MPP Contact: Options and Recommendations

Brian Wainscott

LSTC

Introduction

There are many different contact algorithms currently implemented in LS-DYNA, each having various options and parameters. I will discuss only a few of them. Much of what is covered will only apply to the MPP version of LS-DYNA. I will start with some general background information, and then give details about some of the more interesting recent options that you might find useful.

Background

Most of the contact algorithms in LS-DYNA, at the lowest level, are concerned primarily with the idea that there are slave nodes on one surface which are coming into contact (or are tied to) master segments on a second surface. Fundamentally, nodes are compared to segments. These checks can be fast and efficient, and lead to useful, robust contact behavior. But in certain situations, problems can arise. For example, suppose you have the very simple situation of two identical cubes pressing against each other – say, two sugar cubes (one element each) stacked on a table under gravity:

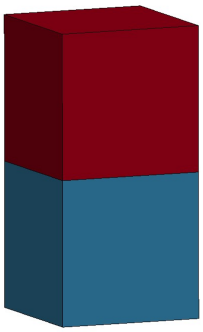

Fig.1: Stacked Cubes

With a standard node to surface contact this configuration is very unstable. It is obvious, for example, that if the top cube is rotated 45 degrees around the vertical axis, then the cubes will pass right through each other without any node to surface interaction at all.

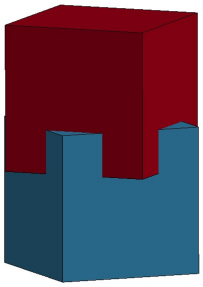

Fig.2: Stacked Rotated Cubes

In fact, if the two cubes are not EXACTLY aligned, then this can happen. In a large model with many more elements you may not see this kind of behavior, but it can happen near the edges of solid elements and is something you should be aware of. I bring this up primarily to point out that there are also segment to segment contact options in LS-DYNA (specifically, the soft=2 option on AUTOMATIC_*_TO_SURFACE and AUTOMATIC_SINGLE_SURFACE) which do not have this particular shortcoming. Those use a different approach, and nothing in the remainder of this paper applies to them.
The *CONTACT keyword has several MPP specific options available via the _MPP keyword option. Some of these options appear to duplicate parameters available in the normal *CONTACT input (e.g. ns2track and parmax). In general, they do not – the similar parameters (depth and maxpar) are ignored. When the MPP contact was being developed, the feeling was that any non-default values input for these parameters were likely due to user experience and tuning with the SMP contacts, and so did not apply to the MPP algorithms. (Due to the very nature of MPP, the pre-existing SMP contact algorithms could not be efficiently modified to be used in the MPP version, so the MPP versions of most of the contacts use entirely different subroutines and where necessary slightly different algorithms.)

Older Options

IGNORE

Probably the single most useful option related to contact is the IGNORE option. This first became available in the MPP version, and so is on the _MPP card. Later it was supported in SMP, and so is also available on optional card C and in CONTROL_CONTACT. They each override the other in that order: *CONTROL_CONTACT provides the default, which is overridden by optional card C, which is overridden by the _MPP value. But you don't really need to know that – just know that you should turn it on: set it on *CONTROL_CONTACT and use it for everything. Well, historically IGNORE was not suggested for metalforming problems. With recent improvements I expect it would be OK even for these, but it should certainly be used for high speed impact problems.
This option is a bit misnamed, because its application has grown over the years. Originally, this option was designed to compensate for any initial penetrations in the model. If the original geometry of a model had some slave nodes already penetrating the master surface, the amount of penetration was remembered for each such node, and compensated for, so that no forces were applied – this initial penetration amount was ignored, hence the name of the option. But now much more is done. If later in the calculation (for whatever reason) a node is discovered to be penetrating a segment significantly more (or less) than it did on the previous cycle, then adjustments to the “ignored“ amout of penetration are made to keep the contact force reasonable. This can substantially aid overall stability.
If IGNORE is set to 0, then each contact interface takes a couple of iterations of trying to move slave nodes to remove any initial penetrations. But this doesn't always work because moving the nodes sometimes creates other penetrations, and of course it distorts the geometry. If IGNORE is set to 1, then no nodes are moved and penetrations are compensated for. If IGNORE is set to 2, then warning messages are issued about initial penetrations (so you can go back and clean up the model, which is always a good idea), but no nodes are actually moved – the calculation is run with penetration compensation enabled.

