DECOMPOSITION FOR DUMMIES

Applications of Optimization — Best Practice and Challenges
Copenhagen, Denmark

Matthew Galati

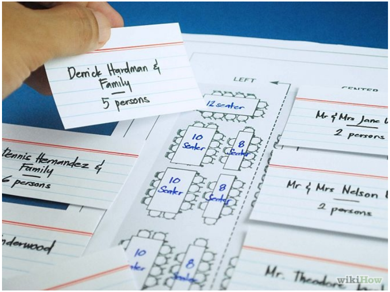
Principal Operations Research Specialist

OUTLINE

1. Wedding Seat Assignments
2. Decomposition
3. Software
4. Conclusion

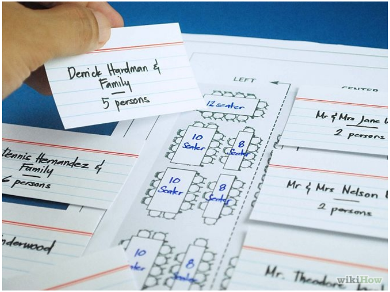
WEDDING SEAT ASSIGNMENTS

DO YOU HAVE AN UNCLE LOUIE (STEWART)?



WEDDING SEAT ASSIGNMENTS

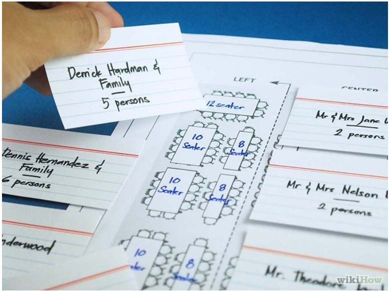
DO YOU HAVE AN UNCLE LOUIE (STEWART)?



where do we put this guy?

WEDDING SEAT ASSIGNMENTS

DO YOU HAVE AN UNCLE LOUIE (STEWART)?



where do we put this guy?

DO YOU HAVE AN UNCLE LOUIE (STEWART)?

- a_{gh} unhappiness for guest g if seated with guest h
- x_{gt} defines a binary variable to seat guest g at table t
- u_t defines a continuous variable for the maximum unhappiness at table t

Wedding Seat Assignments

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{t \in T} u_t & \\
\text{subject to} & \displaystyle\sum_{t \in T} x_{gt} = 1 & g \in G \\
& \displaystyle\sum_{g \in G} x_{gt} \le S & t \in T \\
& u_t \ge a_{gh}(x_{gt} + x_{ht} - 1) & t \in T, g \in G, h \in G \text{ such that } g < h
\end{array}
$$

PROC OPTMODEL

```
proc optmodel;
   /* declare parameters, index sets, data */
   ...

   set GUESTS = 1..num_guests;
   set TABLES = 1..max_tables;
   set GUEST_PAIRS = {g in 1..num_guests-1, h in g+1..num_guests};
   num unhappyPair{<g,h> in GUEST_PAIRS} = abs(g-h);

   /* declare decision variables */
   var x{GUESTS,TABLES} binary;
   var unhappy{TABLES} >= 0;

   /* declare objective */
   min MinUnhappy = sum{t in TABLES} unhappy[t];

   /* declare constraints */
   con Assign{g in GUESTS}:
      sum{t in TABLES} x[g,t] = 1;
   con TableSize{t in TABLES}:
      sum{g in GUESTS} x[g,t] <= max_table_size;
   con TableMeasure{t in TABLES, <g,h> in GUEST_PAIRS}:
      unhappy[t] >= unhappyPair[g,h] * (x[g,t] + x[h,t] - 1);

   /* call MILP branch & cut solver */
   solve with milp / maxtime=3600;
```

WEDDING SEAT ASSIGNMENTS

MILP BRANCH & CUT

$$|G| = 36, S = 6$$

```
NOTE: The presolved problem has 222 variables, 3822 constraints, and 11772 constraint coefficients.
NOTE: The MILP solver is called.
          Node  Active    Sols    BestInteger      BestBound      Gap     Time
             0       1       1     30.0000000              0   30.000        0
             0       1       1     30.0000000              0   30.000        0
NOTE: The MILP solver's symmetry detection found 19 orbits. The largest orbit contains 12 variables.
             0       1       1     30.0000000              0   30.000        0
             0       1       1     30.0000000              0   30.000        0
NOTE: The MILP solver added 31 cuts with 492 cut coefficients at the root.
           100      78       1     30.0000000      6.8217196  339.77%        2
           200     166       1     30.0000000      7.5101042  299.46%        2
-- snip --
       1746100   57997       1     30.0000000     28.5185397    5.19%     3599
       1746200   57957       1     30.0000000     28.5188178    5.19%     3599
       1746202   57958       1     30.0000000     28.5188178    5.19%     3599
NOTE: CPU time limit reached.
NOTE: Objective of the best integer solution found = 30.
```

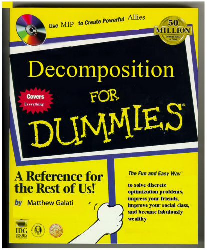
MILP BRANCH & PRICE — DECOMP

Replace this

```
/* call MILP branch & cut solver */
solve with milp / maxtime=3600;
```

with this

```
/* call MILP branch & price solver using method=set */
solve with milp / maxtime=3600 decomp=(method=set);
```



