Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational Physics 211 (2006) 700–718

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

# Robust algorithm for random resistor networks using hierarchical domain structure

Henning Arendt Knudsen [a,b,*], Sándor Fazekas [c]

[a] Department of Physics, Theoretical Physics, University Duisburg-Essen, D-47048 Duisburg, Germany
[b] Department of Physics, University of Oslo, PB 1048 Blindern, NO-0316 Oslo, Norway
[c] Department of Theoretical Physics and Theoretical Solid State Research Group of the Hungarian Academy of Sciences, Budapest University of Technology and Economics, H-1111 Budapest, Hungary

Received 24 August 2004; received in revised form 31 May 2005; accepted 6 June 2005
Available online 26 July 2005

**Abstract**

In this study we discuss methods for solving random resistor networks and similar problems. We discuss the node elimination method and we demonstrate its equivalence to the Gaussian elimination scheme, finding a good elimination order, which makes the method highly efficient. The transfer matrix method is shown to be a special case of the node elimination method with an ordering that is far from optimal. We compare the performance of these exact methods with a state of the art conjugate gradient solver. In general, the node elimination method is the faster method.
© 2005 Elsevier Inc. All rights reserved.

*PACS:* 02.60.Dc; 84.30.−r; 47.55.Mh; 46.50.+a

*Keywords:* Random networks; Numerical linear algebra; Transport theory

## 1. Introduction

A prime example of the study of disordered media is found in the random resistor network. Although formalism, namely Ohm's law and Kirchhoff's transport laws, were known long before, one can safely say that the use of computers started a new era in this field. Computer memory, being a scarce commodity at the time, led to the excellent but limited algorithm presented in [1]. The only studied quantity was the equivalent or global conductivity of the random resistor network, and therefore the network could be generated

* Corresponding author. Tel.: +47 228 56497.
*E-mail address:* arendt@phys.ntnu.no (H.A. Knudsen).

0021-9991/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2005.06.007

and evaluated at the same time without having to store much information in memory. A further restriction was that the network needed to be at (or close to) the percolation threshold – if not, the memory would be insufficient.

Later studies considered the whole current distribution in various networks. The set of transport equations needed to be solved and iterative solvers were the fashion in the late eighties. These are various implementations of the conjugate gradient method [2], but other schemes are also mentioned [3]. An efficient competing algorithm appeared because the special substitution, known as $Y$–$\Delta$ or star–triangle transformation, was shown to be applicable to some particular two-dimensional networks [4]. The idea of this algorithm has not been applicable to other networks. However, the $Y$–$\Delta$ transformation and its generalization can be used in a different way. These transformations were indeed used at the percolation threshold [1,5], but both papers state that the method is "unusable" when not near the percolation threshold. In particular, it is claimed that the conjugate gradient would be more suitable in this case [5]. In this study we demonstrate that exact methods based on substitutions can indeed compete with iterative solvers. Which method performs best depends on the nature of the problem.

In physical terms these exact methods consist of doing physical substitutions, replacing physical elements with equivalent physical elements. These substitutions and the substitution schemes that were used correspond exactly to Gaussian elimination. This fact was to our knowledge not mentioned in [1,5] or similar work. In this study we discuss this equivalence to some extent. Methods and ideas that applied mathematicians have been studying in a purely theoretical fashion have direct physical interpretations when applied to physical networks. Although we use the Gaussian elimination scheme as a theoretical basis, we describe the implementation of the solver in terms of physics. This means that one can implement a close to optimal Gaussian elimination for the networks without reverting to matrix formalism.

The need for a robust exact solver of random resistor networks does not only stem from the need to study such networks in themselves. More complex networks can be studied with the same or only slightly modified methods, that is to say networks that contain other passive elements or voltage sources. Such networks appear as models for other transport problems, for example fluid flow in porous media [6]. Another example is the study of fracture using random fuse networks [7]. Both of these examples require solving the transport equations over and over again. It will be shown, or outlined, how and when these problems can be better attacked with the solver presented in this study than with iterative solvers.

The paper is organized as follows: Section 2 starts out with the efficient implementation of the node elimination method, using physics and without an explicit use of matrix formalism. Thereafter, the equivalence of this method with the well-known Gaussian elimination using matrix formalism is given. The section ends with a discussion of the transfer-matrix method and its relation to the other methods. In Section 3 we discuss a scheme or strategy for the node elimination picture. A performance comparison between transfer matrix, optimized node elimination and the conjugate gradient method is given for different geometries and different parameters in Section 4. Finally in Section 5, the flexible inclusion of boundary conditions in the node elimination picture is discussed. In particular, we point out how certain applications, due to the required boundary conditions, are far better off with elimination than with conjugate gradient.

## 2. The principles of the methods

The principles of the methods can be conceptually divided into two categories. Firstly, we consider how the random resistor network can be solved by a substitution approach. The basic formulae are summarized, and thereafter we show how to construct a forward and backward substitution scheme based on these. Secondly, we address the order of substitution in Section 3. In order to make the algorithm work efficiently, some further principles are needed. The choice of data structures for storage and loops is also essential.

## 2.1. The node elimination picture

In order not to complicate matters, we consider only normal resistors in the presented formalism, in other words no inductors and capacitors. How to handle voltage sources will be discussed later together with boundary conditions. A network of resistors can be characterized as a set of nodes with connections between them. Each node is given an index, say $i$, and it is assigned a voltage $V_i$. Each connection between nodes, say $i$ and $j$, has a resistance denoted by $R_{ij}$. In practice, it turns out that it is more convenient to work with conductances, $G_{ij} = 1/R_{ij}$, instead of resistances. The nodes are either internal nodes or external nodes, the latter being the nodes that are connected in some way to the surroundings, typically a voltage source or a current source. In general, each external node has a certain voltage and an external current flowing out of the node. It is required that one of the two quantities be specified as the boundary condition. The voltages of the internal nodes are the unknown in the problem. They are found by solving Kirchhoff's equations. Consider a node, say 0, being connected to $n$ neighbours, then Kirchhoff's current law reads

$$\sum_{i=1}^{n} G_i(V_i - V_0) = \begin{cases} 0, & \text{internal node,} \\ I_i, & \text{external node.} \end{cases} \tag{1}$$

If node 0 is an internal node, the solution with respect to the voltage $V_0$ is

$$V_0 = \frac{\sum_{i=1}^{n} G_i V_i}{\sum_{i=1}^{n} G_i}. \tag{2}$$

In other words, given the voltages in all the neighbouring nodes, this formula finds the voltage $V_0$. Now, we want to make a change to the network without changing its properties with respect to the boundary conditions. It is allowed to remove a node, say 0, from the system if all connections, already existing or not, between the neighbours of node 0 are updated. The additional conductivity between node $j$ and $k$ is

$$\Delta G_{jk} = \frac{G_{0j} G_{0k}}{\sum_{i=1}^{n} G_{0i}}, \tag{3}$$

which is a generalization of star–triangle substitution [8]. Typically some of these connections were already in existence, so that $\Delta G_{jk}$ is added to their conductivity. However, many connections are not there and need to be created. See Fig. 1 for an illustration of the removal process. The zeroth node is removed and $\Delta G$'s are added according to Eq. (3). Upon the removal of many nodes, the number of connections grows rapidly, see Fig. 2.

Let us consider how the solution of the flow equations is found. Consider that we want to apply Kirchhoff's laws to a network between a number of external nodes with predefined voltage (other boundary

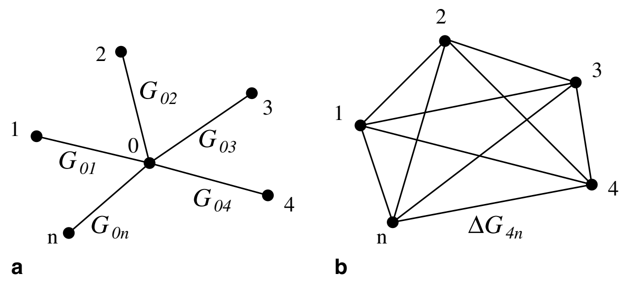

Fig. 1. (a) The state before removing node 0. (b) After removing node 0, its connections to its neighbours are removed, and new connections are added between the neighbours. Already existing connections between the neighbours are updated. The network is equivalent to (a), provided the additional conductivities be calculated from Eq. 3.

