

GHGT-9

## Algorithm to create a CCS low-cost pipeline network

Tomasz Kazmierczak<sup>a</sup>, Ruut Brandsma<sup>a</sup>, Filip Neele<sup>b</sup>, Chris Hendriks<sup>a\*</sup>

<sup>a</sup>*Ecofys, P.O. Box 8408, NL-3503 RK Utrecht, the Netherlands*

<sup>b</sup>*TNO, P.O. Box 80015, NL-3508 TA Utrecht, the Netherlands*

**Elsevier use only:** Received date here; revised date here; accepted date here

---

### Abstract

Within the EU-funded GeoCapacity project an economic analysis computer tool is developed for the evaluation of carbon capture and storage (CCS) systems comprising of a set of multiple sources of CO<sub>2</sub> and storage locations. As a part of that tool, an algorithm has been developed to create low-cost pipeline networks to connect sources of CO<sub>2</sub> and storage reservoirs.

The main features of the algorithm are:

- Input parameters are the size and location of CO<sub>2</sub> sources and storage reservoirs
- The algorithm determines the trajectory, capacity and costs of the connecting pipes
- If necessary junctions are created, i.e. connections are created between sources and pipelines and/or storage locations and pipelines
- The algorithm has a step wise approach. The pipeline network is built-up by successively adding a pipe in each step, until all sources are connected to storage reservoirs
- In each step all possible combinations are examined and the one with the lowest costs is added to the system
- The pipe costs have a non linear relation with the flow rate
- Cost ranges for pipeline network are presented reflecting uncertainties in data of supply of CO<sub>2</sub> from sources as well in storage capacity of sinks

© 2008 Elsevier B.V. All rights reserved

Keywords: GeoCapacity; CSS; algorithm; pipeline network

---

---

\* Chris Hendriks. Tel.: +31 30 6623393; fax: +31 30 280 83 01.  
E-mail address: [c.hendriks@ecofys.com](mailto:c.hendriks@ecofys.com).

