

Final Exam

Adv. Analytics and Metaheuristics

Daniel Carpenter

March 2022

Contents

1 - <i>Question 1</i> (Version 1)	2
1.1 <i>Part 1</i> : Mathematical Formulation	2
1.2 <i>Part 2</i> : AMPL Code & Output	4
2 - <i>Question 2</i> (Version 6)	7
2.1 Code to get Root Node	7
2.2 Branch and Bound Diagram	8
3 - <i>Question 3</i> (Version 2)	9
3.1 <i>Part i</i> : Hill Climbing	9
3.2 <i>Part ii</i> : Path Relinking	10
3.3 <i>Part iii</i> : Simulated Annealing	11
4 - <i>Question 4</i> (Version 3)	12
5 - <i>Question 5</i> (Version 2)	13

1 - Question 1 (Version 1)

1.1 Part 1: Mathematical Formulation

1.1.1 Sets

NewBuildTypes: Set of new build types $b \in (\text{Homes}, \text{Duplex}, \text{MiniPark})$

1.1.2 Parameters

Parameter	Description	Default Value
<i>budget</i>	Federal grant allocation to revitalize neighborhoods	\$15MM total budget
<i>maxBuildingDemod</i>	Max amount of buildings that can be demolished	300 total buildings
<i>demoCost</i>	Cost of demolishing a building	\$4,000 per building
<i>freedUpSpace</i>	Acreage generated from demolishing a building	0.25 per building
<i>newBuildSpace_b</i>	Amount of acreage that a new building ($b \in \text{NewBuildTypes}$) consumes	<i>Homes</i> : 0.2, <i>Duplex</i> : 0.4, <i>MiniPark</i> : 1.0
<i>newBuildTax_b</i>	Amount of tax dollars generated from a new building ($b \in \text{NewBuildTypes}$)	<i>Homes</i> : 1,500, <i>Duplex</i> : 2,750, <i>MiniPark</i> : 500
<i>newBuildCost_b</i>	Amount of dollars used to create a new building ($b \in \text{NewBuildTypes}$)	<i>Homes</i> : 150,000, <i>Duplex</i> : 190,000, <i>MiniPark</i> : 20,000
<i>newBuildPercShare_b</i>	Minimum required percentage share of new buildings ($b \in \text{NewBuildTypes}$) created	<i>Homes</i> : 20%, <i>Duplex</i> : 10%, <i>MiniPark</i> : 5%

1.1.3 Decision Variables

Variable	Description
$numOldBuildsDemod$	Number of old buildings to demolish
$numNewBuilds_b$	Number of new buildings ($b \in NewBuildTypes$) to produce
$newBuildTotalCost$	Variables to hold total cost of new builds ($b \in NewBuildTypes$). Calculation: $\sum_{b \in NewBuildTypes} (numNewBuilds_b \times newBuildCost_b)$
$oldDemoTotalCost$	Variables to hold total cost of old demolitions. Calculation: $numOldBuildsDemod \times demoCost$
$sumOfNewBuilds$	# Variable to hold the sum of all new build types over all New build types ($b \in NewBuildTypes$). Calculation: $\sum_{b \in NewBuildTypes} (numNewBuilds_b)$

1.1.4 Objective

Maximize the tax revenue from the projects

$$maximize \text{ taxRevenue} : \sum_{b \in NewBuildTypes} (numNewBuilds_b \times newBuildTax_b)$$

1.1.5 Constraints

C1 Spend less than or equal to the federal budget (see variable definitions)

$$meetTheBudget : newBuildTotalCost + oldDemoTotalCost \leq budget$$

C2 Can only produce new builds using the demolished buildings land

$$useAvailLand : \sum_{b \in NewBuildTypes} (numNewBuilds_b \times newBuildSpace_b) \\ \leq numOldBuildsDemod \times freedUpSpace$$