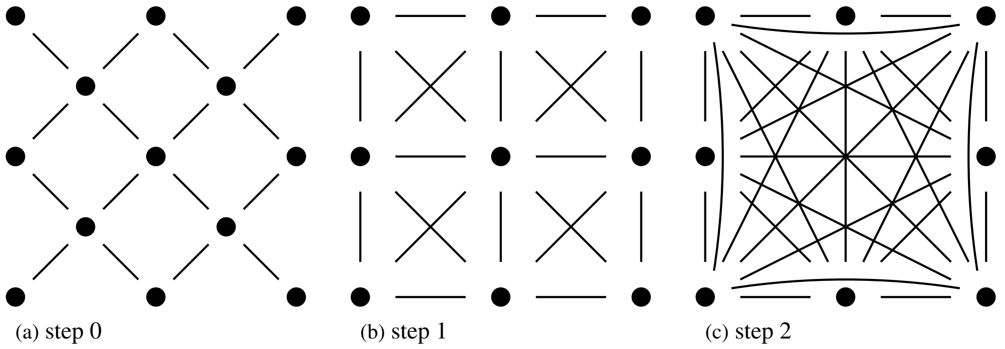

Fig. 2. The figure shows three networks that are equivalent in the following sense. The eight points on the outside of the networks are the same but the interior points and the connections between the points change. Consider (a) to be the initial network. When four nodes are removed, connections are made (or updated) between the neighbours, and the network looks like (b). Removal of the middle node in (b) creates (c). Note here that the process (a)–(b)–(c) is always allowed. The inverse process (c)–(b)–(a) is only allowed for special values of the conductivities (e.g., when generated by the process (a)–(b)–(c)) and therefore of little use.

conditions will be discussed in Section 5). In that case, each of the internal nodes are removed in sequence, always updating the network and its conductivities according to Eq. (3). This is the forward substitution part, which terminates only when the external nodes with given voltage are left. In the special case where the system has only two external nodes, there is only a single connection between them, which contains the global conductivity of the whole system. For some investigations this would be the desired information but more generally one would calculate backwards to find the solution for the voltage in each node. By means of Eq. (2), the voltage division rule, one inserts the nodes in the inverse order. The other voltages in the neighbouring nodes have already been calculated, so this is a straight-forward procedure.

In order to do this in practice one needs to index all nodes. Given an indexing of the nodes, the information which needs to be stored to do the back substitution, is for each node the indices of the neighbours and the respective conductivities at the time of removal. This implies that whatever other connections a node had to other nodes that were removed before the node, this information is not needed to restore the node. Therefore, it is not needed to store this information at any time in connection with the node. In fact, all information *is* stored, but distributed among the nodes so that each node only keeps track of the connections to nodes that are removed at a later time. Efficiency is increased in this way but it requires the order in which the nodes are removed to be pre-defined. Ordering is discussed further in Section 3.1.

At some time a part of the storage structure may look like Fig. 3. The neighbours of node 15 are listed in the first list. The numbers are the indices of the nodes, and the order in the list is the substitution order. If we assume that node 15 is the next node to be removed, then this list contains *all* the neighbours of 15 at that time. Were there other neighbours at an earlier time, then they do not need to be listed in the list of node 15, since they were removed before 15. Now, consider one of the neighbours of 15, for example node 35. The list of this node is also shown. Some observations can be made by comparing the two lists. Nodes 7

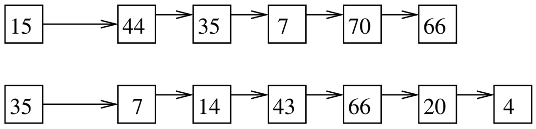

Fig. 3. The figure shows a sample of the storage structure. All numbers are node indices and thus these lists are the linked lists of neighbours belonging to node 15 and 35. The order of the elements in the lists is in accordance with the substitution order, in other words every element in the list corresponds to a node that will be removed before all the nodes that are listed later in the list.

and 66 are common neighbours. Node 70 is a neighbour of 15 but not yet a neighbour of 35. Nodes 14, 43, 20 and 4 are neighbours of 35 but not 15. Finally, node 44 is a neighbour of 15 but we do not know if it is a neighbour of 35 because this node is removed before 35, and hence it is not listed in 35's list (35 could appear in the list of 44 though).

The removal of node 15 means running a double loop over all possible pairs of neighbours and updating the conductances of respective connections in other lists. For instance, the connections between 35 and the nodes which are listed after 35 in the list of 15 are all updates that need to be made in 35's list. Algorithmically this means that the sublist 7-70-66 is merged with the list of 35. This is the key to making the method work. Instead of spending time searching in for example a tree structure for each of $N$ updates, requiring $\mathcal{O}(N\log(N))$ time, the merging of lists requires $\mathcal{O}(N)$ time. The pseudocode for the elimination is given in Appendix A.

## 2.2. The Gaussian elimination picture

We now turn to the matrix formulation of the problem. In the node elimination picture, the use of Kirchhoff's current law provides the necessary formulae for the forward and backward substitution, Eqs. (2) and (3). Kirchhoff's equations in Eq. (1) can be written in matrix form as

$$\mathbb{G}\vec{V} = \vec{I}, \tag{4}$$

where the voltage vector $\vec{V}$ contains unknown voltages and the current vector $\vec{I}$ contains external currents flowing out of the nodes. External means that they are not coming from any of the other nodes that are considered a part of the network. As a consequence, most of its entries are zero and only points which are externally connected, for example electrodes, have non-zero entries. The different possible boundary conditions will be discussed in more detail in Section 5. In Eq. (4) we also introduce the conductivity matrix

$$\mathbb{G} = \begin{pmatrix} -S_0 & \cdots G_{01} \cdots & G_{0k} & \cdots G_{0n} \\ \vdots & \vdots & \vdots & \vdots \\ G_{01} & \cdots - S_1 \cdots & G_{1k} & \cdots G_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ G_{0k} & \cdots G_{1k} \cdots & -S_k & \cdots G_{kn} \\ \vdots & \vdots & \vdots & \vdots \\ G_{0n} & \cdots G_{1n} \cdots & G_{kn} & \cdots - S_n \end{pmatrix}. \tag{5}$$

Here, the diagonal elements are $S_i = \sum_{j\neq i} G_{ij}$. By construction the conductivity matrix is symmetric, since the conductivity between two connected nodes is not dependent on the direction. The matrix is generally a sparse matrix, because each node is only connected to its local neighbours, which is only a small number of nodes, in comparison to the whole network.

To solve the equations in Eq. (4) by Gaussian elimination, one starts out by adding multiples of the zeroth row to each of the other rows, such that all entries in the first column below $S_0$ become zero. In other words, to each element $k$ of row $i$, the corresponding value in the zeroth row times $G_{i0}/S_0$ is added;

$$\Delta G_{ik} = G_{0k}\frac{G_{i0}}{S_0} = \frac{G_{0i}G_{0k}}{\sum_{j\neq i} G_{0j}}. \tag{6}$$

This elimination of the zeroth column in the Gaussian elimination scheme is identical to elimination of the first node in the node elimination picture. This follows from a direct comparison between Eqs. (3) and (6). Further, the remaining sub-matrix of $\mathbb{G}$, without the first row and column is still a symmetric matrix, whose rows and columns add up to zero. This corresponds exactly to the reduced physical network which appears in the node elimination picture after removing one node. In conclusion, the two methods are identical.