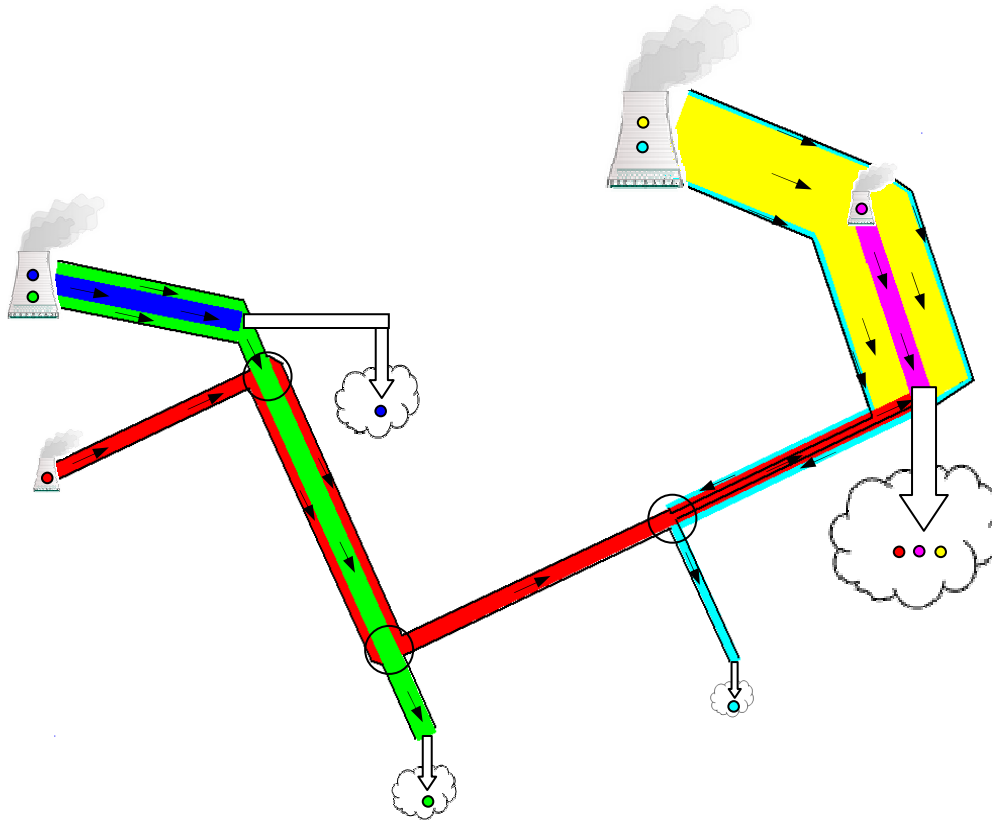


Figure 1 Pipeline network connecting sinks and storage reservoirs. The size of the coloured pipes is a proxy for the capacity of the pipes. The arrows indicate the flow direction. The size of the chimney icon reflects the capacity of the source; the size of the ‘cloud’ icon reflects the size of the storage reservoir. The black circles indicate the location of the junctions. The black thin lines enclose the resulting pipe. For trajectories for which opposite flows (e.g. the red and cyan line) have been calculated pipelines are designed for the net flows

## 1. Introduction

The aim of the EU GeoCapacity project (see Vangkilde-Pedersen et al. [1]), is to assess the European capacity for geological storage of CO<sub>2</sub> in deep saline aquifers, oil and gas structures and coal beds. Other priorities are development of refined methods for capacity assessment, economic modelling and site selection as well as international cooperation, especially with China. The GeoCapacity project will result in an update and extension of the data produced by the earlier EU GESTCO project, which performed a similar task for eight European countries. The latter project also produced a software tool for the analysis of the economic feasibility of carbon capture and storage (CCS) projects by estimating the costs involved in the different elements of a CCS chain. This chain was defined as capture and compression at a single source (source), storage at a single storage location (sink), and the connecting pipeline between the source and sink. The EU GeoCapacity project was tasked to extend the GESTCO project on this aspect, by developing a more advanced CCS analysis tool, capable of handling more realistic scenarios, with multiple sources and multiple storage locations. In addition, the tool should be able to clearly show the uncertainties associated with the analysis of the economic feasibility of CCS projects. As CCS initiatives are being developed in Europe, the uncertainties at various levels in the CCS chain become apparent; it is essential that a feasibility assessment takes into account these uncertainties. Neele et al. [2] gives a full description of this analysis tool. This paper presents the approach that has been developed for the creation of a low-cost pipeline network within the EU GeoCapacity analysis tool.

## 2. Principle of the algorithm

The goal of the algorithm is to create a low-cost network between a set of sources and sinks that have been selected for the economic evaluation in a CCS system. Sources and sinks can be connected either by direct pipelines, or they can be connected by using ‘existing’ pipelines. The grid is constructed step by step, while in each step an additional source-sink connection is created. In order to connect all sources and all sinks to the network, the entire operation is repeated in a loop that is being executed until all sources “run out” of *flow* (amount of emitted CO<sub>2</sub> per time unit – a source “runs out” of *flow* when all emitted CO<sub>2</sub> is transported) or all sinks are “filled up” (sinks also have a *flow* parameter, which in this case determines the maximum amount of CO<sub>2</sub> that can be injected per unit of time). Not all sources and sinks necessarily have to be available in the same year; the algorithm is capable of handling different starting years.

Next sections provide more detailed description of the algorithm itself, as well as short descriptions of parameters and formulas the algorithm uses.

## 3. The stepwise approach of the algorithm

The algorithm ‘creates’ the pipeline network stepwise. The first step is the determination of the cheapest connection from all possible source and sink connections.. Once pipelines ‘exist’, it is also possible to connect sources and sinks indirectly, i.e. by connecting sources and sinks using already existing pipelines and by creating junctions in the pipelines. The algorithm is being executed in a loop – in each step of the loop all direct as well as all indirect connections are examined. After a single loop step, the cheapest connection is chosen (which can either be a direct or indirect connection) and ‘added’ to the network. Figure II shows the steps of the loop – each image shows the network after applying the cheapest connection found in a single step.

