

## Mathematical Programming

especially Integer Linear Programming  
and Mixed Integer Programming

600.325/425 Declarative Methods - J. Eisner

1

## Transportation Problem in ECLiPSe

- Vars = [A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4];
  - Vars :: 0.0..inf, *Can't recover transportation costs by sending negative amounts*
  - A1 + A2 + A3 + A4 \$=< 500, % supply constraints
  - B1 + B2 + B3 + B4 \$=< 300,
  - C1 + C2 + C3 + C4 \$=< 400, *Production capacity of producer "C"*
  - A1 + B1 + C1 \$= 200, % demand constraints
  - A2 + B2 + C2 \$= 400,
  - A3 + B3 + C3 \$= 300,
  - A4 + B4 + C4 \$= 100, *Total amount that must be sent to consumer "4"*
  - optimize(min(10\*A1 + 8\*A2 + 5\*A3 + 9\*A4 + 7\*B1 + 5\*B2 + 5\*B3 + 3\*B4 + 11\*C1 + 10\*C2 + 8\*C3 + 7\*C4), Cost), *Satisfiable?*
- Transport cost per unit*

600.325/425 Declarative Methods - J. Eisner

2

example adapted from ECLiPSe website

## Mathematical Programming in General

- Here are some variables:
- Vars = [A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4];
- And some hard constraints on them:
- Vars :: 0.0..inf,
- A1 + A2 + A3 + A4 \$=< 500, % supply constraints
- B1 + B2 + B3 + B4 \$=< 300,
- C1 + C2 + C3 + C4 \$=< 400,
- A1 + B1 + C1 \$= 200, % demand constraints
- A2 + B2 + C2 \$= 400,
- A3 + B3 + C3 \$= 300,
- A4 + B4 + C4 \$= 100,
- Find a satisfying assignment that makes this objective function as large or small as possible:
- 10\*A1 + 8\*A2 + 5\*A3 + 9\*A4 + 7\*B1 + 5\*B2 + 5\*B3 + 3\*B4 + 11\*C1 + 10\*C2 + 8\*C3 + 7\*C4

## Mathematical Programming in General

- Here are some variables:
  - And some hard constraints on them:
- what kind of constraints?**
- Find a satisfying assignment that makes this objective function as large or small as possible:

**what kind of function?**

## Types of Mathematical Programming

## Types of Mathematical Programming

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
integer linear prog. (ILP)	integer	linear inequalities	linear function
mixed integer prog. (MIP)	int&real	linear inequalities	linear function
quadratic programming	real	linear inequalities	quadratic function (hopefully convex)
semidefinite prog.	real	linear inequalities +semidefiniteness	linear function
quadratically constrained programming	real	quadratic inequalities	linear or quadratic function
convex programming	real	convex region	convex function
nonlinear programming	real	any	any

## Linear Programming (LP)

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function

## Linear Programming in 2 dimensions

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function

2 variables:  
feasible region is a convex polygon

boundary of feasible region comes from the constraints

For comparison here's a non-convex polygon

image adapted from Keely L. Croxton

## Linear Programming in $n$ dimensions

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function

3 variables:  
feasible region is a convex polyhedron

( $n-1$ )-dimensional facet, imposed by a linear constraint that is a full ( $n-1$ )-dim hyperplane

In general case of  $n$  dimensions, the word is polytope

image adapted from Keely L. Croxton

## Linear Programming in 2 dimensions

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function

"level sets" of the objective  $x+y$  (sets where it takes a certain value)

$x+y = 4$        $x+y = 5$        $x+y = 6$        $x+y = 7$

Optimal Solution

images adapted from Keely L. Croxton

## Linear Programming in $n$ dimensions

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function

here level set is a plane (in general, a hyperplane)

If an LP optimum is finite, it can *always* be achieved at a corner ("vertex") of the feasible region.

Optimal Solution

(Can there be infinite solutions? Multiple solutions?)

image from Keely L. Croxton

## Simplex Method for Solving an LP

At every step, move to an adjacent vertex that improves the objective.

Start Here

End up at Optimal Solution

images thanks to Keely L. Croxton and Rex Kincaid

## Integer Linear Programming (ILP)

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
integer linear prog. (ILP)	integer	linear inequalities	linear function

Function to maximize:  $f(x,y) = 6 + 6x - 5y$   
 Optimum LP solution  $(x,y) = (2.8, 3.3)$   
 Pareto optima:  $(0,4), (2,3), (3,2), (4,1)$   
 Optimum ILP solution  $(x,y) = (0,4)$

image adapted from Jop Sibeyn

## Mixed Integer Programming (MIP)

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
integer linear prog. (ILP)	integer	linear inequalities	linear function
mixed integer prog. (MIP)	int&real	linear inequalities	linear function

We'll be studying MIP solvers.  
 SCIP mainly does MIP though it goes a bit farther.

## Quadratic Programming

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
quadratic programming	real	linear inequalities	quadratic function (hopefully convex)

solution no longer at a vertex

level sets of  $x^2+y^2$  (try to minimize)  
 level sets of  $(x-2)^2+(y-2)^2$  (try to minimize)  
 same, but maximize (no longer convex)

at a vertex but how to find it? local max

## Quadratic Programming

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
quadratic programming	real	linear inequalities	quadratic function (hopefully convex)

Note: On previous slide, we saw that the level sets of our quadratic objective  $x^2+y^2$  were circles.

In general (in 2 dimensions), the level sets of a quadratic function will be conic sections: ellipses, parabolas, hyperbolae. E.g.,  $x^2-y^2$  gives a hyperbola.

The n-dimensional generalizations are called **quadratics**.

