INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng. **43**, 131–149 (1998)

NBS CONTACT DETECTION ALGORITHM FOR BODIES OF SIMILAR SIZE

A. MUNJIZA * AND K. R. F. ANDREWS
Department of Engineering, QMW, University of London, London, U.K.

ABSTRACT

Large-scale discrete element simulations, as well as a whole range of related problems, involve contact of a large number of separate bodies. In this context an efficient and robust contact detection algorithm is necessary. There has been a number of contact detection algorithms with total detection time (CPU time needed to detect all couples close to each other) proportional to $N \ln(N)$ (where N is the total number of separate bodies) reported in recent years.

In this work a contact detection algorithm with total detection time proportional to N is reported. The algorithm is termed NBS, which stands for *no binary search*. In other words, the proposed algorithm involves no binary search at any stage. In addition the performance of the algorithm in terms of total detection time is not influenced by packing density, while memory requirements are insignificant. The only limitation of the algorithm is its applicability to the systems comprising bodies of similar size. © 1998 John Wiley & Sons, Ltd.

KEY WORDS: contact detection; discrete elements; binary search

1. INTRODUCTION

Discrete element methods[1–5] are characterized by systems comprising a large number of separate (distinct) bodies often called discrete elements. Discrete elements are usually free to move in space and time and thus interact with each other, while discrete element-like static formulations are often termed Discontinuous Deformation Analysis or DDA.[6]

In discrete element methods, each discrete element is considered as a separate body that interacts with neighbouring elements. Routine discrete element problems may involve millions of interacting discrete elements and in this context a fast and efficient contact detection algorithm is necessary. In other words, the effective solution of large-scale discrete element problems relies upon a robust contact detection algorithm being employed.

This is also the case for the combined finite-discrete element method,[5] where each discrete element is discretised into finite elements, while the distribution of individual discrete elements, shape of individual discrete elements and the total number of discrete elements all change in time.

The optimal contact detection algorithm is in general dependent upon the problem to be solved. The properties of different contact detection algorithms make them suited for different types of problems such as: *dense packing* and *loose packing* (Figure 1); or *quasi-static problems* (where

* Correspondence to: A. Munjiza, Department of Engineering, Queen Mary and Westfield College, University of London, Mile End Road, London, E1 4NS, U.K. E-mail: a.munjiza@gmw.ac.uk

CCC 0029–5981/98/010131–19$17.50

Received 15 April 1997
Revised 23 February 1998

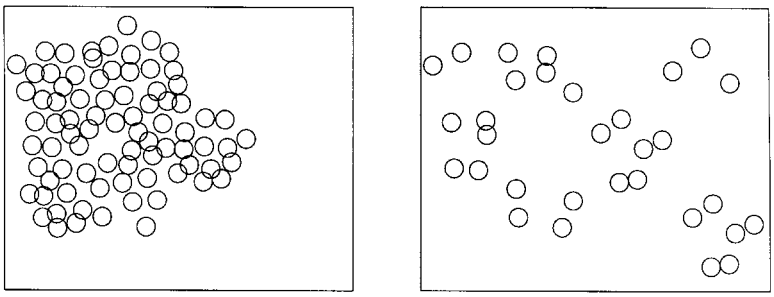

Figure 1. Packing density: (a) dense packing; (b) loose packing

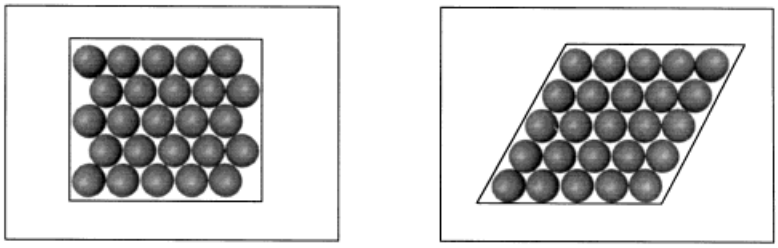

Figure 2. Typical quasi-static problem

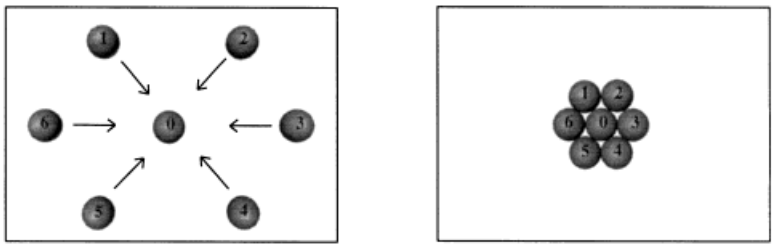

Figure 3. Typical dynamic problem

relative motion of individual bodies is restricted, Figure 2) and *dynamic problems* (where individual bodies move significantly, Figure 3).

For instance, there exist a whole class of contact algorithms employed with finite element systems. Most of these algorithms can handle only problems where relative motion of bodies is restricted and thus the performance of the contact detection algorithm is not very important (because contact detection is performed only once).

Real problems where contact detection is a big issue are dynamic problems, comprising large numbers of discrete elements that are free to move significantly, Figure 3. Contact detection itself can take a considerable proportion of total CPU time required to analyse such problems (up to 60 per cent in some cases). In this context, general requirements on contact detection algorithms include:

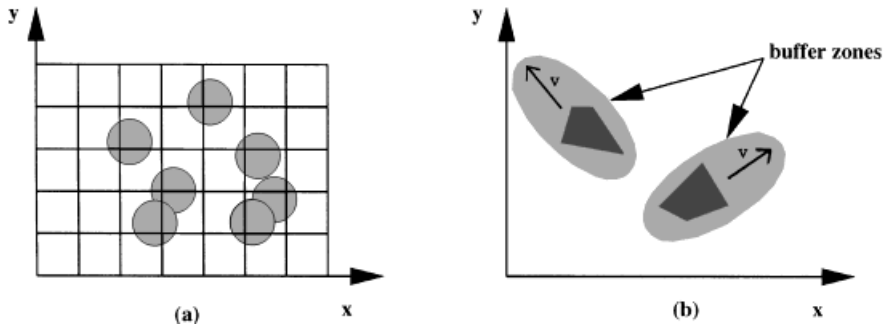

Figure 4. (a) Space divided into sub-spaces; (b) buffer zones for individual bodies

1. Minimization of CPU requirements, i.e. total detection time T defined as total CPU time needed to detect all couples close to each other.
2. Minimization of total memory (RAM) requirements M expressed in terms of total memory size as a function of total number of bodies and packing density.
3. Flexibility in terms of rate of M and T change with change in packing density.

