A user's guide to `methsearch` and associated tools

Richard A. Smith
richard@ex-parrot.com

21st March, 2012

Contents

Chapter 1

Getting started

This chapter gives a brief introduction to `methsearch` — what it is, what it can do, and who it's aimed at (§1.1). This is followed by an introduction to the command line (§1.2), aimed particularly at those unfamiliar with it; it can safely be skipped by those familiar with Unix-style command line interfaces. Finally, are a slightly non-trivial worked example (§1.3), details of how to obtain `methsearch` (§1.4), and its current development status (§1.5).

1.1 What is it?

`methsearch` is a tool for searching for bell-ringing methods. If you're not a bell ringer or don't know what a method is, `methsearch` almost certainly isn't for you. Even if you are a ringer and do know what a method is, unless you have an active interest in method construction or composition, this tool is unlikely to be of interest.

`methsearch` is designed to exhaustively search for methods satisfying some particular criteria. For example, the 'standard' 41 surprise minor methods are the only methods that satisfy a particular set of criteria. In this case, it's possible to verify this with pencil and paper, and indeed this was first discovered a hundred or more years ago. But `methsearch` allows you to verify this in a fraction of a second.

A comparable list of major methods runs to tens of millions of entries — clearly beyond the scope of manual analysis, but on a reasonably fast computer, `methsearch` can enumerate them in a few hours and provide you with statistical information on them.

Perhaps you're looking for an interesting new method to ring? `methsearch` can help here too, but first you need an idea of what you want. There is no magic option that says 'find me the best possible method'. Computers a good at answering quantitative questions, not qualitative ones. Only after 'best' has been defined precisely enough that a computer can understand it can `methsearch` help you find your method.

`methsearch` can also be helpful in finding methods to drop into a composition. Want to replace Stonebow in Hull's 23 spliced major or Ariel in Pipe's cyclic 6 spliced maximus? Maybe `methsearch` can help, but to find a good replacement you'll need a good understanding of the problem.

Above all, `methsearch` is a program designed to help clever people do sophisticated things. If you have little or no understanding of method construction or composition, it's unlikely to allow you to find a killer method or composition overnight.

1.2 Command line interface

`methsearch` is a command line utility. What does that mean? Well, for one thing, it means that there is no whizzy graphical user interface. Instead it is controlled by a sequence of often-cryptic commands and options typed at your system's *command prompt* or *terminal*. To some users this will be a totally alien concept.

1.2.1 Using `methsearch` on Windows

It is rare for most Windows users to encounter tools that require the use of the command line interface, however Windows does provide a terminal that allows you to use `methsearch`. To start the terminal, on the 'Start' menu, choose 'Run' and then type in '`cmd.exe`'.[1] You should then see a new window, probably black with a white font, containing something like the following. (The exact details will depend on your version of Windows.)

```
Microsoft Windows [Version 6.0.6000]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\Users\Richard>
```

[1] On Windows 95, 98 or ME, this will be '`command.com`'.

The last line is called the prompt and indicates that the terminal is waiting for you to type something in; this should always end in a '>' character. First you need to change directory into the directory that contains your copy of `methsearch.exe` (and any associated tools and libraries you may also have got). If you've put it in a directory called '`C:\Documents and Settings\Richard\methsearch`', for example, then you should type:

```
cd C:\Documents and Settings\Richard\methsearch
```

In some versions of Windows, you can avoid this by starting the command prompt in the right directory by right clicking on the folder and choosing 'Open Command Prompt Here' on the context menu. You can now check that `methsearch` is accessible by typing:

```
methsearch --version
```

After pressing enter, this should print some brief information about the version of `methsearch` that you are running.

1.2.2 The Unix philosophy

Although the command line interface originally developed through necessity — early computers had no facilities for graphical output — its use has continued thanks largely to the *Unix philosophy*. This is a set of cultural norms for programs that can be summarised as follows:

— Write programs that do one thing and do it well.

— Write programs to work together.

— Write programs to handle text streams, because that is a universal interface.

The result is programs that are powerful and flexible even if this is at the cost of being less user-friendly.[2] `methsearch` attempts to conform to these ideals. As a result, `methsearch` is designed to search for methods and filter the results: it is not designed to search or prove compositions containing those methods, although for reasons of efficiency it has limited facilities for this. Other programs exist for searching and proving compositions and `methsearch` does not attempt to duplicate their functionality.

[2]Or as one wag put it, "Unix is user-friendly. It just isn't promiscuous about which users it's friendly with."

1.2.3 Command line options

To control `methsearch`'s behaviour, you need to specify *command line options* which tell it what to look for and how to present the information it finds. For example, to obtain a list of plain minimus methods with double symmetry, you could type:

```
methsearch -b4 -d
```

In this example, there are two command line options: '`-b4`' and '`-d`'. The former tells `methsearch` to look for minimus methods (i.e. methods on four bells), the latter tells it to restrict its search to double methods. The `-d` option is a *boolean option*: either it's specified or it's not — it's not parametrised in anyway; for example, it doesn't make sense to ask for methods that are at least 40% double. By contrast, the `-b` option requires a *mandatory argument*: namely the '`4`' in this example. It is not useful to ask for methods that are on bells without saying how many bells there should be.

By convention, command line options can sometimes be merged into one argument. This can be confusing to the uninitiated as it results in a much more terse command invocations. The rule is that any number of boolean options can be concatenated, followed, optionally, by at most one parametrised options. For example, the following `methsearch` invocation searches for all plain minor methods that are double, have regular (i.e. Plain Bob) lead ends, and no adjacent places.

```
methsearch -b6 -drj
```

As before, the `-b` option specifies the number of bells. The last part of the command line, '`-drj`', is three separate options merged into a single argument: specified individually they would be written '`-d -r -j`'. The `-d` and `-r` options, meaning double and regular respectively, are both boolean options.

The `-j` option specifies that the method should have no adjacent places: in other words, the method place notation must be formed solely from the changes `x`, `14`, `16` and `36`. A `12` change would not be allowed because first and seconds place are adjacent. However, the `-j` option is more complicated as it has an *optional argument*. It is also possible to require no more than two adjacent places via `-j2`. This allows the `12` or `1256` changes, but not `1236`. Similarly, `-j3` allows `1236` but not `1234`. Because the `-j` option has an argument, albeit an optional one, if it is one of several options merged together on the command line, it must be last. Thus '`-drj`' is equivalent to '`-rdj`', but specifying '`-djr`' would result in an error as `-j` is not a boolean option.

Each command line option also has a *long form*. These all begin with two hyphens rather than one, and have long names that are intended to make their meaning more obvious to a casual reader instead of the normal single-character ones. The other difference is that when they have arguments, the argument is preceded by an '='. For example, '`-d`' can be written '`--double`', and '`-b4`' can be written '`--bells=4`'. Certain rarely-used options only have a long form: for example, the option to require regular (i.e. Plain Bob) half-leads is `--regular-hls` — there is no equivalent short form. Perhaps unsurprisingly, it is not possible to merge together the long forms of arguments in a way analogous to `-drj`.

1.2.4 Output and redirection

By default, `methsearch` prints the place notation and lead head of each matching method on the screen as it finds them. When very few methods are found, this is satisfactory as the results can be inspected manually. The two preceding examples were both designed to yield very few matching methods (three and one, respectively); for example, the result from the first example (plain minimus methods with double symmetry) was:

```
X.14.X.34.X.14.X.12         1423
34.14.12.14.12.14.34.14     1423
34.14.12.34.12.14.34.12     1342
```

A quick inspection confirms these to be Double Bob, Double Court and Double Canterbury as expected.

Should we need these results again, the search only took a fraction of a second to complete and so is easy to repeat. But what if the search produces lots of results or takes a long time to complete? Fortunately, it is possible to *redirect* output to a file where it can be examined in detail. This is handled by the terminal,[3] and in all popular terminals (Unix or Windows) has the syntax:

```
methsearch -b6 -sd > minor.txt
```

In this example we are looking for plain minor methods with both palindromic and double symmetry (`-s` and `-d` respectively) and saving the list of methods into a file called '`minor.txt`'. This file can then be opened and studied in your favourite text editor (for example, Notepad on Windows). Another strategy is to use a *pager* such as `more` to allow you to scroll through the output without saving

[3] Actually, it's handled by the *shell*, but this distinction is not important here.

it.[4] The usual syntax for this is with a *pipe* which feeds the output of one command (in this case `methsearch`) to the next command (the pager, `more`):

```
methsearch -b6 -sd | more
```

In most pagers, pressing either space or return will scroll down a page.

1.2.5 Help!

As with most command line utilities, you can obtain a brief summary of valid options to `methsearch` by using the `--help` option.

```
methsearch --help
```

This user's guide aims to provide more extensive documentation as well as some worked examples of how it can be used in real world situations. Don't be afraid to experiment. If you're unlucky, you might ask `methsearch` to perform a search that will never complete. The `-u` option provides a good way of monitoring how quickly the search is progressing. If the search looks like it is going to take too long, press Ctrl-C to abort it, and try again with additional constraints. Finally, if you're stuck, drop me an email: my email address is on the title page.

1.3 An example — doubles methods

The Central Council's 1980 *Doubles Collection* lists 177 methods with one plain hunt bell. In its words, this is an exhaustive list of

> ...all 177 methods that have one plain hunt bell and three or four leads in the plain course, which are symmetrical about the treble's lead and lie, and which are not bobbed leads of other methods.

Producing such a list is exactly the sort of task for which `methsearch` is ideally suited, but first we need to translate the problem into a form that methsearch can understand. On a Windows system, we would type:

[4] `more` only allows you to scroll forwards through the output, however it has the advantage of being present on most systems, including Windows. Other pagers, such as `less` allow you to scroll backwards too. `less` is present on many Unix systems, and a Windows version can be downloaded from `http://www.greenwoodsoftware.com/less/`.

```
methsearch -b5 -sAC -R"$q\t$D" -Q"$o>=3" -m"*(1|2|23)"
```

And on most Unix systems (including most Linux shells), the double quotes around the `-R` and `-Q` options' arguments *must* be replaced with single quotes;[5] it would be good practice to do it with `-m` option too:

```
methsearch -b5 -sAC -R'$q\t$D' -Q'$o>=3' -m'*(1|2|23)'
```

Note that the ' character in many of the examples here is the character you get by pressing the apostrophe key on your keyboard. Some systems will display it as a straight (symmetrical) quote, others as a curved close quote.

Here, `-b5` means doubles (i.e. on five bells), `-s` means symmetric about the treble's lead and lie, `-A` means include methods with fewer than four leads in the plain course, and `-C` means print the total number of methods found. The remaining options are more interesting. In both cases, their argument is enclosed in quotes; this is because otherwise the shell would interpret the `>` and `|` characters as redirection and a pipe as described in §1.2.4..

The `-R"$q\t$D"` option controls what information is printed about each method. The argument to the `-R` option is called a *format string* and is described in detail in §3.1. In this case, we request the place notation in a concise notation (`$q`) and the lead head code using the traditional codes used in the 1980 collection (`$D`). The remaining part of the argument, `\t`, instructs `methsearch` to place a *tab character* between the two pieces of information. Most terminals display this as some (varying) number of spaces; however, it has the advantage that, if the output is saved to a file, it can be loaded in a spreadsheet as *tab separated values* and the two values will appear in separate spreadsheet columns.

The `-Q"$o>=3"` option states that the number of leads in the plain course (`$o`) must be at least three. (Strictly speaking, the rubric stated "three or four leads in the plain course" which we would write `-Q"$o==3 || $o==4"`, but as plain courses with more than four leads are not possible, this is unnecessarily pedantic.) The argument to the `-Q` option is called an *expression* and is described in detail in §3.2.

The final `-m"*(1|2|23)"` specifies that the last change in the method must be a `1`, `125` or `123` change. The syntax to this option is documented in §2.6. This is because the only other possibility, a `145` change, corresponds to a bobbed lead of some other method. (In fact, the possibility exists that some method with a `145`

[5] This is because the shell *interpolates* text in double quotes, but not in single quotes, and will therefore interpret `$q`, `$D` and `$o` as a shell or environment variable, and run `methsearch` with the value of this variable (probably nothing) instead. If you see an error message saying `Binary operator ">=" needs first argument` this is likely the cause.

lead end might exist that is not the bobbed lead of some other valid method, for example because the other three lead end variants have some property that we consider to be undesirable.)

If we run the command, we expect to see a list of 177 methods. And ...we don't:

```
&3.1.2.1.2,1    E
&3.1.2.1.2,23   T
&3.1.2.1.2,2    K
&3.1.2.1.4,1    D
        ⋮
&5.4.5.3.34,23  U
&5.4.5.3.34,2   J
&5.4.5.3.5,23   R
&5.4.5.3.5,2    B

Found 258 methods
```

Why are there not 177 methods? The clue is in the very first one, `&3.1.2.1.2,1` — consider the path of the second. It starts by doing eight blows at lead. However, the Central Council don't allow such methods: "no bell shall make more than four consecutive blows in the same position in a plain course".[6] Adding a `-p4` option to the command line takes this into account and produces the desired list of 177 methods. When using `methsearch`, it is not uncommon to discover that it has not accounted for some such implicit assumption about the desired method, and it therefore finds too many methods.

1.4 Obtaining `methsearch`

One way to obtain `methsearch` is to download the source code and compile it yourself. Such things are easy under Linux where most distributions will provide the necessary tools as standard. However, Windows users will likely find this process harder. To compound matters, whilst the source code for `methsearch` should be portable to a whole range of different compilers and operating systems, life is rarely that straightforward, and in all likelihood if you're not using a combination that I've tested, you'll end up having to fix a handful of small problems.

I used to provide Windows binaries compiled with Microsoft's Visual C++ compiler which produced excellent binaries. Unfortunately, since 2007 I've no longer

[6] The Central Council decisions can found at `http://www.cccbr.org.uk/decisions/`. The quoted text is in decision (E)A.1(f).

had access to a Windows machine on which to compile these. Instead, I now compile Windows binaries on a Linux machine using a *cross compiler* with less satisfactory results. What's really needed is for someone with some programming knowledge and a Windows machine to take an interest in the project and (with my assistance, if necessary) provide Windows binaries.

Meantime, the current Windows binary of `methsearch`, the Windows binaries of the other associated tools (see §6), and the PDF version of this manual can be found at the following locations:

- `http://ex-parrot.com/~richard/methsearch/methsearch-win32.tar.gz`
- `http://ex-parrot.com/~richard/methsearch/ringingtools-win32.tar.gz`
- `http://ex-parrot.com/~richard/methsearch/methsearch.pdf`

1.4.1 Compiling from source on Linux

To compile from source, it is best to get the current version from CVS. Unlike the main releases, this does not come with a copy of the `configure` script; this can be generated using the `autoreconf` command. The package is then configured, compiled and installed in the standard way:

```
U=:pserver:anonymous:
H=ringing-lib.cvs.sourceforge.net
P=/cvsroot/ringing-lib
export CVSROOT=$U@$H:$P
unset U H P
cvs login
cvs -z3 co -P ringing-lib
unset CVSROOT
cd ringing-lib
autoreconf
./configure --prefix=$HOME
make
make install
```

This should have installed the Ringing Class Library and all of its associated programs (including `methsearch`) into your home directory. Depending how your system is configured, you may need to add this to your *path* so that the the shell can find methsearch. If, when trying to run `methsearch`, you see an error message saying '`command not found`', you probably need to run the following:[7]

```
export PATH=$PATH:$HOME/bin
```

[7] Or add it to your `~/.bashrc` so they are always available.

1.5 Development status

`methsearch` is a fairly mature piece of software and has been in active use since 2002, since which time a great many new features have been added. Some features will have been tested extensively in many different combinations and are likely to be bug-free. However, it's quite possible that some less-commonly used features may interact with each other in odd ways. If you ever find something which you believe to be a bug, do please email me.

`methsearch` is also a work in progress. It does not presently not support every possible type of method, or allow methods to be filtered in every possible way. Generally speaking, new functionality gets added as and when I want it or someone requests it. For example, at present it has very poor support of alliance, hybrid or treble place methods, and its support for principles is quite limited. If there is missing functionality that you would particularly like to use, drop me an email and I'll see what I can do about adding it.

Finally, `methsearch` is open source software. This means that anyone can download a copy of the source code and extend it.[8] For example, I have no great interest in writing a graphical user interface to `methsearch`, but if you're a programmer and want to, there is nothing to prevent you from doing so. All that is asked in return is that if you make any extensions publicly available, you also make the source code to them freely available.[9] I'm always happy to add such contributions to the Ringing Class Library.

[8] The source code for `methsearch` is included with the Ringing Class Library which can be downloaded from `http://ringing-lib.sourceforge.net/`. Note that releases of the library are made relatively infrequently and if you want a recent copy, it is best to check the code out from CVS.

[9] Specifically, `methsearch` is covered by the GNU General Public License, version 2 or above, which can be viewed at `http://www.gnu.org/licenses/gpl.html`.

Chapter 2

Command line options

This chapter contains a complete list of all of `methsearch`'s command line options together with documentation on their use and meaning. For convenience, the options have been divided into eight sections based on their purpose: specifying the method stage and class (§2.1), applying constraints to places made within the method (§2.2), constraining the lead ends and heads (§2.3), and half leads (§2.4), requiring specific symmetries (§2.5), partially specifying the method's place notation (§2.6), configuring `methsearch`'s output (§2.7), sampling methods randomly (§2.8), and other miscellaneous options (§2.9). Options can also be stored in response files (§2.10).

2.1 Stage and class

`-b`	`--bells=N`	The number of bells
`-G`	`--treble-dodges=N`	The number of times the treble dodges in each position
`-S`	`--surprise`	Require a surprise method
`-T`	`--treble-bob`	Require a treble bob method
	`--delight`	Require a delight method
	`--3rds-place-delight`	Require a thirds place delight method
	`--4ths-place-delight`	Require a fourths place delight method
`-Z`	`--treble-path=PATH`	Require the specified treble path
`-U`	`--hunts=N`	The number of hunt bells
`-n`	`--changes-per-lead=N`	The lead length

The `-b` option is the only option that is always required. Its argument is the number of bells and must be an integer between 2 and 33, inclusive. The lower limit is

simply because, trivially, there is only one possible piece of ringing of any given length on one bell, and I have not yet got around to coding all of the special cases required to make `methsearch` handle it. The upper limit is not a fundamental limitation in `methsearch` — rather it is a limit on what place notations can be displayed.[1]

By default, `methsearch` looks for plain methods. The `-G` option switches this to treble-dodging methods. Most commonly this will be `-G1` meaning methods where the treble dodges once in every dodging position. Higher values can be specified, for example `-G3` has the treble dodging three times in each position. `-G0` is valid and is equivalent to not specifying a `-G` option. Treble-dodging methods are only supported on even stages.

The `-S`, `-T` and `--delight` options require there to be at internal places made at every (for `-S`), none (for `-T`) or some but not all (for `--delight`) of the occasions when the treble moves between dodging position. These will usually be specified together with a `-G1` option in which case the effect is to limit the search to surprise, treble bob or delight methods, respectively, however they can be applied to even-stage plain methods too.[2] Both options are only supported on even stages.

The `--3rds-place-delight` and `--4ths-place-delight` options only apply to methods where the treble travels between lead and sixths place. Typically this will be a treble dodging minor method, but they can be used on higher stages in conjunction with `-Z-6` option. Where these options are permitted, all delight methods will either be characterised as 3rds or 4ths place delight.

The `-U` option specifies the number of coursing hunt bells required. For example, `-U2` will find twin-hunt methods. All of the hunt bells will be required to follow the same path — this means that Grandsire will be found with an appropriate `-U2` search, but a slow course method (which the Central Council also define as a twin-hunt method, albeit of a different type) will not be. With `-U`n, the hunt bells will always be the front n bells — thus, New Grandsire (that is, Grandsire rotated so that 1 and 3 hunt) will never be found; nor will such peculiarities as Bedfont Slow Course Doubles in which 1 and 5 plain hunt. The `-U` option can be used in conjunction with the `-G` option to find treble dodging methods with multiple hunt bells.