Reason, if you're curious: The level set is  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = \text{const}$   
 Equivalently,  $Ax^2 + Bxy + Cy^2 = -Dx - Ey + (\text{const} - F)$   
 Equivalently,  $(x,y)$  is in set if  $\exists z$  with  $z = Ax^2 + Bxy + Cy^2$  and  $z = -Dx - Ey + (\text{const} - F)$   
 Thus, consider all  $(x,y,z)$  points where a right cone intersects a plane

## Semidefinite Programming

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
quadratic programming	real	linear inequalities	quadratic function (hopefully convex)
semidefinite prog.	real	linear inequalities +semidefiniteness	linear function

## Quadratically Constrained Programming

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
quadratic programming	real	linear inequalities	quadratic function (hopefully convex)
quadratically constrained programming	real	quadratic inequalities	linear or quadratic function

curvy feasible region

linear objective in this case, so level sets are again hyperplanes, but optimum is not at a vertex

## Convex Programming

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
convex programming	real	convex region	convex function (to be minimized)

Non-convexity is hard because it leads to disjunctive choices in optimization (hence backtracking search).

- Infeasible in middle of line: which way to go?
- Objective too large in middle of line: which way to go?

## Convex Programming

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
convex programming	real	convex region	convex function (to be minimized)

Can minimize a convex function by methods such as gradient descent, conjugate gradient, or (for non-differentiable functions) Powell's method. **No local optimum problem.**

Here we want to generalize to minimization within a convex region. **Still no local optimum problem.** Can use subgradient or interior point methods, etc.

Note: If instead you want to maximize within a convex region, the solution is at least known to be on the boundary, if the region is compact (i.e., bounded).

## Nonlinear Programming

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
convex programming	real	convex region	convex function
nonlinear programming	real	any	any

Non-convexity is hard because it leads to disjunctive choices in optimization.

Here in practice one often falls back on methods like simulated annealing.

To get an exact solution, you can try backtracking search methods that recursively divide up the space into regions. (Branch-and-bound, if you can compute decent optimistic bounds on the best solution within a region, e.g., by linear approximations.)

## Types of Mathematical Programming

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
integer linear prog. (ILP)	integer	linear inequalities	linear function
mixed integer prog. (MIP)	int&real	linear inequalities	linear function
quadratic programming	real	linear inequalities	quadratic function (hopefully convex)
semidefinite prog.	real	linear inequalities + semidefiniteness	linear function
quadratically constrained linear programming	real	quadratic inequalities	linear or quadratic function
convex programming	real	convex region	convex function
nonlinear programming	real	any	any

## Types of Mathematical Programming

Name	Vars	Constraints	Objective
constraint programming	discrete?	any	N/A
linear programming (LP)	real	linear inequalities	linear function
integer linear programming (ILP)	integer	linear inequalities	linear function
mixed integer programming (MIP)	int&real	linear inequalities	linear function
quadratic programming	real	linear inequalities	quadratic function (hopefully convex)
semidefinite programming	real	linear inequalities + semidefiniteness	linear function
quadratically constrained linear programming	real	quadratic inequalities	linear or quadratic function
convex programming	real	convex region	convex function
nonlinear programming	real	any	any

Lots of software available for various kinds of math programming!

Huge amounts of effort making it smart, correct, and fast - use it!

See the NEOS Wiki, the Decision Tree for Optimization Software, and the COIN-OR open-source consortium.

## Terminology

	Constraint Programming	Math Programming
input	formula / constraint system	model
	variable	variable
	constraint	constraint
output	MAX-SAT cost	objective
	assignment	program
	SAT	feasible
solver	UNSAT	infeasible
	programs	codes
	backtracking search	branching / branch & bound
	variable/value ordering	node selection strategy
	propagation	node preprocessing
	formula simplification	presolving
	(depth,breadth,best,...)-first	branching strategy

## Linear Programming in ZIMPL

## Formal Notation of Linear Programming

- $n$  variables  $x_1, x_2, \dots, x_n$
- max or min objective  $c_1x_1 + c_2x_2 + \dots + c_nx_n$
- $m$  linear inequality and equality constraints

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq c_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = c_2$$

Note: if a constraint refers to only a few of the vars, its other coefficients will be 0

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \geq c_m$$

## Formal Notation of Linear Programming

- $n$  variables  $x_1, x_2, \dots, x_n$
- max or min objective  $c_1x_1 + c_2x_2 + \dots + c_nx_n$
- $m$  linear inequality and equality constraints

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq c_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = c_2$$

Note: if a constraint refers to only a few of the vars, its other coefficients will be 0

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \geq c_m$$

## Formal Notation of Linear Programming

- $n$  variables  $x_1, x_2, \dots, x_n$
- max ~~or min~~ objective  $c_1x_1 + c_2x_2 + \dots + c_nx_n$
- $m$  linear inequality ~~and equality~~ constraints

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq c_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq c_2$$

⋮

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq c_m$$

- Can we simplify (much as we simplified SAT to CNF-SAT)?

## Formal Notation of Linear Programming

- $n$  variables  $x_1, x_2, \dots, x_n$
- objective: max  $\vec{c} \cdot \vec{x}$
- $m$  linear inequality constraints

$$A\vec{x} \leq \vec{b}$$

(where " $\leq$ " means that  $(\forall 1 \leq i \leq m) (A\vec{x})_i \leq b_i$ )

- Now we can use this concise matrix notation

## Formal Notation of Linear Programming

- $n$  variables  $x_1, x_2, \dots, x_n$
- objective: max  $\vec{c} \cdot \vec{x}$
- $m$  linear inequality constraints

$$A\vec{x} \leq \vec{b}$$

(where " $\leq$ " means that  $(\forall 1 \leq i \leq m) (A\vec{x})_i \leq b_i$ )

- Some LP folks also assume constraint  $\vec{x} \geq 0$ 
  - What if you want to allow  $x_3 < 0$ ? Just replace  $x_3$  everywhere with  $(x_{m+1} - x_{m+2})$  where  $x_{m+1}, x_{m+2}$  are new variables  $\geq 0$ .
  - Then solver can pick  $x_{m+1}, x_{m+2}$  to have either pos or neg diff.

## Strict inequalities?

- $n$  variables  $x_1, x_2, \dots, x_n$
- max or min objective  $c_1x_1 + c_2x_2 + \dots + c_nx_n$
- $m$  linear inequality and equality constraints

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq c_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = c_2$$

⋮

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \geq c_m$$

How about using strict  $>$  or  $<$ ?

