From interactive visualization web-application to JavaScript library: a roadmap for parasol.js

Produced by:
Kasprzyk Research Group
Civil, Environmental and Architectural Engineering Dept.
University of Colorado Boulder
josephkasprzyk.wordpress.com

Last updated: May 8, 2018

Table of contents

Introduction

parasol is a new tool providing interactive visualization and statistical analysis methods for multiobjective optimization problems. *parasol* is built on D3.js, a JavaScript library for data-driven documents, and the associated parallel coordinates library, d3.parcoords.js. This document describes the current state of *parasol* as a web-application and provides a comprehensive outline of the mechanics of each feature, including an overview of all associated code and intended functionality.

As a web-application, *parasol* serves as an approachable and streamlined tool for decision makers to explore many-objective optimizations. However, as a Javascript library, *parasol* could provide the same functionality in a more versatile manner. Therefore, in addition to describing the functionality of *parasol*, the relevance of each feature to a library implementation and potential course for transition are additionally addressed throughout.

Features

Import data

Currently, only .csv file uploads are supported. In the future, this may extend to other file formats and database connections. Once the data has been uploaded, the uploader must be removed due to an issue with re-uploading data without reloading the webpage.

Note: If the user is building a personalized tool through the library, they will likely have a specific dataset to automatically import. In this case, functionality is already handled by `d3.csv`.

Current implementation details

The current csv uploader functions as a button and is the first thing the user sees when opening the app. They must click the button and select a csv file from their directory before the plots and grid can be produced. Once a file is selected, the app enters the `visualize` function which drives the interactive capabilities, and the uploader is removed.

```
var uploader = document.getElementById("uploader");
var reader = new FileReader();

reader.onload = function(e) {
  var contents = e.target.result;
  var data = d3.csv.parse(contents);

  // visualize data with default to 3 clusters
  visualize(data, n_objs = 3, k = 4);

  // remove uploader button
  uploader.parentNode.removeChild(uploader);
};

uploader.addEventListener("change", handleFiles, false);
```

```
function handleFiles() {
  var file = this.files[0];
  reader.readAsText(file);
};
```

Proposed library API

This feature is relevant for a two main purposes. It is convenient for those who would like to test their code on multiple files without having to edit the file path, and to those using *parasol* to develop a web application similar to what we have done here.

In the library, this should be a function that takes no argument, and simply appends the uploader button to the webpage.

```
parasol.import()
```

K-means clustering

K-means clustering partitions the imported data into into k clusters in which each datapoint belongs to the cluster with the nearest mean. In this way, clustering groups the solutions by a measure of statistical similarity. These clusters are denoted by corresponding categorical colors depending on the number of clusters specified. The default value is set to 3 clusters, with a maximum value of 6 clusters. This cap is due in part to the loss of relevance with too many clusters, and to the limited number of catagorical colors in the "Dark2" color pallete. Of course, the latter can be easily remedied should we choose to increase the maximum number of clusters.

Current implementation details

Clustering is implemented using the *ML* library.

```
var kmeans = ML.Clust.kmeans;
```

In a library-like functionality, the web-app developer currently specifies the number of clusters in the call to the `visualize` function.

```
visualize(data, n_objs = 3, k = 4);
```

The first task of the `visualize` function is to setup the clusters. We use the *Underscore* library to obtain data in the necessary array format for the *ML* library, preform clustering on the array version of the data, and append clusters id's to the original data.

```
// max clusters = 6
k = (k <= 6) ?  k : 3;

// coerce data to array of arrays for clustering
var clust_form = [];
data.forEach(function(d,i) { clust_form[i] = _.values(d) });

// preform default clustering
var km = kmeans(clust_form, k);
data.forEach(function(d,i) { d.cluster = km.clusters[i]; });
```

We then define a color pallete based on the number of specified clusters.

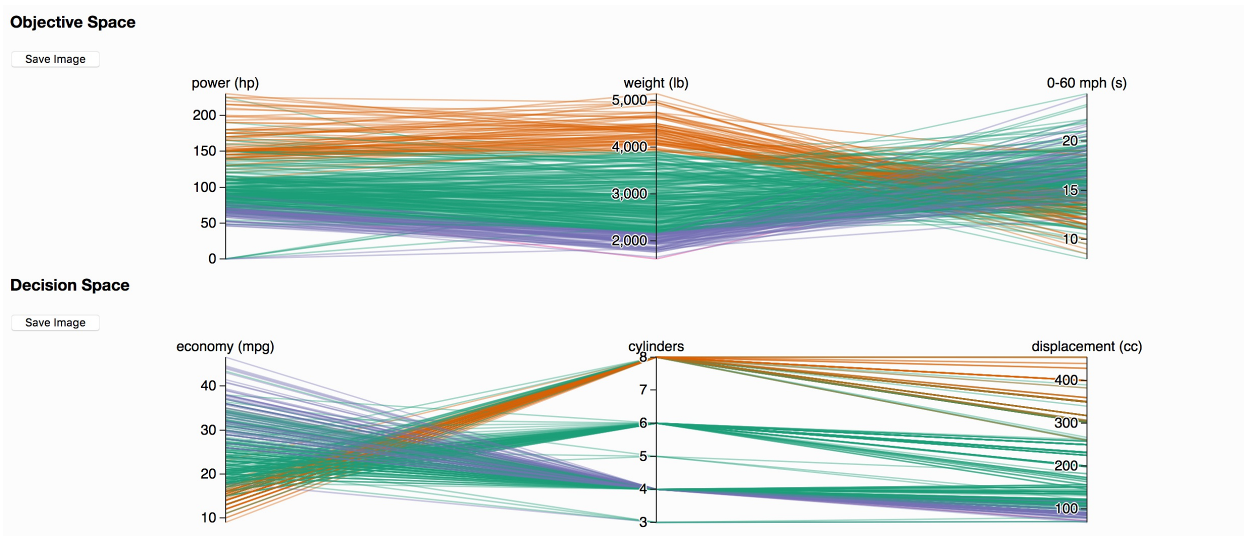
```
// choose default catagorical color scheme
var color_scheme = d3.scale.ordinal()
  .range(colorbrewer.Dark2[k])

var palette = function(d) {
  return color_scheme(d['cluster']);
};
```

