Visualizing SLS Runtime Behaviour

James Styles*
Department of Computer Science
University of British Columbia

1 Introduction

1.1 Overview of SLS

Stochastic Local Searches (SLS) are a class of meta-heuristics for solving hard combinatorial optimization problems. Common examples of such problems are satisfiability (SAT), travelling salesman problem (TSP), job shop problem (JSP) and vehicle routing problem (VRP). The set of all possible solutions to one of these problems is called the search space. Each point (solution) in the search space is associated with some measure of fitness or objective.

The idea behind local search is that some initial point in the search space is chosen as a starting point. From this point, and all future ones, we define a neighbourhood. A neighbourhood for a point is the set of all solutions which can be constructed by modifying the current solution in some well defined way. This modification is typically very small; such as swapping a single assignment for SAT. A point from the neighbourhood is chosen to become the new solution. This process is repeated for some number of iterations until either a solution of sufficiently good quality is located, a time bound is met, or some measure of stagnation is reached.

The key parts of an SLS algorithm are then: (1) how to choose the initial position, (2) how to define the neighbourhood for a given point, (3) how to decide which point within a neighbourhood to choose, (4) when to decide that the search should terminate. For a concrete example we can construct a very simple SAT solver: (1) an initial solution is chosen by generating a random assignment for each variable, (2) the neighbourhood for each point contains every solution that can be obtained by changing a single variable assignment, (3) the neighbour with the fewest violated clauses is chosen, (4) the search terminates when no neighbours can be found with fewer violated clauses than the current solution. This is an example of a simple gradient decent (or hill climbing) search.

This simple solver for SAT should help to illustrate a key point about SLS. There is no guarantee that a SLS method will find an optimal solution. There isn't even a guarantee that an SLS method can find a feasible solution! This means that SLS algorithms are incomplete. So why use them? Well, they're fast. Very fast. And in practice a "good" solution found today is often more desirable than a perfect solution found next year.

1.2 The Task

A substantial portion of the effort in the development of SLS algorithms is spent analyzing and tuning their performance. This analysis task is made difficult due to the extremely high dimensionality of the search space which can thousands of variables and due to the complexity of the algorithms. There are two approaches that can be taken to reduce this effort: (1) provide/improve tools used for the analysis (2) create automatic methods for tuning. The focus of this project is the former.

There is a lot of information about an SLS algorithm that can be recorded as it explores a search space. This includes both general information such as:

- solution quality at each iteration
- solution at each iteration
- time taken for each iteration
- solutions contained in the neighbourhood at each iteration
- switches between intensification and diversification phases

and information very specific to a particular SLS approach, such as:

- tabu tenure / contents of tabu lists
- contents of no-good caches
- justifications for switching neighbourhoods (portfolio based searches)

The instrumentation necessary to extract this information from a local search can add substantial overhead and care must be taken to ensure any timing information recorded is describing the search and not the instrumentation.

There is also a great deal of information about a searches behaviour that can only be determined offline. This includes both information that is can be made available at runtime but is just inconvenient to record and information that is impossible to determine at runtime such as aggregate performance or similarity to elite solutions / local optima over time.

The goal of this project is to provide a framework to support the interactive visual analysis of this data. From personal experience this kind of analysis is often very opportunistic. It is unlikely that any single display could be useful for all aspects of an analysis task. This means that the framework must be flexible and the choice and layout of displays must be driven by the current needs of the user. There are three main components that must be considered in any potential solution: (1) a means of managing a collection of data sets, (2) a means of supporting the computation of derived values, (3) a means of visualizing the collected information.

The remainder of this report is as follows: section 2 discusses related work, section 3 discusses the overall interface for the tool, section 4 discusses the method for organizing data sets, section 5 discusses the facility for supporting derived values, section 6 discusses the time series display, section 7 discusses future work.

2 Related Work

[2] introduces a visualization for comparing the performance of solves on problems with multiple-objectives. The tool provided allows for the comparison of the overall performance of two algorithms where neither one dominates the other. The displays generated highlight areas on a pareto front which each algorithms outperforms the others. This provides a great high level view of the performance difference between two algorithms but can't be used by itself to analyze either. However these kinds of high level views are certainly important to provide initial insight and guidance for future analysis.

[1] introduces a set of visualizations for analyzing the runtime behaviour of an SLS algorithm based on comparisons between the current solution and a set of elite solutions using distance, fitness

*e-mail: jatyles@cs.ubc.ca

and recency as the basis of comparison. There method is particularly well suited for tabu based search strategies as the elite solutions can be the contents of the tabu list. They provide an example of using these displays to provide insight into how to improve the performance of a particular algorithm. This is a good example of a display for analysing the performance of an individual run of an algorithm.

