13. Integer Linear Programming and Introduction to Approximation

CPSC 535

Kevin A. Wortman

CALIFORNIA STATE UNIVERSITY FULLERTON

Recall: General LP Problem

general-form linear programming problem
input:

- Boolean for whether f is maximized/minimized
- vector $c \in \mathbb{R}^n$
- vector $b \in \mathbb{R}^m$
- vector $o \in \{\leq, =, \geq\}^m$
- $m \times n$ matrix A of real numbers

output: one of

1. "unbounded";
2. "infeasible"; or
3. "solution" with a vector $x \in \mathbb{R}^n$ maximizing the objective function

Recall: Reduction Algorithm

```
function SOLVE-A(input-for-A)
    input-for-B = pre-process input-for-A
    solution-for-B = solve-B(input-for-B)
    solution-for-A = post-process solution-for-B
    return solution-for-A
end function
```

Algorithms that Reduce to LP

```
function SOLVE-A(input-for-A)
    program = convert input-for-A to linear program
    result = solve-B(input-for-B)
    if result is "unbounded" then
        return output-for-unbounded-case
    else if result is "infeasible" then
        return output-for-infeasible-case
    else
        solution-for-A = convert LP solution
        return solution-for-A
    end if
end function
```

Integer Linear Programming

- **integer** *linear programming:* like general form, but all variables are integers instead of real
- i.e. each $x_i \in \mathbb{Z}$
- *Mixed Integer Programming (MIP):* mixture of real and integer variables
- i.e. a subset $I \subseteq \{x_1, \dots, x_n\}$ of variables are restricted to integers

MIP problem

mixed-integer programming problem (MIP)

input:

- Boolean for whether f is maximized/minimized
- vector $c \in \mathbb{R}^n$
- vector $b \in \mathbb{R}^m$
- vector $o \in \{\leq, =, \geq\}^m$
- $m \times n$ matrix A of real numbers
- set $I \subset \{1, \ldots, n\}$

output: one of

1. "unbounded";
2. "infeasible"; or
3. "solution" with a vector $x \in \mathbb{R}^n$ maximizing the objective function; if $i \in I$ then $x_i \in \mathbb{Z}$

MIP Applications

- **discrete variables:** can formulate a business-logic whole number concept with
  - variable $x_i, i \in I$
  - example: you can buy 3 or 4 airplanes but not 3.7
- **true/false decision:** can formulate a true/false choice with
  - variable $x_i,\ i \in I$
  - constraints $0 \leq x_i$ and $x_i \leq 1$
- **choose among** k **alternatives:** more generally, can formulate a choice from $\{a, \dots, b\} \subset \mathbb{Z}$ with
  - variable $x_i, i \in I$
  - constraints $a \leq x_i$ and $x_i \leq b$

MIP Hardness

- Recall: hardness of general LP is an open question
- not proven in P, not proven NP-hard
- MIP **is** NP-complete
- specifying integer variables seems to make the problem substantially harder
- worst-case MIP programs are intractible
- **but** MIP solvers use lots of clever heuristics
- so specific MIP formulations are often computationally feasible in practice

Vertex Cover

vertex cover problem
input: an undirected graph $G = (V, E)$
output: a vertex cover C of minimum size

vertex cover: a subset $C \subseteq V$ such that, if $(u, v) \in E$, then $u \in C$ or $v \in C$ (or both)

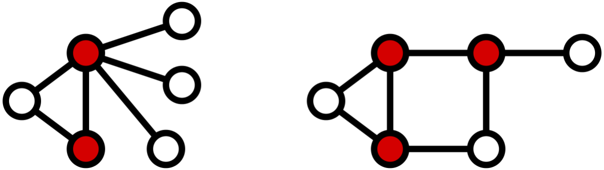


Image credit: https://commons.wikimedia.org/wiki/File:Minimum-vertex-cover.svg

Formulating Vertex Cover

Recall:

- vertex cover is NP-complete
- if vertex cover can be formulated as a MIP problem, then MIP is NP-hard

"Rules" to represent:

- each vertex is either in C or not
- each edge has at least one end in C
- minimize $|C|$

Formulating Vertex Cover

Variables: for each $v \in V$, create an integer variable x_v such that

$$x_v = 1 \Leftrightarrow v \in C$$

Objective: minimize

$$\sum_{v \in V} x_v$$

Constraints:

$$\begin{array}{lll} 0 \le x_v \le 1 & \forall v \in V & \text{(0 or 1 indicator)} \\ x_u + x_v \ge 1 & \forall (u, v) \in E & \text{(each edge is covered)} \end{array}$$

