THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 52, No. 7, September, 1973
Printed in U.S.A.

Heuristic Solution of a Signal Design Optimization Problem

By B. W. KERNIGHAN and S. LIN

(Manuscript received February 15, 1973)

This paper discusses a heuristic solution procedure for a combinatorial optimization problem that originates in designing signal constellations for modems.

The design problem is to place m signals in a two-dimensional space to minimize the average error rate under specified noise conditions, using a maximum-likelihood decoding scheme. Intuitively, it amounts (roughly) to spreading the signal points as far apart as possible, according to the distance measurement implied by the noise function.

We show how this problem can be reduced to a discrete one: Given an ℓ by n matrix P, and $m < \ell$, find an m-row subset $M = \{i_1, \cdots, i_m\}$ of the rows of P that maximizes

$$\sum_{j=1}^{n} \max_{i \in M} p_{ij},$$

and then describe an efficient procedure for finding this maximizing set.

Experiments indicate that the procedure is a useful tool, both for analysis of existing and proposed signal constellations and for finding new, near-optimum ones.

I. THE PHYSICAL PROBLEM

This paper discusses a heuristic procedure for solving a combinatorial optimization problem that arises in designing signal constellations for modems. The solution method is also applicable to the covering problem; we will discuss this at the end of this section.

The underlying physical problem is the following: A digital signal s is to be sent through a noisy channel. In general, s may take on only a finite number, m, of distinct values. (In practice, m will be a power of 2.) Since the transmission line is an analog device, any specific s value is encoded for transmission by modulating a carrier wave for a period of time. For instance, s might take on only the two values 0 and

1, in which case the modulation might be to send either of two amplitudes (amplitude modulation) or to send either of two frequencies (frequency modulation).

The modulation considered here is more complex: A specific value of s will be encoded as

$$g(t)[a \sin \omega_c t + b \cos \omega_c t],$$

where ω_c is the carrier frequency, g is an appropriate pulse shape, and a and b are the amplitudes of the sine and cosine components.

The signal s is quantized into one of m levels; for each level, there is a corresponding a and b, so there are m different (a, b) pairs.

The received signal, as always, is corrupted by noise, so a sample value sent as (a, b) is received as (a', b'). At the receiver, a decoder processes this corrupted (a', b') and attempts, according to some criterion for minimizing the error rate, to reconstruct the (a, b) which was sent originally.

The design question is: What (a, b) pairs should be chosen to minimize the error rate for this decoding process?

In the version of the problem we solve, the main constraint is that $a^2 + b^2$ is bounded for all (a, b) pairs, which implies that peak signal power is bounded. A related but more difficult problem requires that the average of $a^2 + b^2$ over the (a, b) pairs is constant; this corresponds to a bound on average power. We discuss this problem in Section VII.

The combinatorial optimization problem is a discrete version of this design question. Let us replace the continuum of points that could represent (a, b) values (everything inside the circle $a^2 + b^2 = 1$) by ℓ discrete points, spread more or less uniformly throughout the region. We call these "allowable" signal values. Since the noise may add to the received amplitude, the received signal can in fact be outside this circle. Let us define additional $n - \ell$ discrete points to represent the additional possible received signals that lie outside the circle $a^2 + b^2 = 1$.

Now define an ℓ by n matrix $P = \{p_{ij}\}$ by

p_{ij} = probability that, if signal i were sent, it would be received as (discrete) point j

$$i = 1, \cdots, \ell, \quad j = 1, \cdots, n.$$

Suppose for convenience that the chosen signal values are points 1 to m (i.e., rows 1 through m of P). The decoding procedure to be used is simply this: If j is the received signal, it is decoded as that i in $1, \cdots, m$ for which p_{ij} is maximum. If $1, \cdots, m$ have equal *a priori* probabilities, this procedure minimizes the error probability.

The probability that a particular signal i is decoded correctly is the probability that it falls into a column j where it is the largest entry. The probability of correct decoding using any particular set of m rows of P is thus the sum of the column maximum elements in those m rows (divided by m). The problem (at last) is to find those m rows that maximize this probability; these will be the signals used. More formally, find an m-row subset $M = \{i_1, \cdots, i_m\}$ of the rows of P that maximizes

$$V_M = \sum_{j=1}^{n} \max_{i \in M} p_{ij}.$$

We call V_M the *value* of the subset M.

