The International Journal of Life Cycle Assessment (2019) 24:2173–2190
https://doi.org/10.1007/s11367-019-01636-4

LCI METHODOLOGY AND DATABASES

# Computer vision for LCA foreground modelling—an initial pipeline and proof of concept software, lcopt-cv

P. James Joyce [1]

Received: 13 April 2018 / Accepted: 7 May 2019 / Published online: 5 June 2019


## Abstract

**Purpose** The majority of LCA studies begin with the drawing of a process flow diagram, which then needs to be translated manually into an LCA model. This study presents an initial image processing pipeline, implemented in an open-source software package, called *lcopt-cv*, which can be used to identify the boxes and links in a photograph of a hand-drawn process flow diagram and automatically create an LCA foreground model.

**Methods** The computer vision pipeline consists of a total of 15 steps, beginning with loading the image file and conversion to greyscale. The background is equalised, then the foreground of the image is extracted from the background using thresholding. The lines are then dilated and closed to account for drawing errors. Contours in the image are detected and simplified, and rectangles (contours with four corners) are identified from the simplified contours as 'boxes'. Links between these boxes are identified using a flood-filling technique. Heuristic processing, based on knowledge of common practice in drawing of process flow diagrams, is then performed to more accurately identify the typology of the identified boxes and the direction of the links between them.

**Results and discussion** The performance of the image processing pipeline was tested on four flow diagrams of increasing difficulty: one simple computer drawn diagram and three photographs of hand-drawn diagrams (a simple diagram, a complex diagram and a diagram with merged lines). A set of default values for the variables which define the pipeline was developed through trial and error. For the two simple flow charts, all boxes and links were identified using the default settings. The complex diagram required minor tweaks to the default values to detect all boxes and links. An 'unstacking' heuristic allowed the diagram with merged lines to be correctly processed. After some manual reclassification of link directions and process types, the diagrams were turned into LCA models and exported to open-source LCA software packages (*lcopt* and *Brightway*) to be verified and analysed.

**Conclusions** This study demonstrates that it is possible to generate a fully functional LCA model from a picture of a flow chart. This has potentially important implications not only for LCA practitioners as a whole, but in particular for the teaching of LCA. Skipping the steep learning curve required by most LCA software packages allows teachers to focus on important LCA concepts, while participants maintain the benefits of experiential learning by doing a 'real' LCA.

**Keywords** Process flow diagram · Computer vision · LCA foreground modelling · Open-source software

## 1 Introduction

Process flow diagrams (also referred to as process maps, process diagrams and flow diagrams) are a common and useful way to represent LCA foreground models. The ISO 14044 standard on LCA requirements and guidelines recommends drawing a process flow diagram to assist with both the scoping and data collection phases (International Standards Organisation 2006). Specifically, ISO recommend drawing 'unspecific process flow diagrams that outline all [of] the unit processes to be modelled, including their interrelationships' (ibid.). This advice is mirrored in the *Handbook on Life Cycle Assessment* (de Bruijn et al. 2002) and the guide to PAS 2050 published by BSI (2011)—where it is listed as an important initial step when conducting a product carbon footprint. A process flow

Responsible editor: Andreas Ciroth

✉ P. James Joyce
pjjoyce@kth.se

[1] Department of Sustainable Development, Environmental Science and Engineering (SEED), KTH Royal Institute of Technology, 100-44 Stockholm, Sweden

diagram is a reporting requirement under the Greenhouse Gas Protocol Product Life Cycle Accounting and Reporting Standard (World Resources Institute, World Business Council for Sustainable Development 2011).

Thus, it is likely that most LCAs have started with a process flow diagram being drawn, either on paper or on a screen. Data is then collected to quantify the flows on this diagram for each scenario to be assessed, giving us everything we need to create an LCA model. Here however, there is an interruption in the smooth progression of the LCA workflow. LCA models rapidly become complex and require specialised software to create. In most of the commonly used LCA software packages, this involves entering data in a series of tables, listing the inputs and outputs of each unit process. The learning curve for such LCA software can be steep; indeed, Ciroth (2012) reports that the main software houses offer training courses lasting several days. He later predicts however that usability of LCA software will increase over time, 'driven both by users unwilling to accept steep learning curves, and by new technological possibilities' (ibid.).

Computer vision is one such 'new technological possibility' which has the potential to be employed in LCA. Techniques and software libraries are rapidly being developed to allow computers to see and interpret the world in a similar way to human beings (Lecun et al. 2015). In this paper, I present an initial approach for interpreting hand-drawn (or computer-generated) process flow diagrams to create LCA models from a picture, with little or no learning curve. It is accompanied by an open-source, proof-of-concept software package called *lcopt-cv*, which links into the *lcopt/Brightway* (Mutel 2017; Joyce 2017) open-source LCA software ecosystem.

## 2 Methods

### 2.1 Overview

While there are multiple ways to draw an LCA process flow diagram, it is assumed for the purposes of this study that such a drawing would consist of boxes (of a more or less rectangular shape) representing unit processes, inputs from the technosphere and emissions to the biosphere, connected by lines or arrows. Figure 1 shows an example of a process flow diagram of the format used in this study. It is acknowledged that input and emission flows are often simply represented by labelled arrows; however, enclosing these labels in a box is a simple modification which can greatly increase their correct detection.

From a theoretical perspective, transforming a picture of an LCA process flow diagram, like that shown in Fig. 1, into a functioning LCA model requires five main steps.

1. 'Clean up' the image to identify the foreground (boxes and lines) from the background.
2. Identify the boxes which represent unit processes, inputs and emissions.
3. Identify which boxes are linked to one another.
4. Identify 'real' links between boxes and their directions.
5. Name each box and, optionally, assign external LCI datasets or specific flows to inputs and emissions.

For a human observer, each of these steps is a trivial task. In order for a computer to perform these steps, they must be broken down into discrete and tightly defined operations capable of being programmed. In this study, these tasks take the form of a recommended pipeline. In computing terms, a *pipeline* refers to a set of commands where the output of one operation becomes the input to the next operation. The pipeline presented here is written in the programming language Python, making use of the computer vision software library *OpenCV* (Bradski 2000).

In this section, the computer vision concepts utilised in the pipeline are briefly described (Section 2.2). This is followed by a description of the initial pipeline to identify boxes and connections from an image of a process flow diagram (steps 1 to 3 above, Section 2.3). In step 4, spurious connections are identified and corrected, and the directions of links are identified. This is achieved via the application of 'rules-of-thumb' or *heuristics* for the drawing of LCA process flow diagrams, and is described in Section 2.4. The final step requires human input via a graphical user interface (GUI). This is described in Section 2.5. Finally, the testing procedure for the proposed approach is described in Section 2.6.

### 2.2 Computer vision concepts employed

#### 2.2.1 Images as a matrix of pixels

The computer vision techniques described below rely upon the fact that computer images can be described as a grid of pixels. The colour of each pixel is determined by the intensity of red, green and blue (RGB) light required to produce that colour. This means that a colour image can be described as a matrix with the dimensions $w \times h \times 3$, where $w$ is the width of the image, $h$ is the height of the image and 3 is the number of colour channels. Mathematical operations can be performed on this matrix, and the resulting matrices displayed on screen as images.

#### 2.2.2 Conversion to greyscale

One of the most simple but important image matrix manipulations is converting a colour image to greyscale. Converting an image to greyscale reduces the image to a two-dimensional matrix, making calculations easier. Because images of process

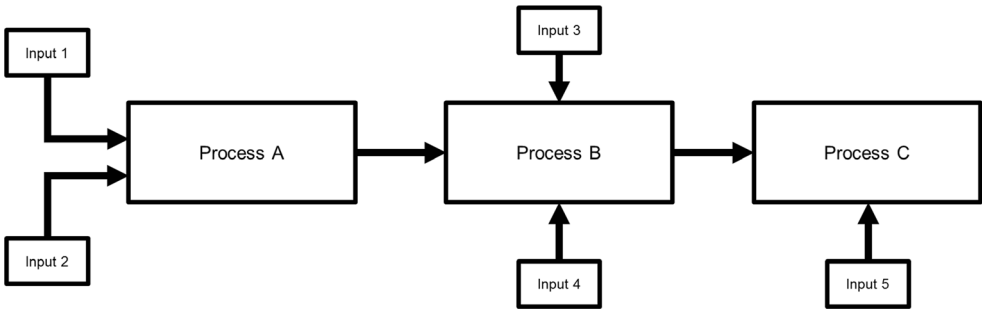



**Fig. 1** An example of an LCA process flow diagram

flow diagrams are likely to be composed of dark boxes and lines drawn on a lighter background, simplifying the image in this way is a sensible approach, as the exact colour of the boxes and lines serves no informative purpose. Mathematically, converting an image to greyscale requires that for each pixel the three colour intensity values be converted to a single value representing the luminance (light intensity) of that pixel. The contribution of red, green and blue wavelengths of light to perceived luminance differ, with green light contributing the most and blue light contributing the least. Coefficients for calculating luminance from RGB values in digital images are given in Recommendation ITU-R BT.601-7 of the International Telecommunication Union (2011). Thus, for each pixel, the intensity value is calculated according to the formula:

$$I = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

The intensity value is recorded on a scale from 0 to 255, representing the maximum number of values able to be represented by 8 binary computer bits, where 0 is black and 255 is white.

#### 2.2.3 Thresholding

Thresholding is used to isolate the foreground of a simple image from the background based on the intensity of each pixel. In the case of LCA process flow diagrams, this is the step which isolates the boxes, lines and text in the image from the background (paper) on which it is drawn. Thresholding allows the background of the image to be ignored in later steps.

Pixels with an intensity above a given 'threshold' value are sent to one extreme of the intensity range, while pixels below this value are sent to the other extreme. In this study, high-intensity pixels (lighter pixels = background) were set to 0 (black), while low-intensity pixels (darker pixels = foreground) were set to 255 (white). This results in a binary image, where an 'on' value (255) represents a foreground pixel, which is of interest for interpreting the flow diagram, and an off value (0) represents a background pixel, which can be ignored. Visually, this gives a representation of the process flow diagram in white on a black background. An example of thresholding is shown in Fig. 3.

