A critique of Dallwitz's 'A comparison of interactive identification programs'

June 2000

K. Thiele and M.J. Dallwitz

This is the text of a posting to the DELTA Software Discussion List (DELTA-L) on 26 June 2000. At Theile's request, it was placed on the Web and linked to the document being discussed.

For the response by Dallwitz (also posted on DELTA-L), see below.

Mike Dallwitz recently published a document (Dallwitz 2000) comparing seven interactive-key programs. Two of the programs compared were Mike's Intkey (Dallwitz, Paine and Zurcher 1995) and the Lucid Player (Anonymous 1999).

Mike's comparison is based on a partial list of essential, desirable and undesirable features of interactive keys, with the programs scored as to whether or not the feature is implemented and whether implementation is, in Mike's opinion, satisfactory or unsatisfactory. The scores are then used to produce a global, weighted score for each program that purports to show the order of relative merit of the programs.

Recent conversations show that Mike's document is being read by some people as a serious and objective comparison. This is a worry, and a short response becomes necessary.

Mike is the principal designer of Intkey, one of the programs being compared. In this circumstance it's no surprise that the comparison is not objective, any more than if one vacuum-cleaner salesman were to assess the products of another. It will be impossible in this short communication to point out all the flaws in the comparison, but hopefully a few will suffice to show that it should be taken with a grain of salt, and people should either seek independent assessments from someone with extensive experience of several programs but close links to none, or better still, compare the main programs themselves.

There are four principal areas of bias with Mike's analysis.

Firstly, and most simply, he is very accurate for the features that Intkey supports, but not very accurate for the other programs (and this inaccuracy always works to the other program's disadvantage). This is understandable, and trivial.

Example: Mike reports that Lucid does not support 'Fixing character values'. It does (through the 'Save Session' command). Further, Lucid has the advantage that it can save these fixed values from one session to the next so that, for instance, every time a key is started it answers one or more predefined characters. Intkey may be deficient in this respect (I haven't checked - my knowledge of Intkey is about on a par with Mike's of Lucid).

Secondly, Mike's list is not a list of all desirable features of a good interactive key program. A quick scan of the scores for Intkey in the table shows that Intkey supports virtually all of the listed features: only 6 of the 71 items are not implemented in Intkey. This does not mean that Intkey is by far the best program, as asserted, just that the list used is principally a list of Intkey features (with a few extra items thrown in to give the impression of objectivity). Given this, it's hardly surprising that Intkey ranks best.

Example: Lucid has a powerful feature called 'Prune Redundants'. This comes into useful play after several character states have been chosen and the list of taxa remaining has been narrowed. In this circumstance, some or many characters and character states become redundant (a state becomes redundant if choosing it would return all taxa or none; a character becomes redundant if all its states are redundant). This command, as the name suggests, prunes these characters and states from the list of

available characters, thereby ‘customising’ the character list and reducing the chance of ending up with no taxa remaining. Intkey doesn’t have this feature, and it does not appear in Mike’s list.

Thirdly, the weighting of listed features is idiosyncratic, and again favours Intkey because it is essentially weighted towards those features that Mike thinks are important.

Example: Why are the ability to score individual states as uncertain, and the feature ‘Best routes’, rated as unimportant? For some characters (such as geographical distribution) scoring uncertain states is essential, and many Lucid users find the Best routes feature very useful. Could the downweighting of these features have anything to do with their lack in Intkey?

Fourth, the determination as to whether a feature is ‘satisfactorily’ or ‘unsatisfactorily’ implemented in a program should be read as ‘implemented the way Mike likes to do it’ versus ‘implemented differently’.

Example: Lucid supports a significantly richer data structure than Intkey (note here that the data structures of the DELTA format and of Intkey need to be considered separately, as some data storable in the DELTA format are stripped out by TOINT). The principal difference here is that Intkey allows only presence/absence scores for a state for a taxon (ie an Intkey state x taxon matrix is filled only with 0’s and 1’s). Lucid allows that a multistate state can be scored not only as absent or present but also rarely present and ‘present by misinterpretation’.

