A Constraint Satisfaction Algorithm for the Automated Decryption of Simple Substitution Ciphers

Michael Lucks
Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275

Abstract

This paper describes a systematic procedure for decrypting simple substitution ciphers with word divisions. The algorithm employs an exhaustive search in a large on-line dictionary for words that satisfy constraints on word length, letter position and letter multiplicity. The method does not rely on statistical or semantical properties of English, nor does it use any language-specific heuristics. The system is, in fact, language independent in the sense that it would work equally well over any language for which a sufficiently large dictionary exists on-line. To reduce the potentially high cost of locating all words that contain specified patterns, the dictionary is compiled into a database from which groups of words that satisfy simple constraints may be accessed simultaneously. The algorithm (using a relatively small dictionary of 19,000 entries) has been implemented in Franz Lisp on a Vax 11/780 computer running 4.3 BSD Unix. The system is frequently successful in a completely automated mode -- preliminary testing indicates about a 60% success rate, usually in less than three minutes of CPU time. If it fails, there exist interactive facilities, permitting the user to guide the search manually, that perform very well with minor human intervention.

1. Introduction

Despite its relative insecurity compared to modern encryption techniques, the simple substitution cipher remains a classical problem that has defied reliable automated decryption. Human cryptanalysis of substitution ciphers is usually begun by obtaining a trial entry to the code, i.e. guessing the decodings one or more letters. The initial guesses may be based on a variety of simple techniques, such as n-gram frequencies, doubled letters or short word patterns. The partial

S. Goldwasser (Ed.): Advances in Cryptology - CRYPTO '88, LNCS 403, pp. 132-144, 1990.

decryption yielded by the entry may then be used deduce full words through visual recognition and by observing syntactic and semantic patterns. The guessed words, in turn, yield further letter decodings and the process is repeated until the entire is message is deciphered. Some of the automated systems have attempted to imitate this method. Carroll and Martin [CM86], for instance, have developed a microcomputer-based program which utilizes expert system methodology to capture the knowledge and heuristics that an experienced cryptanalyst might employ in both the entry and deduction phases. Schatz [S77] uses singular value decomposition of a cipher's digram matrix to obtain a prediction of a cryptogram's vowels. Using the vowels and some special clues (e.g. one-letter words and apostrophes) as an entry, Schatz's program performs a heuristic search for words guided by a small vocabulary and a database of rules which reflect statistical properties of the English language. A very different method, proposed by Peleg and Rosenfeld [Peleg-Rosenfeld], employs a relaxation algorithm to determine all of the plaintext letters in parallel by iteratively updating the joint probabilities for the decoding of each ciphertext letter, with respect to its two nearest neighbors. The above systems assume that the plaintext conforms to various statistical properties of English. For long cryptograms this is a reasonable assumption, however messages that are short in length or contain uncommon combinations of letters (e.g. acronyms), are particularly difficult, if not impossible for such systems to solve.

An exhaustive search that generates all 26! keys is a reliable, but clearly impractical decryption method. A more reasonable (but still exhaustive) approach is to conduct the search at the word level, rather than at the letter level, using a large on-line dictionary. For each word in the ciphertext, the dictionary is searched for all of words that satisfy some known constraints. Since the ciphertext contains word divisions, word length is always a known constraint. Multiple occurrences of the same letter in the same word a second important pattern constraint. If the dictionary is complete, then each plaintext word must appear somewhere in the corresponding list of constrained words. If we examine all possible combinations from the constrained lists, the correct translation of the entire message must eventually appear. The search for the correct combination is conducted as a depth-first tree walk, in which each branch in the search tree corresponds to a guess for the decoding of a particular word in the ciphertext. Although the search space is initially very large, it is greatly reduced during the course of the search because each time a word is chosen as a possible decryption it imposes additional constraints upon other word that shares one or more of its letters. Hence, as a choices are made for each word, the set of possible choices for the other words becomes progressively smaller. Backtracking is performed whenever there are remaining words for which the set of potential decryptions is empty. Hence, if the dictionary is complete, the search will eventually find a set of choices for the ciphertext words which mutually satisfy all known constraints. With high probability, this set of words is very close to the correct plaintext. Even if some plaintext words are not in the dictionary, the constraints imposed

by those that are may be sufficient to provide an unambiguous decryption that is apparent by visual inspection. Wall [W80] describes such a procedure, but claims it is feasible only if special purpose hardware (a content addressable memory) is used to support parallel lookup of words from the dictionary. Wall actually implemented this method, simulating the parallel hardware via APL vector operations and excluding lookup time from the performance analysis. Our approach is much the same as Wall's, but with the following improvements:

1) no special hardware is required; instead the dictionary is compiled into a database designed to facilitate efficient lookup;
2) a control strategy is employed to guide the search toward promising paths;
3) the use of letter multiplicity (i.e. multiple occurrences of the same letter in the same word) as a constraint results in a much smaller search space;
4) certain guesses for words are recognized as yielding inconsistencies, and hence immediately rejected instead of being propagated further in the search.

