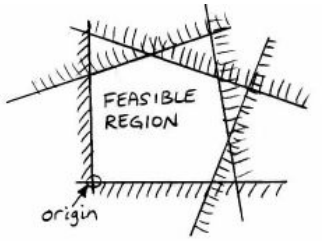
SYSC5004 Optimization for Engineering Applications - Based on the book:

- Practical Optimization: A Gentle Introduction [http://www.sce.carleton.ca/faculty/chinneck/po.html]

1. Simplex

A. Standard Form LP

- Objective Function: Max
- All constraints $\leq$ +ve constant
- All variables ≥ 0



B. Simplex Concept

- The variables v_i correspond to variable non-negativity bounds, and the slack s_i correspond to constraints.
- The basis (v_1,v_2,..., s_1,s_2,...) is always one corner at a time. Some of them set to zero (nonbasic), some not (basic)
- The whole point of Simplex to set variables to *either*:
  - **Nonbasic**: slack or variable *explicitly set* to = 0 ⇒ current corner point is on that constraint.
  - **Basic**: slack or var *explicitly set* to$\neq$ 0

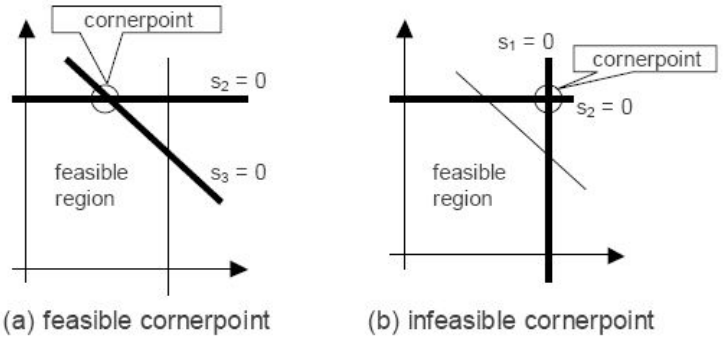


Figure 3.4: Setting variables to zero defines a cornerpoint.

- Exactly 1 basic variable per row (var coef = 1; rest of its column = 0). Basic variable *value* = RHS
- Iterate:
  - Pivot Column: find variable that given *nonbasic* → *basic* will improve obj function the most. Most -ve.
  - Pivot Row: find variable that would hit 0 (*basic* → *nonbasic*) first. Smallest MRT = $\frac{RHS}{var}$.
    - Special case (i): ratio = 0 ⇒ constraints don't intersect, no corner point. no limit.
    - Special case (ii): ratio is -ve ⇒ corner point is *behind* direction of increase. no limit.
    - Special case: all no limit: unbounded.
    - Special case: solved but one main variable is 0 ⇒ multiple optima (edge not vertex)
    - No MRT for obj func row!
  - Update Equations:
    - Pivot row: divide by pivot element
    - Other rows = old row - (intersection with pivot row x intersection pivot column)
- Stop: when no pivot column can be found.

C. Simplex Steps

1. Formulate original
2. Convert formulation for Tableau:
   - Objective: *max* (multiply by -1 if *min* to convert to *max*)
   - $\leq$ constraint: add slack (s)
   - $=$ constraint: add artificial (a_i)
   - $\geq$ constrain: subtract slack, add artificial
3. Setup table

Phase 1: Find an initial cornerpoint feasible solution (basic feasible solution). If none is found, then the model is infeasible, so exit.

Note: If dealing with a standard form LP, the origin is always a basic feasible solution, so you can always start there. If it is not a standard form LP, then you must use a special phase 1 procedure that we will study later.

Phase 2: Iterate until the stopping conditions are met.

.2.1 *Are we optimal yet?* Look at the current version of the objective function to see if an entering basic variable is available. If none is available, then exit with the current basic feasible solution as the optimum solution.

.2.2 *Select entering basic variable*: choose the nonbasic variable that gives the fastest rate of increase in the objective function value.

2.3 *Select the leaving basic variable* by applying the Minimum Ratio Test.

2.4 *Update the equations* to reflect the new basic feasible solution.

2.5 Go to Step 2.1.

Figure 4.1: Overview of the simplex method.

