Disconnecting Networks via Node Deletions

Exact Interdiction Models and Algorithms

Siqian Shen[1] J. Cole Smith[2] R. Goli[2]

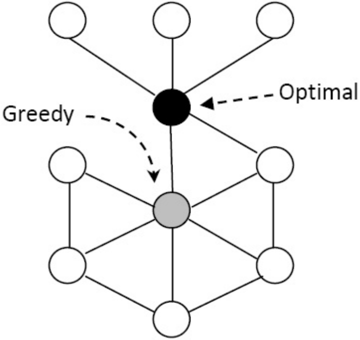
[1]IOE, University of Michigan
[2]ISE, University of Florida

2012 INFORMS Optimization Society Conference, Miami FL

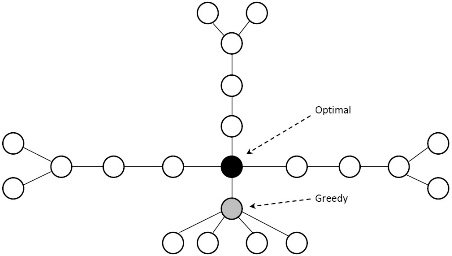
Outline

MaxNum and MinMaxC on General Graphs? ($B = 1$)

Counterexamples:



MaxNum



MinMaxC

Motivation and Contributions

- The MaxNum and MinMaxC on general graphs: $\mathcal{NP}$-hard.
- The MaxNum and MinMaxC on specially structured graphs: Polynomial-time Dynamic Programming Algorithms (Shen and Smith (2011))
- This study will:

Motivation and Contributions

- The MaxNum and MinMaxC on general graphs: $\mathcal{NP}$-hard.
- The MaxNum and MinMaxC on specially structured graphs: Polynomial-time Dynamic Programming Algorithms (Shen and Smith (2011))
- This study will:
  1. Formulate **two-stage** interdiction MIPs having **LP** subproblems
  2. Take the subproblem **dual**s, and **integrate** the two stages
  3. **Linearize** the monolithic MIP, and solve it to optimality

Motivation and Contributions

- The MaxNum and MinMaxC on general graphs: $\mathcal{NP}$-hard.
- The MaxNum and MinMaxC on specially structured graphs: Polynomial-time Dynamic Programming Algorithms (Shen and Smith (2011))
- This study will:
  1. Formulate **two-stage** interdiction MIPs having **LP** subproblems
  2. Take the subproblem **dual**s, and **integrate** the two stages
  3. **Linearize** the monolithic MIP, and solve it to optimality
  4. Reformulate the MIP based on **subgraph partitions** of G, and **generate valid inequalities** by using intermediate polynomial-time optimal **DP solutions** from each partition.

Master Problem (MaxNum)

$$\max \quad \left\{ \eta(x,y) - \frac{1}{n}\sum_{i=1}^{n}(1-x_i) \right\} \tag{1a}$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{V}}(1-x_i) \le B \tag{1b}$$

$$x_i + x_j - 1 \le y_{ij} \quad \forall (i,j)\in\mathcal{E} \tag{1c}$$

$$x_i \in \{0,1\} \quad \forall i \in \mathcal{V} \tag{1d}$$

$$0 \le y_{ij} \le 1 \quad \forall (i,j)\in\mathcal{E}, \tag{1e}$$

- Undirected graph $G(\mathcal{V},\mathcal{E})$, where $\mathcal{V} = \{1,\dots,n\}$ and $\mathcal{E} \subset \mathcal{V}\times\mathcal{V}$
- $\eta(x,y)$: Subproblem objective, e.g., number of components for MaxNum
- $x_i \in \{0,1\}$: $x_i = 1$ if node i is not deleted, and $x_i = 0$ if i is deleted
- $y_{ij} \in \{0,1\}$: $y_{ij} = 1$ if edge (i,j) exists, and $y_{ij} = 0$ otherwise ($y_{ij} = x_i x_j$)
- B: Given node deletion budget (positive integer)