In case a new connection uses (part of) an existing pipeline route, the size of the pipeline is adjusted accordingly to the new flow of CO<sub>2</sub>. The costs – important as it forms the basis of the selection – are the *marginal* costs to increase the pipeline to the required size, as we assume in this modelling exercise that the pipeline network is built at once. In practice – however, it might occur that some capacity is not used from the beginning, but later on when more sources or sinks are coming on stream. In some cases it might even occur that flows may be in opposite directions when additional sources are added to the grid. In that case the marginal costs are negative because the resulting pipeline can be of smaller size.

## 4. Costs of the pipelines

Because the algorithm bases its selection for pipelines on costs, it is important to show the formulas used for calculating the costs.

In case of direct connections, the price of the pipeline is determined by the distance between the connection source and sink, and the maximum amount of transported CO<sub>2</sub> in a time unit (the *flow*, expressed in kg/s). Distance determines the *length* of the pipeline. The diameter of the pipe is calculated by:

$$\text{diameter} = \sqrt{\left( \frac{\text{flow}}{1.5 * \pi * \frac{1}{4} * \text{density}} \right)}$$

Where: *density* is the density of the transported CO<sub>2</sub>.

With the length and diameter of the pipeline known, the investment of a pipeline is calculated by:

$$\text{investment} = (C_1 * \text{length} + C_2 + (C_3 * \text{length} - C_4) * \text{diameter} + (C_5 * \text{length} - C_6) * \text{diameter}^2)$$

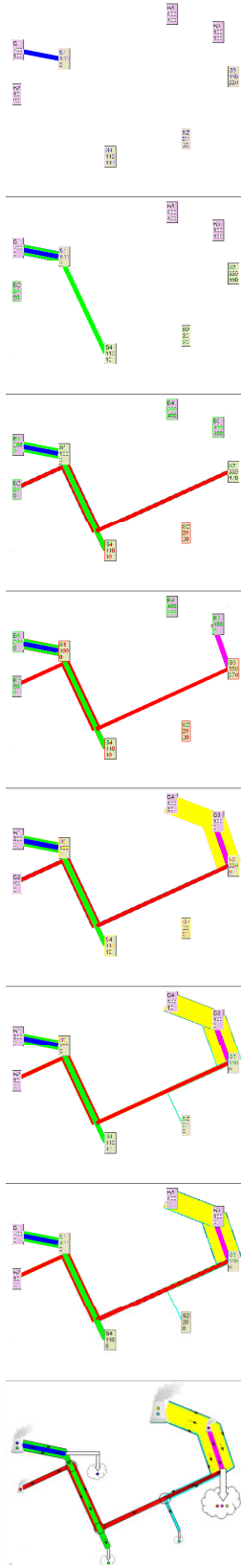


Figure II Hypothetical steps of a loop that executes the network generation algorithm. In each step (every image) both parts of the algorithm, the direct and the indirect, are executed, which results in a complete network after the last step.

The total transport costs (expressed in euro/tonne of CO<sub>2</sub>) are calculated as follows:

$$costs = \frac{\left( \frac{(inv\_pipes + inv\_boosters) * discount}{(1 - (1 + discount)^{-lifetime})} + OM\_pipe * inv\_pipes + OM\_booster * Inv\_booster + Power\_costs \right)}{flow * LoadFactor}$$

Where:

- Inv\_pipes: investment of the pipelines forming the grid
- Inv\_booster: investment of booster station for recompression CO<sub>2</sub> during transport
- OM\_pipe: operation and maintenance of pipeline system (3% of investment)
- OM\_booster: operation and maintenance costs of booster stations, excl power (5% of investment)
- LoadFactor: percentage of time pipeline network is utilized full capacity (based on full load hours)

### 5. The structure of the algorithm

The determination of the position and size of the first pipeline is straightforward – from all sources and sinks under consideration the cheapest connection is selected. This cheapest (direct) connection is found by an iteration process where the first loop iterates over sources and for every source there is another loop that iterates over every sink. In this way the algorithm finds the cheapest connection from all possible combinations. Figure IV shows this process of iteration.

