CS533C Project Propasal

Visualization Tool for Analyzing SLS Runtime Behaviour

James Styles

jastyles@cs.ubc.ca
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada

1 Domain

Stochastic Local Searches (SLS) are a class of meta-heuristics for solving hard combinatorial optimization problems. Common examples of combinatorial problems are satisfiability(SAT), travelling salesman problem (TSP), job shop problem (JSP) and vehicle routing problem (VRP). Each of these problems is NP-complete. Local searches operate by choosing some initial positions within the search space (the set of all possible solutions) then choosing a sequence of moves to new positions based on the neighbourhood of their current position. See [8] for a thorough overview of SLS.

A substantial portion (as high as 90%) of the effort in development of SLS algorithms is in the analysis and tuning of their performance. The analysis task is made difficult due to the high dimensionality of the search space and the complexity in the SLS algorithms. There is however a large amount of information that can be extracted from an SLS algorithm as it is running. Though the exact information that is relevant is highly dependant on the particular SLS being considered, some examples are:

- information that can be gathered with minimal overhead added
  - solution quality at each iteration
  - time taken for each iteration
  - time taken to evaluate neighbourhoods for each iteration
  - time taken to evaluate heuristics for each iteration
  - neighbourhood(s) used for each iteration
  - details about the type of move taken
  - justification for changing neighbourhoods (ie for portfolio based searches, or transitions between intensification & diversification)
  - current best solution at each iteration
  - ...
- information that can be gathered with substantial overhead added
  - current set of solutions at each iterations (for population bases searches like GAs, EAs, ACOs)
  - current set of solutions contained within the neighbourhood at each interation
  - current solution at each iteration
  - ...

The instrumentation necessary to extract information from a local search can add substantial overhead. So you must be careful about any measurement of time taken to perform some task within the algorithm otherwise you could be recording information about the performance of the instrumentation instead of the solver.

On top of the information that can be gathered directly from the search there are additional pieces of information that can be derived such: as the distance between the solution at every iteration and some specific solution (usually the final best solution or some elite solution representing local minima), the aggregate performance of the search over a set of multiple runs on the same problem, or the aggregate performance of different SLS methods on a particular instance of a problem.

2 Data Set and Tasks

The data set used for this project will be generated by logging runtime information from tuned versions of Kjeld Helsgaun's LKH implementation [1] on the TSPLib instances [4]. The tuning has been performed automatically using ParamILS [2]. Several attempts at tuning have been performed each resulting in a different set of parameters to Kjeld's implementation resulting in a tsp solvers that behave differently but provide the exact same types of runtime information. The differences between the solver performance is quite substantial for certain problem instances with some solvers taking 3-4 times as long to find solutions as others.

The tasks this project will be designed at aiding will be (1) comparing the aggregate performance of the solvers to eachother across the various problem instances, and (2) attempting to provide explanations any notable performance differences (ie, showing trade-off between neighbourhood quality and time to compute neighbourhoods for various problem instances, or comparing the quality of diversification phases amongst solvers).

3 Personal Expertise

I work for a company (www.actenum.com) that focuses on developing applications to find solutions to large optimization problems. The problems we consider are very large scheduling or resource allocation problems with numerous side constraints which are specific to a client. To aid in the development of our models and solvers for these problems we developed a set of tools to visualize their runtime characteristics. I was involved in both the development and use of these tools. Unfortunately I can't use any of these tools or the data sets generated for this project.

4 Proposed Solution

The core idea behind the visualization will be the interactive creation of displays. A display is defined by a set of data sources and set of mappings from those data sources to some visual encoding within the display. The compatible visual encodings is determined by the type of display. There are two key parts to creating/editing displays: managing the data sources and managing the mapping to visual encodings. Multiple displays can be created and aligned in a grid to create a small multiples views.

4.1 Managing Data Source

The visualization will manage a catalogue of data sources that can be used within the visualizations. Each item in the catalogue is associates with a set of tags $t_{0...n}$; each tag is a (tag name: tag value) pair. The user can provide an ordering to tag names to organize every item into a tree hierarchy. The catalogue is then displayed using a tree table, see figure 1.