Vertex Cover Outcomes

- **Infeasible:**
  - never happens
  - $\exists$ a solution: setting all $x_v = 1$ satisfies all constraints
- **Unbounded:**
  - never happens
  - objective is bounded: the objective function is to minimize

$$\sum_{v \in V} x_v;$$

    since every $x_v \geq 0$, the minimum objective value is zero, which is finite, so the program is never unbounded
- **Solution:** Construct C as

$$C = \{v \mid v \in V \text{ and } x_v = 1\}$$

TSP

traveling salesperson problem (TSP)
input: a complete, weighted, undirected graph $G = (V, E)$
output: a tour T of minimum weight

tour: a sequence of vertices $\langle t_1, \ldots, t_n \rangle$ that visits each vertex exactly once *Hamiltonian cycle*

Define:

$$n \equiv |V|$$

$$w(u, v) \equiv \text{ the weight of the edge from } u \text{ to } v$$

Formulating TSP

"Rules" to represent:

- each vertex is visited exactly once
- minimize total weight

Formulating TSP

Variables: for each $u \in V$ and $v \in V$, create an integer variable $x_{u,v}$ such that

$$x_{u,v} = 1 \Leftrightarrow \text{ the tour steps from } u \text{ to } v$$

Objective: minimize

$$\sum_{u,v \in V} w(u,v) \cdot x_{u,v}$$

Constraints:

$$\begin{array}{lll} 0 \leq x_{u,v} \leq 1 & \forall u, v \in V & \text{(0 or 1 indicator)} \\ \sum_{u \in V} x_{u,v} = 1 & \forall v \in V & \text{(each vertex is entered once)} \\ \sum_{v \in V} x_{u,v} = 1 & \forall u \in V & \text{(each vertex is exited once)} \\ \sum_{u,v \in V} x_{u,v} = n & & \text{(tour has } n \text{ edges)} \end{array}$$

TSP Outcomes

- **Infeasible:**
  - never happens
  - $\exists$ a solution: G is complete, so certainly contains at least one tour
- **Unbounded:**
  - never happens
  - objective is bounded: observe that $\sum_{u,v \in V} w(u,v) \cdot x_{u,v}$ is minimized when every $x_{u,v}$ is zero; so the minimum objective value is zero; which is finite.
- **Solution:** Construct $T = \langle t_1, \ldots, t_n \rangle$ as

$$t_i = \begin{cases} \text{an arbitrary } v \in V & i = 1 \\ v \text{ such that } x_{t_{i-1},v} = 1 & i > 1 \end{cases}$$

Formulating Sudoku

Sudoku: input is a 9x9 grid, some cells are integers $\{1, \ldots, 9\}$, others are blank

			2	6		7		1
6	8			7			9	
1	9				4	5		
8	2		1				4	
		4	6		2	9		
	5				3		2	8
		9	3				7	4
	4			5			3	6
7		3		1	8			

Rules:

1. Objective: fill every blank
2. Each row contains $\{1, \ldots, 9\}$
3. Each column contains $\{1, \ldots, 9\}$
4. Each 3×3 subgrid contains $\{1, \ldots, 9\}$
5. (implies none of these regions has duplicates)

Formulating Sudoku: Variables

Create binary decision variables

$$x_{ijv} = 1 \Leftrightarrow \text{ row } i, \text{ column } j, \text{ is assigned value } v$$

Specify that every x_{ijv} is an integer variable.

Add constraints for the variables to be used properly:

$$\begin{array}{lll} 0 \leq x_{ijv} \leq 1 & \forall i, j, v \in \{1, \ldots, 9\} & \text{(0 or 1 indicator)} \\ \sum_{v=1}^{9} x_{ijv} = 1 & \forall i, j \in \{1, \ldots, 9\} & \text{(each cell has exactly one value)} \end{array}$$

Rule 1: Pre-Filled Cells

For each pre-filled cell at row i, column j, filled with value v, add one constraint

$$x_{ijv} = 1$$

Rules 2, 3: Each Row, Column is Filled Properly

"Row i is filled in properly" $\Leftrightarrow$ each value v appears exactly once in row i
(and for columns, resp.)