Adding additional pipelines to the system is more complex. This is dealt with a nested loop, which looks for connections between all the sources and sinks, but this time also connections are evaluated that use already ‘existing’ pipelines. For every source-sink pair, the algorithm looks for a *path* in the existing network that may be used to create a new connection. Because of the nature of the problem, it is being solved by a recursive algorithm. First, the shortest possible connection is determined between a source to one of the existing pipelines. This connection is made for the purpose to determine the *path* to a sink. Then the algorithm goes recursively through all the ‘existing’ pipelines. In the next step the algorithm checks the connections from every pipeline and nodal point it goes through. In every step of the algorithm the cost of the already generated *path* is being calculated, so if there are several possible paths that the algorithm finds, only the cheapest one of those is taken into account.

In order to find the cheapest path, the examined source is being “connected” to the existing network. This is also being done in a loop, which iterates through the pipelines (so the *path* finding is being done as many times as there are pipelines in the network).. Figure IVb shows the parts of the algorithm where both direct connections and indirect connections are checked.

After these steps the cost of the generated path are compared to the cost of the cheapest direct connection that has been found in the former part of the algorithm. The cheapest connection is selected and this pipeline or pipeline route is added to the pipeline grid.

Before the algorithm for a given source (*Source*) and a given destination sink (*DestSink*) can be run, a few things have to be checked. For each pipeline *p*:

- it is checked whether *p* is already connected to the *Source* (either directly or through **one** other pipe); if not, then:
- it is checked whether it is possible to connect to *p* by creating a junction in it (the pipe that would connect the *Source* with *p* will be perpendicular to *p*); if that is possible, then create the junction and the connecting pipe; if not:
- connect to *p* using its starting nodal point.

After the above steps are made, the starting parameters for the recursive algorithm are determined. The parameters are a starting pipe and a starting nodal point.

## 6. Recursive algorithm of searching a path to the destination Sink

At the beginning of a recursive function there is a starting *pipe*, and a nodal point *cameFrom*, from which the algorithm ‘goes into’ the *pipe*. The other end of *pipe* is called *goesTo*. The following is done:

- it is checked whether *goesTo* refers to *DestSink*; if it does, then a new path has been found. The costs of the new path are compared to the costs of the cheapest path already found; if the new one is cheaper, then store it in *LowestCostPath* (an object that stores information about the cheapest path) and return.
- if *DestSink* has not been reached yet, then:
  - it is checked whether it is possible to create a direct connection from *pipe* to *DestSink* by creating a junction in *pipe* (the additional pipe would be perpendicular to *pipe*); if it is possible, then check the cost of this option (and compare the cost with the lowest cost so far. If it’s lower, then store the new path in *LowestCostPath*);
  - Then, all possible connections through other pipes connected to *goesTo* are checked – for each pipe *p* (excluding *pipe*) the recursive function calls itself with *goesTo* and *p* as parameters;
  - After checking all pipes connected to *goesTo*, we check a direct connection from *goesTo* to *DestSink*. After this step the recursive function ends.

After finishing the recursive algorithm, the *LowestCostPath* stores a description of modifications which have to be made to the network if we want to actually create the cheapest connection.

The whole procedure is repeated for all possible source-sink pairs – for each of them the cheapest connection is being looked for (the *LowestCostPath* information are being generated) and then the cheapest is selected.

It is possible that while looking for a connection through the existing network, that some part of a generated *path* will have an opposite *flow* direction than the underlying pipe. In such case, the pipe is made smaller, what results in actually directing a portion of the “new” *flow* to a route of the “old” *flow*. At the end of the “reversed” part of the new *path*, the corresponding part of the “old” *flow* is being transported further along the new *path*, just as it would be the part of the “new” *flow*.

## 7. Recursive algorithm for altering pipe flows

The GeoCapacity analysis tool considers also uncertainty ranges in the input data, both related to the production of CO<sub>2</sub> from sources as well as sink capacities. This implies that there is also a range in the amount of CO<sub>2</sub> transported over the pipelines. This may alter in principle the network lay-out and the flow though each pipeline section. Therefore, a network modification algorithm has been added. This algorithm is also recursive of nature. However, contrary to the network creation algorithm, it does not modify connections between sinks and sources (the lay-out of the pipeline grid), but it only modifies pipes’ thicknesses.

Before the algorithm can start, some information has to be gathered. While the network is generated, it is possible that some parts of it are not connected to each other (thus creating sub-networks). For each source that has not already been marked as a part of any sub-network, all elements that are connected to it are checked recursively and marked as connected to this particular sub-network.