MILP BRANCH & PRICE — DECOMP

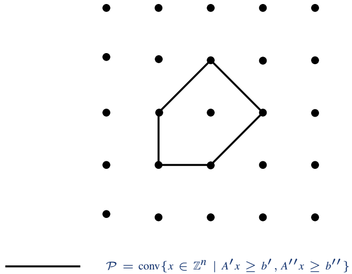
$$|G| = 36, S = 6$$

```
NOTE: The presolved problem has 222 variables, 3822 constraints, and 11772 constraint coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The DECOMP method value SET is applied.
NOTE: The Decomposition algorithm is using an aggregate formulation and Ryan-Foster branching.
NOTE: The problem has a decomposable structure with 6 blocks. The largest block covers 16.51%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 222 (100.00%) variables and 3786 (99.06%) constraints.
      Iter         Best       Master         Best       LP       IP    CPU   Real
                  Bound    Objective      Integer      Gap      Gap   Time   Time
NOTE: Starting phase 1.
         1       0.0000       0.0000            .    0.00%        .     0     0
NOTE: Starting phase 2.
         2       0.0000      30.0000      30.0000 3.00e+01 3.00e+01     0     0
         .       0.0000      30.0000      30.0000 3.00e+01 3.00e+01     0     0
        10       0.0000      30.0000      30.0000 3.00e+01 3.00e+01     0     0
-- snip --
         .      25.2000      30.0000      30.0000   19.05%   19.05%    13     8
       110      25.2000      30.0000      30.0000   19.05%   19.05%    13     8
       112      30.0000      30.0000      30.0000    0.00%    0.00%    13     8
         Node  Active   Sols         Best         Best      Gap    CPU    Real
                                  Integer        Bound             Time   Time
            0       0      1      30.0000      30.0000    0.00%     13      8
NOTE: The Decomposition algorithm time is 8.89 seconds.
NOTE: Optimal.
NOTE: Objective = 30.
```

OUTLINE

By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

$$z_{\mathrm{IP}} = \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$



Assumptions:

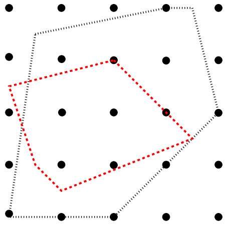
- $\mathrm{OPT}(\mathcal{P}', c)$ and $\mathrm{SEP}(\mathcal{P}', x)$ are *"easy"*
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size)
- $\mathcal{P}'$ must be represented implicitly (description has exponential size)

THE DECOMPOSITION PRINCIPLE

By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

$$z_{\mathrm{IP}} = \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$



Assumptions:

- $\mathrm{OPT}(\mathcal{P}', c)$ and $\mathrm{SEP}(\mathcal{P}', x)$ are *"easy"*
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size)
- $\mathcal{P}'$ must be represented implicitly (description has exponential size)

$\mathcal{Q}' = \{x \in \mathbb{R}^n \mid A'x \geq b'\}$

$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

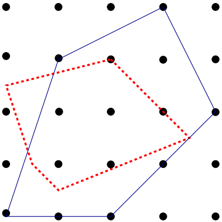
By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

$$z_{\mathrm{IP}} = \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{D}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid x \in \mathcal{P}', A''x \geq b'' \right\}$$

$$z_{\mathrm{IP}} \geq z_{\mathrm{D}} \geq z_{\mathrm{LP}}$$



$\mathcal{P}' = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b'\}$

$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

Assumptions:

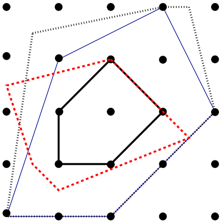
- $\mathrm{OPT}(\mathcal{P}', c)$ and $\mathrm{SEP}(\mathcal{P}', x)$ are *"easy"*
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size)
- $\mathcal{P}'$ must be represented implicitly (description has exponential size)

THE DECOMPOSITION PRINCIPLE

By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

$$
\begin{aligned}
z_{\mathrm{IP}} &= \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\} \\
z_{\mathrm{LP}} &= \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\} \\
z_{\mathrm{D}} &= \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid x \in \mathcal{P}', A''x \geq b'' \right\}
\end{aligned}
$$

$$z_{\mathrm{IP}} \geq z_{\mathrm{D}} \geq z_{\mathrm{LP}}$$



$\mathcal{P} = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b''\}$

$\mathcal{P}' = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b'\}$

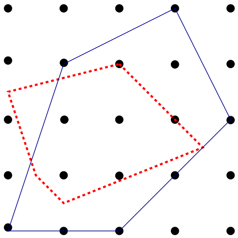
$\mathcal{Q}' = \{x \in \mathbb{R}^n \mid A'x \geq b'\}$

$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

Assumptions:

- $\mathrm{OPT}(\mathcal{P}', c)$ and $\mathrm{SEP}(\mathcal{P}', x)$ are *"easy"*
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size)
- $\mathcal{P}'$ must be represented implicitly (description has exponential size)