In recent years a number of contact detection algorithms for large scale problems have been reported.[7–11] Most of these algorithms can be classified either as body-based search or space-based search, Figure 4, while procedures applied usually involve binary search with efficiency in terms of total detection time T proportional to

$$T \propto N \ln(N) \tag{1}$$

where N is the total number of discrete elements. Again, some of these algorithms perform better for either loose or dense packing, while the others have been designed for both loose and dense packing. For example, the performance of co-ordinate hashing algorithms depends on the number of overlaping boxes, while co-ordinate sorting algorithms suffer from clustering problems.[11]

In this work a so-called No Binary Search (NBS) contact detection algorithm for problems involving a large number of bodies and significant motion of those bodies, is introduced. The algorithm has total detection time T proportional to the total number of discrete elements, i.e.

$$T \propto N \tag{2}$$

which is a performance superior to the one specified by (1). The total memory requirements M are proportional to the total number of discrete elements

$$M \propto \begin{cases} N \dfrac{1}{\sqrt[2]{\rho}} & \text{for 2D problems} \\ N \dfrac{1}{\sqrt[3]{\rho}} & \text{for 3D problems} \end{cases} \tag{3}$$

where ρ is the packing density (the total number of discrete elements per unit volume of space).

The NBS algorithm works equally well for both dense and loose packing, with CPU time being independent of packing density and memory requirements increasing insignificantly with decrease

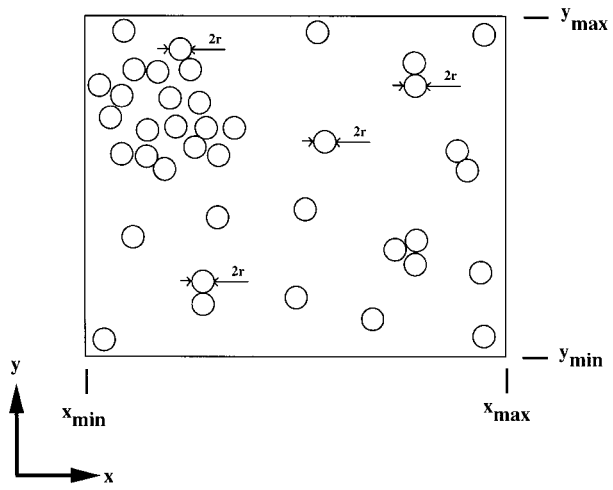

Figure 5. Contact detection problem

in packing density—in 2-D problems a 25-fold decrease in packing density results in less than 5-fold increase in memory requirement while in 3-D problems a 125-fold decrease in packing density results in less than 5-fold increase in memory requirement.

In the remainder of the paper, the algorithmic description of the NBS contact detection algorithm for 2-D problems is presented (extrapolation to 3-D problems, although straight forward, is omitted). The theoretically predicted performance of the algorithm is tested on a whole range of problems differing in *packing density* ρ and in total number of discrete elements N.

At the end of the paper, the algorithm is illustrated through application on a large-scale problem comprising 0·5E+06 circular discrete elements. The example is run on a medium size workstation with an aim to illustrate performance of the algorithm in terms of both memory and CPU requirements.

2. NBS ALGORITHM—DEFINITION OF THE CONTACT DETECTION PROBLEM

The NBS contact detection algorithm is based on the assumption that each discrete element for contact detection purposes can be approximated with a sphere in 3-D or with a circular disc in 2-D. The diameter of an equivalent circular disc $2r$ is obtained from the size of the largest discrete element in the system, i.e. all discrete elements are approximated with identical circular discs.

Thus the NBS contact detection algorithm assumes a system comprising N identical discs occupying finite space of rectangular shape, Figure 5.

The space boundaries are defined by $x_{\min}$, $x_{\max}$, $y_{\min}$ and $y_{\max}$. No circular disc has co-ordinate x of its centre (co-ordinate x in further text) greater than $x_{\max}$ or smaller than $x_{\min}$ and also no circular disc has co-ordinate y of its centre (co-ordinate y in further text) greater than $y_{\max}$ or smaller than $y_{\min}$.

The task is to find all disc couples that are close to each other in a sense that the distance between their closest points is less than or equal to zero, in other words that they overlap or touch.

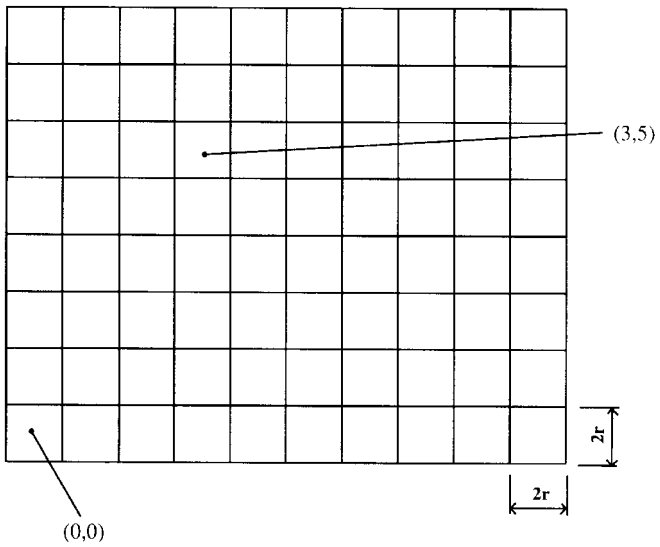

Figure 6. Space decomposition

3. NBS ALGORITHM—SPACE DECOMPOSITION

The NBS contact detection algorithm is based on space decomposition. The space is subdivided into identical square cells of size $2r$, Figure 6.

Each disc is assigned a integer identification number $\{0, 1, 2, \ldots, N-1\}$. In a similar fashion each cell is assigned an identification couple of integer numbers (ix, iy), where $ix = 0, 1, 2, \ldots,$ $\mathrm{ncel}x - 1$ and $iy = 0, 1, 2, \ldots, \mathrm{ncel}y - 1$, while $\mathrm{ncel}x$ and $\mathrm{ncel}y$ are the total number of cells in X and Y direction respectively

$$\mathrm{ncel}x = \frac{x_{\max} - x_{\min}}{2r} \tag{4}$$

$$\mathrm{ncel}y = \frac{y_{\max} - y_{\min}}{2r} \tag{5}$$

Mapping from the set of discs

$$E = \{0, 1, 2, 3, \ldots, N\} \tag{6}$$

to the set of cells

$$C = \begin{Bmatrix} (0,0), & (0,1), & \ldots & (0, \mathrm{ncel}y - 1), \\ (1,0), & (1,1), & \ldots & (1, \mathrm{ncel}y - 1), \\ (\mathrm{ncel}x - 1, 0), & (\mathrm{ncel}x - 1, 1), & \ldots & (\mathrm{ncel}x - 1, \mathrm{ncel}y - 1) \end{Bmatrix} \tag{7}$$

