Viz: A Visual Analysis Suite for Explaining Local Search Behavior

Steven Halim, Roland H.C. Yap
School of Computing
National University of Singapore
{stevenha,ryap}@comp.nus.edu.sg

Hoong Chuin Lau
The Logistics Institute Asia Pacific
National University of Singapore
hclau@smu.edu.sg,dcslauhc@nus.edu.sg

ABSTRACT
$\mathcal{NP}$-hard combinatorial optimization problems are common in real life. Due to their intractability, local search algorithms are often used to solve such problems. Since these algorithms are heuristic-based, it is hard to understand how to improve or tune them. We propose an interactive visualization tool, VIZ, meant for understanding the behavior of local search. VIZ uses animation of abstract search trajectories with other visualizations which are also animated in a VCR-like fashion to graphically playback the algorithm behavior. It combines generic visualizations applicable on arbitrary algorithms with algorithm and problem specific visualizations. We use a variety of techniques such as alpha blending to reduce visual clutter and to smooth animation, highlights and shading, automatically generated index points for playback, and visual comparison of two algorithms. The use of multiple viewpoints can be an effective way of understanding search behavior and highlight algorithm behavior which might otherwise be hidden.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces (GUI), Screen design (e.g. text, graphics, color).

General terms: Design, Human Factors.

Keywords: Local Search, Program Visualization.

1. INTRODUCTION
Local search algorithms (e.g. Tabu Search (TS)) [7] are often used to solve a variety of intractable $\mathcal{NP}$-hard combinatorial optimization problems such as Satisfiability (SAT), Traveling Salesman Problem (TSP), Quadratic Assignment Problem (QAP), circuit/graph layout, etc. Local search algorithms are used to get good (but not necessarily optimal/feasible) solutions. These algorithms are based primarily on heuristics and are incomplete (hence, do not give solution guarantees). This means that the precise behavior and the performance of local search is highly dependent on the chosen search parameters, components (heuristics), and search strategies.

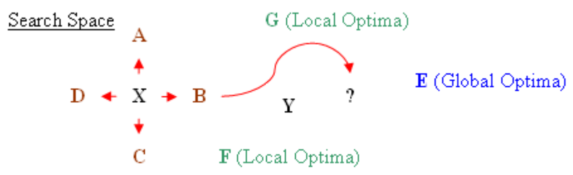


Figure 1: An example of a Local Search: X — current solution; {A,B,C,D} — neighbors of X; Y — another solution; E — global optima; F & G — local optima

Local search algorithms can be abstractly described as follows. Consider a satisfiability or optimization problem, we will use *solution* to mean a possible valuation of the problem variables (e.g. a route in a TSP). Note that a solution does not need to be satisfiable (e.g. a TSP route with several cycles). The set of all solutions is called the *search space* and we also call a solution in the search space, a *point*. In Fig.1, the letters A,B,C,D,E,F,G,X,Y denote solutions/points in a search space. Suppose the current solution is X, local search determines a subset of its search space — the local neighborhood of X here is {A,B,C,D}. A heuristic selects a new solution from the neighborhood (e.g. improving move B), or jumps to another solution outside the neighborhood (e.g. Y). The curve in Fig.1 is meant to illustrate a *search trajectory*, a series of solutions traversed during search, heading towards the desired goal (e.g. E: a global optima). As local search is incomplete, some problems are that: it may be trapped in local optima (e.g. F or G); and may exhibit cycling behavior (e.g. cycling among points X,B,G,X,B,G,...).

Nevertheless, local search is often the method of choice or perhaps the only practical approach for solving many combinatorial optimization problems simply because it may not be feasible to solve them with systematic search. Typically, a local search algorithm is quite simple — consisting of a combination of several heuristics with some search strategies (e.g. the two-edges swap heuristic used as the local move operator in our Re-TS for TSP in Section 6). Usually the search has some stochastic elements which add non-determinism (e.g. the search trajectory can behave differently between runs). Thus, unlike systematic search, it is difficult to analyze and predict the algorithm behavior of local search. For example, the algorithm designer may end up believing that the local search is doing one thing while it is doing something different, i.e. rather than intensification it is actually a semi-diversification or what [20] calls the metaheuristic (local search) 'failure modes'.

Local search algorithms usually require tuning, we call this the TUNING PROBLEM (see [5]). One approach to tuning is the black-box, automatic tuning approaches like CALIBRA [1] or F-Race [3]. Instead of understanding the behavior of local search in order to tune it, they probe the configuration space semi-exhaustively and pick the best found configuration. This is computationally intensive even for small problems. Since the approach does not attempt to explain, it does not address how to debug or improve the underlying local search algorithm. We believe that to improve local search algorithms (including tuning), it is also essential to understand the algorithm behavior.

A typical non-GUI approach for understanding local search behavior is to analyze numerical/statistical information derived from the search runs, e.g. an ad-hoc technique could be: run an experiment, compute some statistics, modify the algorithm and repeat. The idea here is that statistical measures can condense the large amount of data collected in the search. However, it is not easy to understand local search simply from a gross description/statistics.

In this paper, we propose a visualization tool which can enable the algorithm designer to understand the behavior of local search algorithms. A human-oriented tool is necessary as we want to enable not only the tuning of local search algorithms but also help in debugging and algorithm development. More precisely, we want to answer questions related to local search behavior posed in Section 2. Our visualization tool, VIZ, provides *off-line* local search *program visualization* [16, 17]. The visualization uses a number of animations: (i) generic visualizations, most notably, a 2-D abstraction of search trajectories; (ii) algorithm specific visualizations; and (iii) problem specific visualizations. We believe that the generic visualizations capture much of the interesting behavior and gives the main explanation as well as linking the search trajectory to the other specific visualizations. VIZ also allows two algorithms to be animated and compared.

Using VIZ, the algorithm designer has the ability to perceive local search behavior which may not have been noticed. From there, one can adjust/debug/tune/redesign the local search algorithm to attain better performance. This forms a tuning cycle: problem perception → modify the local search → identify the results of modification (including any problems which emerge). The tuning cycle ends when the algorithm designer has understood and tuned/improved the local search to achieve acceptable performance.

We are not aware of any comparable visualization to VIZ for local search and now survey some related work in animation and visualization. In the visual programming taxonomy of Myers [16], VIZ has both static and dynamic data visualizations. Although it is specific to local search, it can support generic visualizations of arbitrary local search algorithms and this can be enhanced with algorithm and problem specific visualizations. Other program behavior visualizations are: Whyline [12] which is a debugging interface for understanding program behavior; and [8] for visualizing dynamic interactions in object-oriented software. While these are quite different, they also emphasize understanding of program behavior from dynamic traces.

2. DESIRABLE VISUALIZATION PROPERTIES

Our main objective for developing VIZ is to enable the algorithm designer to understand the behavior of local search algorithms. We also want to be able to compare local search algorithms against each other. Rather than to visualize/animate the local search program itself (i.e. the algorithm code), we want to visualize aspects of the search trajectories and solution spaces visited throughout the search. This makes sense because although the algorithms may be quite different, the end result is still one or more search trajectories.