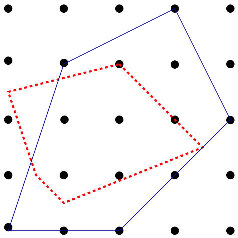
- CP builds an *outer* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$
  - **Subproblem**: $\mathrm{SEP}(\mathcal{P}', x)$ — *exponential number of constraints*



$$z_{\mathrm{CP}} = z_{\mathrm{D}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{\mathrm{LP}}$$

CUTTING PLANE VS DANTZIG-WOLFE METHOD

- CP builds an *outer* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$
  - **Subproblem**: $\mathrm{SEP}(\mathcal{P}', x)$ — *exponential number of constraints*
- DW builds an *inner* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$
  - Reformulation: $\mathcal{P}' = \left\{x \in \mathbb{R}^n \;\middle|\; x = \sum_{s \in \mathcal{E}} s\lambda_s, \sum_{s \in \mathcal{E}} \lambda_s = 1, \lambda_s \geq 0 \; \forall s \in \mathcal{E}\right\}$
  - **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u^\top A''\right)$ — *exponential number of variables*



$$z_{\mathrm{DW}} = z_{\mathrm{CP}} = z_{\mathrm{D}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{\mathrm{LP}}$$

THEORY VERSUS PRACTICE

- Generic cutting planes often cannot obtain even $\mathcal{P}'$

$$z_{\mathrm{DW}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{\mathrm{CP}} \geq z_{\mathrm{LP}}$$

- DW often improves the bound obtained by CP, but:
  - The work to obtain the bound is much more extensive
  - Convergence of dual space is slow (stabilization techniques can help)

Cutting Plane Method

Node	Active	Sols	BestInteger	BestBound	Gap	Time
0	1	1	30.0000000	0	30.000	0
0	1	1	30.0000000	0	30.000	0
0	1	1	30.0000000	0	30.000	0

Dantzig-Wolfe Decomposition

Iter	Best Bound	Master Objective	Best Integer	LP Gap	IP Gap	CPU Time	Real Time
1	0.0000	0.0000	.	0.00%	.	0	0
-- snip --							
110	25.2000	30.0000	30.0000	19.05%	19.05%	13	8
112	30.0000	30.0000	30.0000	0.00%	0.00%	13	8

- One motivation for decomposition is to expose *independent subsystems*.
- The key is to identify *block structure* in the constraint matrix.
- The separability lends itself nicely to parallel implementation.

Generalized Assignment Problem (GAP)

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{i\in M}\sum_{j\in N} c_{ij}x_{ij} & \\ \text{subject to} & \displaystyle\sum_{i\in M} x_{ij} = 1 & j \in N \\ & \displaystyle\sum_{j\in N} w_{ij}x_{ij} \le b_i & i \in M \end{array}$$

RELAXATION SEPARABILITY

- One motivation for decomposition is to expose *independent subsystems*.
- The key is to identify *block structure* in the constraint matrix.
- The separability lends itself nicely to parallel implementation.

$$\begin{pmatrix} A''_1 & A''_2 & \cdots & A''_\kappa \\ \hline A'_1 & & & \\ & A'_2 & & \\ & & \ddots & \\ & & & A'_\kappa \end{pmatrix}$$

$$\mathcal{P}' = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{k \in K} \sum_{s \in \mathcal{E}^k} s\lambda_s^k, \sum_{s \in \mathcal{E}^k} \lambda_s^k = 1 \; \forall k \in K, \lambda_s^k \geq 0 \; \forall k \in K, \; s \in \mathcal{E}^k \right\}$$

- One motivation for decomposition is to expose *independent subsystems*.
- The key is to identify *block structure* in the constraint matrix.
- The separability lends itself nicely to parallel implementation.

Problem	**Linking Constraints**	**Subproblem Blocks**
Multicommodity flow	Arc capacities	Flow balance (each commodity)
Bin packing	Assign each item to a bin	Capacity (each bin)
Cutting stock	Size requirements	Capacity (each roll)
Vehicle routing	Demand satisfaction	Routing and capacity (each vehicle)
Graph coloring	Color each node	Independent set (each color)
Marketing optimization	Campaign budgets	Contact policy (each customer)
Workforce planning	Demand satisfaction	Scheduling (each worker)
Side-constrained network	Side constraints	Network (each component)

SYMMETRY BREAKING — AGGREGATION

- In some cases, the identified blocks are *identical*.
- In such cases, the original formulation will often be highly symmetric.
- DW eliminates the symmetry by aggregating the identical blocks.

Wedding Seat Assignments

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{t\in T} u_t & \\
\text{subject to} & \displaystyle\sum_{t\in T} x_{gt} = 1 & g \in G \\
& \displaystyle\sum_{g\in G} x_{gt} \le S & t \in T \\
& u_t \ge a_{gh}(x_{gt} + x_{ht} - 1) & t \in T, g \in G, h \in G \text{ such that } g < h
\end{array}
$$

SYMMETRY BREAKING — AGGREGATION

- In some cases, the identified blocks are *identical*.
- In such cases, the original formulation will often be highly symmetric.
- DW eliminates the symmetry by aggregating the identical blocks.

