HAL open science

Using graph search algorithms for a rigorous application of emergy algebra rules

Antonino Marvuglia, Benedetto Rugani, Gordon Rios, Yoann Pigné, Enrico Benetto, Ligia Tiruta-Barna

To cite this version:

Antonino Marvuglia, Benedetto Rugani, Gordon Rios, Yoann Pigné, Enrico Benetto, et al.. Using graph search algorithms for a rigorous application of emergy algebra rules. Revue de Métallurgie, 2013, 110 (01), pp.87–94. 10.1051/metal/2013050 . hal-00926422

HAL Id: hal-00926422

https://hal.science/hal-00926422v1

Submitted on 29 May 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright

Revue de Métallurgie **110**, 87–94 (2013)
© EDP Sciences, 2013
DOI: 10.1051/metal/2013050
www.revue-metallurgie.org

Revue de Métallurgie

Using graph search algorithms for a rigorous application of emergy algebra rules

A. Marvuglia[1], B. Rugani[1], G. Rios[2], Y. Pigné[3], E. Benetto[1] and L. Tiruta-Barna[4]

[1] *Public Research Centre Henri Tudor, Resource Centre for Environmental Technologies, 66 rue de Luxembourg, 4002 Esch-sur-Alzette, Luxembourg*
e-mail: antonino.marvuglia@tudor.lu
[2] *Cork Constraint Computation Centre, University College Cork, Western Gateway Building, Cork Ireland*
[3] *LITIS, Normandy University, 25 rue Philippe Lebon CS 80540, 76058 Le Havre Cedex, France*
[4] *Université de Toulouse INSA, UPS, INP, LISBP, INRA UMR792, CNRS UMR5504, 135 av. de Rangueil, 31077 Toulouse, France*

Key words:
Emergy evaluation; graph search; network analysis; Ecoinvent; life cycle assessment

Abstract – *Emergy* evaluation (EME) is an environmental assessment method which is gaining international recognition and has increasingly been applied during the last decade. *Emergy* represents the memory of the geobiosphere exergy (environmental work) – measured in *solar emjoules* (seJ) – that has been used in the past or accumulated over time to make a natural resource available. The rationale behind the concept of *Emergy* is the consideration that all different forms of energy can be sorted under a hierarchy and measured with the common metric of the seJ, which is then the yardstick through which all energy inputs and outputs can be compared with each other. For this reason EME is suggested to be a suitable method of environmental accounting for a wide set of natural resources, and can be used to define guidelines for sustainable consumption of resources. Despite those interesting features, EME is still affected by several drawbacks in its calculation procedures and in its general methodological background, which prevent it from being accepted by a wider community. The main operational hurdle lays in the set of mathematical rules (known as *Emergy algebra* rules) governing EME, which do not follow logic of conservation and make their automatic implementation very difficult. This work presents an open source code specifically created for allowing a rigorous *Emergy* calculation (complying with all the *Emergy algebra* rules). We modeled the *Emergy* values circulating in multi-component systems with an oriented graph, formalized the problem in a matrix-based structure and developed a variant of the well-known *track summing* algorithm to obtain the total *Emergy* flow associated with the investigated product. The calculation routine (written in C++) implements the Depth First Search (DFS) strategy for graph searches. The most important features of the calculation routine are: (1) its ability to read the input in matrix form without the need of drawing a graph; (2) its rigorous implementation of the *Emergy* rules; (3) its low running time, which makes the algorithm applicable to any system described at the level of detail nowadays made possible by the use of the available life cycle inventory (LCI) databases. A version of the *Emergy* calculation routine based on the DFS algorithm has been completed and is being tested on case studies involving matrices of thousands of rows and columns, describing real product production systems.

