
- Max regret:
  Δf_i^q due to insert i into its best position in its q^{th} best route
  $i = \arg\max(\Delta f_i^2 - \Delta f_i^1)$
- Constraint Programming (max 20 costumers)

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

1. Construction Heuristics
   Construction Heuristics for CVRP
   Construction Heuristics for VRPTW

2. Improvement Heuristics

3. Metaheuristics

4. Constraint Programming for VRP

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.

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
```

DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 28
Rich Vehicle Routing Problems

Marco Chiarandini

Outline

1. A Uniform Model

2. Other Variants of VRP

Outline

1. A Uniform Model

2. Other Variants of VRP

Efficient Local Search

Blackboard [Irnich 2008].

Outline

1. A Uniform Model

2. Other Variants of VRP

Rich VRP

Definition

Rich Models are non idealized models that represetn the appliucation at hand in an adequate way by including all important optimization criteria, constraints and preferences [Hasle et al., 2006]

Solution

- Exact methods are often impractical:
  - instancs are too large
  - decision support systems require short response times
- Metaheuristics based on local search components are mostly used

VRP with Backhauls

Further Input from CVRP:

- a partition of customers:
  $L = \{1, \dots, n\}$ Lineahaul customers (deliveries)
  $B = \{n+1, \dots, n+m\}$ 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.

VRP with Pickup and Delivery

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 $V = \{1, 2, \dots, n\}$ and $V_0 = \{n+1, \dots, n+m\}$
Route i defined by $R_i = \{l, 1, \dots, l\}$

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)