Sift and Sort: Climbing the Semantic Pyramid

H. Van Dyke Parunak, Peter Weinstein, Paul Chiusano, Sven Brueckner

Altarum Institute
3520 Green Court, Suite 300, Ann Arbor, MI 48105
{van.parunak,peter.weinstein,paul.chiusano,sven.brueckner}@altarum.org

Abstract. Information processing operations in support of intelligence analysis are of two kinds. They may sift relevant data from a larger body, thus reducing its quantity, or sort that data, thus reducing its entropy. These two classes of operation typically alternate with one another, successively shrinking and organizing the available data to make it more accessible and understandable. We term the resulting construct, the "semantic pyramid." We sketch the general structure of this construct, and illustrate two adjacent layers of it that we have implemented in the Ant CAFÉ.

1. Introduction

In moving from raw data to finished policy recommendations, intelligence analysis must address two fundamental problems. First, the volume of available data is far more than can be processed by a policymaker, and must be reduced. Second, each item of data is susceptible to multiple interpretations, and must be related to other data to increase its semantic content.

Typically, these processes alternate. Sifting removes less relevant data. That which remains is sorted to increase its semantic content. Then further sifting reduces the data even more, permitting more sophisticated sorting. Eventually, a broad base of semantically poor data becomes a much smaller collection of data with rich associated semantics.

We motivate this view of analysis processes theoretically and illustrate it in Section 2. Then we use this distinction to describe a novel method for information retrieval in Section 3. Our sorting and sifting mechanisms are inspired by techniques used by ants to sort their nests and forage for food. Section 4 offers experimental evidence for the effectiveness of our mechanisms, and Section 5 concludes.

2. The Semantic Pyramid

The alternation of sifting and sorting is driven by a fundamental constraint from complexity theory: as the intricacy of analysis increases, the volume of data must decrease to permit timely processing.

2.1. Computational Complexity

An important branch of theoretical computer science classifies various algorithms into different complexity classes [8]. This theory offers the following insights, which are critical for information processing in support of intelligence analysis.

The amount of time (computer cycles) or space (computer memory) needed to solve a problem depends on two things: the structure of the problem (finding records in a database vs. proving a logical theorem) and the size of the problem instance (processing a database of 10^3 records vs. one of 10^9 records). As one might expect, the larger the problem instance, the more time and/or space is needed to solve it. Less obviously, the mathematical function that translates the size of a problem into the time or space needed to solve it depends on the algorithm used to solve it, and thus on the nature of the problem. If n is the size of a problem, the time required to solve the problem might scale (for example) as log(n), or n^2, or n^{10000}, or e^n, or n!, depending on the structure of the problem.

Not surprisingly, processes that we think of as nontrivial (e.g., proving logical theorems) tend to be more sensitive to the size of the problem than simpler processes (e.g., finding the string "abc" in a document). Intelligence analysis requires processes across the complete range of such complexity. Complexity theory warns us that if we want answers in a timely fashion, we must limit the size of the problems that we ask these processes to solve, and that this limitation must be more severe for the more complex processes than for the simpler ones.

2.2. Climbing the Pyramid

Intelligence analysis typically begins with voluminous low-level data, and ends with closely reasoned analyses that support policy makers. Complexity theory shows that it is probably impossible to solve many problems perfectly in a reasonable amount of time. Thus, we must be satisfied with approximate rather than perfect answers. Furthermore, even approximate methods bog down when given problems that are too large. For example, the complex logical analysis that is required at the higher levels of analysis can only be applied to a small body of data. Thus sifting, or removing irrelevant or low-priority data, is an essential step in moving from raw data to finished product.

In general, we can identify at least four levels in the process, summarized in Figure 1.