Received 31 January 2013
Accepted 6 February 2013

Life Cycle Assessment (LCA) represents one of the most accepted and used tools for the environmental assessment of goods and services [1, 2]. The Ecoinvent database [3] is the largest worldwide data repository to provide Life Cycle Inventory (LCI) models for thousands of products (i.e. energy generation, materials and agri-food production, infrastructure and transportation), which can be used as the background system to build LCAs. The organization of the data in Ecoinvent perfectly fits with the framework required by the matrix method for the solution of the inventory problem in LCA [4].

In terms of natural resource protection, LCA takes an anthropocentric perspective, since the value of resources is related to their scarcity or to their usefulness for human systems. Therefore, from the LCA standpoint

the resources to be protected are the ones requested by economic production activities (via market mechanisms) and the stocks of which are scarce. Renewable resources and ecosystem services are often excluded from LCA, although their importance to human activities and their increasing depletion have been recognized [5].

Stemming from the work by Odum [6], a new paradigm, which is based on the concept of *Emergy* (spelled with an "*m*") and aims at quantifying resources from a nature-centred point of view, has emerged in the scientific community. In its traditional definition *Emergy* quantifies the amount of available energy (usually expressed as *solar* energy and measured in *solar emjoules* (seJ), or its multiples) previously directly and indirectly required to generate a product and/or to support a system and its level of organization [7]. In so doing, Emergy Evaluation (EME) aims at accounting for the memory of the geobiosphere available energy (also called exergy) provision (i.e. environmental work) supporting economic systems (e.g. to make a product or service) via the use of natural resources.

However, conventional EME has been strongly criticized due to the lack of accuracy and completeness in its calculation procedures. Indeed EME is very often based on simplified and non-transparent models, as well as incomplete data inventories [8]. This lack of a rigorous analytical and accounting basis has hampered *Emergy* acceptance and use by environmental policy authorities [8–12].

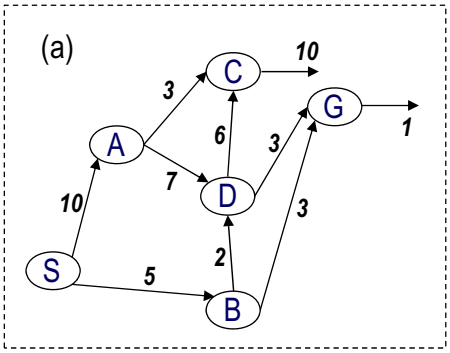
In recent years, complementarities between EME and LCA have been emphasized within the *Emergy* community [13]. As acknowledged by Rugani and Benetto [14], EME could benefit from the use of existing LCI databases, which account for hundreds of environmental interventions in thousands of common industrial processes. On the other hand, through the LCI databases, *Emergy* might bring into LCA a complementary concept to assess impacts from ecosystem services' use. Moreover, it is rather widely recognized that this new way of proceeding to *Emergy* accounting underpinned by the LCA matrix-based structure (very well described in Heijungs and Suh [4]) may potentially contribute to significantly improve the accuracy and completeness of *Emergy* accounting [15–17]. However, a full integration of the two methods has been hampered so far by a number of methodological issues, such as the different system boundaries and allocation criteria [14, 15] and the different mathematical frameworks of the two methodologies, due to a different logical perspective. Indeed, *Emergy* holds a logic of *memorization*, i.e. it aims at calculating how much solar exergy has supported, through natural resources and related ecosystem services, the life cycle of a product. While LCA holds a logic of *conservation*, i.e. it aims at evaluating the amount of resources required by the life cycle of a product and assessing the potential impacts at the level of component/process.

In particular, EME requires compliance with a set of algebraic rules that are completely different from those applied in LCA [6, 15, 18]. For the sake of completeness the list of *Emergy algebra* rules taken from Brown and Herendeen [18] is reported below:

1. all source *Emergy* to a process is assigned to the process' output;
2. byproducts from a process have the total *Emergy* assigned to each pathway;
3. when a pathway splits, the *Emergy* is assigned to each "leg" of the split based on its percent of the total energy flow on the pathway;
4. *Emergy* cannot be counted twice within a system: (a) *Emergy* in feedbacks cannot be double counted; (b) byproducts, when reunited, cannot be added to equal a sum greater than the source *Emergy* from which they were derived.

