NASA/CR- 1998 208290

FINAL
IN-61-CR
289220

NCC2-5227

Final Report:

Data Understanding Applied to Optimization

Wray Buntine and Michael Shilman
Electrical Engineering and Computer Science
University of California, Berkeley
Room 550, Cory Hall
Berkeley, CA, 94720-1770

Original proposal abstract

The goal of this research is to explore and develop software for supporting visualization and data analysis of search and optimization. Optimization is an ever-present problem in science. The theory of NP-completeness implies that the problems can only be resolved by increasingly smarter problem specific knowledge, possibly for use in some general purpose algorithms. Visualization and data analysis offers an opportunity to accelerate our understanding of key computational bottlenecks in optimization and to automatically tune aspects of the computation for specific problems. We will prototype systems to demonstrate how data understanding can be successfully applied to problems characteristic of NASA's key science optimization tasks, such as central tasks for parallel processing, spacecraft scheduling, and data transmission from a remote satellite.

TO: CASI

Table of Contents

1 Some Discrete Optimization Problems

This section briefly introduces the two problems we spent most of our time with during the research period. We have and continue to investigate other problems such as satisfiability [Gu *et al.*, 1997, Battiti *et al*, 1998], however, our successes and the majority of our efforts todate have been with two problems, hyper-graph partitioning and optimization of ordered binary decision diagrams.

1.1 Hyper-graph partitioning

When problems get really large, one possibility is to break them down into pieces, solve the components individually, and then piece together a solution. This approach addresses large optimization problems so they can be solved in pieces, partitions work performed across multiple processes [Ponnusamy *et al.*, 1994], and partitions a circuit too large to fit onto a single chip [Alpert *et al*, 1995]. The same technology, *partitioning*, can also be used for image segmentation [Malik *et al.*, 1997] and document clustering [Han *et al.*, 1998]. With k processors, a problem should be split into k approximately equal-sized, loosely-coupled

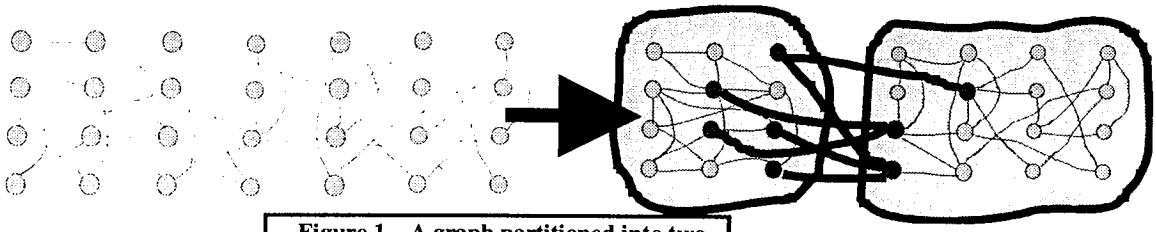


Figure 1. A graph partitioned into two.

pieces, one for each processor. Of course, it might not be possible to find pieces that are sufficiently loosely coupled to make this plan work. An illustration of a 2-ary partition of a generic search problem is given in Figure 1. Here, constraints are represented as arcs, and the set of constraints crossing the partition boundaries, represented as heavy arcs. To apply this kind of problem decomposition requires a good solution to the partitioning problem, so partitioning itself cannot be addressed this way.

Hyper-graph (or netlist) partitioning [Alpert and Kahng, 1995] is one approach to this general idea of problem decomposition. In the two-way min-cut netlist partition problem, one is given a hyper-graph whose nodes are usually referred to as cells, and whose hyper-edges are referred to as nets. The nodes are to be split into two partitions such that the minimum number of nets have nodes in both partitions. These cut nets are referred to as the cut-set. The cost function to be minimized is the size of the cut set, referred to as the cut-size. This is the hyper-graph version of the graph partitioning problem. Typically, a balance constraint is also enforced whereby the area of each partition must lie in a given interval, for instance [0.45,0.55] of the total area. A hyper-graph is represented in Figure 2. This does not satisfy the balance constraint of [0.45,0.55] because the right partition is of area 0.4 of the nodes (assuming each node has equal area). The cut-size is 4 in this case since while 5 wires appear cut, they only belong to 4 nets.

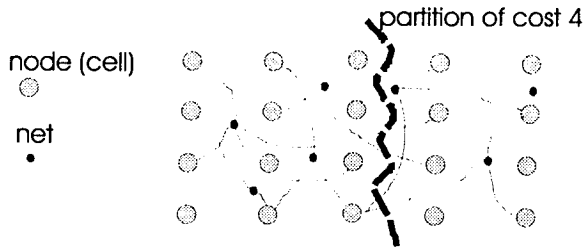


Figure 2. A hyper-graph and partition.

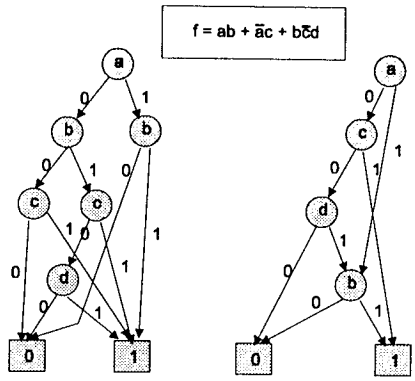
Common local search techniques for netlist partitioning are the Kernighan-Lin (KL) algorithm and the Fidducia-Matheysses (FM) algorithm [Alpert and Kahng, 1995]. Recent state of the art algorithms [Dutt and Deng, 1996; Li *et al.* 1995; Hauck and Boriello, 1996; Alpert *et al.*, 1997; Karypis *et al.*, 1997, Malik *et al.*, 1997] apply more clever heuristics for choosing the next node, techniques for finding an initial partition, spectral (eigenvalue) methods, or they preprocess the problem via clustering. However, most retain the local search FM or KL algorithms at their core.

1.2 Ordered binary decision diagrams

Many synthesis, verification and testing algorithms in VLSI design manipulate large switching formulae, and so it is important to have efficient ways of representing and manipulating them. In recent years ordered binary decision diagrams (OBDDs) have emerged as the representation of choice for many applications [Bryant, 1992; Bryant 1995]. A binary decision diagram is a graphical representation of a logic function. Given a particular variable ordering, the OBDD is a canonical form, which is

desirable as it makes equivalence tests trivial, but unlike most canonical representations, OBDDs are compact for many functions and lend themselves to fast execution of logical operations. For instance, an OBDD can be used to find all solutions to a SAT problem efficiently [Gu *et al.* 1997].

A simple example of two binary decision diagrams which represent the same logic function is shown in Figure 3. If we want to know the value of f for a particular assignment of the variables a, b, c, and d, we simply follow the corresponding path from the top of the OBDD. The function may be reconstructed from the graph by enumerating all paths to the 1 node and, in this case, converting to sum of products form. The disjunctive normal form read this way from the left and right OBDDs appears under the figure. The ordering of variables on every path of an OBDD is forced to be identical, notice the variables in the left OBDD occur in levels. This gives OBDDs their canonicity.



Although both OBDDs shown represent the same function, they differ in terms of the ordering of the variables, and, it is important to note, in terms of the number of nodes. The OBDD represents a function by a graph whose size is measured in terms of number of nodes, and this is the cost function to be minimized when constructing an OBDD from a logic circuit. In the best case the OBDD size is linear in the number of variables, as for the right OBDD, but given that any node can spawn two more nodes on the next level of the OBDD, it is obvious that OBDD size may be exponential in the worst case.

$$left = \bar{a}\bar{b}c + \bar{a}b\bar{c}d + \bar{a}bc + ab \quad right = \bar{a}cdb + \bar{a}c + ab$$

Figure 3. Two equivalent OBDDs.

2 Bayesian learning applied to the "big valley" effect and clustering methods

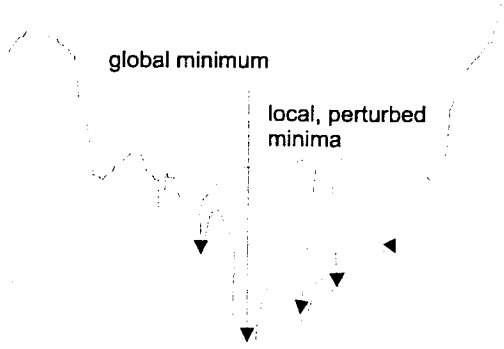


Figure 4. A global minimum in a well.

