INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2006; **66**:46–71
Published online 14 November 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nme.1538

MR linear contact detection algorithm

A. Munjiza*, †, E. Rougier and N. W. M. John

Department of Engineering, Queen Mary, University of London, U.K.

SUMMARY

Large-scale discrete element simulations, as well as a whole range of related problems, involve contact of a large number of separate bodies and an efficient and robust contact detection algorithm is necessary. There has been a number of contact detection algorithms with total detection time proportional to $N \ln(N)$ (where N is the total number of separate bodies) reported in the past. In more recent years algorithms with total CPU time proportional to N have been developed. In this work, a novel contact detection algorithm with total detection time proportional to N is proposed. The performance of the algorithm is not influenced by packing density, while memory requirements are insignificant. The algorithm is applicable to systems comprising bodies of a similar size. The algorithm is named MR (Munjiza–Rougier: Munjiza devised the algorithm, Rougier implemented it). In the second part of the paper the algorithm is extended to particles of different sizes. The new algorithm is called MMR (multi-step MR) algorithm. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: discrete elements; contact; search; linear

1. INTRODUCTION

Discrete element methods [1–4] 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. 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 with the combined finite-discrete element method, where each discrete element is discretized into finite elements.

The optimal contact detection algorithm is in general dependent upon the problem to be solved. The properties of different contact detection algorithms [5–9] make them suitable for different types of problems such as dense packing and loose packing; or quasi-static problems (where relative motion of individual bodies is restricted) and dynamic problems. Real problems,

*Correspondence to: A. Munjiza, Department of Engineering, Queen Mary, University of London, U.K.
†E-mail: a.munjiza@qmul.ac.uk

Received 11 July 2005
Accepted 12 September 2005

where contact detection is a big issue, are dynamic problems, comprising large numbers of discrete elements that are free to move significantly. In this context, general requirements on contact detection algorithms include minimization of the total detection time T defined as total CPU time needed to detect all couples close to each other, minimization of total memory requirements M expressed in terms of total memory size as a function of total number of bodies and packing density, 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 [3]. Most of these algorithms can be classified either as body-based search or space-based search, while procedures applied usually involve binary search [9] with efficiency in terms of total detection time T proportional to

$$T \propto N \log_2 N \tag{1}$$

where N is the total number of discrete elements in the system. In more recent years contact detection algorithms with CPU time proportional to the number of particles

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

have been reported in literature [5, 10]. The advantage of these algorithms is that the CPU time for contact detection is reduced by up to two orders of magnitude for problems comprising millions or billions of particles. These algorithms therefore play an important role in solving grand scale problems and theoretically should enable even systems comprising trillions of particles to be considered.

In this work a linear contact detection algorithm based on spatial sorting is developed. The algorithm is named Munjiza–Rougier (MR). The MR contact detection algorithm comprises a novel so-called MR sort followed by a so-called MR search. Both MR sort and MR search are linear complexity algorithms, thus resulting in linear complexity of the MR contact detection algorithm.

It is worth mentioning that sorting is one of those fundamental problems that offer a variety solutions (quick sort, insertion sort, selection sort, heap sort, merge sort, bin sort, radix sort, bubblesort, etc.) [11, 12]. Each of these solutions has its own comparative advantages and disadvantages. It is well known that the performance of sort algorithms can be significantly improved and linear complexity obtained by exploiting the temporal coherence. This temporal coherence in essence states that sorted lists at two consecutive time intervals should in general not differ much.

In the discrete element method temporal coherence has a special meaning: in the discrete element method contact interaction is processed using the penalty function method, while integration of the governing equation in time is performed using the central difference time integration scheme. This scheme is conditionally stable. The well-known stability criterion [5] states that the time step h must be smaller than T/π, where T is the period of the highest frequency mode. This stability criterion simply means that in a single time step no discrete element moves more than the amplitude of the smallest overlap between any two discrete elements in contact. Temporal coherence of discrete element systems is therefore often exploited to improve CPU performance of contact detection algorithms [13, 14].

As mentioned above, unlike other sorting algorithms used elsewhere [12], in this work a novel MR-linear sort is used. The MR-linear sort algorithm has a unique advantage of

exploiting deterministic temporal coherence of discrete element systems as explained above, thus producing high CPU performance (CPU time proportional to N—worst-case scenario).

The MR contact detection algorithm has performance in terms of both RAM and CPU similar to the NBS algorithm, however it is completely insensitive to spatial distribution of particles, thus performing well for both very dense and extremely loose packs of particles. In the rest of the paper, detailed description of the MR contact detection algorithm is provided together with theoretical analysis of performance and numerical experiments demonstrating this performance. As the CPU performance of the MR contact detection algorithm decreases as the ratio between the diameter of the largest and smallest discrete element increases, a possible extension of the algorithm to systems comprising particles of different size in the form of so-called MMR (multi-step MR) algorithm is also proposed in the paper.

2. COMPONENTS OF MR CONTACT DETECTION ALGORITHM

Bounding box: MR contact detection algorithm is based on the assumption that all the discrete elements in the system can be approximated by a sphere; in other words, spherical bounding box is used (Figure 1). For all discrete elements a bounding sphere of the same diameter is used. The diameter of the sphere, d, is obtained from the size of the largest discrete element in the system.

In addition it is assumed that all discrete elements are contained within a finite space of cubical shape (Figure 1). The space boundaries are taken big enough so that all particles are at all times within these boundaries.

Space decomposition: The entire space is subdivided into identical cubical cells of size d (where d is the diameter of the spherical bounding box as explained above) as shown in Figure 2. Each cell has an identification triplet of integer numbers (i_x, i_y, i_z), where

$$\begin{aligned} i_x &= 0, 1, 2, \ldots, n_x - 1 \\ i_y &= 0, 1, 2, \ldots, n_y - 1 \\ i_z &= 0, 1, 2, \ldots, n_z - 1 \end{aligned} \tag{3}$$

and

$$n_x = \frac{x_{\max} - x_{\min}}{d} + 1, \quad n_y = \frac{y_{\max} - y_{\min}}{d} + 1, \quad n_z = \frac{z_{\max} - z_{\min}}{d} + 1 \tag{4}$$

while n_x, n_y and n_z are the total number of cells in x-, y- and z-directions, respectively.

Mapping of spherical bounding boxes onto cells using MR sorting algorithm: The mapping from the set of spherical bounding boxes onto the set of cells is defined in such a way that each bounding box (discrete element) is assigned to one and only one cell. The formulae for

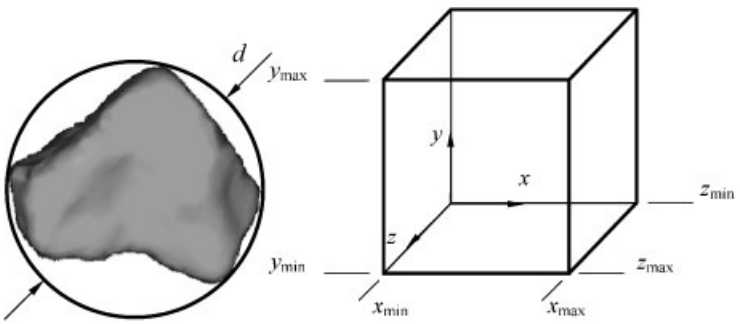

Figure 1. Spherical bounding box (left) and space boundaries (right).

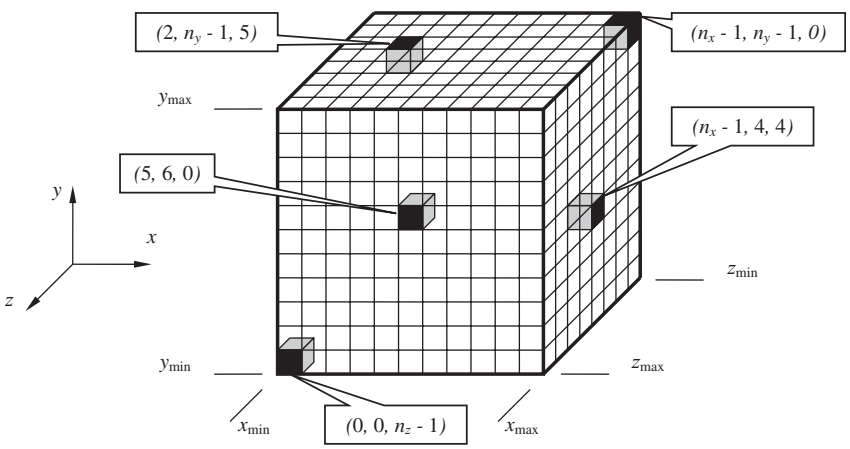

Figure 2. Space decomposition.

the mapping are as follows:

$$i_x = \text{Int}\left(\frac{x - x_{\min}}{d}\right)$$
$$i_y = \text{Int}\left(\frac{y - y_{\min}}{d}\right) \tag{5}$$
$$i_z = \text{Int}\left(\frac{z - z_{\min}}{d}\right)$$

where x, y and z are current co-ordinates of the centre of the bounding boxes, while i_x, i_y and i_z are the respective integerized co-ordinates and Int is the operation of casting a real number into an integer number.

