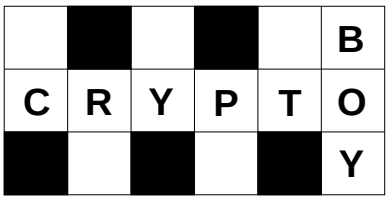
Cryptic Crossword Helper

Chris Brown

January 31, 2007



Contents

1 Goal

Idea is to provide help for creating cryptic crosswords. Cryptic because I like them but more because they have fewer words. Really the abstract goal is to cram as many *words* from a particular subject vocabulary into the *spaces* a crossword *grid* as possible. Done completely, this process would fill up the puzzle.

IF one knew such a solution was possible, the problem could be done with a constraint satisfaction algorithm that actually had failure. Clearly the spaces are variables and the words are values.

But it is pretty unrealistic to hope for that, and so one has to allow for a *blank word* to be assigned to a space, and that is compatible with any other *assignment* of words to spaces. Thus there is no possibility of constraints pruning the search space and we're left with the problem of enumeration of all the possibilities.

But again, why not nail down some words into spaces in advance? For one thing they can establish the theme of the puzzle, and for another if they have to be there they will establish strong constraints on (but only on) their *neighbors*, or spaces with which they share a letter. Enough of these fixed-value words scattered around should constrain the search (see Appendix A).

Further, one can do a hierarchy: pick a sub-theme (say *Pinafore*), within that pick your grid and gimmick, say locking two or more specific words you want into specific spaces, then run program using a Pinafore vocabulary. When that's done, pick a result you like, then lock all those filled spaces and get a new problem instance, which you run with the master vocab. minus the Pinafore words. This idea resulted in the puzzle in the *xword/4/* directory (see Appendix refcode).

2 Data Structures

For details, it's of course best to read the code (Appendix refcode), but my documentation and comments are minimal so be warned. There are many global variables, for instance. Meanwhile maybe this will help.

Words: For each *instance* of a problem (each time a new puzzle is to be generated), the idea is to have a special vocabulary, or *words*, that are to be crammed into the instance's *grid*. I have lists of words kept in a format ready to drop into a word array (e.g. Appendix refmasterwords). For each instance, the index of the global word array WORDS unambiguously defines the word. There is a BLANKWORD, whose number is 1 more than the size of the wordlist, and an UNASSIGNED value (-1) that denotes that a space has not yet been given a word or a blank value.

Grids: A grid (the global XGRID) is represented as an array of 1's (spaces) and 0's (stops?, anyway the black squares). I don't know how to make up these grids. Right now it takes me a lot of fiddling around to get a strategy. To help with bookkeeping, there is a program `makegrid.m` that allows editing and maintains either diagonal or 4-ways symmetry. For reasons of laziness the internal representation has an extra row and column of zeroes, added by `fixXGRID.m`. `makegrid.m` must be edited to set the grid size, then you can further edit to put up the basic matrix you want (where the 1 regularly goes somewhere in the 2x2 square of 0's), and then you can add 1's or 0's at particular rows and columns and have the right symmetrical thing happen. It reminds you that you can save the grid onto disk, where it is read by the `driver.m` (see below). Matlab stores it as a floating point array...so it's hard to read the disk file directly. One probably could fix that by more clever saving operation. The grid has CSPSIZE spaces: that number is the number of assignments of words

An *assignment* is an array mapping spaces onto words: thus it is CSPSIZE long, its members are initially the UNASSIGNED value, and we hope at the end it has lots of members that are positive word index values from the master WORDS list.

3 Algorithms

3.1 Basic algorithms

The only more-or-less interesting algorithms are the following.

1. `backtrack.m`: a recursive depth-first backtracking search, which never backtracks in a smart way because its assignments never fail. Thus it is a special case of Russell and Norvig's algorithm [1].

2. `fixnit.m`, `fixnbrs.m`: another special case – enforces constraints on neighbors of all nodes with only one value for their word. There is no need to try to propagate – the blank word possibility means there are no constraints generated by any node with a blank in its wordlist – so it's a one-pass operation.