MaxNum Subproblem: Solving $\eta(x, y)$

- Formulate on a directed transformation network $\widetilde{G}(\mathcal{N}, \mathcal{A})$
- Design a dummy node 0 and a unit cost for constructing arc $(0, i), \ \forall i \in \mathcal{V}$
- **GOAL**: To flow $|\widetilde{\mathcal{V}}|$ paths from 0 to every active node $i \in \widetilde{\mathcal{V}}$
- **Decision Variables:** z_i: $= 1$ if $(0, i)$ is constructed and $= 0$ otherwise; f_{ijk}: Flow on arc (i, j) with respect to path 0–k

$$\eta(x, y) = \min \quad \sum_{i \in \mathcal{N}} z_i \tag{2a}$$

$$\text{s.t.:} \quad |\widetilde{\mathcal{V}}| \text{ paths from node 0 to every active node } i \tag{2b}$$

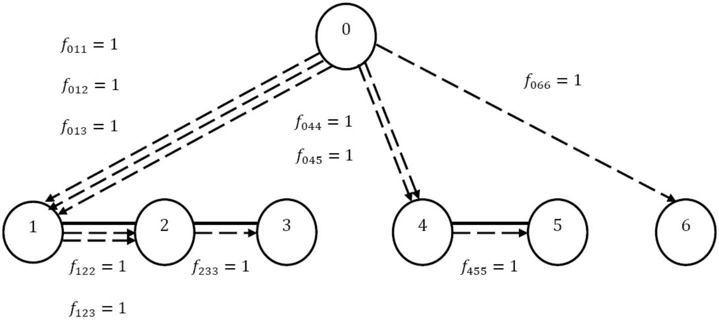
$$-f_{0ik} + z_i \geq 0 \quad \forall i,\ k \in \mathcal{N} \tag{2c}$$

$$-f_{ijk} \geq -y_{ij} \quad \forall (i,j) \in \mathcal{A},\ k \in \mathcal{N} \tag{2d}$$

$$z_i \in \{0, 1\}, \quad f_{ijk} \geq 0. \tag{2e}$$

MaxNum Subproblem: Solving $\eta(x, y)$

A transformed directed graph and a feasible solution illustration:



Solving MaxNum

- Good News:)

  Fix (x, y) at binary values, and a subproblem LP gives the convex hull in terms of variables z.

- Solution Scheme:
  - **Replace** $\eta(x, y)$ in the master problem by the subproblem LP dual
  - **Linearize** bilinear terms of "$x \times$ duals" and "$y \times$ duals" by using McCormick inequalities (since both x and y are binary-valued).
  - Monolithically solve MaxNum in a "max{max} = max" framework

MinMaxC

- The master problem is similar to MaxNum except an obj modification:

$$\min\left\{\eta'(x,y) + \frac{1}{n}\sum_{i=1}^{n}(1-x_i) : \text{(1b)–(1e)}\right\}, \tag{2}$$

where $\eta'(x,y)$ represents the largest component size for a given (x,y).

- Subproblem Notation:
  - $\sigma_{ik} \in \{0,1\}$: $= 1$ if nodes i and k belong to the same component
  - $\sigma_{kk} = 1, \ \forall k \in \mathcal{N}$
  - $\lambda = \eta'(x,y)$ represents the largest component size

MinMaxC: A Monolithic Model

$$\min \quad \left\{ \lambda + \frac{1}{n}\sum_{i=1}^{n}(1 - x_i) \right\} \tag{3a}$$

$$\text{s.t.} \quad \text{(1b)–(1e), and } \sigma_{kk} = 1 \quad \forall k \in \mathcal{N}$$

$$\lambda \geq \sum_{i \in \mathcal{N}} \sigma_{ik} \quad \forall k \in \mathcal{N} \tag{3b}$$

$$\sigma_{jk} - \sigma_{ik} \geq y_{ij} - 1 \quad \forall (i,j) \in \mathcal{A},\ k \in \mathcal{N} \tag{3c}$$

$$\sigma_{ik} \in \{0,1\} \quad \forall i,\ k \in \mathcal{N}. \tag{3d}$$

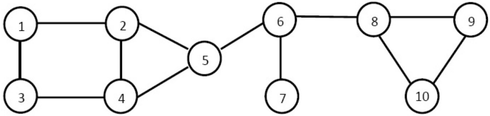
- (3b) enforces λ to be the largest component size
- (3c) pushes $\sigma_{jk} = 1$ if $\sigma_{ik} = 1$ and $y_{ij} = 1$. That is, nodes j and k are in the same component, if nodes i and k are in the same component and j is connected to i
- (3) yields **the convex hull** even with (3d) being linear.