Some of the specific local search behaviors that we want to visualize are the following:

- Does it behave like as what we intended?
- How good is the local search in intensification, i.e. does it have sufficient exploration within a local neighborhood?
- How good is the local search in diversification. i.e. does it make successful non-local moves to previously unexplored parts of the search space?
- Is there any sign of cycling behavior?
- How does the local search algorithm make progress?
- Where in the search space does the search spend most of its time?
- How do two different algorithms compare?
- What is the effect of modifying a certain search parameter/ component/strategy w.r.t the search behavior?
- How far is the starting (initial) solution to the global optima/best known solution?
- Does the search quickly find the global optima/best known solution region or does it wander around in other regions?
- How wide is the local search coverage?

3. VISUALIZING LOCAL SEARCH

Clearly, it is easier to understand large volumes of information if it can be presented graphically in an effective fashion. In VIZ, we have three forms of visualization for understanding local search behavior, namely generic local search visualization, algorithm-specific visualization and problem-specific visualization. We use visualizations which exploit the human visual strengths: understanding *spatial information*, observing *trends* and *patterns*, and detecting *anomalies*.

3.1. Generic Local Search Visualizations

Generic (or abstract) visualization is a powerful concept because it is independent of the underlying local search algorithm and combinatorial problem. Generic visualization is possible in the context of local search for combinatorial problems because every problem has a search space model [7] and every local search algorithm works by mutating the current solution along this search space w.r.t generic properties such as objective values, distance, time (iterations). VIZ has visualizations of the search trajectory, objective value, fitness-distance correlation, and generic events during the search. Each visualization is explained below.

A). 2-D Abstraction of Search Trajectory

Since the search trajectory is closely related to behavior, the question is how to visualize the search trajectory given that each solution is normally in high dimensional space. We start by describing some existing work.

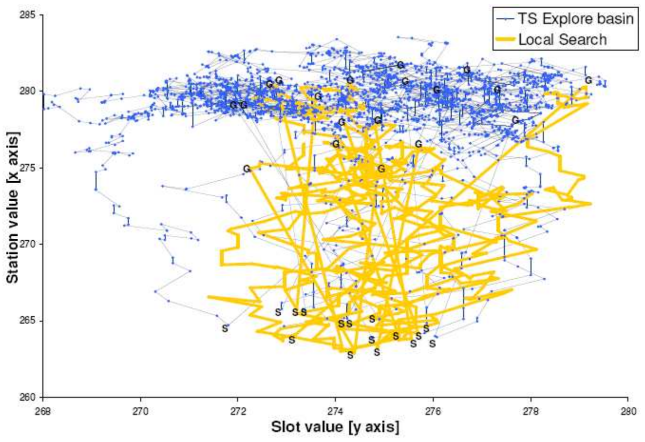


Figure 2: N-to-2-Space Mapping [9]

Kadluczka *et al.* [9] proposed an *N-to-2-Space* mapping of the search space and search trajectory which maps the search space down to 2-D. However the high number of collision between the points in *exponential* search space ($O(k^n)$ or $O(n!)$) versus the available *polynomial* screen space causes such static visualization to be ineffective for large n (See Fig.2). Furthermore, a static diagram like Fig.2 does not convey the dynamic behavior of search well.

Lau *et al.* [13, 5] introduced *Anchor Points*, see Fig.3 (left), as a reference set for approximating movement in the search space by measuring the *distance* (e.g. hamming distance, bond distance) between the current solution and the anchor points. They used the 2-D *Distance Radar* graph which is an animated histogram where the y-axis represents the distance between the current solution and respective anchor point. This visualization is less intuitive than the one in VIZ since the anchor points can change and the transition between the current solution s^t to its neighbor s^{t+1} is less intuitive. It also does not make use of 2-D layout.

We [6] use a different notion of Anchor Points from [13, 5] and make use of a definition which makes it feasible to layout the points in an abstract 2-D space, see Fig.3 (right). This is more intuitive and does not have the difficulty in N-to-2-Space mappings of making the visualization too complex. In [6], the paper focuses on defining appropriate generic measures for explaining local search and develops the idea of anchor points; while in this paper, the focus is on the user interface issues and effective graphical presentation.

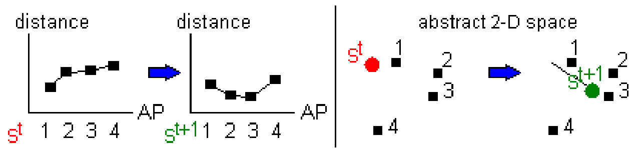


Figure 3: Distance Radar [13] (left) vs Abstract 2-D Layout in VIZ (right) — both reveal similar information: the movement of solution s^t to s^{t+1} from the area near solution 1 to the area near solution 2 & 3.

To layout the points, we use a spring model originally proposed in graph drawing, NEATO [10]. For each pair of points, place a 'spring' which has a natural length equals the distance of that particular pair. As spring will try to return to its natural length when stretched or shrunk, the idea is that a good layout is one which reduces the overall spring tension, see Fig.4. Our current implementation uses an approximation of the spring model for speed. We remark that it is not always possible to have a 'perfect' 2-D layout where the 2-D screen distance always corresponds to actual distance in the abstract 2-D space.

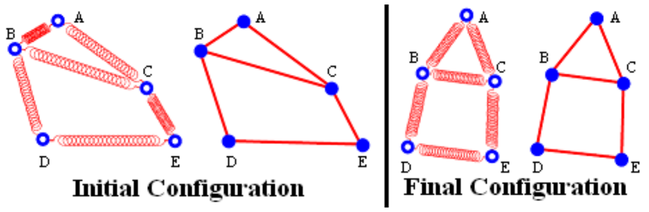


Figure 4: An illustration of the spring model

After the points are laid out, they are drawn with different colors which represent the range of objective value of the particular point w.r.t the best-known value (e.g. blue = 0%..1% off; green = 1%..3% off; brown = 3%..7% off; grey = above 7% off). This forms a 'contour map' which is helpful to highlight the quality of the region currently being searched.

We then produce an animation of the search trajectory by animating the trail of current solution against the set of anchor points. The search trajectory can be drawn as a line to show temporal sequence (see Fig.5) or as an area to show the visited regions (see Fig.12). The trail length is adjustable to provide different level of details. Arrows indicates direction and drawn with different color to indicate uphill vs downhill moves. This gives an effective way of subtly showing uphill and downhill moves without cluttering the animation. The search trajectory animation is the primary tool in VIZ to highlight various behaviors which may have been hidden. Fig.5 illustrates an animation to demonstrate diversification (after visiting blue colored region, the search is moving into different search space) **(A, Red Line)** and solution cycling (moving back and forth in the search space) **(B, Blue Line)**.

