Discovering Life Cycle Assessment Trees from Impact Factor Databases

Naren Sundaravaradan and **Debprakash Patnaik** and **Naren Ramakrishnan**
Department of Computer Science
Virginia Tech, Blacksburg, VA 24061

Manish Marwah and **Amip Shah**
HP Labs
Palo Alto, CA 94304

Abstract

In recent years, environmental sustainability has received widespread attention due to continued depletion of natural resources and degradation of the environment. Life cycle assessment (LCA) is a methodology for quantifying multiple environmental impacts of a product, across its entire life cycle – from creation to use to discard. The key object of interest in LCA is the inventory tree, with the desired product as the root node and the materials and processes used across its life cycle as the children. The total impact of the parent in any environmental category is a linear combination of the impacts of the children in that category. LCA has generally been used in 'forward' mode: given an inventory tree and impact factors of its children, the task is to compute the impact factors of the root, i.e., the product being modeled. We propose a data mining approach to solve the inverse problem, where the task is to infer inventory trees from a database of environmental factors. This is an important problem with applications in not just understanding what parts and processes constitute a product but also in designing and developing more sustainable alternatives. Our solution methodology is one of feature selection but set in the context of a non-negative least squares problem. It organizes numerous non-negative least squares fits over the impact factor database into a set of pairwise membership relations which are then summarized into candidate trees in turn yielding a consensus tree. We demonstrate the applicability of our approach over real LCA datasets obtained from a large computer manufacturer.

Introduction

In recent years, environmental sustainability has received wide attention due to continued depletion of natural resources and degradation of the environment. The growing threat of climate change has led industry and governments to increasingly focus on the manner in which products are built, used and disposed – and the corresponding impact of these systems on the environment. Another imperative for increased interest in sustainability is the plethora of new environmental regulation across the globe, including voluntary or mandatory eco-labels for disclosure of environmental impacts such as toxic effluents, hazardous waste, and energy use.

The most common approach to quantifying broad environmental impacts is the method of life cycle assessment (LCA) (Baumann and Tillman 2004; Finkbeiner and others 2006), which takes a comprehensive view of multiple environmental impacts across the entire lifecycle of a product (from cradle to grave). LCA can, for example, be used to answer questions like: "which is more environmentally sustainable, an e-reader or an old-fashioned book?" (Goleman and Norris 2010).

An LCA inventory for a product can be represented as a tree, as shown in Fig. 1 for a desktop computer, with the product as the root of the tree, and the materials and processes used across its life cycle as its children. Each node of the tree is associated with various environmental impact factors as shown in Table 1. This table shows only three factors, but in practice the total number of impact factors in many commercial databases (Frischknecht and others 2005; PE International 2009; Spatari and others 2001) can run into a few hundred. Similarly, the number of components (rows) in Table 1 could run into hundreds or even thousands. Each of the impact factors follows the tree semantics, i.e., the impact factor of a parent node (e.g. desktop computer in Fig. 1) is a linear combination of the impact factors of its children. The coefficients of the linear combination denote the amount of each component/process that was involved in constructing the root node.

Assessing environmental impacts requires the creation of vast databases containing lists of products, components and processes which have been historically assessed, with specific environmental impact factors attached to each entry in the list. The objective of this paper is to construct LCA trees automatically, given a parent node and such an impact factor database. Note that although the impact factors of all nodes (including the parent) are known, there is no easy way to infer the parent-child relationships from the database. To see why automated discovery of LCA trees is useful, we consider the following two use cases: (1) **Assessment validation:** manufacturers may put carbon labels (the impact factors) on their products, but not necessarily publish any underlying information. No method in the field exists today to validate the claims other than elaborate/expensive man-

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Table 1: Impact factors of some nodes in the desktop computer LCA tree

	Sewage treatment (m^3 waste water) eco-toxicity	Radioactive waste (kg waste) land filling	Impacts on vegetation (RER m^2 ppm h) photochemical ozone formation
Electricity	113.98	6.0638E-5	1.9741
Steel	1726.5	4.8377E-5	11.778
CD-ROM Drive	56106	1.0936E-3	109.59
HDD	27314	7.369E-4	66.808
Power Supply	40160	2.039E-3	224.44
Circuit Board	593750	1.0298E-2	983.24
Aluminium	2035.5	3.1756E-4	34.727
ABS Plastic	1838.7	7.2846E-7	25.36