The `-U` option can also be used to find principles by specifying `-U0`, or just `-U` as the argument to `-U` is optional and 0 is the default. When this is done, the

[1] After 33 bells, the numbers 1–9, the symbols 0, E, T and the letters A–Z, excluding E, I, O, T and X have all been used.

[2] In particular, an `-SG0` search, on six bells at least, finds methods with a distinctly Double Court feel to them. This is hardly surprising as the Central Council's former court method class was close to being to plain as surprise is to treble dodging.

`-n`n option must also be used to specify length of the lead unless `methsearch` is running in filter mode in which case the `-n`n option can be omitted and its value, n, inferred from the length of the methods being filtered. The combination `-AU` is useful when filtering as it avoids the need to specifying details such as the lead length, number of hunt bells, or treble path. Searching for principles is not especially well optimised in the code, nor is it particularly well tested.

The `-Z` option is used to specify the path of the treble (or whichever bell is the primary hunt bell in methods with unusual treble paths). At present, the only treble paths supported by this option are little versions of other supported treble paths — i.e. little plain or treble dodging. The syntax `-Z`x`-`y specifies the range of places that the treble passes through — i.e. that the treble hunts (or treble-dodges if `-G` is specified) from place x to place y. The places should be specified by bell symbol (see §2.6) — thus a plain maximus method is equivalent to `-Z1-T` and not `-Z1-12`. If the lowest place, x, is lead, it can be omitted; likewise, if the highest place, y, is the back, it can be omitted. Thus `-Z-4` is a traditional treble-to-fourths-place little method, and `-Z-` is equivalent to no `-Z` option as it means front to back.

`methsearch` supports little methods in which the primary hunt bell does not reach lead. For example methods such as Very Little Bob Minor (in which the third hunts between thirds place and fourths) can be found with `-Z3-4`, and Double Bishopstoke Little Surprise Major (in which the third treble-dodges between thirds and sixths) can be found with `-G1 -Z3-6`.

2.1.1 Historical classes

`--strict-delight`	Delight methods (1905–69)
`--exercise`	Exercise methods (1955–69)
`--strict-exercise`	Exercise methods (1905–55)
`--pas-alla-tria`	‘Pas-alla-tria’ methods (1905–55)
`--pas-alla-tessera`	‘Pas-alla-tessera’ methods (1905–55)

Until 1969, the Central Council recognised a fourth class of treble dodging methods: namely exercise methods. At that time, a delight method had to have internal places at all but one occasion when the treble moves between dodging positions; methods with internal places at all but two or fewer occasions were classified as exercise methods. To find methods that would formerly have been classified as exercise, or as delight in this stricter sense, use the `--exercise` and `--strict-delight` options.

Similarly, until 1955, the exercise class was restricted to methods with internal places made at exactly all but two occasions when the treble moved between dodg-

ing positions. Methods with internal places at all but three and all but four occasions, respectively, were nominally called 'pas-alla-tria' and 'pas-alla-tessera',[3] though I'm not aware of any evidence that these class names were actually outside the writings of H. Law James. The `--strict-exercise`, `--pas-alla-tria` and `--pas-alla-tessera` options can be used to find these types of methods.

2.2 Internal structure

`-p`	`--blows-per-place[=N]`	At most `N` consecutive blows in one place
`-l`	`--places-per-change[=N]`	At most `N` places in any change
`-j`	`--max-adj-places[=N]`	At most `N` mutually adjacent places in any change
`-w`	`--right-place`	Require right place methods
`-f`	`--no-78s`	Disallow penultimate places above the treble
`-y`	`--symmetric-sections`	Require each treble section to be symmetric
	`--long-le-place`	Allow an exception to -pN across the lead-end

The Central Council requires methods (except minimus methods) to have no more than four consecutive blows in any one place. In practice, on six or more bells, it is common to want to limit this to two consecutive blows. The `-p`n option controls this by limiting the search to methods with at most n consecutive blows in any one place. Omitting n (i.e. just writing `-p`) is equivalent to specifying `-p2`. Omitting the option entirely allows an arbitrary number of consecutive blows in one place.

The `-l`n option can be used to restrict `methsearch` to methods involving changes with at most n places. (Note: this option uses the letter '`l`', not the digit '`1`'.) For example, on six bells it is common to want to avoid single changes, in which case `-l2` would be specified. This means that changes such as `1234` or `1236` will be avoided. If n is omitted (i.e. if just `-l` is specified), a value of `2` is assumed. Omitting the option entirely allows arbitrary changes.[4]

The `-j`n option is similar to the `-l`n option. It limits the number of adjacent places permitted to n. Thus, on six bells, with `-j1`, the method place notation must be formed solely from the changes `x`, `14`, `16` and `36`. A `12` change would not be

[3] These were sometimes spelt as one word: viz., 'pasallatria' and 'pasallatessera'. A variety of spellings can be found, including 'pasalltria' and 'pasallatessara'. They presumably derive from the Ancient Greek *πας αλλα τρια* and *πας αλλα τεσσερα* meaning 'all but three' and 'all but four'.

[4] Except that the *null change* – the change where no bells move – is still prohibited. Use `-Fx` if you want to allow this. See §4.1 for details.

allowed because first and seconds place are adjacent. Specifying `-j2` would allow changes such as `12` or `1256` changes, but not `1236` (as this has three adjacent places). Similarly, `-j3` allows `1236` but not `1234`. Omitting n is equivalent to specifying `-j1`. Note that the `-j` option applies to the lead-end and half-lead changes too — this means that no seconds place methods will ever be listed.

The `-w` option restricts the search to right place methods. On even stages the meaning of this is obvious — every other change, starting with the first one, must be a `x`. On odd stages, this is taken to mean no places below the treble in every other change starting with the first, and no places above the treble in every other change starting with the second. That might sound counter-intuitive when written down, but it yields the expected results. (For example, it makes Plain Bob Doubles right place but not Reverse Canterbury.)

The `-f` option is used to exclude any methods with penultimate places made above the treble. For example, in a major method this excludes all `78` changes (and related changes such as `1478`) except at the half-lead. Despite the name of the long form of this option, `--no-78s`, this option is not specific to eight bells. Penultimate places are often disliked because it makes it hard to get a touch, and impossible to get an extent, which doesn't involve the two tenors sounding backwards at the end of the row at backstroke. Traditionally, this is considered musically poor; however this is perhaps gradually changing. (Note that there is no guarantee even with `-f` that such rows can be avoided at backstroke in an extent: Carlisle Surprise Minor is an example of a method where `65`s at backstroke cannot be avoided.)

The `-y` option limits the search to methods which have symmetric work during each of the treble's dodges. This has no effect in a plain method, but in a treble dodging method (with one treble dodge per position), it limits each section to those of the form `a.b.a.c`. For example, a method starting `3-3.4` will be permitted, but one starting `34-3.4` will not. This option can be useful in reducing an otherwise unmanageable search to one of a feasible length.

The `--long-le-place` option prevents checking of the `-p`n option across the lead-end symmetry point. This allows up to $2n$ consecutive blows in one place across the lead-end, whilst prohibiting more than n blows anywhere else. Thus, Grandsire Minor is found with `-b6 -U2 -p2 --long-le-place`, though Reverse Grandsire is not.

To a limited degree, the `-m` option (§2.6) allows exception to be made restrictions imposed by the `-l`, `-j`, `-w` and `f` options.

2.3 Lead ends and lead heads

`-r`	`--regular`	Require regular (Plain Bob) lead heads
`-c`	`--cyclic`	Require cyclic lead heads
`-e`	`--restricted-le`	Require `12`, `1N`, `1` or `12N` lead end changes
`-E`	`--prefer-restricted-le`	Prefer `12`, `1N`, `1` or `12N` lead end changes
`-A`	`--all-methods`	Allow all lead heads, including differentials or differential hunters, slow course methods, and other methods with multiple hunt bells
	`--offset-cyclic`	Require a method that produces cyclic music when started from treble's snap

The term *lead end* is often used ambiguously to refer to (i) the row at handstroke when the treble is leading full, (ii) the following row, (iii) the change between these two rows. In this documentation, the *lead end* refers is used exclusively for the first, *lead head* for the second, and *lead end change* for the third.

The `-r` option tells `methsearch` to only look at methods which have regular lead heads. For a `-U`n search this means that the lead head (the backstroke when the treble is leading) must be a row that appears at backstroke when the back n bells hunt. For single- and twin-hunt methods this means a Plain Bob or Grandsire lead head, respectively. Note that, unless `-A` is additionally specified, regular differential lead heads are not permitted. Thus, royal methods with lead head `1860492735` will be found with `-Ar` but not with just `-r`.

The `-c` option is similar except that it requires cyclic lead heads — that is, a lead head where the working bells form a wrap of rounds. As with `-r`, differentials are not permitted unless `-A` is additionally specified, so a cyclic royal method with lead head `1567890234` will be found with `-Ac` but not with just `-c`. When used in conjunction with the `-k` option (specifying rotational symmetry), `methsearch` is able to reduce the search space considerably for single-hunt cyclic methods by knowing that half-lead must also be cyclic. The `-c` option may be specified when searching for principles with `-U0`.

The `-e` option limits the search to methods with a `12`, `1N`, `1` or `12N` lead end change (the former two on even stages, the latter two on odd stages). In the most common case of regular methods with palindromic symmetry (i.e. when `-rs` is specified), this is redundant.

The `-E` option is similar to `-e` except that methods with unorthodox lead end changes are only permitted if neither of the orthodox lead end changes produce acceptable methods. For example, if the seconds place variant would produce a one lead course (disallowed without `-A`) and the nth place variant would pro-

duce four blows behind (disallowed with `-p2`), a fourth place variant would be considered.[5] In determining whether the seconds and $n$ths place variants are acceptable, `methsearch` applies all options provided to `methsearch` *except* `-F` options (which govern falseness), `-Q` options (that apply a final layer of filtering to the result set) and `--start-at` (which allows a search to be restarted part-way through). This subtlety is subject to change in a future version of `methsearch`.

By default, `methsearch` requires all of the working bells (i.e. those not covered by a `-U`n option) to be in a single cycle performing the same work. In other words, differentials and differential hunters are never produced as they, by definition, have multiple sets of working bells; nor are methods with multiple hunt bells (except as permitted by `-U`n options). By specifying `-A`, all such methods are included. To include only certain additional types of method — for example, to include differentials, but not hunters or differential hunters when doing an `-U0` search, the `-Q` option should be used. Typically this will involve tests of the `$u` and/or `$o` variables.

The `--offset-cyclic` option is very specialised. It limits the search to methods that, when started from the backstroke snap (or last backstroke snap for `-G`n with $n > 1$) produce a cyclic row one lead further on. Spinning Jennie Delight Royal is an example of such a method. Because `--offset-cyclic` actually effects the row at the backstroke snap, it can be used in conjunction with other options governing the lead head; however, because the lead head and backstroke snap are so close together, doing so severely limits the opening few changes of the method.

2.4 Half leads

`--regular-hls`	Require regular half leads
`--cyclic-hle`	Require cyclic half lead ends
`--cyclic-hlh`	Require cyclic half lead heads
`--rev-cyclic-hle`	Require reverse cyclic half lead ends
`--rev-cyclic-hlh`	Require reverse cyclic half lead heads

For the purpose of this documentation, the row immediately before the half lead change (that is, the change when the treble is lying behind) is referred to the *half lead end* and the row immediately after is referred to as the *half lead head*.

The `--regular-hls` option specifies that the method must have regular half leads. This means that when the half lead end and the half lead head are writ-

[5] The primary motivation for this option is to allow the lists of methods from the Central Council's *Plain Minor Collection* and *Treble Dodging Minor Methods* to be reproduced. These collections distinguish fourths place methods that are bob leads of other methods from other fourths place methods. The `-E` option allows `methsearch` to make this distinction too.

ten backwards, they must be regular lead heads and lead ends or vice versa. For example, both Beverley and Surfleet Surprise Minor have regular half leads even though the two half lead rows come in different orders in the two methods. Cambridge Surprise Minor, however, does not have regular half leads, instead having completely different rows around the half lead. All regular palindromic double (`-rsd`) methods necessarily have regular half heads. If you want to use half-lead calls, it can be useful to have regular half leads.

The `--cyclic-hle` and `--cyclic-hlh` options specify that the half lead end or half lead head, respectively. should be descending cyclic row (e.g. `456231` for a single-hunt minor method). The `--rev-cyclic-hle` and `--rev-cyclic-hlh` options specify that the half lead end or half lead head, respectively, should be ascending cyclic row (e.g. 432651 for a single-hunt minor method).

2.5 Symmetry

`-s`	`--symmetric`	Require palindromic methods
`-d`	`--double`	Require glide symmetric methods
`-k`	`--rotational`	Require rotationally symmetric methods
	`--mirror`	Require mirror symmetric methods
	`--floating-sym`	Allow the symmetry points in arbitrary places

These options govern the symmetry that the method must have.[6]

Palindromic symmetry, specified with `-s`, is the most common form of symmetry. For methods with an odd number of hunt bells, the method must have a plane of symmetry through the lead end change as Plain Bob does. For plain methods with an even number of hunts, the plane of symmetry must be through the following change as with Grandsire. For the esoteric case of treble dodging methods with an even number of hunts, the symmetry plane must pass through the middle of the treble's 1–2 up dodge(s).

Rotational symmetry, specified with `-k`, means that the method is invariant under a rotation about the quarter lead. Brave New World Bob Royal is an example of a method with just this symmetry.

Glide symmetry, specified with `-d`, means that the method is invariant when reversed (i.e. when the frontwork move to the back and vice versa) and then shifted by half a lead. The Central Council refers to this form of symmetry as *double symmetry*, although this term is sometime incorrectly reserved to mean the com-

[6]The definitive mathematical treatment of symmetry in methods is given in a paper by Martin Bright which can be found at `http://www.boojum.org.uk/ringing/symmetry.pdf`.

bination of glide and palindromic symmetry. Methods with just glide symmetry are relatively rare;y rung; Double Resurrection Cyclic Bob Royal is an example of one.

Mirror symmetry is similar to glide symmetry, and is required by the `--mirror` option. It means that method is invariant when each change is reversed without a subsequent shift of half a lead. It is found in Mirror Bob Major. Mirror symmetry heavily restricts the possible changes available for producing a method: for example, on six bells, only `x`, `16`, `34` and `1256` are allowed. By the nature of the symmetry, mirror symmetric methods with a hunt bell will always have a second hunt bell following the same path but half a lead out of sync; as a result `-A` will always need to be specified too. Under these definitions, Original on even stages has mirror symmetry rather than glide symmetry. This is because shifting the method by half a lead (one change) moves the cross changes to the `1N` changes.

For methods with hunt bells, the `-s` and `-k` options prescribe the symmetry points of the method. This means that it is, exceptionally, possible for an `-s` or `-k` search to omit a method with palindromic or rotational symmetry, but where the symmetry point is in an unorthodox position. Such a method can only exist if there is one (or more) additional hunt bells following the same path as the treble but that is not explicitly required by a `-U` option — this means that this possibility can only occur if `-A` is specified too.

When searching for principles (that is, with `-U`), the planes of symmetry (for `-s`) are normally fixed at the half-lead and lead-end, and or point of rotational symmetry (for `-k`) is normally fixed at the quarter and three-quarter leads. The `--floating-sym` allows to be at arbitrary points; however, it makes the search *much* slower as the requirement for floating symmetry point is enforced by filtering the results of the search, instead of pruning the search tree during the search.

If the principle is to have an odd number of changes per lead, then the only meaningful symmetries are rotational and mirror symmetry.[7] `methsearch` does not currently support rotation symmetry in methods of odd length.

If a method has any two of palindromic, rotational or glide symmetry, it necessarily has the third one. It is therefore unnecessary, though harmless, to specify all three of `-skd`.

[7] Palindromic symmetry is possible in theory, but only in trivially false methods.

2.6 Place notation

`-m`	`--mask=MASK`	Specify part of the place notation
	`--prefix=PN`	Specify the first part of the place notation
	`--start-at=PN`	Resume a search from a given place notation
	`--changes=LIST`	Allow only changes from the list

The `-m` option provides a powerful mechanism for specifying part of the method's place notation. Its argument is a *method mask*. At its simplest, a method mask can just be the complete place notation for a method which restricts `methsearch` to looking at just that one method.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	2	3	4	5	6	7	8	9	0	E	T	A	B	C	D	F	G

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
H	J	K	L	M	N	P	Q	R	S	U	V	W	Y	Z

Table 2.1: The default set of bell symbols.

The *place notation* can be expressed in any standard form. Bells are represented using the case-insensitive alphabet shown in table 2.1.[8] External places may be stated explicitly or left implicit, the cross change can be denoted `x`, `X` or `-` as preferred,[9] and changes are separated using a `.` which may be omitted next to cross changes. Symmetric blocks of place notation are prefixed with a `&` to save repetition, asymmetric blocks may (though needn't) be prefixed with a `+`, and blocks are separated by a `,`. For example, the place notation for Plain Bob Minor can be written most concisely as `&-1-1-1,2` or in full as `X.16.X.16.X.16.X.16.X.16.X.12`.[10]

In a method mask, a specific change can be left unspecified by writing `?` in its place; a sequence of several consecutive unspecified changes can be written `*`. A method mask can have at most one `*` because otherwise `methsearch` would be unable to calculate how many changes are represented by each `*`.

[8] This alphabet can be overridden with the `$BELL_SYMBOLS` environment variable. For example, setting `BELL_SYMBOLS="123456789ABCDEF"` tells `methsearch` to use hexadecimal digits instead of the conventional symbols to represent bells (on up to 15 bells). The `$BELL_SYMBOLS` variable must be at least as long as the number of bells specified with `-b`. If the variable is entirely in upper case or entirely in lower case, then bell symbols are considered to be case insensitive; if it is mixed case, they are assumed to be case sensitive. The default alphabet is *not* compatible with the alphabet used in Abel on 20 or more bells as Abel uses `I` to represent bell number 20. The convention used by `methsearch` is the one used in the Central Council's method libraries.

[9] Part 1 of the Central Council's *Doubles Collection* uses `x` in a non-standard way to denote any single change in which only the treble moves. For example, it gives the place notation of the first half-lead of Huntley Place Doubles as `x.1.x.x.5`. This use is not supported in `methsearch`.

[10] These are the two variants produced by the `$q` and `$p` variables described in §3.1.1.

A change needn't be completely unspecified or fully specified: it is also possible to give a list of alternatives for a given change. The list is enclosed in `(...)` and the options are delimited by `|`. For example, `&(34|5).1.5.1.5,2` denotes Reverse Canterbury or Plain Bob Doubles, as well as two false methods produced by putting a `34` in one half and a `5` in the other half. This point is important: the `&` only states that the mask is symmetric, not that the method derived from the mask is symmetric — use `-s` for that.

Parentheses cannot be used to enclose longer fragments of place notation. For example, it is *not* possible to specify that a method must have either a Kent or an Oxford start by writing `&(-34-|34-34).*,?`.

Finally, a mask can specify separate mask for the above and below works. The above and below work masks are separated by a `/`. Within a mask, the `/` operator has higher precedence than the `,` operator — in other words, a`,` b `/` c means the mask a followed by the mask produced by combining b over c. For example, to specify a doubles method with either Plain Bob or Reverse Canterbury above the treble, you would write `&(34|5).1.5.1.5/*,2`.

When specifying masks on the command line, it is often necessary to enclose them in quotes. This is because certain symbols, particularly `&` and `|` have meanings that the shell might try to interpret. On Unix systems, this can usually be done with either single or double quotes; under the Windows command prompt, double quotes appear to be necessary:

```
methsearch -b5 -s -m"&(34|5).1.5.1.5/*,2"
```

When searching for a method with hunt bells (i.e. a search other than with `-U0`), the mask can override the requirements set by `w` and `-f` that the method is right-place or has no penultimate places name above the treble. Thus `-w -m"&3*,?"` will look for methods that are right place except for starting with a `3` place notation.