In Figure 3, an example of the mapping is given. For the sake of clarity, the projections of the mappings into planes x–y, y–z and x–z are given instead of the 3D figure of the system. In

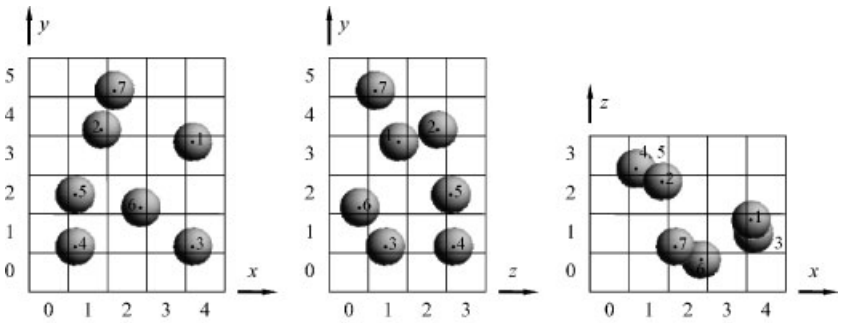

Figure 3. Mapping of discrete elements onto cells.

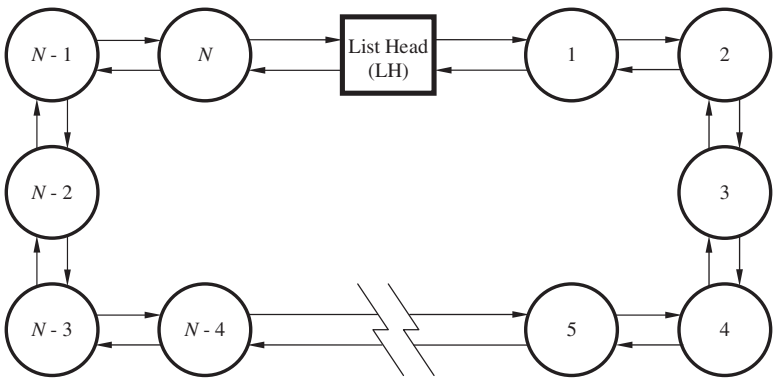

Figure 4. Double connected closed list of bounding boxes. Each bounding box is identified by its number.

this example spherical bounding box 1 is mapped onto cell (4, 3, 1), while spherical bounding box 3 is mapped onto cell (4, 1, 1) and spherical bounding box 7 is mapped onto cell (2, 4, 1). The most obvious way of representing the mapping as described before is by representing cells by a 3D array of size

$$n = n_x \cdot n_y \cdot n_z \tag{6}$$

Such an array would automatically give spatial meaning to the mapping. However, the array would use a large amount of RAM space especially in the cases where most of the cells have no bounding boxes mapped to them, i.e. when discrete elements are far from each other (loose packing). Thus, performance in terms of RAM requirements would be very poor. Thus, in this work mapping of bounding boxes onto cells is represented by a double connected closed linked list of bounding boxes. In C or C++ implementation each bounding box is represented by a structure or a class pointing to the next and previous bounding box as shown in Figure 4.

In order for the list to represent spatial distribution of bounding boxes as shown in Figure 3, it is necessary to sort the list according to a criterion that has spatial meaning. The ordering

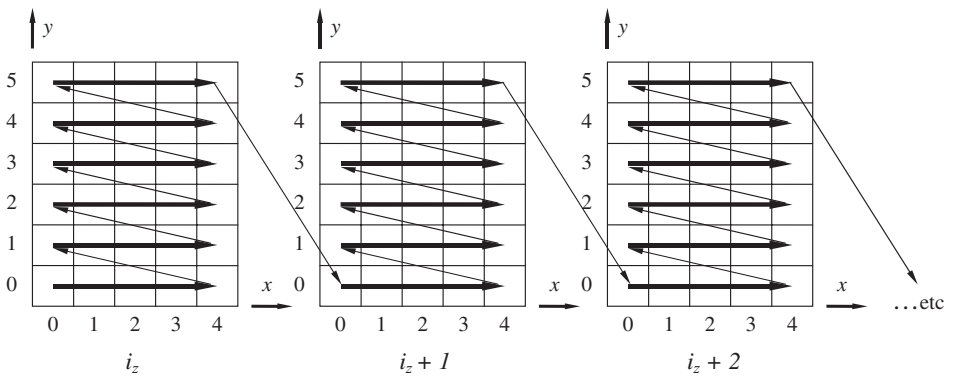

Figure 5. Visual representation of cells ordered according to the ordering criterion (7).

criterion adopted in this work states that a bounding box i is greater than a bounding box j if

$$\{[(i_{zi} > i_{zj})] \text{ or } [(i_{zi} = i_{zj}) \text{ and } (i_{yj} > i_{yj})] \text{ or } [(i_{zi} = i_{zj}) \text{ and } (i_{yi} = i_{yj}) \text{ and } (i_{xi} > i_{xj})]\} \quad (7)$$

where i_{xi}, i_{yi} and i_{zi} are the integerized co-ordinates for bounding box i and i_{xj}, i_{yj} and i_{zj} are the integerized co-ordinates for bounding box j. This criterion implies that the linked list must be sorted relative to the integerized i_z, i_y and i_x co-ordinates. In the case where the integerized co-ordinate i_z is the same for both bounding boxes, sorting relative to the integerized co-ordinate i_y is performed, and in the case where the integerized co-ordinate i_y is the same for both bounding boxes, sorting relative to the integerized co-ordinate i_x is performed. The sorting is performed in an ascending way. The result obtained if this criterion is applied to the ordering of cells in the space is visually illustrated in Figure 5.

Two sorting procedures are used

- Quick sort, which is readily available in the scientific literature. It has CPU time requirements proportional to $N \ln N$.
- MR-linear sort, which is a novel sorting algorithm introduced in this paper. It has CPU time requirements proportional to N and is much more efficient than quick sort.

Contact detection using MR search algorithm: Once spatial meaning to the double connected list of bounding boxes is obtained as described above, MR-linear search is employed to detect actual contacts. MR-linear search is a novel search algorithm introduced in this paper.

3. QUICK SORT ALGORITHM

Quick sort is usually implemented using arrays. Here, a double linked list is used and sorting is done by swapping bounding boxes inside a list (Figure 6).

In the figure LH is the list head that contains all the relevant information about the space boundaries. Each list is characterized by the so-called: (a) minimum bounding box, i.e. the

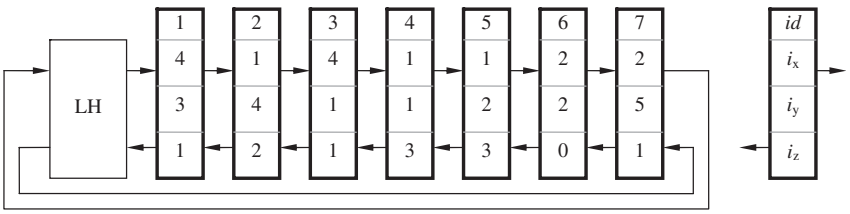

Figure 6. Initial double connected closed list.

bounding box m that has the minimum co-ordinates according to the sorting criterion (7)

$$m_x = i_{x\ \min},\quad m_y = i_{y\ \min},\quad m_z = i_{z\ \min} \tag{8}$$

(b) The maximum bounding box M, with maximum co-ordinates according to the ordering criterion (7)

$$M_x = i_{x\ \max},\quad M_y = i_{y\ \max},\quad M_z = i_{z\ \max} \tag{9}$$

(c) The middle bounding box, which is the bounding box that is half a way (in terms of number of cells), between the maximum and minimum bounding boxes. In order to obtain the middle bounding box the three integerized co-ordinates are assumed to be equivalent to a three-digit number:

$$\begin{aligned} m &= m_z m_y m_x \\ M &= M_z M_y M_x \end{aligned} \tag{10}$$

where m represents the minimum bounding box while M represents the maximum bounding box. m_z and M_z are the most significant digits, and m_x and M_x are the least-significant digits. The limits of the digits are

$$\begin{aligned} &0 \leqslant m_x < n_x, \quad 0 \leqslant M_x < n_x \\ &0 \leqslant m_y < n_y, \quad 0 \leqslant M_y < n_y \\ &0 \leqslant m_z < n_z, \quad 0 \leqslant M_z < n_z \end{aligned} \tag{11}$$