#### 2.2.4 Structuring elements

Many of the following methods require transformations to be made by looking at small sections (i.e. collections of neighbouring pixels), within the image sequentially. These include background equalisation (Section 2.2.5), dilation, erosion and closing (Section 2.2.6). This can be thought of as analogous to reading a block of text character by character. The size and shape of the region which is assessed is defined using 'structuring elements'. These are small two-dimensional matrices containing 1 s or 0 s which are sequentially 'read' across an image (left to right, top to bottom). Mathematical operations can be carried out where 1s in the structuring element overlap with pixels in the image, allowing localised transformations of small sections of the image. The dimensions of the matrix represent the size of the structural elements, while the arrangement of 1 s and 0 s represent the shape (Fig. 2).

#### 2.2.5 Background equalisation

The human eye automatically accounts for differences in lighting in different parts of images. This is best demonstrated by the famous checker shadow illusion (Adelson 1995). Here, two squares on a checkerboard, where one is in the shadow of an object, are perceived as different colours when they are actually the same (Fig. 3, panel 1a). Computers do not have this ability and will (correctly) identify these squares as the same colour. This has an impact on thresholding, as the computer will assign pixels in these two areas of the image the same intensity value and therefore treat them in the same way. For photographs of hand-drawn flow charts, uneven lighting is inevitable. A human observer would automatically be able to account for this and parse the diagram correctly. However, for a computer, uneven lighting may result in some areas of background having the same intensity value as other areas of foreground, causing the thresholding step to incorrectly identify the foreground of the image. An additional processing

a) $\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$ b) $\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$ c) $\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$

**Fig. 2** Different types of 5 × 5 structuring elements. **a** Square. **b** Cross. **c** Ellipse

step is therefore required to account for this. The method employed in this study is one of 'background equalisation'.

Process flow diagrams consist of a lot of 'background'—the page they are drawn upon, with relatively little 'foreground'—the lines and boxes of the image. Heavy blurring of the image, by sequentially averaging large sections of the image, will therefore favour the background colour in a given area. By using a Gaussian blur with a large structuring element (e.g. 151 × 151 pixels), a version of the image essentially consisting of brighter or darker patches of background can be created. The intensity value of each pixel in the original image can then be divided by the intensity value of the blurred image. This has the effect that pixels with similar values in the two images (i.e. light pixels in light patches and dark pixels in dark patches) are evaluated close to 1. These are likely to be areas of background, and therefore, regardless of the local lighting conditions, background pixels can be given the same value. Conversely, dissimilar values in the two images are exaggerated. For example, a light pixel (250) in a dark patch (10) is given a relatively high value (25) while a dark pixel (10) in a light patch (250) is given a relatively small value (0.04). These newly calculated values are then rescaled to the original 0 to 255 range to create a version of the image where the background has been equalised. Thresholding can then more accurately identify the foreground parts of the image from the background (Fig. 3).

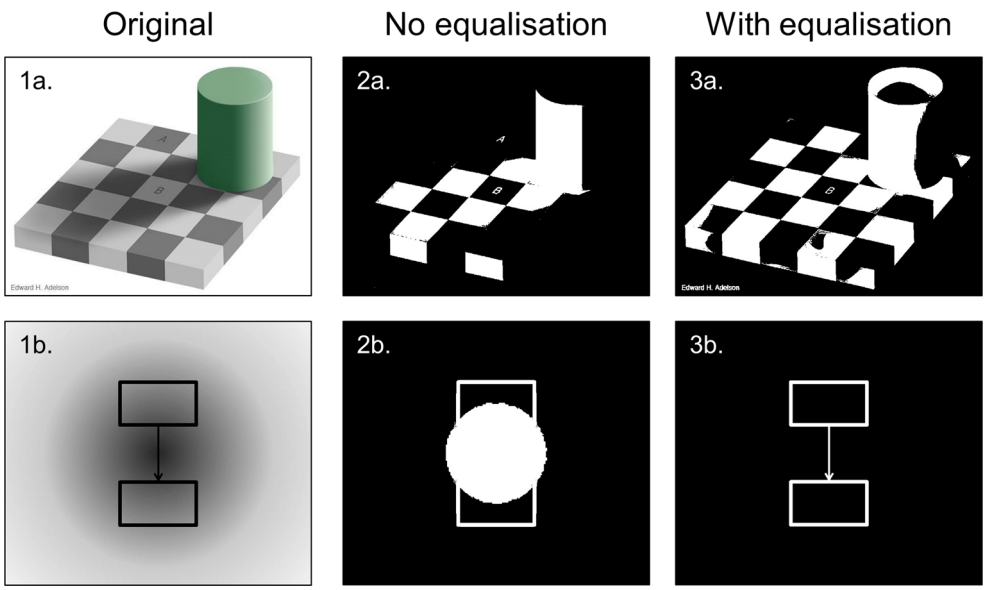



**Fig. 3** The effect of background equalisation on thresholding of images—the checker shadow illusion (a) and a simple process flow diagram with an uneven background (b). Panel (1) shows the original image. Panel (2) shows the thresholded image with no background equalisation. Panel (3) shows the thresholded image after background equalisation

#### 2.2.6 Dilation, erosion and closing

Once the lines and boxes of a flow diagram have been isolated from the background, a related set of operations—dilation and erosion—can be used to 'clean up' messy hand-drawn diagrams. These operations are used to enhance thin or faint lines, and close gaps in poorly drawn or badly thresholded boxes or lines. In each of these operations, a small structuring element is used (e.g. 5 × 5 pixels), which scans across the image and creates a parallel version. In a dilation operation, if any of the points of the structuring element overlaps with a white area of the original image, the area corresponding to the current position of the structuring element in the new image is set to white; otherwise, it is set to black. This has the effect of dilating thin lines and filling gaps within and between lines (Fig. 4, panel 2). Erosion is the opposite of this action. If any of the points in the structuring element are identified as black, the whole region is set to black—i.e. only parts of the image where the whole of the structuring element falls within a region of white pixels remain white. This has the effect of eroding the edges of lines and removing small patches or dots (Fig. 4, panel 3).

Closing is a dilation operation followed immediately by an erosion operation. The dilation serves to close gaps in lines as they merge together, while the subsequent erosion returns the lines to their original width but with the gap closed (Fig. 4).

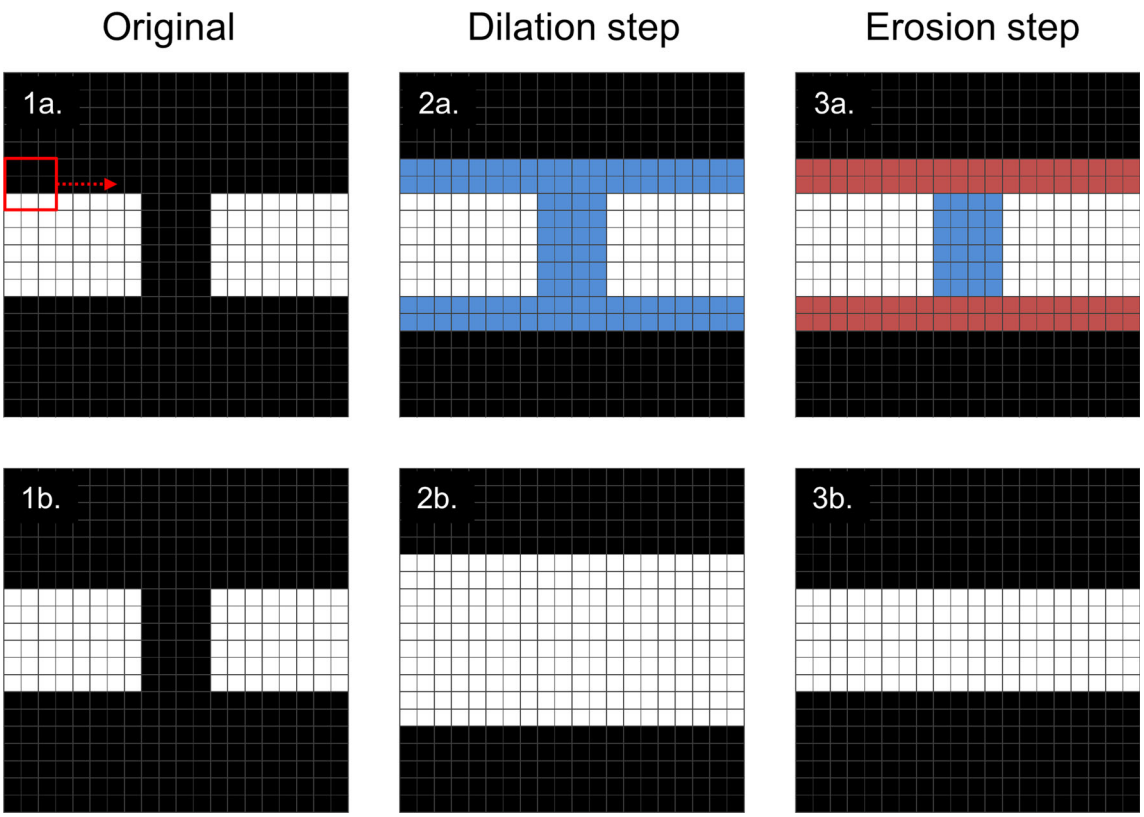



**Fig. 4** The effect of dilation and erosion (closing) with a 3 × 3 square structuring element on a gap in a line. Pixels added in the dilation step are shown in blue in panel (2a). Pixels removed in the erosion step are shown in red in panel (3a). The bottom row of panels (b) shows the actual effect of these steps. The red box in panel (1a) shows the structuring element at the first position at which a dilation operation will be performed as it scans across the image in the direction of the arrow

#### 2.2.7 Contour detection and simplification

Contour detection identifies contiguous edges in an image. This is an important step in finding the boxes in a process flow diagram. The contour finding method used in this study is an implementation of the edge following algorithm described by Suzuki and Abe (1985) in the OpenCV software library (Bradski 2000). Simply put, this algorithm scans a binary image (where black pixels are represented as 0 and white pixels are represented as 1) to find an 'edge pixel', i.e. one where there is a change from 0 to 1 between neighbouring pixels. It then looks for adjoining '1' pixels in the 8 pixels surrounding this pixel that would denote a continuation of this edge. As the name suggests, it then follows this edge, marking the pixels it has already examined as it goes, until it returns to the original point. Each of the pixels in this edge is recorded as a contour. The algorithm then continues scanning for a new edge, which it has not yet visited, until all contours have been identified (Fig. 5).

