DANTZIG-WOLFE DECOMPOSITION WITH GAMS

ERWIN KALVELAGEN

Abstract. This document illustrates the Dantzig-Wolfe decomposition algorithm using GAMS.

1. INTRODUCTION

Dantzig-Wolfe decomposition [2] is a classic solution approach for structured linear programming problems. In this document we will illustrate how Dantzig-Wolfe decomposition can be implemented in a GAMS environment. The GAMS language is rich enough to be able to implement fairly complex algorithms as is illustrated by GAMS implementations of Benders Decomposition [10], Cutting Stock Column Generation [11] and branch-and-bound algorithms [12].

Dantzig-Wolfe decomposition has been an important tool to solve large structured models that could not be solved using a standard Simplex algorithm as they exceeded the capacity of those solvers. With the current generation of simplex and interior point LP solvers and the enormous progress in standard hardware (both in terms of raw CPU speed and availability of large amounts of memory) the Dantzig-Wolfe algorithm has become less popular.

Implementations of the Dantzig-Wolfe algorithm have been described in [5, 6, 7]. Some renewed interest in decomposition algorithms was inspired by the availability of parallel computer architectures [8, 13]. A recent computational study is [16]. [9] discusses formulation issues when applying decomposition on multi-commodity network problems. Many textbooks on linear programming discuss the principles of the Dantzig-Wolfe decomposition [1, 14].

2. BLOCK-ANGULAR MODELS

Consider the LP:

\[
\begin{align*}
\min \ & c^T x \\
Ax & = b \\
x & \geq 0
\end{align*}
\]

(1)

where \( A \) has a special structure:

\[
\begin{pmatrix}
B_0 & B_1 & B_2 & \ldots & B_K \\
A_1 & \quad & \quad & \quad & \\
A_2 & \quad & \quad & \quad & \\
\vdots & \quad & \quad & \quad & \\
A_K & \quad & \quad & \quad & \\
\end{pmatrix}
\begin{pmatrix}
x_0 \\
x_1 \\
x_2 \\
\vdots \\
x_K
\end{pmatrix}
= \begin{pmatrix}
b_0 \\
b_1 \\
b_2 \\
\vdots \\
b_K
\end{pmatrix}
\]

(2)
The constraints

\[ \sum_{k=0}^{K} B_k x_k = b_0 \]

corresponding to the top row of sub-matrices are called the **coupling constraints**.

The idea of the Dantzig-Wolfe approach is to decompose this problem, such that never a problem has to be solved with all sub-problems \( A_k x_k = b_k \) included. Instead a master problem is devised which only concentrates on the coupling constraints, and the sub-problems are solved individually. As a result only a series of smaller problems need to be solved.

### 3. Minkowski’s Representation Theorem

Consider the feasible region of an LP problem:

\[ P = \{ x | Ax = b, x \geq 0 \} \]

If \( P \) is bounded then we can characterize any point \( x \in P \) as a linear combination of its extreme points \( x^{(j)} \):

\[ x = \sum_j \lambda_j x^{(j)} \]

\[ \sum_j \lambda_j = 1 \]

\[ \lambda_j \geq 0 \]

If the feasible region can not assumed to be bounded we need to introduce the following:

\[ x = \sum_j \lambda_j x^{(j)} + \sum_i \mu_i r^{(i)} \]

\[ \sum_j \lambda_j = 1 \]

\[ \lambda_j \geq 0 \]

\[ \mu_i \geq 0 \]

where \( r^{(i)} \) are the extreme rays of \( P \). The above expression for \( x \) is sometimes called **Minkowski’s Representation Theorem**[15]. The constraint \( \sum_j \lambda_j = 1 \) is also known as the **convexity constraint**.

A more compact formulation is sometimes used:

\[ x = \sum_j \delta_j x^{(j)} \]

\[ \sum_j \delta_j \lambda_j = 1 \]

\[ \lambda_j \geq 0 \]

where

\[ \delta_j = \begin{cases} 1 & \text{if } x^{(j)} \text{ is an extreme point} \\ 0 & \text{if } x^{(j)} \text{ is an extreme ray} \end{cases} \]
I.e. we can describe the problem in terms of variables $\lambda$ instead of the original variables $x$. In practice this reformulation can not be applied directly, as the number of variables $\lambda_{j}$ becomes very large.