Rule 4(b) implies that, when summing *Emergy* of inflows or outflows that are byproducts, only the largest one has to be accounted for.

The aim of this paper is to provide a new insight for calculating *Emergy* through an LCI scheme. We present a novel algorithm which is able to quantify the seJ value of every kind of product and process (provided its LCI is known) through a rigorous implementation of the *Emergy algebra* rules. The algorithm treats the system of interconnected processes delivering the product at stake



(b)

	A	B	C	D	G
S	-10	-5	0	0	0
A	0	0	-3	-7	0
B	0	0	0	-2	-3
C	0	0	10	0	0
D	0	0	-6	0	-3
G	0	0	0	0	1

Fig. 1. Example of a simple system with a source S and two delivered outputs C and G. The numbers on each connection represent the (energy or mass) flows exchanged by the nodes of the system (each representing a man-made or an ecosystem production process). (a) graph representing the system; (b) corresponding matrix.

as a graph. It uses the Depth First Search (DFS) strategy [19], memorizes the paths going from each input node (*Emergy* sources) to the output node(s) and finally records the output *Emergy* associated with each of the fully explored paths.

1 Graph search formulation of the problem

Let us consider the simple system depicted in Figure 1a and described by the matrix showed in Figure 1b. The convention used for the signs of the matrix's entries is the same adopted in Heijungs and Suh [4]: the entry at the generic position (i, j) of the matrix is negative if the process represented by column j requires as an input the flow coming from the process represented by row i. Positive entries can only lie on the main diagonal of the matrix and represent the outputs delivered by the concerned process. It is worth noting that the values showed in Figure 1 are not *Emergy* values, but mass or energy values, and they do not respect mass and energy conservation principles, because the system is represented in its non-scaled form. The values of mass and energy flows respecting conservation laws are obtained after the system is solved, i.e. suitable scaling factors are determined using the matrix method, as described in Heijungs and Suh [4].

Furthermore, the first row of the matrix showed in Figure 1b can be interpreted as the *environmental intervention* matrix in the LCA jargon (see [4]). In this simple example only one resource (S) is used by the system, so the environmental intervention matrix is represented by a row vector.

A graph is defined as a set of *vertices* (points) and a set of *edges* (alternatively called *arcs*): each edge joins one vertex to another, or starts and ends at the same vertex. In particular, the graph showed in Figure 1a is a directed graph, because the edges have a direction associated with them. For example, the energy or mass flow sprouting from S goes from S *to* A and from S *to* B, but not vice versa. When there is no direction associated with the edges, one deals with undirected graphs.

As far as we are concerned in this context, the problem we deal with can be formulated as follows: suppose the product for which we want to calculate the *Emergy* is represented by node G; then the *goal* G has to be reached starting from the *root* S through a set of *actions*, i.e. of moves from one node to another in the graph. The possible action sequences starting at the root node form a *search tree*, with the branches as actions. Each time we pass from one node to the other in the search tree, there is an *Emergy* value associated to the branch, which is ultimately linked to the energy or mass flow exchanged between the two nodes delimiting the branch.

An approach based on graph theory has recently been applied to EME by Le Corre and Truffet [20], where *Emergy* circulating

