DIP and DipPy: A Decomposition-based Modeling System and Solver

Ted Ralphs and Jiadong Wang
Lehigh University
Matthew Galati
SAS Institute
Mike O'Sullivan
University of Auckland

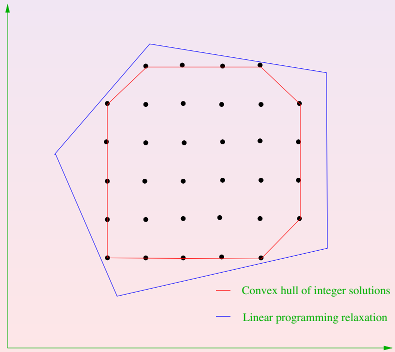
IFORS, Barcelona, Spain, 17 July, 2014

Thanks: Work supported in part by the National Science Foundation

Basic Setting

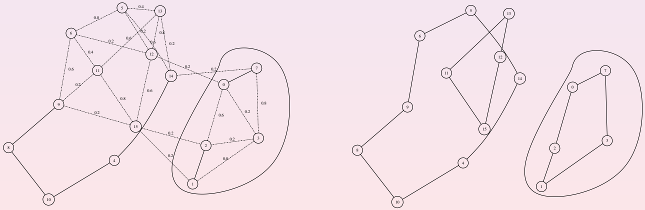
Integer Linear Program: Minimize/Maximize a linear *objective function* over a (discrete) set of *solutions* satisfying specified *linear constraints*.

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



What is Decomposition?

- Many complex models are built up from simpler structures.
  - Subsystems linked by system-wide constraints or variables.
  - Complex combinatorial structures obtained by combining simpler ones.
- Decomposition is the process of breaking a model into smaller parts.
- The goal is either to
  - reformulate the model for easier solution;
  - reformulate the model to obtain an improved relaxation (bound); or
  - separate the model into stages or levels (possibly with separate objectives).



Block Structure

- "Classical" decomposition arises from *block structure* in the constraints.
- By relaxing/fixing the linking variables/constraints, we get a separable model.
- A separable model consists of smaller submodels that are easier to solve.
- The separability lends itself nicely to parallel implementation.

$$\begin{pmatrix} A_{01} & A_{02} & \cdots & A_{0\kappa} \\ A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_{\kappa\kappa} \end{pmatrix} \quad \begin{pmatrix} A_{10} & A_{11} & & & \\ A_{20} & & A_{22} & & \\ \vdots & & & \ddots & \\ A_{\gamma 0} & & & & A_{\kappa\kappa} \end{pmatrix}$$

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} & \cdots & A_{0\kappa} \\ A_{10} & A_{11} & & & \\ A_{20} & & A_{22} & & \\ \vdots & & & \ddots & \\ A_{\gamma 0} & & & & A_{\kappa\kappa} \end{pmatrix}$$

The Decomposition Principle (in MIP)

- Decomposition methods leverage our ability to solve either a relaxation or a restriction.
- Methodology is based on the ability to solve a given *subproblem* repeatedly with varying inputs.
- The goal of solving the subproblem repeatedly is to obtain information about its structure that can be incorporated into a *master problem*.

Constraint decomposition

- Relax a set of *linking constraints* to expose structure.
- Leverages ability to solve either the optimization or separation problem for a *relaxation* (with varying objectives and/or points to be separated).

Variable decomposition

- Fix the values of *linking variables* to expose the structure.
- Leverages ability to solve a *restriction* (with varying right-hand sides).

Example: Facility Location Problem

- We have n locations and m customers to be served from those locations.
- There is a fixed cost c_j and a capacity W_j associated with facility j.
- There is a cost d_{ij} and demand w_{ij} for serving customer i from facility j.
- We have two sets of binary variables.
  - y_j is 1 if facility j is opened, 0 otherwise.
  - x_{ij} is 1 if customer i is served by facility j, 0 otherwise.

