ELSEVIER

Pattern Recognition Letters 21 (2000) 61–68

Pattern Recognition Letters

www.elsevier.nl/locate/patrec

Genetic algorithm with deterministic crossover for vector quantization

Pasi Fränti [*]

Department of Computer Science, University of Joensuu, P.O. Box 111, FIN-80101 Joensuu, Finland

Received 8 February 1999; received in revised form 20 August 1999

Abstract

Genetic algorithm (GA) provides high quality codebooks for vector quantization (VQ) at the cost of high running time. The crossover method is the most important choice of the algorithm. We introduce a new deterministic crossover method based on the pairwise nearest neighbor method. We show that high quality codebooks can be obtained within a few minutes instead of several hours as required by the previous GA-based methods. The method outperforms all comparative codebook generation methods in quality for the tested training sets. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Vector quantization; Codebook generation; Clustering; Genetic algorithms; Combinatorial optimization; Image compression

1. Introduction

We study the problem of generating a *codebook* for a *vector quantizer* (VQ). The aim is to find M *code vectors* (*codebook*) for a given set of N *training vectors* (*training set*) by minimizing the average pairwise distance between the training vectors and their representative code vectors (Gersho and Gray, 1992). The most cited and widely used method is the *generalized Lloyd algorithm* (GLA), as proposed by Linde et al. (1980). It starts with an initial codebook, which is iteratively improved until a local minimum is reached. The result of the GLA is highly dependent on the choice of the initial codebook.

Better results can be achieved using an optimization technique known as *genetic algorithm* (GA). As shown by Fränti et al. (1997), the best GA variant outperforms all comparative methods (including the GLA), but at the cost of high running time. The problematic part of the algorithm is the crossover. The existing crossover methods are heuristic and therefore incapable of generating competitive codebooks as such. A few iterations of the GLA are therefore always needed for fine-tuning the solution. This makes the GA significantly slower than the comparative methods because there are many candidate solutions to be generated, and each of them must be fine-tuned by the GLA.

We introduce a new deterministic crossover method for the GA. The main part of the

[*] Corresponding author. Tel.: +358-13-2513103; fax: +358-13-2513290.
E-mail address: franti@cs.joensuu.fi (P. Fränti).

crossover algorithm is the *pairwise nearest neighbor method* (PNN). The method has three vital improvements over the previously reported implementation (Fränti et al., 1997): (i) The representation of solution is revised so that we do not merge only the codebooks, but maintain both partition and codebook for each solution. In this way, the partition of a new solution can be efficiently computed from those of the parent solutions. Access to the partition gives also a more precise initialization for the PNN, which results in higher quality candidate solutions. (ii) Empty partitions are removed before the application of the PNN. This is vital for avoiding the slowness of the PNN. (iii) As the new candidate solutions are already close to a local minimum, the GLA iterations can be performed extremely fast using the grouping technique recently introduced by Kaukoranta et al. (1999).

For the tested training sets, the proposed method outperforms all comparative methods, including the previous variants of the genetic algorithm. The use of a deterministic crossover achieves also a fast convergence with rather small population size. The algorithm is therefore remarkably faster than any of the previously reported genetic algorithms.

2. Codebook generation

We consider a set $X = \{x_1, x_2, \ldots, x_N\}$ of N training vectors in a K-dimensional Euclidean space. The aim is to find a codebook $C = \{c_1, c_2, \ldots, c_M\}$ of M code vectors by minimizing the average distance between the training vectors and their representative code vectors. The distance between two vectors is defined by their squared Euclidean distance. The distortion of the codebook is then calculated as

$$f(P, C) = \frac{1}{N}\sum_{i=1}^{N} \|x_i - c_{p_i}\|^2. \tag{1}$$

Partition $P = \{p_1, p_2, \ldots, p_N\}$ defines for each training vector x_i the index p_i of the code vector where it is mapped to. A solution for the codebook generation problem can therefore be defined by the pair (P, C), see Fig. 1. These two depend on each

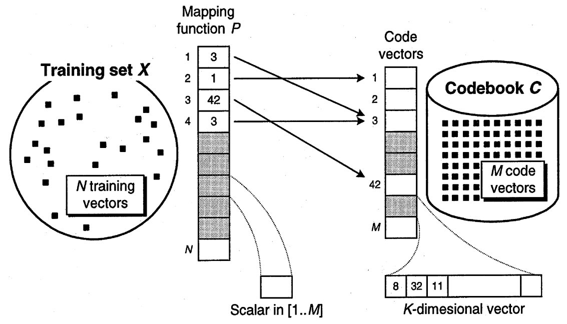


Fig. 1. Illustration of the data structures.

other in such a manner that if one of them is given, the optimal choice for the other one can be constructed using the following optimality conditions.