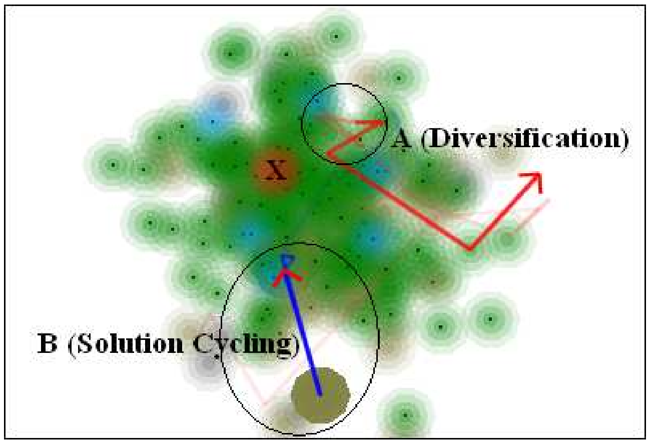


Figure 5: Search Trajectory Visualization. Two search trajectories **A** and **B** are visually compared. **X**: Global Optima/Best Found.

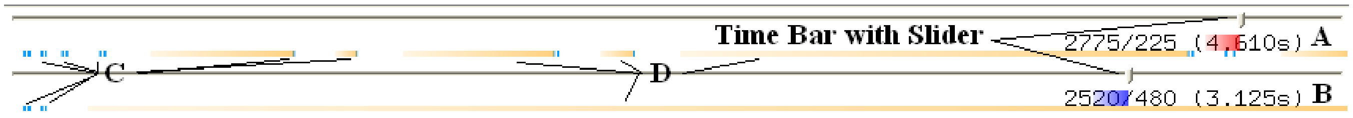


Figure 6: Event Bar to highlight generic events: **(A) & (B)** – two search trajectories can be compared side by side, each trajectory has its own indicator for: [Iterations elapsed/Iterations left (Actual search time)]; **(C)** – examples of 'new best found solution' (blue bars); **(D)** – examples of 'series of non-improving moves' (shades of orange).

B). Objective Value over Time

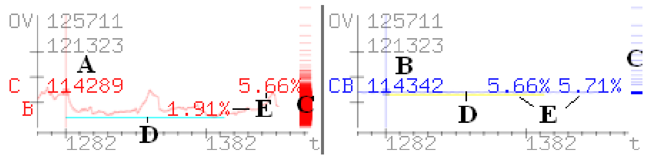


Figure 7: Objective Value over Time, enhanced with various statistical information. The 'C' and 'B' along the y-axis refer to 'current' and 'best found so far' solution at that iteration, respectively.

Objective value (or fitness) is often the key attribute that drives a local search algorithm. Rather than simply plotting a conventional graph of the objective value over time/iterations (see Fig.7: plot **(A)** & **(B)**), we enhance this with various statistical information to help understand the overall context of how the objective value is changing. This includes: max, min, a frequency histogram to highlight the average and distribution of objective values found by a local search run **(C)**, a line to indicate best-found-so-far **(D)**, and an indicator of percentage-off (the gap in percentage between current objective value w.r.t the best objective value found throughout the search run) **(E)**. Notice that we can display two visualizations for the objective value simultaneously for comparing two different algorithms (the red **(A)** and blue **(B)** lines).

C). Fitness Distance Correlation

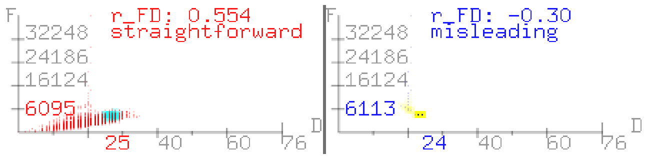


Figure 8: The FDC scatter plots of two search trajectories, the Red Line and Blue Line from Fig.5.

FDC analysis is meant to give a rough measure of the problem difficulty. In FDC analysis, we want to know whether there exists a correlation between the Fitness (F) and Distance (D) of the solutions w.r.t the nearest global optima (or best found solution). The FDC coefficient, r_{FD} (See [7]), is defined as: $r_{FD} = \frac{covariance_{FD}}{variance_F * variance_D}$. It is conjectured that for a minimization problem, it is:

1. 'straightforward', when $r_{FD} \geq 0.15$, as the fitness increase as the local search is approaching global optima ($\approx$ a linear line in FDC scatter plot).
2. 'difficult', when $-0.15 < r_{FD} < 0.15$, as there is very little correlation between fitness and distance w.r.t global optima. ($\approx$ a line parallel to x-axis in FDC scatter plot).
3. 'misleading', when $r_{FD} < -0.15$, as the fitness decrease as the local search is approaching global optima. ($\approx$ an inverse linear line in FDC scatter plot).

The FDC can be visualized by plotting the fitness difference along y-axis and distance along x-axis between the solutions with the nearest global optima (or best known, if the global optima is not available). In VIZ, rather than a static FDC scatter plot, we can incorporate more information with an animation of the position of current solution w.r.t the nearest global optima/best known by plotting the F-D information over time as shown by the movement of the points with inverted color against the backdrop of the overall FDC scatter plot in Fig.8.

D). Event Bar

Like a video recorder, we employ index points/regions for moving around in time during playback. Local search usually needs to run for a large number of iterations, e.g. thousands to millions. As such, the time bar alone may be either too fine or too coarse a granularity for moving around the search trajectory. To highlight interesting portions of the trajectory, VIZ automatically computes highlights (index points/regions) on *generic events*, e.g. 'new best found solution' **(C)**, 'series of non-improving moves' **(D)**. The index points/regions are attached to the time bar, to make it easier to skip possibly boring parts of the animation, see Fig.6.

2.2. Algorithm-Specific Visualizations

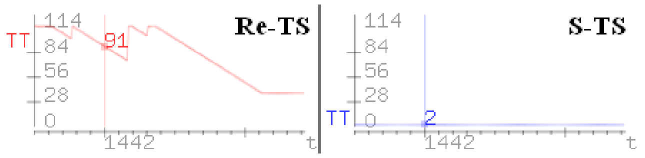


Figure 9: Tabu Tenure over Time for two types of TS, the details will be discussed in Section 6. The 'TT' along the y-axis refers to 'tabu tenure' at that iteration.

Algorithm-specific visualizations refer to those that are specific to a particular local search algorithm. What is being visualized here is usually the dynamic parts of the local search algorithm. More precisely, we are interested to visualize the change in the values of dynamic parameters over time. Some examples are: (i) observing the current temperature T in Simulated Annealing; (ii) observing the tabu tenure over time in Tabu Search, see Fig.9.