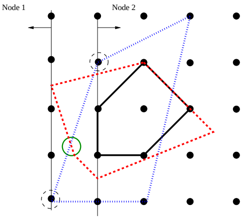
$$\begin{pmatrix} A''_1 & A''_2 & \cdots & A''_\kappa \\ \hline A'_1 & & & \\ & A'_2 & & \\ & & \ddots & \\ & & & A'_\kappa \end{pmatrix}$$

$$A''_1 = A''_2 = \cdots = A''_\kappa$$
$$A'_1 = A'_2 = \cdots = A'_\kappa$$

$$\mathcal{P}' = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{s \in \mathcal{E}} s\Lambda_s, \sum_{s \in \mathcal{E}} \Lambda_s = \kappa, \Lambda_s \geq 0 \; s \in \mathcal{E} \right\}$$

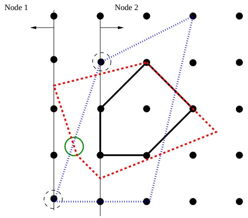
GENERIC BRANCHING

- This is done by mapping back to the compact space $\hat{x} = \sum_{s \in \mathcal{E}} s \hat{\lambda}_s$.
  - Variable branching (x-space) is constraint branching (λ-space)
- Aggregation step breaks our dependence on a 1-to-1 mapping.
- For set partitioning (covering) master, employ *Ryan-Foster* branching.
  - Branching constraints in compact space based on covering two rows
  - SAME: $x_g = x_h$ (wedding: two guests are seated at the same table)
  - DIFF: $x_g + x_h \leq 1$ (wedding: two guests are seated at different tables)
- For general master, Vanderbeck, et al. has a more complex approach.
  - *work in progress (2014)*



GENERIC BRANCHING

- This is done by mapping back to the compact space $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$.
  - Variable branching (x-space) is constraint branching (λ-space)
- Aggregation step breaks our dependence on a 1-to-1 mapping.
- For set partitioning (covering) master, employ *Ryan-Foster* branching.
  - Branching constraints in compact space based on covering two rows
  - SAME: $x_g = x_h$ (wedding: two guests are seated at the same table)
  - DIFF: $x_g + x_h \leq 1$ (wedding: two guests are seated at different tables)
- For general master, Vanderbeck, et al. has a more complex approach.
  - *work in progress (2014)*



DECOMPOSITION | CUSTOMER EXAMPLE — PHARMA

- Major U.S. pharmaceutical company
- Assign sales representatives (reps) to market drugs to doctors
- Decision: Assign reps to doctors and a pre-specified *sales direction* — trio of drugs and the order of presentation
- Objective: Maximize profit — a concave (utility) function on the number of product/order presentations (to a doctor)
- **Subproblem blocks** (per doctor):
  - visit each doctor at least once
  - for each rep/doctor combination, present a sales direction at most once
  - capacity on the total number of office visits to each doctor
  - constraints to model piecewise linear approx of utility function
- **Linking Constraints**:
  - capacity on the total number of office visits assigned to each rep

CUSTOMER EXAMPLE — PHARMA

- Major U.S. pharmaceutical company
- Assign sales representatives (reps) to market drugs to doctors
- Decision: Assign reps to doctors and a pre-specified *sales direction* — trio of drugs and the order of presentation
- Objective: Maximize profit — a concave (utility) function on the number of product/order presentations (to a doctor)
- **Subproblem blocks** (per doctor):
  - visit each doctor at least once
  - for each rep/doctor combination, present a sales direction at most once
  - capacity on the total number of office visits to each doctor
  - constraints to model piecewise linear approx of utility function
- **Linking Constraints**:
  - capacity on the total number of office visits assigned to each rep

DECOMPOSITION CUSTOMER EXAMPLE — PHARMA

MILP Branch & Cut

```
NOTE: The presolved problem has 52638 variables, 3582 constraints, and 142260 constraint coefficients.
          Node  Active     Sols    BestInteger       BestBound       Gap    Time
             0       1        0              .    7342.1209241         .       0
             0       1        0              .    7246.1067338         .      50
-- snip --
        115600  114326        1   6650.5045980    7241.3171431     8.16%    3590
        115699  114424        1   6650.5045980    7241.3171431     8.16%    3599
NOTE: CPU time limit reached.
```

MILP Branch & Price — DECOMP

```
NOTE: The problem has a decomposable structure with 610 blocks. The largest block covers 0.22%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 52638 (100.00%) variables and 3574 (99.78%) constraints.
      Iter         Best       Master         Best        LP        IP  CPU Real
                  Bound    Objective      Integer       Gap       Gap Time Time
NOTE: Starting phase 1.
         1       0.0000     302.6667            . 3.03e+02         .    7    3
         4       0.0000       0.0000            .    0.00%         .    8    4
NOTE: Starting phase 2.
         6    7963.9930    6393.1245    6387.1702   19.72%    19.80%   19    9
         7    7117.5411    6714.4494    6387.1702    5.66%    10.26%   31   13
         8    6979.2463    6939.4008    6387.1702    0.57%     8.48%   46   18
         9    6979.2463    6963.0448    6963.0448    0.23%     0.23%   61   24
-- snip --
        11    6972.3310    6972.3309    6972.3309    0.00%     0.00%   67   27
         Node   Active    Sols         Best         Best      Gap     CPU     Real
                                    Integer        Bound             Time     Time
            0        0       4    6972.3309    6972.3310    0.00%      67       27
NOTE: The Decomposition algorithm time is 27.89 seconds.
NOTE: Optimal within relative gap.
```