When searching for methods without hunt bells (i.e. with `-U0`), a greater range of options can be overriden. Currently they are `-w`, `-f`, `-l`, `-j`, `--changes`, `--mirror` and `-Fx` (insofar as it prohibits the null change).

The `--prefix` option specifies the place notation for the opening section of the method. This option is deprecated in favour of the `-m` option. It is orthogonal to `-m` meaning that if both `--prefix` and `-m` are specified then the method must satisfy both.

The `--start-at` option allows a search to resumed mid-way through. This can be used to resume a search that was cancelled earlier by passing the place notation

of the last method found before the search was cancelled. Restarting a random search is only meaningful if the same random seed is used throughout and passed to `--seed`.

The `--changes` option restricts the changes that `methsearch` will consider to those in the comma-separated list provided. For example, to find methods made solely using the `x`, `14` and `18` changes, you could write `--changes=x,14,18`. Alternatively, the `--changes` option can be used to prohibit certain changes. This is done by prefixing the list of changes with a `!`, so, for example, `--changes=!1234` will allow any changes except for `1234`. (In Unix shells, it will probably be necessary to quote the argument to prevent the shell from interpreting the `!`.)

This option can be useful for generating a method (and particularly principles) where the course forms a group. Any collection of changes will generate a group, though often the group is simply the whole extent, S_n (or the in-course half of it, A_n) which is of little interest. And the changes used in a method whose plain course is a group must be generate that group. So to find a method whose plain course is a group, we just need to identify the changes needed to generate the group and put them in a `--changes` argument. For example, Striking Minor forms a group of order 120 generated by `x`, `14` and `36`.[11] `methsearch` can identify other such principles with a search such as:

```
methsearch -b6 -U -n20 --changes=x,14,36 -p2 -s
```

The `--changes` and `--mask` options are orthogonal, and similarly both options are orthogonal to `methsearch`'s other options that effect the permitted changes such as `-l`, `-j` and `-f` (§2.2). For `methsearch` to find a method, it must satisfy all of the requirements. In particular, a `--mask` cannot be used to override these restrictions in one part of the method. This feature may be included in a future version of `methsearch`.

2.7 Output options

The options in the previous sections control what `methsearch` searches for. By contrast, the options here control what `methsearch` does when it has found a method.

[11] This group is of some interest. It is isomorphic to S_5, but is not conjugate to the standard representation of S_5 as the permutations of five bells. It is the dual of the standard representation of S_5 under the outer automorphism of S_6. Hudson's group is its even-parity subgroup.

`-q`	`--quiet`	Suppress output of methods as they are found
`-R`	`--format=FMT`	Format methods in the specified way
`-o`	`--out-file=FILE`	Write methods to the given file
`-O`	`--out-format=TYPE`	In conjunction with `-o` to specify the file format of `FILE`
`-L`	`--library=LIB`	Use method library to look up method names
`-H`	`--frequencies=FMT`	Count frequencies of method properties
`-C`	`--count`	Count the methods found
	`--raw-count`	A more concise version of `--count`
	`--node-count`	Count the search tree nodes visited
`-u`	`--status`	Keep a running status of progress
	`--status-freq=N`	Display the status every `N` nodes

`methsearch` has three main types of output: method output, statistical output, and counts. If several types of output are to be produced, they come in the order method, stats, counts, and each type is separated by a blank line.

Method output is when `methsearch` prints details of each matching method as it is found. This is enabled by default and can be suppressed with the `-q` option. The information that `methsearch` prints, together with the formatting of it, is controlled by the `-R` option which is discussed in §3.1.1. By default, `methsearch` prints just the methods' place notation and lead head — i.e. omitting the `-R` option is equivalent to specifying `-R$p\t$l` (unless `-I` is supplied in which case `-R$p\t$a` is assumed).

The `-o` option tells `methsearch` to write all method output (but not other types of output) into a file. This is different from using shell redirection with `>` as that redirects all standard output,[12] not just method output. One reason to favour `-o` over shell redirection is that when used with `-u`, shell redirection can result in very flickery status line. (This is due to technical limitations of the terminal and not a bug in `methsearch`.) The special combination `-o-` is equivalent to an omitted `-o` option.

The `-O` option can be used to further control what is written to the `-o` file. At present three arguments to `-O` are supported: `fmt`, the default, which does formatted output as specified by the `-R` option; `utf8`, which is like `fmt` but enables Unicode characters to be output in UTF-8; and `xml`, which tells `methsearch` to produce an XML file containing the methods.[13] It is possible to produce builds

[12]Command line utilities have two types of output: *standard output* and *standard error*, both of which are displayed together on the terminal. `methsearch` uses standard output for all of its output except error messages and the status messages from `-u`. Standard redirection with `>` only redirects standard output, not standard error.

[13]The XML conforms to the *schema* published by the online method database project which is documented at `http://methods.ringing.org/xml.html`. This schema is not the one current used by the Central Council who are using gratuitously bastardised version of it.

of `methsearch` without XML support, and, obviously, if you are using such a build, `-Oxml` will not be supported. If `-Oxml` is specified then a `-o` option must be supplied (and not `-o-`).

It is possible to use `-R` to get `methsearch` to print the name of methods it finds. To do this it needs access to one or more method libraries containing the relevant methods. These may be Central Council text format collections,[14] the antiquated MicroSiril library format,[15] or, if XML is supported, XML libraries. The `-L` option specifies the library file to load. If multiple library files are needed, for example one for surprise, one for delight, and a third for treble bob, multiple `-L` options can be supplied. If the filename is not an absolute path (i.e. if it does not begin with a `/` on Unix, or `\` or `X:\` for some `X` on Windows), and if the `$METHOD_LIBRARY_PATH` or `$METHLIBPATH` *environment variable*[16] is set to a colon-delimited list of directories, these directories are search in turn for the file. How an environment variable is set depends on your system. The three commands below are for Windows, most Unix-like systems, and finally the Unix csh.

```
set METHLIBPATH=C:\Documents and Settings\Richard\methodlibs
export METHLIBPATH=/home/richard/methodlibs
setenv METHLIBPATH /home/richard/methodlibs
```

Statistical output is for gathering basic statistics such as the number of methods with each lead head. It is controlled by the `-H` option which is discussed in §3.1.1. By default no statistical output is produced.

`methsearch` maintains two *counts* while it searches for methods. First it maintains a *method count* which is simply the number of matching methods. The `-C` option prints this value at the end of the search; the `--raw-count` option is similar except that it prints just the number without any associated verbiage. (The latter can be useful if another program is reading the output of `methsearch`.)

Secondly, it maintains a *node count*. This is the number of whole or partial methods that `methsearch` has considered while performing the search. This can be displayed with the `--node-count` option. Unlike the method count, this gives a good idea of the complexity of the search. In 2009, a fast computer performing a straightforward search should be able to do perhaps as many as 500,000 nodes/s; however as the complexity of the search increases, so its speed will decrease considerably.

[14] Available from `http://www.methods.org.uk/method-collections/`.

[15] MicroSiril libraries are not recommended as they have a number of problems, most notably not properly supporting methods with a space in their name (such as Plain Bob).

[16] The two variables are synonymous. If both are set, `$METHLIBPATH` is ignored.

When performing a very long search, especially one that finds few methods, it can be difficult to know how far through the search `methsearch` has got. The `-u` option tells `methsearch` to display a *status line* which is updated by default every 10,000 nodes to show the first 68 characters of the place notation of the current node.[17] The frequency with which this is displayed can be overridden with the `--status-freq` option.[18] This information is displayed on *standard error* meaning that it is not captured by shell redirection using `>`.

2.8 Random sampling

`--limit=N`	Terminate the search after finding `N` methods
`--random`	Randomise the order in which methods are listed
`--loop[=N]`	Repeatedly search for one random method, indefinitely or `N` times
`--seed=N`	Seed the random number generator with `N`
`--timeout=N`	Timeout the request after `N` seconds

The `--limit` option is used to terminate after a certain number of methods have been found. This can be useful if all you care about is whether any method exists with a particular set of properties. In this context, the combination `-q --limit=1 --raw-count` can be useful when `methsearch` is being invoked by a script. The output is just a `0` or `1` depending on whether the criteria can be satisfied.

The `--random` option tells `methsearch` to randomise the order in which it traverses the search space. It is important to note, however, that `methsearch` still does a depth-first search and it is only the order of the child nodes at each point in the search tree that is randomised.[19] This means that all methods beginning with a given place notation sequence will still be listed together, however the first method found will be fairly random.[20]

The `--loop=`n option is used to modify the behaviour of a random search so that it does a searches for a single method n times. In contrast to `--random --limit=`n

[17] The number of characters to display can be overridden with the `$COLUMNS` environment variable in which case `$COLUMNS`-12 characters are printed. Many shells set this variable automatically.

[18] This option also controls the frequency with which timeouts (§2.8) are tested.

[19] The reason for this choice of implementation is that a truly randomised search would require `methsearch` to store every single method it finds, which in turn would used a large amount of memory. If you wish to genuinely randomise the methods, the `-R` option to the standard `sort` utility will do this.

[20] Even the first method isn't selected with uniform probability from the set of possibles. This is because methods in a part of the search space with few matches are more likely to turn up. This means that trivial underwork variants are underrepresented.

which will find n distinct methods close together in the search space, `--random --loop=`n will restart the search after each method it finds. This means that methods will be distributed throughout the search space. However there is no guarantee that the n methods will be distinct (i.e. that there will be no repetition), but if the number of methods requested is much less than the total, it is likely. If the `--loop` option is given without an argument, the loop runs indefinitely.

The `--seed` option seeds the random number generator with the given value. If you are invoking `methsearch` with `--random` lots of times in a short period of time, it is important to use this because `methsearch` simply uses the current time (in seconds) to seed the random number generator.[21] Similarly, if you want the methods to occur in an arbitrary yet reproducible order, this is helpful.

The `--timeout=`n option tells `methsearch` to give up after n seconds on a search (or, with `--loop`, an interation of the search loop).[22] When searching for random methods from a search space with large regions in which no valid methods can exist, for example asymmetric surprise major, it is worth using `--timeout` to restart the search if it appears to be in such a region. In such a situation it is possible to combine both an arguementless `--loop` option and a `--limit=`n option. In this case, the `--loop` option ensures that each search iteration finds just one method, and so the methods are distributed throughout the search space, and the `--limit=`n option ensures that n methods are found. Had `--loop=`n been used, n search iteration would occur, but as some of these might have timed out, the number of methods found could be less than n.

As an example, the following command will generate 23 utterly structureless random treble dodging major methods:

```
methsearch -b8 -G1 --random --loop --timeout=1 --limit=23
```

[21] Frequently, the `--loop` option provides a better means of doing this. However, when required, in `bash` and some other popular Linux shells, the `$RANDOM` shell variable provides a good way of generating a random seed that can be recorded if necessary and passed to `methsearch`.

[22] The `--timeout` option will not kill a command invocation (§6.1) that is taking too long, and the accuracy of its timing is not high. It works by getting the time every 10000 search nodes and testing whether the specified number of seconds has elapsed. The frequency with which the timeout is tested is controlled by the `--status-freq` command.

2.9 Miscellaneous options

`-?`	`--help`	Print an help message and exit
`-V`	`--version`	Print version information and exit
`-I`	`--filter`	Run in filter mode on standard input
	`--filter-lib`	Run in filter mode on specified libraries
	`--invert-filter`	Invert filter so only non-matching methods are listed
`-Q`	`--require`	Apply an additional requirement to the search results
`-M`	`--music`	Configure how `methsearch` evaluates musicality
`-F`	`--falseness`	Configure how falseness is checked
`-P`	`--parity-hack`	Require an equal number of rows of each parity for each position of the treble

The `--help` option was mentioned in §1.2.5. It prints a summary of all the options that `methsearch` knows about.

The `--version` option prints the version number of `methsearch`. (In practice, this is less useful than it might be. Currently, formal releases of `methsearch` are made very infrequently, and most copies in use are development snapshots.)

The `-I` option is used to change `methsearch` from searching for methods *ab initio* to filtering a list of methods supplied on *standard input*. The input format is a list of methods, one per line, with the place notation first on the line followed, optionally, by additional text that is ignored.[23] Any ext after the place notation is called the *filter payload* and can be accessed by the `$a` method variable (§3.1.1). The output from methsearch `-R"$q ..."` is a suitable input. This is particularly useful if you want to do a lengthy search, but are not entirely sure how to restrict the results initially. Dump everything into one big file:

```
methsearch -bn -R"$q" [options] > methods.txt
```

...and then filter the output as many times as you want:

```
methsearch -bn -I [other options] < methods.txt
```

Generally, doing a search once and filtering it several times is much faster than repeating the search once per attempt to filter it.

[23] A UTF-8 byte order mark (BOM) is permitted, though not required, at the start of the first line. This has the hexadecimal encoding `EF BB BF`. Other Unicode formats (such as UTF-16 or UTF-32) are not permitted; nor, in general, are any non-ASCII-compatible character sets.

The `--filter-lib` option is the same, except that instead of reading methods from standard input, it reads methods from all the libraries specified with `-L` options. This is useful for finding named methods with particular properties.

The `--invert-filter` option is used in conjunction with either of the options that enable filter mode (`-I` or `--filter-lib`). It inverts the sense of the filtering so that `methsearch` outputs every method that does *not* match the search criteria. The method count (per `-C`, below) and any statistical output (§3.3) is similarly inverted.

The `-Q` provides a way of imposing more complex requirements on the methods that `methsearch` finds. Its argument is an *expression* whose is described in §3.2. The expression is evaluated for each method found, and only if it evaluates to true is the method found. Many other options can also be rewritten as a `-Q` option; almost invariably this results in a loss of efficiency, sometimes spectacularly so.

The `-M` and `-F` options configure how methsearch analyses the musicality and falseness of a method, respectively. Music is discussed in further detail in §5, while falseness is covered in §4.

The `-P` option is rather specialised. It imposes requirements on the parity structure of the method. For a conventional treble-dodging method, this option imposes a necessary (but not sufficient) criterion for the existence of a true bobs-only extent. This is particularly relevant in minor. This option is discussed further in §4.3.

2.10 Response files

`@`*filename* Import commands from the file

In addition to taking command line options on the command line, `methsearch` can read them from a file which is referenced on the command line by prefixing the filename with an `@`. Such files are called *response files*, and are fairly common on Windows where they can be used to work around deficiencies in the Windows shell. For example, suppose there's a file called `legal-methods` containing the following text:

```
-p4 -Q"$u < $B-$u"
```

This can be referenced on the command line, with the effect that the following two commands are equivalent:

```
methsearch -b6 -s @legal-methods
methsearch -b6 -sp4 -Q"$u < $B-$u"
```

In addition to the fact that this allows commonly-used groups of options to be saved with a meaningful name so they can be reused, this also makes it easier to write complex searches so that they can be run equally easily on Windows and on various Unix shells.

Options in response files can be quoted using either single or double quotes (' or ") — unlike in most popular shells, both forms of quotes are treated the same.[24] Thus in the example above, in most Unix shells, the quotes on the command line need replacing with single quotes, but in the response file, either can be used. Single quotes can be nested within double quotes, or vice versa; if an additional levels of quotes are needed, the innermost ones can be escaped from the response file by preceding them with a \, as in most shells.

[24] In particular, there is no need to escape a $ with a \ inside double quotes as there is in most Unix shells, although it is harmless to do so.

Chapter 3

Formatted output

The options that control output were listed in §2.7. This chapter starts by discussing the format strings (§3.1) that occur in the arguments to the `-R` and `-H` options and provide `methsearch`'s main way of communicating information about the methods back to the user. The expression syntax (§3.2) used by the `-Q` option is also documented here as it shares much in common with format strings. Finally, §3.3 looks specifically at the `-H` option for statistical output.

3.1 Format strings

The argument to the `-R` and `-H` options is a *format string*. This is a string which specifies the information to be printed when each method is found (in the case of the `-R` option) or in each line of the statistical output (in the case of the `-H` option). At its simplest, the format string might just contain literal text, for example, `-RFound` which would simply print '`Found`' each time a method was found. This, however, isn't very useful as it coveys no information about the method.

In addition to literal text, the format string may contain *variables* (§3.1.1), *character escape sequences* (§3.1.2), *command invocations* (§6.1) and *expressions* (§3.2).

3.1.1 Method variables

Variables are pieces of text beginning with a `$` that are replaced by a *method parameter* — a piece of information about the method.[1] For example, `-R$q` causes

[1] A `%` can be used as an alternative to `$` in variables, although not in expressions such as in the argument to `-Q`. This use of `%` is deprecated.

the place notation of each method to be printed out as they are found. Some variables allow (and a few require) an integer to be specified between the `$` and the variable letter. An example is `$12h` which prints the 12th change of the method.

It is usually necessary to enclose format strings in quotation marks. This is needed so that the shell does not try to interpret it itself. Spaces and the `$` are both prone to interpretation by the shell, and both of these commonly occur in format strings. On Windows, double quotes should be used, (e.g. `-R"$q"`); on Unix-like systems, single quotes are needed (e.g. `-R'$q'`) to prevent the shell from interpolating the `$` sign. See §1.3 for an example of this. The quote characters are never seen by `methsearch` — they exist solely to make sure that the options typed in on the command line reach `methsearch` in their intended form.

`$p`	The place notation (verbose format) — see §2.6
`$q`	The place notation (concise format)
`$Q`	As `$q` but without the lead end change
`$`n`h`	Change number n from the method ($n = 1$ for first change)
`$`n`r`	Row number n from the method ($n = 0$ for opening rounds)
`$l`	The lead head row
`$`*[*n*]*`L`	The number of changes per lead
`$`*[*n*]*`b`	Maximum consecutive blows in one place
`$`*[*n*]*`o`	The number of leads per course
`$`*[*n*]*`u`	The number of hunt bells
`$`*[*n*]*`B`	The number of bells
`$d`	The lead head code (modern symbol)
`$D`	The lead head code (traditional symbol)
`$y`	A string representing the symmetries of the method
`$n`	The method name (excluding class and stage)
`$N`	The method title (including class and stage)
`$C`	The class name, e.g. 'Treble Bob'
`$S`	The stage name, e.g. 'Royal'
`$`*[*n*]*`M`	The musical score — see §5
`$F`	The falseness groups — see §4.2.3
`$`n`P`	The path for bell n
`$O`	The coursing order (defined so the tenors course)
`$`*[*n*]*`s`	An index of the staticity of the method
`$`*[*n*]*`i`	The method identifier from the CC libraries
`$`*[*n*]*`a`	The filter payload — see §2.9
`$`*[*n*]*`#`	The number of that method
`$T`	The current time (formatted as hh:mm:ss)
`$`*[*n*]*`c`	The number of matching methods — see §3.3
`$`*[*n*]*`?`	The exit status of the last command invocation — see §6.1

In the above table, n stands for an integer which must be supplied. For example, in

a surprise major method, `-R$32h` will print the lead end change, but `-R$h` without a numeric argument would give an error message. It may seem illogical that the `$r` option uses `$0r` for the first row (that is, rounds), and `$h` uses `$1h` for the first change. This is because there are $n + 1$ relevant rows in the lead, but only n changes.

In most circumstances an offset, n, that's longer than the lead length will result in a error message when `methsearch` starts. The exception is when filtering over methods with variable length leads (e.g. with `-IAU` and no `-n`n option). In this case, an expression error (§3.2.3) results from an out-of-bounds access.

In the table, *[n]* means an optional integer. In each case, this is used as a *field width*. This is used to left-pad numbers with spaces up to the specified with in order to align numbers with different numbers of digits. This is most useful with `$c`. For example, `-H"$5c $y"` contains the number methods with each type of symmetry. The `5` tells `methsearch` to pad each count to five digits.[2]

A small number of variables can only be used in specific contexts. The place notation, name and identifier variables, `$p`, `$q`, `$Q`, `$n`, `$N` and `$i`, cannot be included in `-H` options. The `$c` variable can only be included in `-H` format strings, and the `$#` and `$T` variables can only be included in `-R` format strings.