The network modification is done for each sub-network separately. The algorithm begins from a source and checks all pipes connected to it. For each pipe *p* connected to the source:

- Check if the other ending (let’s name it *goesTo*) of the pipe is a sink; if it is, then store as much flow as possible at that location;
- Recursively go into all pipes connected to *goesTo* (excluding *p*) and do the same;

- When finishing the function, update the pipe's flow (thickness) taking into account the values of flow stored in all sinks found in all subsequent calls to the recursive function;
- Return a list of sinks to which the flow has been stored (and the actual amount of the stored flow in each of the sinks).

This procedure is repeated for all sinks in the network.

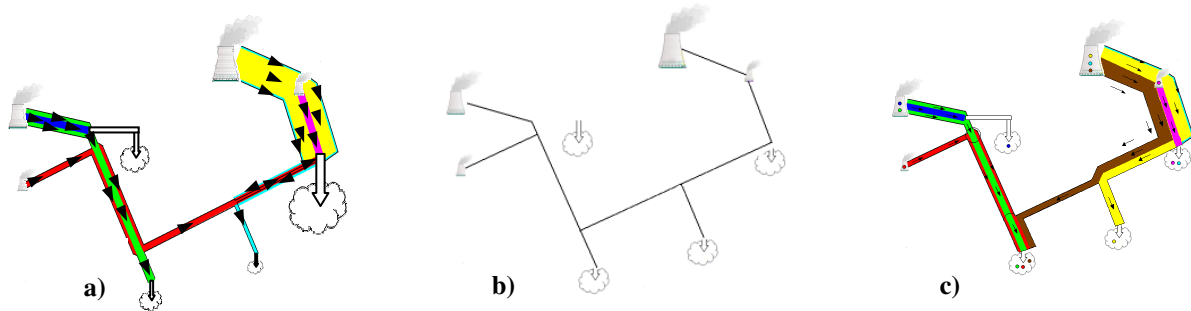


Figure III The network modification algorithm: a) the initial network, b) the network before it is modified, the pipe thickness is set to 0, and new capacities for the sinks are generated, c) the network is modified – the CO<sub>2</sub> produced in some sources is now being transported to other sinks than previously

## 8. Conclusion

In this paper we showed that a relatively simple and effective algorithm can be used to generate a low-cost pipeline network, connecting CO<sub>2</sub> sources (such as factories and power stations) with CO<sub>2</sub> sinks (for example unused oil or gas fields). The algorithm takes into account economical data of pipeline transport that are used to calculate costs of different possible connections within the network.

The algorithm is capable of calculating in an effective way a low-cost lay-out of the pipeline grid. It takes into account both direct and indirect connections between CO<sub>2</sub> sources and sinks. It also provides a methodology to alter the thicknesses of different parts of the pipeline based on uncertainty in CO<sub>2</sub> supply and demand data.

## 9. References

1. T. Vangkilde-Pedersen et al., Assessing European capacity for geological storage of carbon dioxide – the EU GeoCapacity project, this volume, 2008.
2. F. Neele, C. Hendriks, R. Brandsma, Geocapacity: economic feasibility of CCS in networked systems, this volume, 2008