NS2TRACK

By default, each node keeps track of the 3 segments it might have to worry about being in contact with, so if it gets backed into a corner it will behave properly. But if you know you have a smooth surface or simple geometry (e.g. metalforming), then setting ns2track to 1 will reduce memory requirements, improve general behavior, and run slightly faster.

Newer Options

GROUPABLE

The whole point of the MPP version of LS-DYNA is to execute in parallel to the greatest extent possible. The algorithms used for the individual contact routines in the MPP version all work in a distributed fashion, splitting the work among whichever processors end up with part of the contact definition in question. But if there are more than just a few contact interfaces, there can be significant inefficiencies. For example, suppose we have a problem with 4 contact interfaces, which I will number 100, 101, 102, and 103, and we are running the model on 3 processors. Further suppose that the contacts end up being distributed so that the processors are involve in each contact as shown here:

	Proc 0	Proc 1	Proc 2
Con 100	X	X	
Con 101		X	X
Con 102	X		X
Con 103	X	X	X

Table 1: Distribution of Contact Interfaces

The standard MPP contact routines will process each contact in order. That means that while processors 0 and 1 are computing contact 100, processor 2 will be waiting – each blank in the chart above represents an idle processor.
Most of the contact algorithms have the same basic communication pattern. In this example, processors 0 and 1 will communicate 3 sets of similar messages, one for each contact they share. But by rearranging the communication and computation, we can group all of the messages (of the same type) together. This not only reduces the message overhead due to the decreased number of messages, it also allows more calculation to happen in parallel. So instead of each processor doing this:

```
foreach contact:
    send/recv node data for this contact
    compute node/segment interactions and pack results
    send/recv contact results for this contact
    process results
```

we do this:

```
foreach contact:
    pack all node data
send/recv node data

foreach contact:
    compute node/segment interactions and pack results
send/recv contact results

foreach contact:
    process results
```

The communication related parts of the code are considerably more complicated. But fewer messages are sent and, more importantly, no processor has to wait around while some contact it is not involved in is being computed by a processor it needs to communicate with.
This new organization of the communication and computation requires a different set of subroutines to efficiently process the contact calculations. Consequently, not every *CONTACT type supports the GROUPABLE option, and those that do may behave slightly differently when this option is used. Because of the increased performance available with this option, most future development efforts will be directed toward these algorithms instead of the older, non-groupable contacts. Some new options are only available in the GROUPABLE contacts, for example, and thus turn on GROUPABLE whenever they are selected. The GROUPABLE option can be turned on for a specific contact via the 4th field of the second optional _MPP card.

FORCE TRANSDUCERS

Force transducers are not really contact interfaces at all, but work together with the contact routines to record the forces generated by the actual contact interfaces. One-sided force transducers have been in the code for a long time, and are used to find the total contact force on a surface. But more recently, two-sided force transducers have become available. These offer more control, as they only report the forces due to contact between the specified slave and master surfaces, rather than the total of all contact forces. There is also a special “node based” two-sided force transducer (useful for eroding contacts): if the slave surface is given by a node set, then the master surface must be also. Using node sets allows for proper handling of the newly exposed surfaces as elements erode. In the

initial implementation, each node-segment contact that occurred was summed into at most one force transducer, and this is still the default behavior. It is expected that in normal use this will be fine. But some customers have created overlapping force transducers, and requested that each node-segment force be added to ALL the matching transducers. So the FTALL option was added to *CONTROL_CONTACT.
In the past some automakers, for example, would use hundreds of SURFACE_TO_SURFACE contacts in part because they wanted to be able to extract the forces between different portions of the model during a crash event. Even with GROUPABLE contact, this is not very efficient. It is always recommended to have as few contact interfaces as you can – only one for the whole model if possible. Then force transducer contacts can be used as needed to get any desired interaction forces.

TIEDID

The penalty based tied contacts use a simple spring model approach. The nearest point on the master surface (the “tie point”) is determined for each slave node, and a spring force is used to tie the slave node to the master surface at this point. For non-offset contacts, the slave node is first moved to the tie point. This may distort the initial geometry, but does not introduce any non-physical forces, because the spring length is initially zero. For offset contacts, the situation is a bit messier. The nearest point on the master surface is determined, and an offset distance is computed. The slave node is then tied not directly to the master surface, but to the end point of this offset vector. The offset vector is normal to the master surface at the nearest point on the surface, with length equal to the initial offset distance, and the tie point is the end point of this offset vector.