Many of the method variables above are self explanatory; however a few bear further comment. Options such as `$b` (the maximum consecutive blows in one place) return the maximum number actually found in the particular method, rather than simply returning the value passed to the `-p` option. The number of hunt bells, `$u` includes any bell that is back in its starting place at the lead head.

The modern lead head code, `$d`, is the lead head as defined in modern Central Council collections and enumerated in table 3.1. It is a lower case letter `a`–`r` (excluding `i`); on nine or more bells, it may also have a numeric suffix. For compatibility with MicroSiril, the letter `z` is used to denote all lead head types without a code, including methods with regular (i.e. Plain Bob) lead heads, but with a short course, such as a royal method with lead head `1795038264`.

The traditional lead head code, `$D`, only applies on five or six bells. It is a capital letter, or `?` if no code applies; they are listed in table 3.2 for doubles[3] and table 3.3 for minor methods.[4] For minor, six possible fourths place lead heads are given

[2] C programmers may recognise that this is inspired by the `printf` format specifiers, for example `%d` versus `%5d`.

[3] The codes `A`–`K` (excluding `I`) and `Q`–`V` for doubles are defined in the Central Council's 1980 *Doubles Collection*. These are augmented by `M`, `N` and `W`–`Z` for fourths place methods which will be defined in the Central Council's forthcoming doubles collection.

[4] The codes `G`–`O` (excluding `I`) for regular minor methods are defined in the Central Council's

Minimus			Minor			Major			Royal			Maximus			Fourteen			Sixteen		
342	a	g	35264	a	g	3527486	a	g	352749608	a	g	3527496E8T0	a	g	3527496E8A0BT	a	g	3527496E8A0CTDB	a	g
			56342	b	h	5738264	b	h	573920486	b	h	57392E4T608	b	h	57392E4A6B8T0	b	h	57392E4A6C8D0BT	b	h
						7856342	c	j	795038264	z	z	795E3T20486	c	j	795E3A2B4T608	c	j	795E3A2C4D6B8T0	z	z
									907856342	c1	j1	9E7T5038264	c1	j1	9E7A5B3T20486	c1	j1	9E7A5C3D2B4T608	c1	j1
												ET907856342	c2	j2	EA9B7T5038264	c2	j2	EA9C7D5B3T20486	z	z
															ABET907856342	c3	j3	ACED9B7T5038264	z	z
																		CDABET907856342	c4	j4
																		DBCTA0E89674523	d4	k4
															BTA0E89674523	d3	k3	BTD0C8A6E492735	z	z
												T0E89674523	d2	k2	T0B8A6E492735	d2	k2	T0B8D6C4A2E3957	z	z
									089674523	d1	k1	08T6E492735	d1	k1	08T6B4A2E3957	d1	k1	08T6B4D2C3A5E79	d1	k1
						8674523	d	k	860492735	z	z	8604T2E3957	d	k	8604T2B3A5E79	d	k	8604T2B3D5C7A9E	z	z
			64523	e	l	6482735	e	l	648203957	e	l	648203T5E79	e	l	648203T5B7A9E	e	l	648203T5B7D9CEA	e	l
423	f	m	42635	f	m	4263857	f	m	426385079	f	m	42638507T9E	f	m	42638507T9BEA	f	m	42638507T9BEDAC	f	m
			25364	p	r	2537486	p	r	253749608	p	r	2537496E8T0	p	r	2537496E8A0BT	p	r	2537496E8A0CTDB	p	r
									297058364	p1	r1	297E5T30486	p1	r1	297E5A3B4T608	p1	r1	297E5A3C4D6B8T0	p1	r1
															2AEB9T7058364	p2	r2	2AEC9D7B5T30486	p2	r2
															2TB0A8E694735	q2	s2	2TB0D8C6A4E3957	q2	r2
									280694735	q1	s1	2806T4E3957	q1	s1	2806T4B3A5E79	q1	s1	2806T4B3D5C7A9E	q1	r1
			24635	q	s	2463857	q	s	246385079	q	s	24638507T9E	q	s	24638507T9BEA	q	s	24638507T9BEDAC	q	r

Doubles			Triples			Caters			Cinques			Sextuples			Septuples		
2453	a	g	246375	a	g	24638597	a	g	24638507E9	a	g	24638507T9AE	a	g	24638507T9BECA	a	g
			267453	b	h	26849375	b	h	268403E597	b	h	268403T5A7E9	b	h	268403T5B7C9AE	b	h
						28967453	c	j	2806E49375	z	z	2806T4A3E597	c	j	2806T4B3C5A7E9	c	j
									20E8967453	c1	j1	20T8A6E49375	c1	j1	20T8B6C4A3E597	c1	j1
												2TA0E8967453	c2	j2	2TB0C8A6E49375	c2	j2
															2BCTA0E8967453	c3	j3
															2CABET90785634	d3	k3
												2AET90785634	d2	k2	2AEC9B7T503846	d2	k2
									2E90785634	d1	k1	2E9A7T503846	d1	k1	2E9A7C5B3T4068	d1	k1
						29785634	d	k	297E503846	z	z	297E5A3T4068	d	k	297E5A3C4B6T80	d	k
			275634	e	l	27593846	e	l	27593E4068	e	l	27593E4A6T80	e	l	27593E4A6C8B0T	e	l
2534	f	m	253746	f	m	25374968	f	m	2537496E80	f	m	2537496E8A0T	f	m	2537496E8A0CTB	f	m
4253	p	r	426375	p	r	42638597	p	r	42638507E9	p	r	42638507T9AE	p	r	42638507T9BECA	p	r
						86947253	p1	r1	8604E29375	p1	r1	8604T2A3E597	p1	r1	8604T2B3C5A7E9	p1	r1
												T0A8E6947253	p1	r1	T0B8C6A4E29375	p1	r1
												EA9T7058624	q1	s1	EA9C7B5T302846	q1	r1
						79583624	q1	s1	795E302846	q1	s1	795E3A2T4068	q1	s1	795E3A2C4B6T80	q1	r1
3524	q	s	352746	q	s	35274968	q	s	3527496E80	q	s	3527496E8A0T	q	s	3527496E8A0CTB	q	r

Table 3.1: Modern lead head codes as shown by `$d` for methods on four to sixteen bells. For each lead head, the first code is for a seconds place lead end (or thirds place for twin hunt methods), and the second code is for an nth place lead end.

Greek letters[5] which are only displayed by `$D` if UTF-8 output is enabled with the `-Outf8` option.

13254 13524	A	14523 14253	B	12354 12534	G	13245 13425	H	14325 14235	J	12543 12453	K
		15342 13524	C	12435 14253	D	12543 15234	E	14325 13452	F		
15432 15423	Q	14523 14532	R	12435 12453	S	12543 12534	T	14325 14352	U	15342 15324	V
14523 15423	M	15432 14532	N	12435 14235	W	14325 13425	X	12543 15243	Y	13542 13542	Z

Table 3.2: Traditional lead head codes for doubles methods, as printed by `$D`, shown with their corresponding lead ends and lead heads.

2nds / 6ths			2nds		6ths		4ths		4ths	
135264	G	L	142563	P	152364	T	145362	X	162534	Y
156342	H	M	154632	S	165243	V	136524	μ	146325	Ψ
164523	J	N	165324	R	146532	U	152643	θ	154263	Σ
142635	K	O	136245	Q	134625	W	164235	δ	135642	λ

Table 3.3: Traditional lead head codes for minor methods, as printed by `$D`.

The symmetry code, `$y`, is a string containing some of the letters `P`, `M`, `G` and `R` standing for for palindromic, mirror, glide and rotation symmetry, respectively. See §2.5 for the definitions of these types of symmetry. For methods with none of these symmetries, the string is empty.

The method class string, `$C`, includes the tags 'differential' and 'little', even though the Central Council do not consider 'differential' technically to be a class. It does not include 'principle', 'treble-dodging' or 'plain' because these tags never appear in method names. (In the first case, an empty string is used; in the latter two, the classes are subdivided.) It uses the Central Council's current (2009) method classification — there is, at present, no support for older classification schemes, such as those that included the now-obsolete classes court, imperial, college and exercise.

In the terminology of the Central Council's decisions, `$N` contains the method's *title* and `$n` contains its *name*. The title, `$N`, includes all classes that should be included in the method name, even when the method is unnamed. Unnamed methods are assigned the name 'Untitled'. In the case of Little Bob (and only that case),

1961 *Collection of Minor Methods*. These are augmented by `P`–`Y` for irregular methods in Michael Foulds' *Spliced Treble Bob Minor* series of books,.

[5]These are μ, θ, δ, Σ, λ and Ψ which are defined in Anthony S. Bishop's *A Universal System for Extents of Treble Dodging Minor Methods*. It is unfortunate that a more systematic allocation of Greek lead head codes has not been adopted, but unless and until an alternative scheme comes into use, `methsearch` will use this one.

the name, `$n`, is the empty string; for all other methods, `"$N"` is equivalent to `"$n $C $S"`.

The path, `$`n`P`, contains the symbol for each place in the path of bell n during the lead (including both the initial lead head row and the final lead head row). For example, the path of the 2 in Plain Bob Doubles is `21123455434`. Although `$P`'s value looks like an integer, it should generally be treated as a string; for example, in an expression, it should be compared with the `eq` operator (§3.2.1) instead of `==`. This is because the `==` operator implicitly converts its arguments to integers, which will likely result in a silent expression error when the integer overflows.

The coursing order, `$O`, is defined as the place bell order raised to some power such that the two tenors are adjacent in the coursing order and the tenor is last. Most of the time this gives the obvious definition — for example, `7532468` for all regular major methods, but `7542368` for Single Canterbury. As with `$P`, the value of `$O` should generally be treated as a string rather than an integer.

The `$s` variable is a measure of how static the method is. Each consecutive blow that a pair of bells spends together beyond the first scores one point. This means that a point scores one point, a dodge two, an n-pull dodge $2n$ and so on. For example, Plain Bob Minor scores six points — two each for the 3–4 and 5–6 dodges, and two further for the seconds made over the treble which lasts as long as a dodge. By this measure, the most static plain method on all even stages is Double Oxford (jointly with a huge number of variations thereof); and unsurprisingly, Derwent Surprise Major is the most static treble dodging major method with a score of 104.

Ignoring falseness, symmetry, treble path and other such constraints, the static index describes the number of trivial variants that exist for a method — a method with staticity s has 2^s variants. (In case of Plain Bob, the dodges can be replaced with Reverse Canterbury–style places, or with three blows and a hunt or *vice versa* giving four possibilities in each dodging position; similarly the seconds over the treble could be replaced with four blows over each other, or with a point and a place or *vice versa*. This gives $2^6 = 64$ possibilities. Note that variants that alter the direction of bells leaving a piece of work are not counted — thus the difference between Kent and Oxford is not considered a trivial variant for this purpose.)

The method identifier, `$i`, is read from the method libraries. In the Central Council's libraries it is the column headed 'CCC'; it is the method's number in the printed method collections. It is of most use in plain doubles where these identifiers have become the standard shorthand for referring to doubles methods. Note that the lead head code is not included in this — thus to get 131B for Reverse Canterbury, the format string `$i$D` should be used.

The method number, `$#`, is simply a counter that is incremented for each method that is found. (Note that methods discarded using the `suppress` keyword (§3.2.4)

still get a number; this is the principle way in which `suppress` differs from `-Q`.)

3.1.2 Character escape sequences

Format strings may also include *character escape sequences*. These provide ways to include various *control characters* in `methsearch`'s output. Far and away the most common use is to include *tab characters* in the output. This is useful for producing data that a spreadsheet can read and is discussed in §1.3. The full list of character escape sequences is below, together with their ASCII values and meanings.

`\a`	0x07	BEL	alert (or bell) character
`\b`	0x08	BS	backspace character
`\f`	0x0c	FF	form feed character
`\n`	0x0a	LF	line feed character
`\r`	0x0d	CR	carriage return character
`\t`	0x09	HT	(horizontal) tab character
`\v`	0x0b	VT	vertical tab character
`\x`*NN*	0x*NN*		arbitrary character by hexadecimal character number
`\\`	0x5c		a literal \ character
`\$`	0x24		a literal $ character
`\%`	0x25		a literal % character

A \ at the very end of the format string end of the format string is used to suppress the line feed that is implicitly added to the end of a format string.

To print information spanning multiple lines, almost invariably the line feed (`\n`) should be used instead of the carriage return (`\r`).[6] For example, `-R"$q\n\t$N"` would print the place notation on one line followed by a line containing the name indented by one tab stop.

As a backslash introduces a character escape sequence, should you actually want to print a backslash, you need to double-up the backslashes. For example, to format a table in LaTeX, columns are separated by an `&` and rows are ended with two backslashes — `\\`. To get `methsearch` to produce output ready for LaTeX, you can use something like `-R"$q&$l&$N\\\\"`. (As always, Linux users would use single quotes instead of double quotes.)

For similar reasons, to include a literal $ or % in the output, these should be escaped with a backslash too.[7]

[6] On systems, such as Windows, where the standard line ending comprises a CR-LF pair, `methsearch` automatically inserts a CR (`\r`) before each LF (`\n`) that it encounters.

[7] In fact, a $ can be escaped with another $ or with a % too. Similarly with %. In other words, `\$`, `$$` and `%$` all produce the same output — a single $. All combinations involving a % (other than `\%`)

The bell character, backspace character, form feed, carriage return and vertical tab are unlikely to be of use, but are included for completeness. (It might be tempting to try to use `\a` to get alerted whenever a particularly good method is found. In practice, most modern terminals don't act on receiving a `\a`, and even if they did, it's easy to miss the alert.)

Hexadecimal escapes can be used to access additional functionality that your terminal might support. For example, most modern Unix-like terminals support VT102 terminal escapes allowing coloured output, underlining, and many other effects.[8] This can be combined with `methsearch`'s other features in quite sophisticated ways. For example, the following command looks for musical unrung surprise major methods, and highlight the most musical ones in bold red:[9]

```
methsearch -b8 -SG1 -srefj -p2 -M'<4-runs>'                    \
  -R'$[$M>35?"\x1b[1;31m":""]$2#)  $q \x1b[38G$d [$F]\x1b[0m'  \
  -Q'$M>30 && $n eq "Untitled"' -Lsurp8.txt
```

Understanding complex examples like this is typically harder than writing them. The first part of the `-R` option is `$[ $M>35 ? "\x1b[1;31m" : "" ]`. This is an expression saying if the music score is greater than 35, then print the string '`\x1b[1;31m`'. Here, `\x1b` is the ASCII *escape character*, and the whole string means put the console into bold mode and set the foreground colour to red. Similarly, the '`\x1b[0m`' at the end of the format string sets the console back to its default mode; '`\x1b[38G`' advances the cursor to column 38, in order to get nice alignment.[10]

At the time of writing, this example produces 21 interesting unrung methods. Many suffer from unfortunate incidence of O falseness, but #17 has an interesting line, a plain course with quite diverse music, and sane falseness:

```
&-3-4.5-5.36.4-4.5.4.36.4.3,1
```

For a mixture of historical reasons and niche use-cases, there are a few character escape sequences that are introduced by a `$` (or a `%`, though that is deprecated). They are `$$`, `$%`, `$)` which are alternative ways of producing a literal `$`, `%` or `)`.

are deprecated.

[8] Windows users wanting a terminal that supports such things might like to try Cygwin, a Unix emulation platform for Windows, which can be downloaded from `http://www.cygwin.com/`.

[9] The backslash at the very end of the line is the shell's *line continuation character*. If you type the whole command on a single line, you should ignore this backslash. Users of Unix-like shells also have the option of including it.

[10] The console codes available depends on your system, but some documentation can be found at `http://www.kernel.org/doc/man-pages/online/pages/man4/console_codes.4.html`.

3.2 Expressions

The `-Q` option provides a way of specifying more complex requirements that methods must satisfy. Its argument is an *expression* which must, when evaluated and converted to a boolean, must be true in order that the method is found. For example, `-Q"$M >= 40"` only prints methods where the music score (`$M`) is at least 40.

Expressions can also be included in format strings by including them in `$[...]`. In this case, the expression is evaluated and its value printed. For example, to normalise the music of methods with different length plain courses, the average music score per lead in the plain course could be printed with `-R"$[$M / $o]"`.[11]

Expressions are built up from operators (§3.2.1) which can act on literals (§3.2.2), variables (§3.1.1) and command invocations (§6.1). Finally there are a handful of special keywords that may be used in certain specific contexts (§3.2.4).

3.2.1 Operators

Valid expressions are formed from the following operators which are tabulated in precedence order, from highest precedence (at the top of the list) through to lowest precedence (at the bottom). Entries on the same line have equal precedence. The associativity of each precedence level is marked in the second column.

`* / %`	left	multiplication-type operators
`+ - .`	left	addition-type operators
`< > <= >= lt gt le ge`	left	comparison operators
`== != eq ne`	left	equality comparison operators
`~~`	left	pattern matching
`&&`	left	logical and
`||`	left	logical or
`?:`	right	the conditional (ternary) operator
`,`	left	the comma (sequence) operator

In expression involving two *binary operators* (that is, operators taking two arguments), for example, `1+2*3`, the order of invocation of the two operators can be made explicit by the use of parentheses. For example, `1+(2*3)` or `(1+2)*3`. In the absence of parentheses, the operators' *precedence* determines how tightly operators bind, relative to each other. In this example, `1+2*3` is equivalent to `1+(2*3)` because `*` has higher precedence than `+`.

[11] But note that because the `/` operator performs integer division, this will be rounded to the nearest integer.

When an parenthesesless expression involves two operators of equal precedence, the order of their invocation is determined by that precedence level's *associativity*. If the level is left-associative, the operators are evaluated from left to right; and if the level is right-associative, they are evaluated right-to-left. So, for example, `12/2*3` evaluates to 18, and not 2.

The operators `+`, `-`, `*` and `/` fulfil their usual mathematical roles of addition, subtraction, multiplication and division, respectively, with the caveat that they perform *integer arithmetic*. This is of particular relevance to division where the result is rounded to the next integer towards zero — so, `5/3` evaluates to 1.

The `%` operator is the modulus operator. It returns the remainder left from the integer division of its arguments. For example, `8%3` evaluates to 2. It can be defined more mathematically in terms of integer division by the equation:

$$a \texttt{ / } b \texttt{*} b \texttt{ + } a \texttt{\%} b = a$$

Because integer division rounds towards zero rather than down, this means that when a is negative, the result of the modulus operator is also negative.[12]

The `.` operator does string concatenation. It simply returns its two arguments joined together. It is more commonly used on strings, but can also be used on integers too — for example, `1.1` evaluates to 11. (As `methsearch` does not support floating point numbers or any other form of non-integral numbers, there is no ambiguity between the the concatenation operator and the decimal point.[13])

The six arithmetic comparison operators, `<`, `>`, `<=`, `>=`, `==` and `!=` do an arithmetic comparison of two integers and evaluate to 1 (for true) or 0 (for false). The operators `lt`, `gt`, `le`, `ge`, `eq` and `ne` are the equivalent forms that do lexicographic comparisons of two strings. This means that while `11>2` is true, `11 gt 2` is false, as the two integers are simply treated as strings.

The `~~` operator does pattern matching on rows. The left hand argument is parsed as a row, and the right hand argument is parsed as a music pattern (§5.1). If the row matches the pattern, the expression evaluates to `1`; if not, it evaluates to `0`. If the arguments are not well-formed as a row an a pattern, respectively, an expression error (§3.2.2) occurs. For example, `$l ~~ "12*"` can be used to filter slow course methods.

[12] To those familiar with modular arithmetic, this may be surprising. But this is behaviour is typical in most programming languages that use integer arithmetic — for example, the C language, and languages derived from it, define the `%` operator in this way.

[13] A future version of `methsearch` might introduce place notation literals, but these will require `+` or `&` prefixes. This may result in some obscure incompatibilities, e.g. `3+3.1` currently evaluates to 93, but a future version may treat it as a place notation `3.1.3.1.3.1`. Inserting whitespace after the `+`, or around the `.` will force parsing as an integer.