At the lowest level, filtering (a sifting operation) applies very efficient techniques to separate a relatively small proportion of incoming data as potentially interesting. Ideally, the filtering process would touch every piece of incoming data. In practice, however, it may be necessary
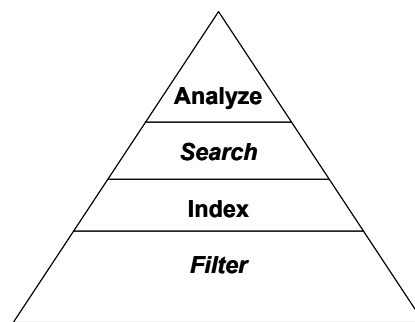

Figure 1. The Semantic Pyramid.— *Italicized* **processes sift; nonitalicized processes sort.**

to sample the input stream instead, with the intensity of sampling increasing in response to positive tests.

Sometimes, filter tests can be direct, for example, looking for the presence of certain terms. In other situations one must look for anomalies – deviations from expectations. (Anomaly detection does, however, require presorting: for example, to know that the data represents credit card purchases of individuals with purchasing histories).

Filtered data is then indexed, to enable it to be accessed quickly. The indices used by current internet search engines, for example, record the presence of words in each document. Semantic indexing, in comparison, strives to capture the meaning of words, phrases, sentences, and possibly other levels of meaning expressed by the data. Sorting processes like indexing do not reduce the size of the data, but make it more orderly, in order to support higher levels of processing.

Search uses the organization imposed by indexing to sift out the items of data that most closely match the analyst's requirements.

Analysis manipulates the smallest set of data: not the full volume of raw data, nor even the filtered data, but the subset that matches the analyst's current interests and requirements. It is another sorting process, detecting logical relationships among different assertions, constructing hypotheses, and verifying or refuting them. This level of analysis is the most complex computationally, and requires the previous levels of processing to reduce the amount of data to be handled.

These four levels are only illustrative. Closer analysis reveals further levels of distinct operations. Our fundamental claim is that however deep one carries this analysis, one will find an alternation of sifting and sorting, reducing the volume of data, then applying organizing processes to what remains. The structure provided by each sorting process enables more complex sifting at the next level, while the reduction in volume at each level of sifting permits application of more complex sorting algorithms at the next level.

3. Sorting and Sifting in Ant CAFÉ

We have been developing a system, called the Ant CAFÉ, that applies these concepts in doing information retrieval from massive docu-bases [16]. In this section we summarize the high-level behavior of the Ant CAFÉ, then discuss how it retrieves information of likely interest to the analyst (sifting) and how it organizes data to make this retrieval more efficient (sorting).

3.1. The Ant CAFÉ Feedback Loop

Figure 2 shows the basic concept of the Ant CAFÉ. The system has two main components. At the top of the figure, the Analyst Modeling Environment (AME) [1] maintains symbolic models at several
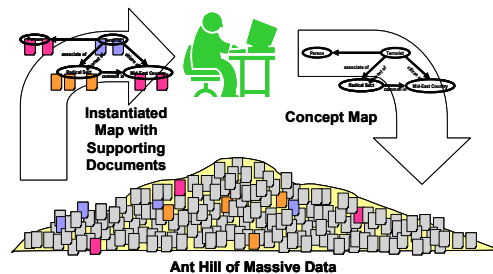

Figure 2. Ant CAFÉ Concept.

levels.

- Community models represent the interest of a group of analysts working on a common topic. They are the most general models and change the least rapidly.
- Tasking models represent a specific tasking assigned to an analyst.
- Analyst models represent the current state of an analyst's interests and hypotheses.
- Query models reflect the immediate information that an analyst wishes to access.

The AME constructs and maintains these models in response to observations of how the analyst interacts with information. Except for the query model, the analyst does not formulate these models explicitly. Currently, we represent the models as concept maps [4].

