The A-CEEI Mechanism for Combinatorial Assignment

Duncan McElfresh
AMSC Program, Department of Mathematics
University of Maryland College Park

Advisor: Dr. John Dickerson
Department of Computer Science
University of Maryland College Park

Project Proposal
3 October, 2017
Combinatorial Assignment Problem

$i \in S \quad j \in C$

$q_a = 2 \quad q_b = 1 \quad q_c = 4$
Combinatorial Assignment Problem: Agent Preferences

\( \succ B = \{ \text{apple} \} \)

\( \succ B \)

\( i \in S \rightarrow q_a = 2 \)

\( \succ B \rightarrow q_b = 1 \)

\( \succ C \rightarrow q_c = 4 \)
Combinatorial Assignment Problem: Agent Preferences

\[ i \in S \quad j \in C \]
\[ q_a = 2 \quad \text{🍎🍎} \]
\[ q_b = 1 \quad \text{🍌} \]
\[ q_c = 4 \quad \text{🥕🥕🥕🥕} \]
Combinatorial Assignment Problem: Agent Preferences

\[ B = \succ \succ \rangle \succ \rangle \]

\[ i \in S \]
\[ j \in C \]
\[ q_a = 2 \]
\[ q_b = 1 \]
\[ q_c = 4 \]
Combinatorial Assignment Problem: Agent Preferences

\[ \succ B = \succ \text{apples} \succ \text{bananas} \succ \text{carrots} \]

\[ \succ B \]

\[ \succ C \]

\[ i \in S \quad j \in C \]

\[ q_a = 2 \quad \text{apples} \]

\[ q_b = 1 \quad \text{banana} \]

\[ q_c = 4 \quad \text{carrots} \]
Agent preferences are a total ordering over the set of all subsets of objects.
Combinatorial Assignment Problem

$N$ Agents $i \in S$

Agents receive bundles of objects (max bundle size = $k$)

Agents $i$ have preferences $\succ_i$ over bundles

$M$ Objects $j \in C$

$q_j \in \mathbb{N}$ Objects $j$ have duplicates (or capacities)
Each combinatorial assignment problem (CAP) is defined by the *economy*

\[(C, S, (q_i)_{i=1}^{M}, (\succ_i)_{i=1}^{N})\]
Each combinatorial assignment problem (CAP) is defined by the economy

\((C, S, (q_i)_{i=1}^M, (\succ_i)_{i=1}^N)\)

The solution to a CAP is an assignment \(x = (x_{ij})\), with

\[x_{ij} = \begin{cases} 1 & \text{if agent } i \text{ receives object } j \\ 0 & \text{otherwise} \end{cases}\]
Each combinatorial assignment problem (CAP) is defined by the economy:

\[(C, S, (q_i)_{i=1}^M, (r_i)_{i=1}^N)\]

The solution to a CAP is an assignment \( x = (x_{ij}) \), with

\[x_{ij} = \begin{cases} 
1 & \text{if agent } i \text{ receives object } j \\
0 & \text{otherwise} 
\end{cases}\]

We want to find an assignment that maximizes agent preferences, without over- or under-assignment for each object…
Each combinatorial assignment problem (CAP) is defined by the *economy*

\[(\mathcal{C}, \mathcal{S}, (q_i)_{i=1}^M, (\succ_i)_{i=1}^N)\]

The **solution** to a CAP is an **assignment** \(x = (x_{ij})\), with

\[
x_{ij} = \begin{cases} 
1 & \text{if agent } i \text{ receives object } j \\
0 & \text{otherwise}
\end{cases}
\]

We want to find an assignment that maximizes agent preferences, without over- or under-assignment for each object…

**Without monetary transfers!**
Example: **University Course Allocation**

- Students (agents) have preferences ($\succ_i$) over different schedules (bundles).
- Each class has a maximum capacity ($q_j$),
- The university wants to fill all courses to capacity (minimize clearing error)

We want to assign students to their most-preferred schedules, with minimal clearing error, without students exchanging money.

