Numerical comparison of some contact detection algorithms

Journal:	<i>Engineering Computations</i>
Manuscript ID	EC-01-2016-0047.R1
Manuscript Type:	Research Article
Keywords:	Contact detection algorithm, Combined Finite-Discrete Element Method (FDEM), comparison, performance

Abstract:

Contact interaction and contact detection remain key components of any discontinua simulations. The methods of discontinua include: Combined Finite-Discrete Element Method (FDEM), Discrete Element Method (DEM), Molecular Dynamics (MD), etc. In recent years, a number of contact detection algorithms have been developed, such as: Munjiza-Rougier (MR), MunjizaRougier-Schiava (MR-S), Munjiza-No Binary Search (NBS), Balanced Binary Tree Schiava (BBTS), 3-D Discontinuous Deformation Analysis (DDA), and many others. In this work a numerical comparison is conducted of certain algorithms often used in FDEM for~~on~~ bodies of the same size. These include MR, MR-S, NBS and BBTS algorithms.

Keywords: Contact detection algorithm; Combined Finite-Discrete Element Method (FDEM); comparison; performance.

1. Introduction

Contact detection (CD) is one of the most important aspects of the DEM Method. It is the process of finding all interacting couples at a particular time. Large-scale DEM simulations involve contact between millions of different entities, usually called atoms, molecules, particles, bodies, discrete elements, etc. In dynamics problems, where most of the particles are in motion, the CPU time required to search for all the contacting couples (those in contact with each other) can, in many cases, be over 60% of the total CPU time (Munjiza, 2004).

For contact detection, a wide range of algorithms have been proposed, such as: Direct Search (DS), Quick Sort (Podgorelec and Klajnvek, 2005), Balanced Binary Tree (BBT) (Knott, 1971), Munjiza-No Binary Search (NBS) (Munjiza and Andrews, 1998), Munjiza-Rougier (MR) (Munjiza *et al.*, 2006), Munjiza-Schiava (MS) (Schiava D'Albano *et al.*, 2013), MR-S (MunjizaRougier-Schiava) (Schiava, 2014), Williams C-grid (Perkins and Williams, 2001), Alternating Digital Tree (ADT) (Bonet and Peraire, 1991), Augmented Spatial Digital Tree (ASDT) (Feng and Owen, 2002), 3D-DDA (Beyabanaki *et al.*, 2008), Discrete Function Representation (Williams and O'Connor, 1999), Sort Moving Boxes (SMB) (Li *et al.*, 2006) to name but a few.

For developers working in FDEM, DEM, MD, etc., it is important to be able to decide which search algorithm to use in a particular situation. None of the searches in current literature are perfect and some may underperform in certain situations. Thus, in this analysis, a comparative study of different search algorithms (namely MR, MR-S, NBS and Balanced Binary Tree Schiava (BBTS) (Schiava, 2014)) for bodies of similar size has been conducted.

2. FDEM

FDEM was developed in the mid 90's by Munjiza (Munjiza *et al.*, 1995) who wrote the first comprehensive book on this subject in 2004 (Munjiza, 2004), followed by a second book in 2011 (Munjiza *et al.*, 2011) and a third book in 2015 (Munjiza *et al.*, 2015). In this formulation, each entity is discretised into $N=1,2,3,...,m$ finite elements (FEs) such as triangles in 2D (Lukas *et al.*, 2014), shells in 2.5D (Munjiza *et al.*, 2013) tetrahedron in 3D (Xiang *et al.*, 2009), etc. This allows the calculus of deformations, fracture and other properties.

There is no need to build a global system of equations, as for the Finite Element Method (FEM) and Discontinuous Deformation Analysis (DDA) (Shi, 1988). Instead, the position, velocity, rotation, etc. are calculated using the equation of motion of each FE.

The governing equation is:

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} + \mathbf{F}_{int} - \mathbf{F}_{ext} - \mathbf{F}_{con} - \mathbf{F}_{jnt} = \mathbf{0} \quad (1)$$

where $\ddot{\mathbf{u}}$ is the acceleration, $\dot{\mathbf{u}}$ is the velocity, $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the damping matrix, $\mathbf{F}_{ext}$ is the external force vector and $\mathbf{F}_{con}$ is the contact (interaction) force vector, $\mathbf{F}_{int}$ is the internal force vector, and $\mathbf{F}_{jnt}$ is the joint force vector.

Independent FEs are “glued” together using joints. Joints are objects that allow the transition from continua to discontinua, acting as an adhesive between elements. One of the most used

mathematical models for joints is the combined single and smeared crack model (Munjiza, 1999).