Capacitated Facility Location Problem

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} & \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1 & \forall i \\
& \sum_{i=1}^{m} w_{ij} x_{ij} \le W_j y_j & \forall j \\
& x_{ij}, y_j \in \{0,1\} & \forall i, j
\end{aligned}$$

DIP/DipPy: Decomposition-based Modeling and Solution

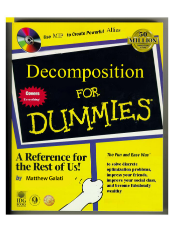
DIP (w/ M. Galati and J. Wang)

DIP is a software framework and stand-alone solver for implementation and use of a variety of decomposition-based algorithms.

- Decomposition-based algorithms have traditionally been difficult to implement and compare.
- DIP abstracts the common, generic elements of these methods.
  - **Key:** API is in terms of the compact formulation.
  - The framework takes care of reformulation and implementation.
  - DIP is now a *fully generic* decomposition-based parallel MILP solver.

DipPy (w/ M. O'Sullivan)

- Python-based modeling language.
- User can express decompositions in a "natural" way.
- Allows access to multiple decomposition methods.



⇐ *Joke* !

CHiPPS (w/ Y. Xu)

- CHiPPS is the COIN-OR High Performance Parallel Search.
- CHiPPS is a set of C++ class libraries for implementing tree search algorithms for both sequential and parallel environments.

CHiPPS Components (Current)

ALPS (Abstract Library for Parallel Search)

- is the search-handling layer (parallel and sequential).
- provides various search strategies based on node priorities.

BiCePS (Branch, Constrain, and Price Software)

- is the data-handling layer for relaxation-based optimization.
- adds notion of variables and constraints.
- assumes iterative bounding process.

BLIS (BiCePS Linear Integer Solver)

- is a concretization of BiCePS.
- specific to models with linear constraints and objective function.

DIP: Overview of Methods

Cutting Plane Method (CPM)

CPM combines an *outer* approximation of $\mathcal{P}'$ with an explicit description of $\mathcal{Q}''$

- **Master**: $z_{\mathrm{CP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid Dx \geq d, A''x \geq b'' \right\}$
- **Subproblem**: $\mathrm{SEP}(\mathcal{P}', x_{\mathrm{CP}})$

Dantzig-Wolfe Method (DW)

DW combines an *inner* approximation of $\mathcal{P}'$ with an explicit description of $\mathcal{Q}''$

- **Master**: $z_{\mathrm{DW}} = \min_{\lambda \in \mathbb{R}_+^{\mathcal{E}}} \left\{ c^\top \left(\sum_{s \in \mathcal{E}} s \lambda_s \right) \mid A'' \left(\sum_{s \in \mathcal{E}} s \lambda_s \right) \geq b'', \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{DW}}^\top A''\right)$

Lagrangian Method (LD)

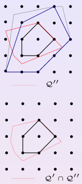
LD iteratively produces single extreme points of $\mathcal{P}'$ and uses their violation of constraints of $\mathcal{Q}''$ to converge to the same optimal face of $\mathcal{P}'$ as CPM and DW.

- **Master**: $z_{\mathrm{LD}} = \max_{u \in \mathbb{R}_+^{m''}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (b'' - A''s) \right\} \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{LD}}^\top A''\right)$

DIP: Common Threads

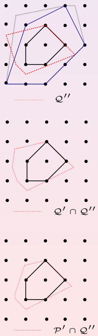
- The LP bound is obtained by optimizing over the intersection of two explicitly defined polyhedra.

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



- The decomposition bound is obtained by optimizing over the intersection of two polyhedra.

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



- Decomposition-based bounding methods have two main steps
  - **Master Problem:** Update the primal/dual **solution** information
  - **Subproblem:** Update the **approximation** of $\mathcal{P}'$: $\mathrm{SEP}(\mathcal{P}', x)$ or $\mathrm{OPT}(\mathcal{P}', c)$
- Integrated decomposition methods further improve the bound.
  - **Price-and-Cut** (PC)
  - **Relax-and-Cut** (RC)
  - **Decompose-and-Cut** (DC)

Generic Decomposition-based Branch and Bound

- Traditionally, decomposition-based branch-and-bound methods have required extensive problem-specific customization.