C3 Can only clear a certain amount of old buildings

$$maxBuildingsCleared : numOldBuildsDemod \leq maxBuildingDemod$$

C4 For each new build type ($b \in NewBuildTypes$), the percentage share of the new build type must meet the minimum required (see variables)

$$share : numNewBuilds_b \geq newBuildPercShare_b \times sumOfNewBuilds, \\ \forall b \in Businesses$$

C5 Non-negativity and integer constraints

$$numOldBuildsDemod \in \mathbb{Z}, \geq 0 \\ numNewBuilds_b \in \mathbb{Z}, \geq 0, \forall b \in NewBuildTypes$$

1.2 Part 2: AMPL Code & Output

1.2.1 AMPL Code

Data *problem1.dat*

```
data;

# SETS =====

# Set of new build types
set NewBuildTypes := Homes Duplex MiniPark;

# PARAMETERS =====

param budget          := 15000000; # federal budget
param maxBuildingDemod := 300;      # max buildings can be demo'd
param demoCost         := 4000;     # Cost of each demolition
param freedUpSpace      := 0.25;    # Freed up space from demolition

# Amount of acreage that a new building (b in NewBuildTypes) consumes
param: newBuildSpace :=
    Homes    0.2
    Duplex   0.4
    MiniPark 1.0
    ;

# Amount of tax dollars generated from a new building (b in NewBuildTypes)
param: newBuildTax :=
    Homes    1500
    Duplex   2750
    MiniPark 500
    ;

# Amount of dollars used to create a new building (b in NewBuildTypes)
param: newBuildCost :=
    Homes    150000
    Duplex   190000
    MiniPark 20000
    ;

# Minimum required percentage share of new buildings (b in NewBuildTypes) created
param: newBuildPercShare :=
    Homes    0.20
    Duplex   0.10
    MiniPark 0.05
```

;

Model *problem1.mod*

```
reset;                # Reset globals
options solver cplex; # Using cplex for simplex alg

# SETS =====
set NewBuildTypes; # Set of new build types

# PARAMETERS =====
param budget          >= 0; # federal budget
param maxBuildingDemod >= 0; # max buildings can be demo'd
param demoCost        >= 0; # Cost of each demolition
param freedUpSpace    >= 0; # Freed up space from demolition

param newBuildSpace    {NewBuildTypes} >= 0; # new build acreage
param newBuildTax      {NewBuildTypes} >= 0; # n.b. tax generation
param newBuildCost     {NewBuildTypes} >= 0; # n.b. cost
param newBuildPercShare {NewBuildTypes} >= 0; # n.b. min % share

# DECISION VARIABLES =====
var numOldBuildsDemod          >= 0 integer; # Num old builds to demo
var numNewBuilds               {NewBuildTypes} >= 0 integer; # Num new builds to create

# Variables to hold total cost of new builds over all types
var newBuildTotalCost = sum{b in NewBuildTypes} ( (numNewBuilds[b] * newBuildCost[b]));

# Variables to hold total cost of old demolitions
var oldDemoTotalCost = (numOldBuildsDemod * demoCost) ;

# Variable to hold the sum of all new build types over all New build types
var sumOfNewBuilds = sum{b in NewBuildTypes}( numNewBuilds[b] );

# OBJECTIVE FUNCTION =====
maximize taxRevenue: sum{b in NewBuildTypes}( numNewBuilds[b] * newBuildTax[b] );

# CONSTRAINTS =====

# C1 Spend less than or equal to the federal budget
s.t. meetTheBudget:
    newBuildTotalCost + oldDemoTotalCost <= budget ;

# C2 Can only produce new builds using the demolished buildings land
```

```

s.t. useAvailLand:
    sum{b in NewBuildTypes}( numNewBuilds[b] * newBuildSpace[b] )
    <= numOldBuildsDemod * freedUpSpace ;

# C3 Can only clear a certain amount of old buildings
s.t. maxBuildingsCleared: numOldBuildsDemod <= maxBuildingDemod ;

# C4 For each new build type (b in NewBuildTypes),
# the percentage share of the new build type must meet the minimum required
s.t. share {b in NewBuildTypes}:
    numNewBuilds[b] >= newBuildPercShare[b] * sumOfNewBuilds ;

# CONTROLS =====
data problem1.dat; # retrieve data file with sets/param. values
solve;

print;

print "Number of old buildings to demolish and cost (dollars):";
display numOldBuildsDemod, oldDemoTotalCost ;

print "Number of new buildings produced and cost (dollars):";
display numNewBuilds , newBuildTotalCost ;

print "Total Budget Used (dollars):";
display newBuildTotalCost + oldDemoTotalCost ;

print "Part 3: Max Tax Revenue generated (dollars):";
display taxRevenue ;