One of the typical applications of FDEM is for numerical simulations of rock mechanics problems. Figure 1 shows a simulation sequence of an open pit slope's collapse (Lukas, 2015).

Fig 1. Open pit slope simulation sequence (stress values in Pa). a) The whole domain at time 0 s. b) Detailed view of the fracture pattern at 2.5 s. c) Detailed view of the fracture pattern at 5 s.

3 BBTS contact detection algorithm

In a binary tree each node a , is a right son of b only if $a < b$, and a , is a left son of b only if $a > b$. For a tree that complies with these two rules and in “which the items are randomly-received”, the search time required to find a matching item is generally a logarithmic function of the number of elements N . However, this only happens when the tree is balanced, which occurs when “no excessively short or long path exists” (Knott, 1971). Such paths tend to increase the average number of comparisons required.

The Balanced Binary Tree is a logarithmic search algorithm, i.e. the contact detection (CD) time needed to find a matching item is given by

$$T \propto \log(N) \quad (2)$$

and the contact detection time needed to find N items is given by

$$T \propto N \log(N) \quad (3)$$

In the case of the MR and NBS algorithms, the CD time increases proportionally with the amount of DEs. The CD time for these algorithms is given by

$$T \propto N \quad (4)$$

BBTS (Balanced Binary Tree Schiava) has two main algorithms: BBTS-load and BBTS-search. Modifications have been included (Schiava, 2014) to make it suitable for contact search in 1D, 2D and 3D, by combining the BBT with space cell decomposition (Munjiza and Andrews, 1998).

The processes involving the storage and retrieval of data in a balanced binary tree are described in depth by Knott (Knott, 1971). One of the differences between Knott and BBTS is the way in which nodes are considered. In the BBTS formulation the nodes are C++ objects, meaning it is

not necessary to include an array which points from one node to another. This information is stored in each node instead of in an array.

4 NBS contact detection algorithm

This algorithm was one of the first to present a proportional CD time with respect to the number N of DEs:

$$T \propto N \quad (5)$$

It is based on the space decomposition into subdomains (cells) of the same size as shown in Figure 2. The size of the square subdomains is defined by the diameter d of the biggest spherical bounding box that contains the biggest element in the domain. The entire domain is represented by a singly connected list using arrays (Munjiza and Andrews, 1998).

Fig 2. NBS domain decomposition and singly connected x and y -lists. Figure adapted from Munjiza (Munjiza, 2004).

Individual discrete elements are mapped into cells in two stages:

- y coordinates of all discrete elements are integerised and normalised by using $y_{\min}$ and placed into a singly connected y -list in a corresponding row (list “Rows” on the left in Figure 2).
- For each non-empty list in “Rows” the x coordinate of each discrete element is integerised and normalised by $x_{\min}$ and new x -lists are created for each row, for instance Row 3 in Figure 2.

Contact detection is performed by travelling on non-empty lists (Figure 2). Contact detection for each separate cell is performed only with neighbouring cells included in a contacting mask. A detailed explanation of this algorithm is given in the FDEM book by Munjiza (Munjiza, 2004).

5 MR contact detection algorithm

The contact detection time for the MR algorithm is proportional to the number of DEs and is given by

$$T \propto N \quad (6)$$

It comprises three algorithms: a) Quick Sort (QS), b) MR-Sort, c) MR-Search. [In this section a short description of these algorithms is presented.](#) QS is not a linear algorithm and its CPU time is given by

$$T \propto N \log_2(N) \quad (7)$$

This does not affect the linearity of MR, as it is used to sort out the list only during the first time step. The MR algorithm is explained in depth by Rougier (Rougier, 2009). [In this section a short description of these algorithms is presented.](#)

The MR domain is subdivided in a similar way to the NBS domain, into cubical cells of [diameter size \$d\$](#) as shown in Figure 4, in which the integerised coordinates of the DEs are calculated using the equation 8.

$$\begin{aligned} i_x &= (\text{int}) \left(\frac{x - x_{\min}}{d} \right) \\ i_y &= (\text{int}) \left(\frac{y - y_{\min}}{d} \right) \\ i_z &= (\text{int}) \left(\frac{z - z_{\min}}{d} \right) \end{aligned} \quad (8)$$

where $x_{\min}$, $y_{\min}$, $z_{\min}$ are the minimum coordinates of the computational domain, x , y , z are the coordinates of the discrete element, d is the diameter of the biggest discrete element and i_x , i_y , i_z are the integerised coordinates of the discrete element.

Figure 3: MR-object shown with id. Figure adapted from Munjiza *et al.* (Munjiza *et al.*, 2006).