[3] focuses on how to visualize the search space explored by an SLS algorithm. The main contribution is an embedding of high-dimensional search space into 2D height map. The resulting images are very similar to what is produced by in-spire's information landscapes. While this provides good insight into the extent of exploration performed it does not provide information on how it was explored.

[4] is the only piece of work I've been able to find which attempts to introduce a full suite of visualization tools to support analysing SLS behaviour. One of the main ideas behind the layout of their suite is that the ability of an individual to use multiple displays degrades as the information density in each display increases. To this end the focus of their suite is a display for analyzing the exploration of the search space. Similar to the previous paper this display focuses on a way of embedding a high dimensional space into a 2D space. They provide auxiliary displays for other metrics such as fitness over time, current solution, and fitness-distance correlations.

This seems like an excellent visualization suite and had it been open source I probably would have focused on extending it rather than building a system from scratch.

3 Overall Design

One of the main goals of this visualization tool is to be as flexible as possible. This is to encourage the opportunistic exploration and analysis of the information recorded from the SLS runs. This of course comes at a cost; increased flexibility means increased time taken by the user to generate displays. The alternative is a more rigid system with a prebuilt layout and set of supported tasks. While this has the potential to time investment it limits what can be done. I'm unclear as to the full implications of this trade-off and unfortunately have not had time to do proper user testing. I may eventually revisit this in favour of a more focused/rigid system such as time searcher or viz.

A secondary design goal that arose during the early development of this system results from the observation that dialogs can be jarring. This problem becomes even more prominent as the complexity of the dialogues increases. This has lead to the design goal of trying to use as few dialogues as possible. With the only true dialogue left being a colour picker for the displays.

The real implication of this design decision is the impact on the flow of data throughout the system. The idea is that every piece of data has a visual handle in the system. All data movement is done by moving dragging and dropping. This includes all data import, export and manipulation within the system. This has some nice advantages; for instance to import a set of data you just drag it from some file manager onto the application and similarly data export is handled by dragging from the application onto a file manager. However just like with the flexibility this comes with a price. There is substantially more mouse movement required to perform a task than would be required with dialogues. There is also the issue of ambiguity about what a particular action means. And just like with the flexibility goal I'm not sure whether this tradeoff is worth it yet.

Before starting into the individual components a comment about the implementation. The entire system is implemented in Java using the graphics2D and swing frameworks. I made the decision to forgo using other libraries to implement features of the system to have more control over how the system behaved. (Although the fact that I have an unfortunate tendency to enjoying reinventing the wheel probably contributed to this decision).

See figure 1 for for what the complete application looks like.

4 Data Management

4.1 Motivation

There's the immediate question of why is it important for this application to be concerned with any kind of data management? What it comes down to is the sheer volume of data that can be generated for analyzing SLS behaviour. To illustrate this I'll give an example of something I'm currently working on. I've been trying to tune Keld Helsgaun's implementation of LKH. This is the state of the art incomplete algorithm for solving TSP. As part of the tuning process I generated 10 promising variants on the default parameter set proposed by Keld. Each parameter set (including the default one) was evaluated by running it on a set of 112 instances from TSPLIB. Since this is a stochastic algorithm it is important to run it multiple times on each instance to get a better feel for the expect and worst case performance. To achieve this each instance was run 15 times per parameter set. This means that for a single recorded attribute there are 18480 data sets that will be involved in the analysis process.

When this many data sets can be involved data management now becomes issue. One thing to note is that while there may be several thousand data sets involved with an analysis process an individual task many only require a few dozen or even just a single data set. Furthermore the user knows which data sets they want to look at for a particular task. The goal for this component of the system is then to provide the means of quickly organizing and searching through the data sets so that interesting items can be quickly and efficiently grouped and selected.

4.2 Tags and Trees

In this application every data set is associated with a set of tags / annotations. Each tag is a ¡name, value¿ pair which describe some aspect of a data set. It is assumed that total ordering is available for tag values. For instance a certain run of a solver could be described using the: (1) the name of the solver, (2) the instance being solved, (3) the id of the run, (4) the measurement being recorded ,and (5) the computer the run was done on. There is no requirement that every item use the same set of tags. The key thing to note is that the user has created these tags. This means that they should all be meaningful and intuitive to the user.

We can now use these tags to organize the collection of data sets into a tree. Each level in the tree corresponds to a particular tag. The children of any node are sorted according to value using the total ordering which was assumed to be available. Since the tags are all meaningful to the user and they know which data sets they are interested in this solves the issue of locating data sets within the collection. However this doesn't solve the issue of grouping and selecting them.