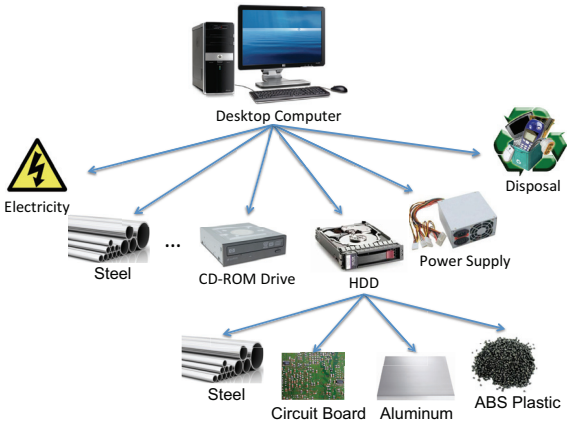


Figure 1: An example of a desktop LCA tree.

ual audits. In such cases, discovering the LCA trees could determine whether the disclosures are reasonable; (2) **Sustainable re-design:** it is usually too expensive and time-consuming for a supplier to estimate the impact of a product (parent) based on all its children, so a node in the impacts database approximately equivalent to the parent (root) is selected and the footprint computed without knowledge of its LCA tree. While this does give us a total footprint of the parent, it does not give any insight into a "hotspot" analysis, i.e., which components/processes (children) are the most significant contributors to the total footprint – information that could be vital in improving the sustainability of the product.

Our primary contributions are: i) identification and formulation of the LCA tree discovery problem; ii) a methodology to organize numerous non-negative least squares fits over the impact factor database into a set of pairwise membership relations which are then summarized into candidate trees in turn yielding a consensus tree; iii) successful demonstration of our approach to reconstruct six trees from a database of impact factors provided by a large computer manufacturer.

Problem Formulation

The LCA tree discovery problem can be formulated as follows. Given an impact factor database L (a $k \times p$ real matrix between k nodes $N = \{n_1, \ldots, n_k\}$ and p impact factors $M = \{m_1, \ldots, m_p\}$) and a specific node $n_i \in N$, the task is to reconstruct the tree of containment relationships rooted at n_i and having elements from $N - n_i$, such that the impact factor vector (of length p) of n_i is a positive linear combination of the impact factor vectors of its children. Since, there are many possible fits we specifically want to find nodes that make a high contribution across many of the impact factors. Note that technically a node can be a child of itself, e.g., desktop computers may be used in building a desktop computer, however, such scenarios are not addressed in this paper.

Essentially, the leaves of the LCA tree can be viewed as defining a subspace (in impact factor space) and the vectors at the internal nodes and the root node lie in this subspace. The leaf vectors may not form a true basis and, furthermore, due to impreciseness in measurements the internal nodes and root vectors may not be exact linear combinations of the leaf vectors. Due to the positivity constraint, classical rank-revealing decompositions such as QR (Smith and others 1989) do not directly apply here. Techniques such as non-negative matrix factorization (NMF) do guarantee positivity of factors but will identify a new basis rather than picking vectors from L to use for the basis. The most directly applicable technique is non-negative least squares regression (Lawson and Hanson 1974) but this has to be combined with a feature selection methodology that aids in pruning out nodes that are unlikely to constitute the object of interest.

Traditional feature selection methods (Koller and Sahami 1996; Peng, Long, and Ding 2005) for classification problems utilize mutual information criteria between features and target classes of interest to identify redundant and irrelevant features. Feature selection for regression, such as implemented in packages like SAS, follows stepwise (backward- or forward-) procedures for identifying and scoring features. Since the space of subsets is impractical to navigate in completeness, most such methods utilize heuristics to greedily choose possibilities at each step. Our problem here is unique due to two considerations: *associativity* and *substitutability*. First, because life cycle inventories constitute concerted groups of components that are composed into a larger product, we require an 'associative' approach to feature selection, i.e., some components are relevant only when other components are present. As a result, the feature selection methodology must be able to reason about conditional relevance of features and clusters of related features in addi-

tion to predictive ability toward the target attributes (here, the impact factors). Second, as described earlier, components have a notion of 'substitutability' among themselves and it is ideal if the feature selection methodology is geared toward recognizing such relationships.