The middle point is given by

$$\mu = \frac{(m + M)}{2} \tag{12}$$

In order to obtain μ, first the sum $s = (m + M)$ is calculated as follows:

$$\begin{aligned} s_x &= (m_x + M_x) \ (\mathrm{mod}\, n_x) \\ s_y &= \left((m_y + M_y) + \frac{(m_x + M_x)}{n_x} \right) \ (\mathrm{mod}\, n_y) \\ s_z &= (m_z + M_z) + \left((m_y + M_y) + \frac{(m_x + M_x)}{n_x} \right) \Big/ n_y \end{aligned} \tag{13}$$

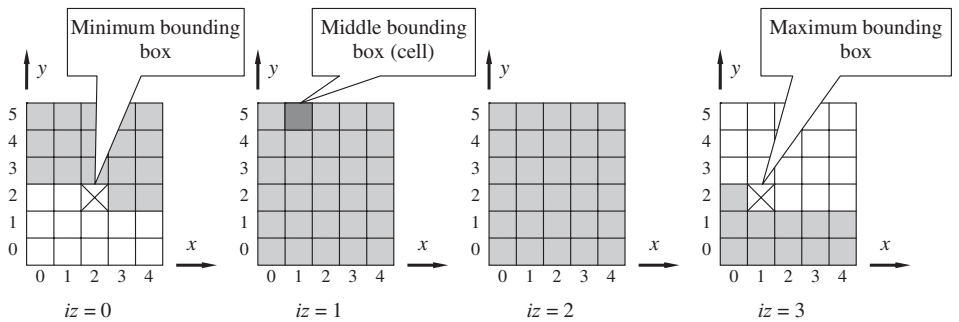

Figure 7. Finding the middle cell between maximum and minimum bounding boxes.

The division by 2 produces the middle point

$$\begin{aligned}\mu_z &= \frac{s_z}{2}\\ \mu_y &= \frac{s_y + (s_z \,(\mathrm{mod}\, 2))n_y}{2}\\ \mu_x &= \frac{s_x + [(s_y + (s_z \,(\mathrm{mod}\, 2))n_y) \,(\mathrm{mod}\, 2)]n_x}{2}\end{aligned} \tag{14}$$

In the above equations $(\mathrm{mod}\, n_x)$, $(\mathrm{mod}\, n_y)$ and $(\mathrm{mod}\, 2)$ return the remainder of the division by n_x, n_y and 2, respectively. In the example shown in Figure 7, the co-ordinates of the middle cell are as follows:

$$\begin{aligned} m &= 022\\ M &= 321\\ n_x &= 5\\ n_y &= 6\end{aligned} \tag{15}$$

the sum of m and M is given by

$$\begin{aligned} s_x &= (2+1)\,(\mathrm{mod}\, 5) = 3\\ s_y &= \left((2+2) + \frac{(2+1)}{5}\right)\,(\mathrm{mod}\, 6) = 4\,(\mathrm{mod}\, 6) = 4\\ s_z &= (0+3) + \left((2+2) + \frac{(2+1)}{5}\right) \Big/ 6 = 3\end{aligned} \tag{16}$$

the co-ordinates of the middle bounding box are

$$\mu_z = \frac{3}{2} = 1$$
$$\mu_y = \frac{4 + (3 \,(\mathrm{mod}\, 2))6}{2} = \frac{10}{2} = 5 \tag{17}$$
$$\mu_x = \frac{3 + [(4 + (3 \,(\mathrm{mod}\, 2))6) \,(\mathrm{mod}\, 2)]5}{2} = \frac{3}{2} = 1$$

The cells between $(2, 2, 0)$ and $(1, 2, 3)$ are shaded in Figure 7. There are 88 cells in between the minimum and maximum bounding boxes, so the middle cell is the 44th one starting from the minimum bounding boxes, which have the same co-ordinates as the middle bounding box calculated as explained above.

4. MR-LINEAR SORT ALGORITHM

It is evident that when using the quick sort algorithm the CPU time for sorting the linked list is proportional to $N \ln N$. In order to obtain a CPU time proportional to N in this work a novel linear sort algorithm is proposed. This algorithm is termed the MR-linear sort algorithm. It can be used to sort either lists or arrays. The MR-linear sort makes use of temporal coherence. It is well known that the performance of sort algorithms can be significantly improved and linear complexity obtained by exploiting the temporal coherence [12]. Temporal coherence in essence states that statistically sorted lists at two consecutive time intervals should in general not differ much. In the discrete element method temporal coherence has a special meaning: in the discrete element method contact interaction is processed using the penalty function method, while integration of the governing equation in time is performed using the central difference time integration scheme. This scheme is conditionally stable. The well-known stability criterion [5] states that the time step h should be smaller than T/π, where T is the period of the highest frequency mode. This stability criterion simply means that in a single time step no discrete element moves more than the amplitude of the smallest overlap between any two discrete elements in contact. As the overlaps are controlled by the penalty term, they are by default small. Thus, the following assumption is always valid: *No discrete element can move more than the size of a single cell in any given time step* (*otherwise discrete elements would be on top of each other*, *i.e. penetration would be extremely large*). In other words, temporal coherence in the discrete element method has a special meaning [14]. Unlike other sorting algorithms used elsewhere [12], the MR-linear sort algorithm has a unique advantage of exploiting this property of discrete element systems, thus producing high CPU performance as explained in the rest of this section.

The MR-linear sort algorithm is a 3D algorithm. It can easily be extended to any number of dimensions. For the sake of clarity of explanation, it is explained using 2D geometry. However, it is implemented in 3D and the numerical results shown in this paper refer to 3D simulations.

According to the above assumption a discrete element located in the cell (i_x, i_y, i_z) at time t can be located at time $t + h$ (where h is the time step) in any of the cells shown in Figure 8.

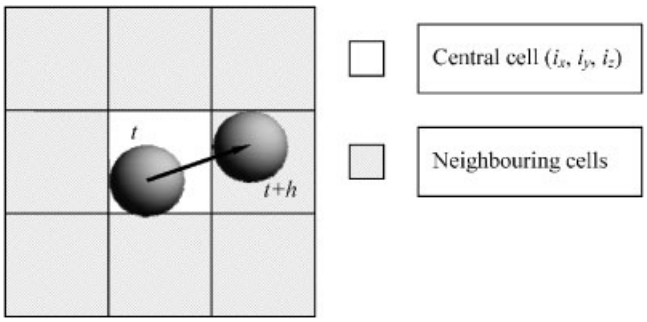

Figure 8. Relocation of a discrete during a single time step: the discrete element has moved from central into one of the neighbouring cells.

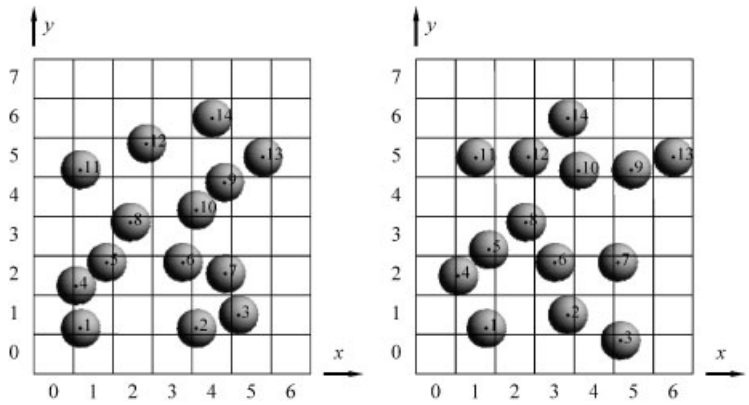

Figure 9. Snapshots of the system at time: (a) t; and (b) $t + h$.

An example of a whole system at time t and $t + h$ is shown in Figure 9 and is used in further text to explain the MR-linear sorting algorithm.

The double connected closed linked lists of bounding boxes at time t is shown in Figure 10.