The `&&` and `||` operators are the logical 'and' and logical 'or' operators. In the former case, it evaluates to true if both its arguments evaluate to true; and in the latter case, it evaluates to true if either (or both) of its arguments evaluate to true. Typically, though not always, the arguments will be comparisons, for example `$o>=3 || $u==1`. Both of these operators evaluate their left-hand argument first and *short circuit* based on the value of it — this means that if the result of the operator does not depend on value of the second argument, then the second argument is not evaluated. In practice this means that if the first argument to `||` is true or if the the first argument to `&&` is false, then the second is not evaluated. This is of particular importance when one of the arguments is a command invocation (§6.1), the evaluation of which is potentially very expensive. In such an example, by reordering the expression such that the command invocation is on right, it is often possible to get a significant speed improvement.

The penultimate operator is the *conditional operator*. It is a *ternary operator* — that is, an operator which takes three arguments.[14] Its syntactic form is a `?` b `:` c, where a, b and c are its arguments. The conditional operator evaluates its first argument. If the first argument evaluated to true, the second argument is evaluated and the value of the whole conditional operator expression is the value of the second argument. And if the first argument evaluated to false, the third argument is evaluated and the expression takes its value. The conditional operator never evaluates both its arguments. An example is `$M>=40 ? "*" : ""` which might be used to print an asterisk next to particularly musical methods.

Finally, the `,` operator evaluates its first argument and ignores its value. It then unconditionally evaluates its second argument, and the argument of the whole expression is that of its second argument. This is only of use when evaluating the first argument has some effect, for example, evaluating a `suppress` or `abort` keyword (§3.2.4) or invoking a command (§6.1).

Except for the conditional operator, all of `methsearch`'s supported operators are *binary operators* — i.e. they take two operators. `methsearch` does not currently support any unary operators, which results in some notable omissions. In particular, there is no unary minus — if you want to use a negative number you need to use a circumlocution such as `0-5`. Similarly there is no logical 'not' operator.[15] This is likely to be addressed in a future version of `methsearch`.

[14] Because it is the only ternary operator in most programming languages, the term 'conditional operator' is often treated as synonymous with 'ternary operator'.

[15] This can also be worked around with the expression `1-(`x`)`, which exploits the fact that true and false are represented as 1 and 0, respectively.

3.2.2 Literals

The simplest expressions are *literals* which come in two forms: integer literals and string literals. Integers are just written in the normal way using the digits `0`–`9`;[16] strings are enclosed in single or double quotes, e.g. `"Untitled"` or `'Untitled'`.

The use of quotes for string literals potentially conflicts with the use of quotes for quoting a command line option from the shell. Generally this is not a problem because the shell quotation can be done with one type of quotes, and the literal quoted with the other sort. However, there are circumstances when this luxury is not available, for example, because the whole shell command line is itself quoted in single quotes, and you need to use double quotes for both. To solve this, the inner two need escaping from the shell. This is done by prefixing them with a backslash: `-Q"$n ne \"Untitled\""`.

3.2.3 Type system

Expressions in `methsearch` are *weakly typed*. This means that strings are integers are converted freely as required. When an integer is converted to a string, the string value is simply the decimal representation of the integer with no leading zeros. Strings can be converted to integers too. If the string contains just digits `0`–`9` then the conversion occurs successfully;[17] otherwise an *expression error* occurs. This means that no further evaluation of the whole expression occurs; if the expression was in a format string, its value is set to `<ERROR>`; and if it was in a `-Q` option, it is considered to have evaluated to false. It is not possible to recover from an expression error within the expression. For example, `"a"-1 > 42 ? "b" : "c"` does not evaluate to either `"b"` or `"c"`. An expression error also occurs when division by zero is attempted, for example in `1/0`.

There is no specific boolean type — any non-zero integer is considered to mean true. The comparison operators and logical operators all return an integer which is either `0` (for false) or `1` (for true).

All variables (and command invocations) are considered to be strings, even those which are logically integers (such as `$o`). However, as strings are freely convertible to integers, this is rarely detectable.[18] `methsearch` does not currently have

[16] Neither hexadecimal nor octal integer literals are supported — in particular, `010` is the decimal number 10 written with an unnecessary leading zero, and not the decimal number 8 written in octal.

[17] In fact, a leading `+` or `-` is permitted too. This is another way of getting around the lack of a unary minus operator. In builds of `methsearch` on 32-bit systems, integer value of the string, i, must satisfy $-2^{31} \leq i \leq 2^{31} - 1$. A similar constraint with 2^{63} applies on 64-bit systems.

[18] This allows `methsearch` to treat `$c` and `$#` as unsigned 64-bit integers on many 32-bit systems.

specific types representing rows or changes. This is likely to change in a future version of `methsearch`, as it is likely that the expression mechanism described in this section will be unified with the row expression mechanism of §4.4.1. An upshot of this is that it is not possible to do row-level operations, such as permutation, within an expression.

3.2.4 Special keywords

`suppress`	Suppress the current method
`abort`	Abort the whole search

If the `suppress` is evaluated in an expression, then display of that method is suppressed. This keyword is different in number of ways to the `-Q` option — methods that are suppressed still have a `$#` number assigned, and are still included in any statistics (§3.3) and counted in the `-C` method count. If the `abort` is evaluated in an expression, then the method is not printed and the whole search is aborted.

In order to prevent these keywords from being evaluated unconditionally, they need to be used in conjunction with an operator that does short circuiting — namely, `&&`, `||` or `?:`.

3.3 Statistical output

The `-H` option provides a mechanism for gathering simple statistical information on methods. The option's argument is a format string (§3.1). For each method that is found, the format string is expanded, any variables (§3.1.1) except `$c`, expressions (§3.2) or command invocations (§6.1) are evaluated and their values substituted. The format string value is then stored and a count kept of how many times that string occurs. At the end search, the `$c` variable gets replaced with the number of times that string occurs.

A simple example of this is counting the number of methods of each class within the 'standard' 147 trebled dodging minor methods.

```
methsearch -b6 -PG1 -srf -p2 -l2 -qH'$2c $C'
```

(Windows users should use double quotes rather than single quotes.) This produces the output:

On such systems, in an extremely long search where these might exceed 2^{31}, `$#` will succeed where `$[$#]` (which forces a conversion to a 32-bit signed integer) will fail.

```
77 Delight
41 Surprise
29 Treble Bob
```

Certain variables — namely `$p`, `$q`, `$Q`, `$n`, `$N`, `$i`, `$T` and `$#` — cannot be included directly in statistical output. This is not a technical restriction *per se*, rather that as place notations are unique to a method, and names nearly so, it would rarely serve any purpose, and as the whole statistical output has to be stored in memory until the search completes, it would consume a large amount of memory and possibly result in `methsearch` running out of memory and crashing. These variables can all be referenced in expressions,[19] for example, to see how many of the 2400 treble dodging minor methods in the Central Council's 2008 *Treble Dodging Minor Methods* are still unnamed, you could run:

```
methsearch -b6 -PG1 -sp2 -Lsurp6.txt -Ldel6.txt -Ltb6.txt       \
  -qH'$4c $[$n eq "Untitled" ? "Unnamed" : "Named"]'
```

(Again, Windows users should use double quotes, and also escape the inner double quotes using a backslash.) At the time of writing, this produces the output:

```
1728 Named
 672 Unnamed
```

`methsearch`'s ability to do useful statistical analysis of methods is quite limited. In most cases, it is likely that you will want to either use (or at least combine `methsearch`'s statistical abilities with those of) some other package, such as a spreadsheet or standard command line utilities such as `awk`, `sort` or `grep`. This is consistent with the Unix philosophy outlined in §1.2.2 — specifically that `methsearch` should do one thing and do it well. That thing is searching for methods; plenty of other programs exist that can do interesting statistical analysis, and `methsearch` aims to make interfacing with them as easy as possible rather than attempting to subsume their functionality.

[19] This gives you a way of circumventing the restriction on placing these variables directly in statistical output: `-H"$[$n]"`. But don't do this.

Chapter 4

Falseness

By default `methsearch` only finds methods that are true within a lead. The `-F` option provides a way of changing this behaviour. The basic options governing whether the method should have a true plain course, true leads, etc., are documented in §4.1. Analysis of false course heads is discussed in §4.2, and ways of checking that an extent is possible is covered in §4.3. The final two sections (§4.4–4.5) discuss using `methsearch` to find methods that are true against other methods.

4.1 Basic falseness levels

`methsearch` supports five basic levels of falseness checking wich can be specified by a single-letter argument to the `-F` option. Longer, more descriptive, synonyms are also provided.

`-Fc`	`-Fcourse`	Require a true plain course
`-Fl`	`-Flead`	Only require truth within a lead [default]
`-Fh`	`-Fhalf-lead`	Only require truth within each half-lead
`-Fn`	`-Fnone`	Do not check falseness
`-Fx`	`-Freally-none`	Allow even trivially false methods

Although `-Fn` option does not check the falseness of the method, two forms of falseness are still excluded. These are referred to as *trivial falsenesses*. The *null change* (that is, the change in which no bells move) is still not permitted, nor is the immediate repetition of change. To allow these trivial falsenesses, the `-Fx` option should be used.

When checking a lead with `-Fl`, `methsearch` proves all the rows between the

opening rounds and the lead end row inclusive — it also requires that the lead head is either distinct, or the same as the initial lead head. Similarly, `-Fh` proves all the rows from the opening rounds to the half lead end, and separately from the half lead head to the lead end. (In practice, if either `-s` or `-d` is specified, only one half-lead is tested — the truth of the other is guaranteed by symmetry.) With `-Fh`, the half lead head row must either be distinct from all rows in the first half lead, or the same as the half lead end row.

`methsearch` contains a number of optimisations to speed up proving falseness in the most common cases. For example, it knows that in a palindromic plain method, there is no possibility of internal falseness.

4.2 False course heads

4.2.1 A digression — the theory of falseness

If a course of a method starting from course head, r, is false against the plain course, then r is known as a *false course head*. This can be explored mathematically. Throughout this section I shall use the convention that xy means x transposed by y, rather than the x transposition operating on y. (Neither convention has universal uptake amongst ringing theoreticians, although the one used here seems increasingly popular.)

Let C denote the set of rows in the plain course of a method, and define the product of a row, r, and set of rows, C to mean the set containing r multiplied by each member of C:

$$rC = \{rx : x \in C\}.$$

Thus rC is set of rows in the course starting from course head, r. If C and rC contain a row (or more) in common, then r is a false course head as defined above. Introducing the notation $F(C)$ for the *falseness set* — the set of all false course heads of the method C — this can be written mathematically,

$$C \cap rC \neq \emptyset \iff r \in F(C).$$

After some manipulation, this can be written as a closed formula for $F(C)$,

$$F(C) = \{ab^{-1} : a, b \in C\}.$$

It can be seen from this definition that if r is a false course head then so too is r^{-1}. (This is because $(ab^{-1})^{-1} = ba^{-1}$ and we are free to relabel a and b.)

It can be useful to introduce notation to refer to the set of inverse elements

$$\bar{C} = \{a^{-1} : a \in C\};$$

the multiplication of two sets can also be defined in the obvious way — as the set composed from each element of the first set multiplied by each element of the second set:

$$AB = \{ab : a \in A, b \in B\}.$$

Using this notation, the falseness set is simply $F(C) = C\bar{C}$.

As most methods have internal structure by virtue of being palindromic and being composed from identical leads, the falseness set also has structure. Consider a palindromic method with lead end row, e, and lead end change, h. The set, L, of lead heads and lead ends of the method is the dihedral group generated by these two elements.

$$L = \langle e, h \rangle = \{1,\ e,\ eh,\ ehe,\ eheh,\ ehehe,\ \ldots\}.$$

The rows in the plain course of the method can thus be expressed in terms of the rows, H, in the first half lead of the methods, $C = LH$, thus allowing the falseness set to be expressed as

$$F(C) = LH\bar{H}\bar{L} = L\ F(H)\ L.$$

(Note that $L = \bar{L}$ because L is a group.[1]) This partitions the falseness set (and the set of all possible course heads) into *double cosets* of L: if r is a false course head, then so are all of LrL. However, it has already been noted that if r is a false course head, so is r^{-1}. Combining these two results pairs certain double cosets such that if one occurs so must the other. These sets have the form $L\{r, r^{-1}\}L$ and are known as *falseness groups.*[2]

4.2.2 Falseness groups

The falseness groups for a method are useful because they can be used to locate compositions that are true to that method. A composition is described as *universally true* to certain falseness groups if it is necessarily true to all methods having at most those falseness groups.[3]

[1] Specifically a set of rows, X, is a group if and only if $X = \bar{X}$ and $XX = X$. These follow from the inverse element and closure group axioms.

[2] This term is a misnomer as falseness groups are not groups in the mathematical sense.

[3] The advent of computer programs that allow people to search for compositions particularly tailored to a specific method has somewhat reduced the use for falseness groups. However not everyone has access to or the inclination to use such programs, and collections of universal compositions are still published; for example, the Central Council's 2001 *Collection of Universal Compositions of Treble Dodging Major Methods.*

For compositions that are *tenors-together* (generally defined as meaning bells 7 and higher remain coursing throughout), it is only necessary to look at falseness groups that include tenors-together course heads; similarly, a bobs-only composition only requires falseness groups including in-course course heads to be considered.

On eight bells (assuming a fixed treble and a seven-lead course), there are 28 such falseness groups of which 25 include tenors-together courses. On higher stages the total number dramatically increases, but the number with tenors-together courses does not significantly and are related to eight bell ones. For regular major methods, these have conventional names using the letters `A`–`U` (except `J` and `Q`) to denote the tenors-together groups containing in-course rows, `a`–`f` for out-of-course tenors-together groups and `X`, `Y` and `Z` for the three remaining groups which have no tenors-together rows.

On ten or more bells, certain tenors-together groups split. This is indicated by a numeric suffix on the falseness group code. The only difference between falseness groups on ten bells and higher stages is that on ten, the `D1` and `B1` groups combine. Table 4.1 gives all of the tenors-together false course heads and their corresponding falseness group.

23456 A	25463 M	34256 U1	36452 T1	45236 F1	52463 e	56234 R	63452 D2
23465 a1	25634 C1	34265 C2	36524 O1	45263 T2	52634 N3	56243 H2	63524 O2
23546 a2	25643 b	34526 K2	36542 H2	45326 d	52643 O1	56324 f	63542 G
23564 M	26345 F2	34562 N1	42356 U1	45362 S	53246 T1	56342 P2	64235 S
23645 M	26354 T1	34625 S	42365 C2	45623 R	53264 N2	56423 G	64253 K2
23654 B2	26435 M	34652 d	42536 c	45632 N3	53426 D2	56432 O2	64325 K3
24356 D2	26453 a2	35246 c	42563 L2	46235 T2	53462 H1	62345 N1	64352 P1
24365 B1	26534 b	35264 U2	42635 U2	46253 D1	53624 K1	62354 d	64523 P2
24536 T1	26543 L1	35426 U1	42653 c	46325 N1	53642 O2	62435 N2	64532 f
24563 F2	32456 B2	35462 N2	43256 B2	46352 K2	54236 d	62453 T1	65234 N3
24635 E2	32465 E1	35624 N3	43265 E1	46523 H2	54263 N1	62534 R	65243 O1
24653 T1	32546 D1	35642 R	43526 T1	46532 O1	54326 P1	62543 H2	65324 I
25346 T1	32564 T2	36245 L2	43562 e	52346 K2	54362 K3	63245 e	65342 f
25364 E2	32645 T2	36254 c	43625 N2	52364 S	54623 f	63254 U1	65423 O2
25436 B2	32654 F1	36425 e	43652 U1	52436 U1	54632 I	63425 H1	65432 K1

Table 4.1: False course heads and their corresponding falseness group.

Although it is possible to generalise these falseness groups codes to irregular methods, `methsearch` does not currently support it.

The `A` falseness group (that is, the one including `23456` as a course head) can be interpreted in different ways. It is sometimes taken as the identity falseness, in which all methods have it; and it is sometimes taken to mean that the method is internally false within the plain course. `methsearch` uses the former definition.

4.2.3 Restricting falseness

The `$F` method variable prints all the falseness groups of a method including `A`, any out-of-course groups, and, on eight bells only, the three split-tenors groups.

For example, for Cambridge Surprise Major, it will print `ABDEe`.

`methsearch` allows you to restrict the search to methods with (at most) certain falseness groups. This is done with the `-F` option. The syntax is `-F:`*groups*. A '`-`' may also be used to allow ranges for falseness groups. The `A` group is implicitly included. For example, on eight bells, `-F:BDEa-f` finds methods with only `B`, `D` and `E` falseness in-course, but arbitrary out-of-course groups. It does not, however, find any major methods with `X`, `Y` or `Z` falseness unless these are explicitly allowed.

`methsearch` has a short hand notation for searching for *clear proof scale* methods. These are methods with no in-course tenors-together falseness, but allowing arbitrary other falseness. This is written `-FCPS` and on eight bells is equivalent to `-F:a-fXYZ`. (It is a fairly common misconception that clear proof scale methods and only clear proof scale methods can produce an extent. Neither is, in general, the case.)

4.3 Extent viability

On lower stages (particularly on seven or fewer bells) it is common to want methods which are capable of producing an extent. Testing this is, in general, a difficult problem. However, one prerequisite is that a large enough set of mutually true leads exist. `methsearch` is able to check for certain special cases of this.

The `-Fe` option, also spelt `-Fextent`, checks the $(n-1)!$ possible fixed-treble lead heads on n bells to see whether a mutually true set of $\frac{1}{2}(n-1)!$ leads exists. The `-Fe+` option, also spelt `-Fpositive-extent`, checks the $\frac{1}{2}(n-1)!$ possible in-course fixed-treble lead heads to see whether a mutually true set of $\frac{1}{4}(n-1)!$ leads exists. These options imply `-Fl` and should not be used with any lower basic falseness level; the `-Fc` option can, however, usefully be used in conjuction with `-Fe` or `-Fe+`.

In practice, `-Fe` should be used for non-palindromic plain methods where it ensures that sufficient true leads exist for an extent to be possible. (The option is safe but unnecessary with palindromic plain methods as these necessarily have enough true leads to form an extent.) Similarly, the `-Fe+` option should be used with treble-dodging methods where it ensures that sufficient true leads exist for a bobs-only exist. Neither option actually check that the leads can be joined into an extent.

These options are implemented by considering the graph whose vertices are the possible lead heads and edges exist between each pair of mutually false leads. Mathematically, this is the Cayley graph $\Gamma(S_{n-1}, F(M))$ where S_{n-1} is the set of possible fixed-treble lead heads on n bells. An example of the largest possible

set of mutually true leads corresponds to a *maximal independent set* of this graph. In the general case, finding the size of this is NP-complete;[4] however it is simple to test whether the maximal independent set is exactly half the size of the graph by testing whether the graph is *bipartite* which can be tested easily. This is what the `-Fe` option does; the `-Fe+` option does the same for the in-course lead heads, A_{n-1}.

For palindromic treble-dodging minor methods that have a double change at the half lead (as all five-lead ones will), the `-Fe+` option is a overkill. A better strategy is to require that in each half lead, there is one even parity row and one odd parity row for each position of the treble. The `-P` option enforces this. Thus a method can start `-34-` or `34-34` which have parity structures $++\ --$ or $+-\ -+$ respectively; but a `-3456-` start has only even parity rows when the treble leads, and only odd parity rows when it is in seconds place: $+-\ +-$.

If you wish to check whether an extent actually does exist, you can get `methsearch` to invoke the `touchsearch` program on each method. See §6.3.1 for more details.

4.4 Avoiding specific rows

The `-Fr` option allows you to specify specific rows that must be avoided in the the method. The syntax is `-Fr=`*row* — for example, `-Fr=12435678` will force the method not to contain the row `12435678`. Depending on the basic falseness level specified (`-Fc`, `-Fl` or `-Fh` — see §4.1), `methsearch` will avoid the specified rows in the plain course, first lead or first half lead of the method.

The initial rounds is counted as part of the first lead; the lead head at the end is not — therefore, `-Fl -Fr=135264` will not prevent Plain Bob Minor from being found.