The models produced by the AME are passed to another component of the Ant CAFÉ, the Ant Hill (bottom of Figure 2). In contrast to the classic machine learning mechanisms of the AME, the Ant Hill uses techniques inspired by insect societies [12]. These techniques offer several benefits compared to more traditional algorithms:

- They are intrinsically distributed and decentralized. Thus the system can be scaled to deal with massive data by distributing it across multiple processors.
- Their processing model is dynamic, not query-driven. A query system does processing only when queried, and typically acts on a static system state. A dynamic system processes continuously, not just when a query is received, and continues to run while the system changes.
- They are anytime rather than input-process-output (IPO). An IPO system provides no information until the final answer is ready. An anytime system quickly produces approximate answers, and gives more refined information the longer the user waits.
- They are stochastic, driven by random sampling of data rather than complete enumeration. Thus users can dynamically trade accuracy against processing time by modulating the degree of coverage applied to the data.

This paper describes two such techniques. Dynamic granularity ant clustering (sorting) is inspired by the way that ants cluster the contents of their nests. Information foraging (sifting) is inspired by the pheromone mechanisms that ants use to construct paths between their nests and food sources.

3.2. Dynamic Granularity Ant Clustering

Ants sort the contents of their nests into piles of similar items [5]. As ants wander around, they maintain a short-term memory of the kinds of things they have recently encountered, and each time they encounter an object, they assess its similarity to other recently seen objects. An ant tends to pick up objects that are dissimilar to their surroundings, and to deposit objects that are similar to those in the ant's current environment. Over time, this distributed algorithm yields global clusters of high homogeneity.

Previously [13, 14], we eliminated the distinction between ants and documents, and gave each document the ability to move itself, based on its perception of its environment. Like the original algorithm, this refinement partitions the set of documents into

internally homogeneous sets, but does not provide any structure to guide subsequent retrieval operations.

Recently, we have developed a hierarchical version of this algorithm that can be distributed across multiple processors. Dynamic Granularity Ant Clustering (DGAC) repeatedly travels down its current hierarchical structure looking for nodes (sets of documents) that are internally cohesive, and moving these nodes to locations where they fit better. The resulting hierarchy reduces the complexity of subsequent retrieval.

The similarity function that drives the clustering is derived from the community model constructed by the Analyst Modeling Environment. In a previous filtering stage of processing, each document is indexed against the concepts represented in the community model. A document is considered to match a concept if it contains a morpheme that is subsumed by the concept in WordNet [7, 11]. The similarity of two documents is then measured by the standard cosine measure between their respective concept vectors.

Figure 3 illustrates the three main steps of this algorithm on a single machine. In the Figure, documents (at the bottom) are distinguished from higher-level nodes (circles), but the algorithm treats them all as "nodes." Each node maintains an estimate of its current cohesiveness in [0, 1] (the average pair-wise similarity across all documents that it currently subsumes) and a summary of its documents that it can use for comparing its similarity to other nodes (say, a vector average of its documents' concept vectors). The system also maintains a threshold $\theta \in [0,1)$, initially 0, whose function is similar to that of temperature in simulated annealing [10]. The cohesiveness of a document is 1, and its summary is its concept vector.

At any given cycle, one node is active. In Figure 3a, the current node is document H. In the "select" step, the node compares its cohesiveness c with θ. If $c <= \theta$, the node selects one of its children stochastically, favoring those that are least cohesive, and activates it. (Because document nodes have c = 1, this case never applies to them.) If c > θ, the node leaves its current parent and is attached to its grandparent (A in Figure 3b). (In doing so, it is moving the entire sub-tree that descends from it.) Then it randomly chooses one of its new siblings (B in Figure 3b) and estimates whether a merger with this sibling would yield coherence greater than θ. If so, it merges with that sibling under a new node (K in Figure 3c), measures the actual similarity σ achieved, and updates θ with the Q-learning rule $\theta_{t+1} = (1 - \alpha)\theta_t + \alpha\sigma$ [15]. (α, the learning rate, is an arbitrary parameter in [0,1]. Our experiments use $\alpha = 0.02$.) Then the root becomes active, and the process repeats.

Initially, when the threshold θ is low, nodes readily climb up the tree and seek new partners. As θ increases, it becomes more difficult for nodes to relocate, leading the system to stabilize.