MR was originally implemented in C++ using objects. All the MR-objects in the system are arranged in a closed double connected list. Each one of these MR-objects has a pointer to the next MR-object in the list, a pointer to the preceding MR-object in the list and a pointer to a discrete element and its integerised coordinates $V(i_x, i_y, i_z)$ as shown in Figure 3.

All the MR-objects are sorted according to the sorting criterion given by the equation 9.

$$\begin{aligned} & \{[(i_{zi} > i_{zj})] \text{ or } [(i_{zi} = i_{zj}) \text{ and } (i_{yi} > i_{yj})]\} \\ & \text{or } \{[(i_{zi} = i_{zj}) \text{ and } (i_{yi} = i_{yj}) \text{ and } (i_{xi} > i_{xj})]\} \end{aligned} \quad (9)$$

where i_{xi} , i_{yi} and i_{zi} are the integerised coordinates of the DE i and i_{xj} , i_{yj} and i_{zj} are the integerised coordinates of the DE j .

Figure 4: System at initial time t . Figure modified from Munjiza *et al.* (Munjiza *et al.*, 2006).

Figure 5: System at time $t+h$. Figure modified from Munjiza *et al.* (Munjiza *et al.*, 2006).

MR-Sort. It is based on the stability criteria (Munjiza *et al.*, 2011) present in discrete element simulations to avoid unnecessary comparisons in the MR-list because the list in the time step $t+h$ is nearly sorted.

After updating the positions of the **DEs** in Figure 4, the list for the system in Figure 5 is no longer sorted as shown in Figure 6. In the example, the element 5 has to swap positions with element 3 because its integerised coordinates V are smaller than those of element 3. Sorting is produced by parsing the entire MR list element by element until there is no more DE to swap. The final sorted list is shown in Figure 7.

Figure 6: Unsorted list in the time step $t+h$. Figure modified from Munjiza *et al.* (Munjiza *et al.*, 2006).

Figure 7: Sorted list in the time step $t+h$. Figure modified from Munjiza *et al.* (Munjiza *et al.*, 2006).

MR-Search. To avoid the processing of contacting couples multiple times, MR-Search utilises a contact checking mask (Munjiza *et al.*, 1998) mapped into pointers on the MR-list.

If the integerised coordinates are short integer numbers, then the total amount of RAM memory used per object is given by:

$$\begin{aligned}
 V &= 3 \text{ short integer numbers} = 6 \text{ bytes} \\
 P_{prv} &= 1 \text{ pointer} = 8 \text{ bytes} \\
 P_{nxt} &= 1 \text{ pointer} = 8 \text{ bytes} \\
 O &= 1 \text{ pointer} = 8 \text{ bytes} \\
 M &= V + P_{prv} + P_{nxt} + O = 30 \text{ bytes}
 \end{aligned} \tag{10}$$

If the pointer O to a discrete object is replaced by an integer id then the total RAM memory is $M = 26$ bytes.

6 MR-S algorithm

The principal difference between MR-Schiava (MR-S) and MR is the use of arrays instead of objects. This has the advantage of reducing the amount of RAM memory utilised by the algorithm (Schiava, 2014). The main algorithms used by MR (Quick Sort, MR-Sort and MR-Search) are employed without great changes. The data corresponding to the whole set of DEs is rearranged into five one-dimensional arrays of sizes equal to N plus one, as shown in Figure 8.

Figure 8: Data structure for the implementation of MR algorithm with arrays.

The first array could either be an array of integers corresponding to the id of each discrete element or it could be an array of pointers O to each discrete element. If the integerised coordinates are short integer numbers, the total amount of memory per element used is given by

$$\begin{aligned}
 V &= 3 \text{ short integer numbers} = 6 \text{ bytes} \\
 O &= 1 \text{ pointer} = 8 \text{ bytes} \\
 P &= 1 \text{ integer} = 4 \text{ bytes} \\
 M &= V + O + P = 18 \text{ bytes}
 \end{aligned} \tag{11}$$

If the pointer O to a discrete object is replaced by an integer id then $M = 14$ bytes. The percentage of RAM memory reduction of MR-S with respect to MR is 40%.

Quick Sort. The pointer to the preceding MR-object in the MR algorithm is only used in the first time step when the list is ordered using QS. Using arrays, there is no need to record the previous or the next element. For ordering the arrays using Quick Sort, only the array corresponding to the id and the three arrays corresponding to V are used as shown in Figure 9.

Figure 9: Arrays used by Quick Sort. a) Before ordering. b) After ordering.

After the Quick Sort Algorithm is implemented, all the objects on MR-S are spatially ordered. For MR-Sort and MR-search only a pointer to the object with V equal to or bigger than the current element is needed. This is achieved using an array of pointers p which is initialised after the list is ordered, as shown in Figure 10. If the elements change their coordinates, the values of V are updated to i_x , i_y and i_z arrays. As the list is no longer in order, it has to be updated.