The list of pixels that constitutes a contour can be simplified into line segments using 'simple chain approximation', where runs of pixels representing a horizontal, vertical or diagonal line can be reduced to their end points. These line segments can be further simplified, using an implementation of the Douglas-Peucker algorithm (Douglas and Peucker 1973), to generate simplified shapes. This algorithm iteratively takes a section of a contour, finds the point furthest from the endpoints and deletes any points which lie within a given distance (denoted as epsilon (ε)) from the lines described by these three points (Fig. 6). This process is repeated until the minimum number of points remains. Contours with only four remaining points can therefore be identified as boxes in the process flow diagram (Fig. 5, panel 3, blue contours).

#### 2.2.8 Flood filling

Flood filling is used to identify connections between boxes in the process flow diagram. Specifically, it is used to identify contiguous areas of the same colour within an image. In the case of the flow diagrams, these contiguous areas represent a white line linking two white boxes. This is a simple operation, familiar from drawing software such as Microsoft paint, where a single point is chosen and all adjacent points of the same colour are changed to another colour.

Arrows which link boxes in process flow diagrams, particularly hand-drawn flow diagrams, can take a variety of forms, with curves, corners etc. rather than simply being straight

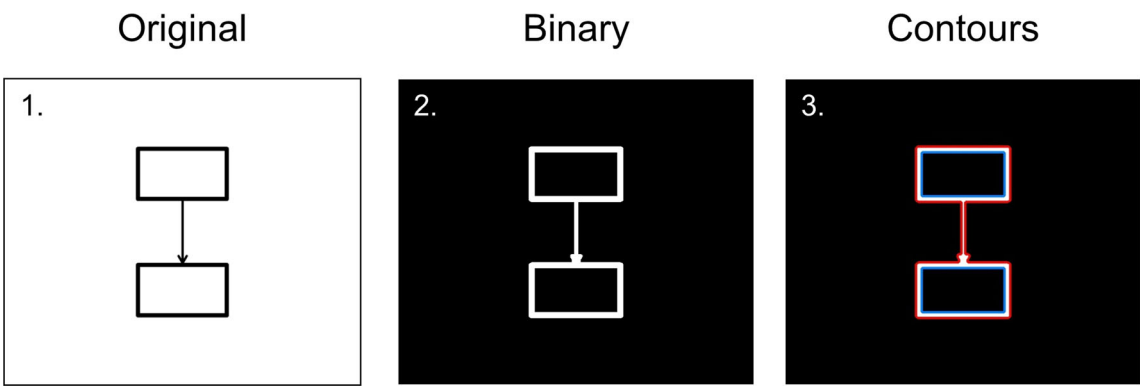



**Fig. 5** Contour detection for a simple flow chart. Three edges are found (1–3). Two of these (the inside edges of the boxes) can be reduced to four points and are classified as boxes (blue). The remaining edge is not classified as a box (red)

lines. Flood filling is therefore a more reliable technique than line detection to establish which boxes are connected.

In order to find connections between boxes, contours identified as boxes can be masked out (set to black, Fig. 7, panel 2) and selectively redrawn in pairs (in white) (Fig. 7, panels 3a and 4a). A flood fill of a third colour can be initiated at the centre point of one box (Fig. 7, panels 3a and 4a), and the colour of the centre point of the second box can be assessed (Fig. 7, panels 3b and 4b). If the chosen colour is detected at this point, then the boxes are linked. All possible pairs of boxes are assessed one by one to establish where the connections are in the flow diagram.

## 2.3 Description of basic computer vision pipeline

The operations described above were combined to create an initial pipeline to identify unit processes and their connections from an image of a process flow diagram. The steps in this pipeline are described in Table 1.

Trial and error during the development of the pipeline led to the introduction of two filtering steps for boxes (steps 11 and 12). In step 11, the area of each of the boxes identified is calculated, and all boxes with an area smaller than a given percentage of the area of the largest box are disregarded. These small boxes detected as during the contour detection may be from text (e.g. the loops in a capital B or D) or from artefacts in the original photograph (e.g. graph paper).

In some situations, the outside edge and the inside edge of the same box can be identified as two different contours. In step 12, the Euclidean distance between the vectors representing the corners of each pair of boxes is calculated, and boxes closer than a given threshold value are considered to be the same box. The smaller of the boxes is discarded.

This pipeline was implemented in a new python package called *lcopt-cv*, using computer vision tools from the OpenCV software library (Bradski 2000). Variables which determine the operation of the pipeline, including the threshold intensity level for step 5, the number of dilation iterations for steps 6 and 14, thresholds for steps 11 and 12 etc., can be set interactively within the software to tweak the pipeline for individual images.

The output of the pipeline is a list of numbered boxes, with their coordinates in the image, and a list tuples denoting the links between these boxes, e.g. (1, 3) means box 1 is linked to box 3.

## 2.4 Heuristic processing

Once the boxes and links have been identified, knowledge of commonly held conventions that are used in drawing process

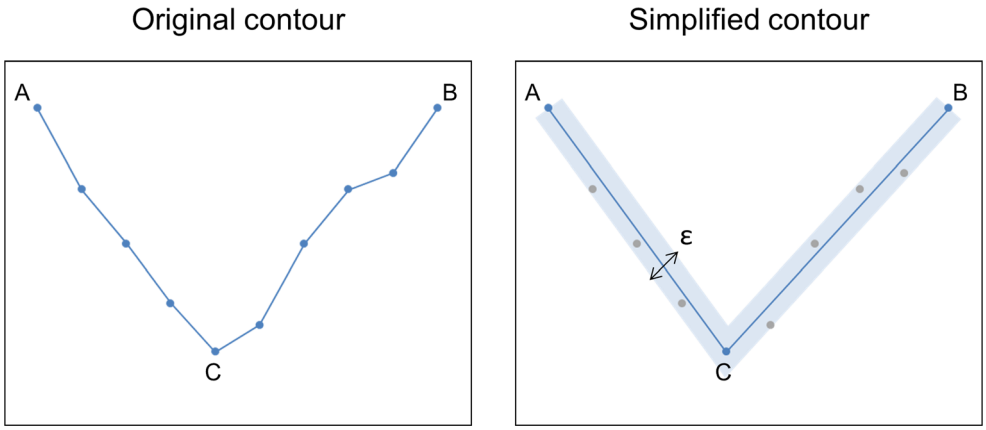



**Fig. 6** Diagram showing the first step in the contour simplification of a section of a contour. The furthest point from the endpoints (A, B) is identified (C), and any point within ε of the lines AC and BC are removed from the contour

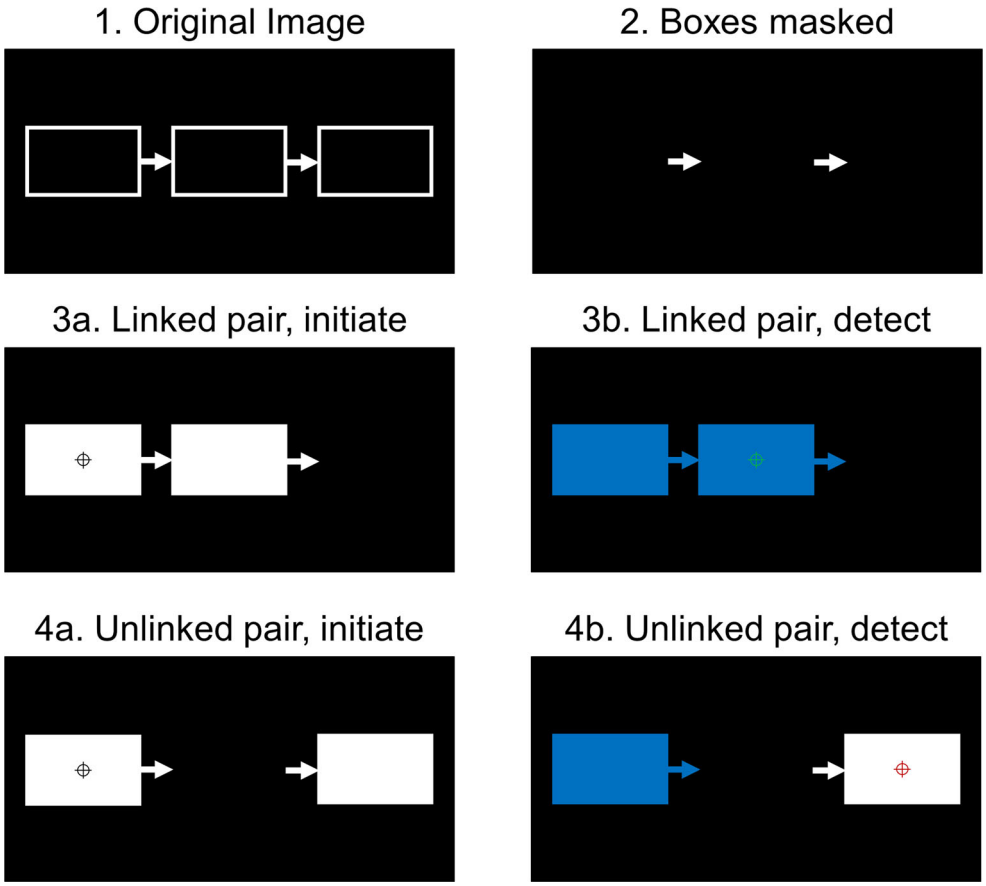



**Fig. 7** Flood filling for link detection. Boxes are masked (panel 2) and then selectively redrawn in pairs (panels 3a and 4a). Linked boxes will take on the same colour after a flood fill is initiated at the centre point of one box (panels 3a and 3b), while unlinked boxes will remain different colours (panels 4a and 4b). The black crosshairs represent the initiation of a flood fill, and the green/red crosshairs denote the detection of the fill

flow diagrams allow the processing of the image to be further refined. This is referred to here as 'heuristic processing', as it is based on known heuristics or 'rules-of-thumb' rather than mathematical relationships. Heuristic processing allows us to add important information to the processing pipeline.

In addition to knowing which boxes are linked in a process flow diagram, it is important to know the direction of the links, in order to ascertain which processes are inputs to other processes, or rely on outputs from other processes. This is a trivial exercise for a human observer, as the links are likely to be drawn as arrows; however, the wide variety of possible arrowhead styles which may be employed in a hand-drawn diagram are such that this cue is difficult for a computer to assess. In addition, it is common when drawing flow diagrams to merge arrows where multiple inputs supply the same unit process. This leads to spurious connectivity between boxes.