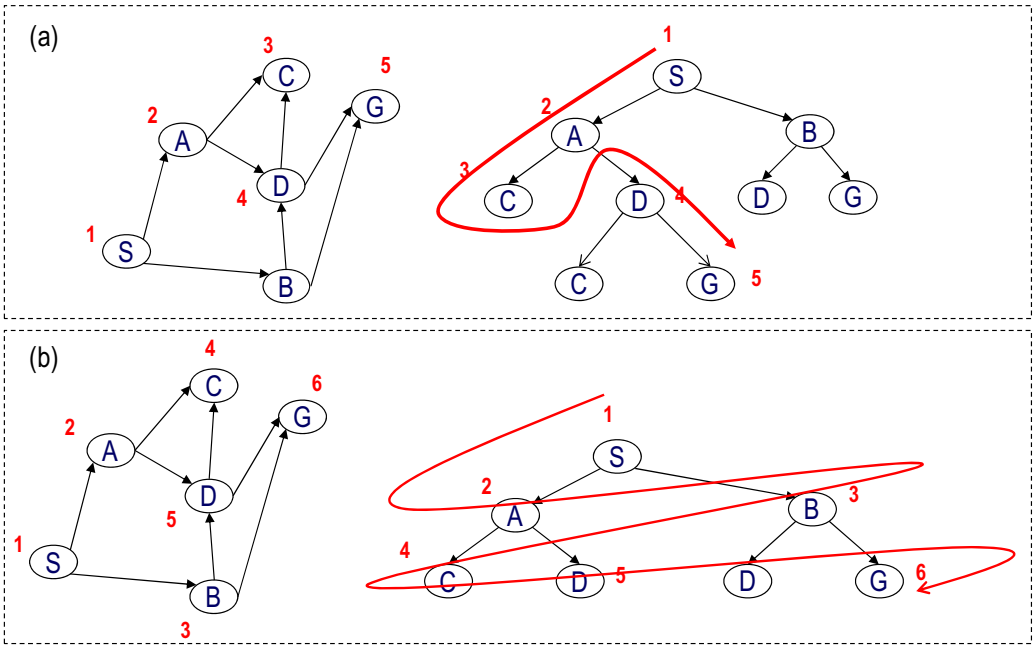


Fig. 2. (a) Depth-First search (DFS) algorithm and (b) Breadth-First search (BFS) algorithm: the left hand side of each picture shows an oriented graph and the right hand side shows the corresponding search tree. The red arrows show the visiting path and the numbers represent the order in which the nodes are visited starting from the source node S until the destination node G.

in a multi-component system is modeled as an oriented graph. However, in their approach a preliminary analysis (called "factorization") of the whole graph is required before starting the actual *Emergy* accounting. It is based on the expression of *Emergy* paths as strings and requires a set of computations on strings. This is certainly time consuming and likely to be impractical for very large networks. Furthermore the algorithm needs the relations between the arcs to be stored in an array, which is a computationally expensive operation, requiring extensive memory usage. However, in [20] the authors assume the *Emergy* paths to be known, anticipating that the computation of *Emergy* paths is the subject of a companion paper.

Finding a path between nodes or finding the shortest path between two nodes is a very common problem in computer science. Two well-known algorithms are used in computer science for graph search, which essentially differ for the order in which the nodes are visited during the search [19]. In the DFS the search starts at the root and explores as far as possible along each branch before backtracking, i.e. before going back to the last explored node from which it was possible to visit a new branch of the tree. BreadthFirst Search (BFS) begins at the root node and explores all the neighbouring nodes. Then for each of those nearest nodes, it explores their unexplored neighbour nodes, and so on, until it finds the goal. The two search algorithms are illustrated in Figure 2, using the same graph showed in Figure 1.

If one wants to comply with *Emergy algebra* No. 4, the problem is complicated by the fact that we have to keep memory of the *Emergy* value propagated along the path and we have to detect the presence of a feedback every time we encounter one (which is not the case for the graphs showed in Figures 1 and 2, since there are no feedback connections in them).

The algorithm we devised to solve the problem (which was coded in C++), essentially follows three steps:

1. it reads the mass and energy flows associated with a given system, in its matrix form. This implies that the user does not need to draw the network describing the system;

2. it performs a graph search and memorizes the paths going from each input node (*Emergy* sources) until the output node, provided that: (a) the path does not have a feedback, i.e. it hits a node in the same path which was visited previously; (b) the *Emergy* value along this path never goes below a fixed threshold (henceforth called *minflow*). If one of the two above mentioned conditions occurs, the algorithm stops propagating *Emergy* values downstream. The choice of a suitable threshold can significantly speed up calculation without increasing too much the information loss;
3. as a last step, it finally records the output *Emergy* associated with each of the fully explored paths.