2.3. Problem-Specific Visualizations

Problem-specific visualization is the most intuitive form of visualization as it is directly related to the problem being solved. For example, early work using TSP visualizations

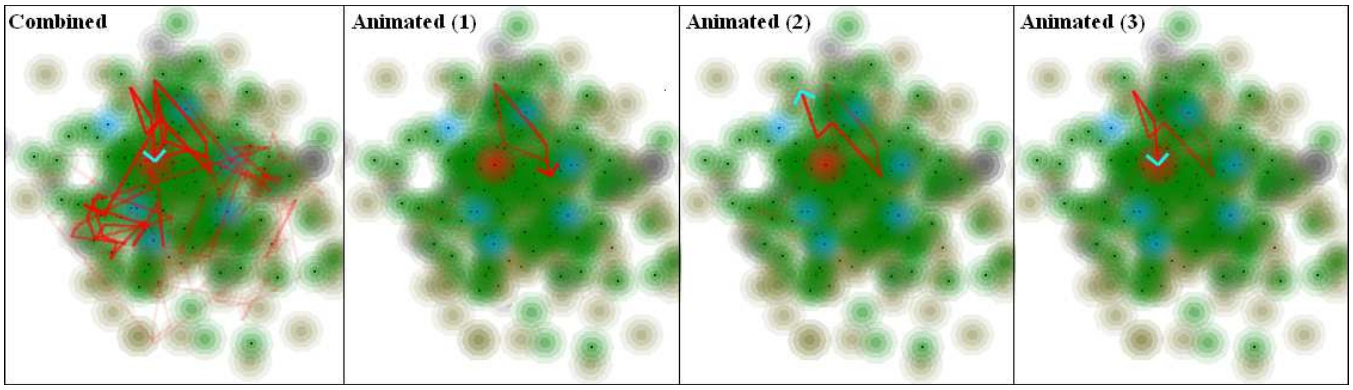


Figure 10: Combined: information is presented all at once (useful for overview) — versus — Animated: information is presented over time (useful for details). *Alpha blending* is used to show the decaying animation.

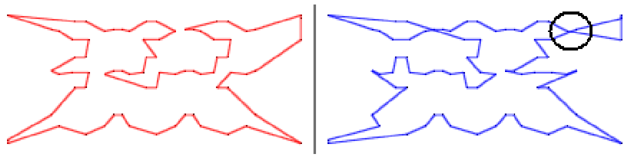


Figure 11: TSP tour visualization of instance pr76 from TSPLIB [18], left: optimal tour, right: non-optimal tour as there is an obvious crossing in the tour (circled).

(see Fig. 11) to assist man-machine interactive optimization already begun in 1960s [15]. More recent work is the Human Guided Search approach, see [11] for a representative paper.

Some local search tools (e.g. COMET [19], a programming language approach for Constraint-Based Local Search) provide a built-in interface that allows the algorithm designer to implement problem-specific visualizations.

Problem-specific visualizations have their limitations. While some problems have natural visualizations, particularly those which can be cast in a spatial setting (e.g. TSP), it is not so clear how to do it in other cases. There is also information overload since looking at the full solution has too much detail. Also it is harder to visualize the search trajectory since a problem specific visualization focuses too much on the current solution in gory detail but does not show what is going on in the local search across a time interval, e.g. it might be difficult to see if the local search is trapped in a local minima simply by looking at several TSP tours or a tour animation.

4. PRESENTATION ASPECTS OF VIZ

We now discuss various presentation aspects of VIZ which are essential for effective explanation of search behavior.

4.1. Multi-Source Visualizations

Each visualization that has just been described before is capable of explicating some aspect of local search behavior which may have been hidden in the search trajectory. However, relying on one type of visualization *alone* can be myopic and does not convey the full picture of what is happening during search. We believe that multiple points of view are required given the difficulty of analyzing local search behavior. For example: we could observe solution cycling in Fig.5, trajectory **(B)** but if we do not also observe Fig.9, right side (S-TS), we may not realize that the cause of such cycling is because of the low tabu tenure at that moment — a detailed explanation about this issue is given in Section 6.

In psychology, Situation Awareness theory [4] points out that the human user is unable to see multiple visualizations that require high attention at the same time. When human is bombarded with a number of displays, the overall scanning ability drops resulting in concentration on only a small fraction of the displays, thus losing Situation Awareness. To address this problem in VIZ, we designate the search trajectory visualization as the main screen where most information about local search behavior will be displayed. The other visualizations serve as peripheral visualizations to backup the main one. Nevertheless, as search playback in VIZ is not on-line, this issue is not so severe as the user can always pause, rewind, and replay the search at any time in case some information was missed. To better support such multi-source visualizations, VIZ presents the visualization windows in a Multiple Document Interface (MDI) style.

4.2. Animating the Information over Time

As local search runs for a large number of iterations, the most natural visualization is to present the information dynamically over time using animation. Such information over time will be hard to be seen statically as trying to draw everything in one display will tend to clutter the visualization. A comparison of static versus animated visualization of the same data is given in Fig.10. The amount of data animated is a user specifiable window of a portion of the search trajectory with l consecutive solutions. The animation makes it look like a trail of length l. By changing the window size, one can choose the tradeoff between a static and dynamic displays.

Animation is achieved by drawing a series of consecutive pictures with small changes drawn with smooth transitions. To avoid flicker, VIZ exploits *alpha blending* to draw smooth transitions by fading out the color of the trail's tail gradually.

VIZ also allows the user to adjust the search playback speed which determine how fast the animation will be drawn. This is essential as different individuals have different visual capacities in discerning the information from animation.

4.3. Color and Highlighting

Given the visual complexity, it is important to make good use of color/grey scale level, object sizes/shapes, texture, orientation, labeling, etc. The idea is to highlight information while minimizing information overload so that one can take in more at a glance.

In Viz, highlighting is used in a number of ways. In search trajectory visualization, color is used to form contour map which highlight the quality of regions in the search space; Alpha blending is used to show search coverage (see Fig.12); Colors are used to label similar objects, e.g. Red for search trajectory 1 and Blue for trajectory 2 in all visualization windows; The invert of a color: (255-R,255-G,255-B) can be used to visually contrasting certain part of the objects labeled with the original color: (R,G,B); In various charts, we draw scale to assist the user in determining the values.

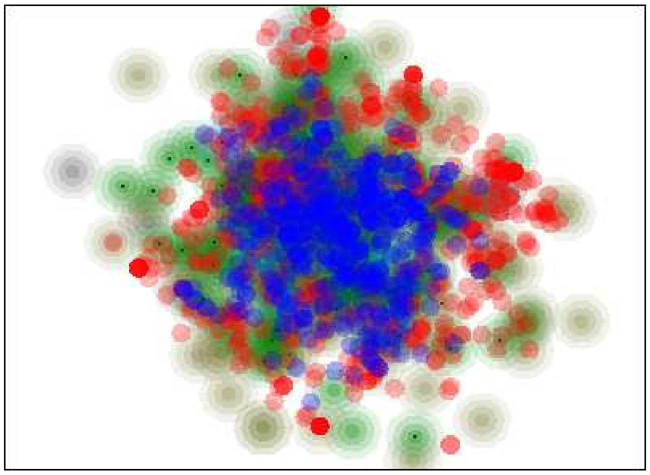


Figure 12: *Alpha blending* is used to highlight search coverage (darker ≈ more search around that region). Here, we can see that the red search trajectory is more spread than the blue search trajectory.