#### 2.4.1 Directional heuristic

Writing systems derived from Greek alphabet, including English, follow a left-to-right, top-to-bottom directional pattern. It is reasonable therefore to assume, as a rule-of-thumb, a cognitive bias towards left-to-right, top-to-bottom layout of links in a process flow diagram. By comparing the $x$ and $y$ coordinates of the centre points of linked pairs of boxes, it is possible to work out where they are relative to one another on the page, and apply the heuristic that links should preferentially go from left to right, then from top to bottom (Fig. 8). By rounding down these coordinates to a given tolerance (20 pixels by default), it is possible to effectively 'snap' the boxes to a grid, helping to refine the heuristic to account for drawing errors where box A is slightly to the right of but clearly above box B to correctly assign the link from A to B.

#### 2.4.2 'Prefer linked' heuristic

The directional heuristic, while providing an overall indication of the direction of most links in a process flow diagram will often be broken in order to save space or simplify a diagram, for example where a unit process has multiple inputs from the technosphere, some may be drawn above, and some below the unit process box in order to use the space more effectively (e.g. Fig. 8, panel 2). However, in addition to the reading direction, we also know that LCA process flow diagrams usually consist of multiple inputs from the technosphere going into a system of linked unit processes describing the life cycle. A unit process in the life cycle process chain will usually have at least two links—incoming from the upstream process (or at least one technosphere input) and outgoing to the downstream process. Inputs to unit processes from the technosphere will conversely

**Table 1** Steps in the image processing pipeline to identify boxes and connections between boxes in a process flow diagram

| No. | Step | Description |
|---|---|---|
| 1 | Load image from file | Load image into memory as a 3D matrix in RGB format. |
| 2 | Resize image to manageable size (width of 750 pixels) | Reduce the image size to save processing capacity. |
| 3 | Conversion to greyscale | Convert the image to a 2D matrix. |
| 4 | Background equalisation | Equalise background to account for inconsistent lighting and help distinguish useful features (boxes and lines) from the background of the image. |
| 5 | Thresholding (conversion of image to binary format) | Isolate the foreground of the image to facilitate further processing. The drawn process flow diagram should appear white on a black background. |
| 6 | (Optional) Dilation operation with 5 × 5 elliptical element | If the image is very faint, or the lines of the drawing are very thin, a dilation operation can used to enhance the foreground of the image. |
| 7 | Closing operation with 5 × 5 square element | Gaps in the drawing are filled using a closing operation. This helps the contour finding stage to correctly identify contiguous contours. |
| 8 | Contour detection | The contour detection algorithm identifies possible boxes in the flow diagram and stores their coordinates in memory. |
| 9 | Contour simplification | The contour simplification algorithm reduces the coordinates of the stored contours to their key points by identifying likely corners. |
| 10 | Identify boxes (4-point contours) | Simplified contours with four remaining coordinates are assumed to be boxes and are stored as a list of candidate boxes. |
| 11 | Filter out small boxes (detection errors) | The area of the largest box in the candidate list is calculated and boxes with an area less that a set threshold percentage of this area are removed from the candidate list. This removes image artefacts (e.g. from lines on the paper), and elements of the foreground which are not boxes, such as the loops in text characters (e.g. P, B, D). |
| 12 | Filter out duplicate boxes (inside and outside edge of single box) | The corner coordinates of each remaining box are compared in a pairwise fashion using Euclidian distance. If two boxes are found to be duplicates of one another (e.g. the inside and outside edge of the same box), the smaller of the boxes is removed from the list. Each box in the final list is given a number to identify it. |
| 13 | Mask boxes | A copy of the image from step 7 is created and black boxes corresponding to the list of boxes identified in step 12 are drawn onto this image to mask out the identified boxes. The resulting image should consist only of the lines between the boxes in the flow diagram, (e.g. Fig. 7, panel 2). |
| 14 | Dilate remaining lines with 5 × 5 elliptical element | A dilation operation is carried out on the image from step 13 containing only lines to enlarge the lines. This ensures that the lines overlap the boxes and form a contiguous area for the flood filling step, e.g. in situations where the line and the box did not quite intersect in the original drawing. |
| 15 | Redraw pairs of boxes and check connectivity | Calculate connectivity between pairs of boxes using the flood filling method described in Section 2.2.8. Store each connection in a list as a pair of box ID numbers. |

only have one link—outgoing to the unit process. Thus, for a given link between two boxes, the number of connections each box is involved with may be a useful indication as to the likely direction of the link, and the typology of the box (Fig. 8). In the 'prefer linked' heuristic algorithm, the number of links each box is associated with is calculated. For each link in the diagram, if the link goes from a box with more than one link to a box with only one link, it is reversed. Links between boxes both with more than one link retain their original direction (Fig. 8, panel 3).

In situations where the heuristics designed to assign link directions fail to do so accurately, links can be manually reversed using the *lcopt-cv* GUI.

### 2.4.3 Unstacking heuristic

Another common convention used to save space in LCA process flow diagrams is the merging of arrows where there are multiple inputs to the same process (Fig. 9, panel 1). This can lead to problems in identifying the connectivity of boxes using the flood filling approach, as the merged arrow leads to a contiguous area between two boxes which are not meant to be linked (Fig. 9, panel 3).

One feature which can be used to combat this however is the convention that multiple inputs of this type tend to be 'stacked', that is vertically or horizontally aligned with one another. Such stacks can be identified by grouping boxes based on the $x$ and $y$ coordinates of their centre points (rounded to given tolerance) to identify vertical and horizontal stacks respectively. The topmost, bottommost, leftmost and rightmost points of the boxes in the stack can then be identified, allowing this region of the image to be isolated (Fig. 9, bottom row). The flood filling linking procedure can then be repeated in these isolated sections of the image, so that 'real' links within a stack can be identified and kept, and spurious links can be removed (Fig. 9, panels 5 and 6).

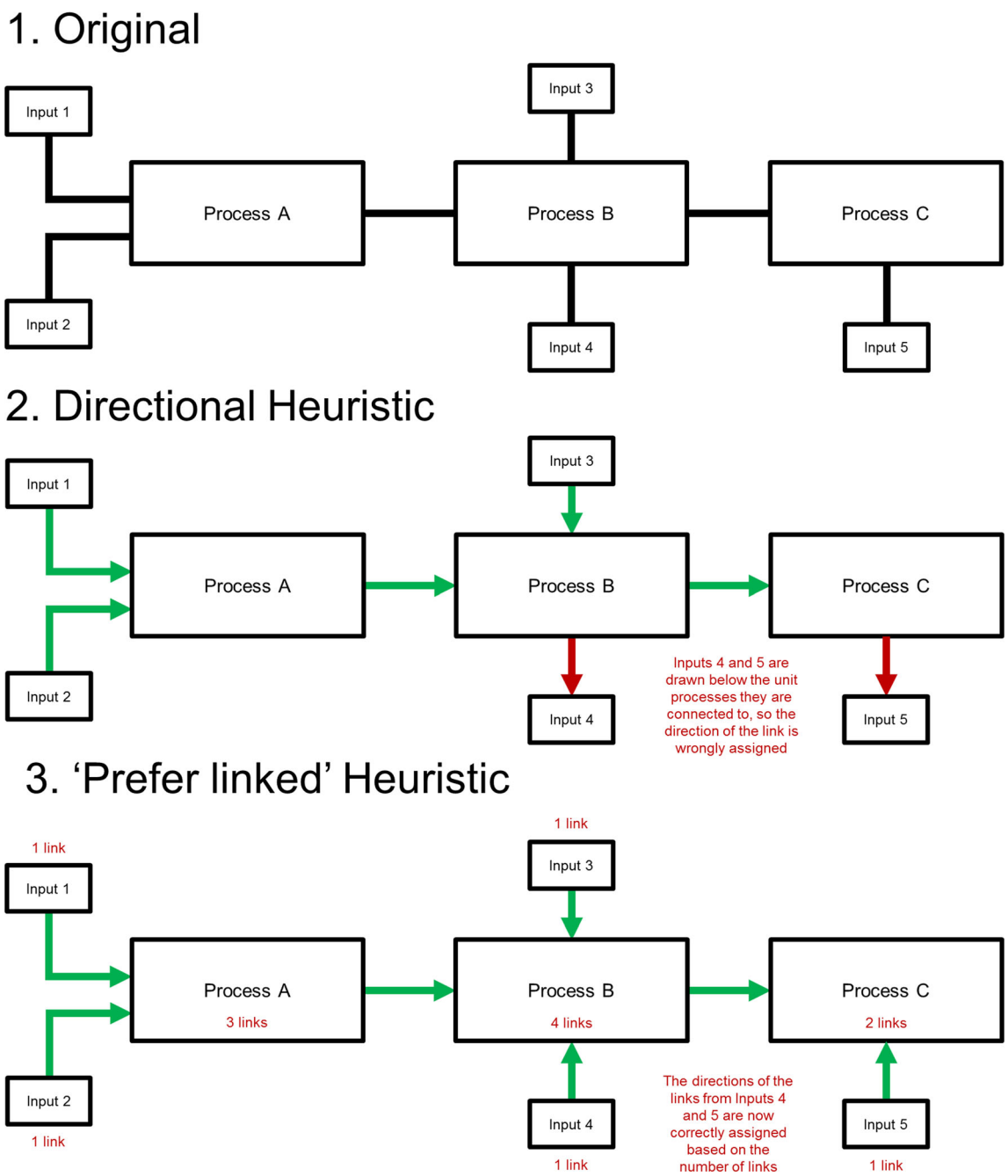



**Fig. 8** Sequential application of the 'directional' and 'prefer linked' heuristics. Panel (1) shows the links and boxes in the example flow diagram from Fig. 1 with the directions of the arrows removed. Panel (2) shows the directions assigned to each link following the 'directional' heuristic. The positions of inputs 4 and 5 lead to their associated link directions being wrongly assigned. Panel (3) shows the directions assigned following the 'prefer linked' heuristic. The wrongly identified links from inputs 4 and 5 are reversed

## 2.5 Generation of LCA model

Once the boxes and links in an image have been identified, the next step is to convert this information into an LCA model. This consists of four steps: naming the boxes, assigning types to the boxes, assigning technosphere and/or biosphere exchanges and reviewing link directions.