is defined in such a way that each discrete element is assigned to one and only one cell. For instance, the disc with coordinates (x, y) is assigned to the cell (ix, iy), where

$$ix = \mathrm{Int}\left(\frac{x - x_{\min}}{2r}\right) \tag{8}$$

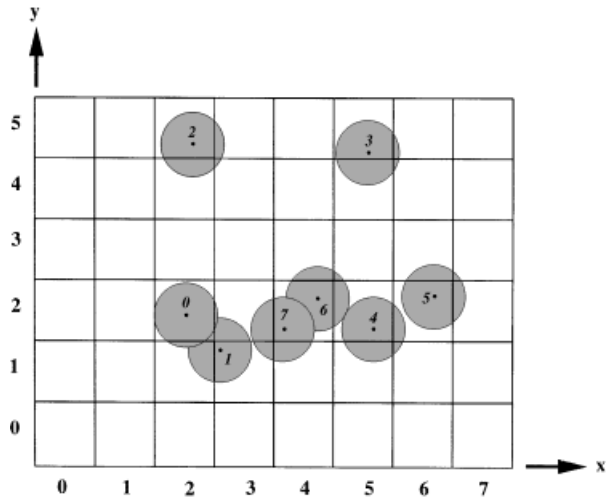

Figure 7. Mapping of discs onto cells

$$iy = \mathrm{Int}\left(\frac{y - y_{\min}}{2r}\right) \tag{9}$$

i.e. ix and iy are integerised relative co-ordinates $(x - x_{\min})/(2r)$ and $(y - y_{\min})/(2r)$ (referred to as integerised co-ordinates ix and iy in further text). An example is given in Figure 7, where for instance, disc 0 is assigned to the cell $(2,2)$, while disc 1 is assigned to the cell $(3,1)$ and disc 3 is assigned to the cell $(5,5)$.

In addition a disc is said to be mapped to a particular row of cells if it is mapped to any cell from that row. In a similar way, a disc is said to be mapped to a particular column of cells if it is mapped to any cell from that column.

The obvious way to represent the mapping as described above is by representing cells by a 2-D array of size

$$\mathrm{ncel} = \mathrm{ncel}x \cdot \mathrm{ncel}y \tag{10}$$

This is extremely expensive in terms of RAM requirements, which is especially so for loose packing, where the total number of discs N may be small compared to the total number of cells ncel. The size of the problem affordable by a given hardware is therfore heavily dependent on the distribution of discs in space. This is an unacceptable constraint if free motion of discs is to be considered.

In this work *linked lists* are used instead. This reduces memory requirements significantly as demonstrated below. The exact procedure employed is performed in two steps as explained through the example presented in Figure 7.

Firstly, mapping of all discs to the rows of cells (Y direction) is performed and a singly connected list of discs (referred to as the Y_{iy} list) for each row iy (where $iy = 0, 1, 2, 3, \ldots, \mathrm{ncel}y - 1$) is formed, Figure 8.

Discs are mapped by looping over all discs in ascending numerical order, checking their integerized co-ordinate iy and 'pushing' them along the list as shown schematically in Figure 8. Thus, the list for row $iy = 2$ (i.e. the list Y_2) in Figure 8 was assembled by placing the disc 0 onto the

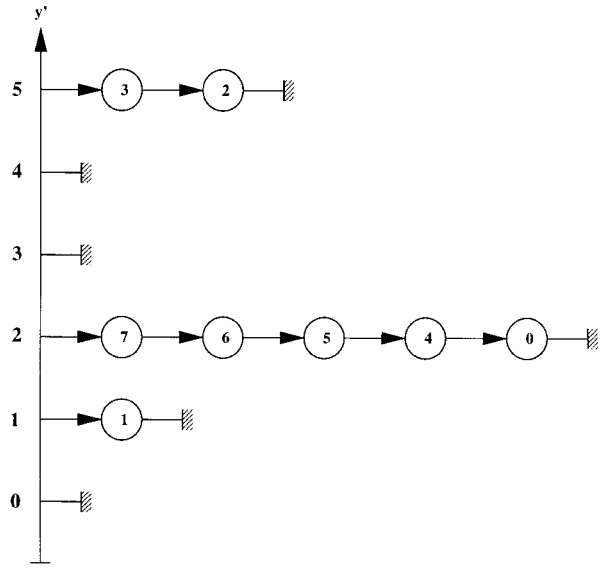

Figure 8. Y_{iy} lists (where $iy = 0, 1, 2, 3, \ldots, \mathrm{ncel}y - 1$)

list, and then 'pushing' it by disc 4, which is 'pushed' by disc 5, which is 'pushed' by disc 6, which is 'pushed' by disc 7, which is the last disc added to the list. This is represented by two integer arrays. The first array head$y$ contains the number of the last disc mapped to each row. The array head$y$ is a 1-D array of size ncel$y$, where ncel$y$ is the total number of cells in Y direction, i.e. the total number of rows of cells. The second array next$y$ is 1-D array of size N, where N is the total number of discs. For each disc the array next$y$ contains the next disc in the singly connected list. For both arrays a negative number is used as termination of a singly connected list. So if there are no elements in a particular row a negative number is assigned to the corresponding element of the array head$y$. A visualization of this representation is shown in Figure 9, where the list Y_2 is highlighted. In short head$y$ represents a 'pointer' to a singly connected list of discs for each iy-row (where $iy = 0, 1, 2, 3, \ldots, \mathrm{ncel}y - 1$) of cells. In a similar way, the table next$y$ points to the next element in the list, terminating with a negative number.

For example, shown in Figure 7, row 0 of cells has no discs mapped to it, so singly connected list of discs mapped to this row (the list Y_0) is empty, which is achieved by setting head$y[0] = -1$. In a similar way, the last dics mapped to the row 2 of cells (the list Y_2) is disc number 7, thus head$y[2] = 7$. The next disc on this list is 6, thus next$y[7] = 6$; the next disc is 5, thus next$y[6] = 5$; the next disc is 4, thus next$y[5] = 4$; the last disc is 0, thus next$y[4] = 0$ and next$y[0] = -1$, Figure 9.

All Y_{iy} lists are marked as 'new' at this point.