Hierarchical clustering is not new [2, 9], but the standard approaches have several limitations that this algorithm
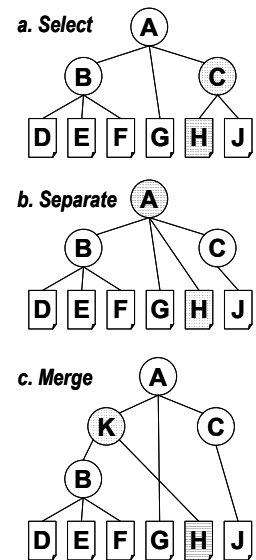

Figure 3. Dynamic Granularity Ant Clustering.

avoids:

- It does not require a centralized similarity matrix over all data items, and thus can readily be distributed across many machines. (In this case, each machine maintains its own θ, and periodically normalizes this value with those on other machines.)
- The population of data items can change dynamically as the algorithm runs, without restarting. The algorithm discovers misplaced documents and moves them dynamically. The similarity estimated when doing the merge can be less than that measured after the merge, particularly if the summary used in the estimation has been invalidated by the addition of new documents. So θ can decrease, permitting previously settled sub-trees to move.
- The similarity function used in the clustering operation (that is, the set of concepts attested in the community model) can also be changed incrementally without restarting the system. To the degree that the new function overlaps the old, the system reuses existing structure that has been constructed.
- Traditional hierarchical clustering is monotonic: once two nodes have merged into a higher-order node, they cannot separate again. This algorithm permits nodes to dissolve as well as to join, enabling the system to search continuously for the best fit to the data.
- This algorithm is stochastic, since node movements are evaluated by comparing a sample of their leaf nodes. As a result, users can trade accuracy against processing time.

Clustering runs continuously against the community model, even when there is no query from the user. It continuously organizes the underlying body of data to take account of changes in both the set of documents available and the slowly evolving community map.

3.3. Information Foraging

Ants construct efficient paths between their nests and food sources by depositing chemicals (pheromones) on the ground whenever they are carrying food, and by climbing the gradient of these chemicals whenever they are searching for food. As many ants discover food and deposit pheromones, discrete pheromone paths emerge that guide the ants' otherwise random movements toward food sources.

Dynamic Granularity Ant Clustering organizes sets of documents into hierarchical clusters. Each document is a leaf in the tree, and the similarity of documents under each node increases as one descends the tree from the root to the leaves. In information foraging, an analyst model or query model generates a stream of forager agents. Each forager agent begins at the root and descends the tree until it reaches a leaf. Then it evaluates a function that computes the relevance of the document that it has found to the higher-level query. This relevance score is deposited at the leaf, and propagates back up the tree toward the root, combining with any other relevance deposits from foragers representing the same model, and diminishing in strength with each step. As successive foragers descend the tree, they select their path at each node stochastically by evaluating a Boltzmann-Gibbs distribution weighted by the rele-

vance scores at each of the accessible next steps. The relevance scores function like ant pheromones, building up paths for later foragers to follow.

In general, searching for a data item in a population of size N requires time on the order of N (look at each item in turn until you find the one you want). If the items can be ordered linearly by their relevance, one can do the search in time logarithmic in N, but a single linear order is not realistic for most documents of interest to intelligence analysis. In our foraging system, the maximum length of the relevance path to documents of interest is the depth of the tree, which is logarithmic in the total number of documents (where the base of the logarithm is the mean branching factor at each node). Thus we achieve searching efficiencies comparable to those for linearly ordered data, even for data that cannot be usefully constrained to a total order.

The cost of this efficiency is constructing the hierarchical clustering of documents in the first place. Thus the sifting process of information foraging requires a preceding sorting process of constructing a hierarchical structure that will support the foraging. The Dynamic Granularity Ant Clustering algorithm described in Section 3.1 provides this structure.

4. Experimental Results

We have explored the potential of information foraging and hierarchical ant clustering with two sets of experiments. Both are based on a static set of 500 documents drawn from the CNS database [3] and hierarchically clustered by similarity. In principle, this kind of algorithm scales very well. We have used more mature ant clustering techniques to organize a dynamically changing population of more than 100,000 documents on a cluster of 16 processors [13, 14], but those techniques produced only a partitioning of the data, not a hierarchy, and so do not support the interactive experiments with foraging that we report here. We are currently preparing an experiment that will demonstrate a refinement of the hierarchical clustering algorithm to a document population on the order of 10^4.

