New 2D MCO navigation method for Astroid

David Craft

February 24, 2010

This report describes a method to navigate a high-dimensional pareto surface. The method we have been using has sliders whose positions map directly to the location in objective space. One issue with this current method (call it ObjectiveSlider method) is that if trying to stay exactly on the Pareto surface, you can get stuck on a part of the surface from which a user request to move in a certain direction is not possible, even though there exists portions of the surface in that direction. Another shortcoming of the ObjectiveSlider method is that the user is not able to visualize the tradeoffs in any static way.

The new method described here is based on the display of a 2D cut of the Pareto surface. The user selects which two objectives to view a 2D tradeoff curve of, and all of the other objectives are free, or upper bounded by some value specified. The user then navigates by either 1) moving along the 2D tradeoff curve or 2) changing the bounds of one of the other objectives, which alters the 2D tradeoff curve when those constraints affect the currently viewed tradeoff.

Let P be the Pareto surface points (i.e. the plan database), stored as a matrix in our standard way, with rows the individual plans and columns the objective values.

The new piece we need for this method is a way to generate points along the tradeoff curve for 2 chosen objectives. To do this, we adopt the method of A. Messac: "Normal Constraint Method with Guarantee of Even Representation of Complete Pareto Frontier."

We assume that P is ordered so that the anchor plans, in order, are the first rows of the matrix. We will further assume that P is normalized so the objectives all have values between 0 and 1. All calculations will be done in this normalized setting, but when the 2D plot is finally presented to the user, and wherever constraint values from the sliders are available for read-out, these should be in non-normalized objective values.

Let x and y denote the two dimensions selected to view the tradeoff curve of (e.g. $x = 7$ and $y = 3$ would be a possibility for say an 8D tradeoff). Let C denote the vector of constraints the user has selected. This is the current position of the sliders, and can include the objectives being traded off (in which case it is just the extent of the 2D curve that will be affected).

We start by finding the anchor plans for the 2D tradeoff selected. This is simple: find the convex combination of plans in the database that minimizes the objective in question, subject to constraints C:

$$\begin{aligned} \text{minimize} \quad & (s'P)_j \\ \text{subject to} \quad & (s'P)_j \leq C_j, \quad \forall j \\ & \textstyle\sum_j s_j = 1 \\ & s \geq 0. \end{aligned} \tag{1}$$

s is the convex combination vector, and j is the index of the objective to minimize (will run for $j = x$ and y).

With those two anchors in hand, compute a set of points, say 10, evenly spaced between them. These are points in the 2D tradeoff space (we will call these points $g_1,\ g_2, \ldots, g_k, \ldots$, Messac calls them p_k but that would get confusing here), see Figure 1. From each of these points (well, the ones in the middle) we will project down onto the 2D tradeoff surface in the direction perpendicular to the line connecting the two anchor points.

To do this, we form the vector from anchor point A_x to anchor point A_y, and call this v:

$$v = A_y - A_x.$$

v has dimension 1×2.

Let P_{xy} be the two user selected columns of the Pareto surface, i.e. if there are m plans in the Pareto database, P_{xy} is an $m \times 2$ matrix. Let g_k be the point we are running. The following optimization yields the result:

$$\begin{aligned} \text{minimize} \quad & (s'P)_y \\ \text{subject to} \quad & (s'P)_j \leq C_j, \quad \forall j \\ & (vP_{xy})'s = vg_k' \\ & \textstyle\sum_j s_j = 1 \\ & s \geq 0. \end{aligned} \tag{2}$$

The constraint $(vP_{xy})'s = vg_k'$ enforces that the solution is on the line perpendicular to the v and passing through g_k. (Writing the constraint as $v(P_{xy}'s - g_k') = 0$ makes this easier to understand).

That's it for generating points along the reduced 2D tradeoff dimensions. Whenever the user changes a bound, those optimizations are re-run. Now for a bunch of smaller points.