```

1.2.2 Part 2/3: Solve AMPL Model and Display Solution

```

CPLEX 20.1.0.0: optimal integer solution; objective 199000
6 MIP simplex iterations
0 branch-and-bound nodes

Number of old buildings to demolish and cost (dollars):
numOldBuildsDemod = 137
oldDemoTotalCost = 548000

Number of new buildings produced and cost (dollars):
numNewBuilds [*] :=
  Duplex 62
  Homes  17
  MiniPark 6
;

newBuildTotalCost = 14450000

Total Budget Used (dollars):
newBuildTotalCost + oldDemoTotalCost = 14998000

Part 3: Max Tax Revenue generated (dollars):
taxRevenue = 199000

```

2 - Question 2 (Version 6)

2.1 Code to get Root Node

Root Node is Node 1 in the diagram

2.1.1 Root Node AMPL Model *problem2.mod*

```
reset;
option solver cplex; # Solver

var x >= 0; # Not integer because this is used to check the optimal when fixing a var
var y >= 0; # ""

# Original problem:
maximize theSolution: 2.5*x + 6*y;
    s.t. first: 3*x + 5*y <= 26;
    s.t. second: x >= 4;

# Used to check the BnB nodes
# Fixing x = 4 to solve for initial values of y
s.t. checkNode: x = 4;

# Solve and display
solve;
display theSolution, x,y;
```

2.1.2 Output of Root Node AMPL Model (fixing x=4 & solving for y)

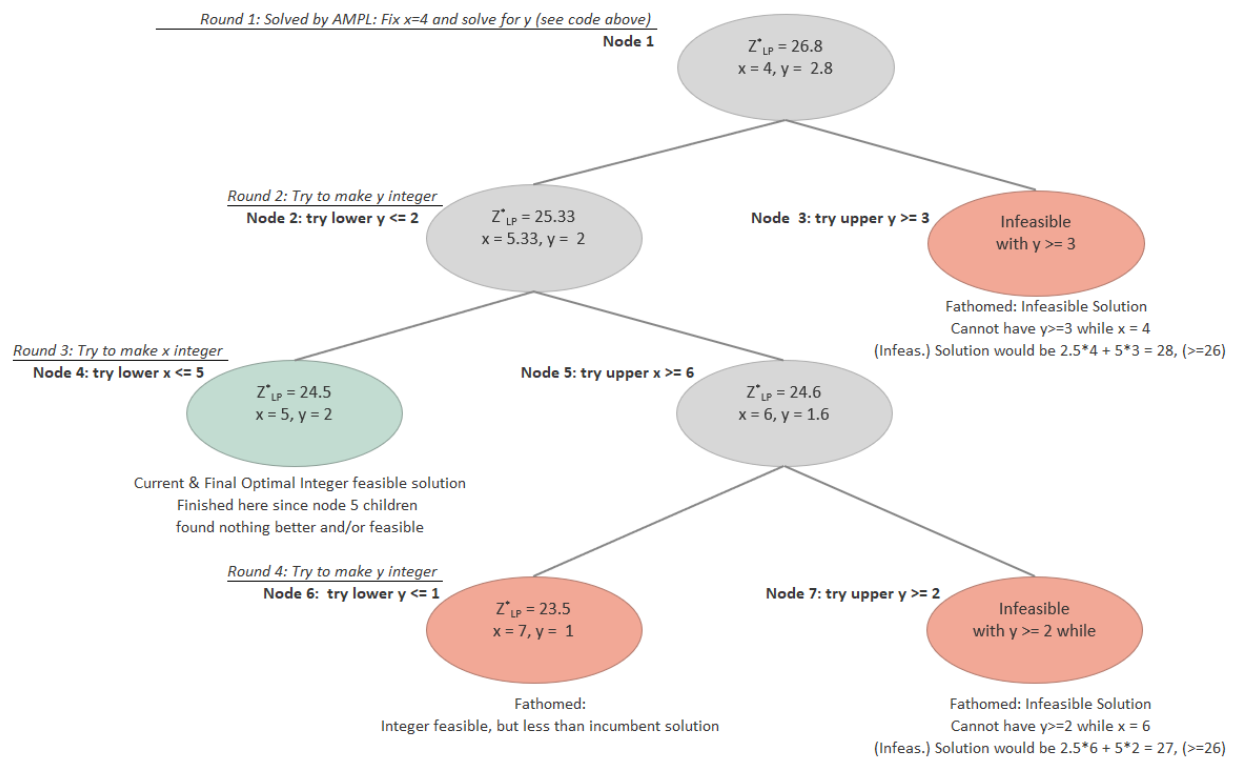
```
CPLEX 20.1.0.0: optimal solution; objective 26.8
0 dual simplex iterations (0 in phase I)
theSolution = 26.8
x = 4
y = 2.8
```