Secondly, by looping over all discs, a so-called 'new' Y_{iy} list is detected and marked as a so-called 'old' list. Each disc from this list is placed onto a corresponding so called (X_{ix}, Y_{iy}) list depending on its integerized co-ordinate ix (each (X_{ix}, Y_{iy}) list is marked as 'new' at this point). A particular (X_{ix}, Y_{iy}) list contains all discs with integerized co-ordinates ix and iy. In addition, all singly connected lists (X_{ix}, Y_{iy}) (where $ix = 0, 1, 2, \ldots, \mathrm{ncel}x - 1$) contain all discs from the list Y_{iy} and are represented by two arrays of integer numbers. The first table is a 1-D table head$x$

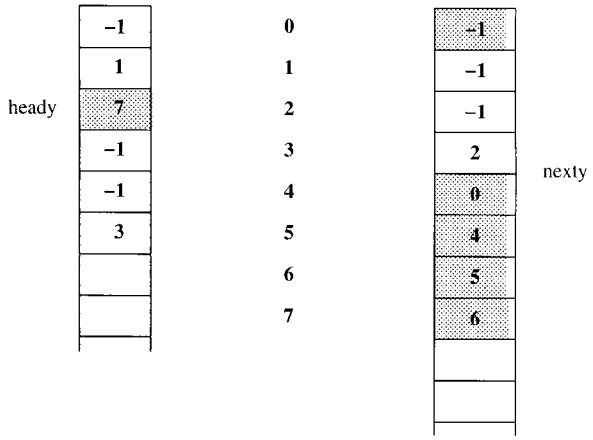

Figure 9. Numerical representation of the singly connected list from Figure 8

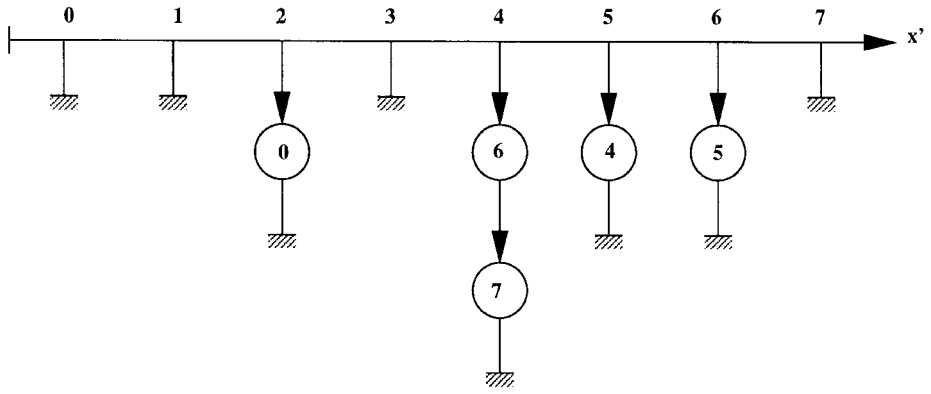

Figure 10. (X_{ix}, Y_2) lists (where $ix = 0, 1, 2, \ldots,$ ncel$x - 1$)

of size ncel$x$, where ncel$x$ is the total number of cells in X direction, i.e. the total number of columns of cells. The second table is a 1-D table next$x$ of size N, where N is the total number of elements.

Singly connected lists (X_{ix}, Y_2) (where $ix = 0, 1, 2, \ldots,$ ncel$x - 1$) for example from Figure 7 are shown in Figure 10. Numerical representation of those lists by tables of integer numbers is shown in Figure 11, where the list (X_4, Y_2) is highlighted.

The list Y_2, Figure 8, contains all discs in row 2 of cells, i.e. discs $7, 6, 5, 4$ and 0. No disc from the list Y_2, Figure 8, has an integerized co-ordinate ix equal to $0, 1, 3$ or 7, thus singly connected lists (X_0, Y_2), (X_1, Y_2), (X_3, Y_2) and (X_7, Y_2) are empty, i.e. head$x$[0], head$x$[1], head$x$[3] and head$x$[7] are all assigned negative value. Only disc 4 has integerized co-ordinate ix equal to 5, thus head$x$[5] $= 4$ and next$x$[4] $= -1$. Only discs 6 and 7 have integerized co-ordinates ix equal to 4, thus head$x$[4] $= 6$, next$x$[6] $= 7$ and next$x$[7] $= -1$.

It is important to note how the lists Y_{iy} and (X_{ix}, Y_{iy}) are assembled. First, a loop over all discs is performed and inside the loop a particular disc is added to the corresponding Y_{iy} list depending

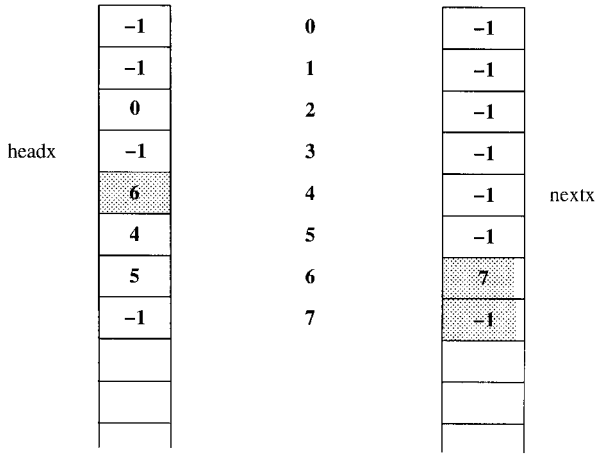

Figure 11. Representation of singly connected lists from Figure 10

on its integerized co-ordinate iy. Second, a loop over all discs for a particular Y_{iy} list is performed and inside this loop a particular disc is added to the corresponding (X_{ix}, Y_{iy}) list depending on its integerized co-ordinate ix.

In order to assemble new lists, discs are removed from the old lists in a similar fashion—a loop over discs from a particular Y_{iy} list is performed and for each disc head$x[ix] = -1$ is set and in a similar way a list over all discs is performed and for each disc head$y[iy] = -1$ is set.

Thus no loop over cells is involved, which leads to a conclusion that the total CPU time needed to perform operations described so far in this section is not a function of ncel$x$ or ncel$y$. In other words it is not a function of packing density ρ.

4. NBS ALGORITHM—DETECTION OF CONTACT

Detection of contact is accomplished by checking all discs mapped to a particular cell against all discs in neighbouring cells. For instance, discs mapped to the cell (ix, iy) shown in Figure 12, are checked for contact against all discs mapped to cells (ix, iy), $(ix-1, iy)$, $(ix-1, iy-1)$, $(ix, iy-1)$ and $(ix+1, iy-1)$. This is equivalent to checking all discs from the list (X_{ix}, Y_{iy}) against all discs from the lists (X_{ix}, Y_{iy}), (X_{ix-1}, Y_{iy}), (X_{ix-1}, Y_{iy-1}), (X_{ix}, Y_{iy-1}) and (X_{ix+1}, Y_{iy-1}).

This way discs mapped to any non-empty cell will be checked against all discs mapped to neighbouring cells (only discs from neighbouring cells can touch each other).