When find a group of data sets to use we observe two extremes: all the desired data sets are very near each other in the tree, or all the desire data sets are spread throughout the tree. Whenever the latter case is encountered it would become a source of frustration to the user. The users' experience now depends entirely on how we decided to layout the tree. But this layout isn't arbitrary. Its determined entirely by the order in which we consider tags. The solution is then to allow the user to have explicit control over the ordering of the tags. This leaves one remaining question, will it be generally be possible for a user to find an ordering which makes their task easy? And how difficult will it be for them to find this order?

From my experience the answer to these questions are (1) Yes, and (2) Very easy. This comes back to the idea that the tags are all meaningful and intuitive ways of describing data sets to the user. For instance consider the set of 5 tags I gave as examples earlier. If my current task is to compare the aggregate performance for solver

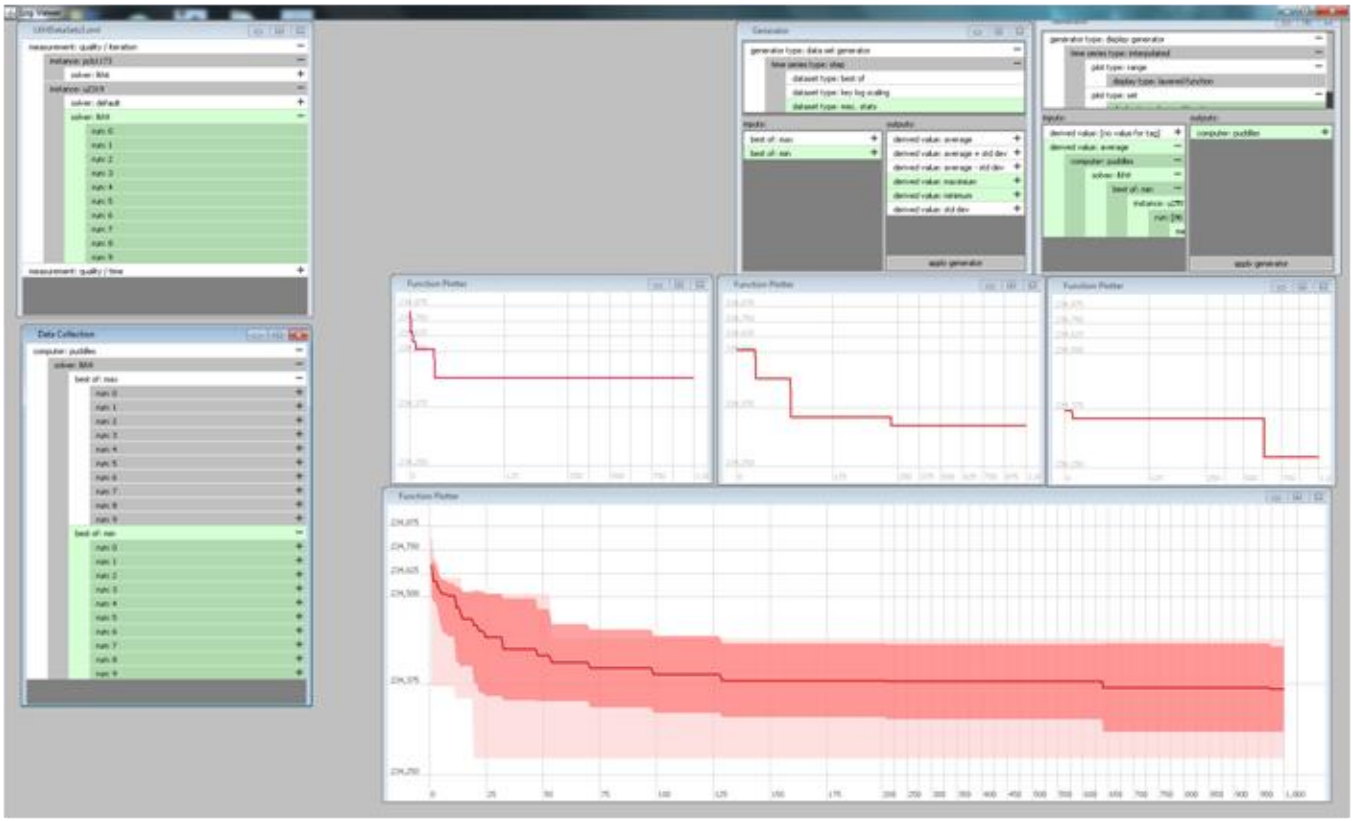

Figure 1: The complete log viewer application

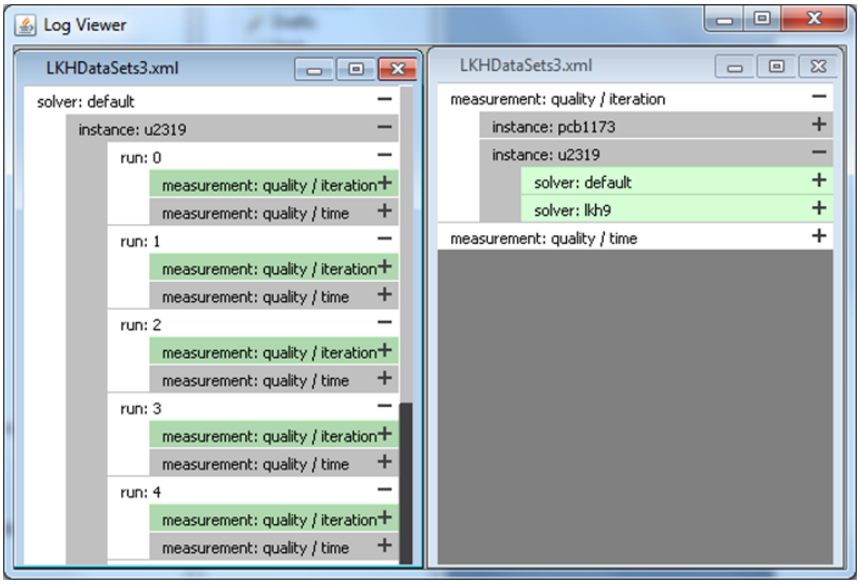

Figure 2: Reorering tags can have a dramatic effect on the locality of items within the tree. Both trees contain the same items and the selection, but use are using a different tag order.

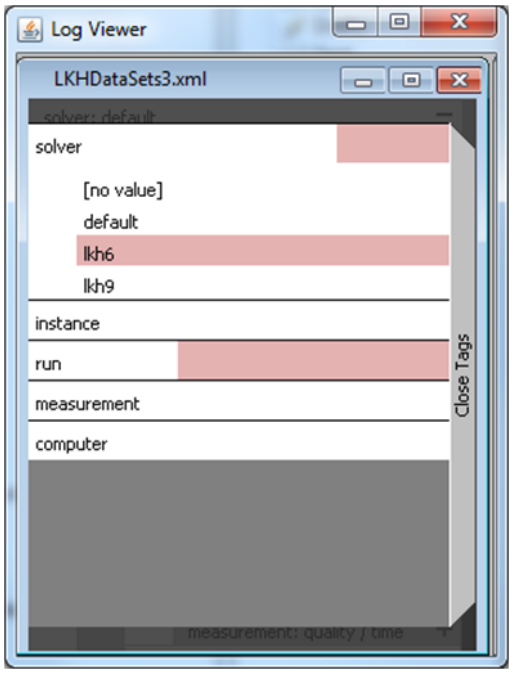

Figure 4: Filters can be applied to tag values to reduce the number of items displayed in the tag trees. Collapsed entries in the tag manager still give an indication of how many of their values are filtered out.