But then you could say "min  $x_1$  subject to  $x_1 > 0$ ."

No well-defined solution, so can't allow this.

Instead, approximate  $x > y$  by  $x \geq y + 0.001$ .

## ZIMPL and SCIP

What little language and solver should we use?

Quite a few options ...

- Our little language for this course is **ZIMPL** (Koch 2004)
  - A free and extended dialect of AMPL = "A Mathematical Programming Language" (Fourer, Gay & Kernighan 1990)
  - Compiles into MPS, an unfriendly punch-card like format accepted by virtually all solvers
- Our solver for mixed-integer programming is **SCIP** (open source)
  - Our version of SCIP will
    1. read a ZIMPL file (\*.zpl)
    2. compile it to MPS
    3. solve using its own MIP methods
      - which in turn call an LP solver as a subroutine
        - our version of SCIP calls CLP (part of the COIN-OR effort)

## Transportation Problem in ECLiPSe

- Vars = {A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4};
- Vars : 0.0..inf, Can't recover transportation costs by sending negative amounts
- A1 + A2 + A3 + A4 \$=< 500, % supply constraints
- B1 + B2 + B3 + B4 \$=< 300,
- C1 + C2 + C3 + C4 \$=< 400, Production capacity of producer "C"
- A1 + B1 + C1 \$= 200, % demand constraints
- A2 + B2 + C2 \$= 400,
- A3 + B3 + C3 \$= 300,
- A4 + B4 + C4 \$= 100, Total amount that must be sent to consumer "4"
- optimize(min(10\*A1 + 8\*A2 + 5\*A3 + 9\*A4 + 7\*B1 + 5\*B2 + 5\*B3 + 3\*B4 + 11\*C1 + 10\*C2 + 8\*C3 + 7\*C4), Cost).

Transport cost per unit

example adapted from ECLiPSe website

600.325/425 Declarative Methods - J. Eisner

33

## Transportation Problem in ZIMPL

- var a1; var a2; var a3; var a4; Amount that producer "C" sends to consumer "4"
- var b1; var b2; var b3; var b4; Variables are assumed real and >= 0 unless declared otherwise
- var c1; var c2; var c3; var c4;
- subto supply\_a: a1 + a2 + a3 + a4 <= 500;
- subto supply\_b: b1 + b2 + b3 + b4 <= 300;
- subto supply\_c: c1 + c2 + c3 + c4 <= 400; Production capacity of producer "C"
- subto demand\_1: a1 + b1 + c1 == 200;
- subto demand\_2: a2 + b2 + c2 == 400;
- subto demand\_3: a3 + b3 + c3 == 300;
- subto demand\_4: a4 + b4 + c4 == 100; Total amount that must be sent to consumer "4"
- minimize cost: 10\*a1 + 8\*a2 + 5\*a3 + 9\*a4 + 7\*b1 + 5\*b2 + 5\*b3 + 3\*b4 + 11\*c1 + 10\*c2 + 8\*c3 + 7\*c4;

Blue strings are just your names for the constraints and the objective (for documentation and debugging)

Transport cost per unit

600.325/425 Declarative Methods - J. Eisner

34

## Transportation Problem in ZIMPL

- set Producer := {1..3}; Indexed variables (indexed by members of a specified set). Variables are assumed real and >= 0 unless declared otherwise
- set Consumer := {1 to 4};
- var send[Producer\*Consumer];
- subto supply\_a: sum <c> in Consumer: send[1,c] <= 500;
- subto supply\_b: sum <c> in Consumer: send[2,c] <= 300;
- subto supply\_c: sum <c> in Consumer: send[3,c] <= 400; Indexed summations
- subto demand\_1: sum <p> in Producer: send[p,1] == 200;
- subto demand\_2: sum <p> in Producer: send[p,2] == 400;
- subto demand\_3: sum <p> in Producer: send[p,3] == 300;
- subto demand\_4: sum <p> in Producer: send[p,4] == 100;
- minimize cost: 10\*send[1,1] + 8\*send[1,2] + 5\*send[1,3] + 9\*send[1,4] + 7\*send[2,1] + 5\*send[2,2] + 5\*send[2,3] + 3\*send[2,4] + 11\*send[3,1] + 10\*send[3,2] + 8\*send[3,3] + 7\*send[3,4];

600.325/425 Declarative Methods - J. Eisner

35

## Transportation Problem in ZIMPL

- set Producer := {"alice", "bob", "carol"}; Variables are assumed real and >= 0 unless declared otherwise
- set Consumer := {1 to 4}; (indexed by members of a specified set).
- var send[Producer\*Consumer];
- subto supply\_a: sum <c> in Consumer: send["alice",c] <= 500;
- subto supply\_b: sum <c> in Consumer: send["bob",c] <= 300;
- subto supply\_c: sum <c> in Consumer: send["carol",c] <= 400;
- subto demand\_1: sum <p> in Producer: send[p,1] == 200;
- subto demand\_2: sum <p> in Producer: send[p,2] == 400;
- subto demand\_3: sum <p> in Producer: send[p,3] == 300;
- subto demand\_4: sum <p> in Producer: send[p,4] == 100;
- minimize cost: 10\*send["alice",1] + 8\*send["alice",2] + 5\*send["alice",3] + 9\*send["alice",4] + 7\*send["bob",1] + 5\*send["bob",2] + 5\*send["bob",3] + 3\*send["bob",4] + 11\*send["carol",1] + 10\*send["carol",2] + 8\*send["carol",3] + 7\*send["carol",4];

600.325/425 Declarative Methods - J. Eisner

36

## Transportation Problem in ZIMPL

- set Producer := {"alice", "bob", "carol"};
- set Consumer := {1 to 4};
- var send[Producer\*Consumer] >= -10000;
- param supply[Producer] := <"alice"> 500, <"bob"> 300, <"carol"> 400;
- param demand[Consumer] := <1> 200, <2> 400, <3> 300, <4> 100;
- param transport\_cost[Producer\*Consumer] :=
 