Time series are added to a display by selecting them from the global data catalogue. Each display then manages its own local catalogue (which is essentially a filtered view into the global one). Each display can have its own ordering for tags. Derived values will appear as items in the local catalogues for a display. Navigation and selection in displays can be linked.

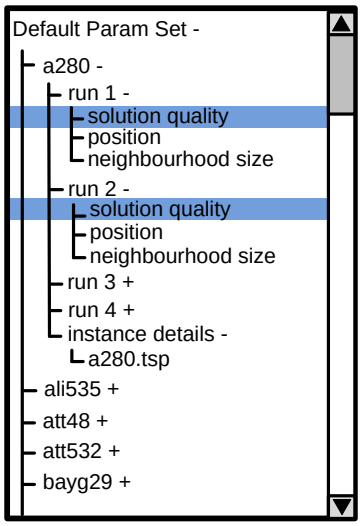

Figure 1: A mockup of the treetable view for data catalogues

4.2 Managing Displays

Each display has 2 modes: a view mode, see figure 2 and an edit mode, see figure 3 and 4. Entering a display's edit mode brings up a list of components that are part of the display and a menu for creating new components and derived values.

The component list provides an overview of every component that has been created for this display whether it is currently visible or not. Each line contains the components name, a small preview of what the component looks like in the display and a toggle for determining whether it is currently visible. Reordering the list of components (done by dragging the row around) reorders the layering of the components. A component near the top of the list will be drawn on top of a component near the bottom. Selecting a component brings up a panel to showing the properties of the component (ie how it should be drawn, what data items its using, etc) which can be used to edit the component.

The types of components that can be added to the display depends on the kind of display. The first type that will be added is a basic time series display. It will allow adding line graphs, filled regions defined by the extremes of a set of time series, and text annotations. If time permits another display type will be added for drawing solutions to TSPs.

5 Use Scenario

A set of runs for two solvers, A & B, has just finished for some the pla33810 TSPLIB instances. Each solver has completed 10 runs and have been instrumented to provide (1) current solution at each iteration, (2) solution quality at each iteration, (3) annotations about

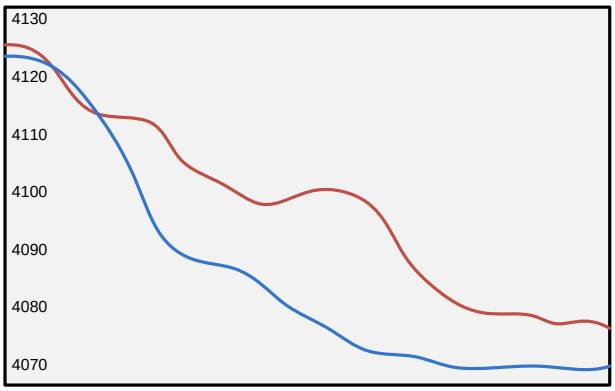

Figure 2: A mockup of the treetable view for data catalogues

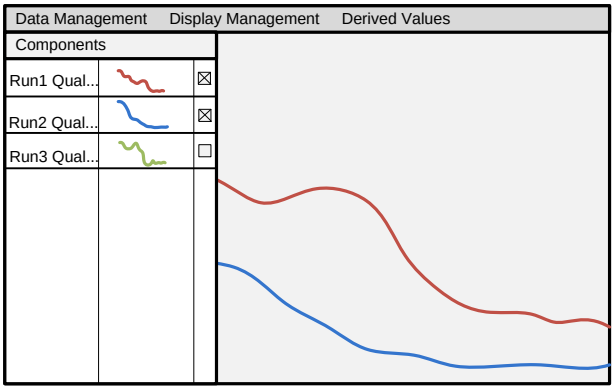

Figure 3: The view mode for a display

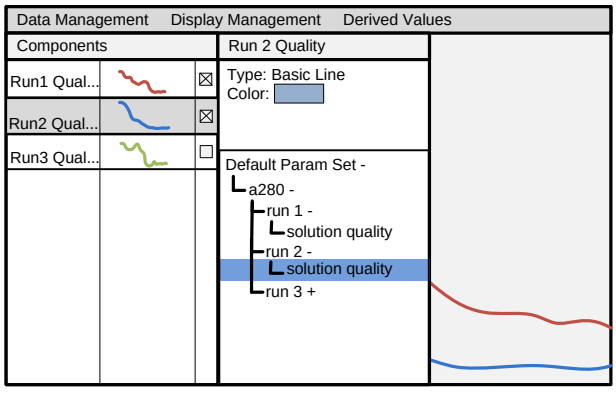

Figure 4: The edit mode for a display with a component of the display selected.

