Partial Dualization and the Optimal Power Flow Problem

Andrew M. Ross

February 8, 2000

1 Introduction

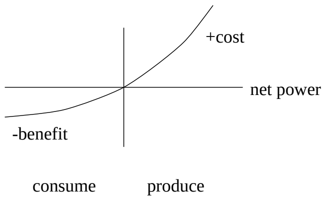
Around the world, governments are de-regulating their electric power industries in the hopes that competition among power producers will spur greater efficiencies and allow people to choose "green" power if they wish. One of the primary advances that allows this de-regulation to work is the quick solution of non-linear programs (NLPs) that describe the power flow in the electric power network. These NLPs help decide where to generate power so that the demand is met at a minimal social cost. This is called an "optimal power flow " (OPF) problem. They also must take into account constraints on the physical transmission equipment[1]. When we solve the NLPs, we get information on the cost of electricity at each location, and the monetary effect of not having enough capacity on some power lines (called "congestion").

We would like to solve the OPF for an entire nation, since that will lead to the solution with lowest social cost. In many cases, though, a country's power system is divided into administrative regions, and so a nationwide optimization is not possible. Even if the administrative barriers fall, though, some power systems are too large for solution by a single computer. Then we must use parallel computers, which is much like solving the problem in many regions. The basic idea is to let each region (or each computer in a parallel computer) solve its own OPF, while taking into account the constraints that the physical equipment in other areas will impose on it. However, we cannot store a complete description of the other areas; that's the reason we broke the problem into regions in the first place. The most common tactic to solve this problem is to put a price on using congested lines. This technique is alternately called Lagrangean relaxation or partial dualization. It is proposed by various papers, including Cadwalader et al. [1]. It is a reliable technique when the problem is convex, but can experience problems otherwise. In this paper, we explore a case where this approach breaks down.

[1] More advanced NLPs also avoid overloads in the case of equipment failures. This paper does not deal vwith that problem.

2 Power Flow Models

We will use the power flow model from Wu et al. [3]. We have n nodes, and at each one we set a phase angle θ. The flow between node i and node j is $Y_{ij}\sin(\theta_i - \theta_j)$, where Y_{ij} is called the "admittance" of the line (if there is no (i,j) line we set $Y_{ij} = 0$). The quantities q_i keep track of the net power flow at each node. If $q_i < 0$ then node i is a net consumer of power; otherwise it is a net generator of power. Each node has a cost curve $C_i(q_i)$ that is convex and increasing, as in this figure:



Our objective is to minimize the total cost. Each power line has a limited capacity for flow M_{ij}, due to thermal concerns (too much power would make it too hot). All power lines are symmetric. Our basic formulation is

$$\begin{array}{lrl}
\underset{\vec{q},\vec{\theta}}{\text{minimize}} & & \sum_i C_i(q_i) \\
\text{subject to} & & \\
\text{node flow: } \forall i & q_i & = \sum_j Y_{ij}\sin(\theta_i - \theta_j) \\
\text{thermal: } \forall i,j & Y_{ij}\sin(\theta_i - \theta_j) & \leq M_{ij}
\end{array}$$

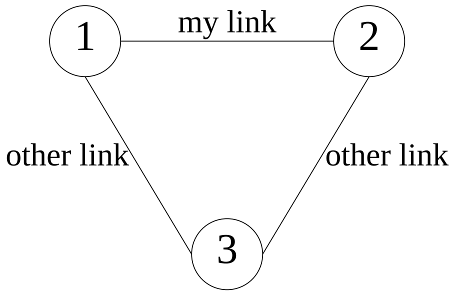
This model ignores reactive power and resistive line losses, but it captures the essence of the planning problem. Because only the differences between phases are used, there is an extra degree of freedom, so we can choose one node's phase and set it equal to zero. We will find it more convenient to substitute out the q_i variables. We get the formulation AC1

$$\begin{array}{lll}
\text{minimize} & \sum_i C_i\left(\sum_j Y_{ij}\sin(\theta_i - \theta_j)\right) & \\
\text{subject to} & & \\
\text{thermal: } \forall i,j & Y_{ij}\sin(\theta_i - \theta_j) & \leq M_{ij}
\end{array} \tag{1}$$

Denote its optimal solution to this problem $\vec{\theta}_1$, with Lagrange multipliers $\vec{\lambda}_1 \leq \vec{0}$. Even though the C_i are convex in their input q_i, here we see that the objective function is not globally convex in $\vec{\theta}$. Among other things, it has period 2π in each θ_i. However, it is locally convex near the feasible region, as pointed out in [2].