### 2.5.1 Naming boxes

While handwriting recognition is possible for well-defined tasks with a small vocabulary (e.g. automated postal city/postcode matching) or a single writer (with known handwriting samples), large vocabulary unconstrained handwriting recognition is a difficult task (Frinken and Bunke 2014). The

**Table 2** Default settings

| Stage | Setting | Default value | Pipeline step |
|---|---|---|---|
| Image processing pipeline | Equalise background | Yes | 4 |
| | Threshold intensity | 130 | 5 |
| | Box dilation iterations | 0 | 6 |
| | Box closing iterations | 1 | 7 |
| | Box size threshold | 0.1 | 11 |
| | Duplicate threshold (Euclidian distance) | 10 | 12 |
| | Line thickness for box mask | 12 | 13 |
| | Line dilation iterations | 1 | 14 |
| Heuristic processing | Use directional heuristic | Yes | – |
| | Use 'prefer linked' heuristic | No | – |
| | Use 'unstacking' heuristic | No | – |

relatively small number of boxes which need to be transcribed in an LCA process diagram is such that manual naming of boxes is currently the preferred approach. The bounding coordinates of each box are recorded as part of the image processing pipeline; therefore, the corresponding part of the image can be cropped and shown to a human user for them to transcribe (Fig. 10).

### 2.5.2 Assigning types to boxes

Boxes in an LCA process flow chart of the type considered in this study should represent one of three things:

1) A unit process/transformation process
2) An intermediate exchange with the background (technosphere) system ('input')
3) An elementary exchange with the biosphere ('emission')

Certain characteristics of these three types of process can help us heuristically classify the boxes which have been detected. Primary among these is that technosphere and biosphere exchanges should only be associated with one link. Therefore, if a box is associated with more than one link, it must be a transformation process. Secondly, if a box is only associated with one link, and the direction of that link is from that box to another, the likelihood is that this is an input from

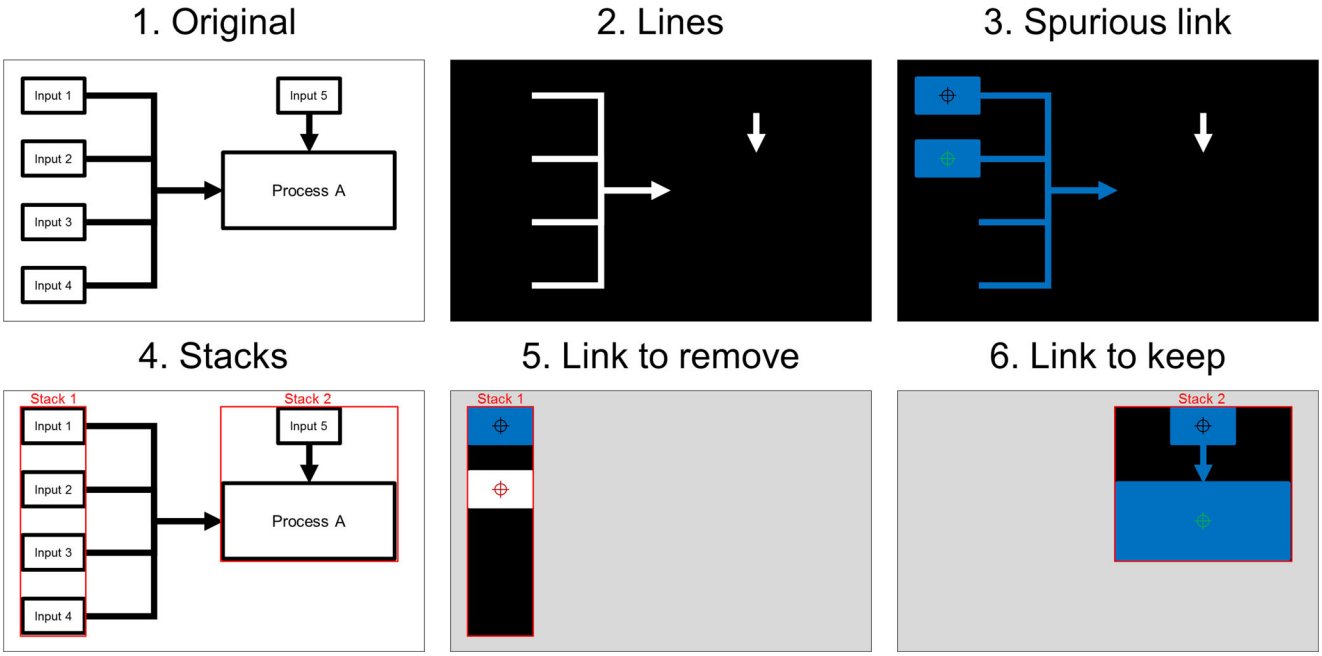



**Fig. 9** Unstacking heuristic. The merged arrow between inputs 1 to 4 and process A (panel 1) leads to spurious links being identified (e.g. between input 1 and input 2, panel 3) by the flood filling process. Stacks of boxes are identified (panel 4), and the flood filling process is repeated only in the parts of the image identified as a 'stack'. Spurious links are identified and removed in stack 1 (panel 5), while real links are identified and kept in stack 2 (panel 6). The black crosshairs represent the initiation of a flood fill, and the green/red crosshairs denote the detection of the fill

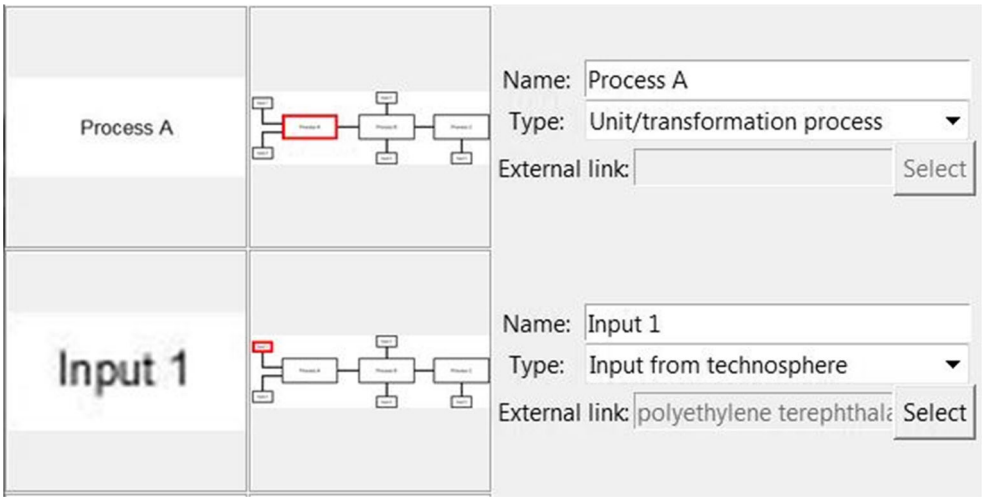



**Fig. 10** Naming boxes, assigning types and assigning external links using the *lcopt-cv* GUI

the technosphere. These features allow each of the boxes in the diagram to be 'pre-classified'. The classification can then be reviewed by a human user (Fig. 10).

The directions of the links between boxes classified as inputs or emissions and unit processes can subsequently be revised, if necessary, to ensure they are recorded correctly.

#### 2.5.3 Assigning technosphere/biosphere exchanges

The *lcopt-cv* software developed for this study links to the lcopt/Brightway (Mutel 2017; Joyce 2017) ecosystem of open-source, Python-based LCA software packages. As a result, inputs from the technosphere can be assigned from the ecoinvent database (for those users with an ecoinvent licence) (Wernet et al. 2016) or the FORWAST database (Forwast 2007) using an inbuilt search function. Likewise, biosphere exchanges can be assigned from the *biosphere3* database.[1] This means that the LCA model generated from the process flow diagram can be pre-populated with links to these external databases.

#### 2.5.4 Review of link directions

There is a possibility that even in spite of the heuristic processing in the preceding steps, the direction of links between unit processes may be misclassified. Therefore, the final step prior to the generation of the LCA model is to allow the user to review the model which the processing pipeline has identified, and reverse these links if necessary (e.g. Fig. 14).

#### 2.5.5 Generation of finalised LCA model

Once these four steps have been completed, an LCA model can be generated by *lcopt-cv* in the format used by *lcopt* (Joyce 2017) and saved as an .lcopt file. This model can be opened either directly from *lcopt-cv* or opened later using the *lcopt* software.

[1] A consolidated biosphere database generated by Brightway2 based on the ecoinvent v3 biosphere flows

### 2.6 Process flow diagrams and testing protocol

Four different process flow diagrams were created to test the image processing pathway: a simple, computer-generated diagram, a simple hand-drawn diagram, a complex hand-drawn diagram and a diagram designed to test the 'link unstacking' heuristic, detailed in Table 3.

Each of the test diagrams was opened in the proof-of-concept software, and the results with the default settings (detailed in Table 2) were recorded. The default settings were originally derived from through trial and error. The values used were then varied using trial and error, and the feedback from intermediate images (see Fig. 11) to attempt to fully recognise the diagram, and the measures taken recorded. The LCA model was subsequently generated as described above, with arbitrarily chosen technosphere inputs. Each model was then opened in *lcopt* to manually verify that it had been correctly generated.

## 3 Results

The results of the image processing for each of the four process flow diagrams are summarised in Table 4.

### 3.1 Simple, computer-drawn diagram

Figure 11 shows a screenshot of the processing of the simple, computer-generated diagram. Using the default settings, all of the boxes and links were identified. Once the 'prefer linked' heuristic was applied (by clicking the check box and clicking 'reprocess image', Fig. 11), the link directions were correctly resolved. The inputs and unit processes were correctly pre-assigned, and the exported LCA model was correct.

### 3.2 Simple, hand-drawn diagram

Figure 12 shows a screenshot of the processing of the simple hand-drawn flow diagram. The default settings correctly identified all of the boxes and links. 'Process 1' was pre-classified as an input rather than a unit process. Once 'Process 1' was reclassified, the LCA model was correct.

### 3.3 Complex, hand-drawn diagram

Figure 13a shows a screenshot of the processing of the complex hand-drawn diagram using the default settings. Six of the nine boxes and their corresponding links were identified. The image intermediates (Fig. 13a, left hand side) showed that the lines in the graph paper were being detected as part of the foreground image in the thresholding step. These spurious features then affected the contour/box finding step. Successively decreasing the threshold level by 5 and reprocessing the image found that a threshold level of 105 successfully isolated the foreground from the background, and all boxes and links were identified (Fig. 13b).