How efficient the Monolithic MIP models are?

- Experimental Tests:
  - CPLEX 11.0 & C++; a Dell PowerEdge 2600 UNIX machine with two 3.2 GHz processors; a one-hour time limit
  - Five 20-node (having 40 - 60 arcs) and five 30-node (having 100-200 arcs) graph instances with varied *B*-values
- Result Observations:
  - CPU time: 10s-100s for most 20-node instances; 100s-800s for 30-node instances
  - CPU time $\uparrow$ as $B \uparrow$ at the begining, and then CPU time $\downarrow$ as B continue to $\uparrow$ above a threshold of approximately $0.25|\mathcal{V}|$

On the other hand...

- Given a tree $T(V,E)$, a DP algorithm can solve:
  - $O(n^3)$ $\Rightarrow$ MaxNum on trees
  - $O(n^3 \log n)$ $\Rightarrow$ MinMaxC on trees
- Extend the results to k-hole-graph for some k:
  - $O(n^{3+k})$ $\Rightarrow$ MaxNum
  - $O(n^{3+k} \log n)$ $\Rightarrow$ MinMaxC



DP Algorithms for Specially-Structured Graphs

For an undirected tree $T(V,E)$,

- r: root node
- T_i: subtree rooted at node i ($T = T_r$)

DP Algorithms for Specially-Structured Graphs

For an undirected tree $T(V, E)$,

- r: root node
- T_i: subtree rooted at node i ($T = T_r$)

Key Concept:

- **Open set** O_i: All nodes in the same component to which subroot i belongs, and $o_i = |O_i|$
- If i is deleted, then O_i is empty and $o_i = 0$

DP Algorithms for Specially-Structured Graphs

For an undirected tree $T(V, E)$,

- r: root node
- T_i: subtree rooted at node i ($T = T_r$)

Key Concept:

- **Open set** O_i: All nodes in the same component to which subroot i belongs, and $o_i = |O_i|$
- If i is deleted, then O_i is empty and $o_i = 0$

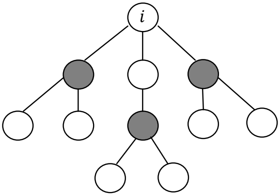
Incumbent Initial Step:

There exists an optimal solution to all MaxNum and MinMaxC instances on tree graphs in which NO leaf node is deleted.

$O(n^3)$ DP algorithms for MaxNum

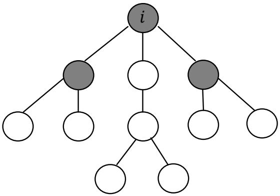
$f_i(p_i, n_i)$: the fewest number of deletions required on subtree T_i, given that

- p_i: $= 0$ if subtree root i is deleted, and $= 1$ otherwise
- n_i: Number of components created, not including O_i
- Note: $f_l(1,0) = 0$ at every leaf node $l \in V$



$f_i(1,6) = 3$

Illustration of $f_i(p_i,n_i)$ when an open set is present. Note that $n_i = 6$ here because the open set itself is not counted in n_i.



$f_i(0,5) = 3$

Illustration of $f_i(p_i,n_i)$ when no open set is present.

Update $f_i(p_i, n_i)$ given $f_v(p_v, n_v),\ \forall v \in S_i$

When $p_i = 0$ (subtree root i is deleted):

$$\begin{aligned} f_i(0, n_i) = \min \quad & \sum_{v \in S_i} f_v(p_v, n_v) + 1 \\ \text{s.t.} \quad & n_i = \sum_{v \in S_i} n_v + \sum_{v \in S_i} p_v \end{aligned}$$

Every open set O_v becomes a new component after merging.