In practice the algorithm tracks *Emergy* values for each *Emergy* independent source[1], spinning off new paths as they are encountered and maintaining the set of previously visited elements to prevent cycles; it finally sums up the results independently.

The addition of the check condition based on the *Emergy* threshold was necessary when we applied the algorithm to systems entailing large matrices (with thousands of rows and columns). In these cases the running time to explore all the paths rises exponentially, but the *Emergy* value propagated downstream the feedback is generally very little.

The algorithm is in practice a variant of the *track summing* algorithm originally due to Tennenbaum [21]. The *track summing* method follows each path in the network of *Emergy* values associated to the energy and mass flows from the sources to the output and divides the total *Emergy* input to a process by the corresponding *Emergy* output to obtain the so-called *transformity* of the process.

In a previous attempt [22], a different instantiation of the algorithm using a multithreaded BFS strategy [19] had been explored. The BFS algorithm exploits concurrency, i.e. allows parallel processing of different paths at the same time, thus allowing a lower running time. It was implemented in the programming environment Scala [23], which has a powerful concurrency model and is thus particularly suited for exploiting parallelization (on multi-core processors). However, although the running time is lower than for its DFS version, the memory requirement for the BFS instantiation of the algorithm is quite high and does not justify a desktop utilization. On the other hand, this multithreaded BFS instantiation will be very amenable to a cluster application of the calculation routine, in case the large scale of the systems dealt with in the future will make the problem untreatable in a singlethreaded fashion.

The pseudo-codes of the two different algorithms are showed in Figures 3 and 4.

The code has already been applied for the solution of a problem involving a simplified system for the production of flat glass [22] and it is currently being tested on problems involving much larger system matrices (with several hundreds of rows and columns).

For the solution of these latter case studies we are using the inventory data contained in the Ecoinvent database [3]. This confronted us with the solution of an additional problem, which is discussed in the following section.

1.1 Dealing with multi-output processes

The Ecoinvent database (version 2.2) contains 227 multioutput processes, each delivering two (in a few cases three) valuable outputs. However, the inventoried values contained in the database always refer to already allocated processes. Indeed, inputs of resources and commodities have already been apportioned to each byproduct (derived from a multi-output process) according to predefined allocation criteria. These, which obviously infringe the *Emergy algebra* rule No. 2, can be deduced for each multioutput process by reading the Ecoinvent documentation.

In order to comply with rule No. 2 it is necessary to equally assign the full amount of the inventoried resources to each byproduct of every multi-output process. To solve this problem we are devising a script in C++ which allows the extraction of text

[1] In the case showed in Figures 1 and 2 there is only one source, but systems are usually fed by multiple *Emergy* sources.

```
Define : Graph G, Stack S, List Inputs, List OutFlows, Parameter
MinFraction

Initialize: G for system with nodes containing tuples defined
(child = node, weight = fraction of emergy flowing to child)
Require: child weights sum to 1.0

Initialize: Inputs for system with tuples (node, flow)

Initialize: MinFraction as minimum fraction of each input to pass
to child process nodes

Initialize: S as empty LIFO stack, Outflows as empty List

For input (node, flow)  in Inputs:
  Define : Set P
  Add: node to P
  Define : MinFlow = MinFraction * flow
  Define : Task T = (node, flow, P, MinFlow)
  Push: T to S

While S has elements:
  Pop: Task U = (node, flow, path, minflow)
  If node has no children:
    Define : Output O = (node, flow)
    Add: O to OutFlows

  For child in node (child, weight):
    Define : childflow = weight * flow
    If childflow >= minflow   AND child NOT IN path:
      Add: child to path
      Add: Task (child, childflow, path, minflow)   to S

Collect: Sum outputs (node, flow) in Outflows by node
```

Fig. 3. Pseudo-code of the DFS algorithm.

and numerical data directly from the .xml files downloaded from the Ecoinvent website, and containing the information about the multi-output processes. The script will allow an automatic 100% re-allocation of the full input inventory to each of the process outputs delivered by each of the 227 multi-output processes.