Two remarks regarding the data storage structure follows. Since the matrix is always symmetric, it is sufficient to store only half the matrix in memory, for instance the right upper part. The order of the equations in the matrix defines the order of elimination. Thus storing the right upper part, means storing in each row the conductivity to all nodes that are to be eliminated at a later stage. This was the case for the recommended structure in the node elimination picture. Secondly, the matrix is, at least initially, rather sparse. If there are few elements in a row it is preferable to store its elements in a linked list. This is the standard for sparse matrix implementations of Gaussian elimination. Again, the physically derived storage structure has a one-to-one correspondence to the optimal matrix representation for Gaussian elimination.

## 2.3. The transfer-matrix method

Before going into a more general discussion of finding the best order for the elimination process, we will present the so-called transfer-matrix method [9,10]. In the literature this method has been considered different from the node elimination method but as we will show, these two methods are equal. That is to say, the principle is equivalent, but the transfer-matrix method corresponds to one specific order of the elimination of the nodes.

Our presentation of the transfer-matrix method is in accordance with the work of Derrida et al. [9]. A square random resistor network between two conducting bars is considered, see Fig. 4(a). The points on the right-hand side, numbered from 1 to $N$, are boundary points, to which a given external current $I_j$ can flow. Each of these nodes has a potential $V_j$, which is dependent on the currents $I_j$. They are related through an $n \times n$ conductivity matrix $A_L$,

$$I_j = \sum_{k=1}^{N} (A_L)_{jk} U_k. \tag{7}$$

No matter the size of the network to the left of the boundary points, its effect can always be reduced to this type of a conductivity matrix. Probably the easiest explanation is that all the elements are linear, meaning that any of the currents $I_j$ must depend linearly on each of the voltages $U_k$. One can say that the entire network, on the left-hand side of the boundary points is expressed in terms of an equivalent network. Each pair of boundary points are connected through a resistor, whose value is the respective component of $A_L$.

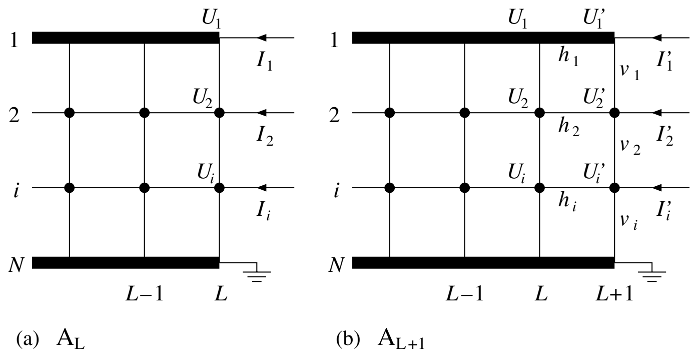

Fig. 4. In the transfer-matrix method the network is said to be built strip by strip, by adding resistor after resistor to the network. In (a) the system is built until strip *no. L*. At this stage the conductance matrix is $A_L$. In (b) one entire strip of nodes has been added. The inserted horizontal and vertical resistors have resistance $h_i$ and $v_i$ respectively. At this point the conductance matrix $A_{L+1}$ is related to the previous one, $A_L$. The illustrations, notation and sign conventions are in accordance with the work of Derrida et al. [9].

One starts out with a simple network, for which the conductivity matrix is known. Thereafter, the network is built strip by strip from left to right. Fig. 4(b) shows how the strip $L+1$ is added to the system. Horizontal resistors, $h_i$ are added to the respective rows and vertical resistor $v_i$ between the respective rows. This new system has a conductance matrix $A_{L+1}$, which can be calculated from the previous matrix $A_L$ [9],

$$A_{L+1} = V + A_L(1 + HA_L)^{-1}. \tag{8}$$

The matrix $V$ includes all the added vertical resistors. Its definition is

$$V_{ij} = [1/v_i + 1/v_{i-1}]\delta_{ij} - [1/v_i]\delta_{i+1,j} - [1/v_{i-1}]\delta_{i-1,j}. \tag{9}$$

The horizontal resistors are contained in the matrix $H$,

$$H_{ij} = h_{ij}\delta_{ij}. \tag{10}$$

Note that $V$ dimensionally contains conductances, whereas $H$ contains resistances. The technical reason for this can be seen in Fig. 4(b). The vertical resistances $v_0$ and $v_{N+1}$, above and below the system, are of course not added physically. However, matrix $V$ holds these values as well. A non-present bond is better stored as zero conductance than as infinite resistance. In contrast, the two horizontal connections $h_1$ and $h_N$ are better stored as zero resistance in $H$ than as infinite conductance.

Now the question arises, how does this method stand in relationship to the node elimination picture and Gaussian elimination? After the strip $L+1$ is added, the system contains $2N$ nodes. In other words there are $2N$ Kirchhoff's equations that we write in matrix form as

$$\begin{pmatrix} A_L + H^{-1} & -H^{-1} \\ -H^{-1} & V + H^{-1} \end{pmatrix} \begin{pmatrix} U \\ U' \end{pmatrix} = \begin{pmatrix} 0 \\ I' \end{pmatrix}. \tag{11}$$

This requires some explanation. The nodes are sorted in the following order: the nodes in column $L$ are the first 1 to $N$ entries, and the nodes in column $L+1$ are the entries $N+1$ to $2N$. Thus, the voltage vector consists of the elements of $U$ followed by $U'$. The current vector's first $N$ entries are equal to zero; this is because all points in column $L$ are now the internal points of the network, or in other words no external current flows into any of these nodes. For convenience we consider the values of $h_1$ and $h_N$ to be non-zero so that node 1 is distinct from node $N+1$ and node $N$ is distinct from node $2N$. Further, it allows us to construct the inverse of the matrix $H$, which is needed to represent to conductance values of the horizontal bonds, without having to consider the upper and lower bar separately.

The conductance matrix in Eq. (11) consists of four $N$ by $N$ blocks. The upper right and lower left block contain inter-connections between column $L$ and $L+1$, namely the diagonal matrix $H^{-1}$. The lower right block contains mainly vertical inter-connections within column $L+1$, the matrix $V$. In addition, the diagonal elements in this block must be updated to meet the requirement that the sum of all entries within a single row or column must be zero. Thus, this block becomes $V + H^{-1}$. For the upper left block, the diagonal elements of the previous conductance matrix $A_L$ is updated similarly.

We want to solve this system of equations by Gaussian elimination. That is to say that the internal nodes, or rather the first $N$ rows of the matrix, are eliminated. First, we eliminate all entries in the lower left block. Upon inspection one sees that by multiplying the upper blocks from the left with $H^{-1}(A_L + H^{-1})^{-1}$ and adding these to the lower blocks, this is accomplished. At the same time the lower right block is updated to

$$V + H^{-1} - H^{-1}(A_L + H^{-1})^{-1}H^{-1}. \tag{12}$$

By using the equality

$$(A_L + H^{-1})^{-1} = (H^{-1}[1 + HA_L])^{-1} = (1 + HA_L)^{-1}H, \tag{13}$$

the block becomes

$$V + H^{-1} - H^{-1}(1 + HA_L)^{-1}. \tag{14}$$

Some further manipulations are

$$\begin{aligned} H^{-1} - H^{-1}(1 + HA_L)^{-1} &= H^{-1}(1 - [1 + HA_L]^{-1}) = H^{-1}([1 + HA_L] - 1)(1 + HA_L)^{-1} \\ &= H^{-1}HA_L(1 + HA_L)^{-1}, \end{aligned} \tag{15}$$

giving the final result

$$V + A_L(1 + HA_L)^{-1} = A_{L+1}. \tag{16}$$

To actually arrive at the voltage in column $L$, one would have to do more work on the upper left block of the matrix. However, if one is only interested in the relationship between $U'$ and $I'$, this information is already contained in the lower right block, which can be correctly named $A_{L+1}$. One sees that the result from [9], quoted in Eq. (8), is exactly the same result as the one obtained by Gaussian elimination. Hence, the methods are equivalent.