Modern optimization algorithms that are considered best of their kind, such as simulated annealing [Kirkpatrick *et al.*, 1983], search using multiple restarts such as "go with the winners" [Aldous and Vazirani, 1994] large-step simulated annealing [Martin *et al.*, 1991], sophisticated stochastic variants of local search [Gu *et al.* 1997, Fukunaga *et al.*, 1996, Martin *et al.*, 1992], as well as some versions of genetic algorithms have at their core the concept of the *big valley* which goes as follows. The search space is not a series of large valleys, but rather is a fractal structure with many valleys within valleys. Figure 4 illustrates this phenomena showing a big valley (see also Figure 12 in [Gu *et al.* 1997] and Fig. 3 in [Aldous and Vazirani, 1994]). The empirical evidence for this phenomona in discrete optimization problems is significant. The local, perturbed minima surrounding the global minimum are far more dense, and thus much more likely to trap a local search algorithm. Recent empirical research also suggests that better quality local minima seemed to be closer to the global minimum [Boese *et al.* 1994], so perhaps there is a broad shape in the big valley despite the disruptions caused by the many local minima [Boyan *et al.*, 1998]. Again, we would expect this to be a fractal property of the search space. The intuition justifying the big valley is as follows: as you move away from the global minimum in Hamming distance, the cost more often increases gracefully, and at least some of those perturbations of the global minimum are themselves local minima.

Simulated annealing is intended to bounce out of the sub-optimal local minima, but unfortunately does so with all the purposefulness of a random walk. Genetic algorithms are intended to combine the best features of these local minima and thus arrive at a superior minima, but unfortunately, again, use randomness when combining solutions. In a search space with many thousands of local minima, variations of local search can only seek out the minimum if they have some stronger overall direction. One approach that has recently seen apparent success with coping with "big valleys" is clustering, described next.

2.1 Clustering methods

State-of-the-art optimization algorithms for adaptive grids and finite-element graphs for scientific computing [Ponnusamy *et al.*, 1994], for CAD tasks like placement and routing [Mallela and Grover, 1988; Sun and Sechen, 1995] and hyper-graph partitioning [Alpert and Kahng, 1995] use a technique called *clustering* to achieve significant performance increases. In fact, on those problems where clustering is used, it seems to be essential to achieve state-of-the-art performance on large problems [Alpert and

Kahng, 1995]. For these problems, clustering is possibly the most significant performance breakthrough in the last two decades. Clustering can scale down a problem in complexity by an order of magnitude or more. It has the effect of smoothing out the big valley since each solution in the clustered space maps to a solution in the original space. Thus clustering allows extensive search

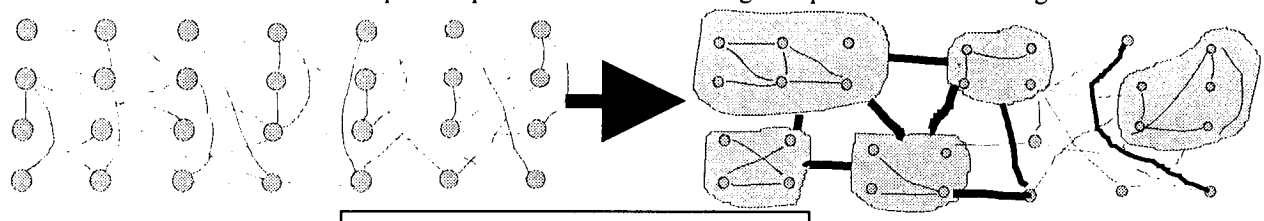


Figure 5. A graph and a clustering of it.

on the reduced problem, and often fast variations of local search will be adequate for this. All approximation methods are motivated by similar goals [Ellman *et al.* 1997] and bounding methods are similar [Selman and Kautz, 1996]. Clustering works as follows: For hyper-graph partitioning, certain nodes are tied together or clustered so that, thereafter, they always occur in the same partition. Nodes so tied together form a single super-node and induce a new problem with fewer nodes and often fewer nets (these merge or are absorbed within a single super-node). For a generic graph partitioning problem (where all nets are of size 2), a representative clustering is illustrated on the right of Figure 5. Clustering has typically been based on *ad hoc* variable strength heuristics together with standard clustering algorithms from the pattern recognition community, such as agglomerative clustering. An engineering-oriented study of several methods appears in [Hauck and Boriello, 1997].

2.2 Learning clustering for partitioning is feasible

One recent innovation in clustering is by Hagen and Kahng [1997] whereby the intermediate results/data of local search are combined to create clusters. Note a p-way partition of a variable space induces a clustering with p super-nodes. Overlaying k binary partitions to form their finest multi-way partitioning, similarly, induces a clustering with up to 2^k super-nodes. Hagen and Kahng recommend using k=1.5 $log_2 C$ partitions for this construction, for C the number of nodes.

The plot in Figure 6 illustrates some features of this purely data driven approach to clustering. These results are for the "industry2" circuit from the ACM/SIGDA Layout Benchmark Suite from MCNC, and similar results hold for all other circuits. This circuit has 12637 nodes and 13419 nets so $log_2 C$ is about 13. The X-axis gives the number of partitions k used to induce a clustering as a factor of $log_2 C$. First, as one increases k, the number of nodes in the induced clustering increases. This is shown by the dot-dashed line that banks upwards to the right with axis on the right side of the plot. Second, the solid line with axis on the left plots the number of nets in the induced clustered hyper-graph divided by the number of nodes. This is a measure of complexity of the induced clustered graph since more nets per node makes moves harder and a good solution less likely. The dashed line with axis on the far right plots the minimum cost partition of the induced clustered hyper-graph from 10 FM runs.

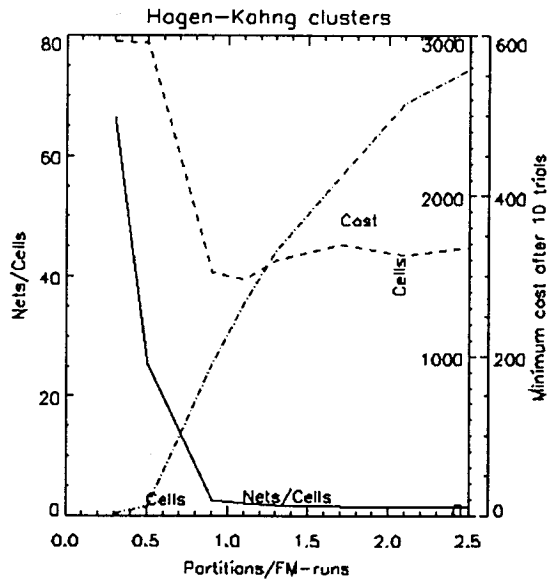


Figure 6. Hagen-Kahng clustering for "industry2"

For k greater than $log_2 C$ the induced clustering can have up to C nodes, so nodes no longer occur in the same finest partition by chance. Thus for k greater than $log_2 C$, the nets/nodes ratio should stay at a value comparable to that for the full-scale problem, but as k increases cost should increase because search fails to scale. Note that if you are learning clusters from data correctly, then as k increases, the quality of the clustering should only *improve* as you get more data. For k smaller than $log_2 C$, clustering is now introducing a random element, and thus as k decreases, cost gets progressively worse, along with the problem complexity. Yet standard connectivity clustering methods work well, essentially with k=0. This approach would benefit from the application of Bayesian learning methods [Bernardo *et al.*, 1994] to interpolate standard connectivity clustering modeled as prior probabilities with data-driven clustering in a statistically optimal sense.

2.3 A simple model for local minima in the big valley

This section presents a simple noise model showing how local minima might be viewed as perturbations of the global minimum, and thus how we might use that model to infer properties of the global minimum. We use this as an illustration of applying more rigorous data understanding methods to in the context of Hagen-Kahng clustering. This section presents one of the major results of our research.

The optimization problem we consider here is standard hyper-graph partitioning where the area of each partition is restricted to be less than 55% of the total area. Experiments were run on some of the larger problems in the ACM/SIGDA Layout Benchmark Suite from MCNC, where our copy was obtained from Charles Alpert's website at UCLA. Smaller problems are uninteresting for clustering methods. Details of the circuits used are listed under "standard results" in Table 1. The "Cut-size" column of these standard results lists best published cut-size on these tasks [Li *et al.*, 1995; Karypis *et al.*, 1997; Buntine *et al.*, 1997; Hauck *et al.* 1997; Alpert *et al.*, 1997].

Our simple probabilistic model of local minima is as follows: we claim characteristics of the global optimum occur with frequency $(1\text{-}q)$ in the perturbed local optimum. When we sample a local minima, it will have on average noise q on top of the global minimum. Therefore, to estimate whether a characteristic X holds in the global minimum, we estimate the frequency with which X occurs in local minima. If this frequency is greater than $(1\text{-}q)$, then under this model with high probability, X also occurs in the global minimum. Note that q is a free parameter in our model. We claim different optimization problems with have different intrinsic noise levels in their big valley, and thus we leave q free to vary. Thus statistical information about the local minima are used to infer characteristics of the global minimum. This is an extremely simple model, and we intend to investigate more extensive multi-dimensional probabilistic models [Buntine, 1996] in our research.