OUTLINE

1. Wedding Seat Assignments
2. Decomposition
3. **Software**
4. Conclusion

DECOMP — HISTORY

- Frameworks supporting branch-and-price, as far back as 1994:
  - MINTO, ABACUS, COIN/SYMPHONY, COIN/BCP
  - Issue — user must derive most algorithmic components for their application
- The idea to *generalize and automate* these components was first proposed (and developed) independently (2000-2004) by Vanderbeck (BaPCod) and Galati/Ralphs (COIN/DIP)
- Current status: several related projects available (free/open-source)
  - COIN/DIP, BaPCod, SCIP/GCG
- SAS DECOMP is the first/only commercial implementation

DECOMP — HISTORY

- Frameworks supporting branch-and-price, as far back as 1994:
  - MINTO, ABACUS, COIN/SYMPHONY, COIN/BCP
  - Issue — user must derive most algorithmic components for their application
- The idea to *generalize and automate* these components was first proposed (and developed) independently (2000-2004) by Vanderbeck (BaPCod) and Galati/Ralphs (COIN/DIP)
- Current status: several related projects available (free/open-source)
  - COIN/DIP, BaPCod, SCIP/GCG
- SAS DECOMP is the first/only commercial implementation

DECOMP — HISTORY

- Frameworks supporting branch-and-price, as far back as 1994:
  - MINTO, ABACUS, COIN/SYMPHONY, COIN/BCP
  - Issue — user must derive most algorithmic components for their application
- The idea to *generalize and automate* these components was first proposed (and developed) independently (2000-2004) by Vanderbeck (BaPCod) and Galati/Ralphs (COIN/DIP)
- Current status: several related projects available (free/open-source)
  - COIN/DIP, BaPCod, SCIP/GCG
- SAS DECOMP is the first/only commercial implementation

DECOMP — HISTORY

- Frameworks supporting branch-and-price, as far back as 1994:
  - MINTO, ABACUS, COIN/SYMPHONY, COIN/BCP
  - Issue — user must derive most algorithmic components for their application
- The idea to *generalize and automate* these components was first proposed (and developed) independently (2000-2004) by Vanderbeck (BaPCod) and Galati/Ralphs (COIN/DIP)
- Current status: several related projects available (free/open-source)
  - COIN/DIP, BaPCod, SCIP/GCG
- SAS DECOMP is the first/only commercial implementation

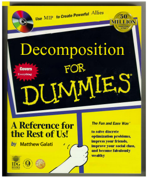
- SET: Searches for a set partitioning (covering) master
  - common modeling paradigm (VRP, Bin Packing, Coloring, etc.)

```
con Assign{g in GUESTS}:
   sum{t in TABLES} x[g,t] = 1;
con TableSize{t in TABLES}:
   sum{g in GUESTS} x[g,t] <= max_table_size;
con TableMeasure{t in TABLES, <g,h> in GUEST_PAIRS}:
   unhappy[t] >= unhappyPair[g,h] * (x[g,t] + x[h,t] - 1);

/* call MILP branch & price solver using method=set */
solve with milp / maxtime=3600 decomp=(method=set);
```

- SET: Searches for a set partitioning (covering) master
  - common modeling paradigm (VRP, Bin Packing, Coloring, etc.)
- USER: User defines the subproblems using constraint suffix .block
  - the user might know best a subproblem's strength/tractability

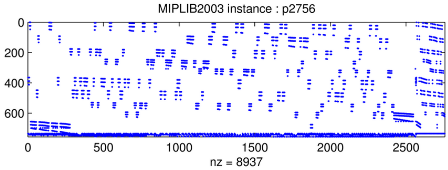
```
con Assign{g in GUESTS}:
   sum{t in TABLES} x[g,t] = 1;
con TableSize{t in TABLES}:
   sum{g in GUESTS} x[g,t] <= max_table_size;
con TableMeasure{t in TABLES, <g,h> in GUEST_PAIRS}:
   unhappy[t] >= unhappyPair[g,h] * (x[g,t] + x[h,t] - 1);

/* call MILP branch & price solver using method=set */
solve with milp / maxtime=3600 decomp=(method=set);
```

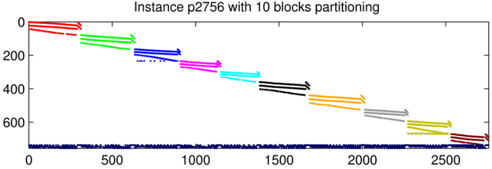
```
/* define the blocks for decomposition */
for{t in TABLES} do;
   TableSize[t].block = t;
   for{<g,h> in GUEST_PAIRS}
      TableMeasure[t,g,h].block = t;
end;

/* call MILP branch & price solver using method=user */
solve with milp / maxtime=3600 decomp;
```