This is a combinatorial assignment problem!
Combinatorial Assignment Problem

How do we solve CAPs? With a **Mechanism:**

- Serial dictatorship (UMD Testudo)
- Draft (like sports!)
How do we solve CAPs? With a **Mechanism:**

- Serial dictatorship (UMD Testudo)
- Draft (like sports!)

These mechanisms aren’t great. In particular, we care about:

1) **Pareto Efficient**  
   (*maximize agent preferences*)

2) **Fairness**  
   (*no envy*)

3) **Strategy-proof**  
   (*no gaming the system*)
Combinatorial Assignment Problem

How do we solve CAPs? With a **Mechanism:**

- Serial dictatorship (UMD Testudo)
- Draft (like sports!)

These mechanisms aren’t great. In particular, we care about:

1) **Pareto Efficient**  
   \((\text{maximize agent preferences})\)

2) **Fairness**  
   \((\text{no envy})\)

3) **Strategy-proof**  
   \((\text{no gaming the system})\)

**Proposition (Budish, 2011)**

There is no mechanism for CAP that is strategy-proof, Pareto efficient, and fair.*

\((C, S, (q_i)_{i=1}^M, (> i)_{i=1}^N)\)

\(i \in S\)  
\(j \in C\)  

\(q_j\)
Combinatorial Assignment Problem

One solution: **Fake Money**

- Each agent has a budget  $b_i \in \mathbb{R}^+$
- Each object has a price  $p_j \in \mathbb{R}^+$
- Each agent “demands” their most-preferred affordable bundle
- We need to find prices for each object to minimize clearing error

\[(C, S, (q_i)_{i=1}^M, (\succ_i)_{i=1}^N)\]
Combinatorial Assignment Problem

One solution: **Fake Money**

- Each agent has a budget \( b_i \in \mathbb{R}^+ \)
- Each object has a price \( p_j \in \mathbb{R}^+ \)
- Each agent “demands” their most-preferred affordable bundle
- We need to find prices for each object to minimize clearing error

Now to define a mechanism...
Combinatorial Assignment Problem: A-CEEI

**Mechanism:** Approximate Competitive Equilibrium from Equal Incomes (A-CEEI) (Budish, 2011)

- Adapts CEEI concept from fair allocation of divisible goods

- Basically a search through “price space”
Combinatorial Assignment Problem: A-CEEI

**Mechanism:** Approximate Competitive Equilibrium from Equal Incomes (A-CEEI) (Budish, 2011)

- Adapts CEEI concept from fair allocation of divisible goods
- Basically a search through “price space”

1) Agents report their preferences $\succ_i$

2) Agents are randomly assigned budgets $b_i \sim 1 + U(0, \beta)$, $0 < \beta \ll \min\left(\frac{1}{N'}, \frac{1}{k-1}\right)$

3) Find prices for each object $\mathbf{p} = (p_j)$, such that

   a) Agents receive their most-preferred bundle they can afford:

   $\mathbf{x}_{i,:} = \arg\max_{\succ_i} \{ \psi \in \{0, 1\}^M : \psi \cdot \mathbf{p} \leq b_i \}$

   b) Clearing error $\alpha$ is minimized:

   $$\alpha = \sqrt{\sum_{j=1}^{M} \xi_j^2}$$

   $$\xi_j = \begin{cases} 
   \sum_i^N x_{ij} - q_j & \text{if } p_j > 0 \\
   \max\{\sum_i^N x_{ij} - q_j, 0\} & \text{if } p_j = 0 
   \end{cases}$$
Combinatorial Assignment Problem: A-CEEI

**Mechanism:** Approximate Competitive Equilibrium from Equal Incomes (A-CEEI) (Budish, 2011)

- Adapts CEEI concept from fair allocation of divisible goods
- Basically a search through “price space”

Theoretical guarantees:

- **Existence:** \( \alpha \leq \sqrt{M \cdot \min(2k, M)}/2 \)
- **Pareto Efficient** (approximately)
- **Fairness:** envy bounded by a single good
- **Strategy-proof** in-the-large