We apply this model to clustering as follows: the characteristics X we investigate take the form "node A and node B fall in the same partition." We have taken 200 FM runs to find a sample of local minima for each of the circuits in Table 1. The least cut-size for the 200 FM runs is listed under the far right column, "Best FM." We have taken statistics from these samples and for every pair of nodes (A,B) we then estimate the frequency with which the nodes lie in the same partition. For a given noise level q under the simple model above, we can therefore estimate which nodes should belong in the same partition of the global minimum. Contingent on a value for q, this information is then collated for all nodes (A,B) to produce a clustering of the nodes, since "node A and node B fall in the same partition" is an equivalence relation.

Table 1. Results from clustering experiments

Dataset	Standard results			Connectivity clusters			Large sample (200 FM) clusters			
	Nodes	Nets	Cut-size	Nodes	Nets	Cut-size	Nodes	Nets	Cut-size	Best FM
industry2	12637	13419	164	2841	3568	291	925	1476	183	396
industry3	15406	21923	241	3402	6489	285	949	2359	262	287
avq-small	21853	22124	128	7450	8326	171	2397	3363	168	305
avq-large	25113	25384	127	7684	8365	130	766	1564	188	320
s9234	5866	5844	40	909	916	47	251	335	41	49
s13207	8772	8651	55	1082	1016	67	325	444	53	85
s15850	10470	10383	42	1425	1348	53	682	810	56	98
s35932	18148	17828	41	3887	3567	43	1323	1391	45	104
s38417	23949	23843	49	1102	1331	67	791	1084	69	378
s38584	20995	20717	47	4017	3830	53	631	814	50	103
19ks	2844	3282	104	741	960	130	335	568	110	128
primary2	3014	3029	142	945	1169	149	748	1118	143	177
Geo. Mean	11335.82	11840.34	81.25	2142.28	2424.63	99.22	704.07	1046.22	93.56	165.27

To empirically measure the success of this clustering method, we use a high-performance non-clustering variant of FM, Adaptive Stochastic FM [Buntine *et al.* 1997] that runs stochastic FM to generate 10 quality local minima and then the best local minima has its cut-size reported in "Cut-size." We refer to this below as ASFM-10, and it takes time approximately equal to 60 FM runs, but since this is on a much smaller (clustered) problem, the run-time is not significant. To account for the free noise parameter q, we generate clusters for q=0.8, 0.85, 0.9, 0.95, 0.97, run ASFM-10 once for each q, and choose the clustering and resultant partition minimizing the cut-size. The cut-size measured from this one run of ASFM-10 is therefore one measure of the quality of the clustering, as reported under "Cut-size." Other quantities relevant to the quality are the number of nodes, the

number of nets, and their ratio. Generally we want fewer nodes, and a smaller nets-per-node ratio. These results are recorded under the column heading "Large sample clusters."

To provide a benchmark, we have compared these results against clustering obtained using a simple connectivity method we label "connectivity clustering" in the table. We use the agglomerative clustering method of Alpert *et al.* [1997] excepting that recursive re-evaluation of the connectivity measure is not done. We also looked at 8 different levels of agglomeration and chose the one giving the minimum cut-size. We claim these modifications are fair since no recursive re-evaluation was done for the large sample clustering method above, and the choice of optimum cut-size from 8 can only favor this method.

From the table, we can see the following: (1) The large sample clustering method generates significantly smaller hyper-graphs with significantly smaller cut-size. The difference is generally consistent across circuits. Thus large sample clustering is significantly superior in forming clusters to methods attaining current best published [Alpert *et al.*, 1997]. (2) Running ASFM once on a large sample clustered hyper-graph, and no other computation, the results on the far smaller clustered hyper-graph are near best published for the problem. Typical state-of-art algorithms, considerably more sophisticated with recursive clustering and multiple iterations, score a geometric mean of about 86 on this measure so this simple approach is near state-of-the-art. (3) The clustering results provide a full 200% increase over the best cut-size resulting from the entire 200 FM runs. Thus there is clear evidence that under this model we learnt significant information from the local minima about the global minimum.

2.4 Applying Bayesian learning to the simple model

The standard technique used for clustering is to agglomerate nodes with high connection strength, where connection strength for two nodes $C(A,B)$ is measured by some measure such as a weighted average of the nets the two nodes share. When two nodes are on a smaller net it contributes more towards connection strength than when two nodes are on a large net which is quite like cut anyway. $C(A,B) = \sum_{n \in nets,\, A,B \in N} 1/|N|$, for $|N|$ the net size. Calibration [Dawid, 1982] in one sense is a process whereby we map measurements about an event to measurable frequencies of the events occurrence. In this example the event we are considering is whether node A and node B occur in the same partition of a local minima. For a given hyper-graph, the frequency for this, $q_{A,B}$, is the frequency over all local minima. This corresponds to the q used in the previous section. Since there are many different hyper-graphs with many different nodes A,B, if all we knew was that the connection strength $C(A,B) = C$, then what can we say about $q_{A,B}$. Figure 7 shows a scatter plot of this situation. For all medium to large problems we found, we generated 200 local minima with FM and thus measured $q_{A,B}$ for pairs of nodes A,B in the problems. Each measurement is indicated by a dot, where the X-axis is $C(A,B)$ and the Y-axis is $q_{A,B}$. The scatter plot clearly shows that for $C(A,B)$ over about 0.8, we have high confidence that A and B will almost always lie in the same partition of a local minima. The solid line on the plot is the function $m(X) = (0.5 + 0.5*(X/0.6)^{0.7})$ which is, from eyeball, our best guess for $q_{A,B}$ given $C(A,B)=X$. In fact, by moment matching (a simple statistical inference procedure), when $C(A,B)=X$, the $q_{A,B}$'s as observed in the data have roughly a Beta distribution with mean $m(X)$ and effective sample size 5. Thus, by mapping the connection strength heuristic to the measurable event "node A and node B occur in the same partition of a local minima," we can use connection strength to predict the frequency of the event. We say we have calibrated connection strength (to the particular event). Clearly, we could also calibrate three different heuristics to one event (using supervised learning) which provides one way of combining the three heuristics to create a more powerful heuristic.

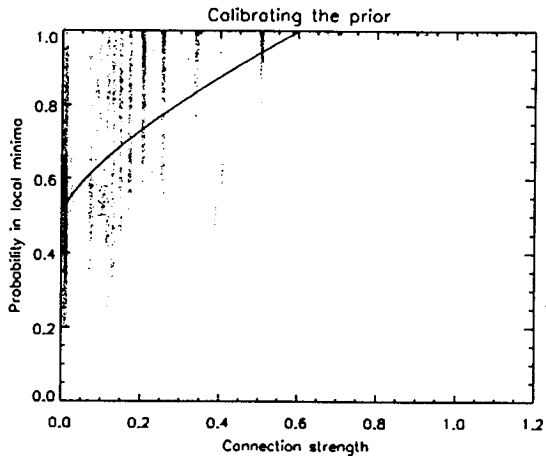


Figure 7. Calibrating "connection strength"

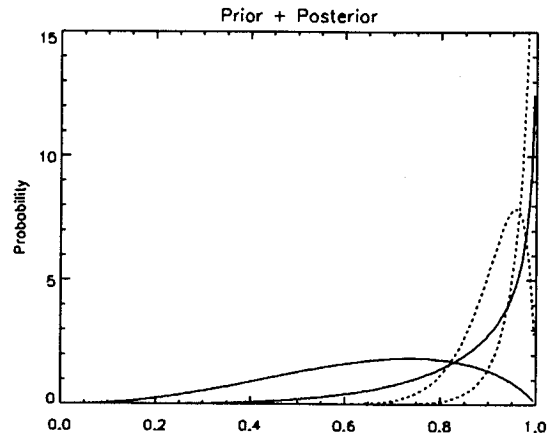


Figure 8. Updating connection probabilities

Note that calibration removes one of the key arguments against Bayesian methods which is the perennial question, "where do the priors come from?" For search, we have a good supply of benchmark problems and thus priors can be estimated by the calibration step above. Calibration in the search context thus eliminates most concerns with using the

Bayesian method [Bernardo and Smith, 1994]. This is excellent: when priors are known, most theoreticians agree that Bayesian methods provide superior statistical solutions to inference problems.