4.4. Learning via Visual Comparison

An interesting feature of Viz, which is available in almost all its visualizations, is its support for learning via visual comparison. This is done by allowing one to playback two local search runs concurrently and draw both visualizations in either juxtapose (side-by-side) or superimpose (overlap) mode.

Here, we want to exploit the human ability for detecting visual similarities and differences. For example, suppose we have two search runs of local search LS: run_1 and run_2, where run_1 is LS with parameters ϕ and run_2 is LS with some different parameters ϕ'. Then, we can investigate whether the changes in the parameters from ϕ to ϕ' is the cause for any significant differences in the search. Another usage of visual comparison is to compare the general behavior of two distinct local search algorithms LS_1 and LS_2; or to compare two parts of the same search trajectories. We can also use visual comparison to check the robustness of a stochastic local search by checking whether the performance of two runs of the same local search with the same settings produce similar performance. Fig.13 depicts a comparison of two trajectories using multiple views.

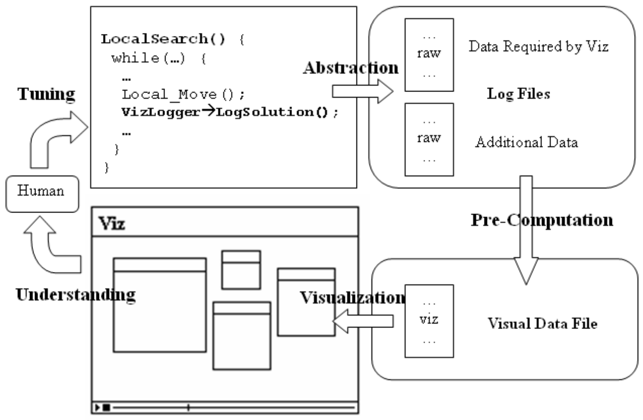
4.5. Viz Graphical User Interface

The overall Viz GUI is shown in Fig.13. The GUI elements should be familiar as it utilizes the Windows XP look and feel. Viz is developed using Microsoft Visual C# 2005 which utilizes .NET Framework 2.0 for the implementation of the common controls used in Windows XP. The core visualizations screens are drawn using CsGL: a C# Graphics Library wrapper for the standard Graphics Library OpenGL.

One notable UI feature of Viz is a trivial but yet very important feature: customizability, as not every user are comfortable with the default visualization settings. The control panel in Viz allows user to zoom/pan the search trajectory visualization screen, to choose between juxtapose or superimpose mode for visual comparison, to change color scheme, etc.

5. USING VIZ

The current version of Viz is designed for *off-line* (*post-mortem*) analysis. This means that Viz is used for analyzing one or more runs of one or more local searches after they have been completed. It uses log files created during runs of the local search to do its visualization. It is important to record enough information (keep the log size small) and to reduce the amount of disturbance (if any) to the local search being analyzed. These issues are outside the scope of this paper. Fig.14 gives an overview of how Viz is used as a local search program visualizer/debugger. an instance of the abstraction and visualization process discussed in [17] in the context of local search.



```
LocalSearch() {
 while(…) {
  …
  Local_Move();
  VizLogger→LogSolution();
  …
 }
}
```

Figure 14: Overview of Viz workflow and usage

We have focused on off-line local search visualization for the following reasons:

1. We want to avoid impacting on local search performance and avoid the overhead of visualization and time spent in the interactive user visualization process.
2. We want interactive visualization where the visualizations and interactions are smooth and not lagging. Depending on the complexity of the local search, it may be computationally intensive such that it causes jerky animation and slow response per iteration. An off-line mode has the advantage that one does not have to wait for the search process to generate new data and furthermore, some of the necessary works can be pre-computed.