	1	2	3	4
"alice"	10	8	5	9
"bob"	7	5	5	3
"carol"	11	10	8	7
- subto supply: forall <p> in Producer:  
(sum <c> in Consumer: send[p,c]) <= supply[p];
- subto demand: forall <c> in Consumer:  
(sum <p> in Producer: send[p,c]) == demand[c];
- minimize cost: sum <p,c> in Producer\*Consumer:  
transport\_cost[p,c] \* send[p,c];

Variables are assumed real and >= 0 unless declared otherwise

unknowns (remark: mustn't multiply unknowns by each other if you want a linear program)

knowns

Collapse similar formulas that differ only in constants by using indexed names for the constants, too ("parameters")

600.325/425 Declarative Methods - J. Eisner

37

## How to Encode Interesting Things in LP (sometimes needs MIP)

### Slack variables

- What if transportation problem is UNSAT?
- E.g., total possible supply < total demand

- Relax the constraints. Change  
subto demand\_1: a1 + b1 + c1 == 200;
- to  
subto demand\_1: a1 + b1 + c1 <== 200 ?

No, then we'll manufacture nothing, and achieve a total cost of 0.

### Slack variables

- What if transportation problem is UNSAT?
- E.g., total possible supply < total demand

- Relax the constraints. Change  
subto demand\_1: a1 + b1 + c1 == 200;
- to  
subto demand\_1: a1 + b1 + c1 <== 200 ?

Obviously doesn't help UNSAT. But what happens in SAT case?  
Answer: It doesn't change the solution. Why not?  
OK, back to our problem ...

- This is typical: the solution will achieve equality on some of your inequality constraints. Reaching equality was what stopped the solver from pushing the objective function to an even better value.
- And == is equivalent to >= and <=. Only one of those will be "active" in a given problem, depending on which way the objective is pushing. Here the <= half doesn't matter because the objective is essentially trying to make a1+b1+c1 small anyway. The >= half will achieve equality all by itself.

### Slack variables

Also useful if we could meet demand but maybe would rather not: trade off transportation cost against cost of not quite meeting demand

- What if transportation problem is UNSAT?
- E.g., total possible supply < total demand

- Relax the constraints. Change  
subto demand\_1: a1 + b1 + c1 == 200;
- to  
subto demand\_1: a1 + b1 + c1 + slack1 == 200; (or >= 200)

Now add a linear term to the objective:

- minimize cost: (sum <p,c> in Producer\*Consumer:  
transport\_cost[p,c] \* send[p,c])  
+ (slack1\_cost) \* slack1 ;

cost per unit of buying from an outside supplier

### Slack variables

Also useful if we could meet demand but maybe would rather not: trade off transportation cost against cost of not quite meeting demand

- What if transportation problem is UNSAT?
- E.g., total possible supply < total demand

- Relax the constraints. Change  
subto demand\_1: a1 + b1 + c1 == 200;
- to  
subto demand\_1: a1 + b1 + c1 == 200 - slack1;

Now add a linear term to the objective:

- minimize cost: (sum <p,c> in Producer\*Consumer:  
transport\_cost[p,c] \* send[p,c])  
+ (slack1\_cost) \* slack1 ;

cost per unit of doing without the product

## Piecewise linear objective

- What if cost of doing without the product goes up nonlinearly?
- It's pretty bad to be missing 20 units, but we'd make do.
- But missing 60 units is really horrible (more than 3 times as bad) ...

- We can handle it still by linear programming:
  - subto demand\_1:  $a1 + b1 + c1 + slack1 + slack2 + slack3 == 200$  ;
  - subto s1:  $slack1 \leq 20$ ; # first 20 units
  - subto s2:  $slack2 \leq 10$ ; # next 10 units (up to 30)
  - subto s3:  $slack3 \leq 30$ ; # next 30 units (up to 60)

Now add a linear term to the objective:

minimize cost: (sum <p,c> in Producer\*Consumer: transport\_cost[p,c] \* send[p,c])

+ (slack1\_cost \* slack1) + (slack2\_cost \* slack2) + (slack3\_cost \* slack3)

not too bad      worse (per unit)      ouch! out of business

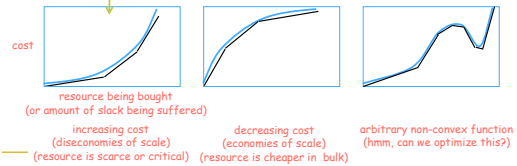
so max total slack is 60; could drop this constraint to allow

## Piecewise linear objective

- subto demand\_1:  $a1 + b1 + c1 + slack1 + slack2 + slack3 \leq 200$  ;
- subto s1:  $slack1 \leq 20$ ; # first 20 units
- subto s2:  $slack2 \leq 10$ ; # next 10 units (up to 30)
- subto s3:  $slack3 \leq 30$ ; # next 30 units (up to 60)
- minimize cost: (sum <p,c> in Producer\*Consumer: transport\_cost[p,c] \* send[p,c])

+ (slack1\_cost \* slack1) + (slack2\_cost \* slack2) + (slack3\_cost \* slack3);

Note: Can approximate any continuous function by piecewise linear. In our problem,  $slack1 \leq slack2 \leq slack3$  (costs get worse).



## Piecewise linear objective

- subto demand\_1:  $a1 + b1 + c1 + slack1 + slack2 + slack3 \leq 200$  ;
- subto s1:  $slack1 \leq 20$ ; # first 20 units
- subto s2:  $slack2 \leq 10$ ; # next 10 units (up to 30)
- subto s3:  $slack3 \leq 30$ ; # next 30 units (up to 60)
- minimize cost: (sum <p,c> in Producer\*Consumer: transport\_cost[p,c] \* send[p,c])

+ (slack1\_cost \* slack1) + (slack2\_cost \* slack2) + (slack3\_cost \* slack3);

Note: Can approximate any continuous function by piecewise linear. In our problem,  $slack1\_cost \leq slack2\_cost \leq slack3\_cost$  (costs get worse).