The transfer matrix method can also be expressed in a slightly modified form [10]. The strips can be added node by node instead of being added as one block. In actuality, it is the same as adding one node and then removing another (the node that became an internal node). It has been argued that the transfer matrix method is faster and easier when implemented in this way. This makes sense, numerically the Gaussian elimination is also easier when performed row by row instead of by block. In conclusion, the transfer matrix method is basically equal to node elimination, as long as the nodes are removed in sequence, column after column. This ordering is not bad, in particular for long and narrow strips. However, the ordering is not optimal; this is further discussed in the following section.

## 3. Elimination order

So far we have shown that the node elimination method and the transfer matrix method are equivalent to Gaussian elimination. The transfer matrix method is special in that it corresponds to a particular ordering of the elimination. Node elimination can be accomplished in any order and the natural question is, what is the best order?

Any physical network, in which the nodes are connected to their neighbours in space, gives rise to very sparse conductance matrices. The reason is that initially the number of neighbours is much smaller than the total number of nodes. The sparseness is exploited in the implementation. Only nonzero elements in the linked list of each node need to be processed. To keep the processing time to a minimum, it is necessary to keep the number of nonzero elements as low as possible upon elimination. In other words the so-called fill-ins in the matrix must be kept to a minimum. The fill-ins in the Gaussian elimination picture correspond to the newly added bonds in the node elimination picture when removing a node.

In principal, one can determine the optimal order of such an elimination process. The drawback is that such an algorithm is NP-complete and therefore not very useful when the goal is to make fast solvers. Heuristic methods which are based on graph theory exist and pretty good orders can be found with rather simple algorithms [11,12]. The random resistor network *is* itself the graph corresponding to the problem, so it need not be constructed from the matrix.

There are two bottom-up strategies worth mentioning. First, the so-called minimum degree ordering scheme. At every point in the elimination process, the next node to be eliminated is the node that has the smallest number of connections. It is likely that removal of this node will give a number of new connections (fill-ins) that is close to the possible minimum. The second possibility is to check for the number of new connections and remove the node which creates the lowest number of such connections, the so-called minimum local fill scheme. Extension of both these ideas is possible. Instead of considering the effect of removing a single node, one determines the effect of removing a group of nodes. This is for example a good strategy if the system contains some kind of clique of nodes, or in other words a group of nodes that is strongly interconnected but which in total has very few connections to other parts of the network.

Top down strategies are also possible. The idea is to take a physical network (or more generally the generated graph) and subdivide the system. For instance one can first divide the system into two equally sized regions. Care must be taken to define sets of nodes that make good boundaries between the regions. The requirement, which must be met, is that every node inside a given region and not belonging to the boundary is only connected to other internal nodes of this region or to the boundary of the region. After such a division is found, the procedure can be repeated for each of the subregions, and so on recursively. The node elimination order is found by starting at the lowest innermost level. The interior points of the smallest regions are removed first. Thereafter, the remaining interior points of all the second smallest regions are removed, and so on until all nodes are removed. For a random network some work is required to implement such a nested dissection. However, when dealing with regular networks, good subdivisions and orderings can be designed as the following special case illustrates.

## 3.1. Domain structure

Here, we consider a special case where the ordering of node elimination can be made optimal or very close to optimal by way of direct construction. The system is a two-dimensional square tilted network. A small section of the initial network would look like Fig. 2(a). Now, the first nodes to be eliminated are every second row and column, as illustrated in Fig. 2(b). A larger portion of the network is shown in Fig. 5(a). Here, only the boundaries are drawn. It is understood that every point on the boundary of each domain is connected to every other point on the same boundary.

Larger domains are made, each from four smaller domains. When the centre nodes of these domains are removed the remaining boundaries look like Fig. 5(b). Again, new domains are formed by joining four smaller domains. The interior points are removed and the result is the domains shown in Fig. 5(c). In the transition from Fig. 5(b) to (c), there are five internal points to be removed within each domain. By applying the principle of keeping the number of connections as low as possible as long as possible, one sees that there is an optimal order of removing these nodes. First, one removes the points (in this case only one) that lay between the two upper subdomains. Then the points between the two lower subdomains are

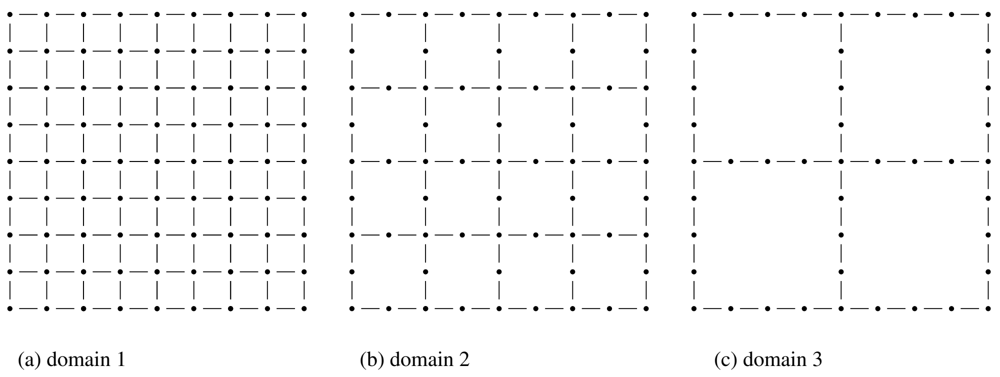

Fig. 5. In a regular network, the elimination scheme is found by means of a hierarchical domain structure. By removing the inner nodes of the domains on the lowest scale first and thereafter the inner nodes of the next scale and so on, the elimination process becomes optimal or close to optimal.

removed, and finally the horizontal line of remaining points. This procedure is easily generalized for larger and larger domains. Finally the outer boundary is removed and the elimination process is finished.

This order of the elimination works well. How well it works still needs to be demonstrated and this is the topic of the following section. The transfer matrix order and our suggested order will be tested against one another and against a conjugate gradient iterative solver.

## 4. Comparison of the methods

### 4.1. The test setup

In order to compare the methods discussed in this study, we have chosen a rather simple and well-defined system setup. A two-dimensional square tilted network of resistors between two conducting bars is used, see Fig. 6(a). In this case the ordering scheme for the node elimination method is exactly as illustrated in Figs. 2 and 5, see the discussion in the related main text.

For convenience only, many of the chosen system sizes are powers of two. It makes the subdivision of the system into domains simpler in terms of programming. Moreover, the repeated doubling of the system size automatically creates data points which are evenly distributed on a logarithmic scale. In the cases where one or more factors differ from two, the subdivision is still made according to the coarsening procedure, starting as illustrated in Fig. 2. When approaching the system size, one reaches a point where one can no longer simply join four and four domains, but one has to choose some other order. As long as the remaining domain numbers are small in at least one dimension, for example $3 \times 3$ or $1 \times 16$, we believe the ordering is still quite close to optimal. As mentioned earlier, actually finding the optimal ordering is a very complex problem. For practical purposes it is more important to test the method with a good heuristic ordering than the optimal, since heuristic orderings will be used in practice.

When comparing with the transfer matrix method, we have chosen to revert to the classic straight network setup, see Fig. 6(b). We make sure that the work needed to be done to solve the equations corresponds to the setup in Fig. 6(a). The number of nodes, i.e., equations, is the same. The distance between the bars is the same, meaning that the number of nodes on a line or the number of resistors on a line (tilted in (a) and straight in (b)) is also the same.