In the physical problem, $m < \ell < n$; as an abstract problem, the latter inequality is unnecessary.

Note that the algorithm we will present is essentially insensitive to the characteristics of the matrix P. In practice, this means, for example, that any noise characteristics can be treated effectively. This includes not only the classical additive white noise, but also phase and amplitude jitter components.

As an example of a matrix with quite different characteristics, suppose P has entries which are either 0 or 1. A *covering problem* is "Find a minimum set of rows of P such that these rows together contain a 1 in each column of P." We can use our heuristic procedure to find approximate solutions for the covering problem as follows: Find a maximum value solution of the original problem, using m rows. If the value is less than the number of columns of P, increase m; if it equals the number of columns, decrease m. Find a new solution with the new m. The smallest value of m for which the value equals the number of columns is a minimum cover of P.

II. A HEURISTIC PROCEDURE

The process is based on iterative improvement of random initial solutions. A random set of m rows is chosen from the ℓ possible rows. (In practice, of course, we can also let the procedure try to improve on a specific initial set.) We augment the m initial rows by one row chosen from the $\ell - m$ unused rows, giving us $m + 1$ rows. We then compute which of these $m + 1$ rows contributes the least to the value of the set and remove it. (The row removed might well be the row added.) We then move to the next row in the unused ones and add it to the current m rows. The process terminates when all the $\ell - m$ currently unused rows have been examined without finding a profitable

replacement. This defines a local optimum solution. We then iterate the entire procedure from a new random start.

The resulting solutions are "1-opt" in the sense of Reference 1—that is, no exchange of a single pair of rows can improve the solution. Although 1-opt procedures are among the weaker heuristics, the results are quite acceptable, as we shall see in the next section.

The process is very fast which counteracts the lower effectiveness of 1-opting: a 100 by 100 problem takes about one second on the Honeywell 6070 (in FORTRAN A). The run time for dense matrices is essentially proportional to ℓn and independent of m. For sparse matrices (the situation that occurs in practice), the run time varies only with the number of non-zero elements, which for real problems is proportional to ℓn.

The process is fast, partly because care is taken to do no extra work. In detail, a basic step of the algorithm is as follows (the next section contains a numerical example, which can be followed in parallel):

Initialization. Suppose without loss of generality that the initial rows are $1, \cdots, m$. Call this set M. Let $v(i)$, $i = 1, \cdots, m$, be the *decrease* in value if row i is removed from M. We will compute v. This is done only once per local optimum solution.

We begin by setting $v(\cdot) = 0$. Each column j $(1 \leqq j \leqq n)$ contributes an increment to exactly one component of v, as follows. Find x_j and y_j, the largest and second-largest elements among the first m elements of column j. Record these, and also the rows in which they were found, i_x and i_y $(1 \leqq i_x, i_y \leqq m)$. Now, since x_j is the largest element in column j, it determines the contribution that column makes to the value of M. But if row i_x were removed from M, the contribution of column j would be determined by y_j, the second largest element. Thus, the decrease in value that would result if row i_x were removed from M is $x_j - y_j$, so we add $x_j - y_j$ to $v(i_x)$. This process is done for each column.

Phase 1—Evaluation of a Replacement Row. When the initialization is finished, we evaluate replacement rows. Suppose row r $(m + 1 \leqq r \leqq \ell)$ is the next proposed replacement. We will compute which of the $m + 1$ rows in $M_r = \{1, \cdots, m, r\}$ decreases the value of M_r least when removed. We will do this without examining any of the matrix P except for row r itself.

Let $\Delta(i)$ be the *change* in $v(i)$ that results if row i is removed from M_r. By computing Δ, we do not need to change v unless we are actually going to exchange two rows. Initially, let $\Delta(i) = 0$, $i = 1, \cdots, m, r$. (Let $v(r) = 0$ as well.) The value $\Delta(i)$ will be $\leqq 0$ for i in M while $\Delta(r) \geqq 0$.

For each column j, let $z = p_{rj}$, $x = x_j$, and $y = y_j$; we will do one of (*i*), (*ii*), or (*iii*):

(*i*) If $z \leqq y$, the new element is smaller than second best; no action is necessary.