A and solver B on instance C the an ideal ordering for tags would be: (4) measurement type, (2) instance, (1) solver, (3) runs. I know only have to expand the tree to the solver layer for a single measurement type and a single instance. I contend that this can always be done for the types of tasks that are typical for analysing SLS algorithms. See figure 2 for a comparison of the original tag ordering and the new tag ordering used for this task.

4.3 Implementation Details

4.3.1 Tag Manager

While being able to reorder tags is then main focus of the tag tree idea the tag order doesn't need to be visible at all times. There are two separate tasks for a tag tree; searching through the tree, and editing tags. These tasks are interleaved, but never done at the same time. This means that both the tag tree and the display for editing tags (the tag manager) can occupy the same space. Switching between the two is done by extending a tab that appears when the user drags the mouse to the left edge of a tage tree.

4.3.2 Hidden Selections

One of the big advantages of using a tree to display the collection of data sets is allowing the use to collapse nodes to make navigation easier. But what happens when the user collapses the parent of a selected node? If all indication of a selection just disappears when a parent is collapsed then it would be very easy for the user to lose track of what items are current selected; especially when the collection contains thousands of items! Instead the solution chosen is to highlight a node if either it is selected, or it is collapsed and one of its descendants is selected. This allows the user to take full advantage of collapsing parts of the tree for navigation without losing track of currently selected items. See figure 3.