The conductance values are drawn at random. For the two exact methods, the choice of distribution of these values is unimportant. However, the conjugate gradient method is quite sensitive to distribution. In all

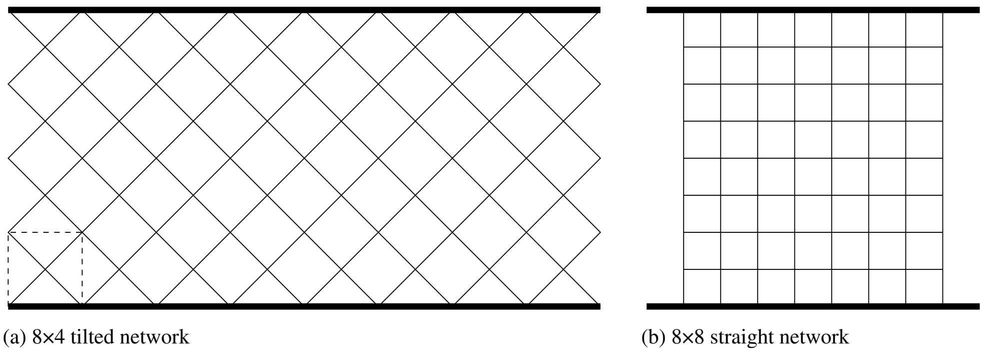

Fig. 6. The main test setup is the square tilted system shown in (a). The basic unit of the system is taken to be the face centred square unit. It follows that the horizontal times vertical length of (a) is $Lx \times Ly = 8 \times 4$. The transfer matrix method was first made for straight networks, as shown in (b). In order to have the same number of nodes and the same number of connections between the upper and lower bar, the corresponding straight system to (a) is the $8 \times 8$ system in (b).

tests we let 50% of the conductances have unit value, i.e., $G = 1$. The other 50% is given a different value, for example $G = 10$. To test the conjugate gradient method we use this second value of the conductances as a parameter.

The two bars are given fixed voltage values. The lower bar is given zero voltage. The upper bar is given a voltage proportional to the vertical length of the system. The idea is that independent of system, the average voltage drop per row between the bars is unity. If one changes this convention it will slightly affect the convergence time of the conjugate gradient. The idea is to be consistent.

All the test runs were done on a PC with an AMD Athlon(TM) 1.8 GHz CPU. The memory was 1 GB, which was large enough for all programs to fit into it. Ten runnings were executed for each test, each with a different random seed and the average execution times were calculated.

## *4.2. Square system*

The required CPU time to solve the equations for a square network is investigated here. The system size is varied from $Lx \times Ly = 16 \times 16$ to $Lx \times Ly = 512 \times 512$. Further, the distribution of the conductances is varied. Three values for the second conductance are tested: 10, 1000 and 100,000. The difference between the first value (being 1) and the second greatly influences the convergence time of conjugate gradient. Therefore, how good or bad the conjugate gradient is may be heavily dependent on the problem at hand and its parameters. In addition, a convergence criterion needs to be given. We have chosen two values for this so-called $\epsilon$, namely $\epsilon_a = 10^{-12}$ and $\epsilon_b = 10^{-17}$. This number is the upper limit for the allowed square of error per equation.

The results for the square network are presented in Fig. 7. First of all we note that node elimination seems to be the better method for the entire system range investigated. However, the conjugate gradient method is not much slower for the mixture of conductances $G = 1$ and $G = 10$. Clearly, the narrower the conductance distribution becomes, the faster the conjugate gradient converges. Hence, there might very well be applications with sufficiently nice coefficients (conductances), where the conjugate gradient is preferable. On the other hand, there are also distributions where conjugate gradient is much slower than node elimination.

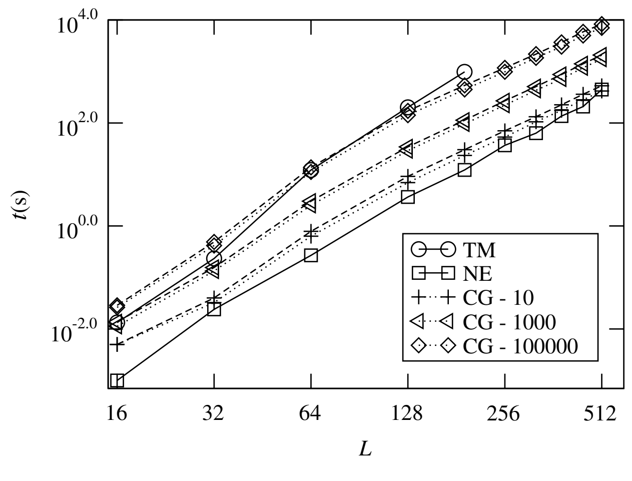

Fig. 7. The required CPU time for solving a square system $Lx = Ly = L$ is shown. The legend means: TM = transfer matrix, NE = node elimination and CG = conjugate gradient. In the case of conjugate gradient, two different convergence criteria were used: dotted curves have $\epsilon = 10^{-12}$ and dashed curves have $\epsilon = 10^{-17}$. The numbers after CG refer to the second conductance value of the connections, see main text.

Regarding the effect of the convergence criterion $\epsilon$. This criterion greatly influences the accuracy with which the equations are solved. It does not have a very strong impact on the CPU time needed to reach the desired accuracy. At least in regards to these systems, one could say that the convergence tends to be slow at the beginning of the iteration and faster towards the end.

There is a tendency that can be seen in Fig. 7. When increasing system size, the node elimination's advantage over conjugate gradient decreases. This is actually good news, because for larger systems there is indeed also a memory limit that might come into play. The necessary memory for a 512 square system is already in the order of half a Giga-byte using node elimination. Conjugate gradient is very memory efficient and can be employed for solving much larger systems.

Completing the discussion, we also include the result for the transfer matrix ordering of these square systems. As expected this ordering is rather bad in this geometry. However, in the following subsection we will see that it is better for very narrow but long strips.

### 4.3. Horizontal and vertical strips

Transfer matrix ordering has been considered very efficient when working with long and narrow strips. This is true, but it is also true that the node elimination scheme based on hierarchical domain structure is very good for strips. We test how good the methods are by direct comparison on horizontal strips. Four different widths have been selected for the tests: $Ly = 16$, 32, 48 and 64. The length of the strips is varied up to roughly $Lx = 10{,}000$.

We also compare with conjugate gradient. The distribution of conductances is taken to be 50% $G = 1$ and 50% $G = 10$. This is one of the distributions that was used in the square system case and it is the one which led to better results for conjugate gradient. We think it presents a fair comparison between the methods.

The over-all result, including all system widths, is that the node elimination scheme is the better method. The shortest systems are so short that they are basically square. Here conjugate gradient is slower in the same way as was shown before in Fig. 7. As the strips become longer, the difference between the methods increases and then seems to saturate.

The most surprising result is that transfer matrix ordering is only able to compete with the other methods when working with very narrow strips. At the width $Ly = 16$, Fig. 8(a), it is roughly as fast as the conjugate gradient method but significantly slower than the node elimination scheme. For wider and wider strips, Fig. 8(b)–(d), transfer matrix ordering becomes less useful.

The fact that the conjugate gradient method seems to saturate at a constant factor slower than node elimination with long strips can probably be understood in terms of the boundary conditions. When the boundaries, the bars, are close to each other in this way the different sections of the long strip do not physically interact much. The solution in one section is rather independent of the solution in a section far away. Thus, the need for information exchange over large distances along the strip in the iterative process is not so great. This can explain why conjugate gradient still works well. In order to check this dependence on boundary conditions, we have made a similar test with vertical strips.

Two widths have been chosen: $Lx = 32$, 64. Basically the systems are of the same size and shape as the horizontal systems but the boundaries are further and further apart. This does not really alter the results for node elimination as can be seen in Fig. 9. The results however for conjugate gradient are very different from the horizontal setup. Using node elimination as a reference, one sees that with the increasing height of the system conjugate gradient becomes slower and slower. No sign of saturation is evident.

Knowing that conjugate gradient is very sensitive to boundary conditions, they become a factor that should indeed be taken into consideration when choosing between these methods. If the boundaries are far apart, the node elimination scheme is clearly better.