3. `grid2cons.m`: takes an XGRID design (an array of 1's and 0's as defined above) and returns the constraints that need to be stuck into the CSP nodes. So it automates the constraint-representation process. It also fills up the global SPACEMAP, which is used by the next program to re-map assigned words back into the grid. SPACEMAP is a 2-D array: spaces(i,j) represents the ith space in the puzzle, with j = 1 is row it starts, j = 2 is col it starts. j = 3 is length, j = 4 is 0 if horizontal, 1 if vertical.

4. `asst2grid.m`: takes an $N \times 1$ assignment specification (above) in and produces a readble ascii grid of the result, something like the following, in which zeroes mean letters that the user now must fill in (originally there were lots of zeros in this assignment!).

```
 m h r w h
bolero inez
 r b d l r
rose eolian
 s   r i
westminster
   i c   s
battik kate
 r i d 0 h
carp hamlet
 c u u 0 r
```

3.2 The CSP Abstraction

Filling a crossword grid is a constraint satisfaction problem [1]. I represent it as a graph, each node repesents a space. A node has *neighbors*, which are the other spaces that intersect it and thus with whose words any word in this space will have to agree. Each node has two properties: a list of possible words (those that satisfy the unary constraint of being the right length) and a list of binary constraints (namely what letters in this space have to agree with letters in the node's neighbors). In all cases the "word" is actually its index in the instance's word array created by `makewords.m`

The node's word list usually looks something like [15, 16, 17, 18, 88, -99], where 88 is the value of BLANKWORD, saying the space can go unfilled, -99 is the stopper indicating "no more items", and the other positive values are word indices in WORDS. The driver puts these wordlists together in a cell array for feeding to the `struct` function that makes array of nodes representing the CSP. The CSP also has constraints; each node's (binary) constraints are represened by an array of the following format:

```
[ mylet1, nbr1word, nbr1let;
  mylet2, nbr2word, nbr2let;
  ...
  myletN, nbrNword, nbrNlet ]
```

That is, I have N letters shared with other spaces (those letters' positions in my space is column 1). For each such position, the perpendicular neighbor space intersecting there is represented by another node, whose index in the CSP array is given by the 2nd column. Last, the position of the intersecting letter in the neighbor is column 3. Luckily given the grid, the program *grid2cons.m* figures all this out.

For this all to make sense, it helps to know that the spaces (or nodes) have an implicit agreed order, which is given by the order of their first letter: left to right within top to bottom within horizontal, vertical. So in the grid below, bolero is first and esther is last.

Further, there is this weird ORDER idea, which is a stupidity that should probably just go. It starts trying to place larger words first. Maybe useful if the depth is limited with the MAXLEVEL variable: that allows cutting off search at some depth.

3.3 Makewords and Drivers

Annoying and error-prone, these are hand-crafted to put the pieces together.

I have one directory per puzzle instance, with each such directory having its own specially-edited program called `makewords.m`. This script makes an array of equally-long rows of characters, each row representing a word, padded by blanks to some standard length. This is a master global variable WORDS, and its indices correspond to individual words. The indices are used everywhere else in the programs to represent words. This program for the example of Appendix A appears in Appendix refmakewords.

The program *driver.m* or whatever you want to call it is also hand-crafted as to how many spaces must be filled, the maximum depth of the search, the filename for the grid, and some more parameters. Mainly it uses the word lists created in *makeword.m* and the constraints generated from the grid by *grid2cons* to construct the representation for the problem (see below). It then calls `fixit.m` to propagate constraints from fixed-value spaces, then `backtrack.m` to search for legal words for spaces. The driver for our continuing example is in Appendix C).

The resulting partial solution (an *assignment*) is presented to the user to finish up, edit, discard. The program *asst2grid.m* maps the assignment into the grid for viewing (see below).

4 Support and Graphics

Use *xfig* for the grid: these grids are basically repetitions of (1,0;0,0) or ... or (0,0;0,1), with various 0's filled in symmetrically afterwards. So I think the thing to do is make 2 rows using course grid, lines and filled squares, then duplicate them, then add any lines and other black squares needed. Save as .eps for use in LaTeX template that has grid, clues, answer.