4.3.3 Tag Filters & Hidden Filters

On top of reordering tags it can be very beneficial to be able to filter out data sets based on tag values. This is done from within the tag

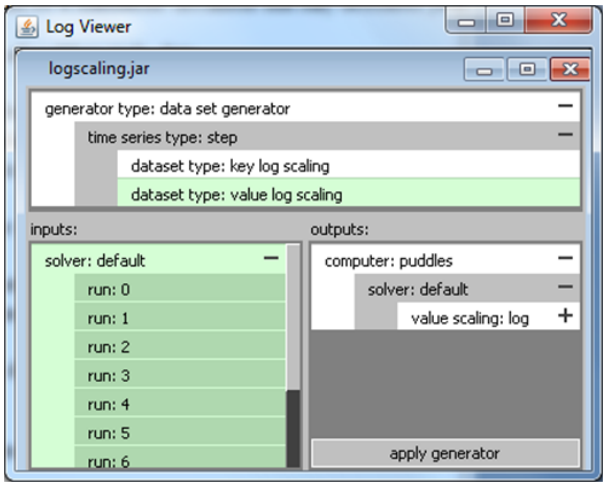

Figure 5: An example of a generator frame. Just like data sets, generators are annotated with a set of tags and organized using tag trees.

manager for a tree. An item in the list can be expanded to list all the values of a tag which are found in the current collection of data sets. Clicking on a value toggles it from being filtered in and out. These lists can become quite long so it's not desirable to leave them in this expanded state when reordering tags. However just like with selections in the trees collapsing a tag that has had a filter applied to it creates a problem.

To address this problem that bar for each tag will gradually fill up with red as more items become filtered out. The first item filtered out causes a disproportionately large portion of the bar to fill red. This guarantees that the presence of a filter is always clearly visible even if only one value out few hundred has been filtered out. See figure 4.

4.3.4 animation

This addition to the system is in response to some early use of the tag tree and tag manager components. Both these components are really just lists of bars. I started to notice a problem when interacting with either one once they surpassed some number of items. It became hard to follow the changes. This was particularly bad with the tag viewer. Animating the movements of these components makes the interaction substantially easier to follow.

5 Derived Values

As mentioned earlier the ability to compute derived values is very important for this tool for two reasons; (1) some values are not easy/convenient to gather during runtime but are relatively easy to determine afterwards, (2) some are impossible to determine until a search has completed. The other important aspect of derived values is that they are often very specific to a particular analysis task, solver, or problem type. This means that they really need to be done in an extensible way.

To achieve this goal of all derived values are handled with the concept of a generator. A generator simply takes as input a collection of data sets and returns a new set based on its input. Due to using this general interface none of implementation details of a specific derived value need to be hardcoded. Every single one is treated as a plugin for the tool.

5.1 Implementation Details: Fun with Class Loaders

As mentioned earlier the ability to compute derived values is very important for this tool for two reasons; (1) some values are not easy/convenient to gather during runtime but are relatively easy to

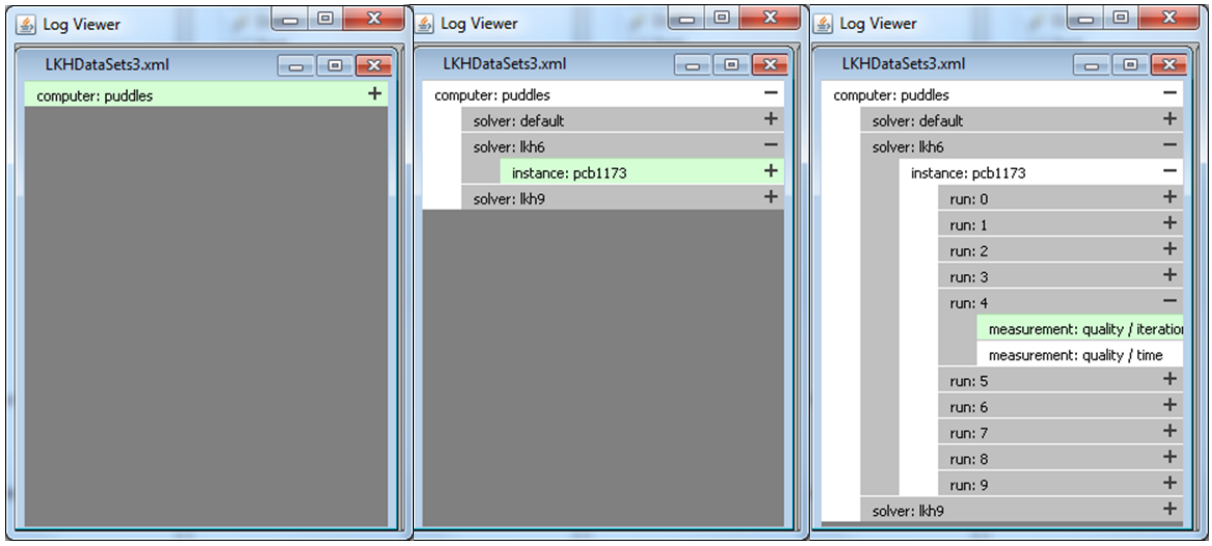

Figure 3: In all three images the same single leaf node is selected. The left and middle images show the visual queue left on the parents when they're collapsed.

determine afterwards, (2) some are impossible to determine until a search has completed. The other important aspect of derived values is that they are often very specific to a particular analysis task, solver, or problem type. This means that they really need to be done in an extensible way.