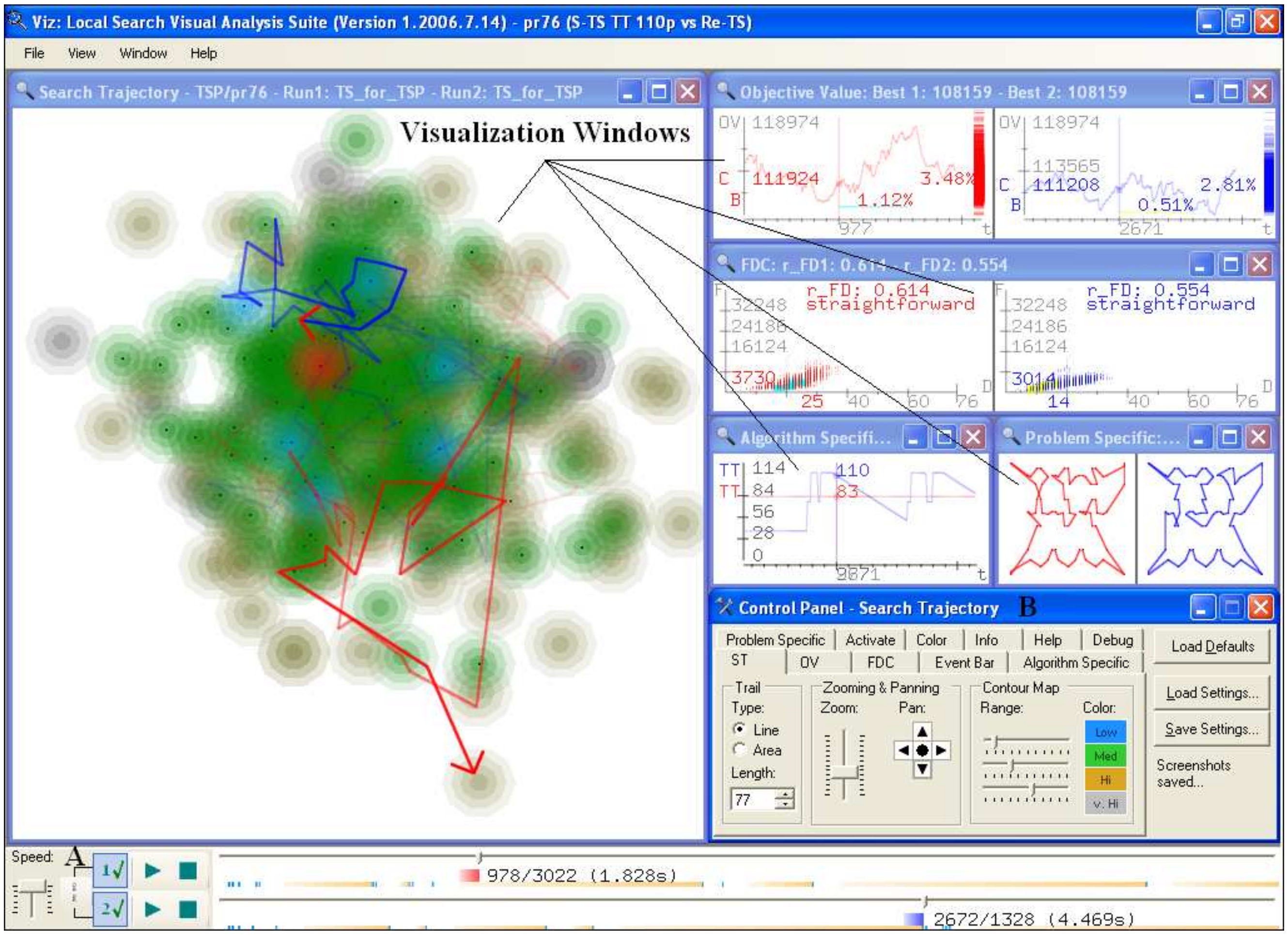


Figure 13: The overall GUI of VIZ; **A** - Search Playback Controller: Each search trajectory can be played back independently or played back simultaneously with the other search trajectory. We can also turned on/off the drawing of the search trajectory and adjust its playback speed; **B** - Control Panel: Most of the UI elements of VIZ are customizable.

3. The VIZ visualizations make use of the fact that we have access to the entire search trajectories, e.g. we know the best solutions found, can select representative anchor points for a search trajectory, pre-compute statistical information, pre-compute index points for the event bar, etc. It also allows one to jump into the 'future', in general, move anywhere within existing search trajectory or trajectories. An *on-line* visualizer would be more limited since it would only has access to past trajectories. For instance, this makes it harder to select anchor points since there is less points to work with. Because VIZ is off-line, it also means that one can make more stable visualizations, e.g. the objective value visualization makes use of the best information rather than relative information. It also makes it easier to do comparisons of two trajectories.

4. The off-line mode uses log files which makes it easy to be interfaced with virtually any local search implementation, e.g. COMET [19], MDF [14], etc.

We remark that on-line visualization for local search is also interesting but since it will be more restrictive, it is natural to first develop good tools for the off-line problem.

Apart from its basic usage as program visualizer/debugger, VIZ is also useful as a presentation tool for explaining to others in a more intuitive fashion how a local search algorithm works. In fact, since local search algorithms are usually not complex, one might on the basis of a VIZ animation be able to implement a variant of the algorithm simply by looking at a VIZ presentation without any other details. We envisage VIZ can be a useful tool for giving talks and teaching!

6. AN IN-DEPTH EXAMPLE: Developing Re-TS for TSP

To better understand the visualization and how to use VIZ user interface, we use a working example which shows how one can use VIZ to aid in developing a Reactive-Tabu Search (Re-TS) [2] local search algorithm for solving TSPs.

Phase 1: Understanding

We begin by analyzing the problem theoretically. A typical TSP search space is $O(n!)$ as there are $n!$ valid tours/solutions — these tours would have vary in their solution quality. Although the search space is very big, one strategy is to make use what is known about TSPs. Here we use the idea that TSPs have been shown to follow a 'Big Valley' property [7] — this means that good local optima, including the global

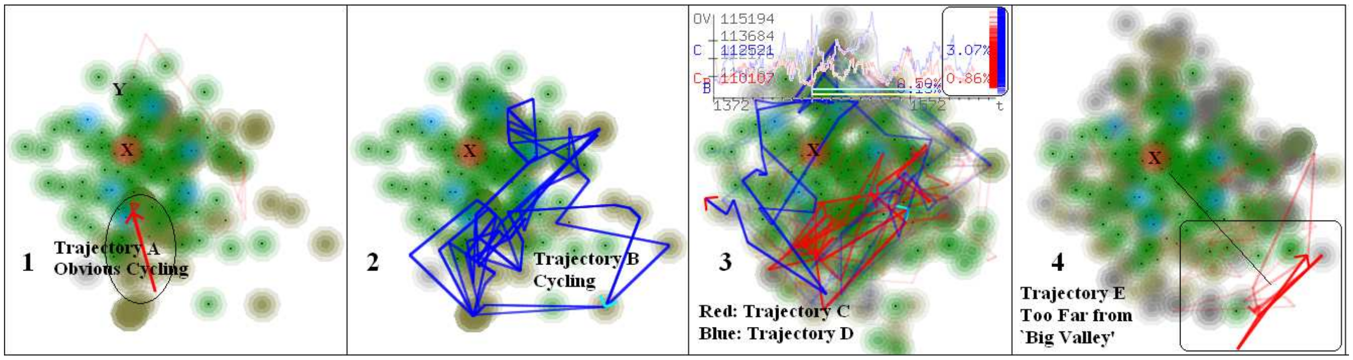


Figure 15: Search behavior of S-TS local search algorithm with different tabu tenure (TT) settings for TSP instance 'pr76'.

optima, are clustered in the 'Big Valley' region. A heuristic is to assume that when the local search finds a local optima for the TSP, this solution shouldn't be too far from the 'Big Valley', and thus not too far from the global optima. In this heuristic, the size of this 'Big Valley' is assumed to be much smaller than the full search space. This heuristic leads us to concentrate the local search effort in the 'Big Valley' region so that it has a better chance to find good solutions quicker than if the local search does not intensify around it.

We can use FDC analysis to check the existence of 'Big Valley' in TSP by checking whether the TSP solutions that are close to global optima/best found also have small TSP tour costs (better fitness), and whether the solutions that are far from global optima/best found are relatively poorer than solutions in the 'Big Valley' region, i.e. we want to see a positive correlation between Fitness and Distance. Let's examine Fig.16: All the scatter plots have linear shape and high r_{FD} (> 0.15) — a positive correlation. We also observe that almost all local optima are close to global optima/best found (distance $\approx \frac{1}{3} * n$). This means that the objective value can provide good guidance for the local search since the FDC analysis shows that when local search moves to a better solution, it is somewhat closer to the global optima.

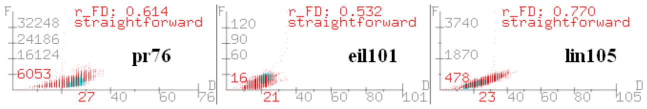


Figure 16: FDC Scatter Plots for TSP instances ([18]).

We will use a local search method called *tabu search* (TS)[1] with two-edges swap heuristic to solve the TSP. For TS, *tabu tenure* is one of the most important parameter to set. To make TS effective, the algorithm designer must find a good tabu tenure length — which is not too short as it may cause TS to be trapped in solution cycling, and also not too long as it may prevent TS to explore promising regions of the search space. Currently, setting a good tabu tenure seems to be an art as many algorithm designers either try a large number of values by trial-and-error, or perhaps set a value that happens to be good. In this example, we show how VIZ is used to intelligently design a Re-TS strategy for adjusting the tabu tenure. Note that in this example, we will only modify the tabu tenure and keep other tabu search configurations (parameter values/components/search strategies) intact.