2. The Database

Our system is based on an exhaustive search for pattern words in a dictionary of over 19,000 entries. The word search entails determining the set of words in the dictionary that satisfy specified constraints on word length, letter position and letter multiplicity. An example of such a pattern is the set all words with six letters having *e* in position 2 and *w* in position 5. A more complicated example is the set of all eight letter words ending in *t* in which the same letter occurs in positions 1, 5 and 7. Extracting such information by repeatedly scanning the dictionary for pattern matches would be impractically slow. Instead, the dictionary is compiled into a database that is partitioned according letter, word length and letter position. Associated with each letter in the alphabet is a list of numbered properties 1, 2,...*m*, where *m* is the maximum length of any word in the dictionary. The value of each property *j* is a vector V_j, indexed from 1 to *j*. If we want to find all words of length *n* containing the letter *l* in position *i*, we look on the property list of *l* and access the *i*th element of the vector found on property *n*. For instance, all 10 letter words containing *r* in position 6 are found by looking in the sixth entry of the vector found in property 10 on the property list of *r*. For simplicity, the database may also be viewed as a three dimensional array *D*, indexed by word length, letter and letter position, in which the entries are lists of words. For parameters *i*, *j* and *k*, an entry $D(i,j,k)$ would contain the list of all words of length *i* in which letter *j* occurs in position *k*. Words satisfying more complicated patterns are found by computing the union and intersection of one-letter patterns. The intersection of $D(9,b,4)$ and $D(9,w,7)$, for example, would be the set of all 9 letter words containing *b* in position 4 and *w* in position 7. To get all 6 letter words containing the same letter in positions 5 and 6, we take the union of all words having letter *l* in positions 5 and 6, where *l* ranges from *a* thru *z*.

The dictionary is compiled in a Franz Lisp session, separate from the execution of the decryption program. The resulting Lisp image may then be stored on disk to permit fast loading of the system. The Lisp image, including the database of approximately 19,700 words, occupies about 2.9 megabytes of disk space.

3. The Search Technique

Viewing a cipher as a list of words $[w_0, w_1, \ldots, w_n]$, our decryption process amounts to a state-space search in which each state T_i is a pair $[P_i,S_i]$. P_i is a list $[p_{i_0},p_{i_1},\ldots,p_{i_n}]$ where each p_{i_j} is itself a list containing all possible decryptions for word w_j in the ciphertext. S_i is a the current substitution list, i.e. a list of pairs of letters $([C_1,d_1], [C_2,d_2],\ldots,[C_m,d_m])$ indicating that letter d_k is currently assumed to be the decoding of the ciphertext letter C_k. Each node in the search tree represents a modified state which reflects the constraints imposed by a new guess for some ciphertext word. At the root node T_0, S_0 is empty and P_0 is obtained by searching the dictionary for the possible decryptions of each word, subject to the constraints of word length and multiple occurrences in a word of the same letter. For example, consider the following cryptogram taken from *The Dallas Morning News*:

MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM

Possible decryptions of the first ciphertext word are words that satisfy the pattern MZDDTK, i.e. all six-letter words having the same letter in positions 3 and 4. In this case, the word search routine returns a list of 320 words, [babble,bobbin,...,sizzle]. The same procedure is then repeated for each word in the ciphertext. Table 1 summarizes these initial possibilities.

Word	Ciphertext	Possible Decryptions	# Possibilities
w_0	MZDDTK	[babble,...,sizzle]	320
w_1	CJQLAPZZ	[absentee,...,megawatt]	90
w_2	DKDM	[afar,...,vivo]	39
w_3	CJQLNZPQ	[academia,...,bayberry]	130
w_4	TZJKDA	?	?
w_5	MPQBPQB	[alfalfa]	1
w_6	TNT	[ala,...,wow]	28
w_7	MNQBM	[aloha,...,widow]	93

Table 1. Initial possible decryptions of ciphertext words

The entries for the word TZJKDA are left blank because it has no multiple

occurrences of any letter. Its possible decipherments (all six-letter words) are so numerous (2,850 words) that it is best to postpone evaluation of this word, as will be discussed later.