Thus, it is necessary at any given time to have singly connected lists (X_{ix}, Y_{iy}) only for two neighbouring rows of cells iy and $iy-1$, i.e. for discs from lists Y_{iy} and Y_{iy-1}. Thus, 2 parallel arrays head$x$ are needed and this is accomplished through a 2-D array heads$x$ of size 2 ncel$x$, i.e. heads$x$[2][ncel$x$], where the array heads$x$[0] points to singly connected lists (X_{ix}, Y_{iy}) (where $ix = 0, 1, 2, \ldots,$ ncel$x - 1$) while the array heads$x$[1] points to singly connected lists (X_{ix}, Y_{iy-1}) (where $ix = 0, 1, 2, \ldots,$ ncel$x - 1$).

Detection of contact is performed only for cells that have one or more discs mapped to them, i.e. for the cells with a non-empty (X_{ix}, Y_{iy}) list of discs. This is accomplished by employing a loop over discs from the list Y_{iy} in order to find a cell (ix, iy) with one or more discs assigned to

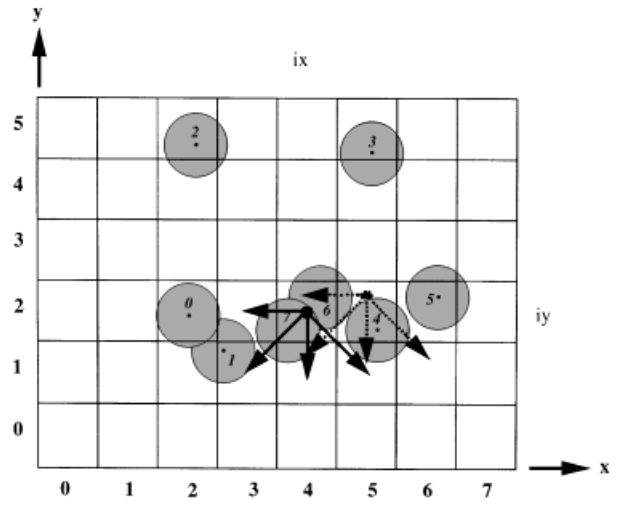

Figure 12. Detection of contact

it, i.e. a 'new' (X_{ix}, Y_{iy}) list, (which is marked as 'old' at this point). Discs mapped to each such cell are checked for contact against all discs mapped to neighbouring cells, as shown in Figure 12. For instance for the list Y_2 from Figure 10 the loop over all discs from the list will include discs $7, 6, 5, 4$ and 0, resulting in 'new' lists (X_4, Y_2), (X_6, Y_2), (X_5, Y_2) and (X_2, Y_2), respectively. Thus discs from lists (X_4, Y_2), (X_6, Y_2), (X_5, Y_2) and (X_2, Y_2) are checked for contact against discs from lists of discs mapped to neighbouring cells.

It is noted that no loop over cells is performed for any operation described in this section, which leads to the conclusion that the total CPU time needed to perform detection of contact as described in this section is independent of ncel$x$ and ncel$y$.

5. NBS ALGORITHM—IMPLEMENTATION

The NBS contact detection algorithm can be summarised in the algorithmic form as follows:

1. loop over all discrete elements
{
- **find equivalent circular disc and calculate integerised coordinates of its centre** ix **and** iy **(8, 9)**
- **place current disc onto the** Y_{iy} **list**
- **mark the** Y_{iy} **list as 'new'**

}
2. loop over all discs
{
if the disc belongs to a 'new' list Y_{iy}
{
- **mark the list** Y_{iy} **as 'old'**

3. loop over all discs from the list Y_{iy}
{


```
                ○ place current disc onto (X_ix, Y_iy) list
                ○ mark the (X_ix, Y_iy) list as 'new'
            }
            4. loop over all discs from the list Y_iy-1
            {
                ○ place current disc onto (X_ix, Y_iy-1) list
            }
            5. loop over all discs from the list Y_iy
            {
                if the disc belongs to a 'new' list (X_ix, Y_iy)
                {
                    • mark the list (X_ix, Y_iy) as 'old'
                    check for contact between discs from the list (X_ix, Y_iy)
                    and discs from lists (X_ix, Y_iy), (X_ix-1, Y_iy),
                    (X_ix-1, Y_iy-1), (X_ix, Y_iy-1), (X_ix+1, Y_iy-1)
                }
            }
            6. loop over all discs from the list Y_iy
            {
                remove the list (X_ix, Y_iy), i.e. set headsx[0][ix] = -1;
            }
            7. loop over all discs from the list Y_iy-1
            {
                remove the list (X_ix, Y_iy-1), i.e. set headsx[1][ix] = -1;
            }
        }
    }
    8. loop over all discs
    {
        remove the list Y_iy, i.e. set heady[iy] = -1
    }
```

6. CPU AND RAM REQUIREMENTS

Detailed analysis of the NBS contact detection algorithm as explained in the previous section leads to a conclusion that the total detection time is proportional to the total number of discs present

$$T \propto N \tag{11}$$

or

$$T = c \cdot N \tag{12}$$

where c is a constant independent of either packing density, or number of discs.

This can be proved by analysing each step of the algorithm presented in the previous section:

To make all singly connected lists Y_{iy} (where $iy = 0, 1, 2, 3, \ldots,$ ncel$y - 1$), a loop over all elements is involved. Inside the loop the current element is added to the corresponding singly connected list as specified by its integerized co-ordinate iy.

To make all singly connected lists (X_{ix}, Y_{iy}) (where $ix = 0, 1, 2, 3, \ldots, \text{ncel}x - 1$) for discs from a particular Y_{iy} list, a loop over discs from the Y_{iy} list is employed and inside the loop the current element is added to the corresponding singly connected list (X_{ix}, Y_{iy}) as specified by its integerised coordinate ix.

In order to detect a 'new' Y_{iy} list a loop over all discs is performed, while in order to detect a 'new' (X_{ix}, Y_{iy}) a loop over discs from the Y_{iy} list is performed.

Thus at no place is a loop over cells employed; loops over discs are employed instead. Those consist of:

1. *3 loops over all discs—loops 1,2 and 8*
2. *3 loops over discs from Y_{iy} list—loops 3,5 and 6* (which is equivalent to 3 loops over all discs in total)
3. *2 loops over discs from Y_{iy-1} list—loops 4 and 7* (which is equivalent to 2 loops over all discs in total)

Thus in total an equivalent of 8 loops over all discs is performed. Operations performed inside the loops do not depend on the number of cells in X direction (ncel$x$) or number of cells in Y direction (ncel$y$), i.e. the total CPU time to detect all contacts is not dependent on the number of cells. In other words it is not dependent on the size of the finite space within which discs are distributed. Operations performed inside the loops do not depend on the number of discs N either, i.e. the total CPU time to detect all contact is proportional to the total number of discs (11).

RAM requirements in terms of required RAM space M related to contact detection are easily calculated as the total space occupied by arrays head$y$[ncel$y$], heads$x$[2][ncel$x$], next$y$[N] and next$x$[N]. Thus,

$$M = \text{ncel}y + 2 \cdot \text{ncel}x + 2 \cdot N(\text{integers}) \tag{13}$$

7. TEST PROBLEMS

The statements about T and M made in Section 6 (equations (12) and (13)) are also verified through careful analysis of the actual code and tested on carefully selected examples as explained below.