How can this calibration be used? Suppose you know that $C(A,B)=X$. For X=0.1 and X=0.4, this situation is plotted on Figure 8. The solid lines give the probability density functions for $q_{A,B}$ in the original calibrated situation where no data exists. The density function with more weight to the left (matching X=0.1) says that $q_{A,B}$ might well take on any value between 0.2 and 1 according to prior problems. The density function given by the solid line with weight to the right says that $q_{A,B}$ is quite likely above 0.9. In this case, our best estimates of $q_{A,B}$ are 0.55 and 0.95 respectively. Thus for X=0.4, one might consider clustering nodes A and B together *a priori*, which in fact is what a good clustering algorithm will do. For X=0.1, one would leave nodes A and B separated. Now run FM 15 times for this problem to obtain 15 local minima. Suppose you now know that nodes A and B occur together in 15 out of 15 local minima found. Then what can you say about $q_{A,B}$ now? Standard Bayesian calculations [Howard, 1970] for this Beta distribution yield the updated probability densities for the $q_{A,B}$ given by the two dashed lines. Our best estimates for $q_{A,B}$ are now 0.95 and 0.98 respectively. Thus from having seen nodes A and B together in 15 partitions, despite their low connection strength, we should be willing to join them in a cluster.

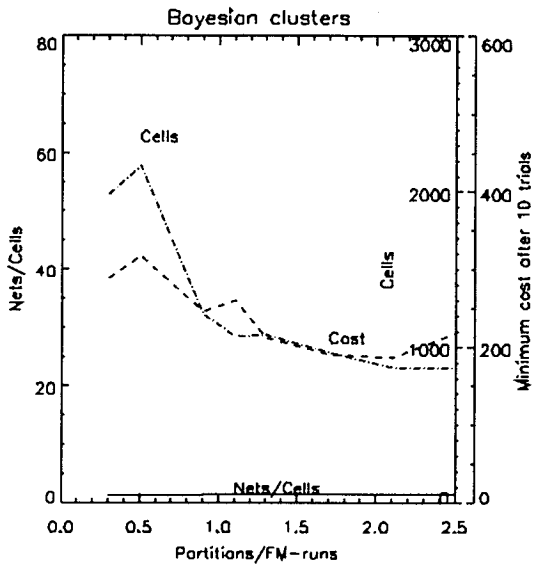


Figure 9. Bayes clustering of "industry2"

This approach yields a cluster-forming algorithm rather like the large sample clustering described in the previous section where q for each pair of nodes is now a function of k, the number of local minima taken. For k=0 this behaves like regular clustering (the connectivity clustering of the previous section). For k=200, this behaves like large sample clustering above and in between things are interpolated. For k=0.8 $\log_2 C$ for C the number of nodes we were able to achieve results near to the performance of k=200 and few hand-worked recursive applications of the methods (our experimental software does not yet embed clustering correctly as per standard methods [Alpert *et al.*, 1995, 1997]) yielded cut-sizes equaling or exceeding those of large sample clustering (above) and in some cases best published. Figure 9 gives results for this method for ACM/SIGDA benchmark dataset "industry2" in the same format as Figure 6 of Section 3.4. The results are stunningly superior on all counts and the problems we had noted earlier with Hagen-Kahng clustering are now eliminated. This result is generally consistent across circuits.

2.5 Conclusion for clustering

We have demonstrated the application of Bayesian learning methods to clustering as it is used in hyper-graph partitioning. We believe there is significant potential for continuing this research. Given the general important of the "big valley" effect for discrete optimization, we believe this research also has significant applications in other domains.

3 Visualization applied to optimization

We have done extensive experiments in several aspects of graph partitioning, some in consultation with Dr. Jeremy Frank of NASA Ames Research Center. These experiments have had mixed successes, however, we believe the lessons learnt are valuable, and we summarize these at the end of this section.

3.1 Static visualization applied to local search algorithms

This research was performed prior to the NASA Data Understanding project [Buntine *et al.*, 1997], however, it is relevant for subsequent discussion so we reproduce it here. We applied the standard FM algorithm [Alpert *et al.* 1995] to the SIGDA/ACM data sets mentioned in the previous section. We instrumented the algorithm and gathered a variety of data, which we visualized in 2-D and 3-D using a variety of tools available in the visualization language IDL. Of several views of the data we created, one stood out as being informative.

If the cost (number of nets cut after each move) is viewed, Figure 10 shows the general behavior characteristic of larger circuits during a typical run of the algorithm. The left plot shows cost for a full run, and the right plot zooms in on the performance during part of that. The X-axis corresponds to swaps, and the Y-axis is the evaluated cost function. The lines drawn vertically from the cost curve down to the X-axis indicate the start/end of a pass. Note that the first three passes have not been included in

the figure since these would dwarf the above curves as the algorithm is mostly performing rapid gradient descent initially.

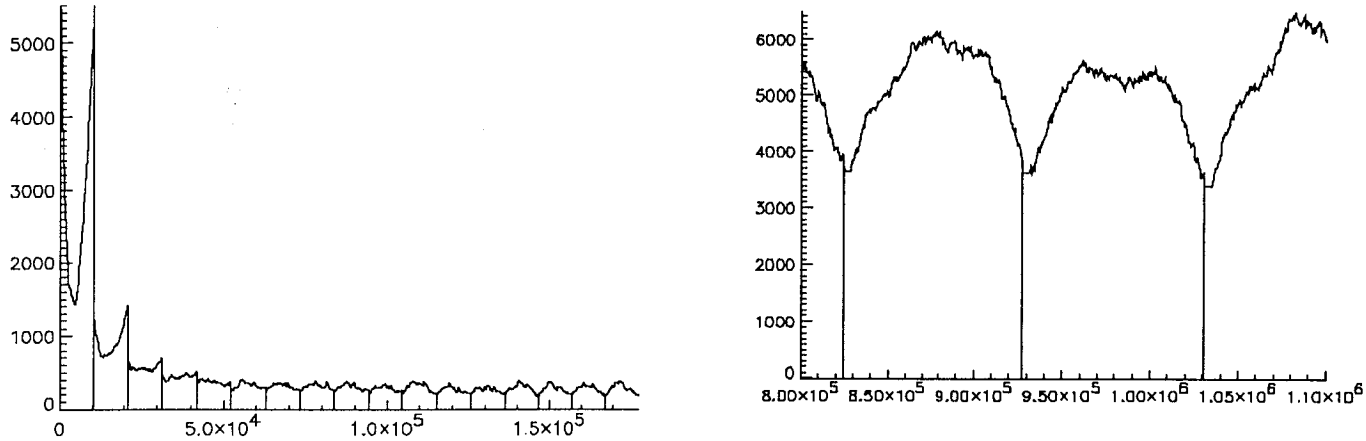


Figure 10. Behavior of cost during FM

What is most interesting about the figure is the inverted "U" shape of the costs during each later pass. The first pass or two (not shown) are "U" shaped as the algorithm does rapid gradient descent. Note that the end point of each pass is identical in cost to the starting point, due to the symmetry of swapping every cell. The inverted "U" shape means that the algorithm invariably chooses the best partition from near the beginning or the end of the pass. This makes sense if one considers that the algorithm is merely making local changes towards the end of a run, and the local changes therefore lie near the beginning or end of the pass. This raises the question, what does the effort for the middle 95% of the pass achieve? We refer to a *full pass* as one that eventually locks every node during its course. In view of this we made the following hypothesis:

> The use of full passes in the later passes of the FM algorithm is serving the purpose of performing a local restart for the algorithm, and this goal can also be achieved by stochastic local moves.

We tested and partially confirmed this hypothesis in a series of experiments reported in [Buntine *et al.*, 1997]. What is most important about this is that it confirms the usefulness of visualization for analyzing the performance of the algorithm. No amount of studying of simple trace information or final results would have yielded this. Note, however, that the analysis proceeds as follows:

- We instrument the code in order to capture data of interest.
- We then pass that data to a powerful visualization/graphics system such as Matlab or IDL and then explore the data in this context.

Thus we need to be able to neatly instrument the algorithm. Moreover, notice that this can be done as a one-off. We do not need to monitor the algorithm dynamically. Rather data is gathered, and examined afterwards.

3.2 Dynamic visualization

Inspired by this success, we then proceeded to create a dynamic visualization. In this scenario, we connect the algorithm to a 3D visualizer written in OpenInventor 3D, a C++ API using OpenGL from Silicon Graphics, and attempt to view different facets of the optimization as the algorithm performs. This effort, while not producing any significant insights, did offer a number of technical challenges and made us realize a more concerted effort would be required. In this section, we briefly review the issues.