3 Feasible Region

To visualize the feasible region for the power flow problem in the phase variables (θ_i), we will start with a triangular network:



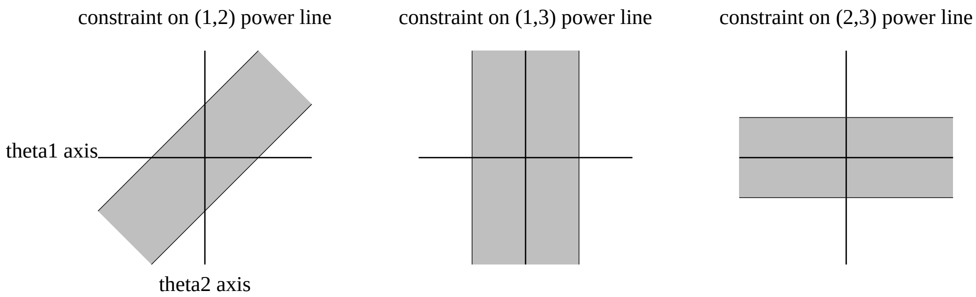
As mentioned above, we can set one of the phase angles to zero, so we will make $\theta_3 = 0$. Our thermal constraints are now as follows:

$$\begin{aligned} Y_{12}\sin(\theta_1 - \theta_2) &\leq M_{12} \\ Y_{13}\sin(\theta_1 - 0) &\leq M_{13} \\ Y_{23}\sin(\theta_2 - 0) &\leq M_{23} \end{aligned}$$

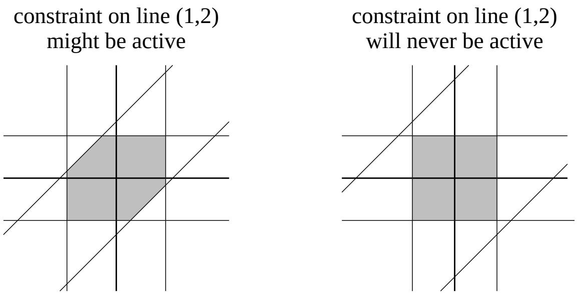
and their complements for flow in the opposite direction. We can linearize these (and their complements) in the neighborhood of $(0, 0)$:

$$\begin{aligned} |\theta_1 - \theta_2| &\leq \operatorname{asin}(M_{12}/Y_{12}) \\ |\theta_1 - 0| &\leq \operatorname{asin}(M_{13}/Y_{13}) \\ |\theta_2 - 0| &\leq \operatorname{asin}(M_{23}/Y_{23}) \end{aligned}$$

If we plot this in the (θ_1, θ_2) plane we see that each constraint results in a feasible strip. The first is diagonal (slope=1), the second is vertical, and the third is horizontal:



These combine to give a feasible region that will often be an irregular hexagon, but could be an irregular quadrilateral if one of the constraints is never binding.



For a network with n nodes, the feasible region would be a polyhedral solid in $n-1$ dimensions with faces that are at 45 and 90 degree angles to each other.

4 Partial Dualization

In order to solve this problem across multiple regions, we partition the set of tie lines into two sets, which we call Mine and Other. We explicitly monitor the thermal constraints on the lines that are Mine, while we apply partial dualization to the Other lines. Our formulation is now called AC2:

$$\min \sum_{\forall i} C_i \left(\sum_{\forall j} Y_{ij} \sin(\theta_i - \theta_j) \right) - \sum_{(i,j) \in \text{Other}} (\lambda_1)_{ij} \cdot (Y_{ij} \sin(\theta_i - \theta_j) - M_{ij})$$

$$\text{subject to} \tag{2}$$

$$\text{thermal:} \qquad \forall (i,j) \in \text{Mine}: \qquad Y_{ij} \sin(\theta_i - \theta_j) \leq M_{ij}$$

Call the optimal solution to this $\vec{\theta}_2$. We know that any solution to AC1 is a first-order stationary point of AC2. But, a minimum point of AC2 might not match the optimal point for AC1. The following things might happen, in order of most worrisome to least worrisome:

1. $\vec{\theta}_2$ might not be feasible for AC1.
2. $\vec{\theta}_2$ might be sub-optimal for AC1.
3. Maybe AC2 is unbounded
4. Maybe $\vec{\theta}_2$ is optimal for AC1, but is different than $\vec{\theta}_1$.