As an example, let us suppose in Ecoinvent one finds a multi-output process producing Y kg of a PRODUCT #1 and Z kg of a PRODUCT #2, using an input of X MJ of energy. Let us assume an allocation criterion (e.g. based on the economic value of the by-products) has been chosen by the Ecoinvent team and allocation factors of 80% and 20% have been assigned to the PRODUCT #1 and the PRODUCT #2, respectively[2]. In the Ecoinvent database one will find the energy allocated to the unit amount of PRODUCT #1 as $(X \cdot 0.8)/Y$ and to the unit amount of PRODUCT #2 as $(X \cdot 0.2)/Z$. The specific C++ algorithm will allow compliance with *Emergy algebra* rule No. 2 by retrieving the allocation factors from the corresponding .xml file describing the multi-output process at stake. Then, from those values it will calculate the total amount X of the inventory item (in this case energy) that had been allocated, so that it can be totally reassigned to both products. This sort of "un-allocation" process assigns to the unit amount of PRODUCT #1 an amount of energy equal to X/Y MJ and to the unit amount of PRODUCT #2 an amount of energy equal to X/Z MJ.

[2] These allocation factors can be found in the Ecoinvent report where the multi-output process at stake is documented, and their values can be found within the .xml file containing the information about the multi-output process. The .xml files can be downloaded from the Ecoinvent website upon licence purchase for the database.

```
Define : Graph G, Queue Q, List Inputs, List OutFlows, Parameter
MinFraction

Initialize: G for system with nodes containing tuples defined
(child = node, weight = fraction of emergy flowing to child)
Require: child weights sum to 1.0

Initialize: Inputs for system with tuples (node, flow)

Initialize: MinFraction as minimum fraction of each input to pass
to child process nodes

Initialize: Q as empty FIFO queue, Outflows as empty List

For input (node, flow)  in Inputs:
  Define : Set P
  Add: node to P
  Define : MinFlow = MinFraction * flow
  Define : Task T = (node, flow, P, MinFlow)
  Push: T to Q

While Q has elements:
  Pop: Task U = (node, flow, path, minflow)
  If node has no children:
    Define : Output O = (node, flow)
    Add: O to OutFlows

  For child in node (child, weight):
    Define : childflow = weight * flow
    If childflow >= minflow   AND child NOT IN path:
      Add: child to path
      Add: Task (child, childflow, path, minflow)   to Q

Collect: Sum outputs (node, flow) in Outflows by node
```

Fig. 4. Pseudo-code of the BFS algorithm (from [22]).

2 Conclusions and future work

The paper describes an algorithm specifically devised to calculate the *Emergy* associated to a product/process using a matrix-based framework. This allows the use of the life cycle data contained in LCI databases, thus supplying an operational tool toward an integration of EME and LCA.

The algorithm implements a variant (based on graph search) of the *track summing* algorithm originally due to Tennenbaum [21].

A different instantiation of the algorithm using a multi-threaded BFS strategy has been previously applied to a simplified case study of flat glass production (7×7 matrix) in Marvuglia et al. [22], yielding consistent results with a very short calculation time (as small as 1.37 s).

The newest (more memory efficient) version of the *Emergy* calculation routine based on the DFS algorithm has been already completed and is being tested on case studies involving matrices of thousands of rows and columns, describing real product production systems.