Two websites I use a lot: check if a word exists given constraints on some letters with
`http://casr.adelaide.edu.au/cgi-bin/wow`, generate anagrams with
`http://www.easypeasy.com/anagrams/`

5 How-To Summary

1. create a directory for this puzzle instance and put all the following in it.

2. Come up with theme and grid. Use `makegrid.m` to create an save program-readable grid.

3. Create special vocabularies. All words must have same length, so pad with spaces. See the `words/` directory for examples.

4. Edit, munge, do whatever in `makewords.m` to cut and paste words from your vocubalary into the WORDS array. Add all-important no-compromise, one-item wordlists with -99 as second element for the themed placed words in the grid. What you want is a bunch of variables, which need not be declared global, for word lists for nodes in the CSP.

5. create some program, in my mind based on `driver.m`, to declare global vars, create the CSP from the grid and wordlists made previously. I have helper functions for all that, so I just copy the last driver over to the directory we're talking about, the one for the current problem instance. Set MAXLEVEL, whatever else occurs...I hope the code has adequate guides for what to do at this point.

6. Run it! you'll find you have to adjust the number of words placed (PLACEDTHRESHOLD). The results go from many to zero, in my limited experience, quite suddenly. So if the results go flickering by, up this number and wait until nothiing happens. Why there's not a graceful degradation is something I'm thinking about.

6 The Code

All available, with examples and wordlists, at
`http://www.cs.rochester.edu/u/brown/bio/xword/`.

7 Errors, Results and Problems

First, for the last two puzzles the only errors are human ones, caused by me giving contradictory, mistyped or inconsistently-constrained input. The results is often some array out of bounds.

I was hoping for some sort of graceful decline in the number of solutions as PLACEDTHRESOLD rises. So far it's feast or famine: for the runs so far, at one threshold, typically a little more than 1/2 the number of spaces (13 for the 21-long puzzle in Appendix refeg), there are scores if not hundreds of possibilities. Trying for 14 runs to completion with no success. So this leaves the puzzler to pick through this huge amount of possibilities, looking for bad bets (words ending in J, etc. etc.) and otherwise on his own. I'm sure there's some explanation for this, but it's not clear to me now.

Later, in the third puzzle (directory `xword/4`), I wound up with the four results shown in `xword/4/results`, so this is nicer. That example was highly constrained and there are interesting questions as to whether things would be better worse if it weren't so much. As I recall the possibilities dropped from 10^20 to 32 or something! Also that result illustrates the problems with forcing words to end in "o"...bad idea I didn't catch in time.

The next tries (in `xword/5`) exposed some difficulties. The first grid (unfortunately lost) had lots of long words, 9, 8, 7 letters, and so had too many constraints. The program only placed 10 G and S words, which anybody could have done by sticking to horizontal spaces. I tried, but had no luck filling in the rest.

The seecond try (in `xword/6`) is very much the other way, with only 3,4,5, and 6 letter words. I have no idea how many could ultimately be placed – I started running with it before spending time trying to improve it or running it to completion. I wound up forcing several words to solve problems in bottom half, and then forcing some more good-looking words it found, to cut down the search. Again unfortunately, I left running overnight but didn't check before killing to see if it found more... at 22 words placed, I think (see the code) it did its usual thing of hundreds of outputs. I just picked one more or less at random (not really, it looked promising but not that different from others and I only checked out about eight of them). So one wonders what would have happened with more time.

Still, this was a big pain, with a real struggle for clues and the resulting words not being nearly as densely G&S as in /4. Sigh. So this one was a bit discouraging.

One thing... in 3 or 4 cases the program generated complete or almost complete words, like asa, ageXt, lacXs, sort of thing. Kind of helpful!

There is still lots of setup (`makewords.m, driver.m`) that it would be nice to automate.

Another level entirely would be to try to automate the appeal to "crossword puzzle helper" to find words matching constraints, such as i?d?c, say, or m?t?c, which arose in our running example. I did such a thing for some cryptography applications (solving simple substitution cryptograms) by writing special-case PERL scripts that looked for patterns in `/usr/dict/words`, or somesuch early UNIX wordlist. That particular dictionary isn't around still but there are others out there I believe.