Methods

Our overall methodology for inferring LCA trees is given in Fig. 2. We now present the conventions and steps underlying this approach. As stated earlier, let $N = \{n_1, \ldots, n_k\}$ be the set of nodes and $M = \{m_1, \ldots, m_p\}$ be the set of impact factors. L is the $k \times p$ impact factor matrix where L_{ij} is the value of impact factor j for node i.

Since the underlying core of LCA tree discovery is the search for good non-negative least squares (NNLS) fits, we will develop some machinery for reasoning about them. Given the root node n_a we aim to find an NNLS fit for the impact factors of n_a by considering the nodes in a set A as potential children of n_a. Let $f : N \times 2^N \longrightarrow \{0, 1\}$ such that $f(n_a, A) = 1$ if and only if the nodes in A can be children of n_a by an NNLS fit. In practice, we will impose specific criteria for the accuracy of this fit and, consequently, the definition of f.

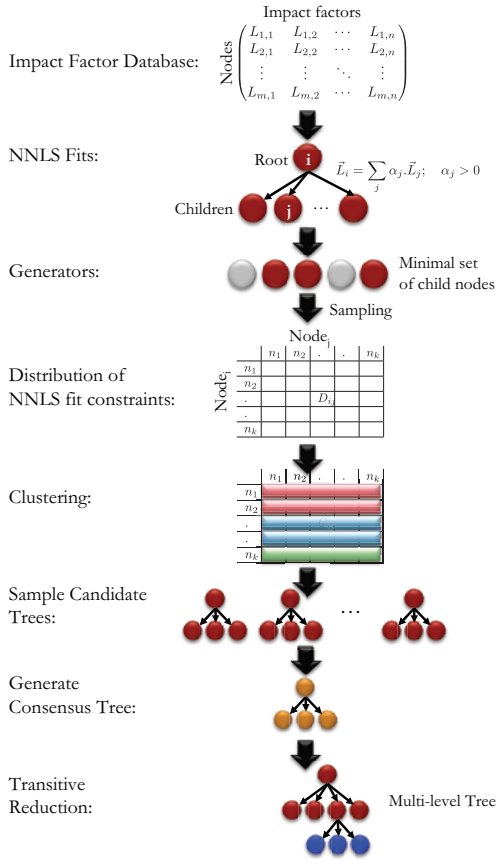


Figure 2: Methodology for discovering LCA trees.

First, when we reconstruct the impact factors for the root node using the discovered tree and compare them to the given impact factors, we will impose a parameter ϵ, the allowable error for an impact factor. Second, we will require at least η impact factors to conform to this error threshold. The value of η is determined by assessing the general quality of the NNLS fit by performing fits on various known trees. Let NNLS : $N \times 2^N \longrightarrow \mathbb{N}$ be the NNLS function that performs a fit with the given nodes and returns the number of impact factors satisfying the ϵ error criteria. We say that $f(n, A) = 1$ if NNLS$(n, A) \geq \eta$.

A generator of a node n_a is defined to be a set of nodes A such that $f(n_a, A) = 1$ and for all $B \subset A$, $f(n_a, B) = 0$. In other words, the set A of nodes gives a good NNLS fit for the node n_a but if we remove any node from A the fit violates our criteria.

Example of generators

Let us consider an example with 9 nodes and 268 impact factors. The parent node is a LiC6 electrode. This parent has 5 'true' children (which are a subset of the given 9 nodes) while the other 3 nodes are irrelevant nodes (recall that the 9th node is the parent). This is illustrated in Figure 3. We will consider $\eta = 214$ and $\epsilon = 0.15$. The following sets of nodes are extracted as generators: $\{664, 1056, 7224\}$, $\{411, 1056, 7224\}$ and $\{364, 1056, 7224\}$. However, $\{664, 1056, 7224, 9272\}$ and $\{364, 1056, 7224, 9272\}$ are not generators because although NNLS will produce a fit we can still remove node 9272 from these fits to produce the sets $\{664, 1056, 7224\}$ and $\{364, 1056, 7224\}$.