4.1. Effectiveness of Foraging

To set up the experiment, we compute each document's relevance to the query and normalize this value so that the sum of relevance is 1. We compare the foraging process with a random process. While this benchmark may seem an unfair comparison, it is quite widely accepted in the pattern recognition community in the context of ROC (receiver operating characteristic) curves, a performance visualization to which our analysis is closely related [6]. At each step, the process under study selects a document, adds its relevance to the process's current total relevance, and then removes it from the population. Thus both processes begin with a relevance of 0, and end with a relevance of 1 (having seen all documents). In the random process, the documents are selected at random. In the foraging process, each successive forager takes advantage of the relevance paths laid down by previous foragers.

Figure 4 compares how the two processes accumulate relevance. Initially, before relevance paths develop, the processes are comparable. Then the rate of accumulating

relevance accelerates dramatically for the foraging process, as subsequent foragers take advantage of the paths developed by previous ones.

4.2. Effectiveness of Clustering

To evaluate our clustering method, we use an artificial retrieval benchmark, called "seekers," that can readily be

Figure 4. Progress of Clustering.

applied periodically during clustering. For each document in the population, we prepare 9 seekers with the same concept vector. Seekers follow the clustering hierarchy from the root to a leaf, following the branch at each level with the greatest similarity to a sample of leaves. The seeker metric is the proportion of seekers that terminate their search at a document that has a similarity of at least 0.95 with their concept vector.

Figure 5 shows how seeker success (square symbols) increases as clustering progresses. (In this example, the collection of documents and the concept vector are static.) Like many stochastic processes with emergent behavior, the system exhibits a phase shift. Before 45000 cycles, there is little apparent improvement in seeker success. After 48000 cycles, seeker success jumps dramatically. The second series in the plot (lozenze symbols) shows the number of immediate descendants of the root node in the emergent hierarchy. At step 0, every document is a child of the root, but as the algorithm progresses, more structure develops below the root, with a corresponding decrease in the number of immediate children of the root. This decrease continues until 48000 cycles, when the multiplicity of the root levels off at an average of 3.7.

The stabilization of the number of root children, and the corresponding increase in the seeker success, show that the data hierarchy has reached a stable configuration that captures the intrinsic structure of the data. Because the clustering process is dynamic rather than query-driven, it can achieve this stable condition in the back-
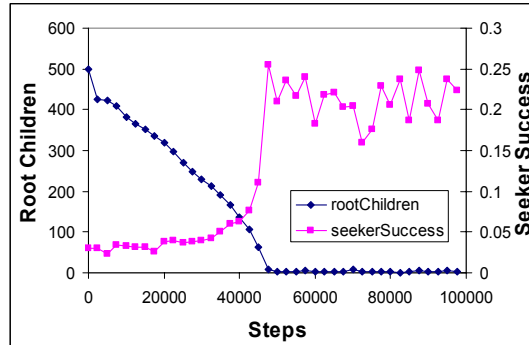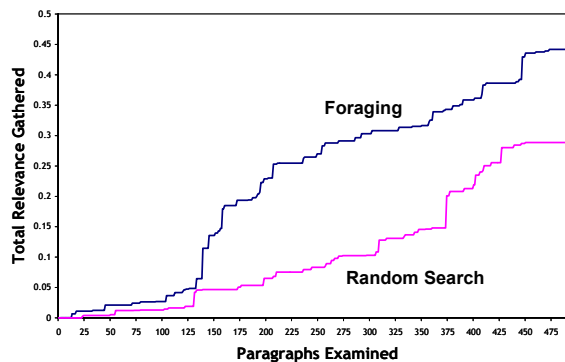

Figure 5. Effectiveness of Foraging in Hierarchically Clustered Data.

ground, so that the hierarchy is immediately available for foragers to service a query. From an engineering perspective, the speed of this convergence is a critical design parameter that can help determine the number of parallel processors that are needed to accommodate a given rate of change of the document population or the community model.