This latter is particularly important as it allows the key builder to maintain integrity of data while at the same time pre-empting likely user mistakes through misinterpretation. For instance, if a key has a character ‘Petals present/absent’ and a taxon in that key is petal-less but has petal-like structures that a user would very likely misinterpret as petals, the taxon in a Lucid key can be scored ‘petals: absent’ AND ‘petals: present-by-misinterpretation’. In ‘identification’ mode such a taxon would be retained if a user chose ‘petals: present’. In ‘interrogation’ mode such a taxon would be discarded if the user requested a list of all taxa that truly have petals. Intkey cannot replicate this behaviour.

Mike includes this feature under a general listing ‘Special values for keys’, rated ‘+’ (hence not all that important in the weighted scoring). He then decides that Lucid’s implementation of this feature is unsatisfactory!, further downweighting Lucid’s score. Curiously, all our key builders think this is a tremendously important feature, and find Lucid’s implementation of it perfectly satisfactory. One of the few allowed features that Lucid supports but Intkey doesn’t, and Mike downweights its importance considerably.

For brevity I have included only one example from each class of bias. There are many more. And I won’t discuss at all the rather bizarre ‘click-list’ table.

In fairness to Mike, pre-release copies of his document were distributed for comment, including to me. I returned some comments pointing out the bias in Mike’s analysis and the reasons for it, but did not address every point of error at length. To do so would have taken considerable time and been pointless, given the essential pointlessness of the whole exercise. Clearly, Mike’s assessment of the relative merits of the various features of the various programs is inextricably linked with his position as principal designer of Intkey, and my alternate assessment would probably be no more nor less weighted by my involvement with Lucid.

I could produce an alternate analysis with a list of features determined and ranked based on my own personal preferences and biases, as Mike has done. I could then analyse each program according to how I think a program would best run, as Mike has done. Such an analysis would undoubtedly rank Lucid best, with Intkey perhaps a distant second. I would expect people to largely ignore the result. I trust that the same treatment will be afforded Mike’s analysis.

Response by Dallwitz

I’m grateful to Kevin for pointing out an error and an omission in my document ‘A comparison of interactive identification programs’ (Dallwitz 2000). Improved versions of this document and related ones are now available on the Web.

Correction

The document wrongly stated that Lucid does not support ‘Fixing character values’. This has been corrected.

Other changes

Kevin suggested another comparison criterion: ‘Prune redundants’. This has been added to the comparison as two separate criteria: ‘Removing redundant characters’ and ‘Removing redundant character values’. The former had been included under ‘Best characters’, and the latter is undesirable for identification (for reasons given in the new version of the document).

Methodology

Mike’s document is being read by some people as a serious and objective comparison. ... Mike is the principal designer of Intkey, one of the programs being compared. In this circumstance it’s no surprise that the comparison is not objective, any more than if one vacuum-cleaner salesman were to assess the products of another.

Most scientific publications are intended to ‘sell’ the ideas of their authors, but the style and the methods of the selling are usually different from those employed by vacuum-cleaner salesmen. The document cited above, ‘A comparison of interactive identification programs’ (Dallwitz 2000), and the accompanying document, ‘Principles of interactive keys’ (Dallwitz, Paine and Zurcher 2000), are certainly intended to be ‘serious and objective’ scientific/technical publications. They differ from typical vacuum-cleaner brochures in several respects.

1. The authors’ names are prominently displayed.
2. Many of the criteria used are discussed and justified, particularly where I thought that they might be controversial or poorly understood.
3. They contain references (links) to sources of information about competing products and ideas.
4. They contain comparative information about features *lacking* in our product.
5. They contain an offer to incorporate other opinions: ‘Comments on the criteria used, or on the evaluation of particular programs, are welcome. They will be incorporated in future versions of this document, either directly or by modifying the criteria or evaluations.’

Mike’s list is not a list of all desirable features of a good interactive key program.

It is intended to be, but no doubt I have missed some. I will incorporate these when they are pointed out to me. Kevin has been able to suggest only one additional feature, which was already partially included in another feature (see above).

The list of criteria against which the programs are evaluated is essentially a character list. As with any character list, it reflects the opinions of the author about how the characters should be defined, which ones should be included, and how they should be weighted (for some purposes). If others give their opinions about these characters and propose additional ones, it may be possible to arrive at a consensus for a character list for describing this ‘genus’ of programs.