2.2 Branch and Bound Diagram

2.2.1 Summary of Diagram

- Optimal solution reached at node 4 with integer feasible values of $x = 5$, and $y = 2$
- Checked 7 total nodes
- Fathomed 3 nodes (see description of “why” in diagram)
- Each “Round” label shows which variable is being checked and for what integer value (lower or upper bound of the parent node)

2.2.2 Branch and Bound Diagram



3 - Question 3 (Version 2)

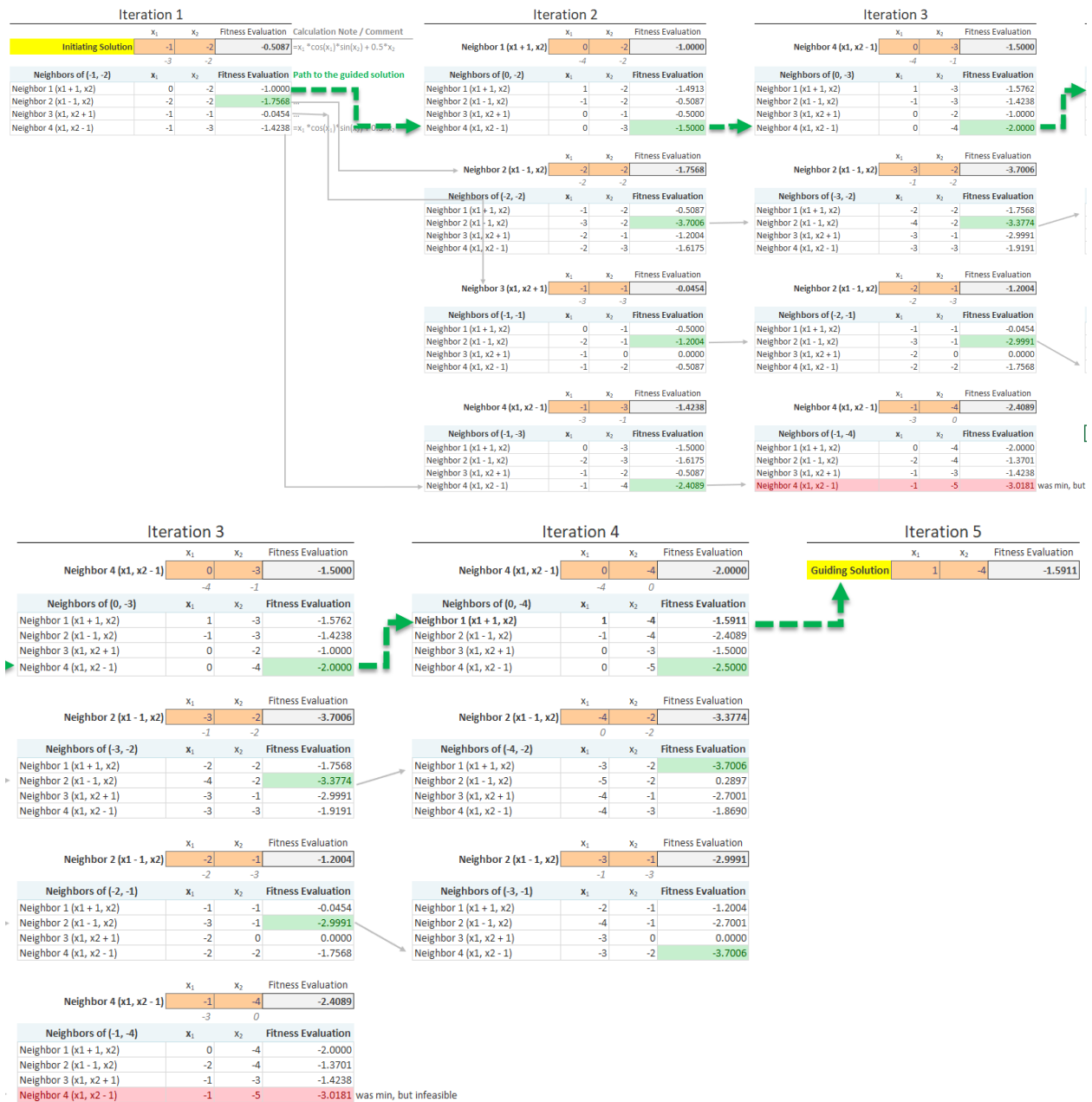
3.1 Part i: Hill Climbing

- Please see the below snippet on rightmost comments about the algorithm
- Ended at iteration 3 with minimum value of -1.9991 due to no change in best neighbor