It is possible to avoid using the array p to update the list, by moving data stored in id , i_x , i_y and i_z arrays from the old position to a new position, but this would be CPU inefficient. It is more CPU time-efficient to update only one array p than to update 4 arrays id , i_x , i_y and i_z .

Figure 10: a) Arrays at time step t . b) Schematic view of the elements order at time t .

MR-sort. MR-sort updates the array p , as shown in Figure 11, every time CD is performed. All the other arrays remain unchanged.

Figure 11: a) Arrays at time step $t+h$. b) Schematic view of the elements order at time step $t+h$.

MR-search. MR-search is applied in the same way as for the previous algorithm using the contact mask (Munjiza, 1998) on the array p .

7. Numerical comparison between MR, MR-S, NBS and BBTS

In order to test these algorithms, a set of numerical experiments was designed. All the discrete elements employed are spherical of diameter d to simplify the calculus of contact detection. All the experiments were carried out on a DELL 4700 (Intel 2.8 GHz / 4G RAM) running on Fedora Core 4.0. To save RAM memory, pointers O to objects in BBTS, MR, MR-S and NBS are replaced by an integer.

The search tests are performed on virtual discrete elements with their integerised position as the only data stored to further save RAM memory. This means that the mapping between float to integerised coordinates (equation 8) is skipped. This map operation mapping (equation 8) is a linear operation in terms of CPU times and the relative comparison of CPU total detection time between the algorithms is still relevant, not affecting the relative comparison between these algorithms. Also, there is no calculus of interaction and no calculus of deformation.

At the beginning of the experiments the positions of each DE are defined on different package shapes and package ordering. As there is no interaction, new positions of the discrete elements are calculated using a random number generator. In order to make the experiments as close to real FDEM simulations as possible, the cumulative contact detection times are measured after 10 cycles of:

1. Perform contact detection
2. Move DEs

Experiment I. The discrete elements are placed at equidistant distance d forming a cuboid shape cluster as shown in Figure 12.

Fig 12. Experiment I cluster. Figure modified from Munjiza *et al.* (Munjiza *et al.*, 2006).

The order into which the elements of the uniform raster are introduced is given by

$$\begin{pmatrix} i_{xmin}, i_{ymin}, i_{zmin} \\ \dots \\ (i_{xmin}, i_{ymax}, i_{zmax}) \end{pmatrix} \quad \begin{pmatrix} i_{xmin}, i_{ymin}, i_{zmin} + 1 \\ (i_{xmin}, i_{ymin}, i_{zmax}) \\ \dots \end{pmatrix} \quad \begin{pmatrix} i_{xmin}, i_{ymin}, i_{zmin} + 2 \\ \dots \\ (i_{xmax}, i_{ymax}, i_{zmax}) \end{pmatrix} \quad (12)$$

where i_{xmin} , i_{ymin} , i_{zmin} are the minimum initial x , y , z integerised coordinates and i_{xmax} , i_{ymax} , i_{zmax} are the maximum initial x , y , z integerised coordinates.

At each time step, each particle (DE) is moved randomly along the three axes. The randomness is obtained by using a random number generator routine called *Generator* based on a subroutine called rand2 (William *et al.*, 2002). The generator produces a random number between 0.0 and 1.0, exclusive of the endpoint values. The parameters r_x , r_y , r_z used at each step by the *Generator* are calculated by

$$\begin{aligned} r_x &= -i_x + iniRand_t \\ r_y &= -i_y + iniRand_t \\ r_z &= -i_z + iniRand_t \end{aligned} \quad (13)$$

where i_x , i_y , i_z are the integerised coordinates and

$$\begin{aligned} iniRand_t &= -1 & t &= 0 \\ iniRand_t &= iniRand_{t-1} - 10 & t &= 1, 2, 3, \dots, 9 \end{aligned} \quad (14)$$

where t is the time step. Each random number is obtained as

$$\begin{aligned} \alpha_x &= Generator(r_x) \\ \alpha_y &= Generator(r_y) \\ \alpha_z &= Generator(r_z) \end{aligned} \quad (15)$$

and each new coordinate is calculated by

Formatted: Font: Italic

Formatted: Font: Italic

Formatted: Font: Italic

Formatted: Font: Italic

$$\begin{aligned}
i_x &= i_x + (int)(\alpha_x) \\
i_y &= i_y + (int)(\alpha_y) \\
i_z &= i_z + (int)(\alpha_z)
\end{aligned} \tag{16}$$

Fig 13. Experiment I: cumulative contact detection times.