Example I. Example I consists of N circular discs of diameter D spaced at distance D in X direction and at distance 2-D in Y direction, Figure 13.

Contact detection is solved 10 times for the problem and each time all contacting couples are detected. This is carried out with a profiled debugging version of the code and total CPU time for contact detection is measured for different values of N. All the results are obtained on 'Indy' Silicon Graphics workstation (R4600 133 MHz with 32 Mb RAM space). The cumulative CPU times for all 10 contact detections as a function of the total number of discs comprising the problem are shown in Figure 14.

The results shown are obtained by changing N from $N = 100$ to $N = 90000$. They accurately fit linear relation (11 and 12), which confirms that the total detection time is indeed proportional to the total number of discs.

Example II. Example II consists of $N = 10\,000$ circular discs of diameter D spaced at distance 2-D in Y direction and at variable distance S (spacing) in X direction (packing 'B'), Figure 15.

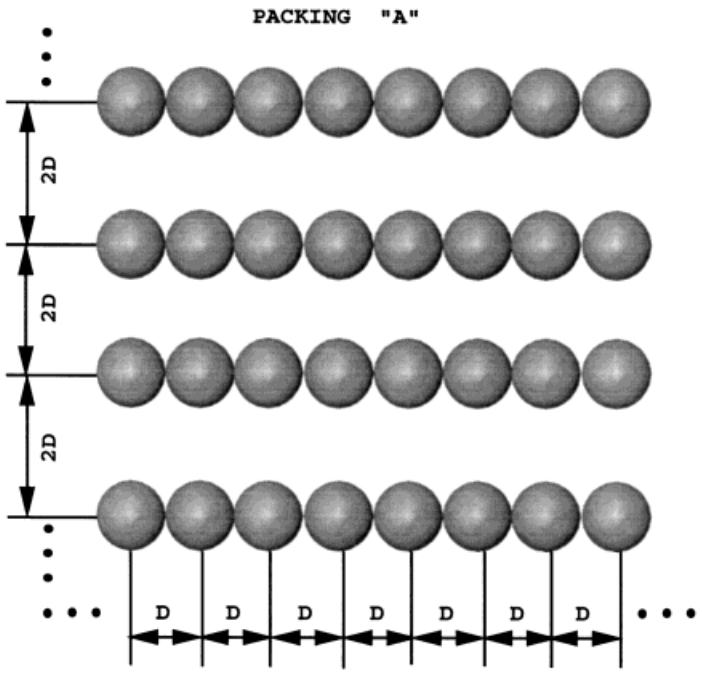

Figure 13. Packing 'A'

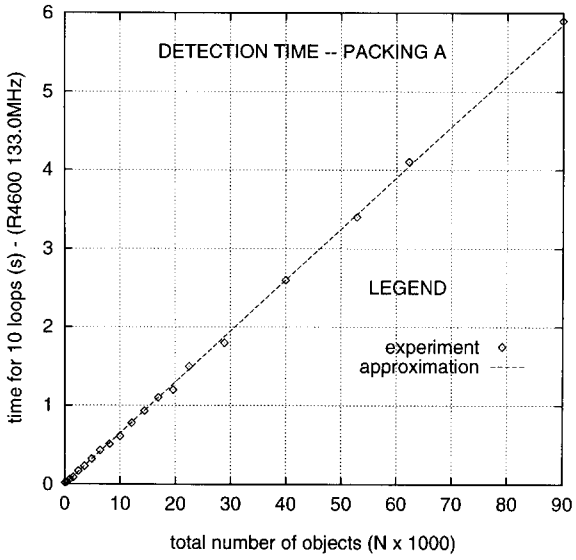

Figure 14. CPU time as function of number of objects

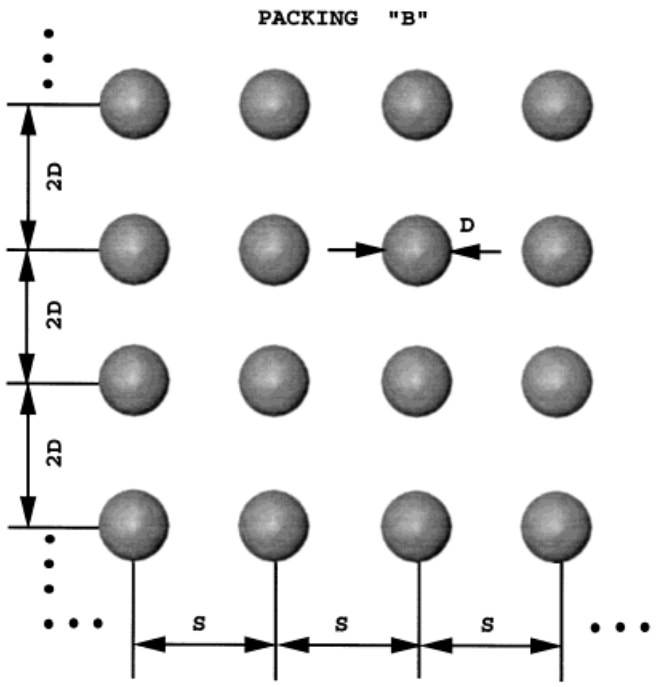

Figure 15. Packing 'B'

The packing density ρ changes with spacing S as

$$\rho \propto \frac{1}{S}$$

Thus, the packing density ρ is changed by changing spacing S.

In Figure 16 the cumulative CPU time for 10 repeated detections of all contacts is shown as a function of spacing. The results obtained in this numerical experiment confirm that the total detection time does not depend on packing density (which is a result predicted in Section 6).

Example III. Example III consists of $N = 10\,000$ circular discs of diameter D spaced at distance S in X direction and at distance S in Y direction, Figure 17.

Contact detection is repeated 10 times for each value of S and cumulative CPU time for all 10 detections of all contacts is recorded as shown in Figure 18. The results obtained show that total detection time is not dependent on packing density, relative change of which goes from 1 to $1/(200 \cdot 200) = 1/40000$.

Example IV. Example IV consists of $N = 10\,000$ circular discs of diameter D spaced at distance 2-D in Y direction and at distance D and S in X direction, Figure 19.

Again for each given value of S contact detection is performed 10 times and cumulative CPU time for all 10 detections of all contacts is recorded using profiled debugging version of the code. The results are shown in Figure 20. The results clearly indicate that total detection time is constant for a large range of packing densities.