We will deal with these in reverse order. It seems unlikely that there will be multiple optimal solutions to AC1. Even if there were, we never really see $\vec{\theta}_1$, since we can't solve AC1 directly in the real world, so as far as we know, $\vec{\theta}_2$ is our only bet.

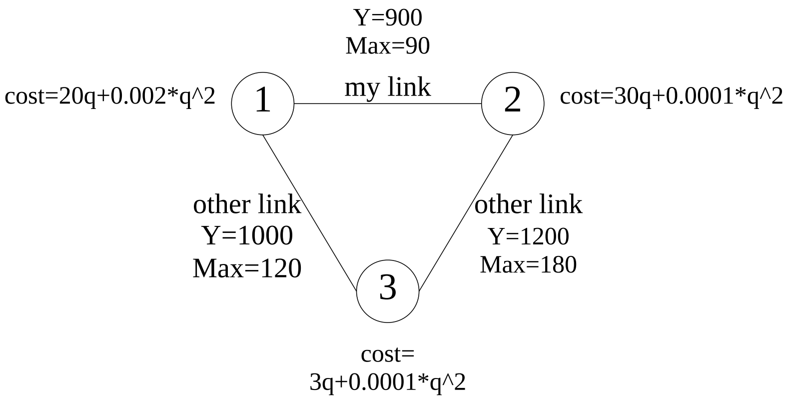
We can easily see that AC2 is bounded. To start, AC1 is bounded in any real situation. Then, AC2 just adds a linear combination of sines to the objective function, and the sine is a bounded function. So, we are not worried about AC2 being unbounded.

The other two are the crux of the matter here. In many other applications of partial dualization, we are able to argue this way: if the dualized constraints are convex, then the dualized objective function is still convex. We know that $\vec{\theta}_1$ is a stationary point for AC2, so it satisfies the first-order necessary conditions (FONC). Furthermore, if the problem was (strictly) convex, the second-order conditions would be satisfied, and we would have a global minimum of AC2 at $\vec{\theta}_1$, so $\vec{\theta}_2 = \vec{\theta}_1$.

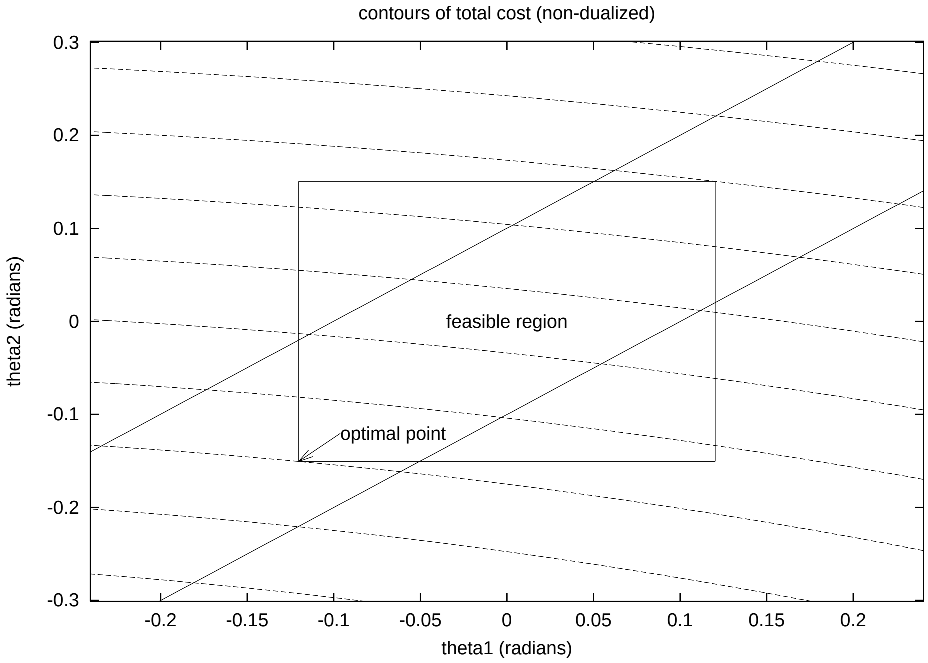
The problem in this case is that the dualized constraints are not convex: each one switches from concave to convex and back with period 2π. Thus, when we add them to the objective function, the solution to AC1 could be a saddle point instead of a minimum of the new objective function. Minimizing might then lead to a different point.