(*ii*) If $y < z \leqq x$, z is a new second-best element. The Δ value for the row containing x, $\Delta(i_x)$, must be decreased by $z - y$, since the contribution to $v(i_x)$ from column j is now $x - z$ instead of $x - y$.

(*iii*) If $z > x$, we have a new largest element in the column. Add $z - x$ to $\Delta(r)$ (x is now second largest) and subtract $x - y$ from $\Delta(i_x)$, since row i_x no longer contains the largest element in this column.

Phase 2—Determination of Which Row to Remove. We have now determined $\Delta(i)$, the change in value that would result if row i were removed from M_r. Find the minimum among $v(1) + \Delta(1)$, $\cdots$, $v(m) + \Delta(m)$, $v(r) + \Delta(r)$.

If the minimum occurs at row $k \neq r$, let us say, then we must exchange rows r and k, update $v(\cdot)$, and update the records of largest and second-largest elements for each column. Go to Phase 3.

If this minimum occurs at r, there is no profit in replacing one of $1, \cdots, m$ by r. If $\ell - m$ rows have been consecutively examined without profit, we have finished; the set M is the local optimum solution. Otherwise, we must set r to $r + 1$ (wrapping around from ℓ to $m + 1$ if necessary) and go back to the Phase 1 calculation.

Phase 3—Updating After Exchange of Two Rows. As in Phase 1, we must perform one of (*i*), (*ii*), or (*iii*) below for each column. The element x is the largest in M, y is the second largest, and z is the new element from row r. Row k is the row being ejected ($1 \leqq k \leqq m$).

Case (*i*): $z \leqq y$. If $k \neq i_x$ and $k \neq i_y$, then we are not replacing either of the two largest elements in this column, so no updating is necessary. Go to the next column.

If $k = i_x$, we are replacing the largest element with something no better than third largest. We replace x by y (and i_x by i_y) and find a new number two element in M_r—call it w. Since y is now largest, we subtract $w - y$ from $v(i_y)$ and then let $i_y = i_w$.

If $k = i_y$, we are replacing the number two element with something no better than third largest. Again we search for w, the new number two, update $v(i_x)$ by subtracting $w - y$, and update i_y.

Case (*ii*): $y < z \leqq x$. If $k = i_x$, we are replacing the largest element with a new and smaller largest element. The element z replaces x as the largest element, and $\Delta(r)$ is increased by $z - y$.

If $k \neq i_x$, z becomes the new number two element, and $z - y$ is subtracted from $v(i_x)$ to reflect the smaller difference between first and second elements.

Case (*iii*): $x < z$. If $k = i_x$, we are replacing the largest element with a new largest element. The value $\Delta(r)$ is augmented by $x - y$ (it already contains $z - x$ from Phase 1). The element z replaces x.

If $k \neq i_x$, we push x down into second place, since z is now the largest element, and subtract $x - y$ from $v(i_x)$, since x no longer contributes.

After this update has been done for each column, we copy $\Delta(r)$ into $v(k)$ and interchange rows r and k. (In the actual implementation, of course, row movement is just pointer manipulation.) Now go back to Phase 1.

This is the end of the algorithm description. The critical part of this algorithm is evaluating the contribution of a row without doing any of the updating necessary to exchange it, and particularly without scanning the matrix to find any column maxima. This latter operation need be performed only after we have decided upon an exchange; furthermore, it is performed only upon a small set of columns—those for which the element of the row being replaced was first or second largest and for which the replacement element is smaller than both [case (*i*) in Phase 3, above]. In practice, this condition holds for about 10 percent of the columns when we actually do a replacement. Since typically we replace relatively few rows in proportion to the number examined, the time saving is large.

III. AN EXAMPLE

This description may be clarified by one step of an example. Suppose $m = 3$, the cost matrix is

$$P = \begin{bmatrix} 1 & 3 & 3 & 6 & 1 & 6 \\ 2 & 2 & 8 & 7 & 2 & 0 \\ 1 & 5 & 3 & 4 & 4 & 1 \\ 4 & 2 & 4 & 5 & 8 & 2 \\ 7 & 9 & 1 & 1 & 3 & 5 \end{bmatrix},$$

and the first three rows are the current set. (This is obviously not a probability matrix—small integers are better for exposition.) Then the best entries are (2, 5, 8, 7, 4, 6), a value of 32.