[1]Tabu Search is a neighborhood based local search that iteratively moves the current solution to the best non-tabu neighbors. Recently applied moves are forbidden/tabu to be re-applied for the duration of a sensitive parameter called *tabu tenure*. This forms a short term memory mechanism to prevent solution cycling. The details of how Tabu Search works can be found in [7].

We first use VIZ to investigate the effects of various tabu tenure settings for Strict-TS (S-TS) — TS with a static tabu tenure (TT). Fig.15 (Trajectory $\{$**(A)**...**(E)**$\}$) shows the S-TS search behavior on the fitness landscape of TSP instance 'pr76' (n is the problem size, here $n = 76$) when we set TT:
(A). TT = 2
(B). TT = $30\% * n$
(C). TT = $50\% * n$
(D). TT = $150\% * n$
(E). TT = $300\% * n$

We observe that for TT $= 2$, obvious solution cycling occurs after steepest descent to the first local optima (Fig.15-1). For TT $< 50\% * n$, we observe solution cycling in the first local optima found with an increasing cycle length (Fig.15-2). This is because with TT $< 50\% * n$, it is inadequate for S-TS to escape from the first local optima that it found. If this situation is not fixed, no matter how long we run the S-TS, the overall solution quality will never improve beyond the best solution found in that first local optima.

In the other extreme, when TT $> 150\% * n$, we observe poor intensification (Fig.15-4). The search trajectory **(E)** tends to go very far from the region where the global optima (**X**) lies — the 'Big Valley'. The quality of solutions found along trajectory **(E)** is also generally poor. When TS is unable to intensify around the 'Big Valley', the search seems poor.

For TT between range [$50\% * n \ldots 150\% * n$], there are differences in the search behavior (see Fig.15-3), but we did not find obvious solution cycling nor obvious poor intensification. However, we observed that using smaller TT, S-TS will explore more around a local optima that it found whereas when using larger TT, S-TS will diversify from that local optima quicker. This is best shown using an animation. However, a static snapshot in Fig.15-3 is roughly sufficient to explain this behavior. Here, trajectory **C** (red) — TT = $50\% * n$, is not too diverse. However, the trajectory **D** (blue) — TT = $150\% * n$, covers more search space and the objective value variance is wider (more diverse) than trajectory **C**.

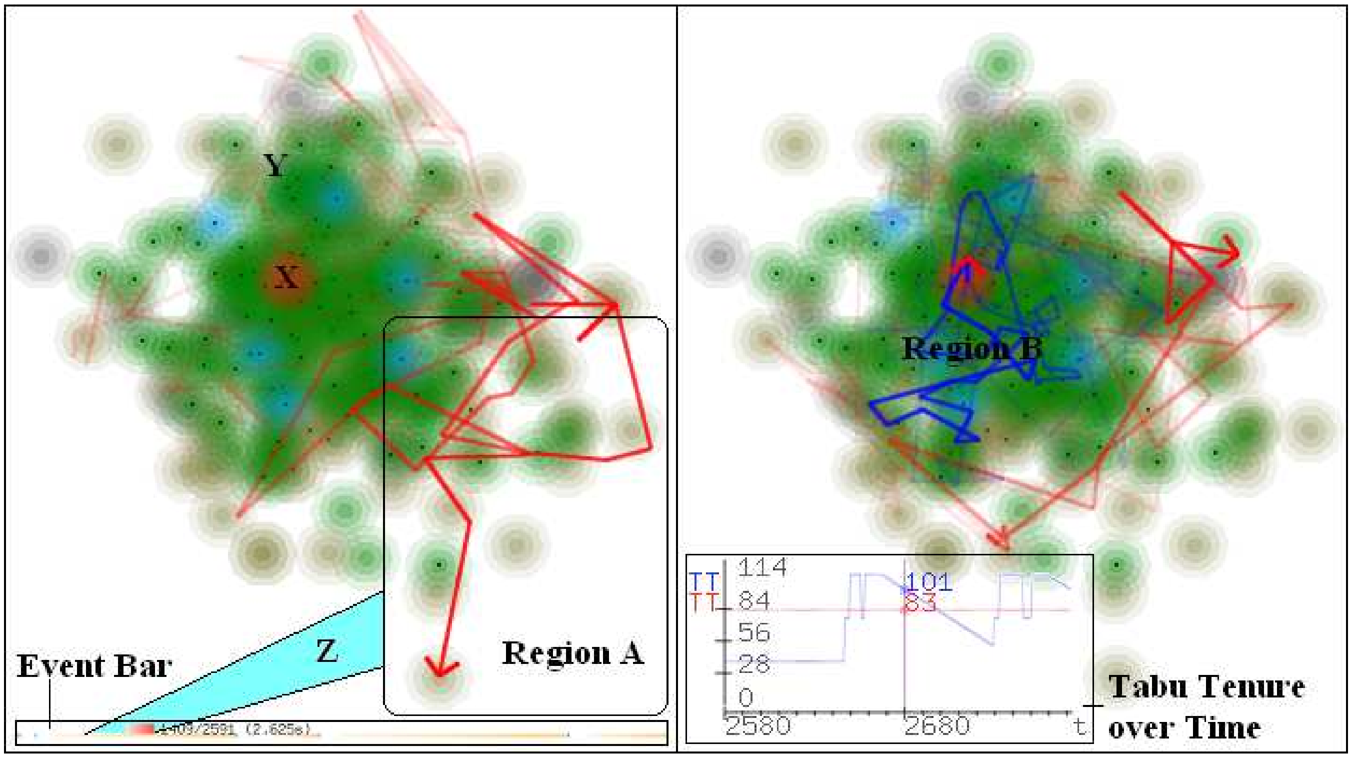


Figure 17: **Left side**: Search Trajectory of S-TS (red) with TT = $110\% * n$; X = global optima; Y = initial solution (greedy nearest neighbor); Z = the non improving period where the search is venturing around region A, which is far from the 'Big Valley'; **Right side**: The same search space with additional Search Trajectory of Re-TS (blue). This Re-TS concentrates its search around 'Big Valley' region B better than S-TS. Thus, in this instance (pr76), by the time Re-TS (blue) able to find the global optima, S-TS (red) is still wandering quite far from region B.

We tried this hypothesis on other TSP instances and conclude that a reasonable range for the tabu tenure setting for most TSP instances tested is in the interval $[50\% * n \ldots 150\% * n]$. Without visualization, it is harder to determine this tabu tenure range nor is it easy to see this tabu search behavior. However, the range is still quite large and furthermore different TSP instances require different tabu tenure settings.

We also observe another pattern in the search trajectory visualization. A full run of S-TS with TT = $110\% * n$ for pr76 managed to obtain the global optima (as reported in TSPLIB) after ≈ 3500 iterations. However, when we animate the run (shown in Fig.17, left side), we observe that starting from **Y** (initial solution), most of the time S-TS is already close to **X** (global optima). However, S-TS ventures to areas very far from the 'Big Valley' during period **Z** and during that period, no improvement is found at all. A natural question is: suppose most of the time the local search is already close to **X** — but can't actually reach it, can we tune the local search to reach **X** from **Y** quicker with a better strategy?