It's actually important that costs get worse. Why?

Answer 1: Otherwise the encoding is wrong! (If slack2 is cheaper, solver would buy from outside supplier 2 first.)

Answer 2: It ensures that the objective function is convex! Otherwise too hard for LP; we can't expect any LP encoding to work.

Therefore: E.g., if costs get progressively cheaper, (e.g., so-called "economies of scale" - quantity discounts), then you can't use LP. ☹

How about integer linear programming (ILP)?

## Piecewise linear objective

- subto demand\_1:  $a1 + b1 + c1 + slack1 + slack2 + slack3 \leq 200$  ;
- subto s1:  $slack1 \leq 20$ ; # first 20 units
- subto s2:  $slack2 \leq 10$ ; # next 10 units (up to 30)
- subto s3:  $slack3 \leq 30$ ; # next 30 units (up to 60)
- minimize cost: (sum <p,c> in Producer\*Consumer: transport\_cost[p,c] \* send[p,c])

+ (slack1\_cost \* slack1) + (slack2\_cost \* slack2) + (slack3\_cost \* slack3);

- Need to ensure that even if the slack\_costs are set arbitrarily (any function!), slack1 must reach 20 before we can get the quantity discount by using slack2.
- Use integer linear programming. How?
- var k1 binary; var k2 binary; var k3 binary; # 0-1 ILP
- subto slack1 <= 20\*k1; # can only use slack1 if k1==1, not if k1==0
- subto slack2 <= 10\*k2; # If we want to allow total slack, should we drop this constraint? No, we need it (if k3=0). Just change 30 to a large number M.
- subto slack3 <= 30\*k3;
- subto slack1 >= k2\*20; # if we use slack2, then slack1 must be fully used
- subto slack2 >= k3\*10; # if we use slack3, then slack2 must be fully used

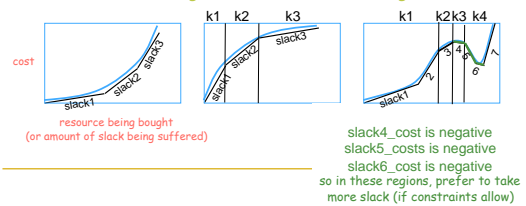
Can drop k1. It really has no effect, since nothing stops it from being 1. Corresponds to the fact that we're always allowed to use slack1.

## Piecewise linear objective

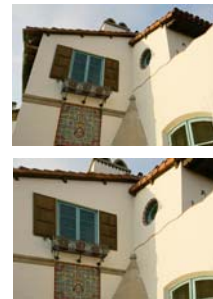
- subto demand\_1:  $a1 + b1 + c1 + slack1 + slack2 + slack3 \leq 200$  ;
- subto s1:  $slack1 \leq 20$ ; # first 20 units
- subto s2:  $slack2 \leq 10$ ; # next 10 units (up to 30)
- subto s3:  $slack3 \leq 30$ ; # next 30 units (up to 60)
- minimize cost: (sum <p,c> in Producer\*Consumer: transport\_cost[p,c] \* send[p,c])

+ (slack1\_cost \* slack1) + (slack2\_cost \* slack2) + (slack3\_cost \* slack3);

Note: Can approximate any continuous function by piecewise linear. Divide into convex regions, use ILP to choose region.



## Image Alignment



## Image Alignment

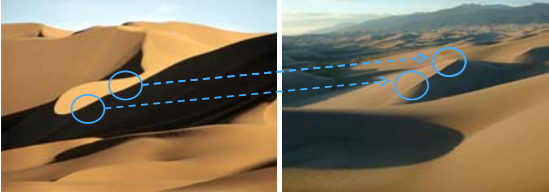
as a transportation problem, via "Earth Mover's Distance" (Monge, 1781)



600.325/425 Declarative Methods - J. Eisner 49

## Image Alignment

as a transportation problem, via "Earth Mover's Distance" (Monge, 1781)



600.325/425 Declarative Methods - J. Eisner 50

## Image Alignment

as a transportation problem, via "Earth Mover's Distance" (Monge, 1781)

warning: this code takes some liberties with ZIMPL, which is not quite this flexible in handling tuples; a running version would be slightly uglier

- param N := 12; param M := 10; # dimensions of image
- set X := {0..N-1}; set Y := {0..M-1};
- set P := X\*Y; # points in source image
- set Q := X\*Y; # points in target image
- defn norm(x,y) := sqrt(x\*x+y\*y);
- defn dist(<x1,y1>,<x2,y2>) := norm(x1-x2,y1-y2);
- param movecost := 1;
- param delcost := 1000; param inscost := 1000;
- var move[P\*Q]; # amount of earth moved from P to Q
- var del[P]; # amount of earth deleted from P in source image
- var ins[Q]; # amount of earth added at Q in target image

600.325/425 Declarative Methods - J. Eisner 51

## Image Alignment

as a transportation problem, via "Earth Mover's Distance" (Monge, 1781)

warning: this code takes some liberties with ZIMPL, which is not quite this flexible in handling tuples; a running version would be slightly uglier

- defset Neigh := { -1 .. 1 } \* { -1 .. 1 } - {<0,0>};
- minimize emd:
  - (sum <p,q> in P\*Q: move[p,q]\*movecost\*dist(p,q))
  - + (sum <p> in P: del[p]\*delcost) + (sum <q> in Q: ins[q]\*inscost);
- subto source: forall <p> in P:
  - source[p] = del[p]; (sum <q> in Q: move[p,q]);
- subto target: forall <q> in Q:
  - target[q] = ins[q] + (sum <p> in P: move[p,q]);
- subto smoothness: forall <p> in P: forall <q> in Q: forall <d> in Neigh:
  - move[p,q]/source[p] <= 2\*move[p+d,q+d]/source[p+d];