Add constraints:

$$\begin{array}{lll} \sum_{j=1}^{9} x_{ijv} = 1 & \forall i, v \in \{1, \dots, 9\} & \text{rows are filled properly} \\ \sum_{i=1}^{9} x_{ijv} = 1 & \forall j, v \in \{1, \dots, 9\} & \text{columns are filled properly} \end{array}$$

Rule 4: Each Subgrid is Filled Properly

For $r, c \in \{1, 2, 3\}$, let

$$G(r, c) = \{(i, j) : i, j \in \{1, \ldots, 9\} \text{ and } (i, j) \text{ is a cell of subgrid } r, c\}.$$

Add constraints:

$\sum_{(i,j)\in G(r,c)} x_{ijv} = 1 \quad \forall v \in \{1, \ldots 9\}; r, c \in \{1, 2, 3\}$ subgrids

Objective Function

- Those constraints model all the rules of Sudoku!
- Still need an objective function
- Sudoku does not involve minimizing or maximizing anything
- Any arbitrary objective function works
- Define objective:
  maximize 0

Outcomes of MLP

- **Infeasible:**
  - it is impossible to fill the grid without breaking a rule
  - the pre-filled cells must break a rule and be invalid
- **Unbounded:** the objective function
  maximize 0
  is a constant function, so is certainly bounded. So our MIP will never be unbounded.
- **Solution:** To fill in the grid: for each row i and column j, search for the v such that

$$x_{ijv} = 1$$

and then write v into cell (i, j).

Integer Linear Programming References

https://en.wikipedia.org/wiki/Integer_programming

http://profs.sci.univr.it/~rrizzi/classes/PLS2015/sudoku/doc/497_Olszowy_Wiktor_Sudoku.pdf

https://towardsdatascience.com/using-integer-linear-programming-to-solve-sudoku-puzzles-15e9d2a70baa

https://dingo.sbs.arizona.edu/~sandiway/sudoku/examples.html

Big Idea: Renegotiating Problems

Sometimes we want to solve a problem, but there is an obstacle

- computational complexity: problem is *NP*-hard or undecidable
- ill-posed: don't know how to phrase problem as precise input/output statement

These are insurmountable; progress is impossible.

Big Idea: Renegotiating Problems

Sometimes we can *negotiate* on the definition of the problem

- adjust input/output def'n to correspond to an easier problem
- more specific input
- or, more general output
- ideally, still helps with the domain problem
- combines CS hard skills with domain soft skills

Approximation

Approximation: output is *nearly-optimal* but not necessarily truly optimal.

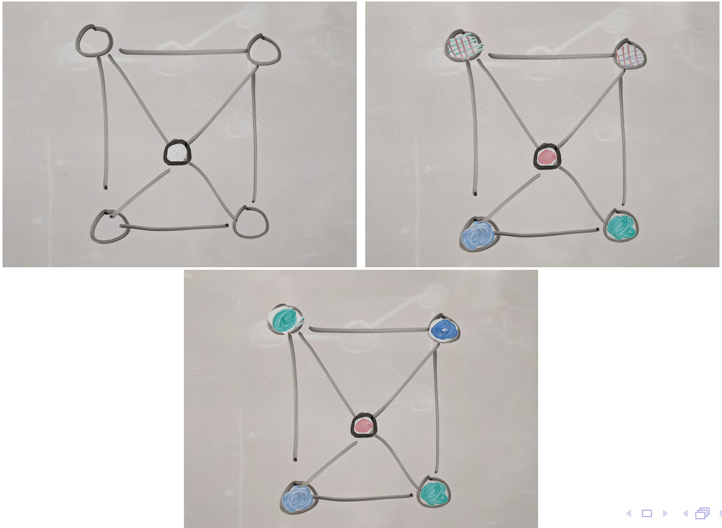
- quality is quantified, **proven**
- "approximation", "approximate" are technical terms; use other words like "decent" for informal ideas about quality
- suitable for use cases where approximate solutions are adequate
- need to rewrite problem definition
- every optimization problem has corresponding approximation problems; but these are distinct problems

Example: optimal vs. approximate graph coloring

graph coloring
input: connected graph $G = (V, E)$
output: coloring c using k colors, where each vertex $v \in V$ is assigned color $c(v) \in \{1, \ldots k\}$, no pair of adjacent vertices are assigned the same color, and the number of colors k is minimal

3-approximate graph coloring
input: connected graph $G = (V, E)$
output: coloring c using k colors, where each vertex $v \in V$ is assigned color $c(v) \in \{1, \ldots k\}$, no pair of adjacent vertices are assigned the same color, **and the number of colors k satisfies $k \leq 3k^\star$, where $k^\star$ is the fewest colors possible for G**

Graph Coloring Example



Approximation vs. Other Renegotiation Approaches

Other ways of dealing with hard problems:

- say "no"
- when n is tiny, settle for exponential-time algorithm
- no *proof* of solution quality, but sometimes good enough:
  - machine learning algorithms (also, humans don't need to precisely define what counts as "correct")
  - heuristic algorithms
  - Monte Carlo algorithms

Approximation

- pros: *provable* solution quality, often fast
- con: relatively difficult alg. design and analysis

Performance Ratios

Approximation ratio $\rho(n)$: ratio between quality of algorithm's output and optimal output

- smaller ratio is better
- 1 is perfect
- ρ is defined differently for minimization, maximization problems

Performance Ratio for Minimization Problem

For **minimization** problem: if optimal quality is $C^\star$ and alg. produces quality C, by definition $C^\star \leq C$ and define

$$\rho(n) = \frac{C}{C^\star}$$

Recall 3-approx. vertex cover: $\#$ colors $\leq 3k^\star$

Performance Ratio for Maximization Problem

For **maximization** problem: if optimal quality is $C^\star$ and alg. produces quality C, by definition $C^\star \geq C$, and define

$$\rho(n) = \frac{C^\star}{C}$$

(Reciprocal of previous definition.)

Consider approximate matching.

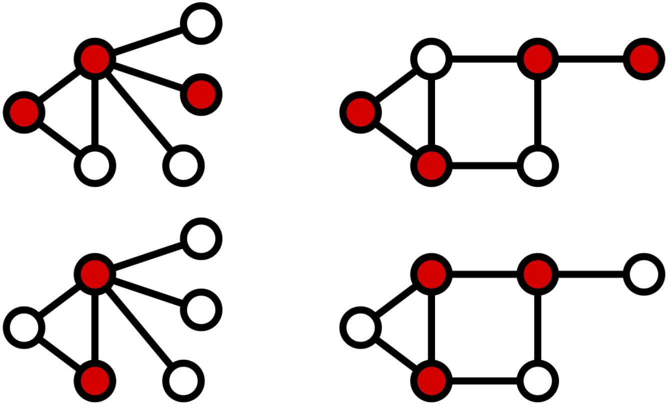
Vertex Cover Problem

vertex cover problem
input: undirected graph $G = (V, E)$
output: set of vertices $C \subseteq V$, of minimal size $|C|$, such that every edge in E is incident on at least one vertex in C

2-approximate vertex cover problem
input: undirected graph $G = (V, E)$
output: set of vertices $C \subseteq V$, such that every edge in E is incident on at least one vertex in C, and $|C| \leq 2|C^\star|$ where $C^\star$ is a minimal vertex cover for G

See Wiki page: https://en.wikipedia.org/wiki/Vertex_cover

Vertex Cover Example



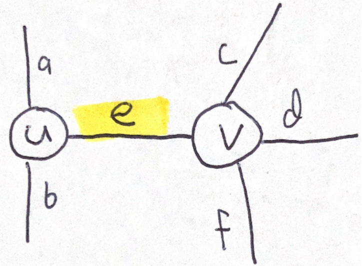
Images credit: Wikipedia user Miym, CC BY-SA 3.0, https://commons.wikimedia.org/wiki/File:Vertex-cover.svg, https://commons.wikimedia.org/wiki/File:Minimum-vertex-cover.svg

Vertex Cover Hardness

- vertex cover is *NP*-complete
- baseline algorithm:
  - exhaustive search
  - for each subset C of vertices, check whether every edge has an endpoint in C
  - return the smallest C that is a valid cover
  - $\Theta(2^n m)$ time, exponential, slow
- goal of a 2-approximate vertex cover algorithm:
  - get a decent (though imperfect) cover much faster

A Greedy Approximation Algorithm

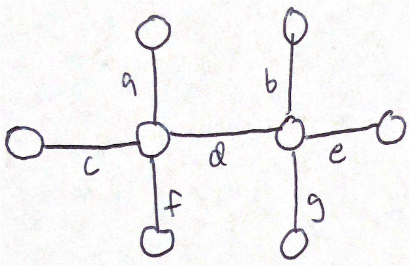
- every edge $e = (u, v)$ needs both $u \in C$ and $v \in C$
- so grab an edge $e = (u, v)$ and include u and v in C
- every other edge touching u or v is now covered, so eliminate them
- continue until every edge is either grabbed or eliminated



A Greedy Approximation Algorithm

- good: definitely finds a correct cover C
- bad: depending on the order of the "grabs", heuristic can get tricked into picking sub-optimal vertices

Example: Greedy Can Be Suboptimal



optimal edge choices: d

suboptimal edge choices: a, b (and others)