4. The Decomposition

The $K$ subproblems are dealing with the constraints

\begin{equation}
A_k x_k = b_k \\
x_k \geq 0
\end{equation}

while the Master Problem is characterized by the equations:

\begin{equation}
\min \sum_k c_k^T x_k \\
\sum_k B_k x_k = b_0 \\
x_0 \geq 0
\end{equation}

We can substitute equation 7 into 10, resulting in:

\begin{equation}
\min c_0^T x_0 + \sum_{k=1}^{p_k} \sum_{j=1}^{p_k} (c_k^T x_k^{(j)}) \lambda_{k,j} \\
B_0 x_0 + \sum_{k=1}^{p_k} \sum_{j=1}^{p_k} (B_k x_k^{(j)}) \lambda_{k,j} = b_0 \\
\sum_{j=1}^{p_k} \delta_{k,j} \lambda_{k,j} = 1 \text{ for } k = 1, \ldots, K \\
x_0 \geq 0 \\
\lambda_{k,j} \geq 0
\end{equation}

This is a huge LP. Although the number of rows is reduced, the number of extreme points and rays $x_k^{(j)}$ of each subproblem is very large, resulting in an enormous number of variables $\lambda_{k,j}$. However many of these variables will be non-basic at zero, and need not be part of the problem. The idea is that only variables with a promising reduced cost will be considered in what is also known as a delayed column generation algorithm.

The model with only a small number of the $\lambda$ variables, compactly written as:

\begin{equation}
\min c_0^T x_0 + c^T \lambda' \\
B_0 x_0 + B \lambda' = b_0 \\
\Delta \lambda' = 1 \\
x_0 \geq 0 \\
\lambda' \geq 0
\end{equation}

is called the restricted master problem. The missing variables are fixed at zero. The restricted master problem is not fixed in size: variables will be added to this problem during execution of the algorithm.
The attractiveness of a variable $\lambda_{k,j}$ can be measured by its reduced cost\(^1\): If we denote the dual variables for constraint $B_0x_0 + B\lambda' = b_0$ by $\pi_1$ and those for the convexity constraints $\sum_j \delta_{k,j}\lambda'_{k,j} = 1$ by $\pi_2^{(k)}$, then reduced cost for the master problem look like:

$$\sigma_{k,j} = (c^T_k - \pi_1^T B_k)x_{k}^{(j)} - \pi_2^{(k)}\delta_{k,j}$$

Assuming the sub-problem to be bounded, the most attractive bfs (basic feasible solution) $x_k$ to enter the master problem is found by maximizing the reduced cost giving the following LP:

$$\min_{x_k} \sigma_k = (c^T_k - \pi_1^T B_k)x_{k} - \pi_2^{(k)}$$

$$B_kx = b_k$$

$$x_k \geq 0$$

The operation to find these reduced costs is often called Pricing. If $\sigma^*_k < 0$ we can introduce the a new column $\lambda_{k,j}$ to the master, with a cost coefficient of $c^T_k x^*_k$.

A basic Dantzig-Wolfe decomposition algorithm can now be formulated:

**Dantzig-Wolfe decomposition algorithm.**

{\textit{initialization}}

Choose initial subsets of variables.

\begin{verbatim}
while true do
\end{verbatim}

\(^1\)The reduced cost of a variable $x_j$ is

$$\sigma_j = c_j - \pi^T A_j$$

where $A_j$ is the column of $A$ corresponding to variable $x_j$, and $\pi$ are the duals.
\{Master problem\}
Solve the restricted master problem.
\[ \pi_1 := \text{duals of coupling constraints} \]
\[ \pi_2^{(k)} := \text{duals of the } k^{\text{th}} \text{ convexity constraint} \]
\{Sub problems\}
\textbf{for } k=1, \ldots, K \textbf{ do}
\begin{align*}
\text{Plug } \pi_1 \text{ and } \pi_2^{(k)} \text{ into sub-problem } k \\
\text{Solve sub-problem } k \\
\text{if } \sigma_k^* < 0 \textbf{ then} \\
\text{Add proposal } x_k^* \text{ to the restricted master} \\
\textbf{end if}
\end{align*}
\textbf{end for}
\textbf{if No proposals generated } \textbf{then}
\begin{align*}
\text{Stop: optimal} \\
\textbf{end if}
\end{align*}
\textbf{end while}