8 Clue Techniques

These stolen directly from `http://home.gil.com.au/~vburton/cryptics/intro.htm` strictly for my own information and inspiration. Many thanks!

8.1 synonyms

Clue: Used to be healthy so breathe out (6): Explanation: used to be = ex healthy = hale breathe out = exhale (meaning of answer)

Clue: Crooked, like London's famous gardens (5): Explanation: crooked = askew (meaning of answer) like = as London's famous gardens = Kew

Clue: Scribe gets low-down on Queen (8): Explanation: scribe = reporter (meaning of answer) low-down = report Queen = ER

8.2 anagrams

Indicated by words such as twisted, confused, turned, changing, for a change, breaking, broken, out, composed, bandied about, in different ways, disturbed, spilt, etc.

Red powder can somehow irk Papa (7): paprika

8.3 double definitions

Hmmm...

gift posted in advance (7) present: gift = present, in advance = pre-sent

Representative is a lady's partner (5) agent: representative = agent, a lady's partner = a gent

8.4 one in another

If the answer is a word or phrase which consists of two words, one completely enclosed within the other, the clue may indicate this by giving the two words or their meanings and using words within the clue such as within, inside, held in, encloses, enclosed, around, included, includes, surrounds, etc to indicate this.

Short avenue in city where the dentist finds a hole (6): cavity. short avenue = av in city = c(av)ity where the dentist finds a hole = cavity

Sky includes nothing back to become stealthy and furtive (6) slinky. nothing (nil) back(wards) = lin : s(lin)ky includes = slinky stealthy and furtive = slinky.

8.5 hidden in clue

Often, these letters span two or more words so it is not always easy to see them. In these clues, the words holds, inside, within, in, hiding, hidden, contains, includes, included, enclosed, from, etc may be used to indicate the presence of the answer within the clue.

In speech one may find repetition (4) echo. echo is within "speech one", repetition = echo.

8.6 abbreviations

```
L  learner
T  tea
AC or DC  current
Y  why
ie  that is
O  nothing (zero)
AM  morning
PM  afternoon
A  first letter
N  north
S  south
E  east
W  west
R  right
L  left
ER  Queen
PS  postscript
CA      California
```

If any of the above words is used in a clue, it may indicate that the abbreviation is part of the answer.

Woman is insane before morning (5) madam. woman = madam, insane = mad, morning = am, insane before morning = madam.

8.7 homophony

When homophones or sound-alikes are used, the clue will usually include something like I hear, spoken, mentioned, sounds, etc.

Well, this is lame but you get the idea:

Does your bear ring sound like the supporting part of a structure? (7) bearing bear ring to sound = bearing; supporting part of a structure = bearing.

8.8 foreign words

Obvious: here's one of mine along with roman numeral;

Crone has French nose after Roman one (4) (Inez).

8.9 initial or final letters

The first letters (or last letters) of some of the words given in the clue may be put together to make the entire answer or part of it. Indicated by "initially, starts, starting, ends, ending, finally", etc.

Obvious: my first was: Initially, Fiametta and Tessa elude destiny... (4). fate.

8.10 odd or even letters

Taking the odd letters in tart is cake would give trick while taking the even letters in trouble earn would give ruler.

An indication of the use of odd or even letters (or alternate letters) may be made by the use of clues such as oddly, evenly, odds, evens, other, every other and every other character.

Chain mail, oddly, in waterway (5) canal. chain mail (oddly, taking odd letters) = canal, waterway = canal.

8.11 roman numerals

I one (1); V five (5); X ten (10); L fifty (50); C one hundred (100); D five hundred (500); M one thousand (1000);

Crone has French nose after Roman one (4) (Inez).

8.12 reversing letters, words

Answer may include words given in the clue with their letters in reverse order. If letters are to be reversed, the clue may use words such as returned, going back or backward to indicate this.

This one's interestingly indirect (yikes, but nice) :

Very keen to go back to being a prima donna (4) diva. very keen = avid, very keen (avid) to go back (backwards) = diva, prima donna = diva,

8.13 word combinations

Two or more words may be combined, either in their original form or with their letters mixed (as anagrams), to make a longer word. If words are to be combined, the clue will usually indicate this by using words such as with, gets, after, and, have, joins, find, etc.