Initialization. We find that v_1 is 5 (from column 6), v_2 is 7 (from columns 1, 3, and 4), and v_3 is 4 (from columns 2 and 5). Thus $v = (5, 7, 4)$. Suppose we want to evaluate row 4 as a replacement for one of these rows.

Phase 1. Considering column 1, $4 > 2$ so we are doing case (*iii*). The value Δ_4 is augmented by 2; at the same time, Δ_2 is decreased by 1, because the element in row 2 is no longer largest in column 1.

There is no change in column 2 [case (*ii*)]. In column 3, row 4 represents a new second-largest element, so Δ_2 is decreased by 1. There is no change in column 4. In column 5, add 4 to Δ_4, and subtract 2 from Δ_3. In column 6, decrease Δ_1 by 1. Thus, $\Delta = (-1, -2, -2, 6)$.

Phase 2. The minimum of $v_i + \Delta_i$ is 2, at $i = 3$, implying that row 3 should be ejected. We now commence the updating operation.

Phase 3. For column 1, the largest value is 4 (coincidentally in row 4) and the second largest is 2 (in row 2). Decrease v_2 by 1, since row 2 no longer contributes in this column [case (*iii*)]. In column 2, we are replacing the largest element by a very small one [case (*i*)]; the old number two element becomes the new largest, and we have to search for the new second largest. Add 1 to v_1, since the largest value is a 3 and the second largest a 2. In column 3, $p_{4,3}$ represents a new and bigger second element; decrease v_2 by 1. No change takes place in column 4. In column 5, we are replacing the largest element by a new largest element. The value of v_4 is increased by $2(p_{5,3} - p_{5,2})$. In column 6, we gain a new second element, so v_1 is decreased by 1.

En route, we compute the new value of the solution as 36.

When we have finished, $v = (5, 5, 8)$ for rows 1, 2, and 4, and the process continues by our considering row 5. Notice that out of six columns we only had to search for a second-largest element *once*; the rest of the time, it was immediately at hand.

IV. EXPERIMENTAL RESULTS

We have tried the procedure on several different types of data: matrix entries random on $\{0, \cdots, n\}$, random 0 − 1 matrices, and various probability matrices based on the physical problem.

For problems formed by generating random entries in the matrix, the fraction of random starts producing the optimum diminishes as m/ℓ increases (except for the trivial cases of very small or very large m), and also as n increases. Although there is significant individual variation, the frequency of obtaining the optimum is close to 100 percent for small problems and still about 20 percent for problems with $m = 10$, $\ell = 60$, $n = 80$, the largest random problems tried. (It should be noted that "optimum" usually means "best solution seen in a large number of trials"; we have strong statistical grounds for believing them optimal, but no proof.)

TABLE I—TYPICAL STATISTICS FOR SMALL RANDOM PROBLEMS (ENTRIES UNIFORM ON $[0, n]$)

m	ℓ	n	Time (ms)	No. of optimums/ Total Trials (range)	No. of Distinct Solutions (range)
5	10	20	21	20/20	1
5	40	40	120	32–39/50	6–9
5	40	80	230	6–47/50	4–19
10	20	20	42	20/20	1
10	40	40	140	25–41/50	3–5
10	40	80	300	11–16/50	5–10
20	60	60	340	18–41/50	2–5
20	60	80	450	12–42/50	7–13

Run time is directly proportional to ℓn; in absolute terms, the run time is about 100 ℓn microseconds per random start.

The procedure almost always makes less than two passes through the $\ell - m$ unused elements; the number of row replacements is roughly equal to m. As we mentioned above, it is only on these occasions that it is necessary to actually update the records of first- and second-best elements, and only for about 10 percent of the columns among the replacements is it necessary actually to scan through the m rows to locate a new second largest. (After initialization, it is never necessary to scan to find the largest.) Table I shows some typical results for these smaller tests.

Limited tests on 0–1 matrices produced similar results, although the run time appears to be slightly lower per case. It is possible in the 0–1 case to make several simplifications that would further decrease run time, and storage requirements could be drastically reduced by

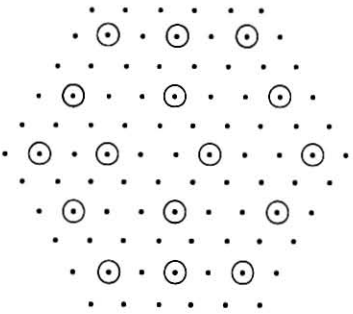

Fig. 1—Optimum solution.

using bit storage. However, we have not experimented extensively on 0–1 matrices. This will be reported in a separate paper.