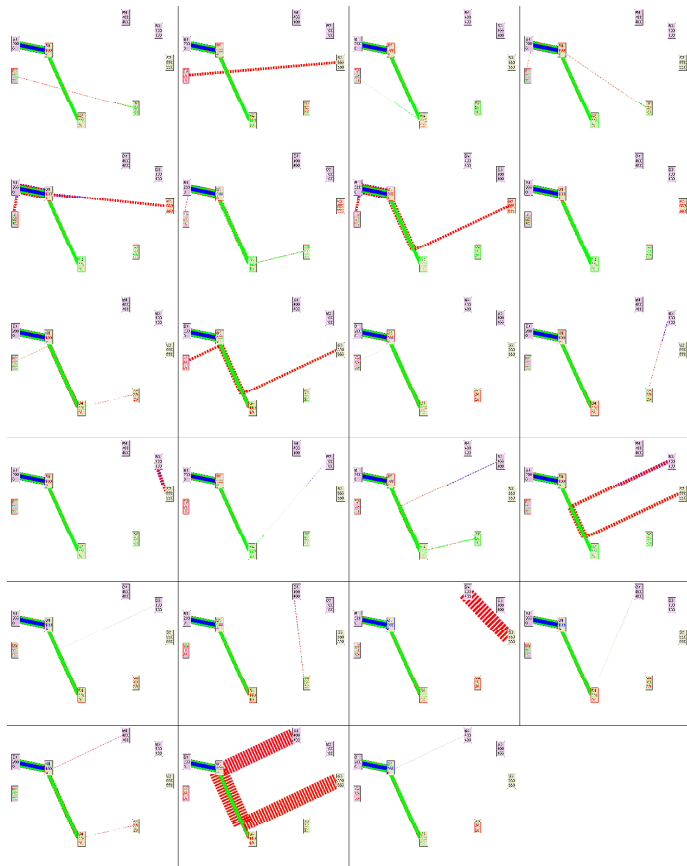
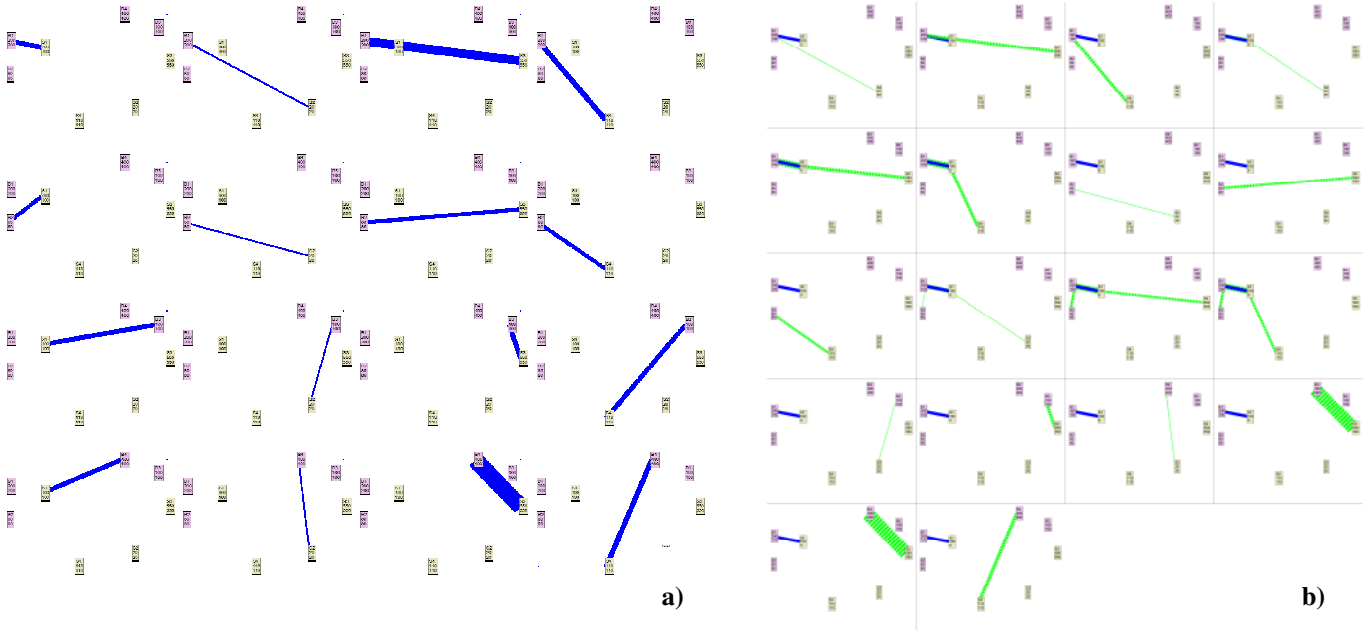


Figure IV a) all the combinations between 4 randomly positioned sources (purple blocks) and sinks (gray blocks); b) second step of the network creation algorithm – in order to find the cheapest connection between a source-sink pair, both direct and indirect connections are checked; c) Third step of the algorithm.

c)