5. Initialization
We did not pay attention to the initialization of the decomposition.
The first thing we can do is solve each sub-problem:
\begin{align*}
\min & \quad c_k^T x_k \\
\text{subject to} & \quad A_k x_k = b_k \\
& \quad x_k \geq 0
\end{align*}
(16)
If any of the subproblems is infeasible, the original problem is infeasible. Otherwise,
we can use the optimal values \( x_k^* \) (or the unbounded rays) to generate an initial set
of proposals.

6. Phase I/II algorithm
The initial proposals may violate the coupling constraints. We can formulate
a Phase I problem by introducing artificial variables and minimizing those. The
use of artificial variables is explained in any textbook on Linear Programming (e.g.
[13]). It is noted that the reduced cost for a Phase I problem are slightly different
from the Phase II problem.

As an example consider that the coupling constraints are
\[ \sum_j x_j \leq b \]
(17)
We can add an artificial variable \( x_a \geq 0 \) as follows:
\[ \sum_j x_j - x_a \leq b \]
(18)
The phase I objective will be:
\begin{align*}
\min & \quad x_a \\
\text{subject to} & \quad c_k^T x_k = 0
\end{align*}
(19)
The reduced cost of a variable \( x_j \) is now as in equation (14) but with \( c_k^T = 0 \).
It is noted that it is important to remove artificials once a phase II starts. We
do this in the example code by fixing the artificial variables to zero.
7. Example: Multi-Commodity Network Flow

The multi-commodity network flow (MCNF) problem can be stated as:

\[
\begin{align*}
\text{min} & \quad \sum_{k \in K} \sum_{(i,j) \in A} c_{i,j}^k x_{i,j}^k \\
\text{s.t.} & \quad \sum_{(i,j) \in A} x_{i,j}^k - \sum_{(j,i) \in A} x_{j,i}^k = b_j^k \\
& \quad \sum_{k \in K} x_{i,j}^k \leq u_{i,j} \\
& \quad x_{i,j}^k \geq 0
\end{align*}
\]

(20)

This is sometimes called the node-arc formulation.

Dantzig-Wolfe decomposition is a well-known solution strategy for this type of problems. For each commodity a subproblem is created.

We consider here a multi-commodity transportation problem:

\[
\begin{align*}
\text{min} & \quad \sum_{k \in K} \sum_{(i,j)} c_{i,j}^k x_{i,j}^k \\
\text{s.t.} & \quad \sum_{j} x_{i,j}^k = \text{supply}_i^k \\
& \quad \sum_{i} x_{i,j}^k = \text{demand}_j^k \\
& \quad \sum_{k \in K} x_{i,j}^k \leq u_{i,j} \\
& \quad x_{i,j}^k \geq 0
\end{align*}
\]

(21)

with data from [4]. A similar Dantzig-Wolfe decomposition algorithm written in AMPL can be found in [3].

Model dw.gms. [*]
table demand(p,j)
    FRA DET LAN WIN STL FRE LAF
bands 300 300 100 75 650 225 250
coils 500 750 400 250 950 850 500
plate 100 100 0 50 200 100 250;

parameter limit(i,j);
limit(i,j) = 625;

table cost(p,i,j) 'unit cost'
    FRA DET LAN WIN STL FRE LAF
BANDS.GARY  30 10  8 10 11  71  6
BANDS.CLEV  22  7 10  7 21  82 13
BANDS.PITT   19 11 12 10 25  83 15
COILS.GARY   39 14 11 14 16  82  8
COILS.CLEV   27  9 12  9 26  95 17
COILS.PITT   24 14 17 13 28  99 20
PLATE.GARY   41 15 12 16 17  86  8
PLATE.CLEV   29  9 13  9 28  99 18
PLATE.PITT   26 14 17 13 31 104 20;