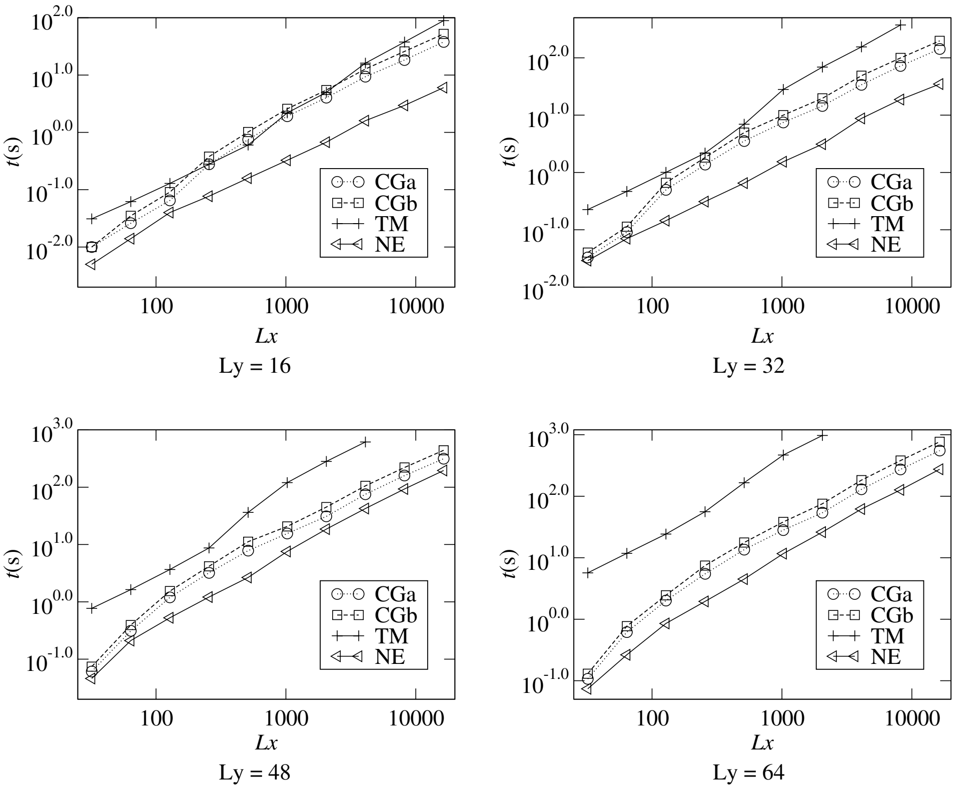

Fig. 8. Horizontal strips of four different widths $Ly$. The legend means TM for transfer matrix ordering, NE for node elimination, CGa for conjugate gradient with $\epsilon_a$, and CGb for conjugate gradient with $\epsilon_b$.

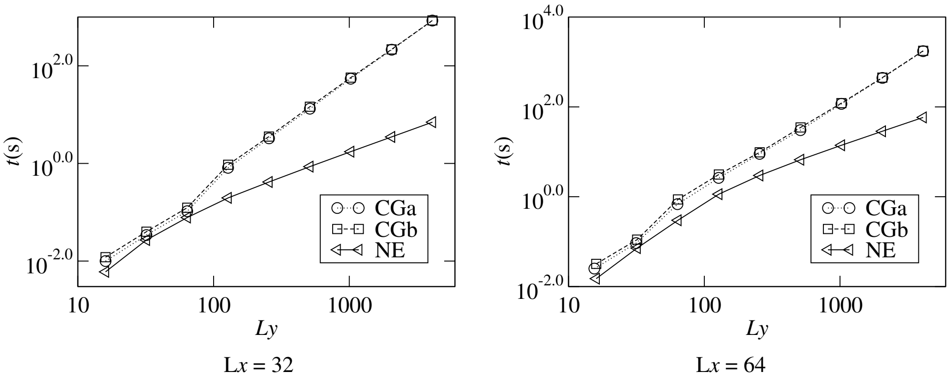

Fig. 9. Vertical strips of two different widths. The legend is the same as in Fig. 8.

## 5. Robust handling of boundary conditions

One of the aspects that should be taken into account when choosing between iterative solvers and exact solvers is the handling of boundary conditions. In order to address this question properly, we first indicate the general framework for solving such a system with boundary conditions. It is of some importance to clarify where the parameters of the problem enter, before discussing how this can be exploited in certain applications.

Before the elimination, the system of equations can be written in matrix form as

$$\begin{pmatrix} G_a & G_b \\ G_c & G_d \end{pmatrix}\begin{pmatrix} V_{\text{unknown}} \\ V_{\text{known}} \end{pmatrix} = \begin{pmatrix} I_{\text{known}} \\ I_{\text{unknown}} \end{pmatrix}, \tag{17}$$

where the $G$'s are sub-matrices and the $V$'s and $I$'s are vectors. All the nodes which have a predefined known voltage are sorted at the end, that is to say the lower part. This lower set of equations contains unknown external currents, which can be calculated in a straight-forward manner once all the voltages have been calculated. Therefore, in practice, manipulations to the lower part of the system need not be performed. For illustration we assume that these rows also were manipulated in the elimination process, giving the following result:

$$\begin{pmatrix} G'_a & G'_b \\ 0 & G'_d \end{pmatrix}\begin{pmatrix} V_{\text{unknown}} \\ V_{\text{known}} \end{pmatrix} = \begin{pmatrix} I'_{\text{known}} \\ I'_{\text{unknown}} \end{pmatrix}. \tag{18}$$

Here, the upper left block $G'_a$ is upper right triangular and the system is prepared for the back substitution. In the simplest case, all the equations in the upper block belong to internal nodes. No external current is applied to any of the nodes and the right-hand side becomes $I_{\text{known}} = 0$. It follows that the updated right-hand side $I'_{\text{known}}$ is also zero. Since the resulting right-hand side is already known, no calculation time is needed for its update.

In the more general case, where there are nodes with known external currents, their values enter into $I_{\text{known}}$. The order of the equations may be so that the first section of the $I_{\text{known}}$ vector contains only zeros. In that case the elimination can be done until the first nonzero element appears in $I_{\text{known}}$ before one has to worry about updating this vector, or in other words the right-hand side of the equations. However, generally the vector is updated to $I'_{\text{known}}$. This update can be done at the same time as the manipulation of the left-hand side of the matrix equation, which would be the normal procedure. Alternatively, it is possible to finish the elimination of the left-hand side first and then calculate the updates of the right-hand side. For a single solution of the system of equations it does not make much sense but for some applications it is useful. The recalculation follows exactly the same scheme as the normal calculation, using the values of the already calculated left-hand side. See Appendix A for the pseudocode of the whole elimination and for the recalculation only.

Either way, the calculation of $I'_{\text{known}}$ is not a difficult point. An interesting remark that should be made at this point concerns the possible presence of voltage sources within the network. This is illustrated in Fig. 10(a). In a connection between two nodes with conductance $G$, there is a voltage source with a voltage drop $E$. Writing down the Kirchhoff's current law for each of the nodes, one sees that the terms $EG$ and $-EG$, respectively, enter on the right-hand side of the equations. An equivalent network is shown in Fig. 10(b). The voltage source is removed and two current sources are added. When the current sources send the current $\pm I_s = \pm EG$, then it is easily seen that the resulting equations are the same. In other words, any configuration of voltage sources is reduced to a set of current sources and its treatment is like the treatment of $I_{\text{known}}$.

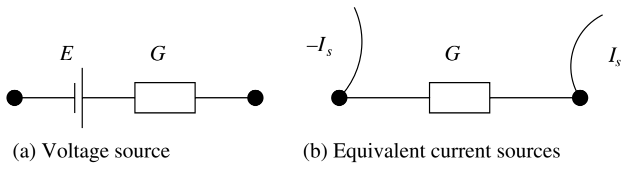

Fig. 10. Any voltage source inside a network can be replaced by two current sources. The value of the current is $I_s = EG$, where $E$ is the voltage drop over the source in (a) and $G$ is the conductance of the connection.