A quick scan of the scores for Intkey in the table shows that Intkey supports virtually all of the listed features: only 6 of the 71 items are not implemented in Intkey. This does not mean that Intkey is by far the best program, as asserted, just that the list used is principally a list of Intkey features (with a few extra items thrown in to give the impression of objectivity).

Kevin's suggestion seems to be that most of the features are in the list *because* they are in Intkey. Actually, it is the other way around — they are in Intkey because they are in the list. The list contains the features that I consider important, and, naturally, we have tried to implement these features, subject to priorities, difficulty of implementation, and available resources. Most of the important features that are still lacking in Intkey have not yet been implemented because the current DELTA/Intkey data structures do not support them. The necessary data structures were described by Dallwitz, Paine, and Zurcher (1993a,b). Unfortunately, implementing these ideas with appropriate generality is not trivial. Our new DELTA Editor is the first step in that process.

The weighting of listed features is idiosyncratic, and favours Intkey because it is essentially weighted towards those features that Mike thinks are important.

The weighting certainly has a subjective component. My categories and weights are as follows:

Category	Weight	Description
+++	16	Essential for correct identification.
++	8	Very important for identification.
+	4	Important for identification.
No '+' or '–'	2	Useful for identification or for other purposes.
–	0	Undesirable for identification.

I don't think that the features weighted 2 are not important, just *less* important *for identification*. Some features that have been given this weight (such as data sharing with other applications, searching the taxon names, and the ability to run from a CD-ROM without installation) are very important for a variety of other reasons.

In any case, the weights don't affect the rankings of the programs much. For example, with the current weights, the scores of Intkey and Lucid are respectively 219 and 137 out of a possible 240; with equal weights they are 127 and 69 out of a possible 146. [These were the values at the time of writing. They are now slightly different because of changes in the criteria and the weights.]

However, I do think that the weights I have assigned are defensible, and I would welcome discussion of individual cases.

The determination as to whether a feature is 'satisfactorily' or 'unsatisfactorily' implemented in a program should be read as 'implemented the way Mike likes to do it' versus 'implemented differently'.

As with the weights, there may be a subjective component in these decisions, but most instances are quite clear cut, and reasons are always given. For example, the Lucid implementation is scored 'unsatisfactory' for 'Character dependencies' because dependency relationships in the information entered by the user are not checked (which can lead to incorrect results), and for 'Unlimited data size' because the number of states is limited to 15.

I would be happy to further discuss the reasons for particular scores, and modify them if necessary.

The criteria

Mike reports that Lucid does not support 'Fixing character values'. It does (through the 'Save Session' command).

I was aware that LucID could restore used character values on startup of a data set, but had overlooked the fact that the values can also be restored when an identification is restarted. I've corrected the comparison table.

Further, Lucid has the advantage that it can save these fixed values from one session to the next so that, for instance, every time a key is started it answers one or more predefined characters. Intkey may be deficient in this respect (I haven't checked — my knowledge of Intkey is about on a par with Mike's of Lucid).

This is treated in the comparison under 'Command files or macros. A mechanism for storing and repeating a series of operations.' Lucid's implementation is scored as 'unsatisfactory', with the

explanation ‘Limited to carrying out a single identification sequence’. (Intkey can execute *any* sequence of stored commands at any time.)

Lucid has a powerful feature called ‘Prune Redundants’. This comes into useful play after several character states have been chosen and the list of taxa remaining has been narrowed. In this circumstance, some or many characters and character states become redundant (a state becomes redundant if choosing it would return all taxa or none; a character becomes redundant if all its states are redundant). This command, as the name suggests, prunes these characters and states from the list of available characters, thereby ‘customizing’ the character list and reducing the chance of ending up with no taxa remaining. Intkey doesn’t have this feature, and it does not appear in Mike’s list.

Removal of redundant characters can and should happen automatically as part of a ‘Best characters’ calculation, and I did not think that a separate treatment was necessary. However, I’ve now added this as a separate feature.

Removal of redundant character states is the only significant feature of Richard Pankhurst’s Online (from which Intkey was derived) that has not been implemented in Intkey. It has not been implemented because I consider it an undesirable feature. However, it should have been included in the comparison. This omission has now been rectified, and the reasons why the feature is undesirable have been given.