This `palette` variable is then passed to the parallel coordinate plot variables, so that each observation will be colored by its cluster id. For example,

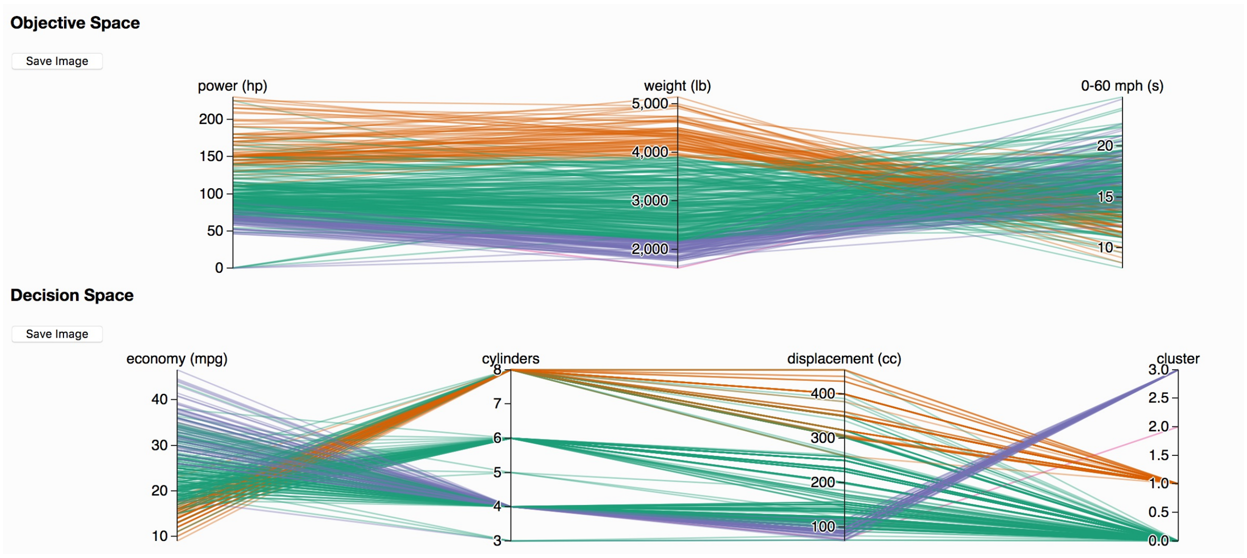
```
// objective space
var pc1 = d3.parcoords()("#plot01")
  .data(data)
  .hideAxis(decision_vars)
  .color(palette)
```

In addition to representing clusters by color, users may also choose to view clusters on an axis for interactive brushing. Because clusters id's are associated with each row of data, an axis is automatically created for them when plotting. By default, the cluster axis is currently hidden to avoid repetetive display of data. However, using the Hide and Show axes feature, they can easily be revealed. See the images below for reference.

Default: Clusters hidden on import.



After cluster axis selected in Hide and Show axes.



Proposed library API

Clustering functionality should be an extension of the main `visualize` function. The user will specify the following:

- k: number of clusters
- type: {"k-means", "spectral"} the type of clustering to be preformed
- group: {"decisions", "objectives", "both"} the group on which clusters will be decided

Defaults are listed below.

```
visualize(data, num_objectives)
  .cluster(k = 3, type="k-means", group="both")
```

Linked brushing

Coming soon.

Reset brushes

Coming soon.

Marking

Coming soon.

Clear markings

Coming soon.

Reorderable axes

The reorderable axes functionality is not a novel feature of this project. It is completely implemented in Kai Chang's *parallel-coordinates* library. Even still, it is included as a feature due to its significance for *parasol*. Whether implemented as a web-application or library, it is critical that the user have the ability to interact with axes of the produced plots in order to overcome the bias of the static relationships between variables. The implementation is simple, one need only extend the plotting variables as follows:

```
// objective space
var pc1 = d3.parcoords()("#plot01")
  .data(data)
  .hideAxis(decision_vars)
  .reorderable()
```

Hide and Show axis

Coming soon.

Set axes limits

Coming soon.

Keep and Remove selection

Coming soon.

Export selection as csv

Coming soon.

Explore selection

Coming soon.

Discussion

Coming soon.