SOFTWARE METHOD=AUTO

- AUTO: Automatically identifies *hidden block structure* [Aykanat99]
  - graph partitioning on symmetrized version of A into doubly-bordered BDF
  - column-splitting (Lagrangian decomposition) into singly-bordered BDF






```
/* call MILP branch & price solver using method=auto */
solve with milp / decomp=(method=auto);
```

- NETWORK: Finds an embedded network [Bixby88] and uses the weakly connected components as subproblem blocks
  - subproblems solved with fast specialized solver (MCF)

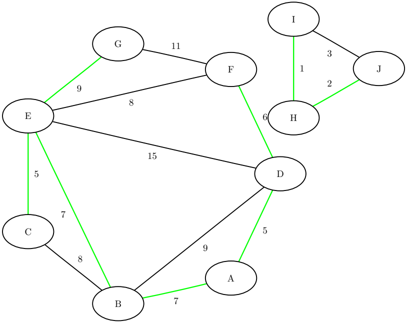
Multicommodity Flow Problem (MCF)

$$
\begin{array}{lrll}
\text{minimize} & \displaystyle\sum_{k\in K} \sum_{(i,j)\in A} c_{ij}^k x_{ij}^k & & \\
\text{subject to} & \displaystyle\sum_{k\in K} x_{ij}^k \leq u_{ij} & & (i,j) \in A \\
& \displaystyle\sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = b_i^k & & i \in N,\ k \in K \\
& 0 \leq x_{ij}^k \leq u_{ij}^k & & (i,j) \in A,\ k \in K
\end{array}
$$

```
/* call MILP branch & price solver using method=network */
solve with milp / decomp=(method=network);
```

- Network structures often occur as subproblems of meta-algorithms
- Specialized algorithms *orders of magnitude* faster than MP equivalents
- Concise representation of network objects (nodes/links)
  - minimizes memory consumption and slow expression evaluation
  - greatly simplifies the complexity of the OPTMODEL code

```
data LinksData;
   input from $ to $ cost @@;
   datalines;
A B  7   A D  5   B C 8   B D 9   B E 7
C E  5   D E 15   D F 6   E F 8   E G 9
F G 11   H I  1   I J 3   H J 2
;
proc optmodel;
   set<str,str> LINKS;
   num          cost{LINKS};
   read data LinksData into
      LINKS=[from to] cost;
   set<str,str> MST;
   solve with network / mst
      links = (weight = cost)
      out   = (forest = MST)
quit;
```



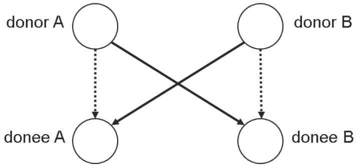
DEGREE-CONSTRAINED MST

```
proc optmodel;
   set<str,str> LINKS;
   num          cost{LINKS};
   read data LinksData into
      LINKS=[from to] cost;

   var x {<i,j> in LINKS} binary;

   /* degree constrained */
   con Degree {i in NODES}:
      sum {<u,v> in LINKS: i in {u,v}} x[u,v] <= max_degree;

   /* define the mapping between graph and variables */
   for {<i,j> in LINKS} do;
      x[i,j].from = i;
      x[i,j].to   = j;
   end;

   solve with MILP / decomp
      subprob = (solver=network mst links=(weight=cost));
quit;
```

work in progress (2014)

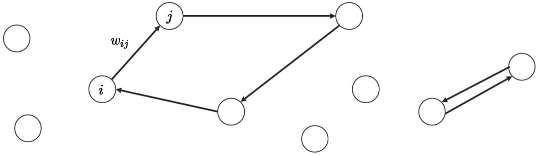
KIDNEY EXCHANGE PROBLEM

- Patient needs kidney, has a willing donor (typically family or friend)
- Suppose two couples:
  - donor A incompatible with donee A (but compatible with donee B)
  - donor B incompatible with donee B (but compatible with donee A)
- Can perform a two-way swap (more generally, n-way swaps)
- **CNN**: 7-way swap (8/5/2009), 13-way swap (12/14/2009)
- Edelman 2014 Finalist
  - Alliance for Paired Donation, Boston College, Stanford University, MIT



KIDNEY EXCHANGE PROBLEM

- Each node is incompatible donor-donee pair
- Arc (i, j) if donor from node i compatible with donee from node j
  - w_{ij} is a measure of the quality of the match (asymmetric)
- Find maximum weight node-disjoint union of (short) directed cycles
  - Mitigates risk
  - Practical considerations (travel, etc.)