Update $f_i(p_i, n_i)$ given $f_v(p_v, n_v), \ \forall v \in S_i$

When $p_i = 0$ (subtree root i is deleted):

$$
\begin{aligned}
f_i(0, n_i) = \min \quad & \sum_{v \in S_i} f_v(p_v, n_v) + 1 \\
\text{s.t.} \quad & n_i = \sum_{v \in S_i} n_v + \sum_{v \in S_i} p_v
\end{aligned}
$$

Every open set O_v becomes a new component after merging.

When $p_i = 1$ (not deleted):

$$
\begin{aligned}
f_i(1, n_i) = \min \quad & \sum_{v \in S_i} f_v(p_v, n_v) \\
\text{s.t.} \quad & n_i = \sum_{v \in S_i} n_v
\end{aligned}
$$

All open sets O_v will merge with O_i to form a larger-cardinality open set at i.

Update $f_i(p_i, n_i)$ given $f_v(p_v, n_v)$, $\forall v \in S_i$

When $p_i = 0$ (subtree root i is deleted):

$$f_i(0, n_i) = \min \sum_{v \in S_i} f_v(p_v, n_v) + 1$$
$$\text{s.t.} \quad n_i = \sum_{v \in S_i} n_v + \sum_{v \in S_i} p_v$$

Every open set O_v becomes a new component after merging.

When $p_i = 1$ (not deleted):

$$f_i(1, n_i) = \min \sum_{v \in S_i} f_v(p_v, n_v)$$
$$\text{s.t.} \quad n_i = \sum_{v \in S_i} n_v$$

All open sets O_v will merge with O_i to form a larger-cardinality open set at i.

- Calculate $f_i(p_i, n_i)$ by sequentially merging one subtree at a time
- Since $n_i \leq n$, computing f_i is $O(n^2)$, for all $i \in V$.
- Total complexity: $O(n^3)$ for solving MaxNum on trees.

$O(n^3 \log n)$ DP algorithms for MinMaxC

- $f_i(o_i, m_i)$: the fewest number of deletions on subtree T_i, given
    - an open set of size o_i exists on i
    - a maximum component size of m_i (excluding O_i)
- However, since both o_i and $m_i \leq n$, merging requires $O(n^5)$ steps

$O(n^3 \log n)$ DP algorithms for MinMaxC

- $f_i(o_i, m_i)$: the fewest number of deletions on subtree T_i, given
  - an open set of size o_i exists on i
  - a maximum component size of m_i (excluding O_i)
- However, since both o_i and $m_i \leq n$, merging requires $O(n^5)$ steps
- Define $f_i(o_i, \tau)$ **instead**: the fewest number of deletions on subtree T_i, given that
  - no component has a larger size than τ (a fixed target)
  - it generates an open set of size o_i where $o_i \leq \tau$
  - $f_l(1, \tau) = 0$ at every leaf node $l \in V$ for any $\tau \geq 1$.

$O(n^3 \log n)$ DP algorithms for MinMaxC

- $f_i(o_i, m_i)$: the fewest number of deletions on subtree T_i, given
  - an open set of size o_i exists on i
  - a maximum component size of m_i (excluding O_i)
- However, since both o_i and $m_i \leq n$, merging requires $O(n^5)$ steps
- Define $f_i(o_i, \tau)$ **instead**: the fewest number of deletions on subtree T_i, given that
  - no component has a larger size than τ (a fixed target)
  - it generates an open set of size o_i where $o_i \leq \tau$
  - $f_l(1, \tau) = 0$ at every leaf node $l \in V$ for any $\tau \geq 1$.
- Employ a binary-search scaling scheme over τ; update $f_i(o_i, \tau)$ for all $i \in V$ for a given τ

Update $f_i(o_i, \tau)$ given $f_v(o_v, \tau)$, $\forall v \in S_i$

When $o_i = 0$ (subtree root i is deleted):

$$f_i(0, \tau) = \min \sum_{v \in S_i} f_v(o_v, \tau) + 1.$$

The largest component size is automatically not more than τ.