Several experiments have been performed on various models of the real problem. Since the modems involved have very low error rates, the error transition probability p_{ij}, $i \neq j$, is small. This means that the matrix P has large diagonal elements, a few small elements, and a large number of zeros. For our purposes, the existence of large values is irrelevant; however, many zeros means that a storage organization that does not store zeros is attractive. We will discuss this shortly.

One problem to be faced is how to represent the continuous (a, b) space as a set of discrete points. One crude model we studied uses a "honeycomb" or hexagonal scheme (shown in Fig. 1), since this is a reasonable approximation to circular symmetry. The inner 61 points represent allowable signal locations; the outer 30 are the extra points to take care of the set of received signals that violate the peak power constraint.

For this test, P is defined as

$$\begin{aligned} p_{ij} &= \epsilon \text{ if } j \text{ is a neighbor of } i \ (\epsilon \ll 1) \\ &= 1 - (\epsilon \times \text{number of neighbors}) \text{ if } i = j \\ &= 0 \text{ otherwise.} \end{aligned}$$

This set of probabilities ignores phase jitter, which adds a radially increasing tangential component to the error transitions.

Figure 1 shows an optimum solution (it is easy to prove it optimum); Fig. 2 shows a local optimum differing by ϵ. In both cases, 12 signals are placed symmetrically on the boundary, and the remaining four are placed as well as possible in the interior. For this problem, all solutions were within 9ϵ of optimum and the median within 3ϵ (the mean random start is about 35ϵ away), run times averaged 370 ms per case, and

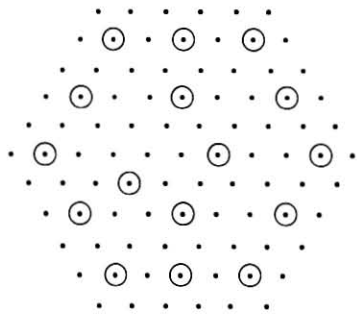

Fig. 2—Suboptimal by one unit.

the frequency of optimum solutions was about 10 percent. This problem appears to be slightly harder than random problems of this size, but run times are smaller.

V. LARGER PROBLEMS

To properly handle larger problems (e.g., to increase the resolution available when discretizing) a version of the program using a sparse matrix representation has been implemented. This involves substantial overhead in accessing elements of the matrix, but it is balanced by the fact that, when most matrix elements are zero, much less processing is required. For example, in the 16/61/91 (i.e., $m/\ell/n$) problem above, average run time increased from 370 to 390 ms, which is not significant. The run time is determined predominantly by the number of non-zero points, which is usually proportional to ℓn.

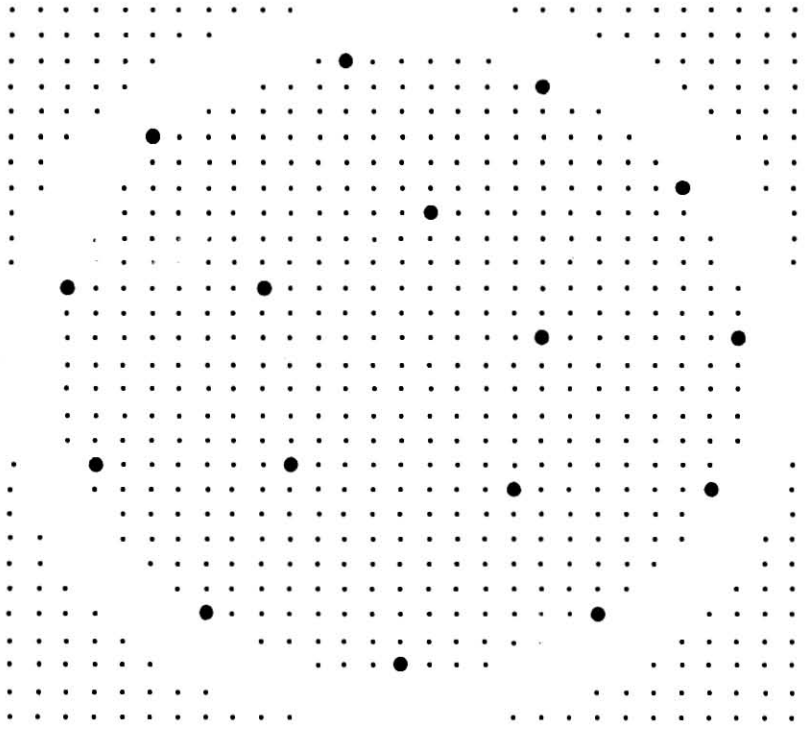

Fig. 3—"5–11" solution in pure Gaussian noise; $\beta = 0.0$; error rate $= 1.07 \times 10^{-7}$.