It is evident that the list is sorted. The same list at time $t + h$ is shown in Figure 11. The only difference between the list shown in Figure 10 and the list shown in Figure 11 is that the integerized co-ordinates for the bounding boxes in Figure 11 are different from the integerized co-ordinates for the bounding boxes in Figure 10. This is because the discrete elements have moved. It is evident that the list shown in Figure 11 is therefore not sorted.

However, the list is nearly sorted. The MR-linear sort algorithm takes advantage of this fact. In order to sort the list shown in Figure 11 with MR-linear sort algorithm, the list is parsed starting from the LH, which has integerized co-ordinates $(0, 0)$, and each bounding box is checked against its previous one. If the bounding box is greater than its previous one, then no change is needed and the parsing of the list continues until a bounding box A that is smaller than its previous one is found (Figure 12). When this happens, the bounding A is relocated into the appropriate place in the list.

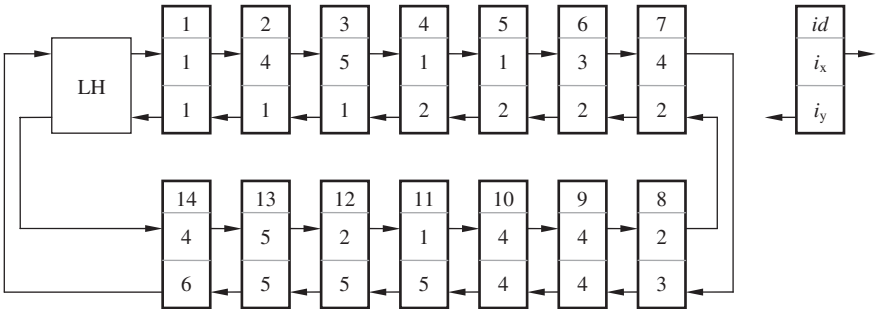

Figure 10. Double connected closed list of bounding boxes at time t: note that the list is sorted.

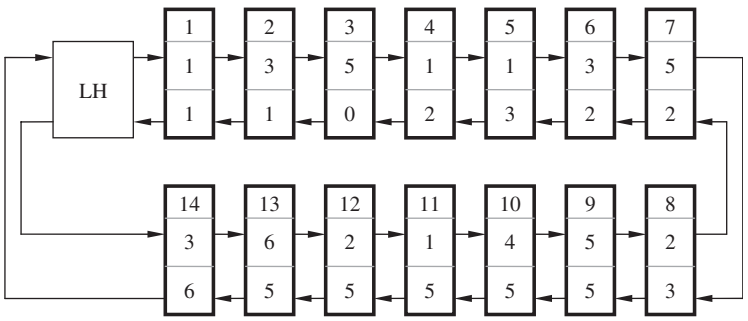

Figure 11. Double connected closed list of bounding boxes at time $t + h$: note that the list is not sorted.

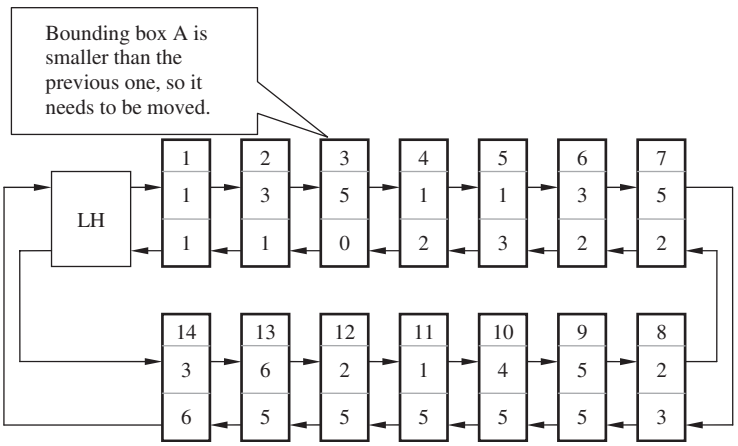

Figure 12. Relocation needed for bounding box 3.

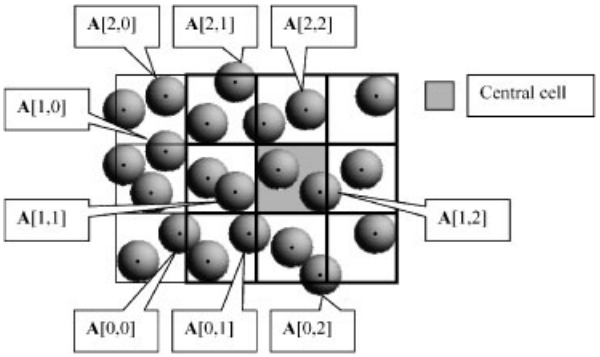

Figure 13. Relationship between matrix **A** and bounding boxes in neighbouring cells.

It is worth noting that for a fully sorted list, the expression

$$(b_{\text{box}})_{t+\Delta t} \geqslant (b_{\text{box}} \rightarrow \text{prev})_{t+\Delta t} \tag{18}$$

is always true for all the bounding boxes in the list, where b_{box} is a current bounding box in the list and $b_{\text{box}} \rightarrow \text{prev}$ is the bounding box immediately before the current bounding box b_{box}. However, this is not always true when the list is 'nearly' sorted, as it was explained above and due to the assumption about motion of discrete elements within a single time step, it can happen that

$$(b_{\text{box}})_{t+\Delta t} < (b_{\text{box}} \rightarrow \text{prev})_{t+\Delta t} \tag{19}$$

in which case

$$(i_{xt} - 1; i_{yt} - 1) < (b_{\text{box}} \rightarrow \text{prev})_{t+\Delta t} \leqslant (i_{xt} + 1; i_{yt} + 1) \tag{20}$$

where i_{xt}, i_{yt} are the integerized co-ordinates of the bounding box b_{box} at time t. Thus, in order to be able to place the bounding box A inside the list, a 3×3 matrix, **A**, of pointers to the bounding boxes immediately before each of the neighbouring cells of the current bounding box is used. The correspondence between the contents of the matrix **A** and the neighbouring cells of a particular bounding box is shown in Figure 13.

These pointers are advanced as the list is parsed. As each pointer is only advanced forward, the cumulative effect of advancing the pointers is equivalent to parsing the whole list once. Thus the theoretical CPU time for MR-linear sort algorithm is therefore given by

$$T \propto N \tag{21}$$

5. MR-LINEAR SEARCH ALGORITHM

Detection of contact is done by checking all the discrete elements mapped to a particular cell against all the discrete elements in neighbouring cells, because only discrete elements from neighbouring cells can touch each other. To avoid repetition in the contact detection process, a contact mask is used for each central cell (Figure 14).

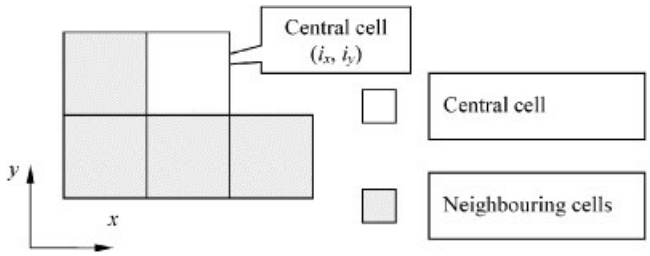

Figure 14. Contact mask in 2D.

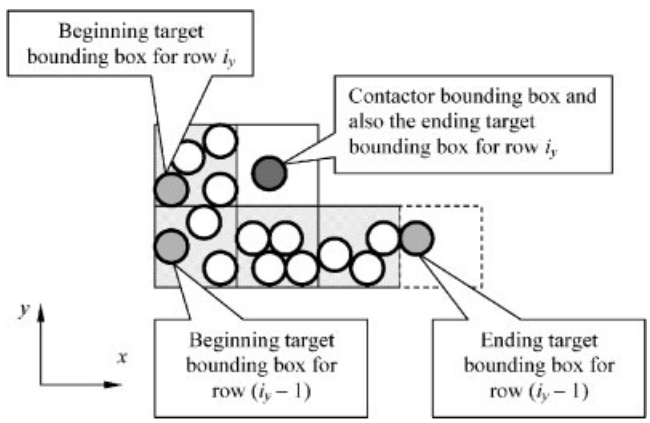

Figure 15. Contact detection in 2D.

The contact mask is divided into two rows. Since the bounding boxes in the list are ordered it is a fact that for each row there will be a beginning target bounding box and an ending target bounding box, as it is shown in Figure 15.