The final version of the algorithm will be included in an *Emergy* calculation software which is currently under development. The software will then be applied case by case to several specific product's life cycles modeled using conventional LCA software tools, thus allowing an exact calculation of their *Emergy* and the creation of an inventory of *Emergy* values. In the integrated combination of EME and LCA, the use of LCA frameworks is expected to greatly improve the quality and consistency of the *Emergy* results.

References

[1] M.A. Curran, Life Cycle Assessment: principles and practice, EPA/600/R06/060 – U.S. Environmental Protection Agency, National Risk Management Research Laboratory: Cincinnati, Ohio, 2006, p. 80

[2] European Commission, Joint Research Centre, Institute for Environment and Sustainability, International Reference Life Cycle Data System (ILCD) Handbook – General guide for Life Cycle Assessment – Detailed guidance, First edition, EUR 24708 EN, Publications Office of the European Union: Luxembourg, 2010, p. 417

[3] B.P. Weidema, et al., Overview and methodology, Data quality guideline for the ecoinvent database version 3. Ecoinvent Report 1(v3), St. Gallen: The ecoinvent Centre, 2011

[4] R. Heijungs, S. Suh, The computational structure of life cycle assessment, Kluwer Academic Publishers, Dordrecht, The Netherlands, Series: EcoEfficiency in Industry and Science, Vol. 11, 2002

[5] R. Hassan, R. Scholes, A. Neville, (Eds.) Ecosystems and Human Well-being: Current State and Trends, Vol. 1, Island Press (available online: http://www.millenniumassessment.org/documents/document.766.aspx.pdf), 2005

[6] T.H. Odum, Environmental Accounting – Emergy and Environmental Decision Making, John Wiley & Sons, New York, USA, 1996

[7] R. Ridolfi, S. Bastianoni, Emergy, Encyclopedia of Ecology, 2008, pp. 1218-1228.

[8] J.L. Hau, B.R. Bakshi, *Ecol. Model.* **178** (2004) 215-225

[9] E. Sciubba, *Energy* **35** (2010) 3696-3706

[10] C.J. Cleveland, R.K. Kaufmann, D.I. Stern, *Ecological Economics* **32** (2000) 301-317

[11] B.A. Mansson, J.M. McGlade, *Oecologia* **93** (1993) 582-596

[12] E. Sciubba, S. Ulgiati, *Energy* **30** (2005) 1953-1988

[13] B. Ness, E. UrbelPiirsalu, S. Anderberg, L. Olsson, *Ecological Economics* **60** (2007) 498-508

[14] B. Rugani, E. Benetto, *Environ. Sci. Technol.* **46** (2012) 4701-4712

[15] B. Rugani, M.A.J. Huijbregts, C. Mutel, S. Bastianoni, S. Hellweg, *Environ. Sci. Technol.* **45** (2011) 5426-5433

[16] W.W. Ingwersen, *J. Ind. Ecol.* **15** (2011) 550-567

[17] M.T. Brown, M. Raugei, S. Ulgiati, *Ecological Indicators* **15** (2012) 227-235

[18] M.T. Brown, R.A. Herendeen, *Ecological Economics* **19** (1996) 219-235

[19] S.J. Russel, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, NJ, USA, 3rd ed., 2009

[20] O. Le Corre, L. Truffet, *Ecol. Model.* **230** (2012) 101-113

[21] S. Tennenbaum, Network energy expenditures for subsystem production, M.S. Thesis, Environmental Engineering Sciences, University of Florida, Gainesville, USA, 1988, p. 132

[22] A. Marvuglia, E. Benetto, B. Rugani, G. Rios, A scalable implementation of the backtracking algorithm for Emergy calculation with Life Cycle Inventory databases, Proc. of 25th International Conference on Environmental Informatics (EnviroInfo 2011), October 5-7, 2011, Joint Research Centre (JRC) Ispra, Italy, Vol. 2, 2011, pp. 755-764

[23] M. Odersky, L. Spoon, B. Venners, Programming in Scala – A comprehensive stepbystep guide, Second edition, Artima developer, 2010, p. 883