Multiple `-Fr` options work in the obvious way. As `-Fr` works by pruning the search tree rather than filtering the results, adding a few `-Fr` options for unwanted rows that might occur early on in the method can vastly speed up a search.

Note that there is no interaction between this and the `-Mlead` and related options. If you wish to prevent a specific row from occur in a different lead, you have two basic strategies. Either you can use a `-Fs` option to specify the starting row of the method; or you can work out the corresponding row to be avoided in the opening

[4] Actually, while the case of the maximal independent set of a general graph is known to be NP-complete, it is not known whether the special case of the maximal independent set of a vertex transitive graph (such as a Cayley graph) is NP-complete.

lead.

The syntax of the `-Fs` option for specifying the starting row is similar to that of `-Fr` except that the argument to `-Fs` must be a literal row (row expressions are not permitted), and multiple `-Fr` options are not allowed.

Suppose you want to ring the method being searched for starting at lead-head s, and you want that lead not to contain the row r, then this is equivalent to saying that the first lead of the plain course should not contain the row $s^{-1}r$. Fortunately `methsearch` provides a syntax to avoid you having to compute these manually: `-Fr="`$s\backslash r$`"` (Linux users will want single quotes). This is equivalent to `-Fs=`s `-Fr=`r. The details of this syntax are explained in §4.4.1.

4.4.1 Row expressions

The argument to `-Fr` is actually a *row expression* — an expression that evaluates to a row or set of rows. Row expressions also occur in the `-FP` option (§4.5), the `-Mlead` family of options (§5.2) and in the `rowcalc` utility (§6.2.2). These row expressions are considerably simpler than the more general expressions in §3.2, and are handled by entirely separate code in `methsearch`. A future version of `methsearch` is likely to merge both into a single expression language, but until that happens, the row expression syntax will remain very limited — unless noted, all operators act only on rows, and there is no facility for integer arithmetic.

Valid expressions are formed from the following operators, arranged in precedence levels from highest to lowest. The operators' associativity is also given. Parentheses may also be used to make the order of invocation explicit.

`^`	right	exponentiation
`* / \`	left	multiplication-type operators
`-`	left	set difference — see §4.4.2

The multiplication operator, `*`, permutes one row by another. The convention used by `methsearch` is that x`*`y means x transposed by y, rather than the x transposition acting on y, thus `1432*2143` is `4123` and not `2341`.[5]

Because row multiplication is non-commutative, division isn't uniquely defined. You need to specify whether the dividend is right- or left-multiplied by the inverse of the divisor. As a result, it is the general convention to define

$$a/b = ab^{-1}$$

[5]Neither convention is used universally by ringing theoreticians and both have advantages.

as right multiplication by the inverse of the divisor. By analogy, I define the notation the notation

$$a\backslash b = a^{-1}b$$

to mean left multiplication by inverse of the divisor. The `/` and `\` operators perform these two types of division, respectively. The latter occurs in ringing-related problems considerably more frequently than the former.

The `*` and `\` operators may need quoting to prevent the shell from trying to interpret them. As usual, on Windows, double quotes should be used; and on Unix-like systems, single quotes will be necessary.

The `^` operator is *overloaded* with two meanings. If the right-hand argument is a row, then it denotes *conjugation*, and if the right-hand argument is an integer then it denotes *exponentiation*; in both cases, the left-hand argument must be a row. In the mathematical literature these operations are often both denoted by right superscripts and this convention is followed here,[6] but using $x\hat{}y$ in place of x^y. The conjugation and exponentiation operations are defined by

$$a^r = r^{-1}ar, \qquad a^n = \underbrace{a\,a\,a\,\cdots a}_{n \text{ times}}, \qquad a^{-n} = (a^{-1})^n,$$

where a and r are rows, n is a positive integer, and a^{-1} denotes the inverse of a.

Row expressions may also contain row literals and integer literals. Row literals may have the treble omitted — for example, `2345` is equivalent to `12345`. This does not generalise further so `345` is not a valid row literal. Rows may also have an arbitrary number of tenors omitted, thus the two preceding examples are equivalent to `123` or even just `1`. Integer literals may optionally be prefixed by a `+` or `-` sign.

An ambiguity exists between rows and integers, although in practice `methsearch` almost always gets it right. The heuristics by which rows and integers are disambiguated are as follows:

— in contexts where integers would be invalid (i.e. anywhere other than the right-hand argument to a `^` operator), it is parsed as a row;

— if a leading `+` or `-` is given, it will parse as an integer;

— if the token is only a single character long, it is parsed as an integer;[7]

[6] One motivation for using the exponentiation syntax for conjugation is that both satisfy the identity $(a^r)^s = a^{(rs)}$. However the analogy shouldn't be taken too far. If a and r are rows and n is a integer, then $(a^r)^n = (a^n)^r$, and one might use a^{nr} as syntactic shorthand for this. Nevertheless, the product, nr, of a integer and a row is otherwise meaningless and `methsearch` does not support this usage.

[7] This case is only relevant in one case — the digit `2`. Without this rule this would parse as rounds

— if the token is a valid row (possibly omitting the treble), it is parsed as a row;

— otherwise it is parsed as an integer.

4.4.2 Multiple rows

Most of the time, there isn't just one row that you wish to avoid. The obvious use is when you have a particular composition in mind and have already selected some of the methods. In this case, you'll probably have hundreds of rows that you wish to avoid. If so, listing them all on the command-line is likely to be unsatisfactory. (Not only is it going to be tedious to write them, and error-prone unless you can automate the process, but there are technical reasons for not wanting the command line to grow too long.) `methsearch` has four short-hand notations for sets of rows: *set literals*, *group generator lists*, *file inputs* and *command invocations*.

A *set literal* is specified by enclosing a list of rows (or row expressions) in braces and separating the rows with columns — thus, `-Fr="{`a`,`b`,`c`}"` is a set literal equivalent to three separate options, one each for each row in the set $\{a, b, c\}$.

A *group generator list* is specified by enclosing a list of rows (or row expressions) in angle brackets; these are used to generate a group. For example, `-Fr="<34562>"` generates the cyclic group on the five working bells, and `-Fr="<342,23465>"` generates a group of order six isomorphic to $C_3 \times C_2$. A future version of `methsearch` is likely to provide an alternative syntax for specifying groups as it is not always readily apparent what group is generated by a particular set of generators. In particular, generating large groups like $S_3 \times S_5$ (which occur in `-FP` options for differentials) is error-prone with the current syntax.

A *file input* reads rows from a file — the syntax for this is `-Fr="<(`*filename*`)"`. The file format is quite flexible: a sequence of rows separated by some combination of spaces, tabs, new lines, commas, semicolons or colons; however a standard *row stream* (§6.2) with one row per line is recommended. As a special case, if *filename* is `-`, instead of reading from a file, `methsearch` reads from standard input. If you do this, you must ensure that nothing else is also reading from standard input or you will end up with bizarre undefined behaviour. In particular, you can only have one `<(-)` anywhere, and you cannot use this in conjunction with `-I` (§2.9).

A *command invocation* has the syntax `-Fr="$(`*command line*`)"` and is discussed in detail in §6.1.

Note that the sequence `<(` is always treated as the start of a file input even if this

with an implicit treble and we would find that $1 = 2$. (Technically, it also changes 1 from rounds to the number 1, but in practice this is undetectable.)

results in a parse error. If you wish to put parentheses inside a group generator list, you should put a space between the `<` and `(`.[8]

Sets (using any of these syntaxes) can appear in expressions. If an expression contains just one set, the expression is evaluated multiple times, once for each member of the set.[9] For example, `-Fr="{ 124365, 132465, 213465 }^654321"` is equivalent to `-Fr=214356 -Fr=213546 -Fr=214565`.

An `-Fr` option may contain two (or more) sets of rows. Using the definitions, AB and $\bar{A}$, introduced in §4.2.1, these are defined as follows for sets A and B:

`-Fr="`$A*B$`"`	$\iff$	`-Fr="`AB`"`
`-Fr="`A/B`"`	$\iff$	`-Fr="`$A\bar{B}$`"`
`-Fr="`$A\backslash B$`"`	$\iff$	`-Fr="`$\bar{A}B$`"`
`-Fr="`$A\hat{}B$`"`	$\iff$	`-Fr="`A^B`"`

In the last case (conjugation), the set A^B is defined

$$A^B = \{b^{-1}ab : a \in A, b \in B\}.$$

Thus $A^B \neq \bar{B}AB$ because the same element of B is used on both sides of A. This was the major motivation for adding support to `methsearch` for conjugation with the `^` operator.

Note that $A\backslash B$ does *not* denote the difference between two sets which is often denoted by a backslash in the mathematical literature. The `-` operator serves that role, and the difference (or complement), $A - B$, is defined in the usual way as all elements of A that are not elements of B:

$$A - B = \{a : a \in A, a \notin B\}.$$

The `-` operator can also be used when one or both of its arguments are rows rather than sets. In such a case, the row is silently replaced with a set containing just that one row. This is consistent with `methsearch`'s general policy that a row and a set containing one row should always be treated the same.

These relations are equally valid irrespective of whether the set is a set literal, read from a file, or generated by a command invocation. Using these set products, it's easy to generate very large row sets. `methsearch` needs to be able to hold all the distinct `-Fr` rows in memory at one time.

---

[8] This will be familiar to many programmers as the *maximal munch* principle which requires that a sequence of characters is parsed as the longest possible token. For example, in many C-like languages, `1--1` is an error as it treats `--` as a single unary operator, but `1 - -1` is parsed successfully (and evaluates to 2).

[9] This behaviour might be counter-intuitive for `{`a`,`b`,`$\cdots$`}^`n, and one might anticipate this to be equal to `<`a`,`b`,`$\cdots$`>` for sufficiently large n. It is not, and `A^2` $\neq AA$.

The `rowcalc` utility (§6.2.2) provides a handy way of checking that any complicated expressions passed to `-Fr` are producing the correct output.

4.4.3 An example — course splices

As an example, suppose we want to find *course splices* of Beverley Surprise Minor. These are methods that can be dropped into an extent of Beverley in place of a whole course of Beverley and still leave a true extent.

Suppose we have a file called `Beverley.txt` containing all the rows in a plain course of Beverley. We can then use `methsearch` to calculate all of the rows in the five other courses (i.e. those besides the plain course), and ask `methsearch` to find methods not containing any of these rows. This can be done as follows.

```
methsearch -b6 -PG1 -sr -p2 -R"$q  $n" -Lsurp6.txt -Fc          \
  -Fr="{134256,142356,152436,154326,153246} * <(Beverley.txt)"
```

The set literal contains the five non-plain course heads. When multiplied by the plain course, this generates a set of 600 rows to avoid, and the `-Fc` option checks the whole plain course against these 600 rows. It generates the following output.

```
&-3-4-2-3.4-2.5,2       Surfleet
&-3-4-2-3.4-2.5,1       Hexham
&-3-4-2-3.4-34.1,2      Durham
&-3-4-2-3.4-34.5,2      Beverley
&-3-4-2-3.4-34.5,1      Berwick
```

Beverley and Surfleet are *lead splices* which are a special case of a course splice; Berwick and Hexham are their sixth-place variants and so contain all the same rows in the course (although are not typically referred to as a course splice); Durham is a genuine course splice of Beverley.

4.5 Part end groups

`methsearch` can also require a method to be true when rung in composition based on a multi-part structure. Only multi-part structures where the parts form a mathemtical group are supported, but in practice virtually all sets of part ends are groups. A simple example might be a three-part structure with part ends {`123456`, `134256`, `142356`}.

The parts are specified with the `-FP` option. Its argument is a row expression, and the rows passed to `-FP` are used to generate the part end group. Thus for the three-part group above only one option is needed `-FP=134256`.

This option can be combined with `-Fs`, `-Fr` (see §4.4) and `-Fc`, `-Fl` or `-Fh` (see §4.1) with powerful effect. These interact as follows. The `-Fc`, `-Fl` or `-Fh` option is used to determine whether to prove the whole course, the first lead (the default), or the first half lead. Let M denote the rows in this section of method. The `-Fs` option specifies the starting row, denoted here by s; the `-FP` options generate the part end group, P; and the `-Fr` options enumerate any rows to be avoided, A. `methsearch` requires that the following rows are mutually true:[10]

$$PsM \cup A.$$

The `-FP` option also provides a way of searching for methods that can produce an extent. For example, if you can find a group, G, and a treble path of length m (whether in the half-lead, lead or course) such that $|G|\, m = n!$ then it might be possible to use the group as the part-end group for an extent. In other words, it might be possible to find a method, M, that satisfies $GM = S_n$.

For example, most treble dodging minor methods produce extents that have the 60 in-course rows of the form `1.....` as lead-ends and lead-heads. The `-P` option guarantees that the methods will be true with a composition of this style. The same effect can be achieved with `-FP=124365 -FP=132546 -FP=132465 -Fh`. (These three rows, which are the changes `12`, `16` and `14` generate the group A_5 of in-course fixed-treble rows.) In practice, the `-P` option is significantly faster and there is no reason to prefer `-FP` in this case.

It makes no difference whether you use multiple `-FP` options, or to put all the generators together with a set literal. For example, the generators for A_5 given above could have been written `-FP="{124365,132546,132465}"`. The quotes may be necessary to prevent the shell from interpreting the braces itself.

However, the `-FP` options can be used in other situations too. For example, there is a second group of 60 in-course rows on six bells — this is *Hudson's group* which is generated by the changes `12`, `16` and `34`.

```
methsearch -b6 -G1 -p3 -l2 -srf -R"$q  $N" -Ldel6.txt          \
  -FP=124365 -FP=132546 -FP=213465 -Fh
```

[10] This is mathematically sloppy as a set, by definition, contains no duplications. What we means is that there is no intersection between each psM for all $p \in P$, and also between each of these and A. A cleaner way of expressing this is with the cardinality of the sets:

$$|PsM \cup A| = |P| \times |M| + |A|\,.$$

Interestingly, this just generates one method:[11]

```
&3-3.4-2-1.4-4.5,2      Norwich Delight Minor
```

4.5.1 Differentials

The usual way to get a palindromic differential method capable of producing an extent is to arrange for all of the bells in one cycle to ring in every position relative to each other during half a lead. For example, in a 2,3 doubles differential, the two trebles would ring in each of the $5 \times 4 \div 2 = 10$ relative positions during half a lead. The two trebles swap over at the half lead or lead head to produce a lead head of 21453 or 21534. During the course (which is a whole extent), the twelve lead ends and lead heads are:

12345	21453	12534	21345	12453	21534
21435	12543	21354	12435	21543	12354

These twelve rows have every combination of the the two trebles in both possible orders and the three tenors in all six orders. It is therefore the group $S_2 \times S_3$. To get `methsearch` to search for 2,3 doubles differentials, you ask it to look for palindromic methods with 20 rows per lead, 6 leads per course, and a half-lead that is true to this group:[12]

```
methsearch -b5 -AU -n20 -sFh -FP="{12435,21453}" -Q"$o==6"
```

In this case, 12435 and 21453 generate the group $S_2 \times S_3$. The need to specify the part-end group in terms of its generators can be error-prone and a future version of `methsearch` is likely to introduce an alternative syntax for specifying groups. In the meantime, the `rowcalc` utility (§6.2.2) help check that the group produced is the intended one. For example,

```
rowcalc -b5 -c "<12435,21453>"
```

[11] This method was originally named Hudson Delight Minor. However, in their wisdom, the Central Council Methods Committee have decided that this method should be renamed Norwich Delight Minor because of some perceived similarity to a major method of that name.

[12] Such methods were first catalogued by Jonathan Deane and published in the *Ringing World* [1994, p. 407]. He recognised that it was impossible to avoid four blows in one place, but wanted limit this the occurence of multiple consecutive blows in one place to just four blows at lead across the lead end. `methsearch` can duplicate his analysis by adding `-p2 --long-le-place -Q"$19h ne '145' || $20h ne '125'"` to the command line. The latter `-Q` option is to avoid methods with four blows behind over the lead end.

prints 12, the size of the group. (Removing the `-c` causes it to print the whole group.)

This extends to differentials on higher numbers. Double Helix Differential Major is a 3,5 major differential, so it uses the part-end group $S_3 \times S_5$ which is generated by $\{$`23156487`, `13287654`$\}$. This is an example where the full search space may be too large to enumerate exhaustively (depending on exactly what additional criteria are used), however it is a good example of when the random sampling techniques in §2.8 can be used.

```
methsearch -b8 -AU -n112 -sFh -FP="{23156487,13287654}"         \
  -Q"$o==15" -dkj -p2 --random --loop --timeout=1
```

The correct choie of timeout is critical to getting such a search to run efficiently. This is because there are large regions of the search space with no methods, and if a loop enters into such a region, it's highly unlikely to ever leave it. Equally, if a loop enters a region with lots of methods, it needs long enough that it will find one of them. Too high a timeout and we waste time in barren parts of the search space: too low and we fail to notice methods when they are there.

In the general case, `methsearch` runs very slowly when the part-end group, P, is large. Each node of search tree requires an operation that's linear in the size of group. However, for the most common case where the group is just a direct product of symmetric groups, (i.e. if $P = \prod S_{n_i}$), a constant-time algorithm is used. What this means in practice is that it is fast to search for differentials using the techniques discussed here, but that other, seemingly-similar searches can end up very much slower.

Chapter 5

Music

`methsearch` is able to analyse the music present in a method. This is controlled by the `-M` option and is used to generate a musical score which is displayed by the `$M` variable. The first section (§5.1) documents how to analyse music in the plain course; more general analysis is discussed in §5.2.

5.1 Music patterns

At its simplest, `methsearch` simply counts the number of rows in the plain course matching a particular *music pattern*. Each pattern is specified with an option of the form `-M`*pattern*. A pattern may contain literal bell symbols (using the bell symbols listed in §2.6) or *wildcards* that can match various bells. Three forms of wildcard are supported: a `?` represents any single bell, a `*` represents any number (including zero) arbitrary bells, and `[`*list*`]` matches exactly one bell from the list of symbols.

For example, on eight bells, `-M????5678` matches a roll-up at the back; this can be written more succinctly as `-M*5678`. A combination roll-up (CRU) on eight has two of 4, 5 and 6 rolling-up followed by 78. It can be written `-M*[456][456]78`.

If a pattern is shorter than the row and contains no `*` to match the surplus bells, then it matches at the start of the row. For example, `-M5678` matches the row `56781234` but not `12345678`. To match a sequence of bells at any place in the row, place a `*` on both sides of the pattern: `-M*5678*`.

5.1.1 Named music patterns

There is also a shorthand notation for matching particular commonly requested patterns such as queens, CRUs, or four-bell runs. This is written `-M<`*name*`>`. Note that it will generally be necessary to enclose the < and > signs (and, for convenience, everything between them in quotes): `-M"<`*name*`>"`. At present, the following named patterns are supported.

`rounds`	The bells in order; e.g. `1234567` on seven bells.
`queens`	The odd bells in order, followed by the even bells in order; e.g. `1357246` on seven bells.[1]
`kings`	The odd bells in reverse, followed by the even bells in order; e.g. `7531246` on seven bells.
`tittums`	The front bells in sequence, alternating with the back bells in order; e.g. `1526374` on seven bells.
`reverse-rounds`	The bells in reverse order; e.g. `7654321` on seven bells.
`CRUs`	Any four bells, followed by two of $\{4, 5, 6\}$, followed by the back bells in order.
`front-`n`-runs`	A run of n bells in sequence (forwards or backwards) off the front of the row.
`back-`n`-runs`	A run of n bells in sequence (forwards or backwards) at the back of the row.
n`-runs`	A run of n bells in sequence (forwards or backwards) at an arbitrary position in the row.

With the n`-runs` named pattern, it is possible for a row to match the pattern more than once. For example, the row `32145678` matches `-M<4-runs>` twice because it has a five-bell run and both of its constituent four-bell runs (`4567` and `5678`) match. This means a longer run scores more than a shorter run.