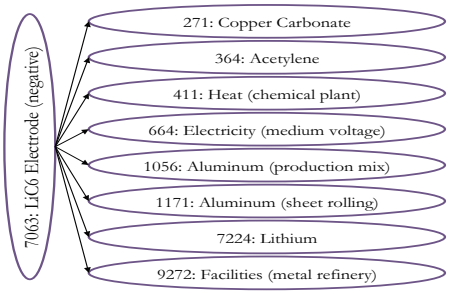


Figure 3: Illustrative example for our methodology.

Finding one generator

Since a generator captures the minimality of subsets necessary to have an NNLS fit at a desired level of accuracy, we organize the search for generators in a level-wise fashion. First, we concentrate on finding just one generator. It is clear that if we regress with NNLS using *all* nodes as children (which includes the correct children), we will obtain an acceptable fit. Hence, an idea that readily suggests itself is to begin with the set of all nodes and incrementally remove nodes to see if the fit continues to satisfy the desired criteria. If it does, we continue removing nodes until we encounter a generator. Algorithm 1 presents this algorithm. It is written to be more expressive, in particular to require a set of nodes in the fit and to search for the (minimal) generator that contains the given set of nodes.

Algorithm 1 FindGenerator(n,F,L)

Require: A parent node, n; a set of nodes to fix, F; a set of nodes, $L \supset F$
Ensure: Final list of nodes L'
1: $h \leftarrow$ head of L
2: $T \leftarrow$ tail of L
3: **if** h has been visited **then**
4: $L' \leftarrow L$
5: **else if** $h \in F$ **then**
6: $L' \leftarrow$ FindGenerator(n,F,append h to end of T)
7: **else**
8: fits $\leftarrow f(n, T)$
9: **if** $h \notin F$ and fits = true **then**
10: $L' \leftarrow$ FindGenerator(n,F,T)
11: **else**
12: $h \leftarrow$ mark h as visited
13: $L' \leftarrow$ FindGenerator(n,F,append h to end of T)

Table 2: Sampled distribution of constraints between NNLS fits for the running example.

	271	364	411	664	1056	1171	7063	7224	9272
271	40	0	0	0	40	0	0	40	0
364	0	40	0	0	40	0	0	40	0
411	0	0	40	0	40	0	0	40	0
664	0	0	0	40	40	0	0	40	0
1056	0	0	0	0	40	0	0	40	0
1171	0	0	0	0	40	40	0	40	0
7063	0	0	0	0	0	40	40	40	0
7224	0	0	0	0	0	40	0	40	0
9272	0	0	0	0	0	40	0	40	40

Organizing the search for generators

Generators give us a good idea about the possible set of nodes we ought to consider. But the nodes in a generator are inextricably tied to the other nodes and thus it is instructive to get some understanding of how nodes co-exist in generators. Note that a generator is not itself a solution because we only require the fit to satisfy η, which is not the maximum number of impact factors that satisfy the error threshold. In addition, we need several generators to understand how two nodes are related to each other. Specifically, for each node n_i we use Algorithm 1 τ times to find generators containing n_i. The resulting distribution is tabulated as matrix D. Table 2 depicts this matrix for our running example with $\tau = 40$; we immediately notice similar rows in the table. We see that fixing 7224 or 7063 produces the same distribution of nodes within the generators indicating that the two ought to be kept together. But, a weaker match is found between the rows for 271 and 1056. The next stages of the algorithm will exploit these similarities by clustering similar distributions so that we can start removing disimilar nodes first before we try to remove similar nodes.