The solution to the contact detection problem is to find which bounding boxes are the beginning and ending bounding boxes for each of the rows of the contact mask. The beginning bounding box for a particular row is found by parsing the list starting from the LH until a bounding box that is greater or equal to the first cell in the row is found. In a similar way, the ending bounding box is found by parsing the list starting from the beginning bounding box until a bounding box that is greater than the last cell in the row is found.

Once this stage is completed, the contact can be calculated between the contactor bounding box and all target bounding boxes that are greater or equal to the beginning bounding box of the row and that are smaller than the ending bounding box of the row:

$$\text{beginning} \leqslant \text{target} < \text{ending} \tag{22}$$

Thus contact search is reduced to parsing between the beginning and the ending bounding box. In the case that there is no bounding box in one of the rows, the beginning and the ending bounding boxes will be the same (Figure 16).

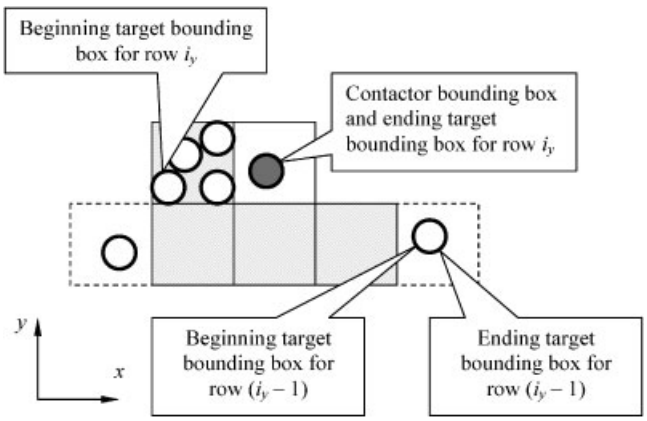

Figure 16. Empty row in the contact mask.

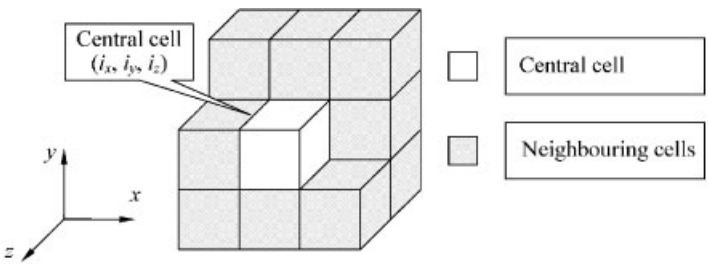

Figure 17. Contact mask in 3D.

In 3D space, the situation is similar, except that in this case instead of two rows in the contact mask there are five rows, as it is shown in Figure 17.

There are a total of 13 neighbouring cells distributed into 5 different rows. Each row has a beginning cell and an ending cell as shown in Figure 18. Since the bounding boxes in the list are ordered, the beginning and ending target bounding boxes for each of the central cells are advanced with the advancing of the central cell. The central cell changes as the list is parsed and so do the beginning and ending bounding boxes for each row of the neighbouring cells. The parsing is done only once and is done by simply moving from the previous to the next bounding box in the list. Thus, the theoretical total CPU time for the MR-linear search is proportional to the size of the list, which is equal to the total number of discrete elements, i.e. the total CPU time for search is given by

$$T \propto N \tag{23}$$

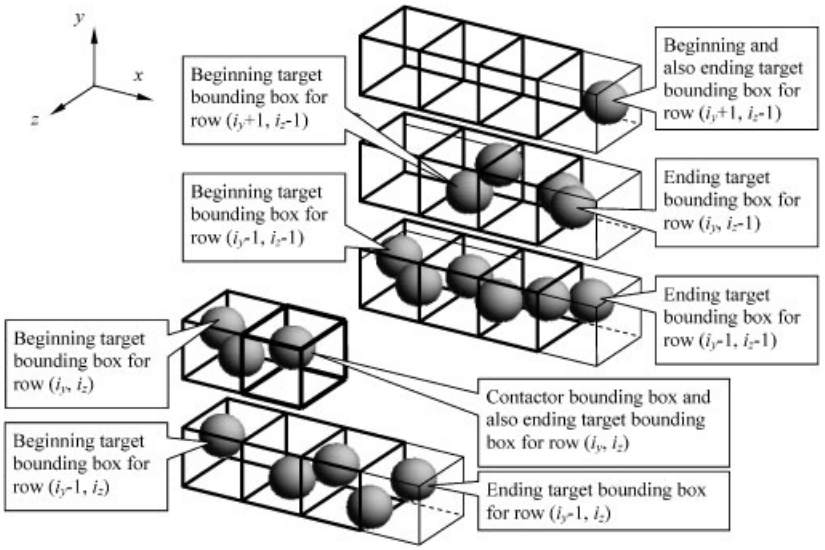

Figure 18. Beginning and ending bounding boxes for contact mask in 3D.

6. NUMERICAL EXPERIMENTS

The MR-linear contact detection algorithm performs each time step MR-linear sort algorithm followed by the MR-linear search algorithm. Thus, the theoretical total CPU time to detect all contacting couples is given by

$$T \propto N \tag{24}$$

In other words, MR-linear contact detection algorithm belongs to the category of linear contact detection algorithms [5]. The memory requirements of MR-linear contact detection algorithm are given by

$$\begin{aligned}
&\text{in 2D}: && M = 4N \text{ integer numbers} \\
&&& \text{(two co-ordinates plus two pointers per discrete element)} \\
&\text{in 3D}: && M = 5N \text{ integer numbers} \\
&&& \text{(three co-ordinates plus two pointers per discrete element)} \\
&\text{in } n\text{-dimensional space}: && M = (2+n)N \text{ integer numbers} \\
&&& (n) \text{ co-ordinates plus two pointers per discrete element)}
\end{aligned} \tag{25}$$

It is also possible to implement both MR sort and MR search without remembering the integerized co-ordinates in which case Int operation is performed when integerized co-ordinate is needed. The result is an increase in CPU requirements, however RAM requirements are

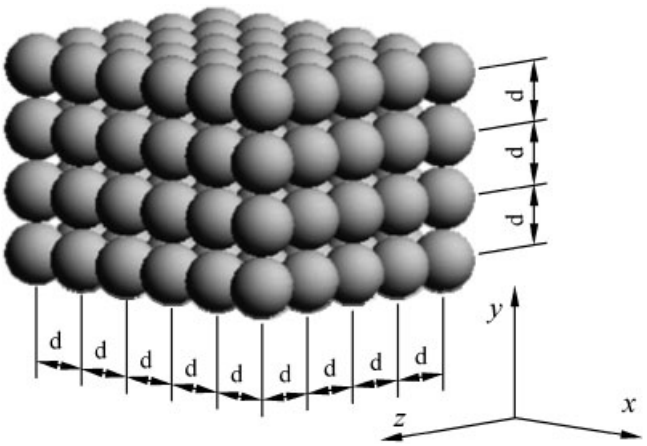

Figure 19. Packing A.

reduced to

$$\begin{aligned} &\text{in 2D:} && M = 2N \text{ integer numbers} \\ &&& \text{(two pointers per discrete element)} \\ &\text{in 3D:} && M = 3N \text{ integer numbers} \\ &&& \text{(two pointers per discrete element)} \\ &\text{in } n\text{-dimensional space:} && M = 2N \text{ integer numbers} \\ &&& \text{(two pointers per discrete element)} \end{aligned} \tag{26}$$

Another important property of both MR search and MR sort is that neither memory nor CPU requirements are a function of spatial distribution of discrete elements; thus, MR contact detection algorithm performs well for both loose and dense packs. In order to demonstrate the above properties of MR-linear contact detection algorithm a set of numerical experiments is presented. The experiments are chosen is such a way that they can be easily reproduced by other researchers in the field. The experiments are as follows:

Example I

Example I consists of N discrete elements of diameter d spaced at distance d in x-, y- and z-directions (Figure 19). Contact detection is solved 10 times for the problem and each time all contacting couples are detected and the total CPU time for contact detection is measured for different values of N. All the results are obtained on a Dimension 4400 PC (Intel 2.0 GHz with 768 Mb RAM space).

The cumulative CPU times for all 10 contact detections as a function of the total number of discrete elements comprising the problem are shown in Figure 20. The results shown are obtained by changing N from $N = 8000$ to 5 000 211. It is worth noting that the total CPU time is proportional to N. This is because both search and sort algorithms have linear CPU time requirements.