5 An Example

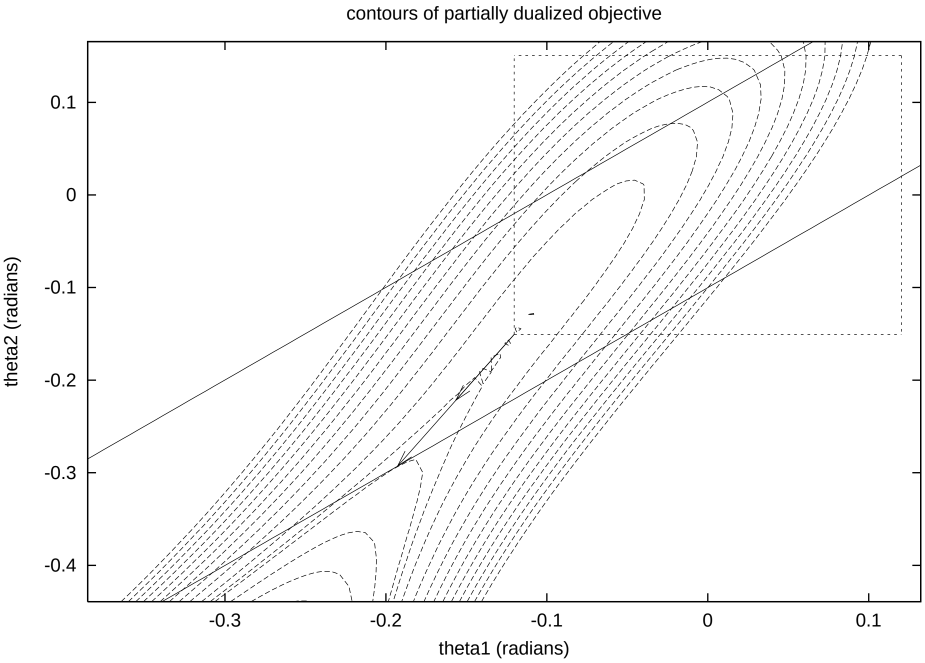
To illustrate the potential problems with the partial dualization, we present a small example. Consider a three node triangular network:



Using the same graphing system as above, we look at the contours of its cost function (without any dualization). The feasible region is the inner hexagonal region.

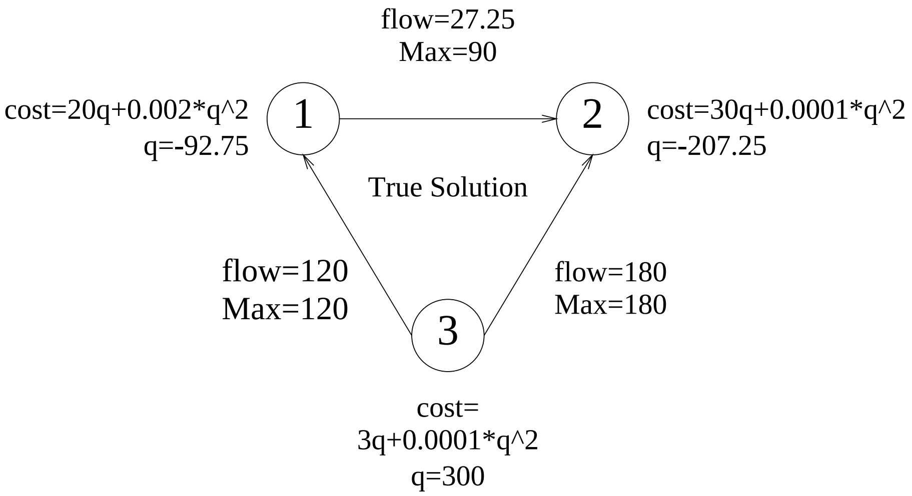


We take the Lagrange multipliers from the optimal point and dualize the constraints for the $(1,3)$ and $(2,3)$ lines. This removes the rectangular constraints from the previous picture, leaving the diagonal strip as the feasible region. The new optimal solution has moved outside the old rectangle, as shown by the arrow.