The miner gets it in the meantime (7) interim. miner gets it = interim (anagram of miner+it), meantime = interim.

Agree to trick a mongrel (6) concur. agree = concur, trick = con, mongrel = cur.

8.14 removing letters

If a clue uses the words almost, mainly, without, losing, abandons or nearly, this may indicate that most of the word given in the clue is used in the answer - but not quite all of it. For example, can is almost can't, joy is mainly enjoy and cur is nearly curt.

Almost return to move around an axis (4) turn. almost (most of) return = turn, move around an axis = turn.

Jan abandons jade-green award (6) degree. Jan abandons jade-green = degree (the letters jan removed), award = degree.

8.15 proper name abbreviations

If so, the clue may indicate the use of a shortened word form by including words such as little or short. Les Lesley, Jo Joseph, Al Albert, Bert Albert, Pam Pamela, Ed Edward, Pat Patrick, Ca California, Tas Tasmania.

Little Lesley follows want with sharp instruments (7) needles. little Lesley = les, want = need, les follows need = needles, sharp instruments = needles.

8.16 spoonerisms

a half-formed wish, a half-warmed fish. butterfly, flutter by. ease my tears, tease my ears. take a shower, shake a tower. waging of combat, caging of wombat.

Use Spooner in clue. Spooner says the arsonist might fight liar (5,4) light fire. an arsonist lights fires, Spooner's version of light fire is fight liar.

8.17 palindromes

Use of a palindrome in the clue may be indicated by inclusion of words/phrases such as both ways, from both ends, backwards and forwards, from both directions, etc.

Rev both ways, I'm in the middle, to get a pick-me-up from both ends (7) (reviver)

Exalted to the rank of a God from both directions (7) deified .

8.18 famous names

If one name of a famous person (either surname or first name) is used in a cryptic clue, this often indicates that the other name of that person is part of the answer. For example, the use of the word Pacino or Capone in a clue always means that the letters Al are part of the answer. Similarly, Lupino may mean the use of the letters Ida in the answer and Martin may mean the use of Dean.

Sometimes, however, a famous person (either past or present) may be referred to not by name but by reference to their title or position. For example, the clue may say former British PM to mean the name of a former Prime Minister of Britain or ex-President to mean the name of a former President of the USA.

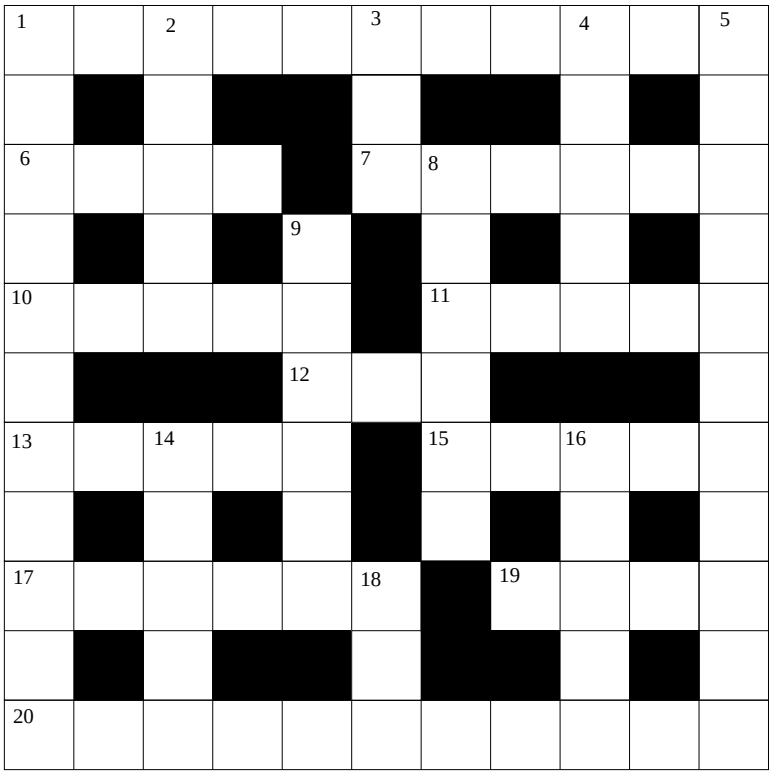
Tea left surrounding Lupino relating to the ebb and flow of the ocean (5) tidal.