All but one of the link directions were identified correctly, the exception being the link from 'Gas' (Fig. 13b, box 1) to 'High-temperature processing' (Fig. 13b, box 5). The generated LCA model pre-classified all unit processes correctly and correctly classified 'Coke' as a technosphere input; however, 'Gas', '$CO_2$' and 'Dust' were incorrectly classified as unit processes. These were manually reclassified using the dropdown boxes shown in Fig. 10. Representative processes from the ecoinvent and biosphere3 databases were chosen to represent the technosphere and biosphere exchanges.

Figure 14 shows the proposed LCA model in the final review step in *lcopt-cv*. The LCA model was generated correctly. The final model as generated in the *lcopt* software is shown in Fig. 14. A dummy parameter set was added to this model and an LCA calculation successfully performed using lcopt's inbuilt links to Brightway.

### 3.4 Testing of unstacking heuristic

Figure 15 shows the processing result of the fourth process flow diagram before and after the application of the unstacking heuristic. After initial processing with the default settings, all of the boxes were identified correctly, but spurious links were found between each of the boxes A–D in the stack of boxes on the left-hand side of the image. Applying the unstacking heuristic removed all of these spurious links and correctly identified that boxes A–D each have a single link to box E.

## 4 Discussion

The computer vision pipeline, implemented in *lcopt-cv*, was successful in identifying the boxes and links in each of the computer-generated or hand-drawn process flow models

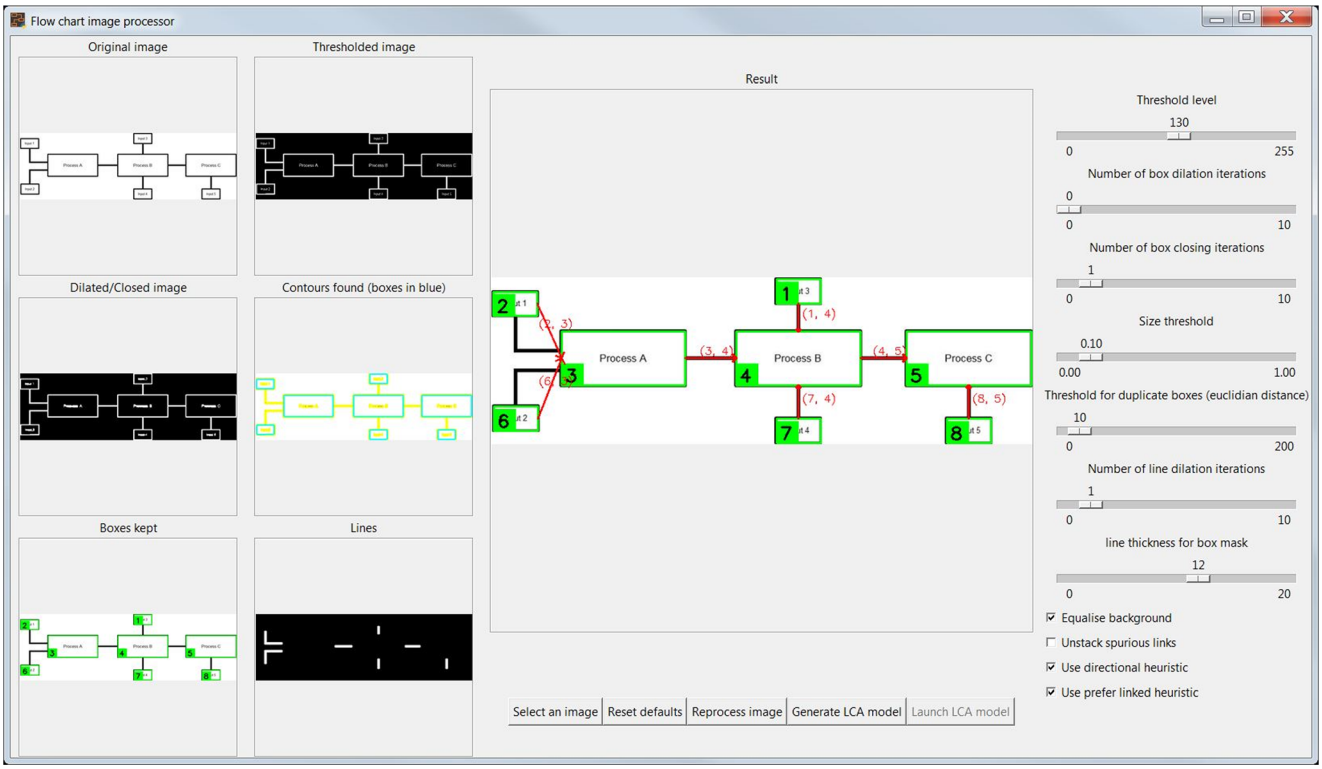



**Fig. 11** Screenshot of *lcopt-cv* for the simple computer-drawn process flow diagram

**Table 3** Process flow diagrams for testing

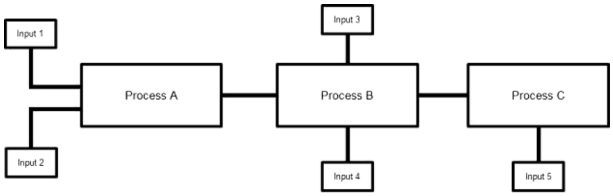
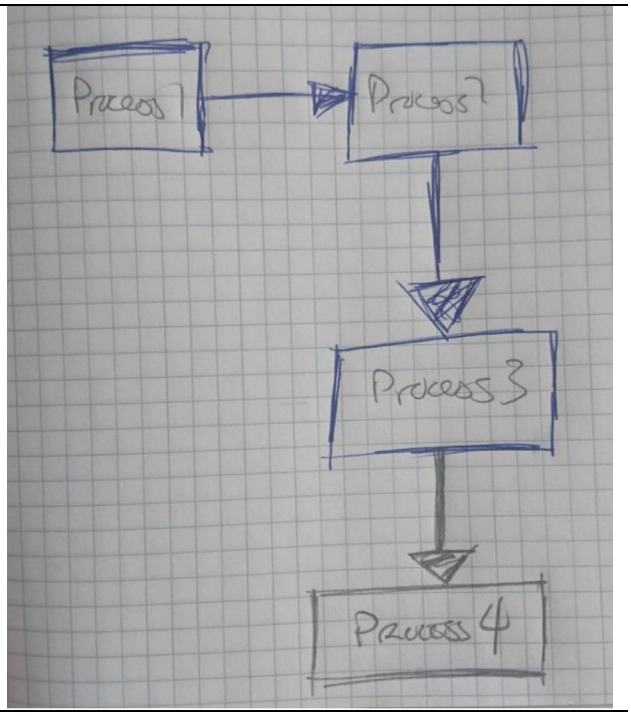
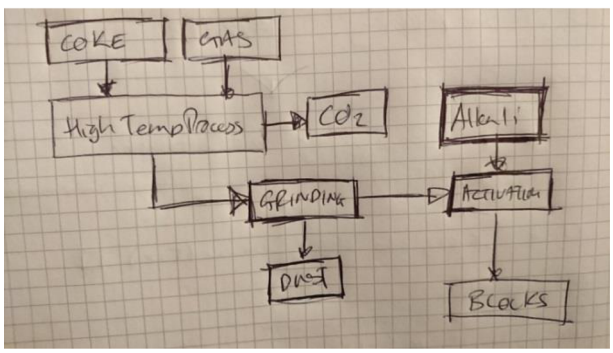
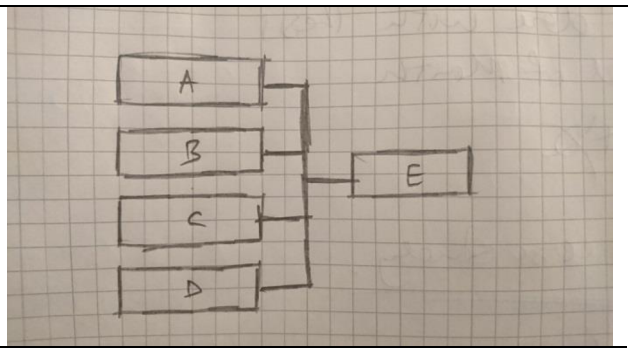
| Description | Image |
|---|---|
| **1. Simple, computer generated**<br>• Plain white background<br>• Black foreground<br>• Boxes perfectly aligned<br>• Created in MS powerpoint and saved as a .jpeg |  |
| **2. Simple, hand-drawn**<br>• Four boxes, each of similar size<br>• Each box links to the next in sequence<br>• All boxes are unit processes<br>• All lines straight and of similar thickness<br>• Reasonable care taken to ensure neatness<br>• Drawn on graph paper<br>• Drawn with blue pen, and amended in pencil<br>• Photographed with imperfect lighting |  |
| **3. Complex, hand-drawn**<br>• Many boxes (9) of differing sizes<br>• Varying number of links per box<br>• Mixture of unit processes, inputs and emissions, based on an actual flow chart from a previous project<br>• One 'elbow' line in addition to straight lines<br>• Thickness of lines varies<br>• Little care taken to ensure neatness<br>• Drawn on graph paper<br>• Drawn with black pen<br>• Photographed with imperfect lighting |  |
| **4. Testing of unstacking heuristic**<br>• Four inputs to one unit process<br>• Link merges prior to connection<br>• Reasonable care taken to ensure neatness<br>• Drawn on graph paper<br>• Drawn with pencil<br>• Photographed with imperfect lighting |  |

**Table 4** Results of image processing for each process flow diagram (PFD) (observed/expected). The adjustments required to fully and correctly classify the PFD. Results for PFD 3 are shown for both the default settings and the first round of reprocessing required

| PFD no. | Settings | Processes | Technosphere inputs | Biosphere exchanges | Links | Link directions | Adjustments required |
|---|---|---|---|---|---|---|---|
| 1 | Default | 3/3 | 5/5 | 0/0 | 7/7 | 6/7 | Apply 'prefer linked' heuristic. |
| 2 | Default | 3/4 | 1/0 | 0/0 | 3/3 | 3/3 | Reclassify box 2 ('Process 1') as process not input. |
| 3 | Default | 3/4 | 3/3 | 0/2 | 5/8 | 5/8 | Decrease threshold to 105. Repeat initial processing. |
| 3 | Reduced threshold (105) | 7/4 | 1/3 | 0/2 | 8/8 | 7/8 | Reverse link direction from box 1 to box 5.<br>Reclassify box 1 ('Gas') as input.<br>Reclassify boxes 3 ('$CO_2$') and 8 ('Dust') as biosphere exchanges. |
| 4. | Default | 1/1 | 4/4 | 0/0 | 8/4 | 4*/4 | Apply 'unstacking' heuristic. |

*Directions of correctly identified links

presented above. The intermediate images and intuitive sliders and checkboxes of the GUI allowed the image recognition settings to be quickly refined in order to correctly identify the process flow diagram correctly. The directional heuristic improved the identification of the correct link directions, while the 'unstacking' heuristic correctly removed the spurious links caused by the commonly carried out merging of arrows seen in many LCA process flow diagrams.