3. Networks (Shortest/Spanning/Min-Max)

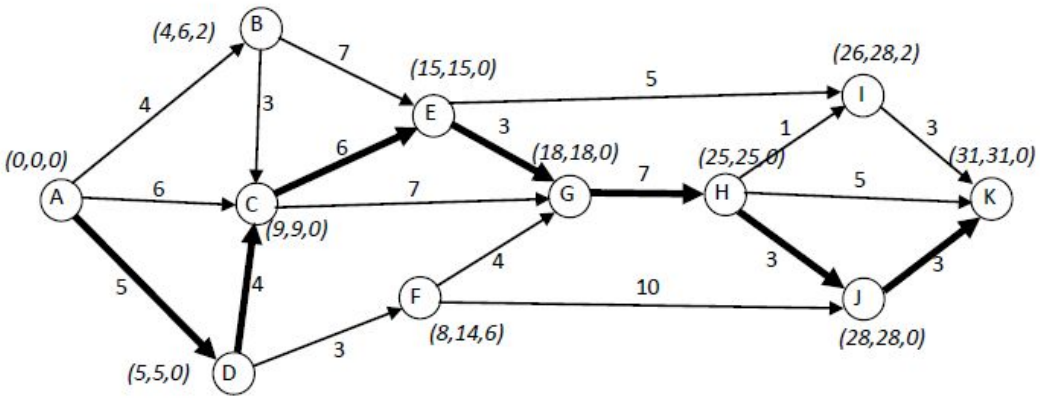
D. Shortest/Spanning/Min-Max

- USE A PENCIL!!
- Focus and get going
- Write out solution statement
- Unique/Unique/Non-unique
- Take a quick look at the assignments

4. Networks (PERT/Scheduling)

E. PERT

- USE A PENCIL!!
- Plot if necessary
- Node Label > (T_E , T_L , S_N)
- Arc Label > Duration (S_A)



- T_E = earliest time you could start this node (just go through it, first one is 000, last is nn0)
- T_L = latest time you could start this node = min [(T_L of node at head of arc) − (arc duration)]
- S_N = slack at that node
- S_A = (T_L of node at arc head) − (T_E of node at arc tail) − (arc duration).
- S_F = free slack on arc = (arc slack) − (Slack at arc head)

F. Scheduling

- No need for free slack: just be as careful as possible
- Range per arc is: from T_E first node to T_L of second node
- List alphabetically
- Adjust dials to meet resource leveling target: ASAP for early ones and ALAP for later ones.

5. Integer Programming

G. Branch & Bound

1. Calculate bounding for live buds (non-pruned). If non left jump to Step 4.
2. If incumbent found better than current incumbent > assign as current incumbent > prune worse values
3. Otherwise locate *global best* > expand > repeat step 1.
4. If no more live buds exit with current incumbent.
   If you got linear constraints get Dakin
   Formulation:
   - **Node:** represents a racer-to-leg assignment
   - **Node selection policy:** global best value of bounding function
   - **Variable selection policy:** choose the next leg in the natural order
   - **Bounding function:** for unassigned legs, assign the best racer even if chosen twice in a row, but the first unassigned racer must not be the same as the last assigned racer.
   - **Terminating rule:** when the best incumbent solution is better than or equal to all of the bounding function values over all of the bud nodes.
   - **Feasibility test:** a solution is feasible if no racer is used twice in a row.

5. **REMEMBER SOLUTION STATEMENT!!!!**

BnB vs Large Problems

- Happy: Guaranteed optimum solution
- Sad: combinatorially explosive
- Happy medium: *no guarantee* of optimality, get *good enough* answer instead
  - Good enough can be "within %5 of optimal", or "best after 10 seconds of computations", etc.
- Methods:
  - **Stop BnB within x% of optimum:** incumbent solution is within x% of best bud node
  - **Beam Search**: for limited memory (pick best *n* nodes; discard rest whenever memory limit approaches)
  - **DFS to first incumbent**: for limited time (dfs reaches a feasible solution fast).
  - **Genetic Algorithm**
    - A string, can be given a value
    - Reproduction: Pick mating pool based on probability, based on fitness per citizen
    - Crossover: flip stuff around between two random parents. might produce infeasible kids.
    - Mutation: rate (e.g. 1 in 1000) change 1 character to a random element. Introduces realistic randomness, inclusive of others, outlier cases have a chance.
  - **Simulated Annealing:**
    - zigzag, with bad decision probability $e^{-d/T}$. Next $T=rT$

H. Balas Additive Algorithm

If something is arranged in *ORDER* call me!!! Binary: Any assignment that has a *YES* or *NO* answer too!!
Just say: Using ***Balas Additive Algorithm***
Depth First

I. Dakin

MILPS examples: Either-or constraints; either-or RHS; modeling startup costs.