Word	Ciphertext	Possible Decryptions	# Possibilities
w_0	MZDDTK	[cobble,...,sizzle]	246
w_1	CJQLAPZZ	[divorcee,...,kilowatt]	32
w_2	DKDM	[afar,...,vivo]	32
w_3	CJQLNZPQ	[charisma,...,petulant]	42
w_4	TZJKDA	?	?
w_5	MPQBPQB	-	0
w_6	TNT	[ala,...,wow]	28
w_7	MNQBM	[aloha,...,widow]	79

Table 2. Reduced possible decryptions of ciphertext words

The size of these initial lists of possibilities may be reduced considerably by removing *inconsistent* words, i.e. words that imply an ambiguous decryption key. For instance, *babble* is inconsistent with MZDDTK because it implies that both M and D translate to *b*. (In Wall's algorithm, such inconsistencies are not recognized.) Table 2 displays the reduced possibility lists in which inconsistent words have been extracted. Note that MPQBPQB has no possible decryption, i.e. the plaintext for the word doesn't appear in the dictionary.

The initial state of the search at the root node of the search tree is $T_0 = [P_0,S_0]$, where S_0 is an empty list and P_0 corresponds to the lists of possible decryptions in Table 2. For instance, p_{0_0} is the list of candidate decryptions for the first word, i.e. p_{0_0} = [cobble,...,sizzle]. Similarly, p_{0_1} = [divorcee,...,kilowatt], p_{0_2} = [afar,...,vivo], ... , p_{0_7} = [aloha,...,widow]. Each descendant node in the tree may be viewed as a guess for some word in the ciphertext. To expand the root node, a particular word is chosen from some p_{0_i} as a trial decryption of w_0. The successor state is $T_1 = [P_1,S_1]$, where S_1 is list of letter substitutions implied by the choice and P_1 is equal to $[P_{1_1},...,P_{1_{(i-1)}},P_{1_{(i+1)}},...,P_{1_7}]$, where each P_{1_j} is the subset of P_{0_j} whose words do not violate the new constraints imposed by S_1. (Note that P_1 does not contain the possible decryptions for w_i, since w_i is the word for which a guess is being made.) If *divorcee* is selected as trial decryption of w_1, for example, then $S_1 = [(C,d),(J,i),(Q,v),(L,o),(A,r),(P,c),(Z,e)]$. Each P_{1_j} is now filtered to remove words which conflict with S_1. The filter succeeds in two ways. In one case,

words are discarded because the same ciphertext letter has two decryptions, e.g. *charisma* is dismissed as a possible decryption for w_3 (CJQLNZPQ) because the implied decoding (C,c) conflicts with the assumption from S_1 that C decodes to *d*. The second way that filtering works is to eject words that require two different ciphertext letters to decode to the same plaintext. For example, *cobble* is no longer a possible decryption of w_1 because the required substitution of *c* for M conflicts with the constraint (P,c) in 5j. The new constraints also provide additional information about w_4 (TZJKDA), the word which had not been previously evaluated. Under the constraints (Z,e) and (D,c), the possible decryptions for for TZJKDA is now the set of all six-letter words containing e in position 2 and c in position 5. The result is a list of 21 words [deduce,...,select] (all of which get filtered out). If we had evaluated w_4 earlier, we would have to filter the entire list of 2,850 six-letter words in the dictionary.

The search space is greatly reduced by the seven constraints of S_v as indicated in Table 3 which corresponds to P_v

Word	Ciphertext	Possible Decryptions	# Possibilities
w_0	MZDDTK	[bellum]	1
w_2	DDTK	[alan,...,sash]	4
w_3	CJQLNZPQ	-	0
w_4	TZJKDA	-	0
«<s	MPQBPQB	-	0
w_5	TNT	[ala,...,tat]	8
w_7	MNQBM	-	0

Table 3. Possible decryptions at state 7\ = [P^SJ with w_1 decoded as CJQLAPZZ = divorcee

The node corresponding to state *T* may now be expanded by choosing among the 13 possible decodings for the w_0, w_2 $^{an<^{\wedge}\ w}$&- ^ *bellum* is chosen for w_0, the resulting additional constraints in state T_2 filter out all of the remaining possibilities for u;$_2$ and w_6, so we have reached a dead end in the search.