    - Identifying the decomposition (which constraints to relax).
    - Formulating and solving the subproblem.
    - Formulating and solving the master problem.
    - Performing the branching operation.

- However, it is possible to replace these components with generic alternatives.

    - The decomposition can be identified automatically by analyzing the matrix or through a modeling language.
    - The subproblem can be solved with a generic MILP solver.
    - The branching can be done in the original compact formulation.

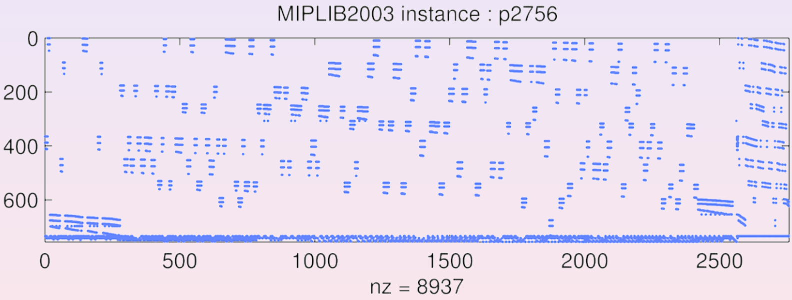
- The remainder of the talk focuses on the crucial first step.

Automatic Structure Detection

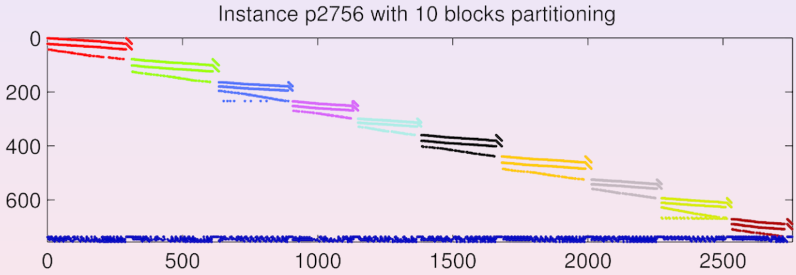
- For problems in which the structure is not given, it may be detected automatically.
- Hypergraph partitioning methods can be used to identify the structure.
- We map each row of the original matrix to a hyperedge and the nonzero elements to nodes in a hypergraph.
- We use a partitioning model/algorithm (hMetis) that identifies a singly-bordered block diagonal matrix with a given number of blocks.

$$
\begin{pmatrix}
* & * & & & & * \\
* & & & * & * & \\
& * & & & * & * \\
* & & * & & & \\
* & * & & * & * & \\
& * & * & & * & \\
& & * & * & * & \\
* & & & & &
\end{pmatrix}
$$

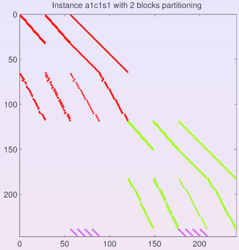
Hidden Block Structure

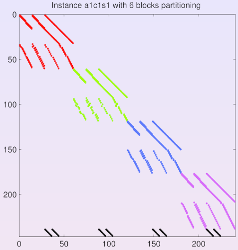


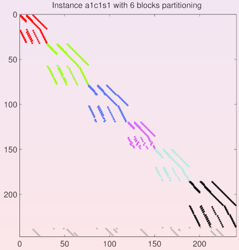
Hidden Block Structure

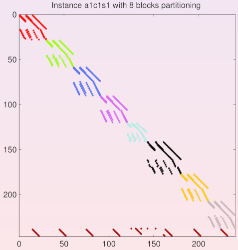


Choosing the Block Number









Quality Measures for Decomposition

- The goal of the partitioning is to have a “good decomposition.”
- Generally, we judge goodness in terms of bound and computation time.