Solve LP-relaxation > Do whoever is non integer > resolve LP > Branch and Bound till no one's left alive!

Maximize $Z = 8x_1 + 5x_2$

s.t. $x_1 + x_2 \leq 6$

$9x_1 + 5x_2 \leq 45$

x_1, x_2 are integer and nonnegative.

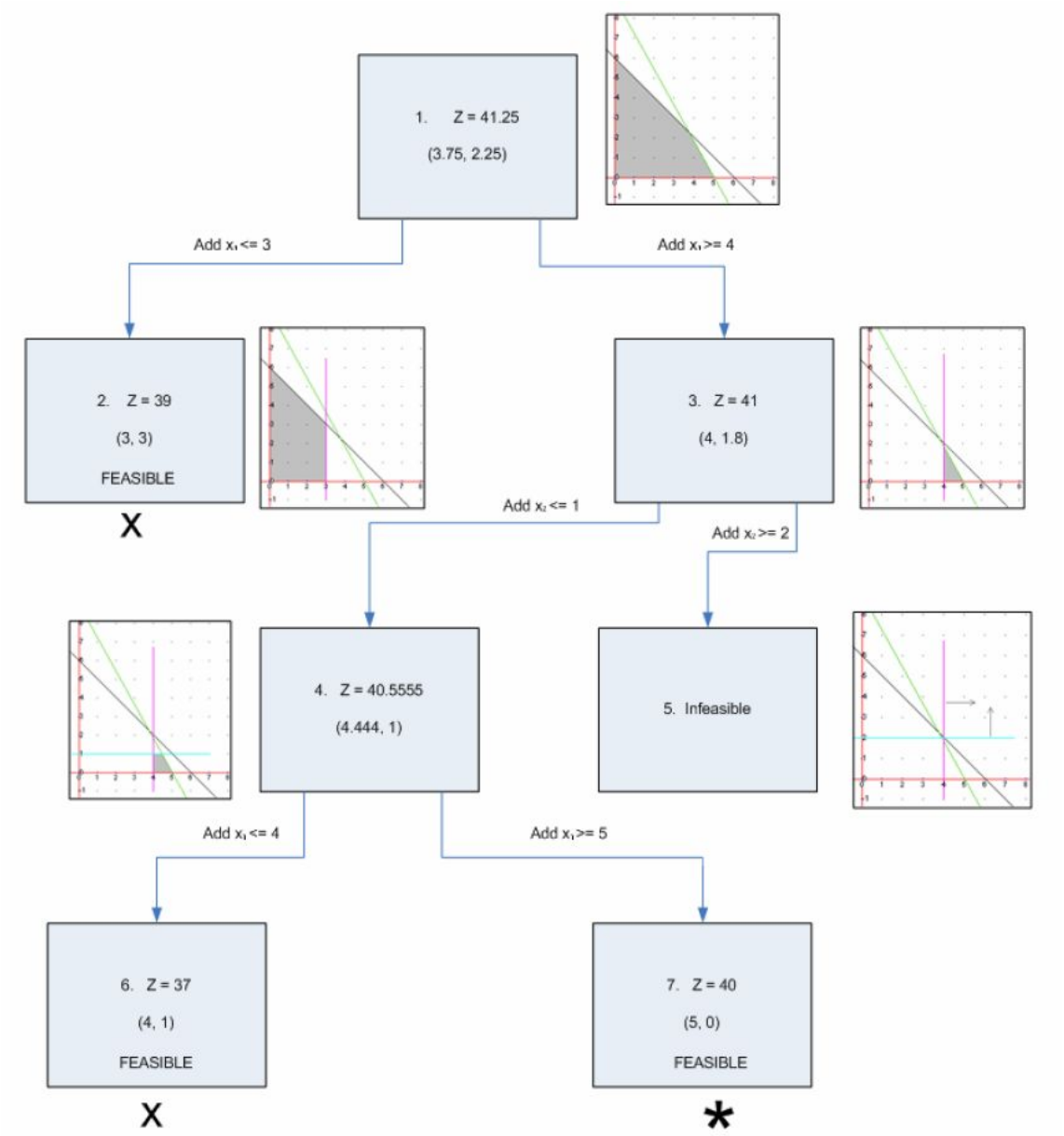


Figure 13.10: Branch and bound tree for Dakin's algorithm.