TABLE II—SOME RESULTS ON DESIGN PROBLEMS
($m = 16$, $N_0 = 0.002$)

	ℓ	n	Non-zero Entries	Run Time (s)	Phase Jitter β (degrees)	Character of Best Solution	Figure
1.	293	421	8,500	6.5	1.5	1–5–10	–
2.	421	577	17,000	14.5	1.5	1–6–9	–
3.	421	577	17,000	14.5	1.5	1–5–10	–
4.	489	665	22,000	19.3	0	5–11	3
5.	489	665	22,500	18.4	1.5	1–5–10	4
6.	489	665	27,000	20.5	3.0	1–6–9	5
7.	489	665	27,000	21.0	3.0	Special design	6
8.	577	749	31,000	24.9	1.5	1–5–10	–

VI. RESULTS ON REAL PROBLEMS

In this section we discuss some of the experimental results obtained from real problems, using formulas for the transition probabilities taken from Reference 2. The probability for the transition from $X = (x_1, x_2)$ to $Y = (y_1, y_2)$ has the general form

$$p(X, Y) = f(X, Y; \beta, N_o) \exp [g(X, Y; \beta, N_o)],$$

where N_o is the noise power of the channel and β is the rms phase jitter in the received signal. The details of f and g do not concern us here; it is sufficient to say that $p(X, Y)$ drops to zero rapidly as Y gets further from X. For example, in the pure Gaussian noise case ($\beta = 0$), we have

$$p(X, Y) = \frac{1}{2\pi N_o} \exp \left[- \frac{\|Y - X\|^2}{2N_o} \right].$$

Thus, the probability matrix is quite sparse when the transition probabilities have been scaled and converted to integers.

The discrete space consists of points on a square lattice, as shown in Fig. 3. The number of rows in P is determined by the number of lattice points within the circle of radius 1. To these are added exterior points approximating all possible additional received points. Since the radial probabilities drop off rapidly, this exterior layer need only be one or two units thick. This extra layer is indicated by the band of omitted points on Fig. 3.

The run time and the storage requirements both grow with the number of lattice points; this limits the resolution we can use. The largest problem tried had 665 rows, 861 columns, and about 40,000

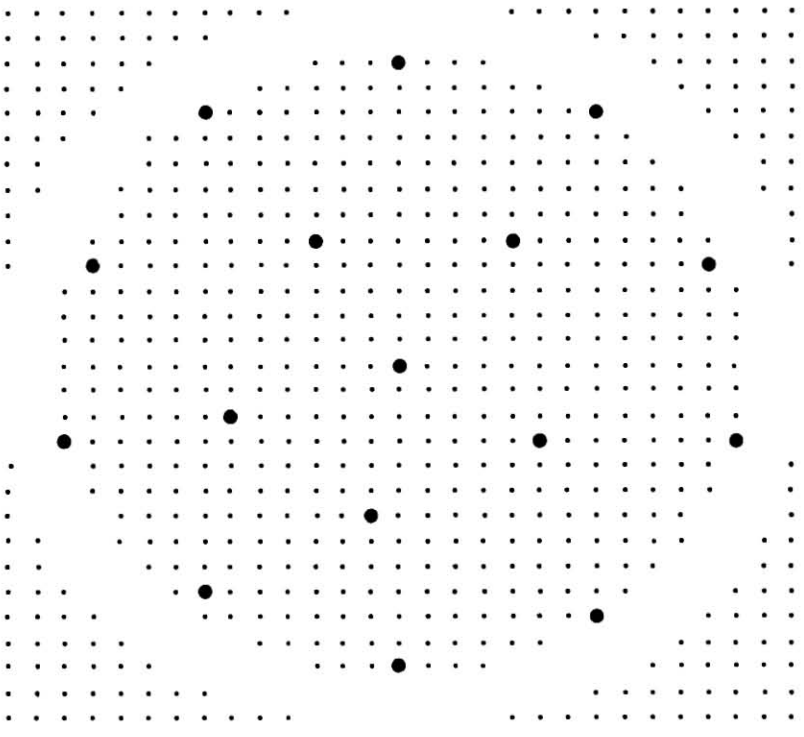

Fig. 4—"1–5–10" solution for 1.5-degree jitter (error rate = 2.97×10^{-7}).