1) Agents report their preferences \( \succ_i \)
2) Agents are randomly assigned **budgets**
\[ b_i \sim 1 + U(0, \beta), \quad 0 < \beta \leq \min\left(\frac{1}{N}, \frac{1}{k-1}\right) \]
3) Find **prices** for each object \( p = (p_j) \), such that
   a) Agents receive their most-preferred bundle they can afford:
   \[ x_{i, \parallel} = \arg \max_{\forall \psi \in \{0, 1\}^M : \psi \cdot p \leq b_i} \psi \]
   b) Clearing error \( \alpha \) is minimized:
   \[
   \alpha = \sqrt{\sum_{j=1}^{M} \xi_j^2} \\
   \xi_j = \begin{cases} 
   \sum_{i}^N x_{ij} - q_j & \text{if } p_j > 0 \\
   \max\{\sum_{i}^N x_{ij} - q_j, 0\} & \text{if } p_j = 0 
   \end{cases}
   \]
Combinatorial Assignment Problem: A-CEEI

**Mechanism:** Approximate Competitive Equilibrium from Equal Incomes (A-CEEI) (Budish, 2011)

**Problem:** It is difficult to find the optimal price vector.

**Solution:** Two-step search through price space (Othman et al. 2010).

1) Agents report their preferences $\succ_i$

2) Agents are randomly assigned budgets

$$b_i \sim 1 + U(0, \beta), \quad 0 < \beta \ll \min\left(\frac{1}{N'}, \frac{1}{k-1}\right)$$

3) Find prices for each object $p = (p_j)$, such that

   a) Agents receive their most-preferred bundle they can afford:

   $$x_{i,:} = \arg \max \{\psi \in \{0, 1\}^M : \psi \cdot p \leq b_i\}$$

   b) Clearing error $\alpha$ is minimized:

   $$\alpha = \sqrt{\sum_{j=1}^{M} \xi_j^2}$$

   $$\xi_j = \begin{cases} \sum_{i}^{N} x_{ij} - q_j & \text{if } p_j > 0 \\ \max\{\sum_{i}^{N} x_{ij} - q_j, 0\} & \text{if } p_j = 0 \end{cases}$$
Tabu price search (Othman et al. 2010)

Each candidate price vector is a search node.

- Start with a random node
- Visit new nodes by generating neighbors of the current node
- Don’t revisit any of the \( t \) most-recently-visited nodes (Tabu)
- Keep track of the best node
- Stop when \( \alpha \) is small enough
Tabu price search (Othman et al. 2010)

Each candidate price vector is a search node.
- Start with a random node
- Visit new nodes by generating neighbors of the current node
- Don’t revisit any of the $t$ most-recently-visited nodes (Tabu)
- Keep track of the best node
- Stop when $\alpha$ is small enough

At each node

1) Assign each agent her most-preferred affordable schedule

$$x_{i,:} = \arg\max_{\psi \in \{0, 1\}^M : \psi \cdot p \leq b_i}$$

2) Calculate clearing error

$$\alpha = \sqrt{\sum_{j=1}^{M} \xi_j^2}$$

$$\xi_j = \begin{cases} \sum_i x_{ij} - q_j & \text{if } p_j > 0 \\ \max\{\sum_i x_{ij} - q_j, 0\} & \text{if } p_j = 0 \end{cases}$$
Solving A-CEEI

At each node

1) Assign each agent her most-preferred affordable schedule

\[ \mathbf{x}_{i,:} = \arg \max_{\psi \in \{0,1\}^M : \psi \cdot \mathbf{p} \leq b_i} \]

2) Calculate clearing error

\[ \alpha = \sqrt{\sum_{j=1}^{M} \xi_j^2} \]

\[ \xi_j = \begin{cases} \sum_{i} x_{ij} - q_j & \text{if } p_j > 0 \\ \max\{\sum_{i} x_{ij} - q_j, 0\} & \text{if } p_j = 0 \end{cases} \]
Solving A-CEEI

1: \( node \leftarrow \text{random} \; p, p_j \sim U(0, \text{max}(b)) \)
2: \( best\_node \leftarrow node \)
3: \( tabu \leftarrow \text{fixed-length queue, length} \; t \)
4: \( \text{while} \; best\_node.\alpha < \text{threshold} \; \text{do} \)
5: \quad tabu.append( node )
6: \quad N \leftarrow \text{neighbors}( node )
7: \quad best\_nbr \leftarrow \text{arg min}_\alpha (N \setminus tabu)
8: \quad \text{if} \; best\_nbr.\alpha < best\_node.\alpha \; \text{do}
9: \quad \quad best\_node \leftarrow best\_nbr
10: \quad \text{end}
11: \text{end}

How do we generate neighbors?