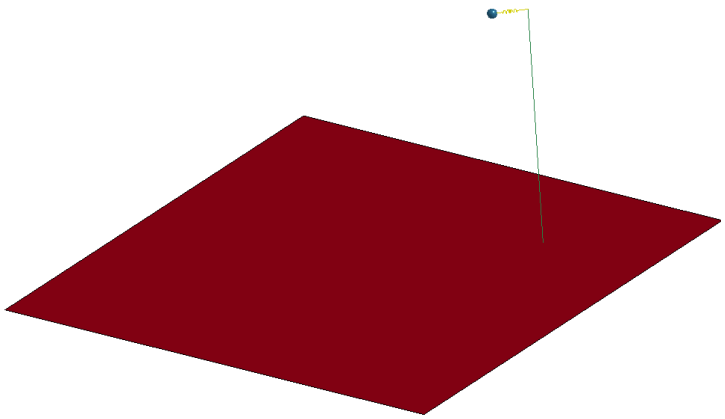

Fig.3: Tied Offset

If the offset distance is large, then small rotations of the master surface can result in large movements of the tie point, which can cause instabilities. Therefore, offset distances should always be reasonable, and preferably as small as possible.
Another problem that can arise is that the tie point may not exactly match the coordinates of the slave node, even in the initial configuration. This can happen due to warped or poorly formed elements which result in computational roundoff in the various calculations involved. When model coordinates are on the order of 2000 to 3000 mm and significant forces can be generated by penetrations as small as 0.01mm, any numerical inaccuracy can be a problem, particularly when running in single precision. This can result in tied interface forces being exerted even in the initial configuration, or in a model where there are neither loads nor movement. To eliminate this problem, the “tied incremental displacement“ option (TIEDID) was implemented. When this option is used, rather than computing the actual location of the end of the offset vector each cycle, and then using the difference between this point and the slave node, an incremental approach is used. The current displacement between the slave node and the tie point is initially set to 0, then updated each cycle based on the relative velocity of the slave node and the tie point, not their actual locations. The force exerted is then based on this incrementally updated displacement vector. In this way no contact foces are generated at time 0, and the problem of computing small displacements as differences in large coordinates is avoided.

This option is available in the MPP version for all penalty based *CONTACT_TIED contacts and also the *CONTACT_AUTOMATIC_TIEBREAK contacts, with these exceptions:

*CONTACT_TIED_SHELL_EDGE_TO_SURACE (the BEAM_OFFSET option is recommended, which already has a similar treatment)

*CONTACT_AUTOMATIC_TIEBREAK options > 5, which already use an incremental approach.

Turning on the TIEDID option for other tied contacts will also activate the GROUPABLE option for those contacts, as the incremental displacement feature is only supported for GROUPABLE contacts. The use of this feature is highly recommended.

IPBACK

In general, constraint based tied contacts should be preferred over penalty based tied contacts, as they more accurately enforce the tying condition and do not have stability issues. However, constraint based contacts cannot be used in explicit calculations when other conflicting constraints may be involved, such as prescribed motion or rigid bodies. This has led some customers to use only penalty based tied contacts, since these conflicting constraint issues do not arise. The IPBACK option is available for the constraint based tied contacts. It causes the automatic creation of a secondary, penalty based contact that has the same master and slave definitions. If a node can be tied with the constraint contact, it is skipped during the penalty contact. But any slave node in the constraint contact that cannot be tied due to constraint conflicts will be tied using the penalty contact. This extra contact is created during the keyword input phase, and will show up in the rcforc and other files as its own independent contact interface. If the original constraint interface has user id N, then the generated interface will have user id N+1 (if that id is not being used by another contact). Leaving the id N+1 available is recommended if using IPBACK, because it makes it easy to determine which is the automatically generated interface.
Contact options for the secondary interface are taken from the primary interface, and so it is reasonable to set options (like TIEDID or penalty stiffness scale factors) that do not apply to a constraint based contact. The contacts for which this option is available, and the type of the created interface are (*CONTACT_TIED omitted in each case):

```
NODES_TO_SURFACE
      → NODES_TO_SURFACE_OFFSET
SURFACE_TO_SURFACE
      → SURFACE_TO_SURFACE_OFFSET
SHELL_EDGE_TO_SURFACE
      → SHELL_EDGE_TO_SURFACE_BEAM_OFFSET
NODES_TO_SURFACE_CONSTRAINED_OFFSET
      → NODES_TO_SURFACE_OFFSET
SURFACE_TO_SURFACE_CONSTRAINED_OFFSET
      → SURFACE_TO_SURFACE_OFFSET
TIED_SHELL_EDGE_TO_SURFACE_CONSTRAINED_OFFSET
      → TIED_SHELL_EDGE_TO_SURFACE_BEAM_OFFSET
```