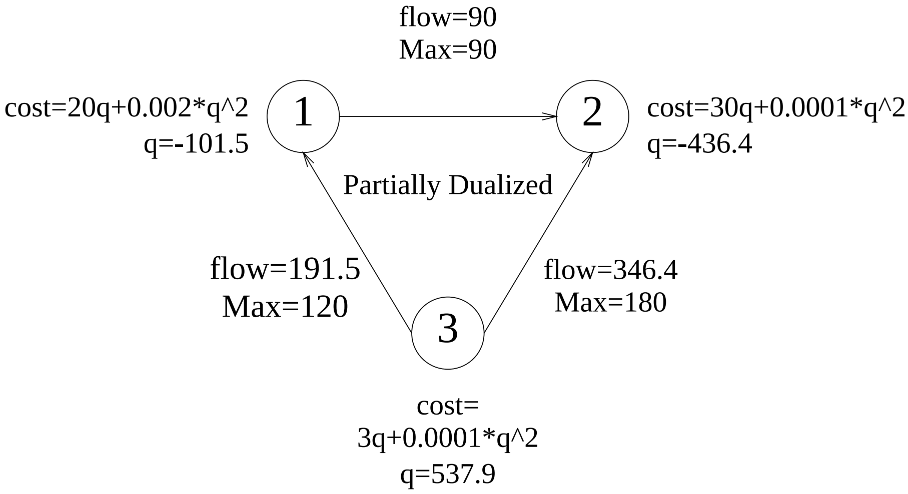


We can tell this solution is locally optimal because the contour line is tangent to the constraint boundary at that point. The solution to AC1 is an inflection point of the partially dualized objective; we can see this because the contour for the old solution has a cusp there.

The true solution to this problem is as follows:



The partially-dualized solution is



Lines $(1, 3)$ and $(2, 3)$ are over their thermal limits.

6 Method

This counter-example was found by random exploration of the set of possible parameters for admittance, thermal maxima, and cost parameters. We formulated the problem in AMPL, using MINOS as the solver. The script would set the parameters randomly, solve the original problem AC1, copy the dual values, then solve the partially dualized problem. If the resulting θ values were different, it would report the occurrence. This happened 60 times in 10,000 trials. Of those, about half were shifts by π in each variable, which would not happen in real power systems. Many of the others saw some of the flows reverse direction between the original and partially dualized problem.

Once we found a suitable result from the random trials, we rounded the values. This is because giving a counter-example to five significant figures is not convincing; we need to see that it is fairly robust to perturbation. Also, it makes displaying it easier.

The random values used for the trials were distributed as follows:

linear cost $\sim$ Exponential with mean 10

quadratic cost $\sim$ Exponential with mean $1/1000$

admittance $\sim$ Erlang-25 with mean 1000

max flow $\sim$ Uniform between 0 and 15% of the admittance on each line.

These mean values were roughly chosen to match sample values seen in the literature. Since we used a parabolic function for the cost, we had to make sure it was monotonic in the range of feasible flows. This requires that the minimum occurs sufficiently far to the left of the feasible region. Otherwise, consuming lots of electricity would start to generate money.

References

[1] Michael D. Cadwalader, Scott M. Harvey, Susan L. Pope, and William W. Hogan. Market coordination of transmission loading relief across multiple regions. Unpublished draft, Dec 1 1998.

[2] Hung-po Chao and Stephen Peck. A market mechanism for electric power transmission. *Journal of Regulatory Economics*, 10(1):25–59, Jul 1996.

[3] Felix Wu, Pravin Varaiya, Pablo Spiller, and Shmuel Oren. Folk theorems on transmission access: Proofs and counterexamples. *Journal of Regulatory Economics*, 10(1):5–23, Jul 1996.