Finding candidate trees

We now cluster the rows of the distribution matrix D into κ groups, with an eye toward balanced clusters. Recall that the rows of D denote conditional distributions and we are hence aiming to identify similar conditioning contexts. The resulting list of clusters C is then subject to several processing steps, as shown in Algorithm 2. The first step, the Remove-Fixed stage, is described in detail in Algorithm 3. Here we are aiming to prune the list of clusters. In this stage, we assess the currently possible number of fits and aim to remove clusters such that the number of fits does not change. Note that this is a fairly stringent requirement unlike our previous stage where we were searching for generators. Next, we invoke the IncreaseFits algorithm, described in Algorithm 4. Here we are aiming to do a finer-grained pruning of clusters by considering the removal of nodes within clusters. Note that we consider such pruning only if the cluster sizes are greater than 1. Finally, we perform another RemoveFixed stage, to cleanup any extraneous clusters. The result of these steps is a set of clusters with possibly different numbers of elements across them.

Algorithm 2 CandidateTree(n,C)

Require: A parent node, n; a list of clusters C
Ensure: A list of reduced clusters C_4
1: $C_1 \leftarrow$ RemoveFixed(n,C)
2: $C_2 \leftarrow$ IncreaseFits(n,C_1)
3: $C_3 \leftarrow$ IncreaseFits(n,C_2)
4: $C_4 \leftarrow$ RemoveFixed(n,C_3)

Algorithm 3 RemoveFixed(n,C)

Require: Parent node n; a set of clusters C
Ensure: A set of reduced clusters C'
1: $t \leftarrow \text{NNLS}(n, C)$
2: $C' \leftarrow \emptyset$
3: **for** each $c \in C$ **do**
4: **if** $f(n, C - \{c\}) = t$ **then**
5: $C' \leftarrow C' \cup \{c\}$

Algorithm 4 IncreaseFits(n,C)

Require: Parent node n; a list of clusters C
Ensure: A list of reduced clusters C'
1: $c \leftarrow$ unvisited cluster in C
2: $T \leftarrow C - \{c\}$
3: $t \leftarrow NNLS(n, C)$
4: **if** all $c \in C$ has been visited **then**
5: $C' \leftarrow C$
6: exit
7: **if** $|c| >= 2$ **then**
8: Mark c as visited
9: $s \leftarrow$ select $s \in c$ s.t. $NNLS(n, T \cup s)$ is minimum
10: $t' \leftarrow NNLS(n, T \cup s)$
11: **if** $t' \geq t$ **then**
12: IncreaseFits(n,$C - s$)
13: **else**
14: IncreaseFits(n,C)
15: **else**
16: IncreaseFits(n,C)

Determining the consensus tree

Algorithm 5 describes our approach to identify a final consensus tree. We first calculate the average number of clusters found in the candidate trees. For each cluster, we select the number of nodes to be the mean sizes of clusters found in

candidate trees. Finally, we pick the most frequent nodes in each cluster as our final tree.

Algorithm 5 ConsensusTree(S)

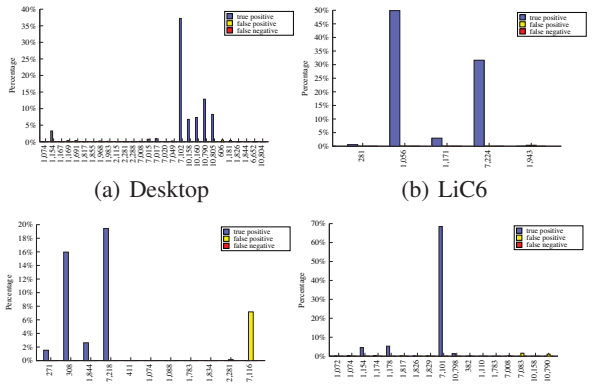
Require: Set of candidate trees S
Ensure: Consensus tree T
1: $avgC \leftarrow$ average number of clusters in S
2: $C \leftarrow$ select $avgC$ (rounded) most frequent clusters
3: $Z_i \leftarrow$ mean number of nodes (rounded) cluster $c_i \in C$
4: $T \leftarrow$ for each cluster $c_i \in C$ select Z_i most frequent nodes in the cluster

Finding multi-level trees

Our final consideration pertains to trees with multiple levels, such as shown in Fig. 1. We breakdown the problem of finding multi-level trees to finding a consensus tree, finding trees within the set of nodes discovered, superposing all found trees, and computing their transitive reduction (Aho and others 1972). Essentially, this 'flattens' out the multi-level tree into a single tree and uses containment relationships within the nodes of this unified tree to reconstruct the hierarchy.