Potential Features

- The fraction of nonzero elements in the matrix appearing in the coupling rows (α),
- The fraction of nonzero elements appearing in the coupling rows that are in integer columns (β),
- The fraction of the nonzero elements in integer columns in the matrix that appear in coupling rows (γ),
- The average fraction of the nonzeros in each block that are in integer columns (η),
- The standard deviation of the fraction of integer elements elements in the blocks (θ).

$$\Pi = (1 - \min(\alpha, \gamma))) \times 100\%,$$

Finding the Structure

- In many cases, there is a "natural" block structure arising from the original model.
- Problems for which decomposition is the "killer approach" often have identical blocks, since this leads to symmetry in the compact formulation.
- We would like to be able to identify this structure automatically.
- One simple strategy is to make a frequency table.

# of Nonzeros	2	11	12	13	24	40	100
# of Rows	2220	20	20	2	1998	100	20

Table: Histogram for atm20-100

# of Nonzeros	2	3	5	6	7	8	9	10	11	13
# of Rows	9	130	221	4	8	8	7	6	2	1

Table: Histogram for glass4

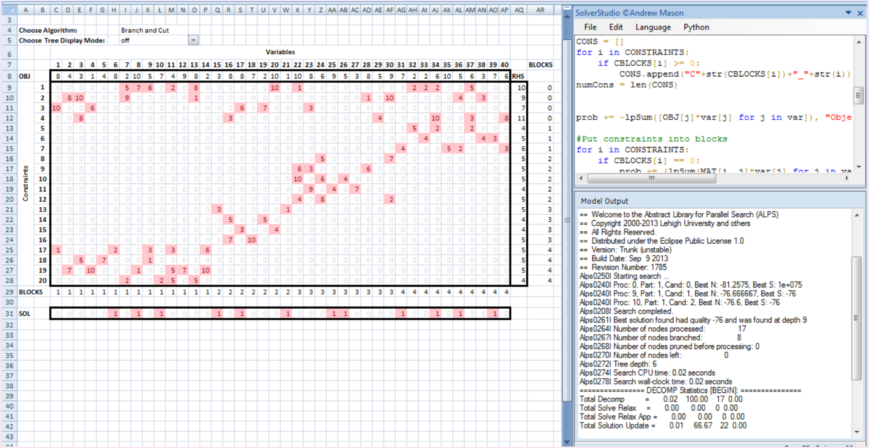
Specifying Blocks with DipPy: Facility Location Example

```
from products   import REQUIREMENT, PRODUCTS
from facilities import FIXED_CHARGE, LOCATIONS, CAPACITY

prob = dippy.DipProblem("Facility_Location")

ASSIGNMENTS = [(i, j) for i in LOCATIONS for j in PRODUCTS]
assign_vars = LpVariable.dicts("x", ASSIGNMENTS, 0, 1, LpBinary)
use_vars    = LpVariable.dicts("y", LOCATIONS, 0, 1, LpBinary)

prob += lpSum(use_vars[i] * FIXED_COST[i] for i in LOCATIONS)

for j in PRODUCTS:
    prob += lpSum(assign_vars[(i, j)] for i in LOCATIONS) == 1

for i in LOCATIONS:
    prob.relaxation[i] += lpSum(assign_vars[(i, j)] * REQUIREMENT[j]
                                for j in PRODUCTS) <= CAPACITY * use_vars[i]

dippy.Solve(prob, {doPriceCut:1})
```