- s_i defines a binary slack variable for node i
- y_{ic} defines a binary variable to indicate if node i is used in cycle c
- x_{ijc} defines a binary variable to indicate if arc (i,j) is used in cycle c

Kidney Exchange Problem

$$
\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{(i,j)\in A}\sum_{c\in C} w_{ij}x_{ijc} & \\
\text{subject to} & \displaystyle\sum_{c\in C} y_{ic} + s_i = 1 & i \in N \\
& \displaystyle\sum_{(i,j)\in A} x_{ijc} = y_{ic} & i \in N,\ c \in C \\
& \displaystyle\sum_{(i,j)\in A} x_{ijc} = y_{jc} & j \in N,\ c \in C \\
& \displaystyle\sum_{(i,j)\in A} x_{ijc} \le L & c \in C
\end{array}
$$

KIDNEY EXCHANGE PROBLEM — OPTMODEL

```
/* Slack[i] = 1 if node i is not used in any cycle */
var Slack{NODES} binary;
/* UseNode[i,c] = 1 if node i is used in cycle c, 0 otherwise */
var UseNode {NODES, CYCLES} binary;
/* UseArc[i,j,c] = 1 if arc (i,j) is used in cycle c, 0 otherwise */
var UseArc {ARCS, CYCLES} binary;

/* maximize total weight of arcs used */
max TotalWeight = sum {<i,j> in ARCS, c in CYCLES} weight[i,j] * UseArc[i,j,c];

/* each node appears in at most one cycle */
con node_packing {i in NODES}:
   sum {c in CYCLES} UseNode[i,c] + Slack[i] = 1;
```

SOFTWARE | KIDNEY EXCHANGE PROBLEM — OPTMODEL

```
/* Slack[i] = 1 if node i is not used in any cycle */
var Slack{NODES} binary;
/* UseNode[i,c] = 1 if node i is used in cycle c, 0 otherwise */
var UseNode {NODES, CYCLES} binary;
/* UseArc[i,j,c] = 1 if arc (i,j) is used in cycle c, 0 otherwise */
var UseArc {ARCS, CYCLES} binary;

/* maximize total weight of arcs used */
max TotalWeight = sum {<i,j> in ARCS, c in CYCLES} weight[i,j] * UseArc[i,j,c];

/* each node appears in at most one cycle */
con node_packing {i in NODES}:
   sum {c in CYCLES} UseNode[i,c] + Slack[i] = 1;
```

```
/* at most one donee for each donor */
con donate {i in NODES, c in CYCLES}:
   sum {<(i),j> in ARCS} UseArc[i,j,c] = UseNode[i,c];
/* at most one donor for each donee */
con receive {j in NODES, c in CYCLES}:
   sum {<i,(j)> in ARCS} UseArc[i,j,c] = UseNode[j,c];
/* exclude long cycles */
con cardinality {c in CYCLES}:
   sum {<i,j> in ARCS} UseArc[i,j,c] <= &max_length;

solve with MILP / decomp = (method = set);
```

SOFTWARE | KIDNEY EXCHANGE PROBLEM — OPTMODEL

```
/* Slack[i] = 1 if node i is not used in any cycle */
var Slack{NODES} binary;
/* UseNode[i,c] = 1 if node i is used in cycle c, 0 otherwise */
var UseNode {NODES, CYCLES} binary;
/* UseArc[i,j,c] = 1 if arc (i,j) is used in cycle c, 0 otherwise */
var UseArc {ARCS, CYCLES} binary;

/* maximize total weight of arcs used */
max TotalWeight = sum {<i,j> in ARCS, c in CYCLES} weight[i,j] * UseArc[i,j,c];

/* each node appears in at most one cycle */
con node_packing {i in NODES}:
   sum {c in CYCLES} UseNode[i,c] + Slack[i] = 1;
```

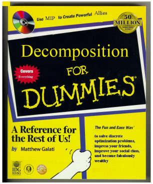
```
/* define the mapping between graph and variables */
set GNODES = setof {i in NODES, c in CYCLES} (i+(c-1)*n);
for {i in NODES, c in CYCLES}
   UseNode[i,c].node = i+(c-1)*n;
set GLINKS = setof {<i,j> in ARCS, c in CYCLES} <i+(c-1)*n, j+(c-1)*n>;
for {<i,j> in ARCS, c in CYCLES} do;
   UseArc[i,j,c].from = i+(c-1)*n;
   UseArc[i,j,c].to   = j+(c-1)*n;
end;

solve with MILP / decomp
   subprob = (solver = network
              links = (include=GLINKS)
              cycle = (maxlength=&max_length));
```

OUTLINE

RESULTS (MILP B&C VS DECOMP)