The computer vision pipeline and *lcopt-cv* software presented here represent a potentially important step in the simplification of the LCA workflow. It may play a particularly important role in LCA education, as it removes the bottleneck which occurs between the important concepts of how to set up an LCA and how to interpret the results of an LCA. It allows students, either in a university setting or in a corporate training environment, to sketch out their own example of an LCA and then create and analyse the resulting LCA model without the need to learn how to use an LCA software package, and without the time-consuming model setup step. This could accelerate their learning and allow them to grasp the basics of LCA in as little as a single day. Experiential learning—'learning by doing'—has been highlighted as an important approach in the teaching of LCA to MBA students (Sroufe 2013). The introduction of LCA software in the third iteration of the course

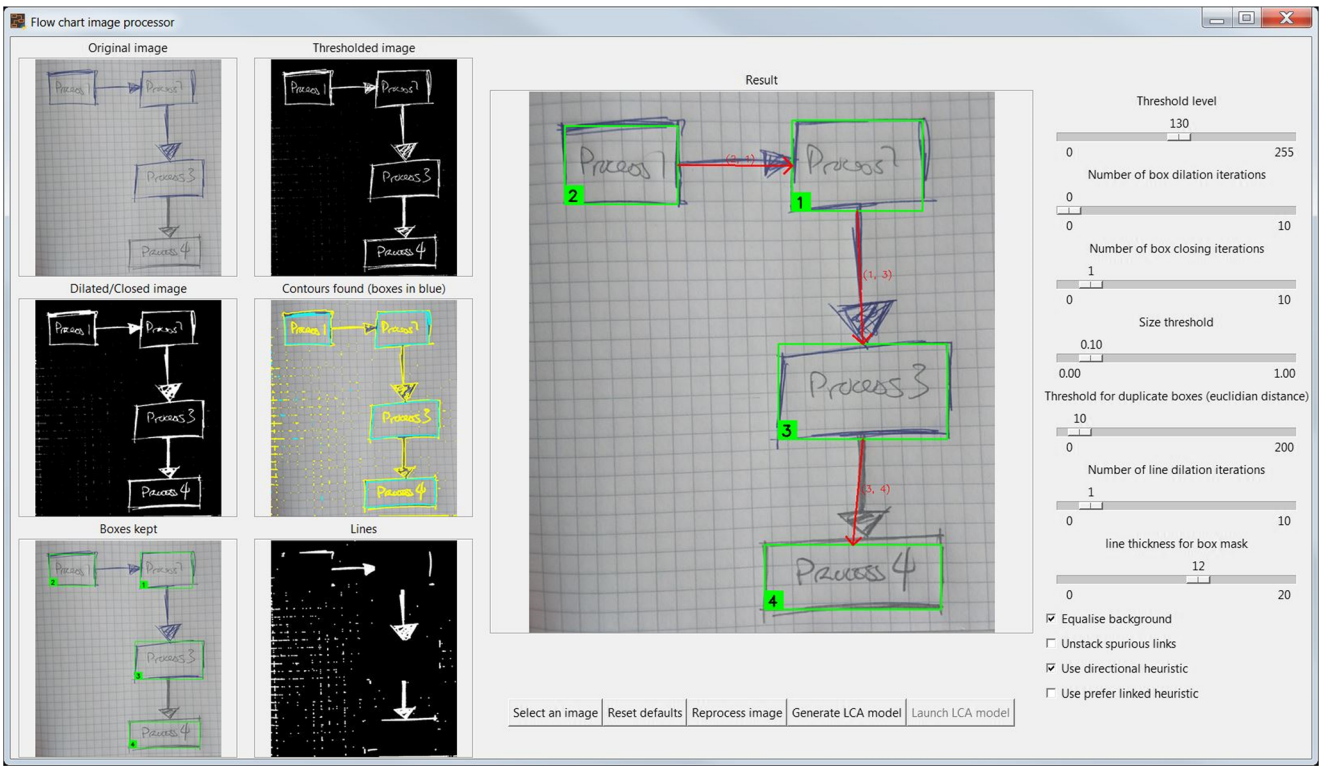



**Fig. 12** Screenshot of *lcopt-cv* for the simple hand-drawn process flow diagram

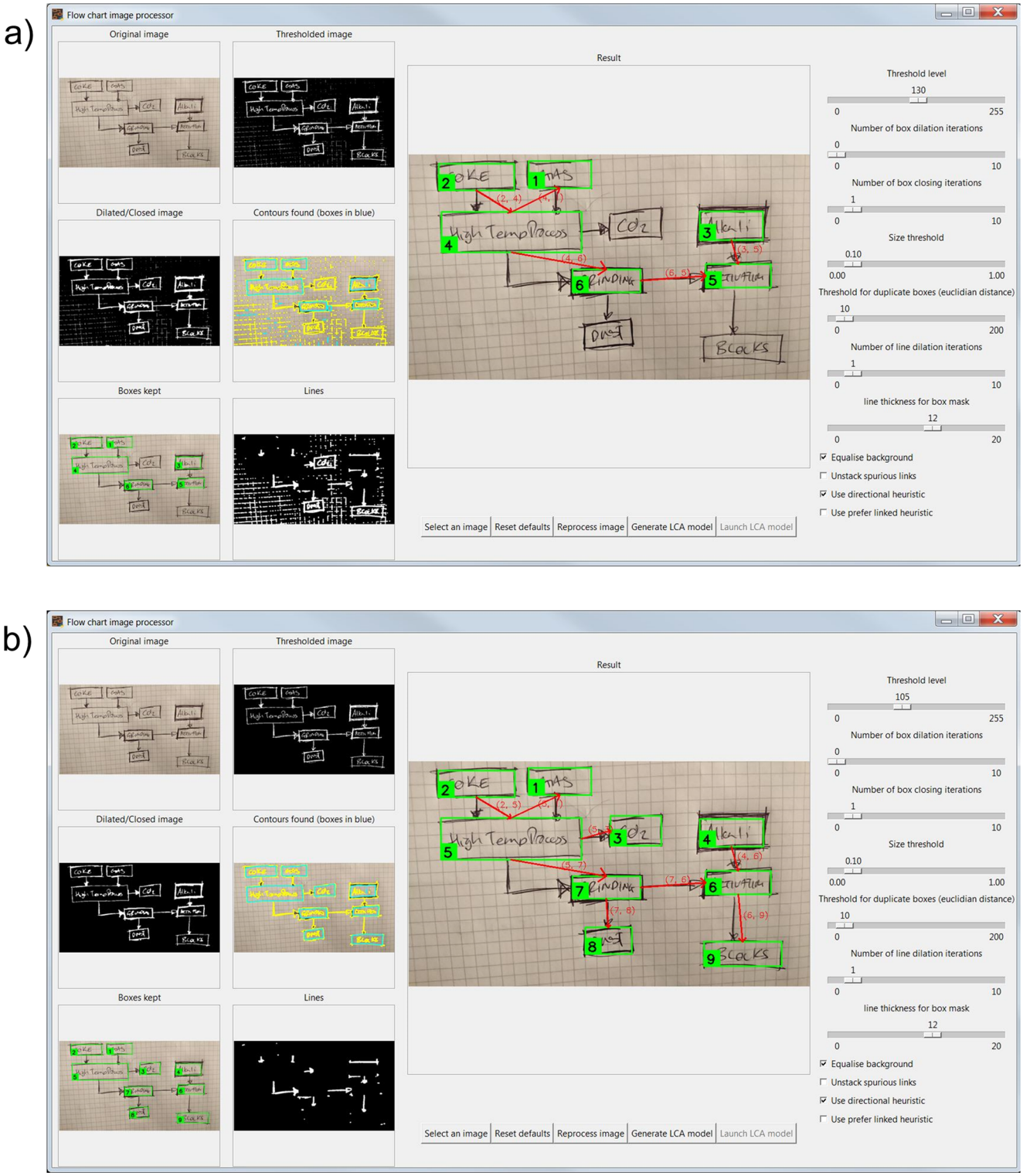



**Fig. 13** Screenshots of *lcopt-cv* for the complex computer-drawn process flow diagram with the default settings (**a**) and with the adjustments made (**b**)

outlined by Sroufe (ibid.) was reported to enrich the learning experience, but required students to complete over 20 online training modules to become proficient in GaBi LCA software. Replacing this learning curve with a technological solution such as that outlined in this study has the potential to enrich the teaching of important LCA concepts to students, such as business students or engineers, for whom a high-level knowledge of LCA is important, but a detailed grasp of the modelling can be gained at a later date if required. The same idea applies in a policy- or decision-making context, where a better

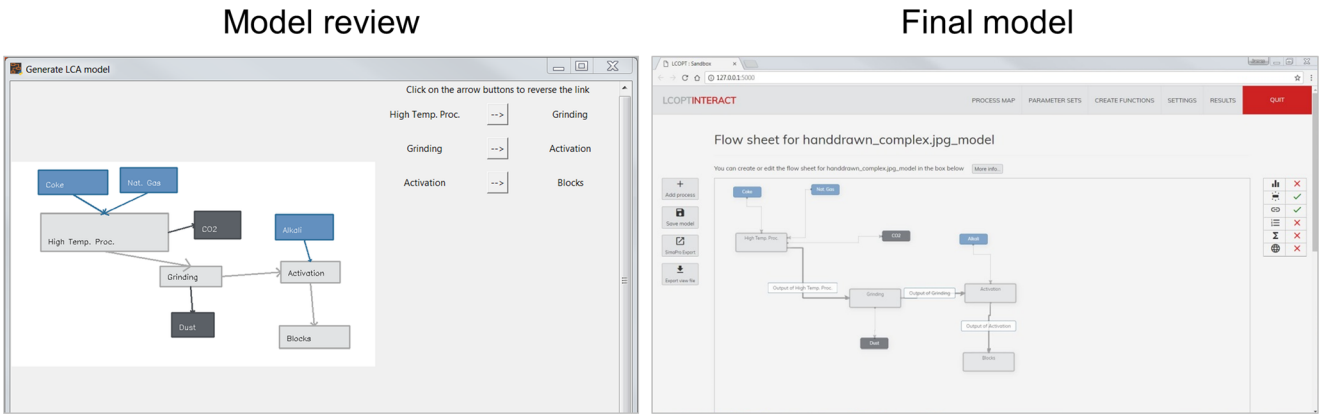



**Fig. 14** Final screen of *lcopt-cv* (left) in which the model can be reviewed and links reversed (optionally), and the full exported model in *lcopt* (right)

understanding of the overall workflow is helpful, but knowledge of the technical details of the modelling is largely unnecessary. Brief, focused and enriched LCA training utilising such technical solutions is likely to prove valuable in these broader contexts too.