*-----------------------------------------------------------------------
* direct LP formulation
*-----------------------------------------------------------------------

positive variable
    x(i,j,p)  'shipments'  
;

variable
    z  'objective variable'  
;

equations
    obj
         supplyc(i,p)
         demandc(j,p)
         limitc(i,j)
    ;

    obj..  z =e=  sum((i,j,p),  cost(p,i,j)*x(i,j,p));

    supplyc(i,p)..  sum(j,  x(i,j,p)) =e=  supply(p,i);
    demandc(j,p)..  sum(i,  x(i,j,p)) =e=  demand(p,j);
    limitc(i,j)..  sum(p,  x(i,j,p)) =l=  limit(i,j);

model m/all/;
solve m minimizing z using lp;

*-----------------------------------------------------------------------
* subproblems
*-----------------------------------------------------------------------

positive variables xsub(i,j);
variables zsub;

parameters
    s(i)  'supply'
    d(j)  'demand'
    c(i,j)  'cost coefficients'
    pi1(i,j)  'dual of limit'
    pi2(p)  'dual of convexity constraint'
ERWIN KALVELAGEN

pi2p
;

equations
  supply_sub(i)  'supply equation for single product'
  demand_sub(j) 'demand equation for single product'
  rc1_sub   'phase 1 objective'
  rc2_sub   'phase 2 objective'
;
  supply_sub(i).. sum(j, xsub(i,j)) =e= s(i);
  demand_sub(j).. sum(i, xsub(i,j)) =e= d(j);
  rc1_sub.. zsub =e= sum((i,j), -pi1(i,j)*xsub(i,j)) - pi2p;
  rc2_sub.. zsub =e= sum((i,j), (c(i,j)-pi1(i,j))*xsub(i,j)) - pi2p;
model sub1 'phase 1 subproblem' /supply_sub, demand_sub, rc1_sub/
model sub2 'phase 2 subproblem' /supply_sub, demand_sub, rc2_sub/

*-----------------------------------------------------------------------
* master problem
*-----------------------------------------------------------------------

set k 'proposal count' /proposal1*proposal1000/;
set pk(p,k);
pk(p,k) = no;
parameter proposal(i,j,p,k);
parameter proposalcost(p,k);
proposal(i,j,p,k) = 0;
proposalcost(p,k) = 0;

positive variables
  lambda(p,k)
  excess 'artificial variable'
;
variable zmaster;
equations
  obj1_master   'phase 1 objective'
  obj2_master   'phase 2 objective'
  limit_master(i,j)
  convex_master
  ;
  obj1_master.. zmaster =e= excess;
  obj2_master.. zmaster =e= sum(pk, proposalcost(pk)*lambda(pk));
  limit_master(i,j)
    .. sum(pk, proposal(i,j,pk)*lambda(pk)) =l= limit(i,j) + excess;
  convex_master(p)
    .. sum(pk(p,k), lambda(p,k)) =e= 1;
model master1 'phase 1 master' /obj1_master, limit_master, convex_master/;
model master2 'phase 2 master' /obj2_master, limit_master, convex_master/;

*-----------------------------------------------------------------------
* options to reduce solver output
*-----------------------------------------------------------------------

option limrow=0;
option limcol=0;
master1.solprint = 2;
master2.solprint = 2;
sub1.solprint = 2;
sub2.solprint = 2;

* options to speed up solver execution

master1.solvelink = 2;
master2.solvelink = 2;
sub1.solvelink = 2;
sub2.solvelink = 2;

* DANTZIG-WOLFE INITIALIZATION PHASE
* test subproblems for feasibility
* create initial set of proposals

display "------------------------------------------------------------------",
"INITIALIZATION PHASE",
"------------------------------------------------------------------";

set kk(k) 'current proposal';
kk('proposal1') = yes;

loop(p,

* solve subproblem, check feasibility
*
  c(i,j) = cost(p,i,j);
s(i) = supply(p,i);
d(j) = demand(p,j);
pi1(i,j) = 0;
pi2p = 0;
solve sub2 using lp minimizing zsub;
abort$(sub2.modelstat = 4) "SUBPROBLEM IS INFEASIBLE: ORIGINAL MODEL IS INFEASIBLE";
abort$(sub2.modelstat <> 1) "SUBPROBLEM NOT SOLVED TO OPTIMALITY";