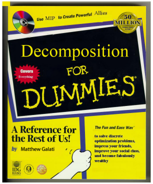
				Decomp		MILP B&C		
Source	Model/Industry	Instance	Method	Time	Gap	Time	Gap	△
Research	Bin Packing	n1c1w4m	Set	0.4	OPT	563	OPT	1443x
Customer	Coal Production	coalprod30	User	0.7	OPT	916	OPT	1237x
Customer	Marketing Optimization	farm500M	Auto	2.4	OPT	119	OPT	48x
Research	Kidney Exchange	kidney2_10_9	Set	12	OPT	481	OPT	41x
Research	Resource Allocation	rap_i71	User	65	OPT	2,252	OPT	34x
MIPLIB/NEOS	Unknown	ash608gpia_3col	Auto	10	OPT	329	OPT	33x
MIPLIB/NEOS	Unknown	roll3000	Auto	269	OPT	1,497	OPT	5x
Customer	Finance	atmnet01192	Net	102	OPT	234	OPT	2x
Customer	Finance	atmorig	User	49	OPT	T	∞	∞
Research	Edge Coloring	color_halljankograph	Set	586	OPT	T	240%	240%
Customer	Finance	design3_miqp	Set	11	OPT	T	168%	168%
Research	Wedding Planner	wed_40_6	Set	16	OPT	T	70%	70%
Research	Vertex Coloring	color_queenstour_7_8	Set	21	OPT	T	57%	57%
Research	Unit Commitment	unitcommitment	User	274	OPT	T	55%	55%
MIPLIB/NEOS	Unknown	neos_787933	Auto	47	OPT	T	43%	43%
Customer	Retail	carretail	User	655	OPT	T	19%	19%
Research	Vehicle Routing	vrp_E_n22_k4_3index	Set	211	OPT	T	17%	17%
Customer	Pharmaceuticals	pharma	User	28	OPT	T	8%	8%
Research	Cutting Stock	test0055	User	233	OPT	T	9%	9%
Customer	Retail Inventory	inventory	User	T	0.04%	T	∞	∞
MIPLIB/NEOS	Unknown	neos_631164	Auto	T	0.92%	T	41%	40%
MIPLIB/NEOS	Unknown	neos_787933	Auto	T	0.02%	T	36%	36%
Customer	Hotel Management	roomassign54	User	T	5%	T	33%	28%
Research	Machine Reassignment	modela12	User	T	8%	T	15%	7%
Customer	Forestry	forestry1	User	T	0.30%	T	2%	1%

- SAS DECOMP is applicable to a wide-range of models
- Provides a very easy-to-use interface through OPTMODEL
  - a good alternative solver to standard branch & cut for difficult models
- Improved bound strength
  - $z_{DW} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{CP} \geq z_{LP}$
- Faster bound resolution (in some cases)
  - exploiting parallelism in block-angular case
  - using specialized oracles (e.g., `method=network`)
- Eliminates symmetry in the case of aggregate formulation

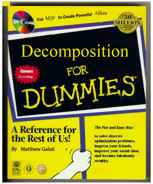


CONCLUSION | SUMMARY

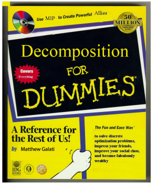
- SAS DECOMP is applicable to a wide-range of models
- Provides a very easy-to-use interface through OPTMODEL
  - a good alternative solver to standard branch & cut for difficult models
- Improved bound strength
  - $z_{\mathrm{DW}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{\mathrm{CP}} \geq z_{\mathrm{LP}}$
- Faster bound resolution (in some cases)
  - exploiting parallelism in block-angular case
  - using specialized oracles (e.g., `method=network`)
- Eliminates symmetry in the case of aggregate formulation



- SAS DECOMP is applicable to a wide-range of models
- Provides a very easy-to-use interface through OPTMODEL
  - a good alternative solver to standard branch & cut for difficult models
- Improved bound strength
  - $z_{\text{DW}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{\text{CP}} \geq z_{\text{LP}}$
- Faster bound resolution (in some cases)
  - exploiting parallelism in block-angular case
  - using specialized oracles (e.g., `method=network`)
- Eliminates symmetry in the case of aggregate formulation

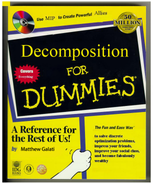


- SAS DECOMP is applicable to a wide-range of models
- Provides a very easy-to-use interface through OPTMODEL
  - a good alternative solver to standard branch & cut for difficult models
- Improved bound strength
  - $z_{\text{DW}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{\text{CP}} \geq z_{\text{LP}}$
- Faster bound resolution (in some cases)
  - exploiting parallelism in block-angular case
  - using specialized oracles (e.g., `method=network`)
- Eliminates symmetry in the case of aggregate formulation

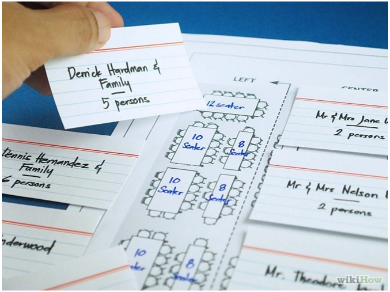


FUTURE ENHANCEMENTS

- In-memory integration of other solvers as DECOMP subprob oracles
  - currently restricted to LP, MILP, and MCF
  - `subprob=(solver=network)` — all network algorithms available
  - `subprob=(solver=nlp)` — nonlinear oracle
  - `subprob=(solver=clp)` — constraint programming oracle
  - `subprob=(solver=lso)` — local search oracle (allows black box evals)
- Automate Benders decomposition for complicating variables



CONCLUSION | WEDDING — SUCCESS!



where do we put this guy?

WEDDING — FAILURE!

http://support.sas.com/or

Decomposition for Dummies