The asymptotic success rate of 25% for the seekers reflects the stochastic nature of the algorithm, and the convergence imposed on the system by the linear learning rate of the threshold θ. The stochastic character of the algorithm is what permits the system to accommodate dynamic data, and one is willing to tolerate an imperfect hierarchy as the price of that flexibility. In addition, we are currently testing a more sophisticated learning mechanism for governing convergence, one that we will expect to yield higher asymptotic rates of seeker success.

5. Conclusion

Successful intelligence analysis requires both sifting masses of available data to reduce their quantity, and sorting them to increase their semantic content. These processes often alternate with one another. Sifting reduces the volume of data so that sorting can apply increasingly complex analyses within the bounds of available time, while sorting structures the data in a way that makes the next round of sifting more efficient.

This alternation is apparent in two insect-inspired processes that we are using to address the massive data problem. Experiments with these mechanisms demonstrate their promise in concurrently increasing the structure of data and reducing its volume in support of higher-level analyses.

Acknowledgements

8. References

[1] R. Alonso and H. Li. Model-Guided Information Discovery for Intelligence Analysis. In *Proceedings of CIKM '05*, Bremen, Germany, 2005.

[2] P. Berkhin. Survey Of Clustering Data Mining Techniques. Accrue Software, San Jose, CA, 2002. http://citeseer.ist.psu.edu/berkhin02survey.html.

[3] CNS. CNS WMD Databases. 2004. CD,

[4] J. W. Coffey, R. R. Hoffman, A. J. Cañas, and K. M. Ford. A Concept Map-Based Knowledge Modeling Approach to Expert Knowledge Sharing. In *Proceedings of IASTED International Conference on Information and Knowledge Sharing*, 2002. http://www.ihmc.us/users/acanas/Publications/IKS2002/IKS.htm.

[5] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien. The Dynamics of Collective Sorting: Robot-Like Ants and Ant-Like Robots. In J. A. Meyer and S. W. Wilson, Editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 356-365. MIT Press, Cambridge, MA, 1991.

[6] T. Fawcett. ROC Graphs: Notes and Practical Considerations for Data Mining Researchers. HPL-2003-4, HP Laboratories, Palo Alto, CA, 2003. http://www.hpl.hp.com/techreports/2003/HPL-2003-4.pdf.

[7] C. Fellbaum, Editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. Cambridge, MA, MIT, 1998.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability*. San Francisco, CA, W.H. Freeman, 1979.

[9] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3), 1999.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671-80, 1983.

[11] G. A. Miller. WordNet: A Lexical Database for the English Language. 2002. Web Page, http://www.cogsci.princeton.edu/~wn/.

[12] H. V. D. Parunak. 'Go to the Ant': Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 75:69-101, 1997. http://www.altarum.net/~vparunak/gotoant.pdf.

[13] H. V. D. Parunak, S. A. Brueckner, R. Matthews, and J. Sauter. Pheromone Learning for Self-Organizing Agents. *IEEE SMC*, 35(3 (May)):316-326, 2005. http://www.altarum.net/~vparunak/ParunakIEEE.pdf.

[14] H. V. D. Parunak, S. A. Brueckner, J. A. Sauter, and R. Matthews. Global Convergence of Local Agent Behaviors. In *Proceedings of Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS05)*, Utrecht, The Netherlands, pages 305-312, 2005. http://www.altarum.net/~vparunak/AAMAS05Converge.pdf.

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

[16] P. Weinstein, H. V. D. Parunak, P. Chiusano, and S. Brueckner. Agents Swarming in Semantic Spaces to Corroborate Hypotheses. In *Proceedings of AAMAS 2004*, New York, NY, pages 1488-1489, 2004. http://www.altarum.net/~vparunak/AAMAS04AntCAFE.pdf.