Update $f_i(o_i, \tau)$ given $f_v(o_v, \tau)$, $\forall v \in S_i$

When $o_i = 0$ (subtree root i is deleted):

$$f_i(0, \tau) = \min \quad \sum_{v \in S_i} f_v(o_v, \tau) + 1.$$

The largest component size is automatically not more than τ.

When $o_i > 0$ (not deleted):

$$f_i\,(o_i, \tau) = \min \quad \sum_{v \in S_i} f_v(o_v, \tau)$$

$$\text{s.t.} \quad o_i = \sum_{v \in S_i} o_v + 1 \leq \tau.$$

Update $f_i(o_i, \tau)$ given $f_v(o_v, \tau)$, $\forall v \in S_i$

When $o_i = 0$ (subtree root i is deleted):

$$f_i(0, \tau) = \min \quad \sum_{v \in S_i} f_v(o_v, \tau) + 1.$$

The largest component size is automatically not more than τ.

When $o_i > 0$ (not deleted):

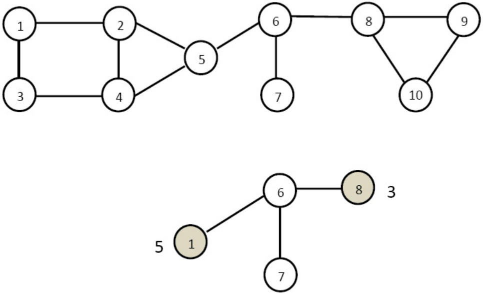
$$f_i(o_i, \tau) = \min \quad \sum_{v \in S_i} f_v(o_v, \tau)$$

$$\text{s.t.} \quad o_i = \sum_{v \in S_i} o_v + 1 \leq \tau.$$

- Initial: Upper bound $UB = n - B$; Lower bound $LB = 1$; $\tau = \lfloor \frac{n-B+1}{2} \rfloor$
- Step 1: Solve MinMaxC for a current τ ($O(n^3)$ steps)
- Step 2: Update τ: If $LB < UB$, update $\tau = \lfloor (UB + LB)/2 \rfloor$; go to Step 1 ($O(\log n)$ iterations)
- Total complexity: $O(n^3 \log n)$ for solving MinMaxC on trees.

k-hole graphs

- A hole of a graph: a set of nodes $v_1, \ldots, v_m$ such that an edge exists between v_i and v_j $(i < j)$ if and only if $i = j - 1$ or $i = 1$ and $j = m$.
- $M^1, \ldots, M^k$: the k holes in a graph, where nodes $\{v_1, \ldots, v_q\}$ compose the union of the nodes in these holes
- Transform a k-hole graph into a weighted "hole" tree



MaxNum and MinMaxC on k-hole-graphs

- Case 0 (no node is deleted in any hole)
  - Every M^j is a *hole-node* with size $|M^j|$
  - Yield a tree structure with weighted hole-nodes
  - Use the same DP recursions as before, but prohibit deletions of hole-nodes

MaxNum and MinMaxC on k-hole-graphs

- Case 0 (no node is deleted in any hole)
  - Every M^j is a *hole-node* with size $|M^j|$
  - Yield a tree structure with weighted hole-nodes
  - Use the same DP recursions as before, but prohibit deletions of hole-nodes
- Case i (delete node v_i and obtain a p-hole-graph such that $p < k$)
  - Recursively solve on a resulting p-hole-graph
  - $\Gamma(k)$ = the complexity on k-hole-graphs, we have that $\Gamma(k) = O(n\,\Gamma(k-1))$
  - Base case: 0-hole-graph (i.e., a tree)

MaxNum and MinMaxC on k-hole-graphs

- Case 0 (no node is deleted in any hole)
  - Every M^j is a *hole-node* with size $|M^j|$
  - Yield a tree structure with weighted hole-nodes
  - Use the same DP recursions as before, but prohibit deletions of hole-nodes
- Case i (delete node v_i and obtain a p-hole-graph such that $p < k$)
  - Recursively solve on a resulting p-hole-graph
  - $\Gamma(k)$ = the complexity on k-hole-graphs, we have that $\Gamma(k) = O(n\,\Gamma(k-1))$
  - Base case: 0-hole-graph (i.e., a tree)
- Complexities on k-hole-graph: $O(n^{3+k})$ for MaxNum, and $O(n^{3+k} \log n)$ for MinMaxC.