Process/implementation issues

First, we had some trouble hooking up the optimization algorithm with the visualization system. Visualizers typically work in a number of modes. Proprietary systems such as IDL or Matlab could interface with the optimization algorithm in one of four ways:

- They are run as the primary process and call the optimization algorithm as a subroutine. This requires installing call-backs so the subroutine can pass data back to the visualizer. This is a difficult process to manage because the optimization system cannot simply stop and restart procedures to return control to the top-level calling function every time data needs to flow back. We were able to use this approach to link our FM partitioning algorithm with OpenGL, but it required creating a full system for saving and restoring search states, something not realistic in general. A screenshot is given in Figure 11.
- They call the optimization algorithm as a sub-process, performing data transfer via pipes. This approach, while conceptually attractive in terms of interfacing with the optimization algorithm, one only now need install reads and writes to the pipe at

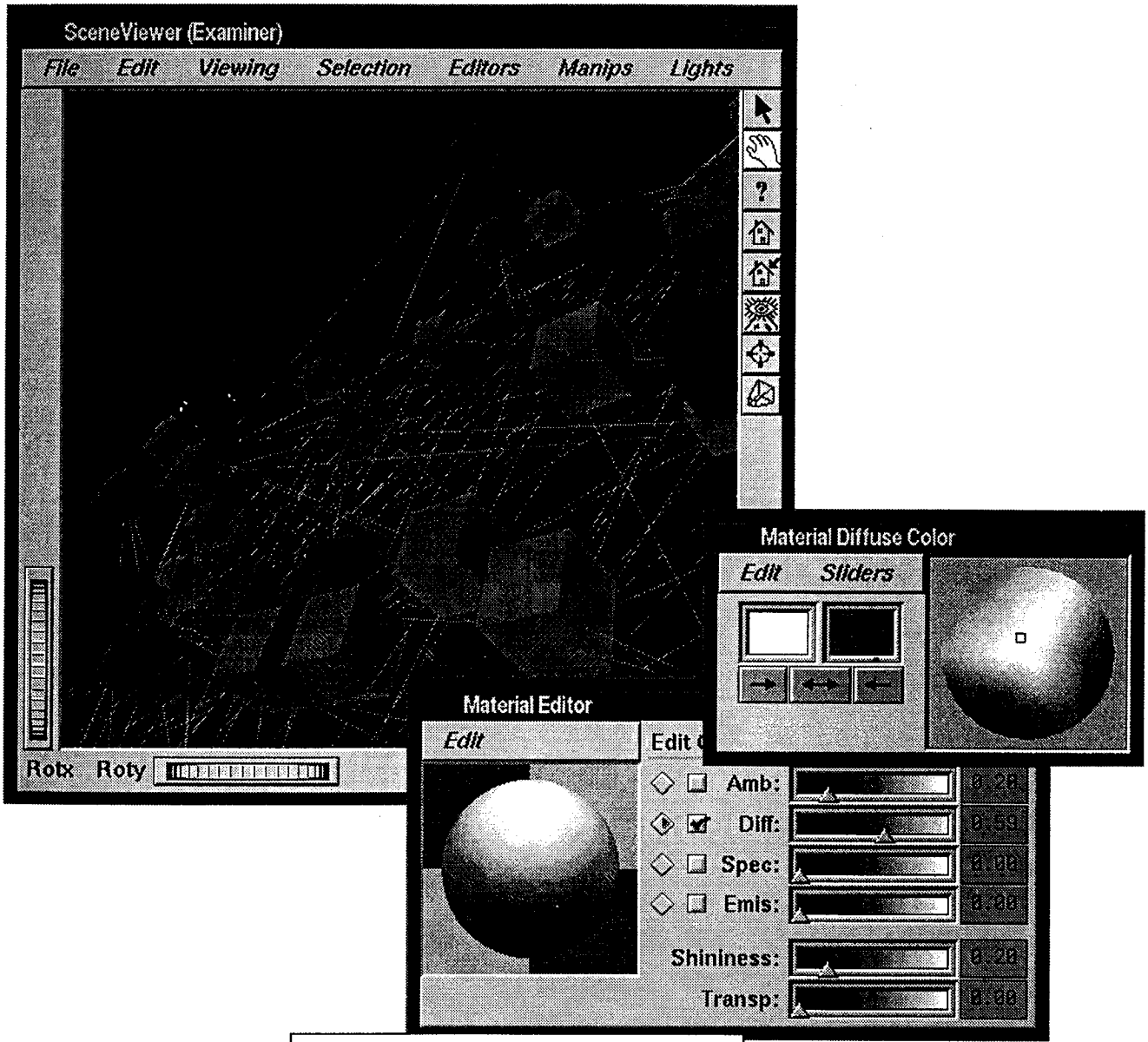


Figure 11. Visualizing nodes during FM

key places, ends up placing undue overhead on the optimization system. If one was visualizing a simple data stream, this would be easy, and in fact corresponds to the static approach done previously. However, the intent of dynamic visualization is to open up more of the internal data structures for inspection and thus visualize the inner workings of the algorithm. One could keep a full copy of the optimization/search data (values of all variables and associated heuristics) in the visualization environment for probing during the visualization, but this requires significant overhead in terms of programming data structures and message passing to maintain the local copy.

- Alternatively, the optimization algorithm is written with two threads, one for the optimization itself and one to service the visualization system by controlling flow and servicing data requests. The visualization system then operates as a primary process and interacts with the second thread of the optimization process, in effect controlling it.
- Finally, the visualization system operates as a sub-process controlled by the optimization system. We claim this is inappropriate because the visualization now operates passively.

Our preference therefore is the following: the optimization system has a second thread to service the visualization system, and both systems operate independently, loosely coupled by commands. However, these approaches still seemed lacking. During our development, we were continually going backwards and forwards between systems adding new data types, modifying the visualization methods, etc., as we attempted to see various features. The programming cycle here had a lot of redundancy in that the optimization system and the visualization system had some redundancy. We fault the general approach was slow and tedious. As a result, we began a new

3.3 Conclusions

The dynamic visualization system we developed enabled us to do some initial experiments before we abandoned it for a new approach discussed next. Our main bottleneck was the prototyping time required, and the tight interface needed between the visualization programmer and the optimization programmer. We felt frustrated with the speed of development, compared with static visualization where the optimization programmer merely dumps data and hands it over to the data analyst.

The main problem we encountered was the mass of nodes needing to be visualized. A medium sized circuit might have 10,000 nodes and these cannot be displayed on the screen. Therefore, we needed ways of better displaying large networks of constraints. One approach is to apply partitioning methods to break the problem up into pieces, as per hyper-graph partitioning. Our belief, however, is that standard local search algorithms and other fast algorithms such as MINCUT are simply not good enough at this to help. We spent considerable time with various versions of MINCUT applied to the OBDD minimization problem and on smaller problems obtained empirical support that the structure of the circuit, as reflected by the graphical decomposition, gave direction about the minimum-ordered OBDD. We now believe that OBDD minimization can be improved through some kind of graphical decomposition, however, we now need better tools for doing this.

We were able to develop a state of the art hyper-graph partitioning algorithm [Buntine *et al.*, 1997] based on experiences with static visualization. We got promising empirical results using dynamic visualization, however, were hampered by a range of implementation issues that lead us into a new direction.

4 The JavaTime Visualization system for component visualization

As a result of our experiences with the previous systems, we began a new project that takes a fresh cut at the problem of instrumenting algorithms and visualizing results. This is the JavaTime Visualization System, in development by Michael Shilman. The goal of the JavaTime Visualization package is to help systems designers gain insights into the structure, behavior, and performance of component-structured systems through new visualization techniques. The package combines an integrated user interface for constructing dynamic visualizations with a small but growing collection of modular visualization components targeted at systems development. Though it is still a work in progress, it has been successfully applied to several simple algorithms. The basic user interface concept is that of visual programming.

The user interface to the package allows the user to probe time-varying signals in the algorithm and flexibly display them. Back-end compiler infrastructure permits arbitrary instrumentation of user code to enrich the set of possible probes automatically. Code to monitor a particular variable at certain places can be automatically inserted and deleted, and hooked up to a variety of visualization types including simple displays, plots, etc.

In its current state of development, the system can instrument a piece of code or monitor signals in a component-structured system, and flexibly control the way the data is displayed. A JavaTime Snapshot of a JPEG decompression algorithm implemented in components is given in Figure 12. The three image windows correspond to probes, which are also seen in the component view.

4.1 Visualization Probes

Visualization probes are a user interface metaphor inspired by physical probes used in hardware design. A user can attach a probe to any point in the system programmatically or via a visual interface which is consistent with the component editing interface. Probes provide a convenient way to view partially computed results, communication between components, and other debugging and performance metrics from the code, based on user-created ports which can be easily added to the components.