The cumulative contact detection times for the different algorithms are shown in Figure 13. For this particular set-up the fastest algorithm is the NBS. This is an unexpected result as the NBS rebuilds singly connected x and y -lists (Domain Decomposition), before performing contact detection, while MR and MR-S only need to update their lists.

The cumulative times of contact detection, sort and search for the BBTS are shown in Figure 14, where the approximation curves were obtained by applying the least square method to the function

$$t = \alpha N \log(N) \tag{17}$$

Fig 14. Experiment I: cumulative times for BBTS.

Experiment II. A total of 1,000,000 discrete elements are placed at equidistant distance s as shown in Figure 15.

Fig 15. Experiment II cluster. Figure modified from Munjiza *et al.* (2006).

The order into which the elements are introduced is given by

$$\begin{array}{ll}
 (i_{xmin}, i_{ymin}, i_{zmin}) & (i_{xmin}, i_{ymin}, i_{zmin} + s) \\
 (i_{xmin}, i_{ymin}, i_{zmin} + 2s) & \dots \\
 (i_{xmin}, i_{ymin}, i_{zmin} + js) & \dots \\
 (i_{xmin}, i_{ymin} + js, i_{zmin} + js) & \dots \\
 \dots & (i_{xmin} + js, i_{ymin} + js, i_{zmin} + js)
 \end{array} \quad (18)$$

where i_{xmin} , i_{ymin} , i_{zmin} are the minimum initial x , y , z integerised coordinates and j is the number of particles per axis. This order is called xyz.

Fig 16. Experiment II: MR and NBS cumulative contact detection times.

Fig 17. Experiment II: cumulative contact detection times.

The density depends on spacing s as follows (Munjiza *et al.*, 2006)

$$\rho \propto \frac{1}{s^3} \quad (19)$$

At each time step, each particle is moved randomly along the three axes as shown in Equation 16. As different values of s are chosen the packing density varies as (Munjiza *et al.*, 2006)

$$1 \leq \rho \leq \frac{1}{8000000} \quad (20)$$

The cumulative contact detection times are shown in Figure 16. Detail of the total cumulative contact detection times for the NBS, MR and MR-S algorithms is shown in Figure 17.

The performance of the BBTS algorithm is independent of the density of the packing. The same can be said for the NBS, MR and MR-S. The performances of the MR and MR-S are relatively similar while the fastest algorithm is the NBS.

Experiment III. The discrete elements are placed randomly in x , y and z directions inside a cube as shown in Figure 18.

Fig 18. Experiment III cluster.

The order in which the elements are introduced is also random. The randomness is obtained by using the *Generator* routine. The constants to initialize the random generator are

$$\begin{aligned} r_x &= -1 \\ r_y &= -2000 \\ r_z &= -500000 \end{aligned} \quad (21)$$

Each coordinate is calculated as

$$\begin{aligned} i_x &= (\text{int}) (\delta \text{Generator}(r_x)) + i_{xmin} \\ i_y &= (\text{int}) (\delta \text{Generator}(r_y)) + i_{ymin} \\ i_z &= (\text{int}) (\delta \text{Generator}(r_z)) + i_{zmin} \end{aligned} \quad (22)$$

where

$$\delta = js \quad (23)$$

and j is equal to the number of elements per axis, in the case of the uniformly packed experiments. The variable s controls the packing density of the system and in this case is set to

$$s = 3 \quad (24)$$

The total number of DEs is given by

$$N = j^3 \quad (25)$$

The process of calculating the random integerised coordinates for each one of the elements is illustrated in the Algorithm 1.

Algorithm 1 3D: random integerised coordinates

```

1:  i=0
2:  for (i1 = 0; i1 < j; ++ i1) do
3:      for (i2 = 0; i2 < j; ++ i2) do
4:          for (i3 = 0; i3 < j; ++ i3) do
5:              x[i] = (int) (δ Generator(rx)) + ixmin
6:              y[i] = (int) (δ Generator(ry)) + iymin
7:              y[i] = (int) (δ Generator(rz)) + izmin
8:              i++
9:          end for
10:      end for
11: end for

```

Formatted: Not Superscript/ Subscript

Algorithm 1. 3D: random integerised coordinates.

At each time step, each particle is moved randomly along the three axes using the Equation 16. The cumulative contact detection times are shown in Figure 19.

This experiment represents a more general case for Discrete Element simulations, where particles are introduced randomly. The fastest algorithms are the MR and MR-S. The performance of the BBTS algorithm is the worst, as the increase in the amount of operations to build and traverse the binary tree is logarithmic with the increase of DEs, as shown in Figure 20.

Experiment IV. All discrete elements are placed in the same cell. During the whole simulation the elements cannot move away from the box. Basically, each algorithm solves a quadratic search.