To achieve this goal of all derived values are handled with the concept of a generator. A generator simply takes as input a collection of data sets and returns a new set based on its input. Due to using this general interface none of implementation details of a specific derived value need to be hardcoded. Every single one is treated as a plugin for the tool.

6 Time Series Displays

There are a lot of ways to represent time series graphically. The goal is to pick one which best fits the tasks needed for analyzing SLS algorithms. So to start we really need to outline what's important. We're interested in finding trends, anomalies, and patterns in generally non-cyclic data. This immediately rules out several types of time series displays such as spirals, time series bitmaps / suffix trees, or calendar based displays. What we are really left with is some variety of line plot.

There are two things left to discuss about this choice in plot. The first is that SLS algorithms often result in time series with a distinctive shape. This is either an exponential decay or a logarithmic growth depending on whether we are looking at minimization or maximization. The important thing about these shapes is that there are vast regions of the time series which are very uninteresting. To compensate for this it would be beneficial to be able to distort the axis to exagerate the interresting regions of the time series.

The last issue to consider is that a comparing several different time series is a common task. When trying to do such comparisons there are two choices; put everything on a single high resolution display or split the information across several smaller displays (small multiples). When putting multiple similar time series on the same display we are going to get line crossings / occlusion issues. There is really no way to avoid that if we take the single display option. So to avoid the potential confusion of a single display we will use the small multiples approach. This adds on additional reqiurement that we be able to link the displays together so that panning/zooming/distorting axis of one display can be duplicated simulataneously to all displays which are being used in a comparison.

6.1 Distortable Axis

There are a lot of ways to represent time series graphically. The goal is to pick one which best fits the tasks needed for analyzing SLS algorithms. So to start we really need to outline what's important. We're interested in finding trends, anomalies, and patterns in generally non-cyclic data. This immediately rules out several types of time series displays such as spirals, time series bitmaps / suffix trees, or calendar based displays. What we are really left with is some variety of line plot.

There are two things left to discuss about this choice in plot. The first is that SLS algorithms often result in time series with a distinctive shape. This is either an exponential decay or a logarithmic growth depending on whether we are looking at minimization or maximization. The important thing about these shapes is that there are vast regions of the time series which are very uninteresting. To compensate for this it would be beneficial to be able to distort the axis to exagerate the interresting regions of the time series. See figure 6.

The last issue to consider is that a comparing several different time series is a common task. When trying to do such comparisons there are two choices; put everything on a single high resolution display or split the information across several smaller displays (small multiples). When putting multiple similar time series on the same display we are going to get line crossings / occlusion issues. There is really no way to avoid that if we take the single display option. So to avoid the potential confusion of a single display we will use the small multiples approach. This adds on additional reqiurement that we be able to link the displays together so that panning/zooming/distorting axis of one display can be duplicated simulataneously to all displays which are being used in a comparison.

6.1.1 Interaction Methods

The two methods for introducing distortions that I tried were a zooming metaphor and a stretching metaphor. In the zooming metaphor a region was selected to be a focus. And zooming done within this region would increase the amount of screen space used within the selection while leaving the amount of screen space used

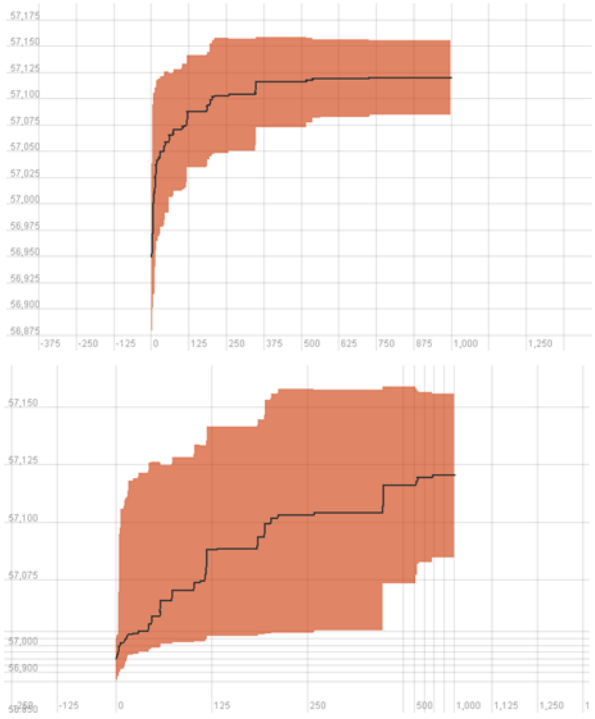

Figure 6: Top: A function displayed on undistorted axis. A large portion of space is devoted to a region where this function is flat. Bottom: The same function with distortions applied to reduce the amount of screen space given to the stagnation at the end of the search.

for elements outside the selection constant. While in the stretching metaphor the user selected a region then resized the selection (like a window) to consume a desired amount of screen space. The latter choice appeared to be more intuitive and is what's being used in the current version of the application.