In addition, JavaTime's code instrumentation support makes it easy to semi-automatically modify Java byte-codes to track low-level performance metrics. An instrumentor object can optionally add ports to the instrumented result, and these ports can be

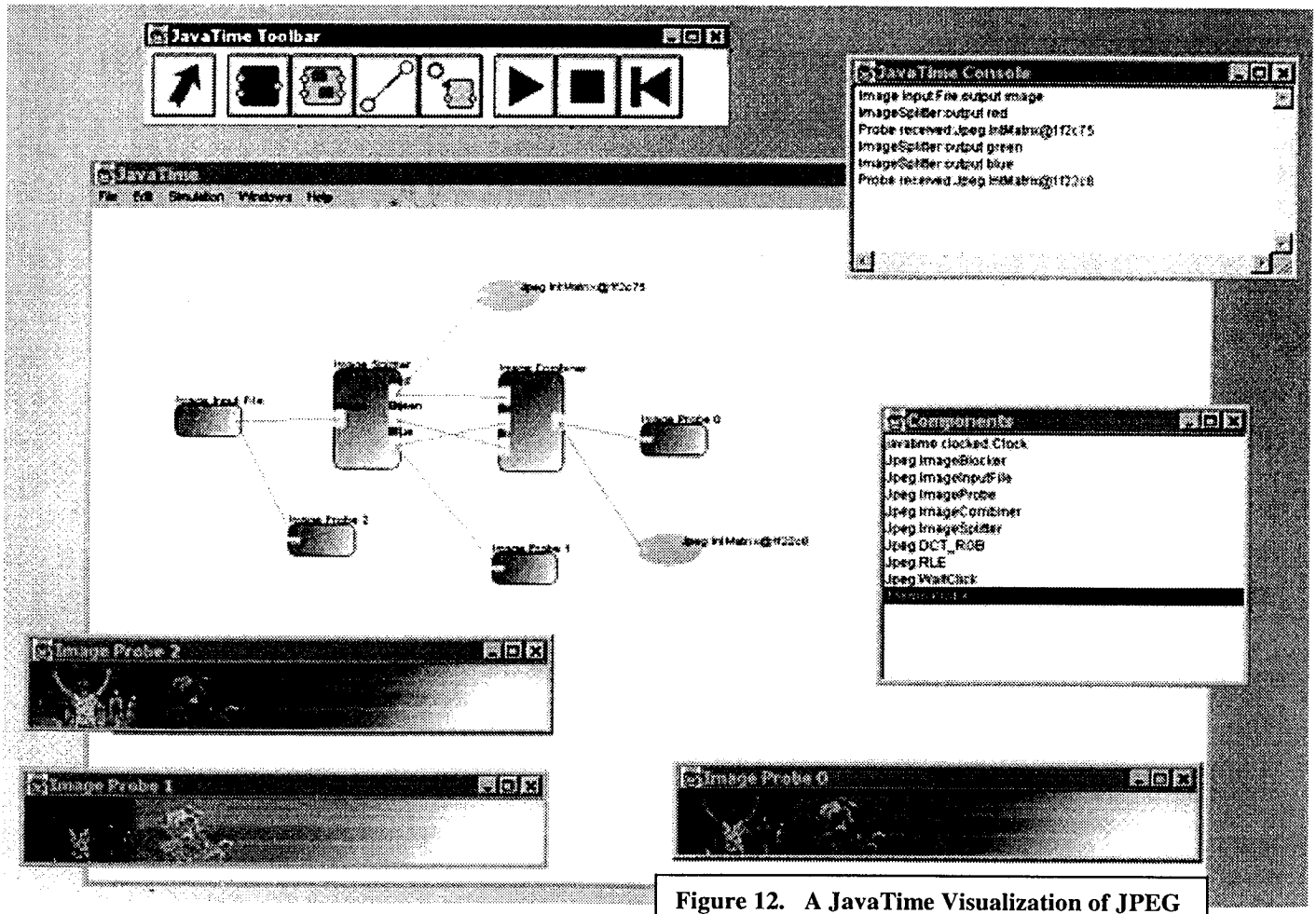


Figure 12. A JavaTime Visualization of JPEG

used to drive dynamic visualizations as the system is being run.

4.2 Component Libraries

Component libraries are the least mature part of the system, as the visualization infrastructure is still under development. The

idea is that there are a number of very general standard data types which we might want to *visualize*. These types might include multivariate data, graphs, trees, structured text, etc. By determining an expressive set of core data types, it will be possible to construct visualizations more rapidly.

4.3 Graph Drawing

The project grew out of an effort to better understand partitioning algorithms through animation and user-controlled filtering. We began by porting the original graph display and filtering code from its original OpenInventor 3D C++ API to 2D Java AWT, for portability. This also included a number of API enhancements which added more flexibility. For example the new package provides a way to efficiently annotate nodes with application-specific information for a simple way to mesh the API with existing Java applications.

This port has matured into a moderately stable graph drawing library. In addition to basic nested graph drawing, the package provides an assortment of visualization techniques ranging from animation to data filtering to graph clustering and layout. These

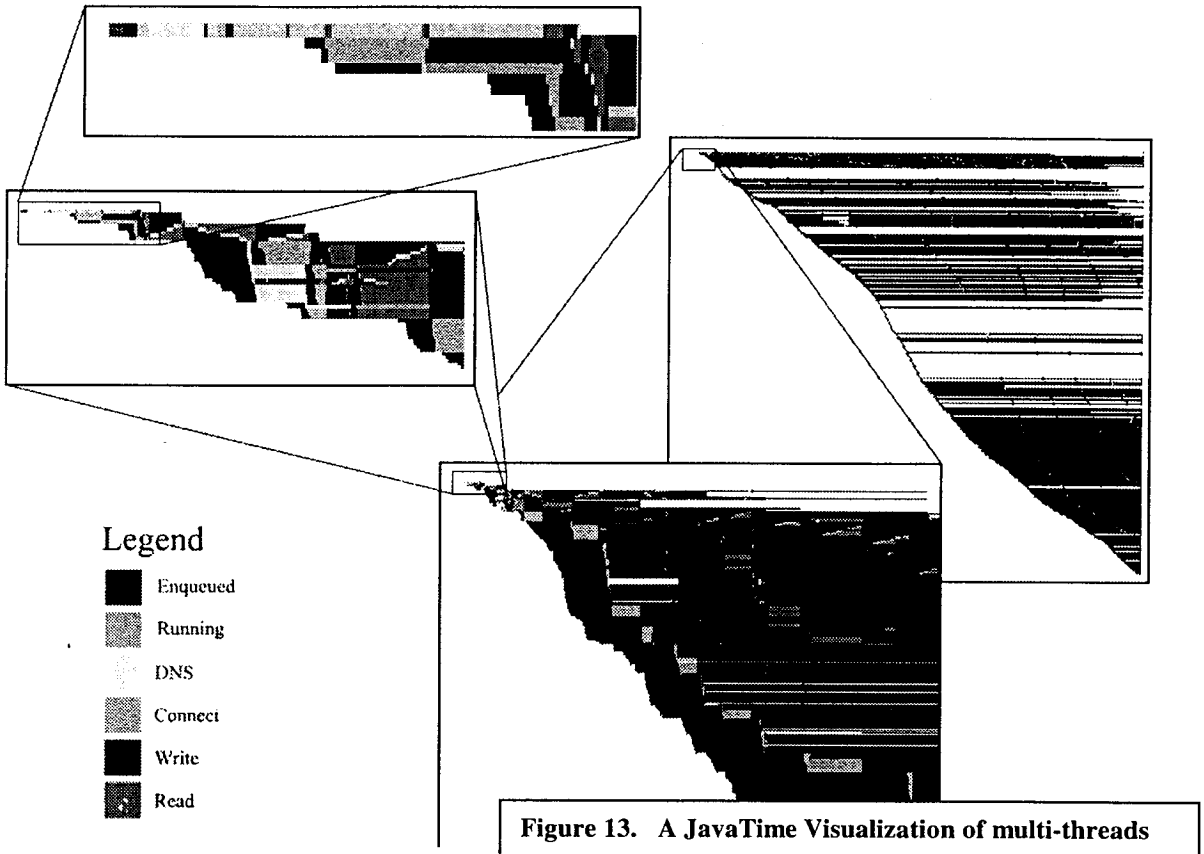


Figure 13. A JavaTime Visualization of multi-threads

techniques are combined with assorted graph editing capabilities which are made available to applications at multiple levels of abstraction.

4.4 Sample Visualizations

We have used the JavaTime system to visualize a number of performance metrics in system design to help debug and optimize systems under development. Our first success was in debugging/optimizing a functioning but slow system which constructed lightweight Java threads for highly-multithreaded data acquisition applications. The system transforms a restricted subset of multi-threaded Java programs into split-phase single-threaded programs.

The bulk of the project was writing the transformation and simply making it work, and this was done without the help of visualization. However, once the transformation was complete, we did not witness the performance improvements and scalability that we expected of the system when we applied it to a web crawling application. Rather than spending a bunch of time profiling the code, we hand-instrumented the code to spit out *markers* at certain points in the code, which showed us at what points in time

threads transitioned between key phases, both in the user application and in our underlying runtime system. For example, putting the thread on the ready queue is part of the thread runtime, the DNS lookup is part of the network library, and the HTML parsing is part of the application.