Tabu price search

**Input:** threshold for \( \alpha \) (threshold)

**Output:** best search node (best_node)
Price space neighbor generation: Two ways

Gradient descent

The gradient of $\alpha$ is simply $\Delta = (\Delta_j)$, with

$$\Delta_j = \begin{cases} \sum_i x_{ij} - q_j & \text{if } p_j > 0 \\ \max\{\sum_i x_{ij} - q_j, 0\} & \text{if } p_j = 0 \end{cases}$$

For some step length $z$, the neighbor’s prices are

$$p_j \leftarrow p_j + z\Delta_j$$

$$\alpha = \sqrt{\sum_{j=1}^{M} \xi_j^2}$$

$$\xi_j = \begin{cases} \sum_i^N x_{ij} - q_j & \text{if } p_j > 0 \\ \max\{\sum_i^N x_{ij} - q_j, 0\} & \text{if } p_j = 0 \end{cases}$$
Solving A-CEEI

Price space neighbor generation: Two ways

**Gradient descent**

The gradient of $\alpha$ is simply $\Delta = (\Delta_j)$, with

$$
\Delta_j = \begin{cases} 
\sum_i x_{ij} - q_j & \text{if } p_j > 0 \\
\max\{\sum_i x_{ij} - q_j, 0\} & \text{if } p_j = 0 
\end{cases}
$$

For some step length $z$, the neighbor’s prices are

$$
p_j \leftarrow p_j + z\Delta_j
$$

**Single price adjustments**

For any object $j$:

- **If $j$ is under-demanded:** $\sum_i x_{ij} < q_j$
  - Set $p_j \leftarrow 0$

- **If $j$ is over-demanded:** $\sum_i x_{ij} > q_j$
  - Increase $p_j$ until $\sum_i x_{ij}$ decreases by one.
Price space neighbor generation: Two ways

Gradient descent

The gradient of $\alpha$ is simply $\Delta = (\Delta_j)$, with

$$\Delta_j = \begin{cases} \sum_i x_{ij} - q_j & \text{if } p_j > 0 \\ \max\{\sum_i x_{ij} - q_j, 0\} & \text{if } p_j = 0 \end{cases}$$

For some step length $z$, the neighbor’s prices are

$$p_j \leftarrow p_j + z\Delta_j$$

Single price adjustments

For any object $j$:

If $j$ is under-demanded: $\sum_i x_{ij} < q_j$
- Set $p_j \leftarrow 0$

If $j$ is over-demanded: $\sum_i x_{ij} > q_j$
- Increase $p_j$ until $\sum_i x_{ij}$ decreases by one.

Method:
Generate several neighbors of each type, and move to the best neighbor during Tabu search (lowest $\alpha$)
Most calculation time will be due to these steps, which require solving many mixed integer programs (MIPs)

At each node

1) Assign each agent her most-preferred affordable schedule

\[ x_{i,:} = \arg \max_{\psi \in \{0, 1\}^M} \{ \psi \cdot p \leq b_i \} \]

2) Calculate clearing error

\[ \alpha = \sqrt{\sum_{j=1}^{M} \xi_j^2} \]

\[ \xi_j = \begin{cases} 
\sum_{i} x_{ij} - q_j & \text{if } p_j > 0 \\
\max\{\sum_{i} x_{ij} - q_j, 0\} & \text{if } p_j = 0
\end{cases} \]

Single price adjustments (Neighbor generation)

For any object \( j \):

If \( j \) is under-demanded: \( \sum_{i} x_{ij} < q_j \)
- Set \( p_j \leftarrow 0 \)

If \( j \) is over-demanded: \( \sum_{i} x_{ij} > q_j \)
- Increase \( p_j \) until \( \sum_{i} x_{ij} \) decreases by one.
Most calculation time will be due to these steps, which require solving many mixed integer programs (MIPs).