5.1.2 Customising scores

By default, each match adds 1 to the music score. Multiple patterns may be specified, in which case each row is matched against each pattern allowing rows to match multiple times. It is possible to change the score awarded to each match of a pattern. The syntax for this is `-M`*score*`:`*pattern*, where *pattern* may be a simple pattern or a named pattern enclosed in <...>.

This allows you to specify the relative merits of particular rows. For example,

[1] The examples are given on an odd number of bells because these named rows are ambiguous at odd stages. The particular choice of queens as `1357246` instead of `2461357` is because odd bell methods are more usually rung with a tenor covering.

`-M<4-runs> -M5:*7568 -M5:*5678 -M5:8765 -M5:5678 -M10:<queens>` will award ten points to queens, five points to each of four particularly musical types of runs, and one point for other four-bell runs. (As a result, rounds would score 10 — one each for the five four-bell runs in `12345678`, and a further five for the `5678` at the back.)

It is also possible to score rows at handstroke and backstroke differently. This can be useful for excluding `87`s at backstroke (on eight bells, or equivalent on other stages). The syntax is `-M`h`,`b`:`*pattern* where h is the score to award if the pattern matches at handstroke and b the score to award for a match at backstroke.[2] By setting $h = 0$ and b to a large negative number, it makes it easy to locate and reject methods with a particular row at backstroke. E.g. `-M0,-1000:*87`.

5.2 Music outside the plain course

Sometimes we are interested in music other than in the plain course and `methsearch` can accommodate this via the `-M`*type*`=`*row* options. The possible values of *type* are as follows:

`course`	Analyse a whole course starting from course head *row*.
`lead`	Analyse one lead starting from lead head *row*, excluding the closing lead head row.
`halflead`	The first half of the lead starting from lead head *row*
`2halflead`	The second half of the lead, starting with *row* as the half lead head row.
`2rhalflead`	The second half of the lead, ending with *row* as the lead end row.
`rhalflead`	The first half of the lead, ending with *row* as the the half lead end row.[3]

Multiple `-M`*type*`=`*row* options may be provided. The `=`*row* component of these options may be omitted, in which case it defaults to the *row* option to the previous such option if there is one, or to rounds if no such previous option exists. For example, `-Mlead -Mlead=13426578 -Mlead` is equivalent to `-Mlead=12345678 -Mlead=13426578 -Mlead=13426578`.

(Note that there is a potential, though highly improbable, ambiguity between these options and the `-M`*pattern* options described in §5.1. This only applies on `22` or

[2] `methsearch` assumes that the method is rung with the first non-rounds change being a handstroke.

[3] This option is largely included for completeness. It is not anticipated that `rhalflead` will often be used.

more bells and is because `-Mlead`, for example, is also a pattern matching a row with the bells 22, 11, 13, 16 in order at the front. In the case of such an ambiguity, the meaning described in this section is chosen. If this is of concern — for example, if you are automatically generating patterns for some purpose other than conventional musical analysis — a pattern that does not already contain a `*` can always have one appended without changing its meaning, and this will force the option to be interpreted as a pattern. Similarly, a pattern can always have the implicit `1:` scoring prefix stated explicitly which will also force interpretation as a pattern.)

These `-M`*type* options do not, of themselves, specify what music to look for — they need using in conjunction with the music pattern options of §5.1. In the simplest usage, a single `-M`*type* option will be followed by a `-M`*pattern* option which will apply to the block specified. For example, `-Mlead=15634827 -M"<queens>"` will look for queens in the row starting with `15634827`.

5.2.1 Batches of music patterns

In more complicated scenarios, it can be desirable to search for multiple types of music in different places. This is possible too. The `-M` options from the command line are considered in sequence.[4] In general, `methsearch` divides the `-M` options into a sequence of batches, each of which consists of one or more `-M`*type* option followed by one or more `-M`*pattern* option. Optionally, the first batch may have no `-M`*type* options specified, in which case `-Mcourse` is assumed; similarly, the last batch may have no `-M`*pattern* options specified, in which case `-M<CRU>` is assumed. (Thus, if no `-M` options are given at all, `methsearch` looks for CRUs in the plain course.)

Within each batch of `-M` options, each of the music pattern options (whether named patterns with `-M<`*name*`>` or regex-style patterns with `-M`*pattern*) applies to all of the `-M`*type* options within that batch of `-M` options. Some examples may make this clearer.

```
-M"<4-runs>" -Mlead -M2:"<4-runs>" -M10:"<reverse-rounds>"
```

This example contains two batches of `-M` options. The first batch looks will score one point for each four-bell run in the plain course. The second batch re-analyses just the first lead, scoring another two points for each four-bell run in that lead, and another further ten points if back rounds occurs in the first lead. Because rows can match multiple patterns, an eight bell method such as Plain Bob which

[4] This is one of the few places where the order of command line options is important.

has reverse rounds at the first lead will score $1 \times 5 + 2 \times 5 + 10 \times 1 = 25$ points for this row alone.[5]

```
-Mlead=15634827 -Mlead=16745238 -Mlead=17856342                \
-Mlead=18267453 -Mlead=12378564 -Mlead=13482675                \
-Mlead=14523786 -M"<4-runs>" -M10:"<queens>"
```

This contains a single batch of `-M` options. For each of the seven leads listed, it scores four-bell runs with an extra bonus for queens.[6]

In practice, however, such a circumlocution is unnecessary because the *row* argument to the `-M`*type*`=`*row* options is actually a *row expression* (§4.4.1). This means that the group generator syntax can be used to write this example more concisely and informatively as:

```
-Mlead="<8234567>*5634827" -M"<4-runs>" -M10:"<queens>"
```

More powerful music analysis of touches can be achieved by getting `methsearch` to invoke `gsiril` to print the touch. This is discussed further in §6.2.4; there are often several ways to achieve the desired analysis and the discussion in §6.1.1 may be helpful in understanding the relative efficiencies of each way.

5.3 Falseness and music

The `-M` options are entirely independent from the `-F` options. This has the unfortunate result that for many real-world searches, it will be necessary to duplicate information between `-Fr` options and `-Mlead` options so that both the falseness engine and the music analysis engine know which leads to look at. This is something that a future version of `methsearch` might address.

[5] Reverse rounds on n bells matches the `<`m`-run>` pattern $n - m + 1$ times, once for each of the separate m bell runs.

[6] This example will be of particular interest to anyone who wishes to replace Stonebow in David Hull's cyclic 23-spliced composition.

Chapter 6

Interoperability

As mentioned in §1.2.2, the Unix paradigm is not one of huge monolithic programs. Instead, it favours separate small programs that do one thing and do it well. This tends to result in greater flexibility, allowing programs to be used in novel combinations to achieve things that their original authors hadn't considered. This is the approach that `methsearch` attempts to take.

`methsearch` interacts with external programs in several main ways. First, it can filter input (in the form of a list of methods) from another source; this behaviour is activated by the `-I` option was discussed in §2.9. Secondly, it can output a list of methods which is parsed by another program; an example of this was also given in the discussion of filtering in §2.9. The final ways are when `methsearch` invokes an external program via a *command invocation* (§6.1).

In order that programs can pass information between each other, it is necessary that they share common way of representing the information. `methsearch`, and the other utilities in the Ringing Class Library, have two main interchange formats: *row streams* (§6.2) for passing lists of rows about, and *method streams* (§6.3) for passing lists of methods.

Also discussed in this chapter are some of more useful utility programs that can generate or parse these streams, however this document does not aim to fully document these utilities.

6.1 Command invocations

External commands can be invoked by `methsearch` in three situations. First, a command invocation may be embedded directly in a format string (§3.1); second,

it may be inside an expression (§3.2), which itself will either be in a format string or as the argument to a `-Q` option; and third, it can be used to generate a list of rows within a row expression (§4.4.1–4.4.2).

In the first case, the command is invoked once per method found. In the second case, the command is invoked once per method under consideration (as the output of the command invocation is used to determine whether the method is satisfactory). But in the third case, the command is only invoked once, at program start up.

In each case, the syntax is the same: `$(`*command line...* `)`. Here, *command line* is a command (or sequence of commands, such as a pipe line) that can be executed by the shell.[1] Standard output from the command is read by `methsearch`.[2] In a row expression this is expected to yield a row stream as documented in §4.4.2 and §6.2. A straightforward example is when it just invokes a single command, for example:

```
-Fr='$(printmethod -b8 -s 13578264 -F "&-3-4-5-6-2-3-4-7,2")'
```

This invokes the `printmethod` command to enumerate all of the rows in a lead of Yorkshire starting from the lead head `13578264`. This would be useful when trying to find a method for a composition of spliced that already contains this lead.

When a command is invoked from a (normal, non-row) expression (§3.2) or format string, `methsearch` will substitute any method variables (§3.1.1) or nested expressions (enclosed in `$[...]`) that it finds on the command line. (For obvious reasons this does not apply when evaluating row expression — there is no method under consideration, so the method variables would have an undefined value.) The command can generate arbitrary output which is like string literal (§3.2.2). However in most practical situations the string will subsequently be converted to an integer. For example, the `touchsearch` program (§6.3.1) could be invoked by a `-Q` option to check that a touch of the method is actually possible:

```
-Q'$(touchsearch -b6 -r -l720 -q --raw-count --limit=1          \
       -Cb=4 -Cs=$[$12h eq "12" ? "234" : "456"] $p)'
```

[1]The particular shell that is used to handle command invocations is governed by the `$SHELL` *environment variable*. If this is set, then the command line is passed as a single argument to a `$SHELL -c` process. If it is not set, on Unix-type systems, the default is to use `sh -c`, and on Windows, the default is to use `cmd.exe /c`.In most situations, the default behaviour should be fine as the `$SHELL` environment variable is normally set correctly by the shell.

[2]Anything written to standard error is ignored by `methsearch`, but will be displayed to the user. It is therefore recommended that if the program would routinely write to standard error this is discarded, for example on a Linux system by appending `2> /dev/null` to the command. The exit status of the command being invoked is currently ignored. This is likely to change.

This command invocation contains both a variable (the place notation, `$p`) and a more complicated expression used to determine whether the method is a seconds place one, in which case a 234 single is standard, or a sixths place one, in which case 456 single is required. The `--raw-count` option ensures that the command only prints the number of matching touches without surrounding text, and the `--limit=1` option halts the search after the first touch is found as we do not care about the number of matching touches.

In practice, this example can be handled more efficiently using the `--filter` option to `touchsearch` and using this to filter the output of `methsearch`.

Within format strings or non-row expressions, `methsearch` also accepts the sequence `$)` as a way of escaping a parenthesis so it can be included in the command to be executed. This is necessary because otherwise `methsearch` would interpret the `)` as terminating the command invocation. The same does not apply to command invocations in row expressions, a deficiency that will likely be addressed in a future version of `methsearch`.

6.1.1 Efficiency

Overuse of per-method command invocation can have a severe impact on efficiency. A good example of this is with music analysis (§5). It is possible to get all of the power of the various `-M` options (and more beyond) by invoking various external programs to analyse the music. For example, the following search for musical double surprise major methods

```
methsearch -b8 -SG1 -srdk -p2 -R'$q $M' -M'<4-runs>'
```

can be rewritten to invoke the external `printmethod` (§6.2.3) and `musgrep` (§6.2.5) programs instead of using `methsearch`'s in-built `$M` variable:[3]

```
methsearch -b8 -SG1 -srdk -p2                                    \
  -R'$q $(printmethod -b8 -cS $p | musgrep -b8 -s \"<4-runs>\")'
```

However, doing so slows the search dramatically. Timing the two commands on a 1.5 GHz laptop running Linux shows the former to take about 6.4 s, and the latter, 71.2 s — a factor of 11 times slower.[4]

[3] Both of these examples are presented here with Unix-style quoting. On Windows, the outer single quotes will need replacing with double quotes.

[4] This is due to the overhead of *forking* (that is, creating) a new process to evaluate the music. In this particular example, three processes are created for every one of the 4408 methods evaluated — one shell process, one `printmethod` process, and one `musgrep` process.

Examples with falseness can be even more extreme as `methsearch` is able to use knowledge of the falseness to prune large branches of the search tree in one go. A good instance of this is searching for CPS treble dodging major methods.

```
methsearch -b8 -G1 -srfp2 -FCPS
```

On the same laptop, this command took 77.7 s to complete. If instead the methods were filtered using an external program, as in the previous example, it is likely that this search would take days, if not weeks, to complete.

These examples illustrate that following the Unix philosophy of separate programs to its logical conclusion can sometimes have unfortunate consequences for efficiency. `methsearch` attempts to be pragmatic where a dogmatic separation of functionality would result in extreme inefficiency.

Another efficiency bottleneck can come from writing data to disk. `methsearch` only directly writes data to disk when a `-o` option (§2.7) is given, but indirect writing to disk can occur in two situations: when the machine has insufficient memory and *swaps* (that is, starts using the disk as additional memory), and when standard output is redirected by the shell to a file. `methsearch` is generally quite economical with its memory usage (an exception being when exceptionally lengthy `-H` statistics are being gathered as discussed in §3.3).

6.1.2 Exit status

When a command exits it communicates an *exit status* back to whatever launched it. This is an integer, and, by convention, zero is used to denote success and any non-zero number represents failure. Frequently programs will ignore the exit status of any other programs they spawn, as those programs are able to print a much more descriptive error messages in the event of failure. This is what `methsearch` does. However, occasionally it's useful to be able to find out whether or not a command exited successfully and the `$?` variable stores the exit status of the most recent command to be executed.

Before each format string is printed and any command invocations in it are executed, `$?` gets set to zero. The format string is then evaluated left-to-right with command invocations being invoked as they're encountered. This can be seen using the standard Unix programs `true` and `false` which simply exit successfully and unsuccessfully, respectively: `"$? $(false)$? $? $(true)$?"` will evaluate to `"0 1 1 0"`, with the value of `$?` changing through the format string.

To use the exit status instead of the output of a command in an expression (such as the argument to a `-Q` option), it's usually necessary to use the comma opera-

tor to discard the output of the command. For example `-Q"$(true), $?"` will always succeed, despite the fact that `true` does not provide any output; similarly, `-Q"$(false), $?"` will always fail. However, trying to use `-Q"$(true)"` will give an error as the textual output of `true` is neither true nor false.

6.2 Row streams

Row streams are used to stream a list of rows between programs. The standard syntax is very straightforward: one row per line.[5] The only place where `methsearch` uses row streams is when reading the output from command invocations in row expressions, for example in `-Fr` or `-Mlead` options.

The `extent` (§6.2.1), `rowcalc` (§6.2.2), `printmethod` (§6.2.3) and `gsiril` (§6.2.4) programs can be used to generate row streams. The `musgrep` (§6.2.5) program acts upon a row stream.[6] Both `rowcalc` and `musgrep` can be optionally be used as a row filter — that is, a program that takes one row stream and uses it to generate another.

6.2.1 Enumerating rows — `extent`

The `extent` program is a very simple utility for listing all of the rows on a given number of bells. It accepts the following options:

`-b`	`--bells=N`	The total number of bells
`-h`	`--hunts=N`	The number of fixed hunt bells at the front of each row
`-t`	`--tenors=N`	The number of fixed tenors at the back of each row
`-i`	`--in-course`	Only list in-course (even parity) rows

For example, `extent -b8 -h1 -t2 -i` lists the in-course tenors-together lead heads of the form `1.....78` of which there are 60. The rows are ordered lexicographically:[7] `12345678`, `12346578`, `12354678`, ..., `16543278`.

[5] Some programs will accept more permissive formats: for example by allowing (and ignoring) annotations after the row, by allowing multiple rows per line, or by accepting alternative separators. You should consult the documentation for the specific program before using these.

[6] The experimental `ringmethod` program also acts on a row stream which it converts into an audio WAV file by synthesising bell sounds; however, there is very little call for this program to be used in conjunction with `methsearch`, so it is not documented here. The `--help` text gives a brief summary of how to use it.

[7] Strictly, this is not quite lexicographic order as `E` and `T` (for 11 and 12) are found before `A` (13). So `1234567890ETA` comes before `1234567890EAT`.

It is rare to want to use this program in conjunction with `methsearch`. However, one use for this program is, in conjunction with `musgrep`, for calculating the maximum possible musical score. Assuming no rows have been given a negative score and that the stroke on which rows occur is irrelevant then the full extent must have the maximum score possible in a true touch. For example, the following command will confirm that the maximum 4-run score is 1200:[8]

```
extent -b8 | musgrep -b8 -s "<4-runs>"
```

6.2.2 Calculating rows — `rowcalc`

The `rowcalc` program is a stand-alone calculator for evaluating row expressions (§4.4.1). The only command line option is the optional `-b` option which specifies the number of bells. If this is supplied, the results are padded or truncated to that number of bells (and an error given if truncation produces an invalid row). The command line must contain a row expression. For example, the following command generates a list of the seven Plain Bob lead ends on eight bells:

```
rowcalc "<13254768*12436587>"
```

If the row expression contains the special `<(-)` file input (§4.4.2), it will read a row stream from standard input and apply the calculation to each row in turn. A interesting example of this might be calculating which leads of Cambridge Surprise Major contain reverse rounds. If l is a lead head and M is the set of rows in the first lead, then `87654321` $\in lM$ implies that lead l contains reverse rounds. This can be expressed as follows:

```
printmethod -b8 -F "&-3-4-25-36-4-5-6-7,2"                    \
  | rowcalc "87654321 / <(-)" | grep ^1
```

The final `grep` command is there to restrict the list of lead ends to those that have the treble leading.

6.2.3 Printing methods — `printmethod`

The `printmethod` program takes a method place notation and uses it to generate a list of rows in a lead or course. The following options are of particular relevance:[9]

[8] This has lead to a surprising degree of confusion. These runs are contained within 1002 rows, but because of the multiple counting of longer runs, some rows score more than once.

[9] For a full list of options, run `printmethod --help`.

`-b`	`--bells=N`	The total number of bells
`-c`	`--course`	Print a whole course instead of just a lead
`-F`	`--omit-final`	Omit the final row to avoid duplication of the lead head or course head
`-S`	`--omit-start`	Omit the starting row to avoid duplication of the lead head or course head
`-s`	`--start=ROW`	Start from `ROW` instead of rounds

The `-F` option is of especial importance. By default `printmethod` will print the rows in a lead from lead head to lead head, inclusive. For many applications this behaviour is not what is wanted and the `-F` option suppresses the final lead head.

This program is of particular use in command invocations in `-Q` or `-R` options. A example of this was given in §6.1.1 as an illustration of the degree to which unnecessary command invocations slow `methsearch` down. Nevertheless, if `methsearch`'s built-in falseness or music analysis isn't sufficiently powerful, this can be a useful technique.

6.2.4 Printing a touch — `gsiril`

`gsiril` is a tool for proving and printing compositions. It is designed to be familiar to anyone who has used other peal proving software such as MicroSiril or Sirilic. It also offers a significant number of additional features.

The composition must be specified in the `gsiril` language. It is beyond a scope of this manual to document `gsiril`.[10] By default this is read from standard input, however the `-f` option can be used to specify a file to read the composition from, or the `-e` option can be used to specify the composition on the command line.

If the `-E` option is passed to `gsiril`, then instead of proving the composition, the rows in it are printed without being proved. This row stream can then be processed by other programs. For example, Charles Middleton's 5600 Cambridge Surprise Major can be defined as follows:

```
8 bells

m ?= &-3-4-25-36-4-5-6-7
p  = m, +2
b  = m, +4
```

[10]Currently the most detailed source of documentation for `gsiril`, and in particular its language, can be found at `http://ex-parrot.com/~richard/gsiril/`. In the future, a manual similar to this will probably be produced.

```
M  = 2p,b,4p
MW = 2p,2b,3p
WH = 3p,b,2p,b
H  = 6p,b

prove 5(M,MW,WH,H,H)
```

If this is stored in a file named `middletons.gsir`, then the following command will count the CRUs in it:

```
gsiril -E -f middletons.gsir | musgrep -b8 -c "<CRUs>"
```

It is sometimes useful to invoke this from within `methsearch` — for example, to see which method is most musical for a given composition. As it stands, the file contains the place notation of Cambridge hard-coded into it. This can easily be resolved by removing the definition of `m` from the file and putting it on the command line instead. This allows it to be substituted by `$Q` in a command invocation, such as:

```
-R'$q  $(gsiril -b8 -Dm="$Q" -E -f middletons.gsir            \
            | musgrep -b8 -c "<CRUs>")'
```

It is important to note, however, that this does not ensure that the composition is true — it just analyses the music until it runs false or comes round.[11] For details on how to use `gsiril` to prove compositions, see §6.3.2.