	x_1	x_2	Fitness Evaluation	Calculation Note / Comment
Starting Solution, S_0	2	2	0.2432	$=x_1 * \cos(x_1) * \sin(x_2) + 0.5 * x_2$
Current Solution	x_1	x_2	Fitness Evaluation	
	2	2	0.2432	$=x_1 * \cos(x_1) * \sin(x_2) + 0.5 * x_2$
Done?	FALSE			
Iteration 1				
Best Neighbor (current)	2	2	0.2432	current solution from past iteration
Neighbors of (2, 2)	x_1	x_2	Fitness Evaluation	
Neighbor 1 ($x_1 + 1, x_2$)	3	2	-1.7006	$=x_1 * \cos(x_1) * \sin(x_2) + 0.5 * x_2$
Neighbor 2 ($x_1 - 1, x_2$)	1	2	1.4913	...
Neighbor 3 ($x_1, x_2 + 1$)	2	3	1.3825	...
Neighbor 4 ($x_1, x_2 - 1$)	2	1	-0.2004	...
(New) Best Neighbor	3	2	-1.7006	$=\min(\text{neighbors}, \text{best_neighbor})$
is Best Neighbor Same as Old?	FALSE			
Done?	FALSE			done if same
Current Solution	x_1	x_2	Fitness Evaluation	
	3	2	-1.7006	$=x_1 * \cos(x_1) * \sin(x_2) + 0.5 * x_2$
Iteration 2				
Best Neighbor (current)	3	2	(1.7006)	current solution from past iteration
Neighbors of (3, 2)	x_1	x_2	Fitness Evaluation	
Neighbor 1 ($x_1 + 1, x_2$)	4	2	-1.3774	$=x_1 * \cos(x_1) * \sin(x_2) + 0.5 * x_2$
Neighbor 2 ($x_1 - 1, x_2$)	2	2	0.2432	...
Neighbor 3 ($x_1, x_2 + 1$)	3	3	1.0809	...
Neighbor 4 ($x_1, x_2 - 1$)	3	1	-1.9991	...
(New) Best Neighbor	3	1	-1.9991	$=\min(\text{neighbors}, \text{best_neighbor})$
is Best Neighbor Same as Old?	FALSE			
Done?	FALSE			done if same
Current Solution	x_1	x_2	Fitness Evaluation	
	3	1	-1.9991	$=x_1 * \cos(x_1) * \sin(x_2) + 0.5 * x_2$
Iteration 3				
Best Neighbor (current)	3	1	(1.9991)	current solution from past iteration
Neighbors of (3, 1)	x_1	x_2	Fitness Evaluation	
Neighbor 1 ($x_1 + 1, x_2$)	4	1	-1.7001	$=x_1 * \cos(x_1) * \sin(x_2) + 0.5 * x_2$
Neighbor 2 ($x_1 - 1, x_2$)	2	1	-0.2004	...
Neighbor 3 ($x_1, x_2 + 1$)	3	2	-1.7006	...
Neighbor 4 ($x_1, x_2 - 1$)	3	0	0.0000	...
(New) Best Neighbor	4	1	-1.9991	$=\min(\text{neighbors}, \text{best_neighbor})$
is Best Neighbor Same as Old?	TRUE			
Done?	TRUE			done if same: end if end while

3.2 Part ii: Path Relinking

- Below shows the path re-linking overview.