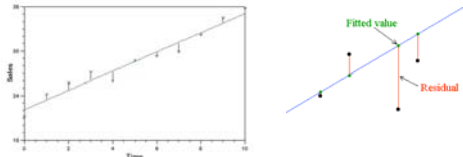
constant, so ok for LP (if > 0)

don't have to do it all by moving dirt: if that's impossible or too expensive, can manufacture/destroy dirt!

no longer a standard transportation problem: solution might no longer be integers

600.325/425 Declarative Methods - J. Eisner 52

## L1 Linear Regression



- Given data  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$
- Find a linear function  $y=mx+b$  that approximately predicts each  $y_i$  from its  $x_i$  (why?)
- Easy and useful generalization not covered on these slides:
  - each  $x_i$  could be a vector (then  $m$  is a vector too and  $mx$  is a dot product)
  - each  $y_i$  could be a vector too (then  $mx$  is a matrix and  $mx$  is a matrix multiplication)

600.325/425 Declarative Methods - J. Eisner 53

## L1 Linear Regression

- Given data  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$
- Find a linear function  $y=mx+b$  that approximately predicts each  $y_i$  from its  $x_i$
- Standard "L2" regression:
  - minimize  $\sum_i (y_i - (mx_i+b))^2$
  - This is a convex quadratic problem. Can be handled by gradient descent, or more simply by setting the gradient to 0 and solving.
- "L1" regression:
  - minimize  $\sum_i |y_i - (mx_i+b)|$ , so  $m$  and  $b$  are less distracted by outliers
  - Again convex, but not differentiable, so no gradient!
  - But now it's a linear problem. Handle by linear programming:
    - subto  $y_i = (mx_i+b) + (u_i - v_i)$ ; subto  $u_i \geq 0$ ; subto  $v_i \geq 0$ ;
    - minimize  $\sum_i (u_i + v_i)$ ;

600.325/425 Declarative Methods - J. Eisner 54

### More variants on linear regression

- L1 linear regression:
  - minimize  $\sum |y_i - (mx+b)|$ , so m and b are less distracted by outliers
  - Handle by linear programming:
    - subto  $y_i = (mx_i+b) + (u_i - v_i)$ ; subto  $u_i \geq 0$ ; subto  $v_i \geq 0$ ;
    - minimize  $\sum (u_i + v_i)$ ; If you heard of Ridge or Lasso regression: "regularize" m (encourage it to be small) by adding  $\lambda|m|$  to objective function, under  $1/2$  or  $1/1$  norm
- Quadratic regression:  $y_i \approx (ax_i^2 + bx_i + c)$ ?
  - Answer: Still linear constraints!  $x_i^2$  is a constant (if  $(x_i, y_i)$  is given).
- $L^\infty$  linear regression: Minimize the maximum residual instead of the total of all residuals?
  - Answer: minimize z; subto forall  $i$  in I:  $u_i + v_i \leq z$ ;
  - Remark: Including  $\max(p,q,r)$  in the cost function is easy. Just minimize z subject to  $p \leq z, q \leq z, r \leq z$ . Keeps all of them small.
  - But: Including  $\min(p,q,r)$  is hard! Choice about which one to keep small.
  - Need ILP. Binary a,b,c with  $a+b+c=1$ . Choice of (1,0,0),(0,1,0),(0,0,1).
  - Now what? First try:  $\min ap+bq+cr$ . But ap is quadratic, oops!
  - Instead: use lots of slack on unenforced constraints. Min z subj. to  $p \leq M(1-a), q \leq M(1-b), r \leq M(1-c)$ , where M is large constant

### CNF-SAT (using binary ILP variables)

- We just said " $a+b+c=1$ " for "exactly one" (sort of like XOR).
- Can we do any SAT problem?
  - If so, an ILP solver can handle SAT ... and more.
- Example:  $(A \vee B \vee \neg C) \wedge (D \vee \neg E)$
- SAT version:
  - constraints:  $(a+b+(1-c)) \geq 1, (d+(1-e)) \geq 1$
  - objective: none needed, except to break ties
- MAX-SAT version:
  - constraints:  $(a+b+(1-c)-u1) \geq 1, (d+(1-e)+u2) \geq 1$  (u1, u2 are slack)
  - objective: minimize  $c1*u1+c2*u2$  where c1 is the cost of violating constraint 1, etc.

### Non-clausal SAT (again using 0-1 ILP)


- If A is a [boolean] variable, then A and  $\neg A$  are "literal" formulas.
- If F and G are formulas, then so are
  - $F \wedge G$  ("F and G")
  - $F \vee G$  ("F or G")
  - $F \rightarrow G$  ("If F then G"; "F implies G")
  - $F \leftrightarrow G$  ("F if and only if G"; "F is equivalent to G")
  - $F \text{ xor } G$  ("F or G but not both"; "F differs from G")
  - $\neg F$  ("not F")
- If we are given a non-clausal formula, easy to set up as ILP using auxiliary variables.

### Non-clausal SAT (again using 0-1 ILP)

- If we are given a non-CNF constraint, easy to set up as ILP using auxiliary variables.
- $(A \wedge B) \vee (A \wedge \neg(C \wedge (D \vee E)))$ 
  - $Q \geq D; Q \geq E; Q \leq D+E$
  - $R \leq C; R \leq Q; R \geq C+Q-1$
  - $S \leq A; S \leq (1-R); S \geq A+(1-R)-1$
  - $T \geq P; T \geq S; T \leq P+S$
- $P \leq A; P \leq B; P \geq A+B-1$
- Finally, require  $T=1$ . Or for a soft constraint, add  $\text{cost}*(1-T)$  to the minimization objective.

Note: Introducing one intermediate variable per subexpression is a bit less efficient than the CNF conversion tricks we learned long ago. Either approach would work in either setting.