7. Dynamic Programming

Remember: Formulation > Table > Solution Statement

- **Stage:** that splits the problem into smaller parts (look at columns/rows with things to assign/be assigned)
- **State:** amount/capacity left to be assigned/filled
- **Decision:** amount/capacity assigned at current stage
- **Decision update to the state: reduces/increases/etc. the amount/capacity left to assign/fill**
- **Recursive Relationship:**
  - x_i : decision at stage i
  - d_i : "states" left at start of stage i
  - r_{xi} : "result of decision" in stage i if x_i is chosen
  - $f_i(d_i)$: optimum decision from stage i given current state is d_i
  - $f_i(d_i) = \max\ x_i(r_{xi} + f_{i+1}(d_i - x_i))$

Stage 4 $f_{4i}(v_4) = \max_{x_4}(r_{x_4})$

v_4	$x_4=1$	$x_4=2$	$x_4=3$	$x_4=4$	$x_4=5$	$f(v_4)$
1	**7**	-	-	-	-	7
2	7	**8**	-	-	-	8
3	7	8	**11**	-	-	11
4	7	8	11	**14**	-	14
5	7	8	11	14	**16**	16

Stage 3 $f_3(v_3) = \max_{x_3}(r_{x_3} + f_4(v_3 - x_3))$

v_3	$x_3=1$	$x_3=2$	$x_3=3$	$x_3=4$	$x_3=5$	$f(v_3)$	$v_4=v_3-x_3$
2	**15**	-	-	-	-	15	1
3	16	**17**	-	-	-	17	1
4	19	18	**20**	-	-	20	1
5	**22**	21	21	**22**	-	22	4 or 1
6	**24**	**24**	**24**	23	23	24	5 or 4 or 3

Stage 2 $f_2(v_2) = \max_{x_2}(r_{x_2} + f_3(v_2 - x_2))$

v_2	$x_2=1$	$x_2=2$	$x_2=3$	$x_2=4$	$x_2=5$	$f(v_2)$	$v_3=v_2-x_2$
3	**21**	-	-	-	-	21	2
4	23	**24**	-	-	-	24	2
5	**26**	**26**	**26**	-	-	26	4 or 3 or 2
6	28	**29**	28	28	-	29	4
7	30	**31**	**31**	30	30	31	5 or 4

Stage 1 $f_1(v_1) = \max_{x_1}(r_{x_1} + f_2(v_1 - x_1))$

v_1	$x_1=1$	$x_1=2$	$x_1=3$	$x_1=4$	$x_1=5$	$f(v_1)$	$v_2=v_1-x_1$
8	38	38	38	**39**	37	39	4

So the maximum number of votes is 39,000 and is obtained by distributing volunteers

district	1	2	3	4
number of volunteers	4	2	1	1

8. Non-Linear Programming

1. Introduction

❖ Simple Theory

Model has at least one nonlinear function: objective and/or constraint(s). Variables are real-valued.

❖ List a few reasons why NLP is hard:

- ❖ It's hard to distinguish a local optimum from a global optimum.
- ❖ There may be multiple disconnected feasible region
- ❖ Optima can be interior
- ❖ Not deterministic →
  - o different starting point might lead to different result
  - o different algorithms arrive at different results
  - o Might not find starting point in the first place.
- ❖ Equality is hard
- ❖ Solvers are complex and hard to control parameters correctly

❖ Bisection – 1D

Line Search method: Select two points with opposite signs, so it's known that the 'zero' optimal is somewhere in between. Check midpoint and replace the one with the same sign. Repeat until either the 'zero' optima is found, or the difference between the current two points is less than twice a predetermined tolerance value, in which case the optima is the midpoint of those two points.

❖ Special Cases

- o Quadratic
- o Separable
- o Frank-Wolfe

2. Unconstrained NLP

o Hessian Matrix

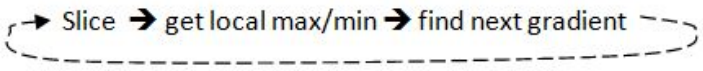
Matrix of second partial derivatives of the objective function. The matrix is POSITIVE DEFINITE at *x* if all determinants are all positive, and NEGATIVE DEFINITE if the first determinant is negative and the rest alternate in sign.