non-zero matrix entries. It should be noted that the matrix has a fourfold symmetry, so, at the price of an increase in computing time (in practice, about 50 percent), only 10,000 entires need be stored.

Two types of experiments were performed. First, several constellations of intrinsic interest were used as initial solutions; the heuristic procedure attempted to improve upon them. Second, the procedure was used to produce good solutions from a large number of random initial configurations. For all solutions, approximate error rates were computed and the constellations displayed.

Table II lists some typical parameters for several experiments at various sizes; Figs. 3 to 5 show the best solutions found for particular parameter settings.

Each signal point is surrounded by a set of points which, when received, will decode into that signal point. This set of points is the "decision region" for that signal point. Because we have quantized a continuous space into small squares, the decision regions surrounding

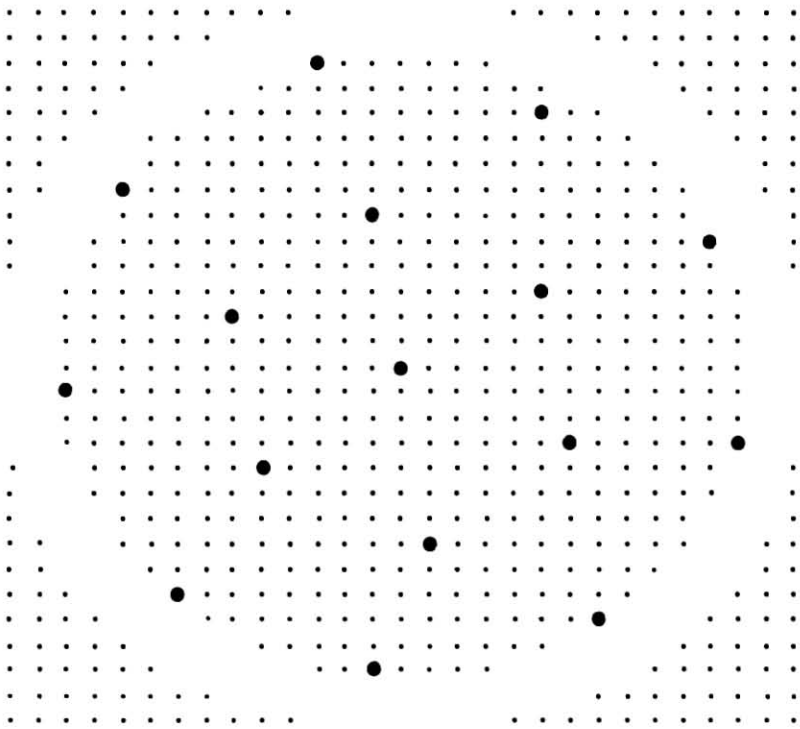

Fig. 5—"1–6–9" solution for 3-degree jitter (error rate = 3.26×10^{-6}).

each signal point have "ragged edges" and are of necessity somewhat arbitrary. As the resolution is made finer, this effect is less serious, and in fact the procedure can make more subtle choices of points and of boundaries, so the apparent error rate decreases with increasing resolution. For this reason, error rate comparisons between different resolutions are not appropriate. However, the rates are internally consistent in that, for any given resolution, the solution character and error rates vary with noise as would be expected.