Incorporate DP Solutions into the MIP Framework

- Idea 1: Optimal DP solutions obtained on k-hole subgraphs of G provide bounds for the real subproblem objectives. However...
- Our computational results show:
  - Bounds are generally **not very tight**, but tighter on smaller G instances (i.e., 20-node as opposed to 30- and 40-node graphs)
- Idea 2: Employ a graph-partition strategy, solve the DP on each partition, and generate valid inequalities for MIPs.

Reformulating the MIP

Notation (MaxNum for instance):

- Partition graph G into m subgraphs $G_1, \ldots, G_m$
- k_i: the number of holes in each subgraph G_i, $\forall i = 1, \ldots, m$
- Execute DP on each k_i-hole subgraph G_i for a budget B $\Rightarrow$
- $\eta_i(B_i)$: maxnum obtained on G_i for deletion budgets $B_i = 0, \ldots, B$ (variables)
- $g_i(B_i)$: Piecewise-linear concave envelope function of $\eta_i(B_i)$ such that $\eta_i(B_i) \leq g_i(B_i)$ for all $B_i = 0, \ldots, B$.

Reformulating the MIP

Notation (MaxNum for instance):

- Partition graph G into m subgraphs $G_1, \ldots, G_m$
- k_i: the number of holes in each subgraph G_i, $\forall i = 1, \ldots, m$
- Execute DP on each k_i-hole subgraph G_i for a budget B $\Rightarrow$
- $\eta_i(B_i)$: maxnum obtained on G_i for deletion budgets $B_i = 0, \ldots, B$ (variables)
- $g_i(B_i)$: Piecewise-linear concave envelope function of $\eta_i(B_i)$ such that $\eta_i(B_i) \leq g_i(B_i)$ for all $B_i = 0, \ldots, B$.

Append the following valid inequalities into the MaxNum MIP:

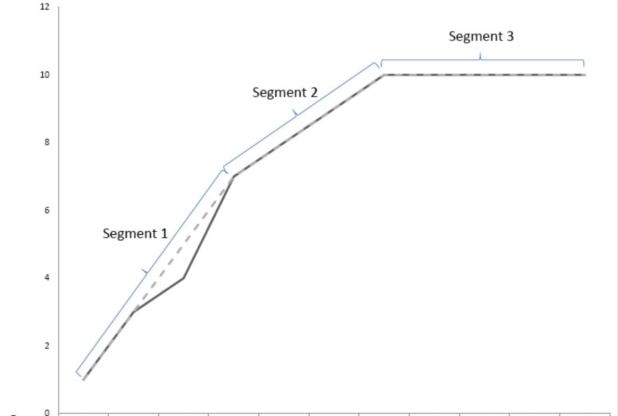
$$\eta - \sum_{i=1}^{m} \eta_i \leq 0 \tag{4a}$$

$$\eta_i - g_i(B_i) \leq 0 \quad \forall i = 1, \ldots, m \tag{4b}$$

$$B_i = \sum_{j \in V_i} (1 - x_j) \quad \forall i = 1, \ldots, m. \tag{4c}$$

Example 1: Solving MaxNum

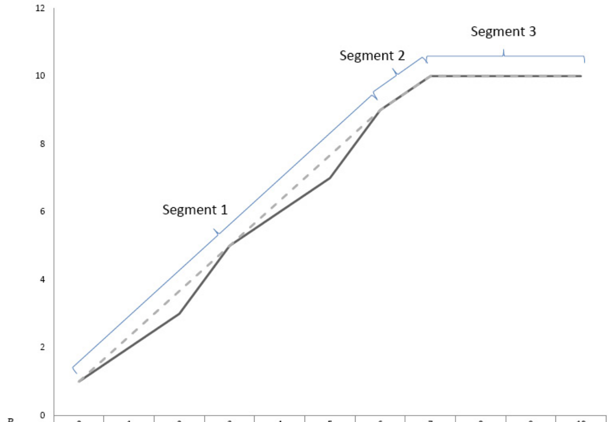
Given a 20-node graph G and $B = 10$, solving the 1st partition G_1 (10-node):