REGION

This option is input as field 8 of optional *CONTACT card E. It is used to define an “active region” for the contact. It does not limit or in any way alter the list of slave or master nodes or segments in the contact definition. It is used not to restrict the definition of the contact interface, but to limit the region in space where the contact is active. Nodes and segments that are outside of the given region are skipped in the contact calculation. But if at some point they enter the contact region, they become active again.
For example, suppose you have a long plate rolling across the top of a cylindrical roller. Defining a REGION as indicated by the rectangular box here would make sense.

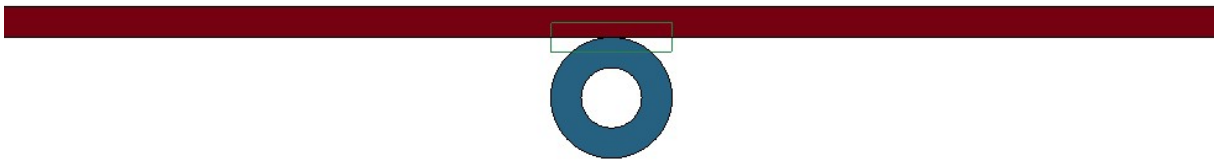

Fig.4: Plate on roller with REGION

The REGION acts as a pre-filter during the contact bucketsort, throwing out nodes and elements outside of the REGION. As such, it should not be too large – if model elements outside the REGION might come into contact before the next bucketsort, their contact would not be detected. But excluding portions that are currently far from the region of interest will speed up the bucketsort. Of course, portions of the model that will never come into contact should not be included in the contact definition.

SRNDE

This flag affects how the exterior edges of shell elements are treated during contact. The default treatment is that the free edges are rounded over by adding a cylindrical cap to the edge, with a radius equal to the shell thickness. As a result, the contact behavior will extend out past the edge of the shell element, which may be undesireable. If SRNDE is set to 1 in field 4 of optional card E, then the edge of the shell is rounded over, but not extended. If SRNDE is set to 2, then the edge of the contact element is square and ends at the edge of the shell element.

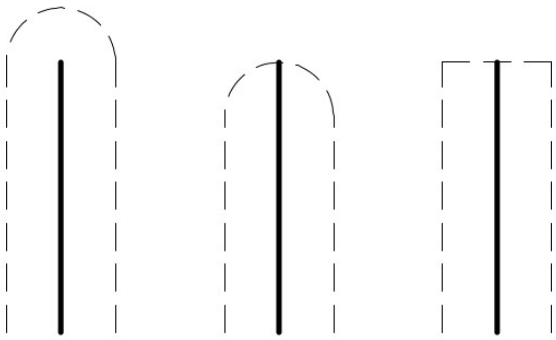

Fig.5: SRNDE=0, 1, and 2

This option is avaliable in the AUTOMATIC_SINGLE_SURFACE and AUTOMATIC_*_TO_SURFACE contacts. There is a small performance penalty when using this option, due to the extra calculation required to get the desired behavior.

FTORQ

This option only applies to the beams in *CONTACT_AUTOMATIC_GENERAL. When friction is computed between two beams in contact, the frictional forces are distributed to the beam nodes. But the actual contact point is at the surface of the two beams. In certain problems, the torque introduced by this discrepancy can result in undesirable rotations. If this option is enabled by setting field 7 of optional card E to 1, then compensatory torques are applied to the beam nodes, resulting in improved behavior.

Summary

As part of the ongoing development and improvement to the contact capabilities of LS-DYNA, new features and options will continue to be added. When helpful new options are implemented, they are often not enabled by default, due to the desire to preserve the behavior of pre-existing user models. In order to achieve the best behavior, the following settings mentioned in this paper are suggested where applicable:

IGNORE=2
GROUPABLE=1
TIEDID=1