At each node

1) Assign each agent her most-preferred affordable schedule

\[ x_{i,:} = \arg \max_{\psi \in \{0, 1\}^M} \{ \psi \cdot p \leq b_i \} \]

2) Calculate clearing error

\[ \alpha = \sqrt{\sum_{j=1}^{M} \xi_j^2} \]

\[ \xi_j = \begin{cases} \sum_{i}^{N} x_{ij} - q_j & \text{if } p_j > 0 \\ \max\{\sum_{i}^{N} x_{ij} - q_j, 0\} & \text{if } p_j = 0 \end{cases} \]

Single price adjustments (Neighbor generation)

For any object \( j \):

If \( j \) is under-demanded: \( \sum_{i} x_{ij} < q_j \)
- Set \( p_j \leftarrow 0 \)

If \( j \) is over-demanded: \( \sum_{i} x_{ij} > q_j \)
- Increase \( p_j \) until \( \sum_{i} x_{ij} \) decreases by one.
Solving A-CEEI: Implementation

Most calculation time will be due to these steps, which require solving many mixed integer programs (MIPs)

- **Each agent’s MIPs are independent** (my preferences don’t depend on yours)
- Run them in parallel

**At each node**

1) Assign each agent her most-preferred affordable schedule

\[ x_{i,:} = \arg \max_{\psi \in \{0, 1\}^M} \{ \psi \cdot p \leq b_i \} \]

2) Calculate clearing error

\[ \alpha = \sqrt{\sum_{j=1}^{M} \xi_j^2} \]

\[ \xi_j = \begin{cases} \sum_{i} x_{ij} - q_j & \text{if } p_j > 0 \\ \max\{\sum_{i} x_{ij} - q_j, 0\} & \text{if } p_j = 0 \end{cases} \]

**Single price adjustments (Neighbor generation)**

For any object \( j \):

**If \( j \) is under-demanded:** \( \sum_{i} x_{ij} < q_j \)
- Set \( p_j \leftarrow 0 \)

**If \( j \) is over-demanded:** \( \sum_{i} x_{ij} > q_j \)
- Increase \( p_j \) until \( \sum_{i} x_{ij} \) decreases by one.
Generate Random CAPs, resembling university course scheduling (Othman et al. 2010)

- \( N = 250 \) (students)
- \( M = 50 \) (courses)
- \( k = 5 \) (student schedule limit)
- \( q_j = 27 \) (course limit)

- Preferences

\[ x_1 > x_2 > x_3 > \ldots > x_{2^M} \]
Generate Random CAPs, resembling university course scheduling
(Othman et al. 2010)

- \( N = 250 \) (students)
- \( M = 50 \) (courses)
- \( k = 5 \) (student schedule limit)
- \( q_j = 27 \) (course limit)

- Preferences

\[ x_1 > x_2 > x_3 > \ldots > x_{2^M} \]

\[ u(x_1) > u(x_2) > u(x_3) > \ldots > u(x_{2^M}) \]
Generate Random CAPs, resembling university course scheduling
(Othman et al. 2010)

- \( N = 250 \) (students)
- \( M = 50 \) (courses)
- \( k = 5 \) (student schedule limit)
- \( q_j = 27 \) (course limit)

- Preferences

\[ x_1 > x_2 > x_3 > \ldots > x_{2^M} \]

\[ u(x_1) > u(x_2) > u(x_3) > \ldots > u(x_{2^M}) \]
Generate Random CAPs, resembling university course scheduling (Othman et al. 2010)

- $N = 250$ (students)
- $M = 50$ (courses)
- $k = 5$ (student schedule limit)
- $q_j = 27$ (course limit)