Partition optimality: Given a codebook C, the optimal partition P minimizing (1) is obtained by assigning each training vector x_i to its nearest code vector c_j

$$p_i = \arg\min_{1 \leqslant j \leqslant M} \|x_i - c_j\|^2. \tag{2}$$

Codebook optimality: Given a partition P, the optimal codebook C minimizing (1) is obtained by calculating the code vectors c_j as the centroids of the clusters

$$c_j = \frac{\sum_{p_i=j} x_i}{\sum_{p_i=j} 1}, \quad 1 \leqslant j \leqslant M. \tag{3}$$

In codebook generation, we usually concentrate on finding a codebook C and the mapping function P is assumed to be optimally defined according to Eq. (2). Next we recall three known methods for generating codebook.

GLA (Linde et al., 1980) starts with an initial codebook, which is iteratively improved using the two optimality conditions in turn. In the first step, each training vector x_i is mapped to its nearest code vector c_j in the previous codebook. In the second step, the code vectors are recalculated as the centroids of the new partitions. The new solution is always better than or equal to the previous one. The algorithm is iterated as long as improvement is achieved. The algorithm, however, makes only local changes to the previous solution and is therefore highly sensitive to the initialization.

The method by Zeger and Gersho (1989) applies stochastic relaxation technique with the GLA by adding noise to the code vectors after each iteration. The amount of noise gradually decreases with the iteration and eventually, when the noise has been completely eliminated, the algorithm converges back to the GLA. The method performs progressive refinement of the solution and is therefore capable of finding better global settlement of the code vectors than the GLA. We refer this method as *stochastic relaxation* (SR).

PNN by Equitz (1989) uses a different approach by generating the codebook hierarchically. It starts by initializing each training vector as a separate code vector. Two code vectors are merged at each step of the algorithm and the process is repeated until the desired size of the codebook is obtained. The code vectors to be merged are always the ones whose merge increase the distortion least. The increase is calculated as

$$d_{a,b} = \frac{n_a n_b}{n_a + n_b} \cdot \|c_a - c_b\|^2, \tag{4}$$

where c_a and c_b are the merged code vectors, n_a and n_b are the size of the corresponding clusters. The original implementation of the PNN takes $O(N^3)$ time but a significantly faster $O(\tau N^2)$ time algorithm was recently introduced by Fränti and Kaukoranta (1998). The idea is to maintain for each code vector a pointer to its nearest neighbor and in this way, avoid unnecessary distance calculations. After the merge operation, the pointers must be updated only for clusters whose nearest neighbor is one of the merged vectors. On average, the number of updates (τ) is significantly smaller than N.

3. Genetic algorithm

GA is based on the model of the natural selection in real life. The main idea is to maintain a set of solutions (*population*), which is iteratively regenerated using *genetic operations* (*crossover* and *mutations*) and *selection*. The general structure of our GA method is shown in Fig. 2. Each initial solution is created by selecting M random training vectors as the code vectors and by calculating the optimal partition according to Eq. (2). The solutions for the next population are then created by crossing the best solutions of the current population.

The number of iterations (T) and the population size (S) are the main parameters of the algorithm. In general, a large population should be used because this would guarantee enough genetic variation in the population. The number of iterations should be as high as time can be afforded. Even if the solution does not improve during a single iteration it is possible that improvement will appear later. Mutations can also be applied for increasing genetic variation in the population, and it can be useful if the algorithm is iterated long time. The GLA is used as a local optimizer for fine-tuning the new solution towards a local minimum. The necessity of the GLA depends on the choice of the crossover and mutation algorithms.

3.1. Representation of solution

The representation of a solution is an important choice in the algorithm because it determines the data structures which are to be modified in the crossover and mutation operations. Wolpert and Macready (1997) have pointed out the importance of incorporating problem-specific knowledge into

```
Genetic algorithm:
    Generate S initial solutions.
    REPEAT T times
        Generate new solutions.
        Sort the solutions.
        Store the best solution.
    END-REPEAT
    Output the best solution found.
```

```
Generate new solutions:
    REPEAT S times
        Select pair for crossover.
        Cross the selected solutions.
        Mutate the new solution.
        Fine-tune the new solution by GLA.
    END-REPEAT
```

Fig. 2. Main structure of the genetic algorithm.

the behavior of the algorithm. A problem-specific representation is therefore used.

For evaluating a solution we need both partition (P) and the codebook (C). The optimality conditions (2) and (3), on the other hand, indicate that only one of them is sufficient because the other one can always be generated. This implies three alternative choices for representing a solution:

- Partition: (P)
- Codebook: (C)
- Combined: (P, C)

The first approach operates with P and generates C using (3). The problem is that only local changes can be generated to the solution by modifying the partition. The second approach operates with C and generates partition using (2). The advantage of this approach is that the entire clustering structure can be revised through modifications of the code vectors. A drawback is that the generation of partition is computationally expensive, requiring $N \cdot M$ distance calculations.