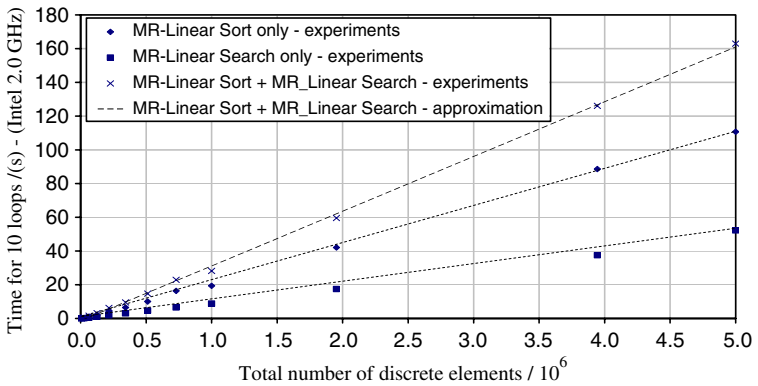

Figure 20. CPU time as a function of the number of objects. Total contact detection CPU time is made of CPU time for MR sort to which CPU time for MR search is added. This is because MR contact detection algorithm comprises two steps: MR sort followed by MR search.

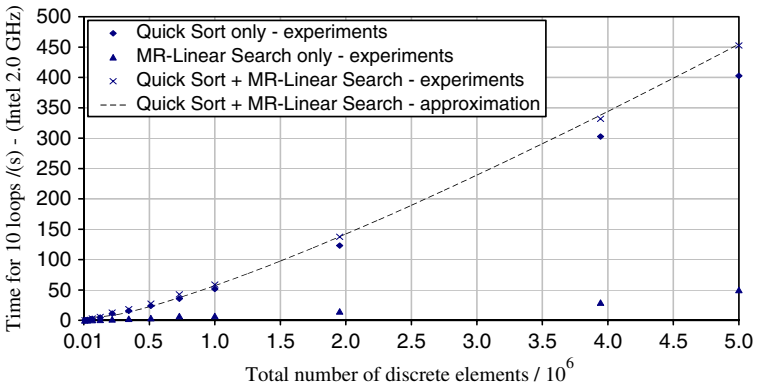

Figure 21. CPU time as a function of the number of objects. Quick sort plus MR search.

Should one use quick sort instead of MR sort, the total CPU time would no longer be linear as it is shown in Figure 21. It is evident that the performance would deteriorate. Furthermore, should one use binary search instead of linear MR search, then further deterioration of performance would occur as shown in Figure 22.

Example II

Example II consists of $N = 125\,000$ discrete elements diameter d spaced at distance $2d$ in y- and z-directions and at variable distance s (spacing) in x-direction (packing B) (Figure 23).

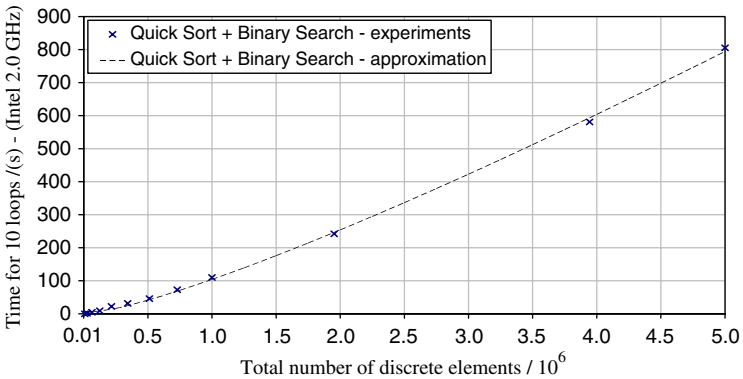

Figure 22. CPU time as a function of the number of objects. Quick sort plus binary search.

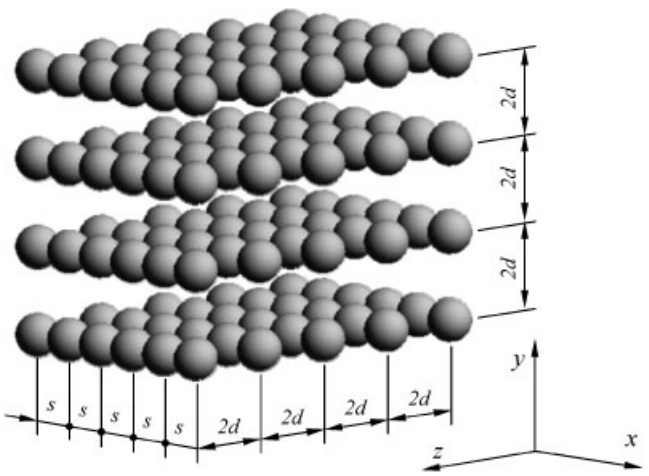

Figure 23. Packing B.

The packing density ρ changes with spacing s as

$$\rho \propto \frac{1}{s} \tag{27}$$

Thus, the packing density ρ is changed by changing spacing s. In Figure 24, the cumulative CPU time for 10 repeated detections of all contacts is shown as a function of the spacing. The results obtained in this numerical experiment demonstrate that the total detection time does not depend on packing density (Figures 25–26).

Example III

Example III consists of $N = 125\,000$ discrete elements of diameter d spaced at distance s in x- and y-directions and at distance $2d$ in z-direction (Figure 27). The density changes with

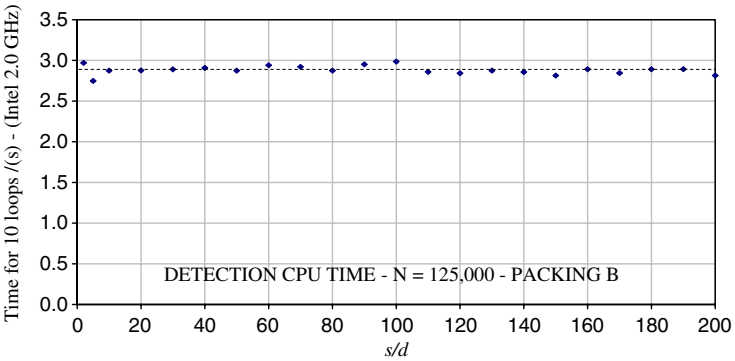

Figure 24. CPU time as function of packing density (packing B)—MR-linear sort plus MR search.

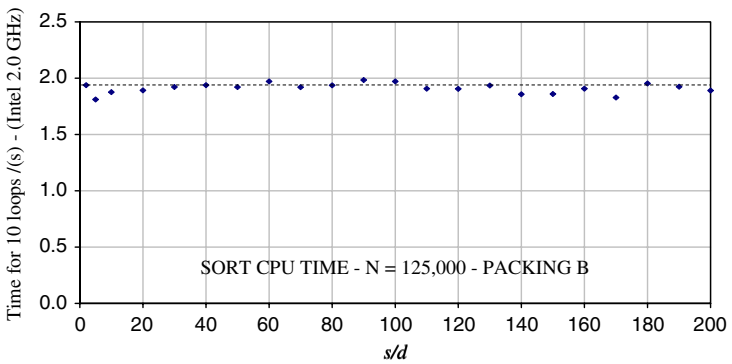

Figure 25. CPU time as function of packing density (packing B)—MR-linear sort only.

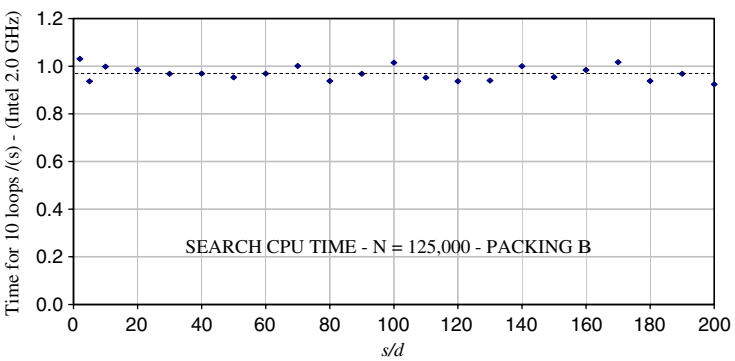

Figure 26. CPU time as function of packing density (packing B)—MR-linear search only.