When a dead end is encountered, the trial plaintext under the current set of constraints is evaluated to decide whether or not the constraints yield a likely decryption of the ciphertext. The main criterion considered by the evaluation routine is the number of words in the ciphertext which are completely determined. The evaluation function awards points to any completed word, whether or not its decipherment is in the dictionary -- the mere fact that all of the letters can be unambiguously decoded is a positive sign. Greater weight, of course, is

given to words found in the dictionary and longer words are assigned more points than shorter ones. Extra credit is given to completed words that were not among those selected for expansion, i.e. words that were filled in as a result of other selections. If the score returned by the evaluation function is sufficiently high and is equal to or greater than the previous highest score, the current state is considered to be a possible solution and the trial plaintext is displayed to the user. In any case, the search continues by backtracking to the previous node and re-expanding with a new word choice.

In the present example, the constraints derived from the first selected word are sufficient to shrink the search space to a manageable level after the expansion of only one node — fortunately the selected word provides sufficient constraints. This is not always the case, however. For instance, if we had selected *ala* as a decryption for w_6 (TNT) at state T_o (instead of *divorcee* for Wj), P_x would contain far more possibilities, as shown in Table 4. Here the number of combinations of remaining possible decryptions is 3,456 (6x 6x 16x 6) rather than 32 (4x 8) as in Table 3.

Word	Ciphertext	Possible Decryptions	# Possibilities
w_0	MZDDTK	[giddap,...,hurray]	6
w_x	CJQLAPZZ	[divorcee,...,princess]	6
w_1	DKDM	[divorcee,...,princess]	]6
w_3	CJQLNZPQ	[preclude]	1
w_4	TZJKDA	-	0
w_5	MPQBPQB	-	0
w_7	MNQBM	[elide,...,plump]	6

Table 4. Possible decryptions at state $T_1 = [P_1, S_1]$ with *WQ* decoded as TNT = *ala*.

The striking contrast is due, of course, to the difference in the number of constraints imposed by the two choices. Word w_x has 7 distinct letters, yielding 7 constraints, as opposed to only 2 constraints produced by the 2 distinct letters in w_6. Short words (i.e. words with less than 5 letters) hence pose a problem for our algorithm. Not only do they fail to provide the desired constraints on other words, they also are less likely to be filtered out themselves because there are fewer possibilities for letter conflicts. If there are several unresolved short words, the combinatorics involved in checking all possible combinations rapidly gets out of hand. Since most cryptograms contain a high percentage of such words, the full tree may be extremely bushy and a complete traversal usually cannot be

executed in a reasonable amount of time. It is therefore advisable that the traversal be directed toward cheap, promising paths and steered away from expensive, dubious ones, so that a satisfactory solution may be displayed to the user at a relatively early stage of the search. Fortunately, it is usually possible to achieve this goal if we are careful in the selection of nodes to be expanded.

To deal with the short word problem, the ciphertext is separated into two groups. Group A contains the longer words in the message, i.e. words of six or more letters, while group B contains the rest. (If there are not three or more long words in the message, the definition of "long" is dynamically redefined so that there are at least three.) No words from group B are considered for expansion until all of the words in group A have been either been expanded or have no remaining possible decryptions. By examining longer words first we hope that the search space will already be somewhat constrained before the short words are processed. When only words from group B remain, the current state is evaluated and a decision is made whether to continue on the current path or to backtrack. The node is expanded only if there is some evidence that the current path looks promising or if the cost of expansion is relatively small. The primary measure for evaluating the promise of a path is the number of completely deciphered words which are also found in the dictionary, particularly words that were not chosen as guesses. A secondary measure is the number of letters remaining to be deciphered -- the fewer the better. If a path is not found to be promising by the above criteria, the next node may still be expanded if it can be done cheaply, i.e. if the number of successors is small and the tree is already of sufficient depth.

Another useful heuristic for optimizing the search is Wall's suggestion that the most constrained word, i.e. the word with the fewest number of possible decryptions, should be expanded first. If a word found in the dictionary happens to be highly constrained at the root node, expanding it right away will almost always yield a speedy correct decryption because the search converges very fast once the right path is found. (This rule should be subordinate to the short word heuristics, however -- a short word should not be expanded prior to a long one even if it is more constrained.)