- Preferences

\[
x_1 > x_2 > x_3 > \ldots > x_{2^M}
\]

\[
u(x_1) > u(x_2) > u(x_3) > \ldots > u(x_{2^M})
\]

\[
u(\text{🍎}) = 10
\]
\[
u(\text{🍌}) = 10
\]
\[
u(\text{🥕}) = 5
\]
\[
u(\text{🍎} + \text{🍌}) = 5
\]
\[
u(\text{🍎} + \text{🥕}) = -20
\]

$\succ B = \succ \succ \succ \succ$
Generate Random CAPs, resembling university course scheduling (Othman et al. 2010)

- \( N = 250 \) (students)
- \( M = 50 \) (courses)
- \( k = 5 \) (student schedule limit)
- \( q_j = 27 \) (course limit)

- Preferences

\[ x_1 > x_2 > x_3 > \ldots > x_{2^M} \]
\[ u(x_1) > u(x_2) > u(x_3) > \ldots > u(x_{2^M}) \]

\[ u(\text{🍎}) = 10 \]
\[ u(\text{.offsetTop}) = 10 \]
\[ u(\text{🥕}) = 5 \]
\[ u(\text{🍎 + トップ}) = 5 \]
\[ u(\text{🍎 + トップ}) = -20 \]

\[ \succ B = \succ \text{🍎} \succ \text{香蕉} \succ \text{香蕉} \succ \text{胡萝卜} \succ \text{胡萝卜} \]
Generate Random CAPs, resembling university course scheduling (Othman et al. 2010)

- $N = 250$ (students)
- $M = 50$ (courses)
- $k = 5$ (student schedule limit)
- $q_j = 27$ (course limit)

- Preferences

$$u(x_1) > u(x_2) > u(x_3) > \ldots > u(x_{2^M})$$
Generate Random CAPs, resembling university course scheduling (Othman et al. 2010)

- \( N = 250 \) (students)
- \( M = 50 \) (courses)
- \( k = 5 \) (student schedule limit)
- \( q_j = 27 \) (course limit)

- Preferences

\[ u(x_1) > u(x_2) > u(x_3) > \ldots > u(x_{2^M}) \]

\[ u(x_i) > u(x_{i+1}) > u(x_{i+2}) > \ldots > u(x_{2^M}) \]

\[ u(\text{apple}) = 10 \]
\[ u(\text{banana}) = 10 \]
\[ u(\text{carrot}) = 5 \]
\[ u(\text{apple} + \text{banana}) = 5 \]
\[ u(\text{apple} + \text{carrot}) = -20 \]

\[ +10 \]
\[ +10 \]
\[ +5 \]
\[ +25 \]
\[ +10 \]
\[ +5 \]

\[ \succ B = \succ \succ \succ \succ \]
Generate Random CAPs, resembling university course scheduling (Othman et al. 2010)

- $N = 250$ (students)
- $M = 50$ (courses)
- $k = 5$ (student schedule limit)
- $q_j = 27$ (course limit)

- Preferences

$x_1 > x_2 > x_3 > \ldots > x_{2^M}$

$u(x_1) > u(x_2) > u(x_3) > \ldots > u(x_{2^M})$

$u(\text{apple}) = 10$
$u(\text{banana}) = 10$
$u(\text{carrot}) = 5$

$u(\text{apple} + \text{banana}) = 5$
$u(\text{apple} + \text{carrot}) = -20$

$\succ B = \succ \text{apple} \succ \text{banana} \succ \text{carrot} \succ \text{banana}$
Generate Random CAPs, resembling university course scheduling  
(Othman et al. 2010)

- $N = 250$ (students)
- $M = 50$ (courses)
- $k = 5$ (student schedule limit)
- $q_j = 27$ (course limit)

Preferences

$x_1 > x_2 > x_3 > \ldots > x_{2^M}$

$u(x_1) > u(x_2) > u(x_3) > \ldots > u(x_{2^M})$
Generate Random CAPs, resembling university course scheduling (Othman et al. 2010)

- $N = 250$ (students)
- $M = 50$ (courses)
- $k = 5$ (student schedule limit)
- $q_j = 27$ (course limit)