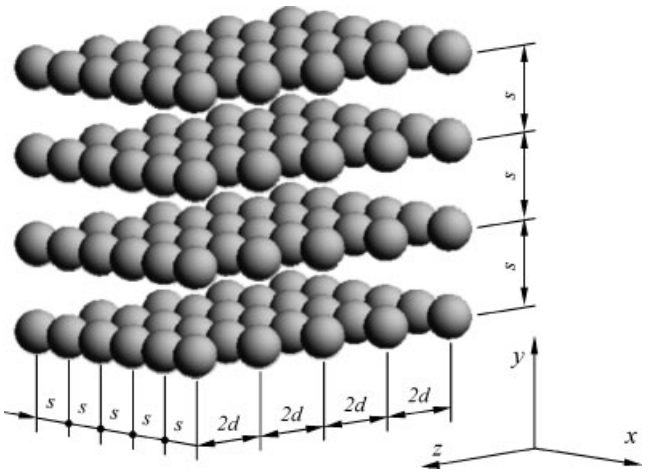

Figure 27. Packing C.

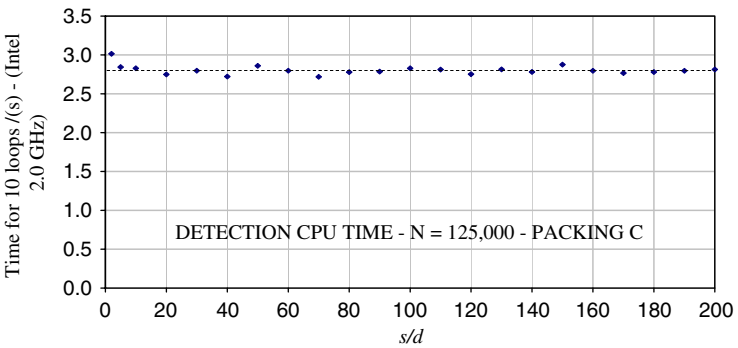

Figure 28. CPU time as function of packing density (packing C)—MR-linear sort plus MR-linear search.

spacing s as

$$\rho \propto \frac{1}{s^2} \tag{28}$$

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 28. The results obtained show that total detection time is not dependent on packing density, relative change of which goes from 1 to $1/(200 \times 200) = 1/40\,000$.

Example IV

Example IV consists of $N = 125\,000$ discrete elements of diameter d spaced at distance s in x-, y- and z-directions (Figure 29). The density changes with spacing s as

$$\rho \propto \frac{1}{s^3} \tag{29}$$

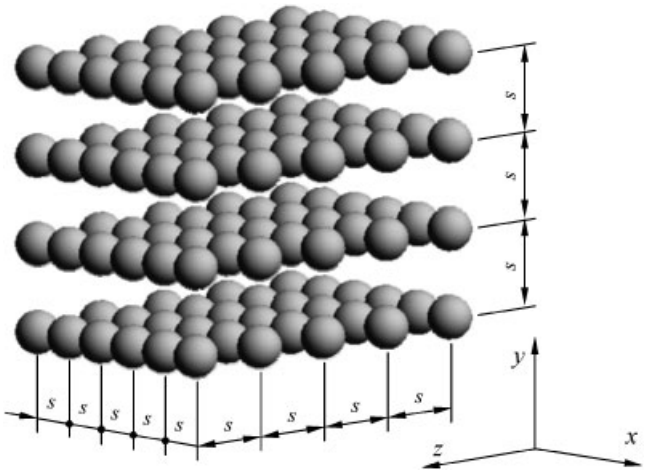

Figure 29. Packing D.

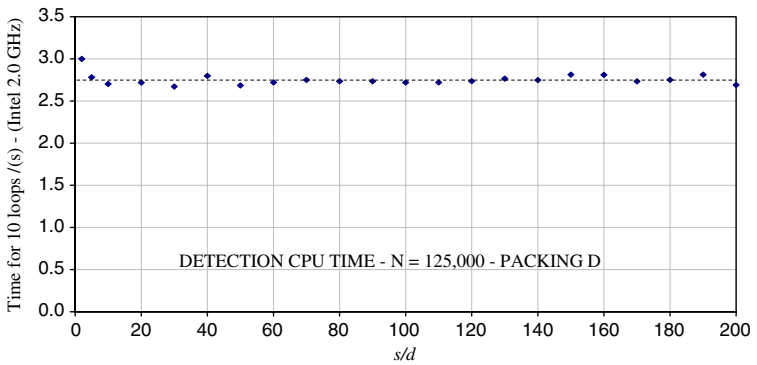

Figure 30. CPU time as function of packing density (packing D)—MR-linear sort plus MR-linear search.

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 30. The results obtained show that total detection time is not dependent on packing density, relative change of which goes from 1 to $1/(200 \times 200 \times 200) = 1/8\,000\,000$.

It is worth mentioning that in all the examples shown above, the RAM space required is proportional to N and is completely independent of the packing pattern or packing density.

7. MULTISTEP MR ALGORITHM BASED ON SPHERICAL BOUNDING BOXES

The major drawback of the MR algorithm is the fact that all discrete elements are represented with bounding boxes of same size. In addition the diameter of the bounding box is such that

the largest discrete element comprising the system is contained within a bounding box. The result of this is that the contacts are over-reported, i.e. the MR algorithm reports a large number of contacts between small particles although these particles may be relatively far from each other and therefore not in contact. Due to this the performance of the algorithm deteriorates with the increase in ratio between the size of the largest and smallest particle.

Description of the multi-step MR algorithm (*MMR*): The basic idea of the MMR is relatively simple and involves dividing particles into groups according to size

$$\begin{aligned}
&\textit{Group } 0\text{: Particles represented by bounding box of size } d_0 = d\\
&\textit{Group } 1\text{: Particles represented by bounding box of size } d_1 = d/\alpha,\ \ \alpha>1\\
&\textit{Group } 2\text{: Particles represented by bounding box of size } d_2 = d/\alpha^2,\ \alpha>1\\
&\textit{Group } 3\text{: Particles represented by bounding box of size } d_3 = d/\alpha^3,\ \alpha>1\\
&\qquad\ldots\\
&\textit{Group } n\text{: Particles represented by bounding box of size } d_n = d/\alpha^n,\ \alpha>1
\end{aligned} \tag{30}$$

MMR is implemented in n steps:

Step 0: Initially all the particles are represented by the bounding box associated with group 0. All the bounding boxes are mapped onto the cells. It is worth mentioning that the size of the cells is equal to the size of the bounding box, i.e. d_0. Contact detection is performed using the MR procedures in the usual way. However, separate lists are assembled for particles from group 0 and all other particles and contact search is performed for contact between particles from group 0 (contactors) and particles from all other groups including group 0 (targets). Thus no contacts between particles from say group 1 to particles of group 1, 2 or 3 will be reported at this stage. In a similar way no contact between particles from group n to particles from group n are reported at this stage. Only contacts between very big particles to all other particles are reported at this stage.

Step 1: Particles of group 0 are first removed from the system. All the remaining particles are then represented by the bounding box of size d_1 and contact detection is performed in the usual way, except that a separate list for particles of group 1 is kept and a search only performed for contacts between particles from group 1 and all other remaining particles. Thus this step will report contacts between particles from group 1 and particles of all other groups that are smaller than the particles from group 1.

Step 3: In step 3, all the particles from group 1 are eliminated from the system in addition to the particles from group 0, which have already been eliminated during step 1. All the remaining particles are represented by the bounding box of size d_2 and contact of all particles of group 2 to all the smaller particles (group 3, 4, etc.) are detected.

Step 4: In step 4, all the particles up to and including particles from group 3 are removed from the system and using usual procedures contacts of particles from group 4 to all the other particles still in the system are detected.

Step n: By analogy steps 5, 6, etc. are performed. During execution of step n only particles from group n remain in the system. At this stage contact detection will detect all the contacts between these particles.

Polydispersity: The key characteristic of the multi-step algorithm described above is that it uses MR algorithm to detect contacts between particles of different sizes. The ratio between the smallest and the largest particle depends on the total number of steps n and parameter α and is given by

$$\frac{d_{\max}}{d_{\min}} = \alpha^n \tag{31}$$

For, say,

$$\alpha = 2 \tag{32}$$

in 10 steps the ratio between the largest and smallest particle in the system is

$$\frac{d_{\max}}{d_{\min}} = 2^{10} = 1024 \tag{33}$$

If the number of steps increases from 10 to 20 (i.e. doubles), the ratio between the largest and smallest particle in the system increases by over 1024 times to over 1000 000, which would for example allow the system in which the largest particle is 1 m in diameter and the smallest particle is 1 μm. Tripling the number of steps would reduce the smallest particle to a size similar to that of a single atom.