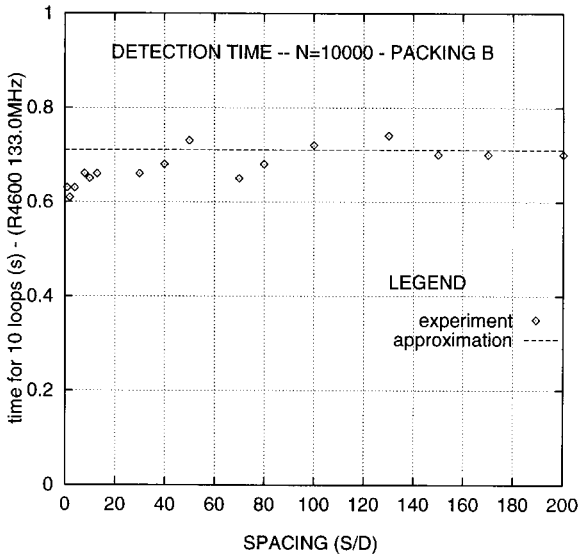

Figure 16. CPU time as function of packing density (packing B)

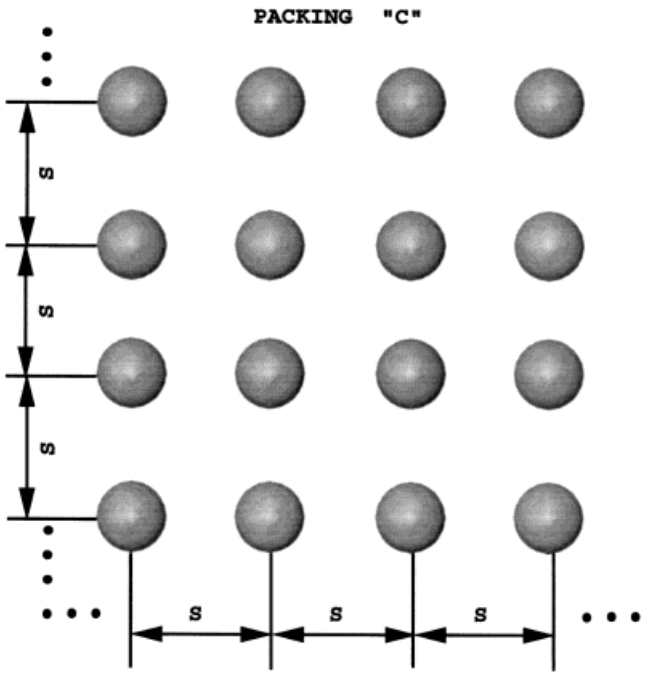

Figure 17. Packing 'C'

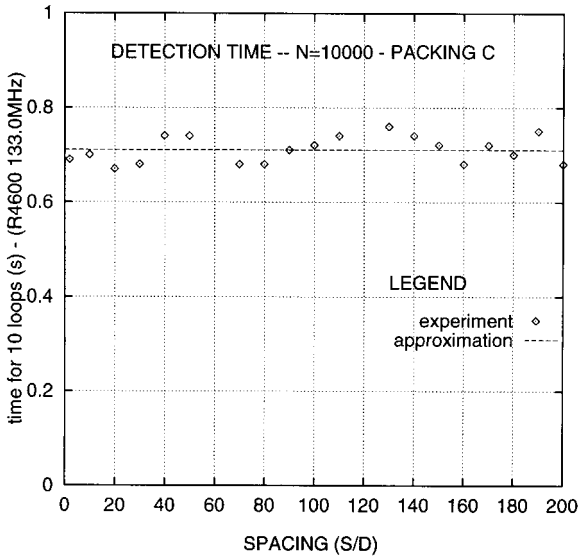

Figure 18. CPU time as function of packing density

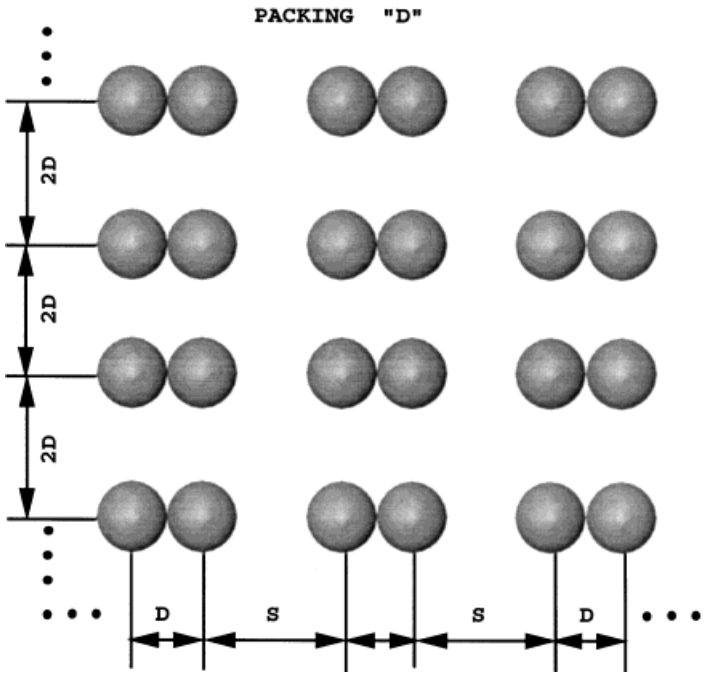

Figure 19. Packing 'D'

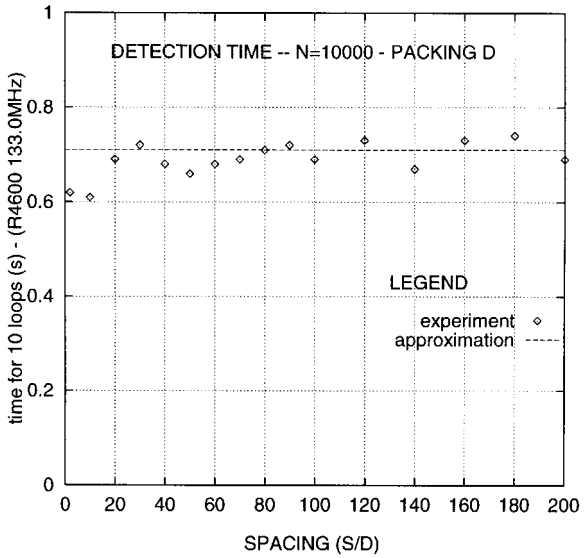

Figure 20. CPU time as function of packing density (packing D)

8. CONCLUDING REMARKS

A so-called *NBS* (no binary search) contact detection algorithm for large-scale discrete element simulations involving bodies of similar size is presented in this paper.

Memory requirements of the algorithm are insignificant and do not change significantly with considerable change in packing density.

Total detection time for the algorithm does not depend on packing density, a result confirmed by both theoretical investigations and numerical experiments.