By visualizing these markers as *extents* over time, the system's performance was literally opened up before us. The dynamic visualization we developed here is shown in Figure 13. Within minutes we had determined the cause of the performance hit (our DNS timeout was too short), and had also optimized a number of other parameters at various points in the network library. Some of these visualizations have been *point tools* for specific tasks; recently we have been working on infrastructure that is applicable to general component systems, particularly in the areas of component communications and memory usage.

5 Configuring and tuning optimization systems

Any review article of a popular discrete optimization problem will mention an overwhelming variety of options one can code. We illustrate this below with hyper-graph partitioning, but analogous comments hold about SAT, any well studied optimization problem such as traveling salesman or constrain satisfaction, and to a lesser degree OBDDs. For instance, for hyper-graph partitioning there are multiple search spaces: moving a node across the partition (in FM), swapping two nodes (in K-L), moving a net in the dual graph, clustering two nodes together, clustering all nodes on a given net, and deleting a net entirely [Alpert and Kahng, 1995; Karypis *et al.* 1997; Alpert *et al.*, 1997]. There is also a range of options for performing kicks [Fukunaga *et al.* 1996]. For each move type, there are often multiple heuristics for selecting a move of varying complexity to compute. Moreover, moves could be local greedy (choose the first that improves cost) or local optimum (choose the best of all local moves). Researchers typically combine several of these options together to produce a single algorithm variation, and it is not uncommon for papers to report two, three, or even ten different variations being tested at once. This exploration of the algorithm variations reached a peak in the paper by Hauck and Borriello [1997] who managed to achieve excellent results by trying many combinations.

Methods of dealing with these many options in configuring an optimization algorithm, which we will refer to as the *configuration problem*, are confounded by three key issues for practical optimization:

- We only have a limited supply of benchmark problems to tune our systems with. In early speech recognition development, DARPA found the need to change benchmarks to prevent algorithms from being tuned so specifically to the particular benchmarks that their performance suffered in general.
- Components of search algorithms are such that it can be difficult to characterize their effect on the problem in terms of time and cost decrease other than empirically. Proofs and average case analysis when they exist can be invaluable [Purdom *et al.*, 1997].
- There is no general purpose, "superior" algorithm for any one problem domain [Wolpert *et al.* 1995]. Many NP-complete problems become average-case polynomial time depending on how you define your distribution of problems [Wang, 1997]. For SAT problems particularly, a particular application will usually produce problems over a certain distribution, for instance for stuck-at-fault test pattern generation [Larrabee, 1992]. To gain the advantages of average-time polynomial behavior, one needs algorithms that can adapt to the distribution of problems being presented.

How does one explore the full range of algorithm variations given a sparsity of benchmark data and an obligation to tune algorithms to the distribution of problems being encountered? Perhaps moves in the dual graph, considered to perform poorly as full passes [Alpert and Kahng, 1995] would make excellent kick moves? Perhaps the different move types should be used at different stages of the problem (for instance, during rapid gradient descent, or during exploration of wells)? For iterated local search, when should one perform a kick and when should one simply restart the entire search? The question of exploring algorithm variations applies equally well to clustering and thus scaling down a problem. How would one trade-off computation at different scales? When should scaling or clustering recurse? How could one combine the fixed clustering strategies based on connectivity measures with data-driven clustering?

With these kinds of concerns in mind, recent research in constraint satisfaction [Minton, 1996] and scheduling [Gratch and DeJong, 1996] demonstrates automatic configuration of search systems leads to improved performance, and suggests schemes for doing this. Minton uses simple code generation methods to create variations of backtracking algorithms and then runs empirical tests to find the best, and Gratch *et al.* use data-efficient approximate statistical tests to differentiate between different branches in pseudo-code. Another possibility is simply to leave a few real-valued parameters in the system and apply methods such as optimum control [Moore *et al.*, 1998] to find which setting yields average-case peak performance. Concepts such as bounded rationality and continual computing from decision theoretic computing provide a clear theoretical context for this kind of research [Horvitz, 1989; Russell, 1997; Horvitz, 1997]. Briefly, decision theory is the probabilistic analysis of the effect of taking

different actions performed in order to minimize the cost [Clemen, 1990]. Computation itself can be viewed as a cost, so the analysis of different computations and expected outcomes can be performed during computation. One would apply decision theory "in the large" to configure the algorithm and tune its parameters.

Finally, note that the JavaTime Visualization system with its visual programming interface is an ideal platform on which to implement these tools for decision theoretic control. Probes and such can be installed on components, and then the decision theoretic controllers added where needed.

6 Further research opportunities for applying data understanding to optimization

As part of our research, we have surveyed the field of optimization to understand ways in which learning could be used to gain advantage. This section briefly recounts some of the key opportunities we see.

Learning and clustering: We believe that clustering is a particular form of a broader capability that we shall refer to as *scaling* and outline in more detail subsequently. We claim the clustering technique has not seen application in other optimization problems such as satisfiability because heuristics for its use can be difficult to conceptualize, except in a few CAD applications such as place-and-route and partitioning where simple spatial or neighborhood heuristics apply. There is no coherent theory of how to find a good clustering of a problem, and current methods are limited to classes of problems where intuitions about locality hold. The initial work by Hagen and Kahng [1997] suggest learning good clusters is feasible, and opens the door for the application of rigorous learning methods.

Combining and learning heuristics: While isolated cases of learning or tuning heuristics exist, we know of no thorough study encompassing state of the art algorithms that takes a variety of definitions of a heuristic and maps them onto a learning problem which allows efficient evaluation. There is a need to establish a theory under which different kinds of heuristics can be efficiently learnt from data, and multiple heuristics combined. Because existing problems provide a source of prior knowledge about the next problem encountered, Bayesian decision theory, the statistical theory for incorporating prior knowledge [Bernardo and Smith, 1994] could calibrate and combine existing best-practice search heuristics and optimally combine them with probing information from the current search (such as local minima or failed paths) to develop more computationally economical methods of estimating properties of the global minimum.

Opportunities for learning and data-mining are barely explored: Some significant successes exist [Samuel 1963; Tesauro, 1994; Baluja *et al.*, 1998; Boyan *et al.* 1998], yet we contend that there remain significant untapped opportunities for using learning more generally to improve optimization algorithms. Careful empirical studies of a variety of different learning scenarios are needed, crafted with inside knowledge of state-of-the-art algorithms. In the project we would investigate the many rich families of multivariate learning algorithms available to us [Buntine, 1996; Ripley, 1996] and apply learning methodology in a rigorous manner. On this project, the positioning of a data-mining/learning expert in an applied optimization (CAD) department makes this uniquely possible.

Tuning algorithms: In many cases, the top level of optimization algorithms could be configured and tuned automatically. Limited research shows this is possible, but there is no thorough study of the range of methods required, and general methods have yet to be explored. We claim the automatic configuration and tuning of optimization algorithms may soon may be necessary to attain state-of-the-art performance. A research challenge is to develop a simple palette of techniques that provides good coverage of problems encountered in practice. We need to make developers aware that this capability is realistic. Once they start using it, we claim it will become necessary.

Probabilistic modeling: Careful probabilistic analysis such as is done for backtracking with SAT [Purdom *et al.*, 1997] and "go with the winners" [Aldous *et al.*, 1994] yields significant insights into the process. We would also like to develop a noise/probability model for local minima in order to better understand the big valley effect. We need to place these in the context of a realistic but simplified search problem/algorithm.

7 Conclusions

We have explored methods for performing dynamic visualization of intermediate optimization results, and based on our experiences, we have proposed and prototyped a system called JavaTime Visualization. These system is intended to aid prototyping. Moreover, we have understood some of the issues involved with clustering and problem decomposition, used both to improve optimization performance and to aid visualization. We see these techniques as being crucial tools for subsequent work. We have surveyed current opportunities for applying learning technology to optimization and arrived at a number of

scenarios. We explored one of those, "learning to cluster," and achieved significant results. Another of these opportunities is decision theoretic control at the large scale of optimization algorithms. We note that the implementation of this dovetails nicely into our JavaTime Visualization system. Thus, in this research effort, we have explored and proposed a comprehensive set of tools for integrating visualization, learning, and decision theoretic control of optimization.

References Cited

D. Aldous and U. Vazirani, "Go with the winners' algorithms." In *Proc. 35th Symp. Foundations of Computer Sci.*, pages 492-501, 1994.

C.J. Alpert, J.-H. Huang and A.B. Kahng, "Multilevel Circuit Partitioning." In *Proc. Design Automation Conference*, p 530-533, Anaheim, CA, 1997.

C.J. Alpert and A.B. Kahng , "Recent directions in netlist partitioning: a survey." In *Integration, the VLSI journal*, **19** pages 1-81, (1995).

S. Baluja and S. Davies, "Fast Probabilistic Modeling for Combinatorial Optimization." In *Fifteenth National Conference on Artificial Intelligence* (AAAI), 1998.