Experimental Results

We present experimental results in inferring LCA trees for six products where reference trees are available (obtaining good evaluation datasets is cumbersome in this domain due to proprietary concerns and difficulties in manually organizing the relevant information). In the LCA dataset considered here, we are given 76 nodes and 268 impact factors. For each product for which we would like to reconstruct an LCA tree, we consider all remaining 75 nodes as potential children.



(a) Desktop (b) LiC6

(c) LiMn2O4 (d) CDROM

Figure 4: Median impact factor contributions of true child nodes, false positives and false negatives.

The questions we seek to answer are:

1. How accurate are the nodes correctly selected by our approach (true positives) compared to the reference trees?
2. For nodes that are selected in error (false positives), can we characterize them further in terms of substitutability?
3. What is the median contribution to impact factor vectors of nodes selected by our approach vis-a-vis missed nodes (false negatives)?

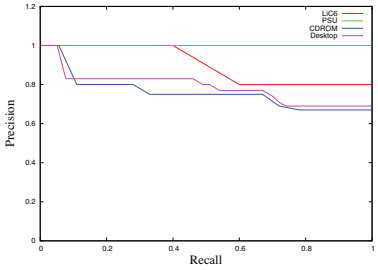


Figure 5: Interpolated Precision-Recall Plot.

In the study presented here, we use the following parameter settings: $\epsilon = 15\%$, $\eta = 80\%$, $\kappa = 25$, and we experimented with τ values of 40 and 80. Runs against our database for $\tau = 40$ typically take approximately 1 hour for each discovered tree with a proportional increase for $\tau = 80$. The overall statistics for the number of true nodes and of discovered nodes for each tree is listed in Table 3. As we can see a large fraction of nodes are discovered correctly with some extraneous and missed nodes. Fig. 5 plots a curve of the precision vs recall for various settings of top-k results in tree reconstruction. As this figure shows, we are able to achieve a very good balance between both criteria using our methodology.

Next, we undertook a qualitative study with LCA domain experts to help categorize three types of errors in our methodology: (i) nodes that were not discovered, but are generally believed to have a small contribution to the overall system so much so that these might not even have been included by another LCA practitioner. The error from these nodes may be reasonable to ignore ('reasonable'). (ii) nodes which were discovered and not on the reference tree, but are very similar in properties to the another node on the reference tree. An example of this might be discovering a node of one type of plastic where the known tree contained a different but very similar type (in terms of impact factors) of plastic. These represent examples of nodes which could easily have been substituted into the existing tree without changing the form or purpose of the tree, and are therefore reasonable to accept within the discovered tree ('substitutable'). (iii) nodes which were discovered and not on the known tree, and bear no resemblance to any nodes on the known tree; or nodes which are on the known tree and have a large contribution to the parent's impact factors but not discovered. These are nodes which have no explanation for being part of the tree, or nodes which should have been discovered and are not ('unusual'). This last category is the most significant example of error in our methodology. As can be seen from Table 3, there are at most one or two nodes in the 'unusual' category across trees of different sizes.

Finally, Fig. 4 depicts the median error across impact factor contributions for the true child nodes selected by our approach vis-a-vis false negatives and false positives. The nodes that our approach misses have minuscule contribu-

tions compared to other nodes whereas the false positive nodes have significantly higher contributions and are hence objects for further study by the LCA specialist. This plot hence shows that our approach does not overwhelm the specialist with possibilities and at the same time is able to narrow down on most of the important nodes without difficulty.

Table 3: Overall tree reconstruction statistics where discovered #nodes = true positives + incorrect nodes.