- ***x*** is a local **MAX** if:
  - Gradient = 0
  - Hessian is **NEGATIVE** DEFINITE (i.e. everything around *x* is going *down*)

- ***x*** is a local **MIN** if:
  - Gradient = 0
  - Hessian is **POSITIVE** DEFINITE (i.e. everything around *x* is going *up*)

o Gradient Methods (Steepest Ascent/Descent)

Series of 1D problem:

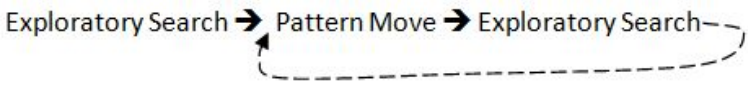


Startup: Choose an acceptable tolerance '**E**'
Choose initial point ***x***

Iterate: Calculate gradient at ***x*** (direction of best improvement) [*Negate if minimization!!!*]
Local optimal *x* = ***x*** + **t** (gradient at **x**)
- *To find **t**: > use 1D Bisection search on the objective function in **t***
  *> or just try small ones until you find one small enough to improve **x***

Repeat until gradient is less than or equal to **'E'**

o Pattern Searches (Hooke and Jeeves)



** Easy to program, Nondifferentiable functions are welcome, Can be fooled by "stepping over" features (because it uses Cartesian coords)

Startup: Choose perturbation vector **pv**
Choose perturbation tolerance vector **ptv**
Choose initial point ***x***
Choose α (usually 2)

Iterate: Step1. Exploratory Search to find ***x'*** that improves on ***x***
> if ***x'*** *improves* → *Pattern move*
> else → reduce **pv** by ½ → *Exploratory Search from previous best **x***
Step2. **Local optimal *x*** = ***x*** + **t** (gradient at **x**)
Step3. Look at your notes, they're fairly adequate

3. Constrained NLP

o SUMT – Barrier

- Mixed Constraints (equalities and inequalities)
- Find point that satisfies all inequalities → Make sure to stay within inequalities → approach equalities
- Method:

1. Convert to Barrier Function: MAX P(x, r) = f(x) – B(g(x), r)
   MIN P(x, r) = f(x) + B(g(x), r)
   - Inequality Barriers = $\frac{r}{b-g(x)}$, for constraint $g(x) \leq b$
   - Equality Barriers = $\frac{[b-g(x)]^2}{\sqrt{r}}$, for constraint $g(x) = b$
   - ***r*** regulates the strength of the barrier, making inequality barrier progressively weak to allow approach to inequalities. Makes equality attraction effect bigger and bigger. Usually ***r*** = 1
2. Solve iteratively while reducing r and using solution as next initial point, until difference between solutions is within tolerance, then stop and exit with that initial point.

o Lagrange

- Equality Constraints only
- Linear combination of partial gradients matches the gradient of objective at the optimum
- # constraints (m) < # variables (n) → suitable
- # constraints (m) = # variables (n) → one solution (the intersection point of all constraints)
- # constraints (m) > # variables (n) → no solution (over-constrained)
- Method:
  1. Convert to Lagrange Function: h(x, λ) = f(x) + λ Σ [g(x) - b]
  2. Solve simultaneous equations of partial derivatives=0
  3. Test solution to decide: min, max or saddle point
  4. Is it a feasible critical solution for original f(x) ?
  5. If no λ can be found, then no optima

o GRG

- 1. Reduce the dimensionality of the original problem by incorporating the equalities into the obj. function
  ➔ Take out as many variables as there are equality constraints....from the end (xn, xn-1, ...x1)
- 2. Solve an unconstrained problem using gradient methods
  ➔ Now you have an unconstrained version → solve using gradient method or Hooke and Jeeves

o KKT conditions

1. Lagrange Gradient Conditions [partial derivatives of h(x, λ) = 0]
2. Orthogonality [λ g(x) = 0]
3. Feasibility (Original Constraint) [g(x) ≤/≥/= 0]
4. If: g(x) ≤ 0 → λ ≥ 0
   if: g(x) ≥ 0 → λ ≤ 0
   if: g(x) = 0 → λ unconstrained
5. NOTE: Don't forget to test the given point and write out a solution statement!
6. If MIN problem: Convert to a MAX problem by multiplying the obj. function by -1.