## 5.1. Application to flow problems

Physical problems other than current transport in resistor networks obey similar transport laws. One example is the modelling of fluid flow through porous media. Frequently, the medium is modelled as a network of tubes in which the fluid flows. Each tube corresponds to a resistor having a certain mobility. Each node has a pressure, which corresponds to the voltage in the electric current case. Furthermore, the counterpart of electric current $I$ is the volume flux $Q$. See Table 1 for a list of corresponding quantities. In order to find the fluid flow, one applies the Kirchhoff's equations in the same way as one would when solving for electric current in a network.

Returning to mobility. Each tube's mobility is based on two factors. One is the so-called permeability, which is a geometrical property and which is usually fixed. The other factor is the viscosity of the fluid, which in a given problem might show time dependence. Disregarding the time development and only looking at an instantaneous solution of the flow equations, one has to solve the same flow problem as with electric current. Thus, the results of this paper apply also to the flow problem. We want to make a general remark about the boundary conditions that appear in this problem.

In general, in flow simulations one does not want to provide a fixed pressure drop over a system (corresponding to a fixed voltage drop). Instead one wishes to solve for a constant volume flux through the whole system, that is to say it is a global constraint. A special case occurs when one can define, for instance, one of the end rows as a single point and connect to it a flux source with the desired flux. In cases like these it is possible to apply this global boundary condition directly when using the conjugate gradient method for solving the equations. It would also be possible if beforehand one knows the global conductance of the whole system but that would require that one had already solved the problem. Generally for the global constraint on the flux, one has to give a fixed pressure drop and solve the equations. Afterwards the actual global flow is calculated. By chance, it could be the desired flux. Normally it is not. Two solutions exist in this case. One is to solve the flow equations twice for two different pressure drops. Based on the two solutions one can calculate the solution with the desired global flux, see [6,13]. A different approach is to accept some error in the global flux. That is to say that one is able to make a rather good guess on the global pressure drop, such that the flux is close to the desired flux. In the case of a time development the guess for the pressure drop needs to be updated. This approach is described in more detail in [14].

These problems concerning solving for constant flux are not really an issue for Gaussian elimination procedures. After the forward elimination is performed, both boundary conditions can be applied equally easily. In the simplest case when the flow is between two points, the value of the global conductance is known. If the pressure drop is given, the global flow is calculated or vice versa. Should there be a global constraint on the flux, then one has to apply two different voltage sources as with the conjugate gradient method. The difference is that only the calculation of the right-hand side and the backward substitution need to be performed twice. Therefore, the node elimination method has a clear advantage over conjugate gradient methods for this class of flow simulations.

Flow simulations often deal with two immiscible fluids flowing in the system. Due to time development the location of fluids changes, and thus each tube may contain different amounts of the two fluids at

Table 1
The table lists the corresponding variables of fluid flow within networks and electric current variables

| Flow quantity | | Electric quantity | |
|---|---|---|---|
| Pressure | $P$ | Voltage | $V$ |
| Mobility | $M$ | Conductance | $G$ |
| Capillary pressure | $P_c$ | Voltage source | $E$ |
| Volume flux | $Q$ | Current | $I$ |

Both problems lead to the same kind of equations.

different times. The fluids normally have different viscosities such that the mobilities in the system change with time. Two fluids also imply the presence of interfaces between the two fluids. Interfaces imply interfacial tension, which means that at the location of the interfaces there are small pressure drops, so-called capillary pressure. The analogue of such a pressure drop in current networks is the electro-motoric voltage source. As already discussed, voltage sources are equivalent to current sources and they enter only in the right-hand side of the equations.

A special case occurs when the two fluids actually have the same viscosity. In that case the mobilities are always equal, i.e., the left-hand side of the equations does not change with time. However, the location of the interfaces may change with time, such that the right-hand side of the system of equations also changes with time. Hence, when following such a system through time, having to solve the same equations over and over again with a new right-hand side, it is enough to do the forward substitution of the left-hand side once. Thereafter, for every time step, the right-hand side is recalculated as described in the previous subsection and the entire backward substitution is done. Knowing that the backward substitution is much faster than the forward substitution, it is clear that much CPU time can be saved. In the literature this problem is usually attacked with the conjugate gradient method, with which the whole system of equations needs to be solved from scratch every time.

A final situation for which the node elimination method is better than conjugate gradient is the simulation of gas-liquid systems. The viscosity of gas and liquid can easily differ with as great a factor as $10^5$. This difference enters into the mobility values. As was seen in Section 4, the conjugate gradient method becomes slower and slower when the coefficients in the equations vary over more and more orders of magnitude. In fact, at some point the iterative scheme simply does not converge anymore. These systems which are unsolvable using this method can still be solved with the node elimination method. Clearly some precision is lost when the coefficients vary so extremely, but still a good solution can be found.

### 5.2. Application to fracture models

Fracture has been modelled by means of random fuse models [7]. The system of equations is the same as with the random resistor network. By letting the fuses burn out one after one, a time development resembling a fracture process is achieved. Whenever a fuse burns out, its conductance becomes zero. This small change in the network leads to a new current distribution that needs to be calculated from scratch.

If the fuse that burned out was connected to nodes that are eliminated at a late stage in the node elimination procedure, then its burn-out only affects the latter part of the elimination. Thus, time can be saved by only redoing a part of the elimination process. In order to exploit this possibility a very clever ordering of the equations is needed. That is only possible if beforehand one is able to identify some regions in space, in which the fuses will burn out first. By sorting these regions at the end of the elimination list, only a part recalculation is needed as long as the changes to the network stay within these regions.

The presence of already burned out fuses forces neighbouring fuses to carry more current. Thus, the chance that these neighbours burn out is larger than for the fuses which are further away. Therefore, in some cases, it should be possible to identify regions in which new burn-outs will certainly take place.

The implementation might be somewhat tedious and we have not tried it ourselves. However, we think the idea is promising. With the conjugate gradient solver this possibility does not exist. Much time could be saved in a simulation if only part recalculations were needed in most time steps.

## 6. Conclusion

In this study we discuss methods for solving the linear transport equations that arise in physical networks, i.e., Kirchhoff's laws. The methods are: the node elimination method, the transfer matrix method, the Gaussian elimination and the conjugate gradient iterative solver.

In physics, the node elimination method, the transfer matrix method and Gaussian elimination are often considered different methods. However, we show that this is not the case. Although the perspectives are different, the actual operations or calculations which are done in both the node elimination method and in the transfer matrix method are the same as in the Gaussian elimination scheme. With emphasis on the node elimination method, we discuss an implementation that avoids matrix storage structures. Instead the use of linked lists and clever orderings make the method efficient. This corresponds to the sparse matrix implementation of Gaussian elimination.

Furthermore, ideas which have been developed in applied mathematics and graph theory but have been little known to physicists are also discussed. The concept of domains and subdomains makes it possible to generate a rather good and close to optimal ordering of the elimination process. This is the key reason the node elimination method works so well.

We have performed tests which directly compare the node elimination method with the conjugate gradient method on some sample systems. The general result is that the node elimination method is faster. However, there are many parameters which play a role. Tests on square networks of size $16 \times 16$ to $512 \times 512$ show that node elimination is faster independent of system size in this range. The stiffness of the equations is a key factor here. The conjugate gradient method is very sensitive to the distribution of the coefficients of the problem. It was shown that when the coefficients do not vary much (say within one order of magnitude), then the difference between the methods is minimal. On the other hand, when the coefficients vary with several orders of magnitude, the node elimination method is more than an order of magnitude faster than the conjugate gradient method.