6.1.2 Vizualizing the Distortion

When drawing the grid lines and axis labels for a line plot there are two goals; (1) it should be easy to spot the regions which have been distorted or expanded relative to the surrounding regions, (2) grid line clutter and label overlap should be minimized. These are somewhat contradictory goals. To achieve the first one some regions have to be more cluttered than others, potentially by a substantial amount. To achieve the second one there needs to be a limit on how many grid lines can be drawn within an image.

Both these requirements are for still images. There is yet another requirement when considering interaction: the grid lines and labels need to be "stable". This means that grid lines and labels should remain relatively unchanged as you interact with the display. For instance while panning, if there's a line and a label for 150 on the x-axis then this should only really disappear when looking at intervals that don't include 150. Similarly a line should never disappear from zooming in and a line should never reappear from zooming out.

This stability is very important for making the display easy to follow for the user. And sudden shift in what's being displayed can be jarring or confusing to the user. It is the ability to distort the axis which really turns this into a problem. (When the axis remain constant this is a trivial problem to solve). The important thing to remember is that the decision about which lines/labels to exclude/include as the user zooms in/out and distorts the axis must be made in the data space, not the display space. Attempts to solve this in the display space will result in unstable solutions.

The biggest difficulty is for deciding from a set of overlapping labels which one should be drawn. A potential solution is based off the following, first decide on a fixed interval (in data space) to draw labels at. This should be the same interval chosen to draw grid lines at. From there each label can be defined by the number of interval it takes to get to a label starting from 0, if two labels are determined to overlap on screen, the following rules are applied:

- 0 always wins
- an even label always beats an odd label
- if two label are both even then divide both 2 and repeat the application of these rules
- if two labels are both odd then add one to the lower number and subtract one from the top number and repeat the application of these rules

This results in a very stable way of determining which label will get drawn when a conflict is detected that can be resolved in time logarithmic to the value of the largest label in the worst case. See figure 7.

6.1.3 Linking Displays

As part of the model for distorted axis each display contains a pair of dimension objects. Dimension objects provide a listening framework that allow other components of the system, particularly other dimension objects, to be synched with them. This synching is asymmetric and is performed by duplicating any calls to methods which alter the extent or scaling of a dimensnion. As each dimension is independant of the other view linking can be done on independantly on each axis.

6.2 Managing Displays

The initial goal was to allow a flexible system for constructing displays. To further this the user is given control over how each display is presented through a plot manager tab that's part of each plot. This tab contains a list of each item within plot. Items in the list can be reordered to adjust occlusion betweens different plot elements. Items can also be expanded to provide additional information and control about the plot. For the two varities of plots currently supported the control is limited to changing the display colour and the additional information is limited to the collection of data sets which contribute to this plot. See figure 8.

7 Discussion and Future Work

Three core issues were outlined as being necessary for a tool to be used for SLS analysis. These were data management, derived values and interactive displays. The tool developed as part of this project provides a solution for each of these issues while trying to acheive two additional design goal of minimizing the use of dialogues in favour of drag and drop. As for the design goal of trying to remain flexible instead of following a prescribed set of displays, I can't really say this was achieved until more displays and derived values are provided to the user.

There are three categories of work that still need to be done. The first category requires doing some amount of user testing to determine if certain design choices (ie. drag and drop vs dialogues) are really worth it. There also needs to be some time spent improving the affordances for the various control schemes. For many cases it likely that the only reason certain aspect of the control is intuitive is because I implemented them. This will definitely be the case for interactions with the time series plots where time needs to be spent to simplify the controls.