The workings of the search may be illustrated by completing the decryption of our example. (An abbreviated trace of the search is found in the appendix.) The words in group A are w_0, w_1, w_3, w_4 and w_5. In the initial state (Table 2) the most constrained long word is w_1, with 32 possibilities, hence CJQLAPZZ is chosen to be expanded first. From its list of possible decryptions, *kilowatt* is selected as the first trial word. Since there are no possible decryptions for any of the other long words under the new constraints, this choice is rejected and the search immediately backtracks and the word *waitress* is tried. This choice is rejected for the same reason, as are the next 10 choices for w_1. The first trial guess for w_1 that is considered promising is *buckaroo*. This path is considered worthy to pursue because it allows another long word (w_0) to be deciphered into a word appearing in the dictionary, namely *sodden*. After *sodden* is selected to

expand w_0, the state is considered promising enough to warrant expansion of a short word, so *eye* is chosen for TNT. At this point a dead end is reached. Since there has been no previous solution offered, the current decryption is the best available so far, hence it is displayed to the user as:

```
sodden buckaroo dnds buckyorc eounda src-rc- eye syc-s
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM
```

As shown in the appendix, the search now backtracks to *sodden* and selects *ewe* for TNT. This yields a solution which appears equally good as the first, so it too is displayed. After 13 possible solutions involving *buckaroo* are discovered, the search backtracks to top level and other choices are tried for w_1. Several other paths are explored, but the depth of the search never exceeds 2. Eventually *mandrill* is selected for w_1, which happens to be the correct decryption. This leads immediately to *mandolin* for w_3. The only word that now satisfies the constraints for w_4 is *slater* (*player* is not in the dictionary). This path terminates in the solution

```
-leest mandrill ete- mandolin slater -in-in- sos -on--
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM
```

which is still very obscure. Backtracking to the *mandolin* level, *sleety* is now tried for w_0, yielding the somewhat intelligible

```
sleety mandrill eyes mandolin tlayer sin-in- tot son-s
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM
```

The next choice for w_0 is *sleepy* which yields the correct answer

```
sleepy mandrill eyes mandolin player sin-in- pop son-s
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM
```

The full plaintext is obvious by inspection, however there is no way for the system to determine that B decodes to *g* because neither *singing* nor *songs* is in the dictionary and only these words contain *g*. (Since *player* is not in the dictionary, the score of 4100 is no better than the score of the previous decryption.)

4. Interactive Mode

When the system fails in the fully automated mode, a backup interactive mode is provided through which the user may analyze the cipher and supply his/her own guesses for letters. Commands exist which permit the user to display first order statistics, to add and delete guesses for letters, and to simultaneously display the message and its partial decryption. With some guesses for letters, the automated search may then be repeated, this time guided by the user-supplied constraints. In many cases where the automated system fails, a successful decryption is achieved via correct guesses for only one or two letters.

5. Extensions

The current system might be improved in a variety of ways that have yet to be attempted. An ability to recognize plural, prefixed and suffixed forms as words, for instance, would take care of the majority of examples that the present system can't handle automatically. Whether these forms should be added to the dictionary (at the cost of a significantly larger search space) or detected by a separate routine is under investigation. A second desirable extension would be to integrate the various heuristics and statistical approaches found in [S77], [PR79], [CM86], [A84] and [A86]. The information obtained from the statistical analyses might be valuable both in guiding the automated search as well as aiding the interactive user. Finally, an moderate improvement in performance would almost certainly result from a careful editing of the dictionary, which currently contains a many extremely rare words and omits many common ones. It would also be desirable to order the words in the database, so that more frequently used words are considered first. These tedious tasks have not yet been undertaken.

6. Performance

The system has been implemented in Franz Lisp on a Vax 11/780 computer. In tests on more than 100 examples chosen at random from newspapers and magazines, the system was successful in a completely automated mode about 60% of the time. Usually the solution was obtained in less than three minutes of CPU time. In approximately 30% of the trials, the program required rather trivial human intervention, such as the guessing of a common short word such as *the* or *and*. Failure most commonly occurred on examples in which none of the longer words in the plaintext were present in the dictionary. This situation occurs, for instance, when all of the long words are plurals or suffixed, since these forms are not likely to be found in our limited dictionary. When this happens, the system is forced to use small words as trial entries, thereby establishing few constraints and hence greatly expanding the search space. The second most common cause of failure was that none of the words in the plaintext contained any repeated letters. In this case, the program is unable to proceed (unless there are some one-letter words) because there are no entry candidates. This situation is most likely to arise in very short messages or in examples composed mostly of short words.