- Shows each iterations that occurred in order to find the guided neighbor
- Green arrow highlights the, best solutions found along the path
- Logic overview:** Evaluates neighbors from initiation solution, then makes the best move. Updates the difference between current and guided solution. Repeats until finding the guided solution
- Iteration 3 duplicated for ease of viewing*



3.3 Part iii: Simulated Annealing

- Current temperature = 3

Using the evaluation function for this questions

```
# Evaluation function
evaluateFitness <- function(x1, x2) {
  x1*cos(x1)*sin(x2) + 0.5*x2
}
```

- The probability fomula is the following: $p = e^{\frac{-(f(s_1)-f(s_2))}{T}}$
- Where $f(s_1)$ is the current solution, and $f(s_2)$ is the candidate solution since minimization.

```
# Uses the evaluation function to return the probability (see above formula)
calculateProb <- function(temperature, currentPosition, newPosition) {
  f_s1 = evaluateFitness(currentPosition[1], currentPosition[2]) # f(s[1]) current
  f_s2 = evaluateFitness(newPosition[1],      newPosition[2]      ) # f(s[2]) candidate

  # Probability using minimization
  p = exp(1)^(- (f_s1-f_s2) / temperature )

  return(round(p, 3))
}
```

Now calculate the probability from the current to the candidate solutions

```
# What is the probability of accepting a move from
# (4, 0) to candidate neighbor solution (4, 1)?
paste('At temp. 3, the probability from (4, 0) to (4, 1) is',
      calculateProb(temperature = 3,
                    currentPosition = c(4, 0),
                    newPosition = c(4, 1)) )
```

```
## [1] "At temp. 3, the probability from (4, 0) to (4, 1) is 0.567"
```

```
# What is the probability of accepting a move from
# (4, 0) to candidate neighbor solution (4,-1)?
paste('At temp. 3, the probability from (4, 0) to (4, -1) is',
      calculateProb(temperature = 3,
                    currentPosition = c(4, 0),
                    newPosition = c(4, -1)) )
```

```
## [1] "At temp. 3, the probability from (4, 0) to (4, -1) is 1.762"
```

4 - *Question 4* (Version 3)

5 - *Question 5* (Version 2)