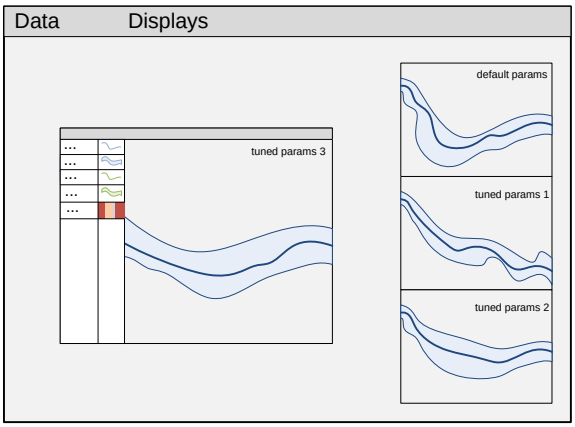

Figure 5: A mockup of the entire application

transitions between intensification and diversification.

The user creates a new display for solver A and enters its edit mode. The user opens up the data manager for the new display and adds the time series information for solver A's runs from the global catalogue. The user then creates a set of derived values containing the statistics for solution quality (min, max, avg) for all the runs and creates one component to draw the average as at the top layer as simple line and another component to draw the filled in region between the min & max values on the next lower layer. The user then creates a new display and repeats the previous steps but with solver B's runs.

With both displays side by side the user notices that solver A seems to stagnate much quicker than solver B. The user brings enters the edit mode in the display for solver A and creates a component to directly show the solution quality for one of the runs and another component to show the annotations for transitions to diversification phases and hides the old components showing min/max/average statistics. The user finds one of the transitions and selects (in the graph) the time point before one of the diversification phases. The user then constructs a new derived time series showing the similarity between the solution at each iteration and solution at the selected time point and then hides the component for that runs solution quality over time. With just the similarity score over time and the diversification annotations showing the user notices that the solution at the end of diversification is very similar to the solution at the right before diversification. This shows that the diversification is too weak causing the solver to be trapped around a single local minima which explains the early stagnation.

6 Implementation Tools

The visualization will be implemented in java using Java2D.

7 Milestones

- data set (logging format and instrumentation of [1])
- core components (data catalogue, basic zoomable display for time series)
- basic derived values (statistics, basic arithmetic, composition, convolution)
- TSP-specific derived values (similarity measures, etc)
- (if time permits) TSP solution display
- (if time permits) alternative time series displays (see [9], [5])

8 Previous Work

Previous work for time series visualization [5][9][3]. Previous work for analysis of SLS algorithms [10][6][7][12][11].

References

[1] `http://www.akira.ruc.dk/~keld/research/LKH/`.

[2] `http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/`.

[3] `http://www.cs.umd.edu/hcil/timesearcher/`.

[4] `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

[5] Lior Berry. Binx: Dynamic binning and scaling of time series, 2004.

[6] Steven Halim and Roland H.C. Yap. Designing and tuning sls through animation and graphics: An extended walk-through, 2007.

[7] Holger H. Hoos and Thomas Stutzle. Analyzing the runtime behaviour of iterated local search for the tsp, 1999.

[8] Holger H. Hoos and Thomas Stutzle. Stochastic local search: Foundations and applications, 2005.

[9] Robert Kincaid and Heidi Lam. Line graph explorer: Scalable display of line graphs using focus+context, 2006.

[10] Franco Mascia and Mauro Brunato. Techniques and tools for local search landscape visualization and analysis, 2009.

[11] Hartmut Pohlheim. Visualization of evolutionary algorithms - set of standard techniques and multidimensional visualization, 1999.

[12] Hoong Chuin Lau Steven Halim, Roland H.C. Yap. Viz: A visual analysis suite for explaining local search behavior, 2006.