The pipeline presented here represents a first step and is not yet meant to provide a perfect solution. One potential pitfall when introducing new technology is that one bottleneck (LCA modelling step) is replaced by another (image processing step). Indeed, the need to tweak the default settings and the knowledge required to understand how to do so may represent a separate learning curve of its own. Technological solutions may exist to overcome this however. Instructing the computer to automatically vary the image processing settings is a relatively trivial task. The challenge is in helping the computer understand when it has discovered the 'correct' answer. Machine learning techniques such as neural networks (Lecun et al. 2015) used for image classification have been successfully employed in complex tasks ranging from directing self-driving cars (ibid.) to classification of fruit (Zhang et al. 2014) and fish (Storbeck and Daan 2001). These techniques require large training datasets which provide an example image and the correct answer. The lack of such a training dataset for LCA flow charts may represent a hurdle to this type of solution. A rule-based, multi-criteria optimisation algorithm using expected features of LCA process flow diagrams may represent a more suitable approach, for example maximising the number of boxes identified, while ensuring all boxes in the diagram have at least one link, with no loops in the network.

One possible alternative to a technological solution would be to mandate drawing rules for flow charts for use with *lcopt-cv*. Drawing rules are common in technical fields, for example the ISO 128 series of standards sets out rules for technical drawing (International Organization for Standardization 2004). In the context of *lcopt-cv*, strict guidance on aspects such as the types of arrows to use may enable the use of pattern matching rather than flood filling for identifying connections. Alternatively, a strict adherence to the left-to-right, top-to-bottom orientation of links or disallowing the use of merged arrows could replace the need for heuristic processing. Such rules however would rather defeat the object of *lcopt-cv*, which is to allow

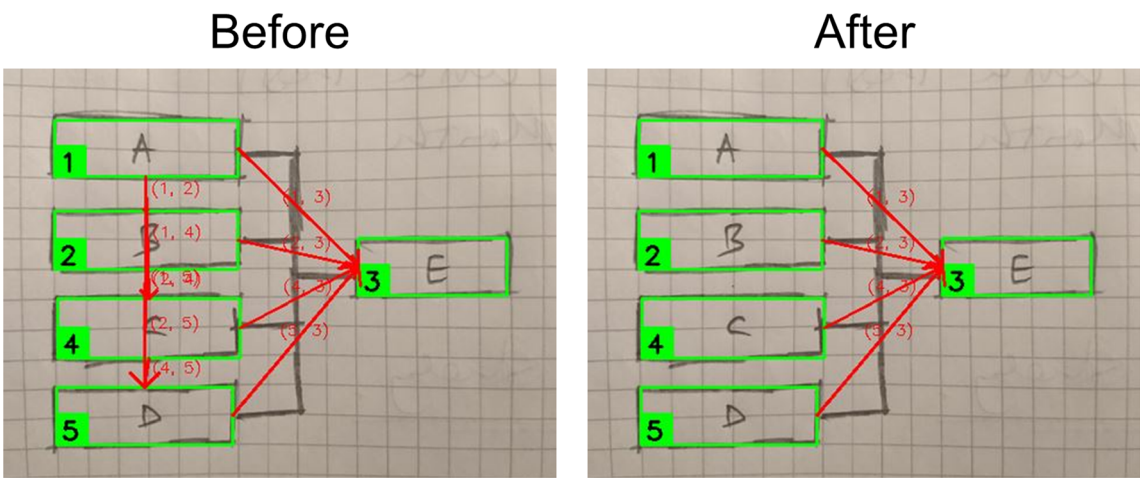



**Fig. 15** Result of image processing for the 'stacked' flow chart before and after application of the unstacking heuristic

simple LCA models to be drawn and analysed quickly and without the need for extensive training. In addition, for hand-drawn pictures, the natural variation in even mandated shapes would potentially be enough to render pattern-matching algorithms unusable. Basic guidelines, which reflect common convention, such as using rectangles to represent unit processes and inputs, or making sure the arrows join onto the boxes in the diagram, should suffice to make *lcopt-cv* usable.

The approach presented here is designed to broaden the range of people who can use LCA, not to replace the need for skilled LCA practitioners. Its use is not intended to completely replace the LCA modelling step. As such, the initial implementation of the pipeline can only deal with simple cases. Recycling loops, internal material or energy transfers and the production of by-products cannot be identified using *lcopt-cv*. While this is a limitation of the approach, for simple systems and for educational purposes, this is a relatively minor concern. One potential future use of such an approach however could be in data harvesting, for example as proposed by the BONSAI initiative (BONSAI 2018). Enhancements to allow complex systems to be analysed may be necessary for a computer vision approach to be applied effectively for this use case.

The initial implementation of *lcopt-cv* also depends upon *lcopt* (Joyce 2017) and *Brightway2* (Mutel 2017) to export and analyse the LCA models it generates. This may represent a limitation for LCA practitioners not familiar with these software packages. As the software develops beyond proof-of-concept, interoperability with other LCA modelling frameworks will be desirable. One potentially important recent development in this regard is the approach set out by Kuczenski (2018) for the transparent and platform-independent disclosure of product system models.

The software presented alongside this study is open-source and publicly available on Github (https://github.com/pjamesjoyce/lcopt_cv). This allows interested members of the LCA community to explore, question and, most importantly, contribute to the underlying code. The future development of, for example, automated recognition algorithms, further heuristics for better diagram identification, or features that are yet to be thought of, all benefit from the pooling of skills, resources and imagination offered by the open-source approach. Open-source approaches to LCA, including *lcopt* for foreground model development (Joyce 2017), Brightway for fast LCA calculations (Mutel 2017), Ocelot for system model generation (Ocelot Project 2017) and the more fully featured OpenLCA software (Ciroth 2007), represent a growing set of useful, adaptable, community-driven tools. The software presented here is a potentially useful addition to this set of tools.

## 5 Conclusions

In this study, the possibility of generating an LCA foreground model from a photograph of a process flow diagram has been demonstrated. This represents a significant short circuiting of one of the most time-consuming steps in the LCA workflow. The time saved is likely to be of benefit to LCA practitioners in general, but there is a particular application of this type of approach in teaching LCA, by allowing new LCA students to skip the steep learning curve associated with LCA modelling in order to more quickly grasp the key concepts of model design and interpretation of results. The proof-of-concept software, *lcopt-cv*, is a further addition to the growing resource of open-source LCA software available to the LCA community.

**Acknowledgements** I would like to thank Anna Björklund for PhD supervision and useful comments on a draft of this manuscript.



## References

Adelson EH (1995) Checker shadow illusion. http://persci.mit.edu/gallery/checkershadow. Accessed 12 Apr 2018

BONSAI (2018) Home—BONSAI. https://bonsai.uno/. Accessed 23 Feb 2019

Bradski G (2000) The OpenCV Library. Dr Dobb's J Softw Tools 25: 120–125

BSI (2011) Guide to PAS 2050—how to assess the carbon footprint of goods and services

Ciroth A (2007) ICT for environment in life cycle applications openLCA—a new open source software for life cycle assessment. Int J Life Cycle Assess 12:209–210

Ciroth A (2012) Software for life cycle assessment. In: Curran MA (ed) Life cycle assessment handbook: a guide for environmentally sustainable products. Wiley, pp 143–157

de Bruijn H, van Duin R, Huijbregts MAJ (2002) Handbook on life cycle assessment. Kluwer Academic Publishers, Dordrecht

Douglas DH, Peucker TK (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartogr Int J Geogr Inf Geovisualization 10:112–122

Forwast (2007) Forwast—Home. http://forwast.brgm.fr/. Accessed 24 Jul 2017

Frinken V, Bunke H (2014) Continuous handwritten script recognition. In: Doermann D, Tombre K (eds) Handbook of document image processing and recognition. Springer London, London, pp 391–425

International Organization for Standardization (2004) Technical drawings—general principles of presentation—part 1: introduction and index (ISO 128-1:2004)

International Standards Organisation (2006) ISO 14044:2006—environmental management—life cycle assessment—requirements and guidelines

International Telecommunication Union (2011) Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios (ITU-R BT.601-7). Recomm ITU-R BT601-7 7:19

Joyce PJ (2017) Lcopt—an interactive tool for creating fully parameterised life cycle assessment (LCA) foreground models. J Open Source Softw. https://doi.org/10.21105/joss.00339

Kuczenski B (2018) Disclosure of product system models in life cycle assessment: achieving transparency and privacy. J Ind Ecol. https://doi.org/10.1111/jiec.12810

Lecun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444. https://doi.org/10.1038/nature14539

Mutel C (2017) Brightway: an open source framework for life cycle assessment. J Open Source Softw. 2. https://doi.org/10.21105/joss.00236

Ocelot Project (2017) Ocelot. https://ocelot.space/. Accessed 12 Apr 2018

Sroufe R (2013) Life cycle assessment within MBA courses: a tool for integrating sustainability. Oper Manag Educ Rev 7:95–130

Storbeck F, Daan B (2001) Fish species recognition using computer vision and a neural network. Fish Res 51:11–15

Suzuki S, Abe K (1985) Topological structural analysis of digitized binary images by border following. Comput Vision, Graph Image Process 30:32–46

Wernet G, Bauer C, Steubing B, Reinhard J, Moreno-Ruiz E, Weidema B (2016) The ecoinvent database version 3 (part I): overview and methodology. Int J Life Cycle Assess 21:1218–1230

World Resources Institute, World Business Council for Sustainable Development (2011) Product life cycle accounting and reporting standard. pp 1–148

Zhang Y, Wang S, Ji G, Phillips P (2014) Fruit classification using computer vision and feedforward neural network. J Food Eng 143:167–177