The total cumulative times are shown in Figure 21. The approximation curves shown in Figure 21 are obtained by fitting the experimental data to the following function

$$f(N) = aN^2 + bN + c \quad (26)$$

The value of the coefficients a , b and c are determined through the least square method.

Fig 19. Experiment III: cumulative contact detection times.

Fig 20. Experiment III: cumulative times for BBTS.

The behaviour for all the algorithms in terms of cumulative contact detection time is quadratic. This shows the disadvantage of these particular contact detection algorithms. When the size of the cell, relative to the size of the objects, is excessively big, the contact detection time becomes a quadratic function of the number of objects. These algorithms are not designed to work with objects with large variances in their sizes.

Fig 21. Experiment IV same cell: cumulative contact detection times.

8. Discussion and Conclusions

Memory speed test. In Experiment I (Figure 12) and Experiment II (Figure 15) the NBS algorithm outperforms any other. However, in Experiment III (Figure 18) the MR/MR-S outperforms the other algorithms as expected. This difference in the speed between NBS and MR/MR-S occurs when the elements are introduced in a different order. A simple test is performed to demonstrate that the differences in time may be due to the internal use of memory. This test cannot be considered as the general behaviour of an x86 computer, and may be particular to the OS/Machine employed in this work.

Fig 22. List parsing for a case A.

Position Array	First Path Pointers	Second Path Pointers
0	1	
1	2	
2	3	
3	-1	
4		5
5		6
6		7
7		-1

Table 1. Case A.

The velocity required to traverse a 1D vector with two heads is tested. In Case A the vector is ordered in such a way that each of its values is equal to the number of the next position, as shown in Figure 22 and in Table 1.

In the second test (Case B), the vector is ordered in such a way that each value of the vector is equal to the number of the next position plus 1, as shown in Figure 23 and in Table 2.

Fig 23. List parsing for a case B.

Position Array	First Path Pointers	Second Path Pointers
0	2	
1		3
2	4	
3		5
4	6	
5		7
6	-1	
7		-1

Table 2. Case B.

In theory there should be no difference in the time spent to “travel” through this array. However, as is shown in Figure 24 and Figure 25, there is a notable difference in the time of around 10%.

Contact Detection Times. Different tests were performed comprising up to 0.12 billion particles ~~of the same size~~. The linear behaviour of the NBS and the MR, as well as the new MR-S algorithms, ~~were was~~ shown. This exhaustive experiment can be used as a tool to select the best algorithm to be implemented in various cases.

~~The algorithms presented analyzed in this work were where developed for systems with comprising particles with of the similarame size-particles. And their performance will be greatly compromised by the relative difference between the biggest and the smallest bodies, as in the case of polydisperse systems, because the cell size in the whole domain is given by the diameter of the biggest particle (Munjiza, 2004). The contact detection time does not increase linearly with the size difference between the biggest and smallest particle. bigger cell could pack more (smaller sized) particles and the CD time is quadratic to the cell size. For thieses cases other algorithms can be applied such as the MULTISTEP MR (MMR) (Munjiza *et al.*, 2006), and MunjizaSchiava (MS) (Schiava, 2014) among others. ;~~

Fig 24. CPU time as a function of the size of the arrays.

The BBTS was the slowest algorithm. However, ~~the main objective of these simulations was to compare different algorithms in different real situations. Alternatively,~~ for researchers who are already working with BT or who need to implement a BT for other reasons (i.e. for a data base), a simple modification to the structure of the BT (Schiava D’Albano, 2014) will allow them to use it as a CD-search algorithm without the difficulty associated with complex discretisation of the space.

It was also shown that, for vector algorithms, the way in which the elements are introduced can affect their velocity (Experiment I and Experiment III) for particular combinations of OS and hardware.

The algorithms tested are not affected in case of loose packing problems (Experiment II) and their CD time remains constant.

The maximum theoretical displacement of each DE for 10 time steps is 8 for experiments I and III. If the experiments would be run for enough steps, this would result in a situation similar to Experiment II, where would be produced as the particles will be so far from each other that no CD would be produced- and the CPU times will remain constant.

Experiment IV shows the problem encountered when a relatively large cell size is used. This is expected since these algorithms were developed for systems comprising bodies of similar size. There is practically no difference in the search time, as all algorithms have quadratic behaviour.

In most of the Discrete Elements simulations the particles are randomly allocated to the space (Experiment III). For these situations the MR and MR-S algorithms were the fastest.

Fig 25. Percentage difference as a function of the size of the arrays.