CPU performance: It is worth noting that in each step of the multi-step algorithm MR algorithm is in essence repeated. Thus in the worst-case scenario doubling the number of steps means doubling the CPU time. Thus, the total CPU time for multistep algorithm is given by

$$T < nT_{\mathrm{MR}} \tag{34}$$

where T_{MR} is the total CPU time for MR contact detection for a system comprising the same number of particles but all of the same size. In real application, the CPU performance is dependent on the size distribution and is always better than the one given by (34). This is because only in the step 0 are all particles considered, while in subsequent steps the number of particles reduces. In addition, there is no need to repeat space mapping in each time step and in each time step only detection of contacts with particles from the group of the largest particles is performed, thus making each step much faster than the original MR implementation.

RAM requirements: RAM requirements of multi-step algorithm implemented for particles of different sizes as described above are identical to RAM requirements of MR contact detection algorithms when implemented for particles of same size. For instance, it is possible to implement multistep algorithm using MR search and MR sort without remembering the integerized co-ordinates—RAM requirements in such a case are given by

$$\begin{aligned}
\text{in 2D}: &\quad M = 2N \text{ integer numbers} \\
&\quad \text{(two pointers per discrete element)} \\
\text{in 3D}: &\quad M = 2N \text{ integer numbers} \\
&\quad \text{(two pointers per discrete element)} \\
\text{in } n\text{-dimensional space}: &\quad M = 2N \text{ integer numbers} \\
&\quad \text{(two pointers per discrete element)}
\end{aligned} \tag{35}$$

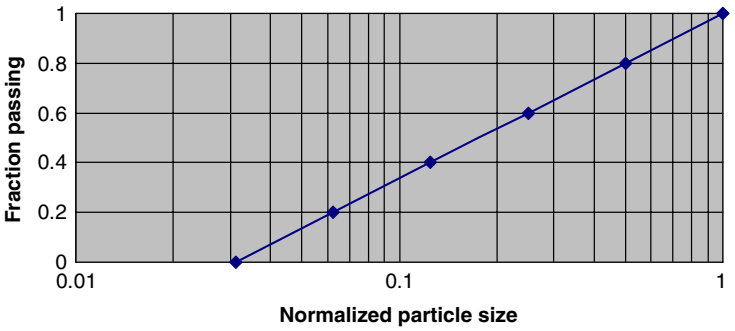

Figure 31. Uniform size distribution used to assemble the packs—points show different particle sizes comprising individual packs.

Robustness: An important property of MR multi-step contact detection algorithm is that neither memory nor CPU requirements are a function of spatial distribution of the discrete elements, thus the algorithm performs well for both loose and dense packs.

In order to demonstrate the above properties of MMR-linear contact detection algorithm a set of numerical experiments on systems consisting of N discrete elements of varying diameter d that changes from 1 to 1/32 according to uniform size distribution is performed. The particles are placed randomly inside a cube-shaped space in such a way that a relatively dense pack is obtained without the particles overlapping each other. This is achieved by first placing particles at a regular raster as shown in Figure 19. Particles are assigned uniform diameter randomly following uniform size distribution. After that the particles are allowed to move until they reposition within a prescribed volume in such a way that no overlap exists. The size of space is predefined so that a packing density of 0.5 was achieved, i.e. the ratio between the volume of solid inside the space and the volume of space was 0.5. Uniform size distribution (Figure 31) is given by the following formula:

$$y = \frac{\log(x/x_{\min})}{\log(x_{\max}/x_{\min})} \tag{36}$$

where y is passing, x is the sieve size, $x_{\min}$ is the size of the smallest particle (in this case $d = 1/32$ corresponding to 0% passing) and $x_{\max}$ is the size of the largest particle (in this case $d = 1$ corresponding to 100% passing) (Figure 31). For each problem contact detection is repeated 10 times and the cumulative CPU times for all 10 contact detections as a function of the total number of discrete elements comprising the problem were recorded. The CPU time as function of number of particles is shown in Figure 32. The results shown were obtained by changing N from $N = 8000$ to $N = 5\,000\,211$.

It is worth noting that the total CPU time is proportional to N. This is because both search and sort algorithms have linear CPU time requirements. When the results from Figure 32 are compared to the results from Figure 20, it is evident that the total CPU time for systems comprising particles of different sizes is much greater than the total CPU time for mono-sized particles.

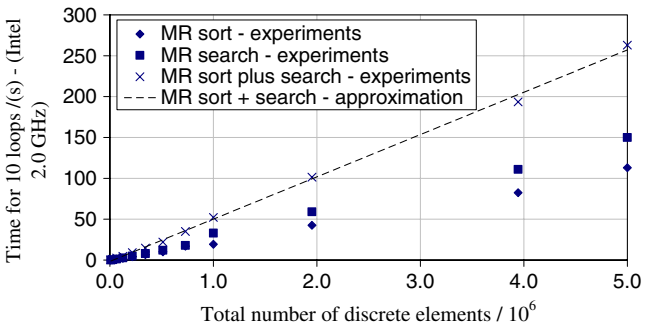

Figure 32. CPU time as a function of the number of particles. MMR contact detection for uniform size distribution.

8. CONCLUSION

The MR-linear contact detection algorithm presented in this paper is yet another example of the so-called linear contact detection algorithms. These include NBS, C-grid and SMB algorithms. Unlike NBS algorithm, in MR-linear contact detection algorithm the data structure is constantly available. In NBS algorithm, the data structure is rebuilt each time step. With MR-linear contact detection algorithm the data structure is only modified each time step. Thus MR-linear contact detection algorithm has all the advantages of a binary tree-based algorithm. In addition, it has much better performance as it was demonstrated by the examples shown 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.

The multi-step MR contact detection algorithm presented in this paper is suitable for systems comprising particles of greatly varying sizes. The RAM requirements are independent of size distribution of the pack and are the same as those for the MR algorithm. However, CPU requirements depend on the number of steps and size distribution. The upper limit for CPU time of the *n*-step algorithm (the worst-case scenario) is *n*-multiple of CPU time for the single size MR algorithm.

REFERENCES

1. Chung SH, Mustoe GGW. Effects of particle shape and size distribution on stemming performance in blasting. *Proceedings of the 3rd International Conference on Discrete Element Methods*, Santa Fe, NM, U.S.A., 2002.
2. Preece DS, Chung SH. An algorithm for improving 2-D and 3-D spherical element behavior during formation of muck piles resulting from rock blasting. *Proceedings of the 3rd International Conference on Discrete Element Methods*, Santa Fe, NM, U.S.A., 2002.
3. Munjiza A. *The Combined Finite-discrete Element Method*. Wiley: New York, 2004.
4. Williams JR, Mustoe GGW. *Proceedings of the 2nd U.S. Conference on Discrete Element Methods*, MIT, MA, 1993.

5. Perkins E, Williams JR. Generalized spatial binning of bodies of different sizes. *Proceedings of the 3rd International Conference on Discrete Element Methods*, Santa Fe, NM, U.S.A., 2002.
6. Owen DRJ, Feng YT, Cottrel MG, Yu J. Discrete/finite element modelling of industrial applications with multi-fracturing and particulate phenomena. *Proceedings of the 3rd International Conference on Discrete Element Methods*, Santa Fe, NM, U.S.A., 2002.
7. O'Connor R, Gill J, Williams JR. A linear complexity contact detection algorithm for multi-body simulation. *Proceedings of the 2nd U.S. Conference on Discrete Element Methods*, MIT, MA, 1993.
8. Oldenburg M, Nilsson L. The position code algorithm for contact searching. *International Journal for Numerical Methods in Engineering* 1994; **37**:359–386.
9. Bonet J, Peraire J. An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems. *International Journal for Numerical Methods in Engineering* 1991; **31**:1–17.
10. Munjiza A, Andrews KRF. NBS contact detection algorithm for bodies of similar size. *International Journal for Numerical Methods in Engineering* 1998; **43**:131–149.
11. Podgorelec D, Klajnšek G. Acceleration of sweep-line technique by employing smart quicksort. *Information Sciences* 2005; **169**:383–408.
12. Knuth DE. *The Art of Computer Programming* (2nd edn). Addison-Wesley Professional: Reading, MA, 1998.
13. Koziara T, Bićanić N. Bounding box collision detection. *13th ACME Conference*, University of Sheffield, Sheffield, U.K., 2005.
14. Li CF, Feng YT, Owen DRJ. SMB: collision detection based on temporal coherence. *Computer Methods in Applied Mechanics and Engineering*, in press.
15. Feng YT, Owen DRJ. An augmented spatial digital tree algorithm for contact detection in computational mechanics. *International Journal for Numerical Methods in Engineering* 2002; **55**:159–176.