Preferences

$x_1 > x_2 > x_3 > \ldots > x_{2^M}$

$u(x_1) > u(x_2) > u(x_3) > \ldots > u(x_{2^M})$

**Preference Language** assigns utility to each bundle

- $u_{ij}$: additive utility for each object $j$ ($\mathbb{R}$)
- $\eta_s$: for a set $s$ of objects, if the agent receives all objects in $s$, they receive utility $\eta_s$ ($\mathbb{R}$)

Hard constraints $c$:

“an agent may receive no more than $n$ objects in set $s$.”
Generate Random CAPs, resembling university course scheduling (Othman et al. 2010)

- $N = 250$ (students)
- $M = 50$ (courses)
- $k = 5$ (student schedule limit)
- $q_j = 27$ (course limit)

Preferences

\[ x_1 > x_2 > x_3 > \ldots > x_{2^M} \]

\[ u(x_1) > u(x_2) > u(x_3) > \ldots > u(x_{2^M}) \]

Preference Language assigns utility to each bundle

- $u_{ij}$: additive utility for each object $j$ ($\mathbb{R}$)
- $\eta_{is}$: for a set $s$ of objects, if the agent receives all objects in $s$, they receive utility $\eta_s$ ($\mathbb{R}$)

Hard constraints $c$:

“an agent may receive no more than $n$ objects in set $s$”.

Random Preferences:

\[ u_{ij} \sim N(\mu = j, \sigma = 10) \]

10 pairwise complement/substitute effects ($j,j'$):

\[ \eta_{i,j,j'} \sim U(-10,10) \]

Hard constraints only limit bundle size ($k$)
Validation

Metrics for success:

- **Final clearing error**
  - Average error at each iteration
  - Number of iterations to reach bound

- **Runtime per iteration**
  - Scale linearly in $N$
  - Scale nearly-linearly in $M$

- **Bounded Envy**
  - What is the maximum envy?

- **Pareto Efficiency**
  - Are trades possible? *(hard)*
Validation

Metrics for success:

- **Final clearing error**
  - Average error at each iteration
  - Number of iterations to reach bound

- **Runtime per iteration**
  - Scale linearly in $N$
  - Scale nearly-linearly in $M$

- **Bounded Envy**
  - What is the maximum envy?

- **Pareto Efficiency**
  - Are trades possible? (hard)
Validation

Metrics for success:

- Final clearing error
  - Average error at each iteration
  - Number of iterations to reach bound

- Runtime per iteration
  - Scale linearly in $N$
  - Scale nearly-linearly in $M$

- Bounded Envy
  - What is the maximum envy?

- Pareto Efficiency
  - Are trades possible? \textbf{(hard)}
Possible Extension: Coverage Constraints

\[ (C, S, (q_i)_{i=1}^M, (\succ i)_{i=1}^N) + \text{ coverage constraints } r_j \]

**Coverage constraints:**

at least \( r_j \) copies of object \( j \) must be assigned.

\[ \forall i \in S \quad j \in C \]

\[ \succ i \quad q_j \]
Possible Extension: Coverage Constraints

\[(C, S, (q_i)_{i=1}^M, (\gamma_i)_{i=1}^N) + \text{ coverage constraints } r_j\]

**Coverage constraints:**
- at least \(r_j\) copies of object \(j\) must be assigned.
(C, S, (q_i)_{i=1}^M, (∀)_{i=1}^N) + coverage constraints r_j

Coverage constraints:
at least r_j copies of object j must be assigned.

This is a version of Nurse Scheduling:
- Hospitals assign nurses to shifts
- Nurses have restrictions & preferences over schedules
- Hospitals have coverage requirements
Project Timeline

1) Implement A-CEEI
   - Feb. - Mar. 2018 : Parallelize MIP calculation (many options in Python: multiprocessing, threading, etc.)

   Deliverable (March 2018): Mid-term report with validation results.

2) Extend A-CEEI to Nurse Scheduling (time allowing)
   - Mar. - May 2018 : Adapt A-CEEI code for solving basic nurse scheduling

   Deliverable (May 2018): Final report with validation results and possible extension.
Thank you!

Duncan McElfresh
dmcelfre@math.umd.edu
AMSC PhD Student
Department of Mathematics
University of Maryland College Park