In most of the Discrete Elements simulations the particles are randomly allocated to the space (Experiment III). For these situations the MR and MR-S presents an advantage in terms of contact detection times. In discrete element simulations where particles are introduced randomly or in which the relative position between particles is constantly changing, the MR algorithm presents an advantage in terms of contact detection times. The conclusions of the previous experiments are shown in Table 3

Experiment	Uniform	DE	Equidistant DE	One cell	Fastest Algorithm
------------	---------	----	----------------	----------	-------------------

Formatted: Space Before: 0 pt, After: 6 pt, Don't hyphenate

Formatted Table

	<u>Raster</u>	<u>ordered</u>			
<u>I</u>	<u>Yes</u>	<u>Yes</u>	<u>No</u>	<u>No</u>	<u>NBS</u>
<u>II</u>	<u>Yes</u>	<u>Yes</u>	<u>Yes</u>	<u>No</u>	<u>NBS</u>
<u>III</u>	<u>No</u>	<u>No</u>	<u>No</u>	<u>No</u>	<u>MR-S</u>
<u>IV</u>	<u>Yes</u>	<u>Yes</u>	<u>Yes</u>	<u>Yes</u>	<u>MR-S</u>

Table 3. Fastest algorithm per experiment.

Formatted: Centered, Space Before: 0 pt, After: 6 pt, Don't hyphenate

In discrete element simulations where particles are introduced randomly or in which the relative position between particles is constantly changing, the MR/MR-S algorithms presents an advantage in terms of contact detection times.

Formatted: Centered, Space Before: 0 pt, After: 6 pt, Don't hyphenate

<u>Algorithm</u>	<u>Object Oriented</u>	<u>Language</u>
<u>NBS</u>	<u>No</u>	<u>C</u>
<u>BBTS</u>	<u>Yes</u>	<u>C++</u>
<u>MR</u>	<u>Yes</u>	<u>C++</u>
<u>MR-S</u>	<u>No</u>	<u>C</u>

Formatted Table

Table 4. Programing paradigms implemented in each algorithm.

Formatted: Centered, Space Before: 0 pt, After: 6 pt, Don't hyphenate

When choosing a contact detection algorithm, not only raw CD velocity should not be the only be-consideration as Scientific / Engineering applications are complex projects. A trade-off between contact detection velocity, memory management and integration must be considered.

It is worth noting that unlike MR-S, the MR algorithm was developed using the object oriented programming paradigm (OOPP), as shown in Table 4. Thus, MR algorithm is making it a natural choice for C++, C#, and Java languages over the MR-S algorithm.

<u>Algorithm</u>	<u>Object Oriented</u>	<u>Language</u>
<u>NBS</u>	<u>No</u>	<u>C</u>
<u>BBTS</u>	<u>Yes</u>	<u>C++</u>
<u>MR</u>	<u>Yes</u>	<u>C++</u>
<u>MR-S</u>	<u>No</u>	<u>C++</u>

Table 4. Programing paradigms implemented in each algorithm.

Formatted: Centered, Space Before: 0 pt, After: 6 pt, Don't hyphenate

The algorithms presented in this work were not ported or tested in functional languages such as F# or Scala where parallel engines such as Apache Spark are available. As these are becoming more popular, future research will have to be conducted in these disruptive languages.

The work presented did not directly investigated the impact of packing density on CD time, thus future works also have to look into other packaging cases apart from the ones presented in this paper should be investigated in the future research.

Formatted: English (U.S.)

Formatted: Space Before: Auto, After: Auto, Hyphenate

Role of the funding source

Not applicable.

REFERENCES

- Beyabanaki, S. A. R., Mikola, R. G., Hatami, K., (2008) “Three-dimensional discontinuous deformation analysis (3-D DDA) using a new contact resolution algorithm” *Computers and Geotechnics*, Vol 35, pp. 346 – 356
- Bonet, J., Peraire, J. (1991), “An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems”, *Int. J. Numer. Meth. Engng.*, Vol. 31, pp. 1–17
- Feng, Y. T., Owen, D. R. J. (2002), “An augmented spatial digital tree algorithm for contact detection in computational mechanics”, *Int. J. Numer. Meth. Engng*, Vol. 55, pp. 159–176
- Knott, G. (1971) “A balanced tree storage and retrieval algorithm”, in *Annual ACM Conference on Research and Development in Information Retrieval*, ACM, Maryland, pp. 175-196
- Li, C. F., Feng, Y. T., Owen, D. R. J. (2006) “SMB: Collision detection based on temporal coherence”, *Computer Methods in Applied Mechanics and Engineering*, Vol. 195 Nr. 19-22, pp. 2252 – 2269
- Lukas, T. (2015) “Space decomposition ~~Based-based~~ parallelization solutions for the combined Finite-Discrete Element Method in 2D” *PhD Thesis Queen Mary, University of London, London, UK*
- Lukas, T., Schiava D’Albano, G. G., Munjiza, A. (2014) “Space decomposition based parallelization solutions for the combined Finite-Discrete Element Method in 2D”, *J Rock Mech Geotech Eng*, Vol. 6, pp. 607-615
- Munjiza, A. (2004), *The Combined Finite Discrete Element Method*, Wiley, Chichester.
- Munjiza, A., Andrews, K. (1998), “NBS contact detection algorithm for bodies of similar size”, *Int. J. Numer. Meth. Engng*, Vol. 71, pp. 469-483
- Munjiza, A., Andrews, K. R. F., White, J. K. (1999), “Combined single and smeared crack model in combined finite-discrete element analysis”, *Int. J. Numer. Meth. Engng*, Vol. 44, pp. 1097-0207
- Munjiza, A., Knight, E. and Rougier, E. (2011), *Computational Mechanics of Discontinua*, Wiley, Chichester.