### MAX-SAT example: Linear Ordering Problem



- Arrange these archaeological artifacts or fossils along a timeline
- Arrange a program's functions in a sequence so that callers tend to be above callees
- Poll humans based on pairwise preferences: Then sort the political candidates or policy options or acoustic stimuli into a *global* order
- In short:
  - Sorting with a flaky comparison function
    - might not be asymmetric, transitive, etc.
    - can be weighted
      - the comparison "a < b" isn't boolean, but real
      - strongly positive/negative if we strongly want a to precede/follow b
  - maximize the sum of preferences
  - NP-hard

### MAX-SAT example: Linear Ordering Problem

- set  $X := \{ 1 \dots 50 \}$ ; # set of objects to be ordered
- param  $G[X * X] := \text{read "test.lop" as "<1n, 2n> 3n"$ ;
- var  $\text{LessThan}[X * X]$  binary;
- maximize goal:  $\text{sum } \langle x,y \rangle \text{ in } X * X : G[x,y] * \text{LessThan}[x,y]$ ;
- subto **irreflexive**: forall  $\langle x \rangle$  in X:  $\text{LessThan}[x,x] = 0$ ;
- subto **antisymmetric\_and\_total**: forall  $\langle x,y \rangle$  in  $X * X$  with  $x < y$ :  $\text{LessThan}[x,y] + \text{LessThan}[y,x] = 1$ ; # what would  $\leq$  and  $\geq$  do?
- subto **transitive**: forall  $\langle x,y,z \rangle$  in  $X * X * X$ : # if  $x < y$  and  $y < z$  then  $x < z$   $\text{LessThan}[x,z] \geq \text{LessThan}[x,y] + \text{LessThan}[y,z] - 1$ ;
- # alternatively (get this by adding  $\text{LessThan}[z,x]$  to both sides)
- # subto **transitive**: forall  $\langle x,y,z \rangle$  in  $X * X * X$
- # with  $x < y$  and  $x < z$  and  $y \neq z$ : # merely prevents redundancy
- #  $\text{LessThan}[x,y] + \text{LessThan}[y,z] + \text{LessThan}[z,x] \leq 2$ ; # no cycles

## Why isn't this just SAT all over again?

- Different solution techniques (we'll compare)
- Much easier to encode "at least 13 of 26":
  - Remember how we had to do it in pure SAT?

## Encoding "at least 13 of 26"

(without listing all 38,754,732 subsets!)

A	B	C	...	L	M	...	Y	Z
$A \geq 1$	$A-B \geq 1$	$A-C \geq 1$		$A-L \geq 1$	$A-M \geq 1$		$A-Y \geq 1$	$A-Z \geq 1$
	$A-B \geq 2$	$A-C \geq 2$		$A-L \geq 2$	$A-M \geq 2$		$A-Y \geq 2$	$A-Z \geq 2$
		$A-C \geq 3$		$A-L \geq 3$	$A-M \geq 3$		$A-Y \geq 3$	$A-Z \geq 3$
				...	...		...	...
				$A-L \geq 12$	$A-M \geq 12$		$A-Y \geq 12$	$A-Z \geq 12$
					$A-M \geq 13$		$A-Y \geq 13$	$A-Z \geq 13$

26 original variables  $A \dots Z$ ,  
plus  $< 26^2$  new variables  
such as  $A-L \geq 3$

- SAT formula should require that  $A-Z \geq 13$  is true ... and what else?
- yadayada  $\wedge A-Z \geq 13 \wedge ((A-Z \geq 13 \rightarrow (A-Y \geq 13 \vee (A-Y \geq 12 \wedge Z))) \wedge ((A-Y \geq 13 \rightarrow (A-X \geq 13 \vee (A-X \geq 12 \wedge Y)))) \wedge \dots$

one "only if" definitional constraint for each new variable

600.325/425 Declarative Methods - J. Eisner

62

## Why isn't this just SAT all over again?

- Different solution techniques (we'll compare)
- Much easier to encode "at least 13 of 26":
  - $a+b+c+\dots+z \geq 13$  (and solver exploits this)
  - Lower bounds on such sums are useful to model requirements
  - Upper bounds on such sums are useful to model limited resources
  - Can include real coefficients (e.g.,  $c$  uses up 5.4 of the resource):
    - $a + 2b + 5.4c + \dots + 0.3z \geq 13$  (very hard to express with SAT)
    - MAX-SAT allows an overall soft constraint, but not a limit of 13 (nor a piecewise-linear penalty function for deviations from 13)
- Mixed integer programming combines the power of SAT and disjunction with the power of numeric constraints
  - Even if some variables are boolean, others may be integer or real and constrained by linear equations ("Mixed Integer Programming")

## Logical control of real-valued constraints

- Want  $\delta=1$  to force an inequality constraint to turn on: (where  $\delta$  is a binary variable)
- **Idea:**  $\delta=1 \rightarrow a \cdot x \leq b$
- **Implementation:**  $a \cdot x \leq b + M(1-\delta)$  where  $M$  very large
  - Requires  $a \cdot x \leq b + M$  always, so set  $M$  to **upper bound** on  $a \cdot x - b$
- Conversely, want satisfying the constraint to force  $\delta=1$ :
- **Idea:**  $a \cdot x \leq b \rightarrow \delta=1$  or equivalently  $\delta=0 \rightarrow a \cdot x > b$
- **Implementation:**
  - approximate by  $\delta=0 \rightarrow a \cdot x \geq b + 0.001$
  - implement as  $a \cdot x + \text{surplus} \cdot \delta \geq b + 0.001$
  - more precisely  $a \cdot x \geq b + 0.001 + (m - 0.001) \cdot \delta$  where  $m$  very negative
    - Requires  $a \cdot x \geq b + m$  always, so set  $m$  to **lower bound** on  $a \cdot x - b$