7. Conclusions

We have described an automated method for decrypting simple substitution ciphers based on exhaustive search and controlled thru constraints imposed by word patterns. No statistical analyses or language-specific heuristics are employed. Although quite successful in its own right, we believe that the technique could be used as a driver to an even more powerful system in which heuristics and statistical information would assist in directing the search. This hybrid approach would exploit the somewhat unstructured methods of the human

cryptanalyst while retaining the systematic character of the exhaustive search that enables successful automation.

Acknowledgement

I would like to thank Dr. James G. Dunham, SMU Dept. of Electrical Engineering, for his advice and assistance in the development of this project.

References

[A84] Roland Anderson, "Finding Vowels in Simple Substitution Ciphers by Computer", Cryptologia, vol. 8, no. 4, Oct. 1984, pp. 348-358.

[A86] Roland Anderson, "Improving the Machine Recognition of Vowels in Simple Substitution Ciphers", Cryptologia, vol. 10, no. 1, Jan. 1986, pp.10-33.

[CM86] John H. Carroll and Steve Martin, "The Automated Cryptanalysis of Substitution Ciphers", Cryptologia, vol. 10, no. 4, Oct. 1986, pp. 193-209.

[PR79] Shmuel Peleg and Azriel Rosenfeld, "Breaking Substitution Ciphers Using a Relaxation Algorithm", CACM, vol. 22, no. 11, Nov. 1979, pp. 598-605.

[S77] Bruce R. Schatz, "Automated Analysis of Cryptograms", Cryptologia, vol. 1, no. 2, April 1977, pp. 116-142.

[W80] Rajendra Wall, "Decryption of Substitution Cyphers with Word Divisions Using a Content Addressable Memory", Cryptologia, vol. 4, no. 2, April 1980, pp. 109-115.

Appendix

Program Execution with Trace of Word Search

The current depth of the search tree is indicated by the number on the left. The ciphertext of the word currently being examined is denoted in upper case, while the trial decryption for the word is in lower case. (A portion of the trace has been omitted to save space.)

```
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM

0 CJQLAPZZ kilowatt
0 CJQLAPZZ waitress
0 CJQLAPZZ ruthless
```

```
0 CJQLAPZZ princess
0 CJQLAPZZ marquess
0 CJQLAPZZ giantess
0 CJQLAPZZ dutchess
0 CJQLAPZZ congress
0 CJQLAPZZ compress
0 CJQLAPZZ baroness
0 CJQLAPZZ buckaroo
|1 MZDDTK sodden
||2 TNT eye

*** -- Solution #1
       Score = 3100
sodden buckaroo dnds buckyorc eounda src-rc- eye syc-s
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM

||2 TNT ewe

*** -- Solution #2
       Score = 3100
sodden buckaroo dnds buckworc eounda src-rc- ewe swc-s
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM

||2 TNT eve

*** -- Solution #3
       Score = 3100
sodden buckaroo dnds buckvorc eounda src-rc- eve svc-s
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM

|1 MZDDTK toddle
|1 MZDDTK soffit
|1 MZDDTK joggle
|1 MZDDTK toggle
|1 MZDDTK pollen
||2 TNT eye

*** -- Solution #4
       Score = 3100

pollen buckaroo lnlp buckyorc eounla prc-rc- eye pyc-p
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM
                          .
                          .
                          .

{ To save space, the next 15 trial solutions are omitted }
                          .
                          .
                          .

0 CJQLAPZZ nutshell
|1 MZDDTK bloody
||2 TNT did
```

```
*** -- Solution #20
       Score = 3100
bloody nutshell oyob nutsilet dluyoh bet-et- did bit-b
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM

||2 TNT dad

*** -- Solution #21
       Score = 3100
bloody nutshell oyob nutsalet dluyoh bet-et- dad bat-b
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM

|1 MZDDTK gloomy
0 CJQLAPZZ mandrill
|1 CJQLNZPQ mandolin
||2 TZJKDA slater

*** -- Solution #22
       Score = 3100
-leest mandrill ete- mandolin slater -in-in- sos -on--
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM

||2 MZDDTK sleety

*** -- Solution #23
       Score = 4100
sleety mandrill eyes mandolin tlayer sin-in- tot son-s
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM


||2 MZDDTK sleepy

*** -- Solution #24
       Score = 4100
sleepy mandrill eyes mandolin player sin-in- pop son-s
MZDDTK CJQLAPZZ DKDM CJQLNZPQ TZJKDA MPQBPQB TNT MNQBM
```