8.19 repeats

Answer may consist of a sequence of letters repeated (in words such as koko or yumyum). Clue may indicate this by the use of words such as twice, doubles, repeated or again.

Start calling Otto up, Sally, twice over for the Tunisian dish (8) couscous.

A Example



Across

1. Family's inside scrap disturbed the fiancee. (8,3)
6. Filthy frog spoiled bets on Ernest. (4)
7. I hear Golden State fairways mean vice in paradise. (6)
10. Oh, cruel one to follow young sheep! (5)
11. Canonized in 1 Down. (5)
12. The star lost her music! (3)
13. Std cure: prescribe penicillin. (5)
15. Need disinfectant for 402 diseased Legionaires and Mostel. (5)
17. Particularly remiss in hiding particulates. (6)
19. Balanced on splayed feet. (4)
20. Grow toenail broken by Fairfax. (2,5,4)

Down

1. Goulash: prepare, dine with 11 Across. (4,3,4)
2. Dirs. for orch.'s use. (5)
3. Bald utterance or suchlike. (3)
4. Like original 1 Across, arthured by Alfred. (5)
5. Misinformed rival slew nun but reheated this seasonal chestnut. (2,4,5)
8. In Cairns, they analyze an inverse trig. fn. (6)

9. Sounds like a nice ride for a grenadier! (6)

14. Finger Robin's henchman. (5)

16. Celibate celebrity. (5)

15. Anger head of green land. (3)

Even with the words princessida intowergaol, peerandperi, anxmascarol, calynx, idyll, and etc fixed to specific spaces, there were more than 2×10^{18} possibilities to search. Propagating the constraints to these spaces' neighbors dropped the search to 943488 possibilities.

Answer

```
princessida
e n  t  d n
elsa calynx
r t a r l m
april celia
n   eos   s
depox iodic
p o i n i a
emissi fair
r n  r  n o
intowergaol
```

B Sample Makewords.m

```
% The list of words,
% Idea is each one gets unique ID, its
% row number.  Including blanks

%     global WORDS BLANKWORD;

WORDS = [
'etc         ';
'eos         ';  %Thespis
'ida         ';  %Ida
'bob         ';  % Becket,
'sam         ';  % Pirates
'zara        ';  %Utopia princess 6  <<---the word's index, for use
                 %                     with expandwords below
'daly        ';  %Vicar of Ploverleigh Sorc
'mars        ';  %Thespis
'lisa        ';  % Grand Duke soubrette
%
% etc etc etc...
%
'becket      ';  %Pina
'willis      ';  %Iol
```

```
'peepbo      ';  %Mik
'bailly      ';  %Utopia
'salata      ';  %Utopia maiden
'mylene      ';  %Utopia maiden
'phylla      ';  %Utopia maiden
'nekaya      ';    %Utopia princess younger  sis of Zara
'kalyba      ';   %Utopia princess younger  sis of Zara
'yumyum      ';   %Mik
'yeomen      ';  %yeo
'princessida ';  % 84
'peerandperi ';
'anxmascarol ';
'intowergaol ';
];

% define what the wordlists in nodes will look like. Up to the user
%Some are longer lists (ranges is easy) from array above,
%others are single words, which is where the constraints and puzzle
% theme come from.  -99 is a ‘‘stopper’’ showing end of list.
%  The -99 is for when we start deleting words in constraint
%  propagation.

BLANKWORD = size(WORDS,1) +1;
WORDS3 = expandwords(1,5);  % creates explicit lists from array index ranges
WORDS4 = expandwords(6,24);
WORDS5 = expandwords(25,45);
WORDS6 = expandwords(46,83);
WORDida = [84,-99];
WORDpeer = [85,-99];
WORDxmas = [86,-99];
WORDgaol = [87,-99];
WORDcal = [46,-99];
WORDetc = [1,-99];
WORDid = [27,-99];


%% end makewords.m
```