Why are the ability to score individual states as uncertain, and the feature ‘Best routes’, rated as unimportant [i.e. given the default weight]*? For some characters (such as geographical distribution) scoring uncertain states is essential, and many Lucid users find the Best routes feature very useful. Could the downweighting of these features have anything to do with their lack in Intkey?*

The weights are justified in the notes on these topics.

The note for ‘Unknown state values’ reads (in part):

> [Usually] the observed values are recorded against the taxon, and it is assumed that the other values are not present in the taxon. Although this is usually satisfactory, it should also be possible to record as ‘unknown’ those states that have not actually been observed. This is particularly important for characters such as geographical distribution, and the host range of parasites.

Note that ‘particularly important’ refers to general importance, not importance for identification. Even in the general context, I doubt whether most taxonomists regard it as ‘essential’ for geographical distributions — it’s not often done in taxonomic papers.

The note for ‘Best routes’ reads:

> Paths, similar to conventional keys, embedded in the interactive key. A path may be followed from the start of an identification; after it is left, ordinary interactive identification is resumed. This provides some guidance in the choice of characters. The method is inherently much less flexible than ‘Best characters’, and this limits its usefulness.

I don’t doubt that ‘many Lucid users find the Best routes feature very useful’, because the implementation of the preferable ‘Best characters’ feature in Lucid is very poor, and so slow as to be effectively unusable at the start of an identification in moderately sized data sets. This does not make the feature important per se. [‘Best routes’ (also known as ‘expert routes’) were not implemented in Lucid Version 3.]

Lucid allows that a multistate state can be scored not only as absent or present but also rarely present and ‘present by misinterpretation’. This latter is particularly important as it allows the key builder to maintain integrity of data while at the same time pre-empting likely user mistakes through misinterpretation. ... [This feature is] rated ‘+’ (hence not all that important in the weighted scoring).

This feature is very important for information retrieval (‘integrity of data’), but less so for identification. Treating ‘present by misinterpretation’ as ‘absent’ leads to quicker identification, and is therefore a useful option for users who are confident that they will not misinterpret the characters. In the absence of this feature, an author can record states as ‘present’ if they might be interpreted as such, particularly if the data are intended primarily for identification.

The rating should be judged in relation to the ratings of other features. Is this feature more important (for identification) than the following, which are all rated '+': character deletion and changing; locating errors; separating a taxon; character reliabilities; character dependencies; etc? Almost certainly not. Is it even *as* important as these features? Probably not, but I rated it '+' because of a conscious effort to give the benefit of the doubt to features possessed by other programs but lacking in Intkey.

Only 5 features are rated '++' ('very important for identification'): unrestricted character use; error tolerance; expressing uncertainty; numeric characters; and best characters.

A generalized version of 'present by misinterpretation' is among the proposed enhancements to DELTA (Dallwitz, Paine and Zurcher 1993a,b), primarily because of its importance in information retrieval and classification.

User interface

I won't discuss at all the rather bizarre 'click-list' table.

The table referred to shows the number of operations (mainly mouse clicks or double clicks) required for various components of an identification. This is a measure of the difficulty of the user interface for carrying out an identification. It is not the only measure, nor necessarily the best one, but it is objective and fairly easily obtained.

Kevin's objection to the table is presumably that it is too simplistic: there is more to the user-friendliness of an interface than just the number of operations. I agree. However, the table is only a summary. Full descriptions of the processes are also given, and should be consulted by those who would like to compare the interfaces in more depth.

References

Anonymous. 1999 onwards. Lucid home page. http://lucidcentral.org

Dallwitz, M.J., Paine, T.A., and Zurcher, E.J. 1993a. Preliminary suggestions for new features for the DELTA system. *DELTA Newsletter* 9: 2–13.

Dallwitz, M.J., Paine, T.A. and Zurcher, E.J. 1993b onwards. Proposed new features for the DELTA System. http://delta-intkey.com

Dallwitz, M.J., Paine, T.A., and Zurcher, E.J. 1995 onwards. User's guide to Intkey: a program for interactive identification and information retrieval. http://delta-intkey.com

Dallwitz, M.J. 2000 onwards. A comparison of interactive identification programs. http://delta-intkey.com

Dallwitz, M.J., Paine, T.A. and Zurcher, E.J. 2000 onwards. Principles of interactive keys. http://delta-intkey.com