A AMPL Model

```
# AC real-current approximation
# from "Folk Theorems" pages 9 (formula 5)
# with fixed Lagrange multiplier variables
# and optional partial dualization

set Nodes ordered ;
set Links within {Nodes,Nodes} ;
set MyLinks within Links ; # the links I monitor
set OtherLinks := Links diff MyLinks ;

var q{Nodes}; # net power flow at each node, positive=generate, negative=consume
var theta{Nodes}; # phase at each node

param ratio default 0.5;
param reference_node symbolic in Nodes; # this one will have theta=0
param c1{Nodes} default
        Exponential()*10;        #    linear cost per megawatt at each node
param c2{Nodes} default
        Exponential()/1000;        # quadratic cost per megawatt at each node
param Y{Links} >= 0 default
        Gamma(25)*40 ;                # admittance on each line
param Max{(i,j) in Links} default
        ratio*Y[i,j]*Uniform01(); # maximum flow on each line due to thermal constraints
param given_thermal_dual {Links} default 0;

# these are to match other outputs in the data files
# so we don't get an error when we read them in.
param parab_min {Nodes};
param old_theta {Nodes};
param old_q     {Nodes};
```

```
param oltb      {Links};
param mltb      {Links};
param olts      {Links};

var qij{(i,j) in Links}=Y[i,j] * sin(theta[i] - theta[j]);
 #flows on each line; these are just for convenience


var primal_cost =
        sum {i in Nodes}
                (c1[i] * sum {(i,j) in Links}
                Y[i,j] * sin(theta[i] - theta[j])+
                 c2[i] * (sum {(i,j) in Links}
                Y[i,j] * sin(theta[i] - theta[j])) * (sum {(i,j) in Links}
                Y[i,j] * sin(theta[i] - theta[j])) )
;
var dual_cost =
        sum {(i,j) in OtherLinks}
                given_thermal_dual[i,j] * (
        Y[i,j] * sin(theta[i] - theta[j]) - Max[i,j]
        )
;
minimize theta_cost: primal_cost             ;
minimize total_cost: primal_cost - dual_cost ;
objective theta_cost;

subject to reference_node_constraint:
        theta[reference_node] = 0;

subject to node_flow{i in Nodes}:
        0 = q[i]
        -
        sum {(i,j) in Links}
                Y[i,j] * sin(theta[i] - theta[j])
        ;

#subject to line_thermal{(i,j) in Links }:
subject to my_line_thermal{(i,j) in MyLinks }:
        Y[i,j] * sin(theta[i] - theta[j])
        <=
        Max[i,j]
        ;
subject to ot_line_thermal{(i,j) in OtherLinks }:
        Y[i,j] * sin(theta[i] - theta[j])
        <=
        Max[i,j]
;

param data_name symbolic ;
```