B_1	0	1	2	3	4	5	6	7	8	9	10
η_1	1	3	4	7	8	9	10	10	10	10	10
g_1	1	3	5	7	8	9	10	10	10	10	10

Example 1: Solving MaxNum

Given a 20-node graph G and $B = 10$, solving the 2nd partition G_2 (10-node):



B_2	0	1	2	3	4	5	6	7	8	9	10
η_2 (solid)	1	2	3	5	6	7	9	10	10	10	10
g_2 (dashed)	1	2.33	3.67	5	6.33	7.67	9	10	10	10	10

Example 1: Solving MaxNum

Inequalities (4a) and (4c) are:

$$\eta \leq \eta_1 + \eta_2,\ B_1 = \sum_{i \in G_1} (1 - x_i),\ B_2 = \sum_{i \in G_2} (1 - x_i). \tag{5}$$

Associated with the three-segment $g_1(B_1)$, for G_1, we generate (4b) as

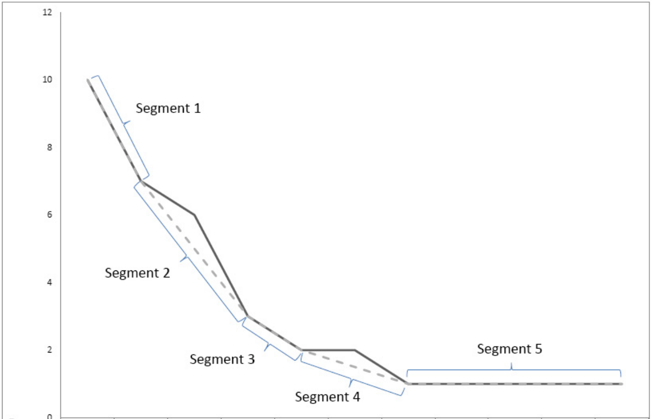
$$\eta_1 \leq 2B_1 + 1, \qquad \eta_1 \leq B_1 + 4, \qquad \eta_1 \leq 10. \tag{6}$$

Similarly, corresponding to each segment of $g_2(B_2)$, for G_2, (4b) become

$$\eta_2 \leq (4/3)B_2 + 1, \qquad \eta_2 \leq B_2 + 3, \qquad \eta_2 \leq 10. \tag{7}$$

Example 2: Solving MinMaxC

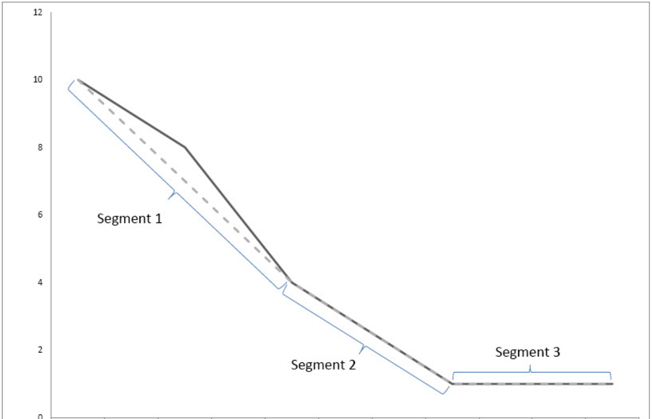
$g_i'(B_i)$ is the convex envelop of $\eta_i'(B_i)$, and signs in (4a) and (4b) are flipped.



B_1		0	1	2	3	4	5	6	7	8	9	10
η_1'	▬▬	10	7	6	3	2	2	1	1	1	1	1
g_1'	‒ ‒	10	7	5	3	2	1.5	1	1	1	1	1

Example 2: Solving MinMaxC

$g_i'(B_i)$ is the convex envelop of $\eta_i'(B_i)$, and signs in (4a) and (4b) are flipped.