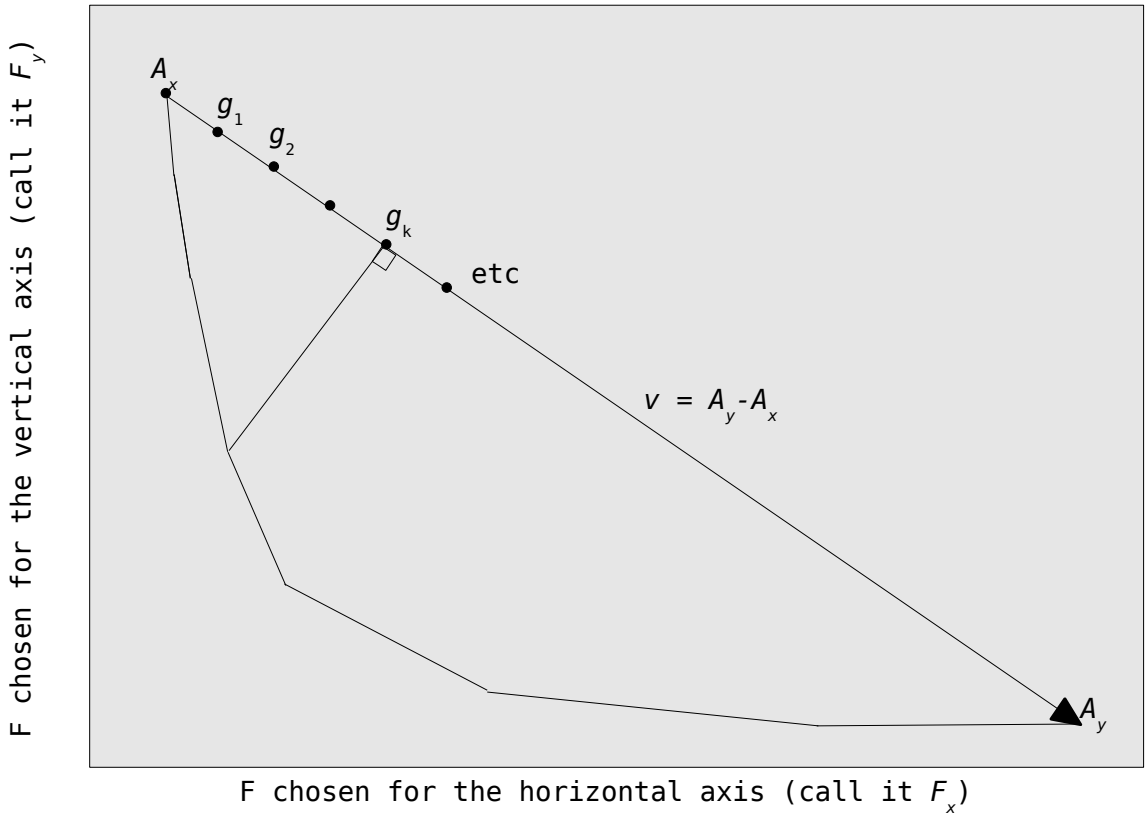

Figure 1: Diagram showing key components of computing points of a 2D tradeoff curve.

1: The result of running Formulation 2 for all the points g_k is a set of convex combination vectors s_k. One could use these convex combination vectors to combine the underlying dose cubes and compute the actual objective values for that plan, which will be at least as good as the linear interpolations otherwise used, and use those to plot tradeoff curve. If fast enough, one could also linearly interpolate between the points found, compute the full dose cube, and get a potentially smoother 2D tradeoff curve.

2: Moving around on the 2D curve is just a matter of track which two points you are in between and updating the convex combination used.

3: In order to display to the user the information about how moving a constraint will affect the 2D curve, we use duality information obtained from the two above linear programs. Duality information is output from any decent LP solver, including the one we are using. The basic idea is, we can get an objective sensitivity value for any of the constraints. In particular, for the constraints $(s'P)_j \leq C_j$, the dual values tell us how much the objective will change per change in C_j. Gathering

up (summing) this information for all of the g_k we get a sense of how much the 2D tradeoff curve will move. The duals from Formulation 1 have a different meaning geometrically in the 2D space then the duals from Formulation 2. I ended up scaling the anchor duals by 100 then taking the mean of all of them to get the global effect on the curve. I can look more into this, but it works okay as is.

4: When the user wants to change one or both of the two objectives currently being viewed, Thomas Bortfeld asked when I demonstrated the prototype system if we could preserve the current location. Turns out I don't think this can be reliably done in general (I will refrain from an attempt at what would no doubt be a tangled explanation of why). It would require altering constraints and even then is not guaranteed, and furthermore can result in a single point tradeoff "curve". So, when the user changes objectives, in my current implementation they move to a new spot in the global tradeoff space. We should offer a `<state save>` option. When the user changes 2D view, I think we just start in the middle of the new 2D surface, but I am of course open to other thoughts.