R. Battiti and M. Protasi, "Approximate Algorithms and Heuristics for MAX-SAT." In D.-Z.-Zu and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization (Vol 1)*, pages 77-148, Kluwer, 1998.

E. B. Baum and W. D. Smith, "A Bayesian Approach to Relevance in Game Playing." In *AI Journal*, **97**(1), p195, 1997.

J.M. Bernardo and A.F.M. Smith, *Bayesian Theory*. John Wiley, 1994.

K. D. Boese, A. B. Kahng and S. Muddu, "New Adaptive Multistart Techniques for Combinatorial Global Optimizations", *Operations Research Letters* **16**(2), pages 101-113, 1994.

J.A. Boyan, *Learning Evaluation Functions for Global Optimization*. PhD Dissertation from the Comp. Sc. Dept at CMU, pending 1998.

J.A. Boyan and A.W. Moore, "Learning Evaluation Functions for Global Optimization and Boolean Satisfiability." In *Fifteenth National Conference on Artificial Intelligence* (AAAI), 1998.

R.E. Bryant, "Symbollic Boolean Manipulation with ordered Binary-Decision Diagrams." In *ACM Computing Surveys*, **24**(3), 1992.

R.E. Bryant, "Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification." In *IEEE Int. Conf. on CAD*, 1995.

W.L. Buntine, "Operations for Learning with Graphical Models. " In *Journal of Artificial Intelligence Research (JAIR)*, **2**, December, 1994.

W.L. Buntine, "Graphical models for discovering knowledge. " In *Advances in Knowledge Discovery*, edited by U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy, MIT Press, 1995.

W. L. Buntine, "A guide to the literature on learning probabilistic networks from data." In *IEEE Trans. On Knowledge and Data Engineering*, **8**(2), pages 195—210, 1996. At http://www.ultimode.com/~wray/refs.html.

W. L. Buntine, L. Su, A. Richard Newton and A. Mayer, "Adaptive Methods for Netlist Partitioning." In *IEEE Int. Conf. on CAD*, 1997.

R.T. Clemen, *Making Hard Decisions*. PWS-KENT Publishing, Boston, 1990.

J.M. Crawford and A.B. Baker, "Experimental results on the application of satisfiability algorithms to scheduling problems. " In *Proc. Of the 12th National Conf. On AI (AAAI-94)*, pages 1092-1097, 1994.

S. Dutt and W. Deng, "A probability-based approach to VLSI circuit partitioning." In *Proc. Design Automation Conference*, 1996.

T. Ellman and J. Keane and M. Schwabacher and K.-T. Yao, "Multi-level modeling for engineering design optimization." In *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* **11**(5), 1997.

A. Fukunaga, D. J.-H. Huang, and A. B. Kahng, "Large-Step Markov Chain Variants for VLSI Netlist Partitioning." *Proc. IEEE Intl. Symp. on Circuits and Systems*, May 1996, pp. IV/496-499.

F. Glover. Tabu search - part I. *ORSA Journal on Computing*, 1989.

J. Gratch and G. DeJong, "A decision-theoretic approach to adaptive problem solving." In *Artificial Intelligence*, **88**(1-2), pages 365-396, 1996.

J. Gu, P.W. Purdom, J. Franco, and B.W. Wah, "Algorithms for the Satisfiability (SAT) Problem: A Survey. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, 1997, pages. 19-152.

L. Hagen and A. B. Kahng, "Combining Problem Reduction and Adaptive Multi-Start: A New Technique for Superior Iterative Partitioning", *IEEE Trans. on CAD,* **16**(7) (1997), pages 709-717.

S. Han, D. Boley, M. Gini, R. Gross, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, J. Moore, "WebACE: A Web Agent for Document Categorization and Exploration." *Autonomous Agents'98 Conference*, Minneapolis, May 1998.

O. Hansson and G. B. Holt and A. Mayer, "Toward the Modeling, Evaluation and Optimization of Search Algorithms." D. Brown and C. White editors, *Operations Research and Artificial Intelligence*, Kluwer, Boston, 11-28, 1990.

S. Hauck and G. Borriello, "An Evaluation of Bipartitioning Techniques." In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **16**(8) pages 849-866, August 1997.

E. Horvitz, "Reasoning about Beliefs and Actions under Computational Resource Constraints." In L. Kanal, et al. ed., *Uncertainty in Artificial Intelligence 3*, Elsevier, 1989, pages 301-324.

E. Horvitz, "Models of Continual Computation." In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, July 1997.

E. Horvitz and A. Klein, "Reasoning, Metareasoning, and Mathematical Truth: Studies of Theorem Proving under Limited Resources." In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, August 1995.

D.S. Johnson, "A Theoretician's Guide to the Experimental Analysis of Algorithms." Unpublished report available from the author's web-site (http://www.research.att.com/~dsj/papers.html), 1996.

D.S. Johnson and L.A. McGeoch. The travelling salesman problem: A case study in local optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*. John Wiley and Sons, 1995.

G. Karypis, R. Aggrawal, V. Kumar and S. Shekhar , "Multilevel Hyper-graph Partitioning: Application in VLSI Domain." In *Proc. Design Automation Conference*, p 526-529, Anaheim, CA, 1997.

S. Kirkpatrick, Jr., C. D. Gelatt and M.P. Vecchi, "Optimization by simulated annealing. " *Science*, 220(4598):671-680, May 1983.

T. Larrabee. "Test pattern generation using Boolean satisfiability." *IEEE Transactions on Computer-Aided Design*, pages 4-15, January 1992.

J. Li, J. Lillis, and C.-K. Cheng. Linear decomposition algorithm for{ VLSI design applications. In *IEEE Int. Conf. on CAD*, pages 223-228, 1995.

S. Mallela and L.K. Grover, "Clustering-based Simulated Annealing for Standard Cell Placement." In *Proc. Design Automation Conference*, p 312-317, 1988.

T. Maridou, P.M. Pardalos, L. Pitsoulis, and M.G.C. Resende, "Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search, and GRASP." In Proceedings of the Workshop on *Parallel Algorithms for Irregularly Structured Problems,* Lyon, France, 1995..

O. Martin, S.W. Otto, and E.W. Felten, "Large-step Markov chains for the TSP incorporating local search techniques." *Operations Research Letters*, **11**:219-224, 1992.

S. Minton, "Automatically Configuring Constraint Satisfaction Programs: A Case Study." In *Constraints*, **1**(1), 1996.

W. Moore, J. G. Schneider, J. A. Boyan and M. S. Lee. "Q2: Memory-Based Active Learning for Optimizing Noisy Continuous Functions." In *Machine Learning: Proceedings of the Fifteenth International Conference (ICML)*, 1998.

R. Ponnusamy, N. Mansour, A. Choudhary, and G. Fox. Graph Contraction for Mapping Data to Parallel Computers: A Quality-Cost Trade-off. In *Journal of Scientific Computing*, **3**(1), pages 73-82, 1994.

B.D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press, 1996.

S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

S. Russell and E. Wefald, "Principles of Meta-Reasoning." In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario: Morgan Kaufmann, 1989.

S. Russell and E. H. Wefald, *Do the Right Thing: Studies in Limited Rationality*. Cambridge, MA: MIT Press, 1991.

S. Russell, "Rationality and Intelligence." In *Artificial Intelligence*, 1997.

A.L. Samuel, "Some studies in machine learning using the game of checkers." In, E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York, 1963.

B. Selman, H.A. Kautz, "Knowledge Compilation and Theory Approximation." *JACM* **43**(2), 193-224, 1996.

B. Selman, H. Kautz, and B. Cohen. "Local search strategies for satisfiability testing." In *Cliques, Coloring, and Satisfiability: Second DIMACS implementation Challenge,* Amer. Math. Soc., 1996.

B. Selman, H. Kautz, and D. McAllester, "Computational Challenges in Propositional Reasoning and Search." In Proc. of *IJCAI-97.*

J. Shi and J. Malik, "Normalized Cuts and Image Segmentation." In *IEEE Conf. Computer Vision and Pattern Recognition*, June 1997.

W.-J. Sun and C. Sechen, "Efficient and Effective Placement for very Large Circuits." In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **14**(3) pages 349-359, 1995.

R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

G. Tesauro, "TD-Gammon,a self-teaching backgammon program achieves master-level play." In *Neural Computation*, **6**(2), pages 215-219, 1994.

J. Wang, "Average-Case Computational Complexity Theory." In *Complexity Theory Retrospective II*, L. Hemaspaandra and A. Selman, editors, Springer, pages 295-328, 1997.

D.H. Wolpert and W.G. Macready, "No Free-Lunch Theorems for Search." *Santa Fe Institute 1995 Working Paper* No. 95-02-010.