As predicted by analytic techniques,[2] solutions like "1–5–10" and "1–6–9" are better for high jitter ($\beta > 1.5°$), while "5–11" solutions are better in low jitter cases. (The notation will be evident after examining the figures.) These trends are clearly indicated in Figs. 3 through 5, which show, respectively, the best solution (a 5–11) for zero phase-jitter (pure Gaussian noise) with an error rate of 1×10^{-7}, the best solution (1–5–10) for 1.5 degrees of phase jitter (error rate

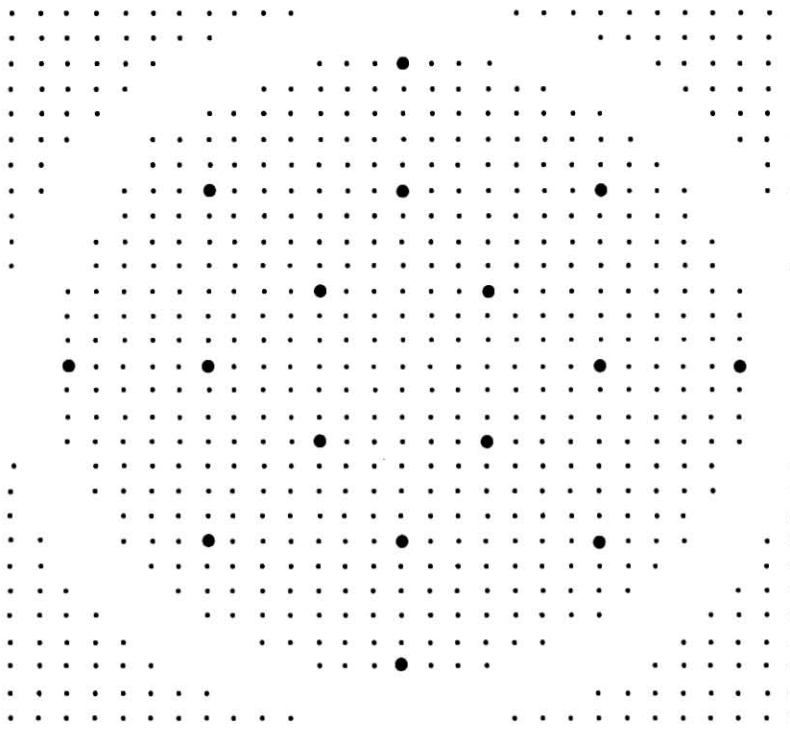

Fig. 6—Competing design, for 3-degree jitter (error rate = 3.47×10^{-5}).

3×10^{-7}), and the best solution (1–6–9) for 3 degrees of jitter (error rate 3×10^{-6}).

For comparison, we experimented with the competing design shown in Fig. 6, at various levels of jitter. This design is intended to be robust over a wide range of jitter. This configuration does degrade less than the others as jitter increases, but its overall performance is very much inferior. Figure 6 shows that, at 3 degrees, its error rate is 3.5×10^{-5}, a factor of 10 worse than the 1–6–9 configuration. These results agree closely with predictions of independent theoretical studies.[2]

VII. CONCLUSIONS

As a solution to a combinatorial optimization problem, the heuristic procedure presented here is quite good for small-to-medium problems, say up to about 100 rows, even for dense matrices. It remains useful, but not strong, for large sparse problems.

As for the original design problem, the number of rows and columns in the matrix both rise with the resolution; thus, highly accurate representations are computationally expensive, so the procedure is generally not appropriate for generating precise answers to specific design questions. Rather, it is most useful in providing quick approximate and comparative optimizations or evaluations, either to furnish insight or to supplement results obtained by analytic techniques.

The extension of this technique to problems with an average power constraint, rather than peak power, appears to be straightforward, although we have not implemented it. [The average power constraint requires that $\sum(a^2 + b^2) \leqq 1$.]

As the simplest solution, start with a random feasible set of m rows. Then, before each possible replace row is selected, test to see if it would violate feasibility; if so, it cannot be used. A more powerful algorithm would permit temporary violations of feasibility in a controlled way. Either of these approaches should serve reasonably well.

VIII. ACKNOWLEDGMENTS

The authors are indebted to G. Foschini for originally suggesting the problem and for providing data and physical insight. We are also indebted to R. Gitlin for several valuable lectures on communication theory.

REFERENCES

1. Lin, S., "Computer Solutions of the Traveling Salesman Problem," B.S.T.J., *44*, No. 10 (December 1965), pp. 2245–2269.
2. Foschini, G., Gitlin, R., and Weinstein, S., "On the Selection of a Two-Dimensional Signal Constellation in the Presence of Phase Jitter and Gaussian Noise," B.S.T.J., *52*, No. 6 (July-August 1973), pp. 927–965.