Tests on longer horizontal or vertical strips confirm these findings. Node elimination can in all cases compete well with the conjugate gradient method. Whereas node elimination is rather insensitive to boundary conditions, conjugate gradient is not. For the horizontal strip, where the boundaries are close together, the node elimination method is slightly faster and the methods seem to scale equally with length. For vertical strips, where the boundaries are far apart, the conjugate gradient method is much slower and it scaled much worse with size as was the case for horizontal strips.

The transfer matrix method, which was shown to be a special case of the node elimination method, the elimination being done in one specific order, was also tested on the horizontal strips. It has been the common opinion that this method is well suited for long strips. We find that this is only true for very narrow strips. The node elimination method with domain based ordering is faster than transfer matrix ordering in all cases.

In conclusion, the node elimination method is the better method if a good ordering can be found based on division into domains and subdomains, the coefficients (the conductances) vary over more than one order of magnitude, and for strips in any direction. Furthermore, in some application, for example flow networks, node elimination can be much better combined with the boundary conditions of the problem, possibly avoiding recalculations that would be necessary with the conjugate gradient method. The weakness of the method is the large memory consumption. Thus, very large systems still need to be solved with conjugate gradient or similar methods since they are very memory efficient. If, from the outset, the topology makes it difficult to create sensible domains, the conjugate gradient method may also be preferable.

## Appendix A. Algorithm pseudocodes

### A.1. Forward elimination

It is supposed that there are $n$ nodes. The nodes are numbered from 1 to $n$. Each node (e.g., node $i$) has an associated list of links (*list*[$i$]). Each link in *list*[$i$], except for the head element (*list*[$i$].*head*), represents a connection between node $i$ and a node with a higher index. The inverse connections are not stored because of symmetry. This means that *list*[$n$] contains only the head element. The links (e.g., $L_{ij}$) store a triplet composed of a conductivity value ($L_{ij}.g$), the index of the connected node ($L_{ij}.ind$) and a reference to the next link in the list ($L_{ij}.next$). The conductivity value stored in the head element of *list*[*i*] should be the sum of the conductivities of all the connections of node *i*.

It is supposed further that the external currents $I[i]$ are given, and that their sum is zero (i.e., $\sum_{i=1}^{n} I[i] = 0$). After the node elimination step the conductivity values stored in the head elements should be the sum of the conductivities of the other elements in the same list. Consequently the conductivity values stored in the head of the last list (i.e., *list*[*n*].*head*.*g*) should be zero. Similarly $I[n]$ should be zero, meaning that the voltage on the last node can be chosen freely.

```
for $i \leftarrow 1$ to $n-1$
  $L_{ii} \leftarrow list[i].head$
  $L_{ij} \leftarrow L_{ii}.next$
  while $L_{ij} \neq$ NIL
    $j \leftarrow L_{ij}.ind$
    $gRatio \leftarrow L_{ij}.g/L_{ii}.g$
    $L_{jj} \leftarrow list[j].head$
    $L_{jj}.g \leftarrow L_{jj}.g - gRatio * L_{ij}.g$
    $L_{ik} \leftarrow L_{ij}.next$
    $L'_{jk}, L_{jk} \leftarrow L_{jj}, L_{jj}.next$
    while $L_{ik} \neq$ NIL
      $k \leftarrow L_{ik}.ind$
      $gComp \leftarrow gRatio * L_{ik}.g$
      while $L_{jk} \neq$ NIL and $L_{jk}.ind < k$
        $L'_{jk}, L_{jk} \leftarrow L_{jk}, L_{jk}.next$
      if $L_{jk} \neq$ NIL and $L_{jk}.ind = k$
        $L_{jk}.g \leftarrow L_{jk}.g + gComp$
      else
        $L'_{jk}.next \leftarrow$ newLink$(g, ind, next \leftarrow gComp, k, L_{jk})$
        $L'_{jk} \leftarrow L'_{jk}.next$
      $L_{ik} \leftarrow L_{ik}.next$
    $I[j] \leftarrow I[j] + gRatio * I[i]$
    $L_{ij} \leftarrow L_{ij}.next$
```

### A.2. Backward substitution

The calculated voltage of node *i* will be stored in $V[i]$. The voltage of the last node (i.e., $V[n]$) can be choosen arbitrarily as a reference value. Isolated nodes will have undefined voltages.

```
for $i \leftarrow n-1$ downto 1
  $L_{ii} \leftarrow list[i].head$
  $L_{ij} \leftarrow L_{ii}.next$
  if $L_{ij} \neq$ NIL
    $iSum \leftarrow -I[i]$
    while $L_{ij} \neq$ NIL
      $j \leftarrow L_{ij}.ind$
      $iSum \leftarrow iSum + L_{ij}.g * V[j]$
```

```
            $L_{ij} \leftarrow L_{ij}.next$
        $V[i] \leftarrow iSum/L_{ii}.g$
    **else**
        $V[i] \leftarrow$ undefined
```

## A.3. Recalculation of the right-hand side

In the recalculation of the right-hand side, one can make use of the fact that some external currents may be zero, avoiding a loop and a number of calculation steps.

```
**for** $i \leftarrow 1$ **to** $n-1$
    **if** $I[i] \neq 0$
        $L_{ii} \leftarrow list[i].head$
        $L_{ij} \leftarrow L_{ii}.next$
        **while** $L_{ij} \neq$ NIL
            $j \leftarrow L_{ij}.ind$
            $gRatio \leftarrow L_{ij}.g/L_{ii}.g$
            $I[j] \leftarrow I[j] + gRatio*I[i]$
            $L_{ij} \leftarrow L_{ij}.next$
```

## References

[1] R. Fogelholm, The conductivity of large percolation network samples, J. Phys. C: Solid State Phys. 13 (1980) L571–L574.

[2] G.G. Batrouni, A. Hansen, Fourier acceleration of iterative processes in disordered systems, J. Stat. Phys. 52 (1988) 747–773.

[3] G.G. Batrouni, A. Hansen, B. Larson, Current distribution in the three-dimensional random resistor network at the percolation threshold, Phys. Rev. E 53 (1996) 2292–2297.

[4] D.J. Frank, C.J. Lobb, Highly efficient algorithm for percolative transport studies in two dimensions, Phys. Rev. B 37 (1988) 302–307.

[5] J.P. Clerc, V.A. Podolskiy, A.K. Sarychev, Precise determination of the conductivity exponent of 3D percolation using exact numerical renormalization, Eur. Phys. J. B 15 (2000) 507–516.

[6] H.A. Knudsen, E. Aker, A. Hansen, Bulk flow regimes and fractional flow in 2D porous media by numerical simulations, Transport Porous Med. 47 (2002) 99–121.

[7] G.G. Batrouni, A. Hansen, Fracture in three-dimensional fuse networks, Phys. Rev. Lett. 80 (1998) 325–328.

[8] J. Bird, Electrical Circuit Theory and Technology, Newnes, 2003.

[9] B. Derrida, J. Vannimenus, A transfer-matrix approach to random resistor networks, J. Phys. A: Math. Gen. 15 (1982) L557–L564.

[10] B. Derrida, J.G. Zabolitzky, J. Vannimenus, D. Stauffer, A transfer matrix program to calculate the conductivity of random resistor networks, J. Stat. Phys. 36 (1984) 31–42.

[11] S. Parter, The use of linear graphs in Gauss elimination, SIAM Rev. 3 (1961) 119–130.

[12] J. Schulze, Towards a tighter coupling of bottom–up and top–down sparse matrix ordering methods, BIT Num. Math. 41 (2001) 800–841.

[13] E. Aker, K.J. Måløy, A. Hansen, G.G. Batrouni, A two-dimensional network simulator for two-phase flow in porous media, Transport Porous Med. 32 (1998) 163.

[14] M. Al-Gharbi, M.J. Blunt, Dynamic network modeling of two-phase drainage in porous media, Phys. Rev. E 71 (2005) 016308.