Munjiza, A., Knight, E. E., & Rougier, E. (2015). *Large strain finite element method: a practical course*. John Wiley & Sons.

Munjiza, A., Lei, Z., Divic, V., Peros, B (2013) “Fracture and fragmentation of thin shells using the combined finite–discrete element method”, *Int. J. Numer. Meth. Engng*, Vol. 95, pp. 478-498

Munjiza, A., Owen, D.R.J. and Bicanic, N. (1995), “A combined finite-discrete element method in transient dynamics of fracturing solids”, *Eng. Comput.* Vol. 12, pp. 145-174.

Munjiza, A., Rougier, E., John, N. (2006) “MR linear contact detection algorithm”, *Int. J. Numer. Meth. Engng*, Vol. 66, pp. 46-71

Perkins, E., Williams, J. (2001) “C-Grid: Neighbour Searching for Many Body Simulation”, in *Proceedings of ICADD-4, Fourth International Conference on Analysis of Discontinuous Deformation*, University of Glasgow, Scotland, pp. 427–438.

Podgorelec, D., Klajnvek, G. (2005), “Acceleration of sweep-line technique by employing smart quicksort”, *Inf. Comput Sci.* Vol. 169, pp. 383-408

Rougier, E. (2009) “Discrete Element Method for Simulation of Gas Micro-Flows” *PhD Thesis Queen Mary, University of London, London, UK*

Schiava D’Albano, G. G. (2014) “Computational and algorithmic solutions for large scale combined finite-discrete elements simulations” *PhD Thesis Queen Mary, University of London, London, UK*

Schiava D’Albano, G. G., Munjiza, A., Lukas, T. (2013) “Novel MS (MunjizaSchiava) contact detection algorithm on Multicore PC” in *ECCOMAS*, Stuttgart, pp. 35-45

Shi, G. (1988) “Discontinuous deformation analysis a new numerical model for statics and dynamics of block systems” *PhD Thesis University of California Berkeley*

William, H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P. (2002) *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press

Williams, J., O’Connor, R. (1999) “Discrete element simulation and the contact problem”, *Archives of Computational Methods in Engineering*, Vol. 6, pp. 179-304

Xiang, J., Munjiza, A., Latham, J. P. (2009) “Finite strain, finite rotation quadratic tetrahedral element for the combined finite–discrete element method”, *Computer Methods in Applied Mechanics and Engineering*, Vol. 79, pp. 946 – 978

517x376mm (72 x 72 DPI)

517x376mm (72 x 72 DPI)

517x376mm (72 x 72 DPI)

6
6
4

5

id

--	--

id

id

1	2	3	-1	5	6	7	-1
---	---	---	----	---	---	---	----

2	3	4	5	6	7	-1	-1
---	---	---	---	---	---	----	----

0 1 2 3 4 5 6 7

Table 1. Case A.

Position Array	First Path Pointers	Second Path Pointers
0	1	
1	2	
2	3	
3	-1	
4		5
5		6
6		7
7		-1

Table 2. Case B.

Position Array	First Path Pointers	Second Path Pointers
0	2	
1		3
2	4	
3		5
4	6	
5		7
6	-1	
7		-1

Table 3. Fastest algorithm per experiment

Experiment	Uniform Raster	DE ordered	Equidistant DE
I	Yes	Yes	No
II	Yes	Yes	Yes
III	No	No	No
IV	Yes	Yes	Yes

Table 4. Programming paradigms implemented in each algorithm.

Algorithm	Object Oriented	Language
NBS	No	C
BBTS	Yes	C++
MR	Yes	C++
MR-S	No	C++

One cell	Fastest Algorith m
No	NBS
No	NBS
No	MR-S
Yes	MR-S