Tree Name	Known #nodes	Discovered #nodes	True +ve	Incorrect nodes		
				Substitutible	Reasonable	Unusual
LiC6	5	5	4	1	0	0
PSU	6	6	6	0	0	0
CDROM	18	17	10	5	0	2
Desktop	37	28	21	5	1	1
Battery, Lilo	19	13	10	1	0	2
LiMn2O4	11	12	7	2	1	2

Tree for PSU

Due to space limitations, we showcase in detail the reconstructed tree for PSU (power supply unit), an electronics module. Interestingly, both the reference and computed trees here fit all 268 impact factors. The results are shown in Fig. 4. Here, we reconstruct 5 out of 6 child nodes with a setting of $\tau = 40$. With a higher sampling $\tau = 80$ we are able to reconstruct all 6 nodes with almost exact replication of coefficients.

Table 4: PSU LCA tree with 6 children.

Known Nodes	Known Coeff.	40 Samples		80 Samples	
		Found	Coeff.	Found	Coeff.
1154: Steel low-alloyed	0.572	Y	0.577	Y	0.572
1174: Sheet rolling aluminum	0.572	Y	0.529	Y	0.572
10806: fan at plant	0.074	Y	0.083	Y	0.074
7018: plugs inlet...	1.0	N	-	Y	1.004
7116: cable ribbon	0.194	Y	0.223	Y	0.193
10804: printed wiring...	0.604	Y	0.602	Y	0.602

Discussion

As motivated in the introduction, we have automatically discovered LCA trees from a database of environmental impact factor information. In particular, we have been able to reconstruct, with satisfactory accuracy, the components and processes underlying complex artifacts such as the desktop computer HDD. Our novel formulation of alternating rounds of NNLS fits and clustering/pruning has proved to be accurate in reconstructing LCA trees.

Future work revolves around several themes. First, we would like to incorporate other domain-specific information (e.g., textual descriptions of components and products) to further steer the reconstruction of LCA trees. Second, we would like to formally characterize the nature of approximate solutions and the tradeoffs in the various guarantees that can be provided. Finally, we would like to develop a broader semi-supervised methodology for the inference of LCA trees.

To summarize, the current state-of-the-art requires a system designer to specify the quantity and content of each component within the system, and then link these to relevant impact factors manually. Using the approach of tree discovery outlined in this paper, this problem has been sufficiently simplified so that a designer is able to automatically reconstruct trees and estimate the associated quantities and impacts. Beyond eliminating one of the most costly steps of traditional LCA, the proposed methodology actually streamlines the LCA process: by hiding the computational complexity and labor from the designer, environmental assessments can become more widely accessible to practitioners who do not possess specialized domain expertise. We consider this a key first step in our vision of empowering product and system designers to estimate the environmental footprints associated with any arbitrary system.

References

Aho, A. V., et al. 1972. The transitive reduction of a directed graph. *SIAM Journal on Computing* 1(2):131–137.

Baumann, H., and Tillman, A. 2004. *The Hitchhiker's Guide to LCA: An orientation in Life Cycle Assessment Methodology and Application*. Studentlitteratur Sweden.

Finkbeiner, M., et al. 2006. The New International Standards for Life Cycle Assessment: ISO 14040 and ISO 14044. *The Intl. Journ. of Life Cycle Assessment* 11(2):80–85.

Frischknecht, R., et al. 2005. The Ecoinvent Database: Overview and Methodological Framework. *The Intl. Journ. of Life Cycle Assessment* 10(1):3–9.

Goleman, D., and Norris, G. 2010. How green is my ipad? http://www.nytimes.com/interactive/2010/04/04/opinion/04opchart.html?hp.

Koller, D., and Sahami, M. 1996. Toward optimal feature selection. In *ICML'96*, 284–292.

Lawson, C. L., and Hanson, R. J. 1974. *Solving Least Square Problems*. Englewood Cliffs NJ: Prentice Hall.

PE International. 2009. GaBi - Life Cycle Assessment (LCE/LCA) Software System. http://www.gabi-software.com.

Peng, H.; Long, F.; and Ding, C. 2005. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Trans. Pattern Anal. Mach. Intell.* 27:1226–1238.

Smith, J., et al. 1989. All possible subset regressions using the QR decomposition. *Computational Statistics & Data Analysis* 7(3):217–235.

Spatari, S., et al. 2001. Using GaBi 3 to perform Life Cycle Assessment and Life Cycle Engineering. *The Intl. Journ. of Life Cycle Assessment* 6(2):81–84.