B Trial loop

```
model ac13rnd.mod;
data  ac3rnd.dat;

# these appear in the output
param parab_min {Nodes} default 0;
param old_theta {Nodes} default 0;
param old_q {Nodes} default 0;

#these are more temporary
param new_theta {Nodes} default 0;
param new_q {Nodes} default 0;
param old_cost ;
param difference;
param all_line_feas ;
param costs_increase ;

let ratio := 0.15;
display ratio >> ac13rnd.out ;

for {loopindex in 1..10000} {
        # doing a "reset" on these data will re-randomize them
        reset data c1,c2,Y,Max ;
        # make lines symmetric
        let {(i,j) in Links }   Y[i,j] :=   Y[j,i];
        let {(i,j) in Links } Max[i,j] := Max[j,i];
#        # set initial values (to avoid influence from previous loops)
        let {i in Nodes} theta[i] := 0;
        let {i in Nodes} q[i] := 0;
        # clear out the stuff from last time
        reset given_thermal_dual ;
        option dual_initial_guesses 0;
        # solve the non-dualized problem
        objective theta_cost;
        solve;
        # save the results
        let {i in Nodes} old_theta[i] := theta[i];
        let {i in Nodes} old_q[i] := q[i];
        let old_cost := total_cost;
        # set up the partially dualized problem
        let {(i,j) in OtherLinks}        given_thermal_dual[i,j]:=
                                        ot_line_thermal[i,j].dual;
        drop ot_line_thermal;
#####################################################
        # we now solve the partially dualized problem
        # by starting elsewhere
        let { i in Nodes} theta[i] := -old_theta[i];
        option dual_initial_guesses 0;
        objective total_cost;
```

```
        solve;
        let {i in Nodes} new_theta[i] := theta[i];
        let {i in Nodes} new_q[i] := q[i];

        let difference := sum {i in Nodes} abs(old_theta[i] - theta[i]);
#        print "#difference = ", difference >> ac13rnd.out;
        if difference > 0.00001
        then {
                print "#Opposite loopindex = ",loopindex >> ac13rnd.out ;
                let {i in Nodes} parab_min[i] := -c1[i]/(2*c2[i]);
                # display the problem data first
                display Y,Max,given_thermal_dual         >> ac13rnd.out;
                display c1, c2                                 >> ac13rnd.out;
                # now display the solutions
                display parab_min, old_q, q, old_theta, theta >> ac13rnd.out;
                restore ot_line_thermal;
                display ot_line_thermal.body, my_line_thermal.body,
                        ot_line_thermal.slack >> ac13rnd.out;
                let all_line_feas := 1;
                for{ (i,j) in OtherLinks } {
                        if ot_line_thermal.slack[i,j] < 0
                        then {
                                let all_line_feas := 0;
                        }
                }
                let costs_increase := 1;
                for{ i in Nodes } {
                        if parab_min[i] > old_q[i]
                        then {
                                let costs_increase := 0;
                        }
                        if parab_min[i] > q[i]
                        then {
                                let costs_increase := 0;
                        }
                }
                print "# all_line_feas = ", all_line_feas >> ac13rnd.out ;
                print "# costs_increase = ", costs_increase >> ac13rnd.out ;
                print "#old obj ",old_cost, " new obj ",total_cost >> ac13rnd.out ;
                print "#old cost ",old_cost, " new cost ", theta_cost >> ac13rnd.out;
        } else {
        };
#####################################################
        # we now solve the partially dualized problem
        # start in the original spot
        let { i in Nodes} theta[i] :=  old_theta[i];
        option dual_initial_guesses 1;
        solve;
        let {i in Nodes} new_theta[i] := theta[i];
        let {i in Nodes} new_q[i] := q[i];
```

```
        if difference > 0.00001
        then {
                print "#trying original starting point" >> ac13rnd.out;
        } else {
        };
        let difference := sum {i in Nodes} abs(old_theta[i] - theta[i]);
#        print "#difference = ", difference >> ac13rnd.out;
        if difference > 0.00001
        then {
                print "#Orig loopindex = ",loopindex >> ac13rnd.out ;
                let {i in Nodes} parab_min[i] := -c1[i]/(2*c2[i]);
                # display the problem data first
                display Y,Max,given_thermal_dual         >> ac13rnd.out;
                display c1, c2                                 >> ac13rnd.out;
                # now display the solutions
                display parab_min, old_q, q, old_theta, theta >> ac13rnd.out;
                restore ot_line_thermal;
                display ot_line_thermal.body, my_line_thermal.body,
                        ot_line_thermal.slack >> ac13rnd.out;
                let all_line_feas := 1;
                for{ (i,j) in OtherLinks } {
                        if ot_line_thermal.slack[i,j] < 0
                        then {
                                let all_line_feas := 0;
                        }
                }
                let costs_increase := 1;
                for{ i in Nodes } {
                        if parab_min[i] > old_q[i]
                        then {
                                let costs_increase := 0;
                        }
                        if parab_min[i] > q[i]
                        then {
                                let costs_increase := 0;
                        }
                }
                #display old_theta, theta >> ac13rnd.out;
                print "# all_line_feas = ", all_line_feas >> ac13rnd.out ;
                print "# costs_increase = ", costs_increase >> ac13rnd.out ;
                print "#old obj ",old_cost, " new obj ",total_cost >> ac13rnd.out ;
                print "#old cost ",old_cost, " new cost ", theta_cost >> ac13rnd.out;
        } else {
        };

        reset data given_thermal_dual;
        restore ot_line_thermal;
}
```