It's often possible to rewrite examples like this so that instead of using a per-method command invocation it only invokes `gsiril` once when `methsearch` starts, and uses `methsearch`'s internal music testing abilities. In this case, we would use `gsiril` to generate a list of lead-heads in Middleton's and pass that to a `-Mlead` option. First, we modify the definition of `p` and `b` in `middletons.gsir`:

```
p  = "@", m, +2
b  = "@", m, +4
```

This makes `gsiril` print each lead head, which can then be passed to a `-Mlead` option. If lots of methods are being tested, this will speed the search up significantly.

[11] When run with `-E`, `gsiril` gives no visual indication of whether the composition is true or false: it just stops printing rows if it runs false. The exit status of `gsiril` will indicate whether or not the touch was true, but in POSIX-compliant shells (which most Unix shells are), the exit status of a pipeline is that of its last command. In the `bash` shell this can be changed by preceding the command with `set -o pipefail;` which causes the pipeline to exit unsuccessfully if `gsiril` fails. This can then be tested with `$?`, say by doing `$[$? && suppress]`. But such circumlocutions are best avoided where possible.

```
-Mlead='$(gsiril -qf middletons.gsir)' -M'<CRUs>' -R'$q $M'
```

6.2.5 Analysing music — `musgrep`

The `musgrep` program reads a row stream and analyses the music in it. The following options are of relevance:

`-b`	`--bells=N`	The total number of bells
`-p`	`--positive`	Print the number of rows with a positive score
`-n`	`--negative`	Print the number of rows with a negative score
`-c`	`--count`	Print the number of matching rows
`-s`	`--score`	Print the score for matching rows
`-i`	`--in-course`	Match only in-course rows
`-o`	`--out-of-course`	Match only out-of-course rows

In addition, the command line should contain one or more music pattern. These may be basic patterns (§5.1), named patterns (§5.1.1) or customised music scores (§5.1.2). The syntax for these is as given in chapter 5, except without the leading `-M`. When giving a negative score to a pattern the syntax (e.g. `-1:*28` to negatively score 28s at the back) looks like the standard form of an option because of the leading minus sign. This is resolved by preceding the music with the special option `--`. This tells `musgrep` to stop looking for options and to parse everything after it on the command line as an argument. This syntax is standard to most command line tools.

If no `-p`, `-n`, `-c` or `-s` is supplied, `musgrep` prints out all of the matching rows; if any of these are supplied, only the request count or counts are printed on a single line separated by tabs in the order `-p`, `-n`, `-c`, `s`.

The score, `-s`, is the total number of points assigned to the piece of ringing as described in §5. The count, `-c`, is the number of matching rows. The two can be different because a row can match more than one pattern (e.g. `42135678` is both a CRU and a 4-run), because a pattern can be assigned a score other than one (e.g. with `2:*5678` to score `5678`s at the back double), or because a pattern can match multiple times (e.g. `21345678` matched the 4-run pattern three times).

It is also possible for the positive and negative counts, `-p` and `-n`, not to sum to count, `-c`. This can happen if a row matches a pattern but has zero net score.

The `-i` and `-o` options can be used to filter the input stream to only in-course or out-of-course rows, respectively. This is applied before any pattern matching is applied. Even if `musgrep` is only being used to filter a row stream based on parity,

a pattern must still be supplied:[12]

```
musgrep -b8 -i ?
```

6.3 Method streams

Method streams are used to stream a list of methods between programs. The standard syntax is again very straightforward: methods are given, one per line, in any recognised form of place notation (§2.6). Whitespace is not permitted in the place notation, and any text after the first whitespace character on the line is ignored.[13] When invoked with the `-I` option (§2.9), `methsearch` requires a method stream on standard input. Frequently, `methsearch`'s output is a method stream — specifically whenever a `-R` option starting with `$p` or `$q` is given (or no `-R` is given) and no statistical output or counts are requested.[14]

The `touchsearch` (§6.3.1) and `gsiril` (§6.3.2) programs, as well as `methsearch` itself (§2.9), can be used as a filter on methods. In each case, this behaviour is activated with the program's `--filter` option (which for `methsearch` itself, can be abbreviated to `-I`). The programs all read a method stream from standard input, test whether each method has the desired properties, and if so, writes the method to its output method stream.

6.3.1 Searching for touches — `touchsearch`

The `touchsearch` program is a simple program to searching for touches in a single method. It accepts the following options.

[12] Using `?` as a catch-all pattern can be easier than `*` as most shells do not require `?` to be quoted, whereas `*` generally does need quoting.

[13] `methsearch` considers the space, tab, carriage return and line feed to be whitespace.

[14] A `-u` option is permitted as the status is written to standard error rather than standard output.

`-b`	`--bells=N`	The default number of bells.
`-C`	`--call=CHANGE`	Specify a call
`-r`	`--ignore-rotations`	Filter out touches only differing by a rotation
`-l`	`--length=MIN-MAX`	Length or range of lengths required
`-P`	`--part-end=ROW`	Specify a part end
`-m`	`--mutually-true-parts`	Only require mutually true parts without requiring them to be joined
`-q`	`--quiet`	Don't output the touches
`-c`	`--count`	Count the number of touches found
	`--raw-count`	Count the number of touches found, and print it without surrounding text
	`--limit=N`	Limit the search to the first `N` touches
	`--filter`	Run as a filter on a method library

In its most usual operation, `touchsearch` is invoked with the place notation for a single as a command line argument, and (unless `-q` is given) it outputs all the compositions it finds for that method. By using `-q --raw-count --limit=1` together, this behaviour can be adapted to simply check whether any compositions exist. An example of this is given in §6.1.

However it is `touchsearch`'s `--filter` option that is of interest here. This allows a single invocation of `touchsearch` to search a series of methods for one that can produce a touch of the specified type. For example, the following command searches for asymmetric plain minor methods that can actually produce an extent.

```
methsearch -b6 -Fe -Q'$y eq ""' -rf -m'*2'                    \
  | touchsearch -b6 --filter -r -l720 -Cb=4 -Cs=234 --limit=1
```

The `-Cb=4` and `-Cs=234` options are the two import ones. They define lead end calls using their place notation and provide them with names, '`b`' and '`s`' respectively. The names are optional, and `-Cb=4` can be simply written `C4`. However, if no name is supplied and the touch is to be printed out, the resulting output can be confusing.

The `-r` and `--limit=1` options don't actually change the result of the search but do speed it up: in the first case by rotationally pruning the search space, and in the latter, by stopping once one touch has been found. (A future version of `touchsearch` will probably handle this optimisation automatically.) The `-l720` says that a touch of exactly 720 changes is required. Ranges are also permitted in the `-l` option's argument: e.g. `-l540-720` requires any touch of between 540 and 720 changes, inclusive.

The `-P` option looks for multi-part compositions. For example, `-P1342` looks for

a three-part composition with 2,3,4 cycling. Note that, like `methsearch`'s `-FP` option (§4.5), this option specifies a part end group — the specified part end does not have to occur as the first part end. In this case, a composition bringing up the first part end `142356` would be acceptable. If `-m` is additionally specified, no effort is made to join the parts as long as a set of mutually true parts exist.

6.3.2 Proving a touch — `gsiril`

`gsiril` has already been mentioned in §6.2.4. As mentioned there, it is also possible to use `gsiril` as a filter to discover which methods are true to a particular composition. This is achieved by using the `--filter` option to `gsiril`.

In this mode, `gsiril` reads methods from its input stream and sets the `m` and `lh` variables[15] to the method place notation excluding the lead-end change (i.e. `methsearch`'s `$Q`), and the lead-end change respectively. A composition file in the `gsiril` language should be specified with the `-f` option, and the number of bells specified with `-b`. (The latter requirement is likely to be removed in a future version of `gsiril`.)

For example, to find right place methods true to Middleton's 5600 of Cambridge Surprise Major, the following invocation could be used:

```
methsearch -b8 -SG1 -p2 -l2 -srfw                                \
  | gsiril -b8 --filter -f middletons.gsir
```

In this example, the file `middletons.gsir` needs to slightly different from the version used in the example in §6.2.4. First, it needs to accommodate arbitrary seconds place leads, and secondly, it shouldn't define the place notation of the method (as this is passed via the `m` symbol). The following `gsiril` code will do the job.

```
8 bells

m ?= &-3-4-25-36-4-5-6-7
lh ?= +12

p = m,lh,"  @"
b = m,+4,"- @"

H = repeat( {b, /*8/:   b, break; p} )
```

[15]The names of these symbols can be overridden with `--lead-symbol` and `--lh-symbol` options, respectively.

```
W = repeat( {b, /*8?/:  b, break; p} )
M = repeat( {b, /*8??/: b, break; p} )

finish = repeat({rounds: break}, p)

prove 5(2M,2W,3H)
```

There are several things to note in this example. First, `m` and `lh` are given conditional definitions via the `?=` operator. This means that the definitions from the file are only used if `gsiril` does not provide a definition. This means that the file can be tested with the command:

```
gsiril -f middletons.gsir
```

Secondly, more complex definitions of the middle, wrong and home calling positions are needed to work with arbitrary lead ends including 8ths place methods.[16]

[16]A full explanation of how the these definitions work is beyond the scope of this document. The `gsiril` documentation found at `http://ex-parrot.com/~richard/gsiril/` contains a similar example in the section entitled 'Rule-based touches'.

Chapter 7

History of `methsearch`

7.1 Early development

I have no precise record of when I began writing `methsearch` though it must have been sometime in the first quarter of 2002. It was around this time that interest was developing in cyclic methods, largely driven by Philip Earis, and I found myself writing C++ code to explore the space of cyclic methods in search of methods that we might want to ring. The earliest reference I have found to `methsearch` is from an email exchange with Philip on 5 April 2002 when I enumerated (incompletely as it turned out) rotationally-symmetric cyclic plain major methods.

The first version of `methsearch`, or `meth-search` as it was then spelt, that I made available for anyone else was on 22 May 2002, and even at this stage it already had a surprising amount of functionality. `methsearch` releases have always been somewhat *ad hoc* and have not had meaningful release numbers. The following entries are dates when I advertised new Windows binaries and produced an accompanying list of new features. For the first two months this happened very frequently before slowing to a more steady rate. The earlier releases also involved a fair amount of bug fixing that is not mentioned below.

7.1.1 22 May 2002

In the first public version of `methsearch`, the `-b`, `-G`, `-S`, `-p`, `-l`, `-f`, `-r`, `-c`, `-e`, `A`, `-s`, `-k`, `-d`, `-q`, `-L`, `-H` and `-P` options all existed in much their present form. The `-w` option supported right-place methods on even stages, and the current `-R` functionality existed, though the option was called `-F`.

At this time, by default `methsearch` would only look for methods with no more than three places made at the lead-end or lead-head. A `-h` option disabled this behaviour and allowed arbitrary places. `methsearch`'s handling of falseness was still ill-defined, though probably behaved much like the current `-Fn` option. There was no facility for analysing music.

Format strings were supported for both the `-F` (now `-R`) and `-H` options and could only contain literal text, variables and a limited number of character escape sequences (namely `\t` and `\n`). Variables were written with a `%` rather than a `$` — the former is still supported but is deprecated, and only the following were available: `%l`, `%p`, `%n`, `%N`, `%C`, `%S`, `%%` and (probably) `%c` and `%b`.

7.1.2 27 May 2002

- There's now a `-j` option which prohibits adjacent places within the lead.
- Similarly to `-S`, `-T` searches for treble bob methods.
- It now has a status option (`-u`) which keeps a display of the the place notation currently being considered.
- The `-F` option has been renamed `-R` so that I can use `-F` for falseness.
- The new `-F` options (`-Fx`, `-Fn`, `-Fh`, `-Fl`, `-Fc`) can be used to filter out methods that are false with the lead or plain course. The default is `-Fh`.
- It recognises `$` as an alternative to `%` in format strings.

7.1.3 5 June 2002

- New `%`n`h` and (probably) `%`n`r` variables to print the nth change and row.
- The `--limit` option aborts the search after a specified number have been found.
- There's now the start of a `-M` option for musical analysis. Its argument is a musical pattern, e.g. `-M*1234*`. The `%M` variable prints the musical score.

7.1.4 19 June 2002

- Multiple hunt bells supported with `-U`.
- Right-place methods (`-w`) now supported on odd stages too.
- Testing for extent viability with `-Fe`.
- Counting the total number of methods with `--count`.
- Music analysis extended to handle named music including `<4-runs>`.

7.1.5 21 June 2002

- Testing for clear proof scale falseness: `-FCPS`.
- Preventing 'U' falseness `-FU`, generalised to requiring a round block of homes (specified using `--bob`) to be true.[1]
- Added `%q` for more concise place notation.

7.1.6 2 September 2002

- Added `-m` to partially specify the place notation.
- Generalised `-M<4-runs>` to `-M<`n`-runs>`.
- Added `--start-at`.
- Added `--hl-change` and `--le-change`.[2]
- Added `--cyclic-hlh`, `--cyclic-hle` and their `--rev-`... variants.
- Support for MicroSiril libraries.

7.2 In the Ringing Class Library

On 28 November 2002, the code was added to the Ringing Class Library CVS repository, and, at the same time, renamed `methsearch` (without a hyphen). This is the earliest code snapshot that is still available. From this point onwards, there is a complete version history of `methsearch`.

7.2.1 5 December 2002

- Program renamed from `meth-search` to `methsearch`.
- Code added to the Ringing Class Library CVS repository.
- Support for principles with `-U0` and `-n`.
- Changed `-j` to take an optional argument.
- Added the variables `%F` (only in-course tenors-together codes for regular palindromic major methods and cyclic rotationally-symmetric major methods) and `%o`.

[1] This functionality has since been removed.

[2] Both options have been since removed.

7.2.2 10 February 2003

- Added `-M<front-`n`-runs>` and `-M<back-`n`-runs>`.
- Added `--offset-cyclic`.

7.2.3 13 May 2003

- Added `--regular-hls`.
- Generalised `%F` to regular palindromic methods on any even stage greater than six.
- Removed `%F` support for cyclic rotationally-symmetric major methods.
- Added `%F` support for out-of-course falseness groups and ‘A’ falseness.
- Allow expressions in format strings using `$[...]`.
- Added the `suppress` and `abort` keywords.
- Improved masks: alternative blocks (e.g. `(36|34)`) and over/underworks (e.g. `&34.1.5.1.5/*`).
- Definition of `-FCPS` changed on six bells.
- Removed `--le-change` and `--hl-change`.
- Removed `-FU` and `--bob`.
- Added `--require`.

7.2.4 14 May 2003

- Allow string literals in expressions to be quoted with `'...'`.
- Added `$d` for the lead head code.

7.2.5 23 April 2004

- Removed the `-h` option.
- Added `-E`, `-o`, `-O` and `--raw-count`.
- Added support for XML output.
- The arguments to `-l` and `-p` are now optional — the default is 2.
- The `-j` option now applies to the lead head and lead end.
- Added `$Q`, `$u`, `$y`, `$O` and `-Fe+`.
- Music patterns may now take a score.

- Expressions may now contain the `.` operator.
- Support for command invocations `$(...)`.

Relatively little development happened during the next few years and it wasn't until 2009 that I next produced a release of `methsearch`.

7.2.6 15 June 2009

- Added `-Mcourse` and `-Mlead`.
- Added `-Fr` and support for row expressions, including file inputs and command invocations (though not set literals or group generator literals).
- Falseness groups can now be found with `-F:`*groups*.
- The `-C` option is now accepted as shorthand for `--count`.
- Added `--node-count`, `--prefix`, `--filter`.
- Multiple `--require` options can be given.

7.2.7 18 September 2009

- Added `-Mhalflead`, `-M2halflead`, `-Mrhalflead` and `-M2rhalflead`.

7.2.8 30 April 2010

- Wrote the first version of this manual.
- Little method: `-Z` (§2.1).
- Symmetric sections: `-y` (§2.2).
- Mirror symmetry: `--mirror` (§2.5).
- Improved falseness: `-Fs` (§4.4), `-FP` (§4.5).
- Library filtering: `--filter-lib` (§2.9).
- Random ordering: `--random`, `--random-count` [removed], `--seed` (§2.8).
- Method staticity: `$s` (§3.1).
- Support for three-bell methods.
- Set and group literals: `{...}`, `<...>` (§4.4.2).

7.3 Pending release

`methsearch` is getting close to the point where I shall build a release branded version 1.0 and announce it publicly on appropriate mailing lists. The following new features are now implemented ready for 1.0. The main features that are still missing are support for alliance methods and improved support for principles and differentials.

- Method identifiers: `$i` (§3.1.1).
- Filter payloads: `$a` (§2.9 and §3.1.1).
- Filter inverting: `--invert-filter` (§2.9).
- Delight methods: `--delight`, and (for minor) `--3rds-place-delight` and `--4ths-place-delight` (§2.1).
- Historical classes: `--strict-delight`, `--exercise`, `--strict-exercise`, `--pas-alla-tria` and `--pas-alla-tessera` (§2.1.1).
- Fourths-place doubles and minor lead-end symbols: `$D` (§3.1.1).
- Conjugation and set difference in row expressions: `^` (§4.4.1) and `-` (§4.4.2).
- Music pattern matching in expressions: `~~` (§3.2.1).
- Comma operator (for sequencing) in expressions: `,` (§3.2.1).
- Abbreviations for the `--require` and `--filter` options: `-Q` and `-I`, respectively (§2.9).
- Command exit status: `$?` (§6.1.2).
- Number of bells: `$B` (§3.1.1).
- Response files: `@`*filename* (§2.10).
- Customising bell symbols: `$BELL_SYMBOLS` environment variable (§2.6).
- Restricting or prohibiting changes: `--changes` (§2.6).
- Support for Unicode output in UTF-8: `-Outf8` (§2.7).
- Status update frequencies: `--status-freq` (§2.7).
- Allowing exceptions to `-p`n across the lead-end: `--long-le-place` (§2.2).
- Improved symmetry handling for principles: `--floating-sym` (§2.5).
- Improved random sampling: `--loop`, `--timeout` (§2.8); `--random-count`'s functionality subsumed in `--limit` and `--loop`, and the option removed.
- The current time: `$T` (§3.1.1).

Index

Underlined page references refer to the main definition of that option or variable. Subjects are not generally indexed when they appear in the discussion of a similarly-named option. For example, 'surprise methods' do not appear in the index because they are only discussed in conjunction with the `--surprise` option.

Colophon

This manual was typeset using XeLaTeX, a document typesetting package written by Jonathan Kew. It is derived from LaTeX, written by Leslie Lamport, but includes support for the Unicode character set and OpenType fonts. Both are derived from Donald Knuth's original TeX typesetting engine. These are all open source products and are freely available on a variety of operating systems, include GNU/Linux.

The main font used is Linux Libertine, an open source OpenType font designed by Philipp Poll. The sans serif font used in chapter and section headings is Linux Biolinum, also designed by Philipp Poll. Despite their names, neither font is specific to Linux. Equations are set using Donald Knuth's Computer Modern font. The monospaced font used for code fragments and row literals is the Computer Modern Teletype font, again by Donald Knuth.

All of the software described in this manual (except some passing references to a certain product gestated in Redmont) are themselves open source software. All of the ringing-related programs are available as part of the Ringing Class Library project, and `methsearch` uses it for manipulation rows, changes and methods, reading method libraries, analysing music, proof and much more. The Ringing Class Library was started by Martin Bright and contains contributions from Richard Smith and Mark Banner.

The source code for this manual is available from the Ringing Class Library CVS repository and is itself open source. Any additions or corrections to the manual will be gratefully received.