B_2	0	1	2	3	4	5	6	7	8	9	10
η_2'	10	9	8	6	4	3	2	1	1	1	1
g_2'	10	8.5	7	5.5	4	3	2	1	1	1	1

Example 2: Solving MinMaxC

Inequalities (4a) and (4c) are:

$$\eta' \geq \eta_1' + \eta_2',\ B_1 = \sum_{i \in G_1} (1 - x_i),\ B_2 = \sum_{i \in G_2} (1 - x_i). \tag{8}$$

The following two sets of inequalities are generated to describe $g_i'(B_i)$, for $i = 1, 2$:

$$\begin{aligned} &\eta' \geq -3B_1 + 10,\ \eta' \geq -2B_1 + 9, \\ &\eta' \geq -B_1 + 6,\ \eta' \geq -0.5B_1 + 4,\ \eta' \geq 1 \end{aligned} \tag{9}$$

$$\eta' \geq -1.5B_2 + 10,\ \eta' \geq -B_2 + 8,\ \eta' \geq 1 \tag{10}$$

CPU Times for 20-node Instances Using 2-Partition

Instance	Prob.	$B = 4$		$B = 8$	
		Orig.	2-Partition	Orig.	2-Partition
20-1	MaxNum	24.62	[34.52]	5.94	[16.85]
	MinMaxC	16.56	8.15	1.27	[1.90]
20-2	MaxNum	49.67	43.28	79.48	42.52
	MinMaxC	8.17	6.53	16.22	12.53
20-3	MaxNum	51.94	44.24	16.34	[33.84]
	MinMaxC	19.55	15.66	13.57	[19.29]
20-4	MaxNum	41.77	[88.48]	36.81	34.13
	MinMaxC	30.71	24.06	15.26	7.72
20-5	MaxNum	71.06	54.73	21.55	[34.65]
	MinMaxC	33.40	22.19	14.76	14.49

CPU Times for 30-node Instances Using 3-Partition

Instance	Prob.	$B = 4$		$B = 8$	
		Orig.	3-Partition	Orig.	3-Partition
30-1	MaxNum	467.92	384.43	289.14	235.28
	MinMaxC	462.93	391.20	166.24	[204.12]
30-2	MaxNum	467.93	452.96	209.58	[218.07]
	MinMaxC	331.22	[334.29]	98.64	87.35
30-3	MaxNum	502.85	479.30	725.49	623.58
	MinMaxC	217.05	172.45	117.54	[121.11]
30-4	MaxNum	516.72	446.82	202.18	183.71
	MinMaxC	345.67	[351.84]	94.25	[96.36]
30-5	MaxNum	432.40	328.66	189.62	171.55
	MinMaxC	479.24	443.74	143.82	143.30

40-node Instances Using 2- and 4-Partition

None 40-node instances can be solved within a one-hour time limit. Thus, we report gaps (%) reported by CPLEX instead

Instance	Prob.	$B = 4$			$B = 8$		
		Orig.	2-Partition	4-Partition	Orig.	2-Partition	4-Partition
40-1	MaxNum	131.39%	58.12%	131.35%	87.82%	48.07%	87.79%
	MinMaxC	27.82%	11.11%	27.85%	62.47%	32.82%	[62.49%]
40-2	MaxNum	124.51%	124.51%	110.29%	84.68%	33.78%	74.97%
	MinMaxC	26.19%	6.95%	20.42%	58.52%	21.10%	[58.68%]
40-3	MaxNum	122.56%	44.99%	112.14%	85.94%	85.92%	[88.85%]
	MinMaxC	25.92%	25.77%	25.85%	58.17%	57.09%	47.38%
40-4	MaxNum	114.68%	59.95%	[128.20%]	95.47%	49.98%	86.80%
	MinMaxC	27.93%	27.93%	27.87%	61.52%	47.38%	47.50%
40-5	MaxNum	125.26%	44.99%	120.01%	84.15%	53.76%	[100.18%]
	MinMaxC	26.25%	26.21%	26.20%	59.17%	44.65%	51.80%

Future Research

- Vary partition patterns, and test the computational efficacy of different valid inequalities
- Dynamically update partitions within a branch-and-bound (B&B) tree
- The locally valid inequalities may lead to a quicker termination and more effective fathoming rules for the B&B algorithm

Thank you

Questions? ...