5: While exploring a chosen 2D tradeoff, I show all curves that have been visited thus far as grayed out curves, and show the active one in black. When the user changes objectives, I clean the slate.

6: Each point on the current 2D tradeoff curve represents a complete plan. Looking at all of those points and all of the objective values for each, we can find the maximum that any of the objectives takes for the current curve, and automatically lower the constraints to that level. This is what the `lower` button does, and I implemented it so I could quickly see when all the constraints would become influential. When `lower` gets clicked, I have the sliders all lower to a value epsilon = 0.001 lower than the maximum value so that when I re-run the optimizations, I get useful dual information (if I put them to exactly the value, the dual information is not reliable).

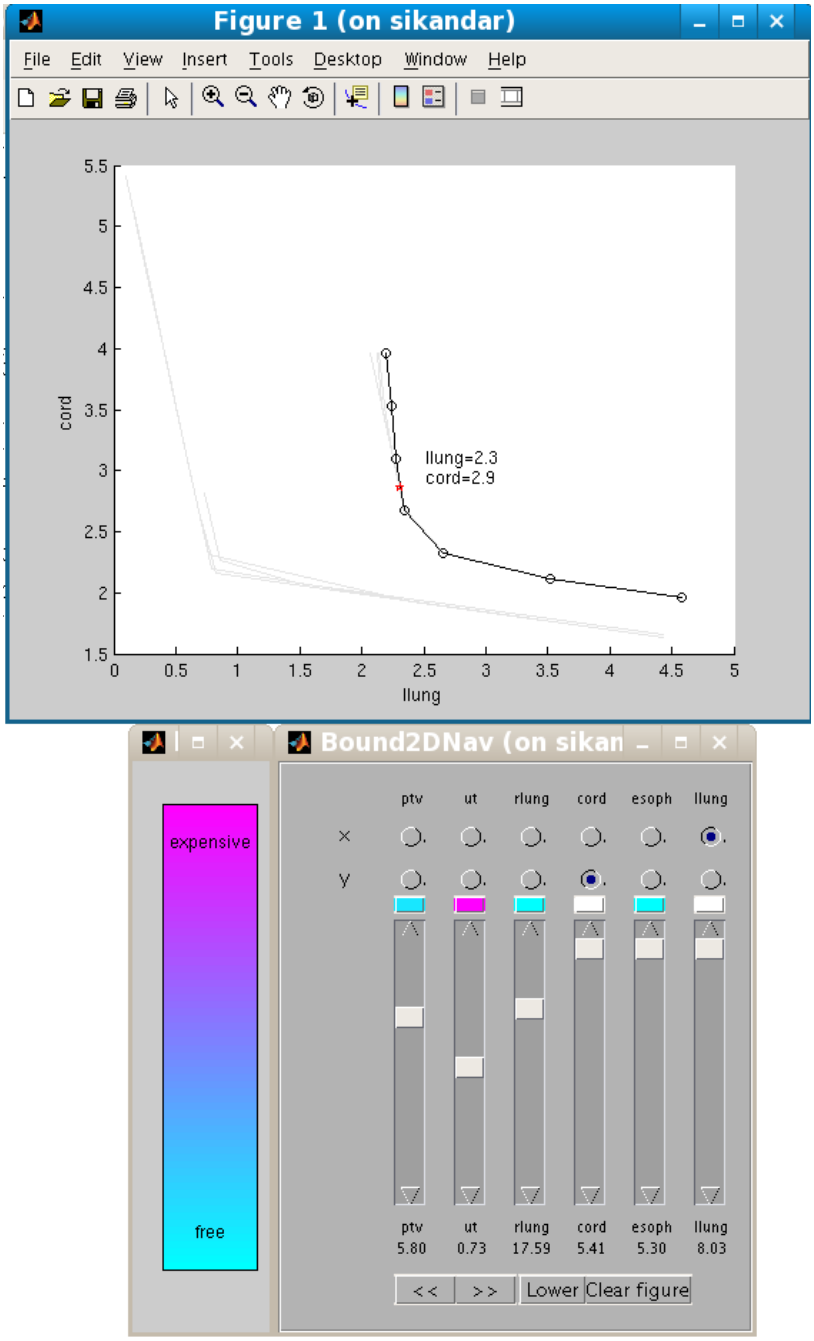

Figure 2: Prototype system.