C Sample Driver.m

```
global CSP CSPSIZE;

global WORDS WORDS3 WORDS4 WORDS5 WORDS6 WORDS7 WORDS8 WORDS9;
global    WORDS10 WORDS11 WORDS12  BLANKWORD;
 global UNASSIGNED ORDER SPACEMAP;
    global PLACEDTHRESHOLD XGRID;
```

```
    global MAXLEVEL LEVEL CALLS;

global TEMPASST TEMPGRIDARR;

addpath /u/brown/xword/programs

UNASSIGNED = -1;

%%%%%%%%%change below for how many words need

     PLACEDTHRESHOLD = 13;


%%%%%%%%%%%% change next lines for different grids

     CSPSIZE = 21;


load grid3best
    XGRID = grid3best;
fixXGRID;              % puts in zeroes

%%%%%%%%%%%% this next line gets changed for different words.

makewords;   % generates global wordlists etc.

    %%%%%%%%And these (could be automated) need to be fixed
    % so the nodes, which I assume are done right in grid2cons,
%have right unary constraints...
    % so these are in order left to right top to bottom,
% first  horizontal, then vertical... room for screwing up here

    thewords = { WORDida;
 WORDS4;
 WORDcal;
                 WORDS5;
                 WORDS5;
 WORDS3;
                 WORDS5;
                 WORDS5;
 WORDS6;
 WORDS4;
 WORDgaol;
 WORDpeer;
                 WORDS5;
                 WORDS5;
 WORDS6;
```

```
 WORDetc;
 WORDS3;
 WORDS6;
                 WORDid;
                 WORDS5;
 WORDxmas};


assignment = UNASSIGNED * ones(CSPSIZE,1)

%%% here you can force assignments
%% we no longer do it this way though...
%    assignment(9) = 23;
%assignment(5) = 45;
%assignment(14) = 46;
%    forcedassts = 3;

%MAXLEVEL = CSPSIZE +1 - forcedassts;

MAXLEVEL = 22;

%%%%%%%%%%%%OK NOW ON AUTOMATIC



CALLS = 0;
LEVEL = 0;


%generates constraints and map of where to place assigned words.

[cons, SPACEMAP] = grid2cons(XGRID);




 CSP = struct('words', thewords,'constraints', cons);


ORDER = zeros(CSPSIZE,2);

    for (i=1:CSPSIZE)
    ORDER(i,1) =9999-size(CSP(i).constraints,1);
    ORDER(i,2) =i;
end;
ORDER = sortrows(ORDER)
```

```
ORDER(:,1) = []

assignment

    fixit;

backtrack(assignment);
```

D Sample List of Pinafore Words

```
%% Pinafore

======================
3
'         ';
'cat         ';
'tar         ';
'sea         ';
'tom         ';
'kcb         ';
'hms         ';
'sir         ';

============================
4
'        ';
'ahoy        ';
'blue        ';
'guns        ';
'glee        ';
'tide        ';
'dick        ';
'sail        ';
'noon        ';
'hebe        ';
'bill        ';

============================
5
'       ';
'berth       ';
'damme       ';
'balls       ';
'ocean      ';
'night       ';
```

```
'aunts       ';

============================
6
'      ';
'topman      ';
'polony      ';
'anchor      ';
'edward      ';
'tucker      ';
'cripps      ';
'porter      ';
'joseph      ';

============================
7

'     ';
'captain     ';
'bumboat     ';
'deadeye     ';
'topsail     ';
'marines     ';
'cousins     ';
'sisters     ';
'bobstay     ';

============================
8
'    ';
'corcoran    ';

============================
9
'   ';
'buttercup   ';
'tomtucker   ';
'josephine   ';
'rackstraw   ';

============================
10
'  ';

============================
11
' ';
```

```
'billbobstay ';

==========================
```

References

[1] S.J. Russell and P. Norvig. *Artificial Intelligence A Modern Approach (2nd Ed).* Prentice Hall, New Jersey, 2003.