## Logical control of real-valued constraints

- If some inequalities hold, want to enforce others too.
- ZIMPL doesn't (yet?) let us write
  - subto foo:  $(a \cdot x \leq b \text{ and } c \cdot x \leq d) \rightarrow (e \cdot x \leq f \text{ or } g \cdot x \leq h)$
- but we can manually link these inequalities to binary variables:
  - $a \cdot x \leq b \rightarrow \delta_1$  implement as on bottom half of previous slide
  - $c \cdot x \leq d \rightarrow \delta_2$  implement as on bottom half of previous slide
  - $(\delta_1 \text{ and } \delta_2) \rightarrow \delta_3$  implement as  $\delta_3 \geq \delta_1 + \delta_2 - 1$
  - $\delta_3 \rightarrow (\delta_4 \text{ or } \delta_5)$  implement as  $\delta_3 \leq \delta_4 + \delta_5$
  - $\delta_4 \rightarrow e \cdot x \leq f$  implement as on top half of previous slide
  - $\delta_5 \rightarrow g \cdot x \leq h$  implement as on top half of previous slide
- Partial shortcut in ZIMPL using "vif ... then ... else .. end" construction:
  - subto foo1: vif  $(\delta_1=0)$  then  $a \cdot x \geq b + 0.001$  end;
  - subto foo2: vif  $(\delta_2=0)$  then  $c \cdot x \geq d + 0.001$  end;
  - subto foo3: vif  $((\delta_1=1 \text{ and } \delta_2=1) \text{ and not } (\delta_3=1 \text{ or } \delta_4=1))$  then  $\delta_1 \geq \delta_1 + 1$  end; # i.e., the "vif" condition is impossible
  - subto foo4: vif  $(\delta_4=1)$  then  $e \cdot x \leq f$  end;
  - subto foo5: vif  $(\delta_5=1)$  then  $g \cdot x \leq h$  end;

## Integer programming beyond 0-1:

### N-Queens Problem

- param queens := 8;
- set C := {1 .. queens};
- var row[C] integer >= 1 <= queens;
- set Pairs := {<i,j> in C\*C with i < j};
- subto alldifferent: forall <i,j> in Pairs: row[i] != row[j];
- subto nodiagonal: forall <i,j> in Pairs: vabs(row[i]-row[j]) != j-i;
- # no line saying what to maximize or minimize

Instead of writing  $x \neq y$  in ZIMPL, or  $(x-y) \neq 0$ , need to write  $vabs(x-y) \geq 1$ . (if  $x,y$  integer; what if they're real?)  
This is equivalent to  $v \geq 1$  where  $v$  is forced (how?) to equal  $|x-y|$ .  
 $v \geq x-y$ ,  $v \geq y-x$ , and add  $v$  to the minimization objective.  
No, can't be right def of  $v$ : LP alone can't define non-convex feasible region.  
And it is wrong: this encoding will allow  $x=y$  and just choose  $v=1$  anyway!  
Correct solution: use ILP. Binary var  $\delta$ , with  $\delta=0 \rightarrow v=x-y$ ,  $\delta=1 \rightarrow v=y-x$ .  
Or more simply, eliminate  $v$ :  $\delta=0 \rightarrow x-y \geq 1$ ,  $\delta=1 \rightarrow y-x \geq 1$ .

program example from ZIMPL manual

## Integer programming beyond 0-1: Allocating Indivisible Objects

- Airline scheduling  
(can't take a fractional number of passengers)
- Job shop scheduling (like homework 2)  
(from a set of identical jobs, each machine takes an integer #)
- Knapsack problems (like homework 5)
- Others?

## Harder Real-World Examples of LP/ILP/MIP

## Unsupervised Learning of a Part-of-Speech Tagger

- based on Ravi & Knight 2009

600.325/425 Declarative Methods - J. Eisner

69

## Part-of-speech tagging

Input: the lead paint is unsafe

Output: the/Det lead/N paint/N is/V unsafe/Adj

- Partly supervised learning:
  - You have a lot of text (without tags)
  - You have a dictionary giving possible tags for each word

600.465 - Intro to NLP - J. Eisner

70

## What Should We Look At?

*correct tags*

PN Verb Det Noun Prep Noun Prep Det Noun  
 Bill directed a cortege of autos through the dunes

PN Adj Det Noun Prep Noun Prep Det Noun  
 Verb Verb Noun Verb  
 Adj  
 Prep  
 ...?

*some possible tags for  
each word (maybe more)*

Each unknown tag is **constrained** by its word  
and by the tags to its immediate left and right.  
But those tags are unknown too ...

600.465 - Intro to NLP - J. Eisner

71

## What Should We Look At?

*correct tags*

PN Verb Det Noun Prep Noun Prep Det Noun  
 Bill directed a cortege of autos through the dunes

PN Adj Det Noun Prep Noun Prep Det Noun  
 Verb Verb Noun Verb  
 Adj  
 Prep  
 ...?

*some possible tags for  
each word (maybe more)*

Each unknown tag is **constrained** by its word  
and by the tags to its immediate left and right.  
But those tags are unknown too ...

600.465 - Intro to NLP - J. Eisner

72

## What Should We Look At?

*correct tags*

PN Verb Det Noun Prep Noun Prep Det Noun  
Bill directed a cortege of autos through the dunes

PN Adj Det Noun Prep Noun Prep Det Noun  
Verb Verb Noun Verb  
Adj  
Prep *some possible tags for  
each word (maybe more)*  
...?

Each unknown tag is **constrained** by its word  
and by the tags to its immediate left and right.  
But those tags are unknown too ...

600.465 - intro to NLP - J. Eisner

73

## Unsupervised Learning of a Part-of-Speech Tagger

- Given  $k$  tags (Noun, Verb, ...)
  - Given a dictionary of  $m$  word **types** (aardvark, abacus, ...)
  - Given some text:  $n$  word **tokens** (The aardvark jumps over...)
  - Want to pick:  $n$  **tags** (Det Noun Verb Prep..)
- 
- Encoding as variables?
  - How to inject some knowledge about types and tokens?
  - Constraints and objective?
    - Few tags allowed per word
    - Few 2-tag sequences allowed (e.g., "Det Det" is bad)
    - Tags may be correlated with one another, or with word endings

600.325/425 Declarative Methods - J. Eisner

74

## Minimum spanning tree ++

- based on Martins et al. 2009

600.325/425 Declarative Methods - J. Eisner

75

## Traveling Salesperson

- Version with subtour elimination constraints
  
  
  
  
  
  
  
  
  
  
- Version with auxiliary variables