*
* proposal generation
*
  proposal(i,j,p,kk) = xsub.l(i,j);
  proposalcost(p,kk) = sum((i,j), c(i,j)*xsub.l(i,j));
  pk(p,kk) = yes;
  kk(k) = kk(k-1);
);

option proposal:2:2:2;
display proposal;

* DANTZIG-WOLFE ALGORITHM
* while (true) do
*   solve restricted master
*   solve subproblems
*   until no more proposals

set iter 'maximum iterations' /iter1*iter15/;
scalar done /0/;
scalar count /0/;
scalar phase /1/;
scalar iteration;

loop(iter$(not done),

  iteration = ord(iter);
  display "------------------------------------------------------------------",
  "iteration",
  "------------------------------------------------------------------";
* solve master problem to get duals
  * if (phase=1,
    solve master1 minimizing zmaster using lp;
    abort$(master1.modelstat <> 1) "MASTERPROBLEM NOT SOLVED TO OPTIMALITY";
    if (excess.l < 0.0001,
      display "Switching to phase 2";
      phase = 2;
      excess.fx = 0;
    );
  );
  if (phase=2,
    solve master2 minimizing zmaster using lp;
    abort$(master2.modelstat <> 1) "MASTERPROBLEM NOT SOLVED TO OPTIMALITY";
  );
  pi1(i,j) = limit_master.m(i,j);
  pi2(p) = convex_master.m(p);
  count = 0;
  loop(p$(not done),
    * solve each subproblem
    * c(i,j) = cost(p,i,j);
    s(i) = supply(p,i);
    d(j) = demand(p,j);
    pi2p = pi2(p);
    if (phase=1,
      solve sub1 using lp minimizing zsub;
      abort$(sub1.modelstat = 4) "SUBPROBLEM IS INFEASIBLE: ORIGINAL MODEL IS INFEASIBLE";
      abort$(sub1.modelstat <> 1) "SUBPROBLEM NOT SOLVED TO OPTIMALITY";
    else
      solve sub2 using lp minimizing zsub;
      abort$(sub2.modelstat = 4) "SUBPROBLEM IS INFEASIBLE: ORIGINAL MODEL IS INFEASIBLE";
      abort$(sub2.modelstat <> 1) "SUBPROBLEM NOT SOLVED TO OPTIMALITY";
    );
    * proposal
    * if (zsub.l < -0.0001,
      count = count + 1;
      display "new proposal", count,xsub.l;
      proposal(i,j,p,kk) = xsub.l(i,j);
      proposalcost(p,kk) = sum((i,j), c(i,j)*xsub.l(i,j));
      kk(k) = yes;
      k(k) = k(k-1);
    );
    * no new proposals?
    * abort$(count = 0 and phase = 1) "PROBLEM IS INFEASIBLE";
    done$(count = 0 and phase = 2) = 1;
  );
  abort$(not done) "Out of iterations";

*-----------------------------------------------------------------------
* recover solution
*-----------------------------------------------------------------------
parameter xsol(i,j,p);
xsol(i,j,p) = sum(pk(p,k), proposal(i,j,pk)*lambda.l(pk));
display xsol;

parameter totalcost;
totalcost = sum((i,j,p), cost(p,i,j)*xsol(i,j,p));
display totalcost;

The reported solution is:

--- 317 PARAMETER xsol
    bands  coils  plate
GARY.STL  400.000  64.099  160.901
GARY.FRE   625.000
GARY.LAF   110.901  39.099
CLEV.FRA   264.099  10.901
CLEV.DET   67.906  457.094  100.000
CLEV.LAN   400.000
CLEV.WIN   43.972  169.025  50.000
CLEV.STL   250.000  260.901  39.099
CLEV.FRE   162.003  100.000
CLEV.LAF   74.024  150.976
PITT.FRA   35.901  500.000  89.099
PITT.DET   232.094  292.906
PITT.LAN   100.000
PITT.WIN   43.972  169.025  50.000
PITT.STL   250.000  260.901  39.099
PITT.FRE   162.003  100.000
PITT.LAF   74.024  150.976

--- 321 PARAMETER totalcost = 199500.000

References
1. V. Chvatal, Linear programming, Freeman, 1983.

Amsterdam Optimization Modeling Group LLC, Washington D.C./The Hague
E-mail address: erwin@amsterdamoptimization.com