Phase 2: Tuning
After understanding the problems encountered, we move on to the tuning phase. We use Re-TS to help alleviate the problems found. In Re-TS, we avoid setting the tabu tenure statically, but instead we allow the tabu tenure to change within a pre-defined range according to certain event(s) encountered during search. Reactive tabu tenure value is quite logical because there are different local optima with different depths in the search space which requires different tabu tenure to make TS able to escape from it. Our task now is to set a reactive strategy: when and by how much should we increase or decrease the tabu tenure?

We have observed that smaller tabu tenure tends to intensify tabu search around a local optima and larger tabu tenure tends to diversify tabu search from a local optima. Our aim is to intensify around the 'Big Valley' region without being trapped in solution cycling. This gives the following Re-TS strategy — keep decreasing the current tabu tenure by 1 for every iteration until we reach the lower bound ($50\% * n$). Re-TS trajectory will be more focused on intensification in the local optima region. However, once the distance between the current solution to the best found solution (our guess for the location of 'Big Valley') is very close (< 10, possibly cycling), we set tabu tenure to be the upper bound ($150\% * n$). This produces a pattern of tabu tenure over time as shown in Fig.17, right side. The resulting Re-TS local search algorithm behaves better than S-TS, as it manages to concentrate its search on TSP 'Big Valley' region more than S-TS.

We have tested this Re-TS strategy with several other TSP instances and found that this Re-TS strategy produces reasonable results *across* different TSP instances though not always better than S-TS with a 'lucky' tabu tenure setting on a particular instance (see lin105 in Table 1).

Instance	S-TS ($90\%*n$)	S-TS ($100\%*n$)	S-TS ($110\%*n$)	Re-TS
pr76	109307	109091	109091	108159
eil101	630	629	629	629
lin105	14484	14479	14379	14484

Table 1: Best results of Strict-TS with different tabu tenure settings vs Reactive-TS after 3000 iterations

To summarize, this extended example illustrates how one can use VIZ to solve a problem by making some initial observations, form a hypothesis, improve the algorithm and test it. We can iterate this process as needed.

7. CONCLUSION

We have seen the power of visualization to explain the kinds of behavior listed in Section 2 that we would like to observe and understand in order to 'demystify' local search. We emphasize that some of these observations are hard to derive *without* proper visualizations. We believe that with visualization tools like VIZ, the algorithm designer can gain new insights in problem solving which can lead to the development of better as well as novel local search algorithms.

On a more practical front, it gives the algorithm designer, who is trying to solve a combinatorial optimization problem with local search, a tool for tuning the local search program to obtain better or faster solutions.

A takeaway message for program visualization is that perhaps the ideas of generic visual abstractions could be also useful in other domains. Having a variety of simple visualizations to enable insights from multiple viewpoints is also a basic idea which could be effective elsewhere.

ACKNOWLEDGEMENT

The authors would like to thank the reviewers for their constructive inputs.

NOTE

This paper is in color — the animations in the visualizations are best experienced with the prototype VIZ system or with the video demos. More details on VIZ can be found at: `http://www.comp.nus.edu.sg/~stevenha/viz`

REFERENCES

1. B. Adenso-Diaz and M. Laguna. Fine-tuning of Algorithms Using Fractional Experimental Designs and Local Search. *Operations Research*, 54(1):99–114, 2006.
2. R. Battiti and G. Tecchiolli. The Reactive Tabu Search. *ORSA J. on Computing*, 6(2):126–140, 1994.
3. M. Birattari. *The Problem of Tuning Metaheuristics as seen from a machine learning perspective*. PhD thesis, University Libre de Bruxelles, 2004.
4. M.R. Endsley. *Theoretical Underpinnings of Situation Awareness: A Critical Review*. Endsley and Garland (eds). Situation Awareness Analysis and Measurement. Lawrence Erlbaum Associates, Mahwah, NJ., 2000.
5. S. Halim and H.C. Lau. *Tuning Tabu Search Strategies via Visual Diagnosis*. To appear in MIC2005 Post-Conf. Volume. Kluwer Academic Press, 2007.
6. S. Halim, R. Yap, and H.C. Lau. Visualization for Analyzing Trajectory-Based Metaheuristic Search Algorithms. In *Poster Paper of European Conf. on Artificial Intelligence (ECAI06)*, 2006.
7. H.H. Hoos and T. Stuetzle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann, 2005.
8. D.F. Jerding, J.T. Stasko, and T. Ball. Visualizing interactions in program executions. In *Intl. Conf. on Software Engineering*, pages 360–370, 1997.
9. M. Kadluczka, P.C. Nelson, and T.M. Tirpak. N-to-2-Space Mapping for Visualization of Search Algorithm Performance. In *IEEE Intl. Conf. on Tools with Artificial Intelligence*, pages 508–513, 2004.
10. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.
11. G.W. Klau, N. Lesh, J. Marks, and M. Mitzenmacher. Human-Guided Tabu Search. In *Natl. Conf. on Artificial Intelligence*, pages 41–47, 2002.
12. A.J. Ko and B.A. Myers. Designing the Whyline: A debugging interface for asking questions about program failures. In *ACM Conf. on Human Factors in Computing Systems*, pages 151 – 158, 2004.
13. H.C. Lau, W.C. Wan, and S. Halim. Tuning Tabu Search Strategies via Visual Diagnosis. In *Metaheuristics Intl. Conf.*, 2005.
14. H.C. Lau, W.C. Wan, S. Halim, and K. Toh. A Software Framework for Fast Proto-typing of Meta-heuristics Hybridization. *To Appear in ITOR*, 2006.
15. D. Michie, J.G. Fleming, and J.V. Oldfield. *A Comparison of Heuristic, Interactive, and Unaided Methods of Solving a Shortest-route Problem*, pages 245–256. Michie, D. (eds). Machine Intelligence series 3. Edinburgh University Press, 1968.
16. B.A. Myers. Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. In *ACM SIGCHI '86 Conf. on Human Factors in Computing Systems*, pages 59–66, 1986.
17. M.J. Oudshoorn, H. Widjaja, and S.K. Ellershaw. Aspects and Taxonomy of Program Visualisation. In P.D. Eades and K. Zhang, editors, *Software Visualisation*, volume 7, pages 3–26. World Scientific, 1996.
18. G. Reinelt. TSPLIB - A Traveling Salesman Problem Library. *ORSA J. on Computing*, 3:376–384, 1991.
19. P. van Hentenryck and L. Michel. *Constraint-Based Local Search*. MIT Press, 2005.
20. J. Watson. On Metaheuristics "Failure Modes". In *Metaheuristics Intl. Conf.*, 2005.