On the other hand, the relationship between the total detection time and the total number of discrete elements comprising the problem is linear, again a result confirmed by both theoretical investigations and numerical experiments. In this context, for the class of problems defined in Section 2, the NBS contact detection algorithm has better performance than binary search-based algorithms as shown in Figure 21.

In Figure 21 the theoretical total detection time T as a function of total number of discrete elements N for various types of algorithms (see Section 5) is shown. It is evident that for large systems the NBS algorithm outperforms algorithms based on binary search, while binary search-based algorithms outperform direct checking or direct evidence-based algorithms (see Section 5).

In Figure 22 the total detection time divided by the total number of discrete elements $t = T/N$, i.e. the detection time per discrete element for different classes of algorithms is shown. The advantage of the *NBS* contact detection algorithm with t being constant in respect to the size of the problem, is self evident.

APPENDIX

In order to illustrate contact detection capabilities of the proposed algorithm, a completely academic problem comprising half a million circular discs is shown in Figure 23. The problem actually

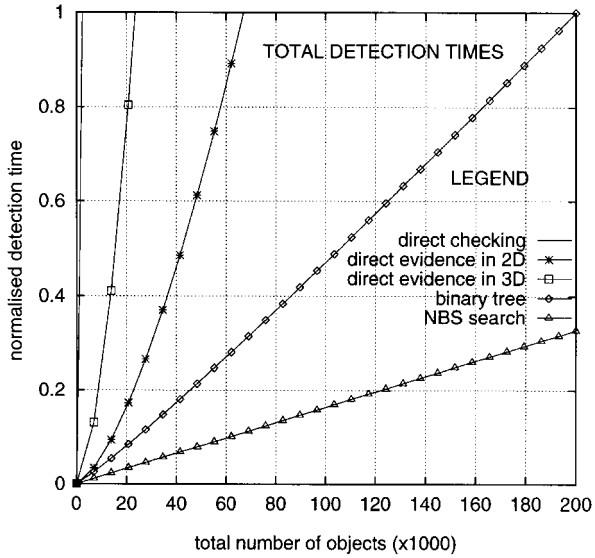

Figure 21. Comparison of contact detection algorithms (total detection time)

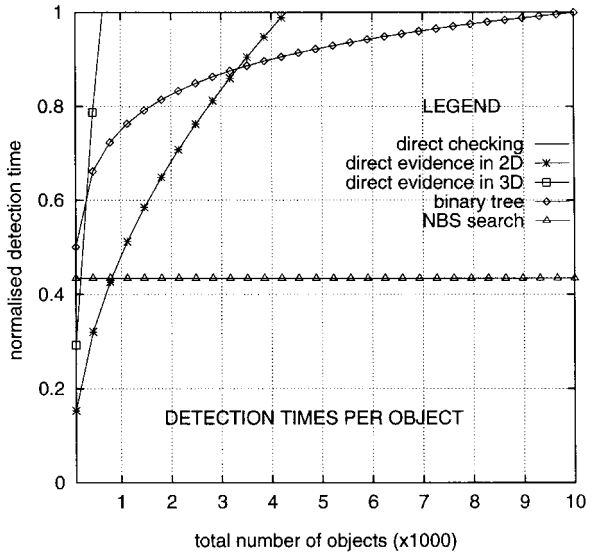

Figure 22. Comparison of contact detection algorithms (total detection time per element)

represents a jet of circular discs. Initially, all the discs move at constant velocity. The contact among discs is assumed to be limited to the normal force only (no friction). At some point the discs start interacting with the rigid boundary and the shape of the jet therefore changes, as shown in Figure 23.

Once again, there is no physical meaning to this problem, it is a purely academic problem chosen to illustrate the ability of the algorithm proposed to detect contacts with minimum resources in

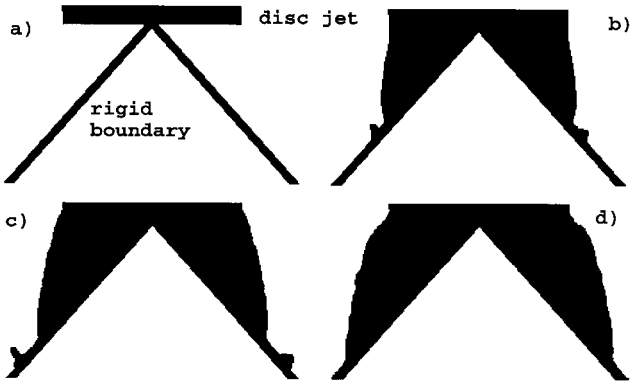

Figure 23. A problem comprising 0.5 million circular discs

terms of both CPU and RAM—it is worth noting that an Indy (32Mb RAM) workstation has been employed for simulation.

REFERENCES

1. J. Lemos, R. D. Hart and P. A. Cundall, 'A generalized distinct element program for modelling jointed rock mass', in O. Stephansson (ed.), *Fundamentals of Rock Joints*, Centek Publishers, Lulea, 1985.
2. J. R. Williams, G. Hocking and G. G. W. Mustoe, 'The theoretical basis of disc methods', *NUMETA '85 Numerical Methods in Engineering*, Theory and applications, Balkema, Rotterdam, 1985.
3. J. R. Williams and G. Mustoe, *Proc. 2nd U.S. Conf. on Discrete Element Methods*, MIT, MA, 1993.
4. G. Mustoe and J. R. Williams, *Proc. 1st U.S. Conf. on Discrete Element Methods*, Golden, Colorado, 1989.
5. A. Munjiza, D. R. J. Owen and N. Bićanić, 'A combined finite-discrete element method in transient dynamics of fracturing solids', *Int. J. Engng. Comput.*, **12,** 145–174 (1995)
6. S. Gen-Hua and R. E. Goodman, 'Discontinuous deformation analysis—a new method for computing stress, strain and sliding of block systems', *Key Questions in Mechanics*, Balkema, Rotterdam, 1988 pp. 381–393.
7. M. Oldenburg and L. Nilsson, 'The position code algorithm for contact searching', *Int. J. Numer. Meth. Engng.*, **37,** 359–386 (1994).
8. J. Bonet and J. Peraire, 'An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems', *Int. J. Numer. Meth. Engng.*, **31,** 1–17 (1991).
9. R. O'Connor, J. Gill and J. R. Williams, 'A linear complexity contact detection algorithm for multy-body simulation', *Proc. 2nd U.S. Conf. on Discrete Element Methods*, MIT, MA (1993).
10. D. S. Preece and S. L. Burchell, 'Variation of spherical element packing angle and its influence on computer simulations of blasting induced rock motion', *Proc. 2nd U.S. Conf. on Discrete Element Methods*, MIT, MA (1993).
11. B. Mirtich, 'Impulse-based dynamic simulation of rigid body systems', *Ph.D. Thesis*, Berkeley, California, 1988.