DipPy Callbacks

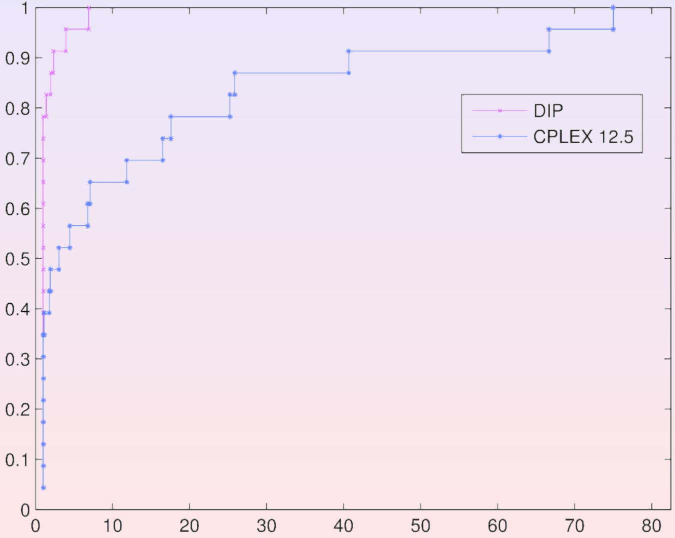
```
def solve_subproblem(prob, index, redCosts, convexDual):
   ...
   return knapsack01(obj, weights, CAPACITY)
def knapsack01(obj, weights, capacity):
    ...
    return solution
def first_fit(prob):
    ...
    return bvs
prob.init_vars = first_fit
def choose_branch(prob, sol):
   ...
   return ([], down_branch_ub, up_branch_lb, [])
def generate_cuts(prob, sol):
    ...
    return new_cuts
def heuristics(prob, xhat, cost):
    ...
    return sols
dippy.Solve(prob, {'doPriceCut': '1'})
```

DipPy with Solver Studio

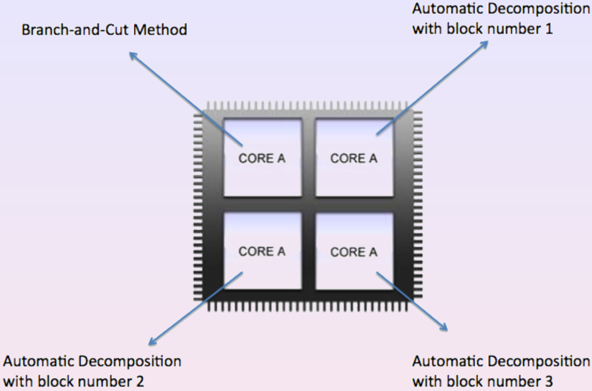


Using DipPy with SolverStudio

Brief Computational Results



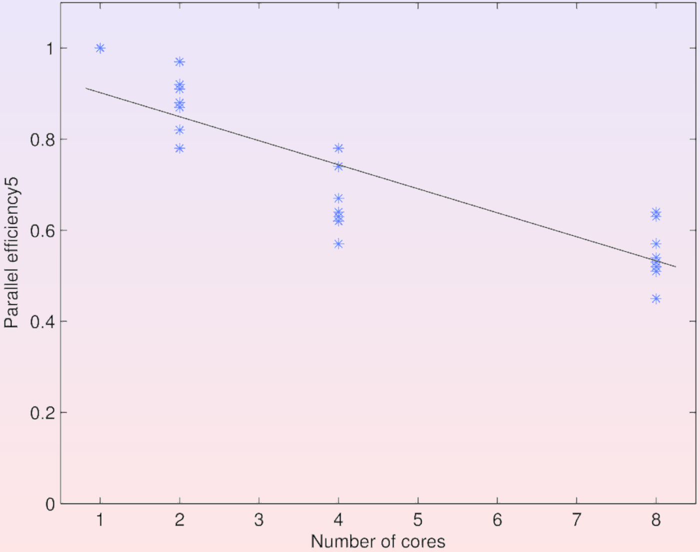
Exploiting Concurrency



Concurrency can be exploited in multiple ways.

- Solving the subproblems
- Exploring the tree
- Determining the decomposition (or whether to use decomposition)

Brief Computational Results



Future work

Where do I start??

- We have only scratched the surface of what is needed to make a true generic decomposition-based solver.
- The implementation needs many improvements in basic components.
- We need a better decision logic for when to use which algorithm.
- We need better support for identical blocks.
- To exploit parallelism, we need the ability to dynamically allocate cores after the initial phase.
- We need more testing on hybrid distributed/shared parallelism.
- Methods that hybridize CP and MIP through the decomposition would be interesting.

Want to help :)?

Get DIP and DipPy

www.coin-or.org/DIP

easy_install coinor.dippy

Questions?