On top of these usability and user testing issues there are a set of features which really need to be developed to improve the value of the tool:

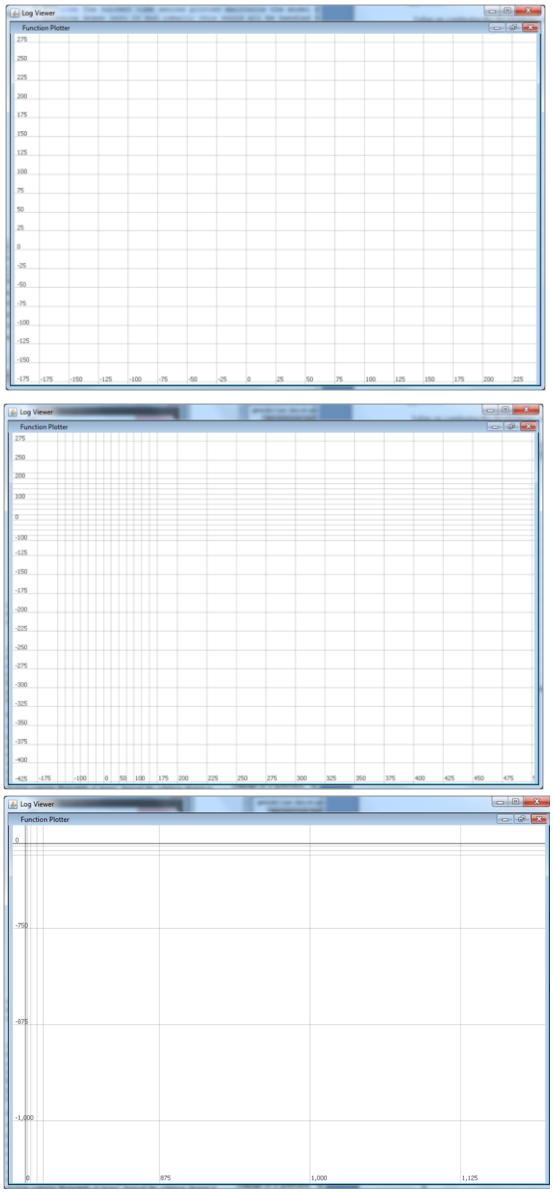

Figure 7: Top: No distortion, grid lines and axis labels are evenly spaced. Middle: Minor distortion, grid lines become grouped near the center of it and the number axis labels is reduced to prevent overlap. Bottom: Substantial distortion, grid lines become very clumped near the distortion and there is only room for 1 axis label.

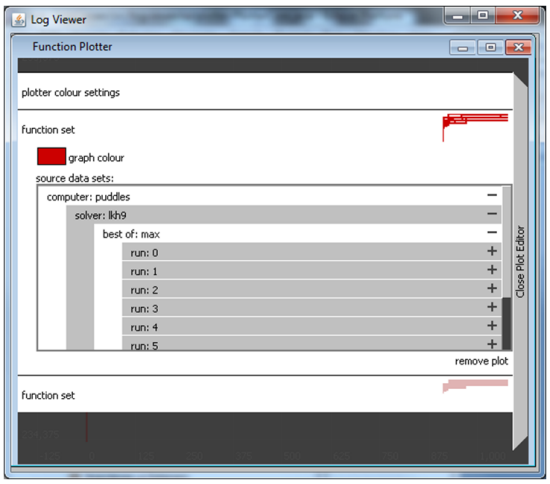

Figure 8: Top: The plot manager for a time series display.

- The current time series plotter maintains the model for the axis distortion and provides this information to whatever is being drawn onto it but ideally this would all be handled transparently. A component shouldn't have to know how to add inflection points into what used to be a straight line in the data because the display is heavily distorted. What this really means is that the plotter should not directly provide a graphics object to its subcomponents and instead provide a wrapper that includes knowledge of the distortion. This would allow for a general purpose distortable vector graphics display.
- The existing time series display needs to be enhanced to allow various annotations to be displayed.
- The original goal was to provide several more types of displays and derived values tailored to the TSP problem. These were all cut due to time restrictions and they really should be added back in.
- The types of data import are limited to a (poor) xml serialization of data set objects at the moment. This needs to be extended to be more robust.
- There is no facility to edit the tags for a data set from within the application. This really need to be addressed.
- The support for linking views need to be generalized to go along with the addition of new display types.
- The framework for supporting derived values also needs to be extended substantially. While allowing plugins to be developed and dropped in is definitely going to be a feature that remains it would also be beneficial to allow additional ways of computing derived values such as support for simple scripting.

I plan on continuing the development of this tool to aid in my own investigation of SLS algorithms and tuning.

References

[1] W. C. W. Hoong Chuin Lau and S. Halim. Tuning tabu search strategies via visual diagnosis, 2005.

[2] L. P. Manuel Lopez-Ibanez and T. Stutzle. Exploratory analysis of stochastic local search algorithms in biobjective optimization, 2009.

[3] F. Mascia and M. Brunato. Techniques and tools for local search landscape visualization and analysis, 2009.

[4] H. C. L. Steven Halim, Roland H.C. Yap. Viz: A visual analysis suite for explaining local search behavior, 2006.