We take the third approach and maintain both P and C. The key point is that both data structures are needed for evaluating the solution, and it would be computationally inefficient to recalculate either data structure from scratch in every step of the algorithm. Instead, the data structures of the existing solutions can be fully utilized.

3.2. Selection method

The main goal of the selection is that better solutions are chosen more often in the crossover than worse solutions, the exact implementation of the selection is not so vital. We use an elitist approach, in which only the best solutions are considered and the rest are discarded. The solutions are sorted in an increasing order given by their distortion values. We permute all possible pairs in a greedy manner until the population is completed.

3.3. Crossover

Several crossover methods were summarized by Fränti et al. (1997). These crossover methods have two weaknesses. Firstly, the methods are heuristic and can rarely generate competitive solutions without the application of a few GLA iterations. Secondly, they cross only the codebooks and ignore the partitions of the parent solutions. The partition of the new solution must therefore be recalculated from scratch, which requires $N \cdot M$ distance calculations.

We take an alternative approach and perform the crossover in deterministic manner. We combine the existing solutions (C^1, P^1) and (C^2, P^2) so that the new solution (C^{new}, P^{new}) is competitive already before the use of the GLA. In addition to that, unnecessary computation is not wasted for a complete repartition but the partitions of the parent solutions are utilized. The sketch of the new crossover algorithm is shown in Fig. 3.

The crossover starts by merging the parent codebooks by taking their union (*CombineCentroids*). The partition P^{new} is then constructed on the basis of the existing partitions P^1 and P^2 (*CombinePartitions*). The partition of training vector x_i is either ${p_i}^1$ or ${p_i}^2$. The one with smaller distance to x_i is chosen. In this way, P^{new} can be generated using $2 \cdot N$ distance calculations only. The codebook C^{new} is then updated (*UpdateCentroids*) using (3) in respect to the new partition P^{new}. This procedure gives a solution in which the codebook has twice the size as it is allowed to. The final task is to reduce the codebook size from $2 \cdot M$ to M.

Empty clusters are first removed (*RemoveEmptyClusters*) because they would cause computational inefficiency in the PNN. It is possible (even likely) that the same code vector is obtained from both parents. In this case, all training vectors in their clusters are mapped to the same vector and leaving the other cluster empty. In the worst case, there are M empty clusters and this would lead to $O(M^3)$ time for the PNN.

The final size of the codebook is then obtained using the PNN algorithm (*PerformPNN*) as given by Fränti and Kaukoranta (1998) with the following two differences. Firstly, we do not perform the PNN for full training set but start from an initial solution of at most $2 \cdot M$ vectors. The crossover can therefore be performed in $O(\tau M^2)$ time instead of the original $O(\tau N^2)$ time. Secondly, the partition data is also updated during the crossover and therefore not needed to be recalculated after the PNN.

```
CrossSolutions($C^1$, $P^1$, $C^2$, $P^2$) → ($C^{new}$, $P^{new}$)
    $C^{new}$ ← CombineCentroids($C^1$, $C^2$)
    $P^{new}$ ← CombinePartitions($P^1$, $P^2$)
    $C^{new}$ ← UpdateCentroids($P^{new}$)
    RemoveEmptyClusters($C^{new}$, $P^{new}$)
    PerformPNN($C^{new}$, $P^{new}$)

CombineCentroids($C^1$, $C^2$) → $C^{new}$
    $C^{new}$ ← $C^1 \cup C^2$

CombinePartitions($C^{new}$, $P^1$, $P^2$) → $P^{new}$
    FOR $i$←1 TO $N$ DO
        IF $\|x_i - c_{p_i^1}\|^2 \le \|x_i - c_{p_i^2}\|^2$ THEN
            $p_i^{new} \leftarrow p_i^1$
        ELSE
            $p_i^{new} \leftarrow p_i^2$
    END-FOR

UpdateCentroids($C^1$, $C^2$) → $C^{new}$
    FOR $j$←1 TO $|C^{new}|$ DO
        $c_j^{new}$ ← CalculateCentroid($P^{new}$, $j$)

PerformPNN($C^{new}$, $P^{new}$)
    FOR $i$←1 TO $|C^{new}|$ DO
        $q_i$ ← FindNearestNeighbor($c_i$)
    WHILE $|C^{new}|>M$ DO
        $a$ ← FindMinimumDistance($Q$)
        $b$ ← $q_a$
        MergeClusters($c_a$, $p_a$, $c_b$, $p_b$)
        UpdatePointers($Q$)
    END-WHILE
```

Fig. 3. Pseudocode for the PNN crossover. (More detailed pseudocode is available in Electronic Annexes of PATREC.)

At the first step of the PNN, we search for each code vector its nearest neighbor (*FindNearestNeighbor*) that minimizes the merge cost according to (4). The nearest neighbor pointers are stored in $Q = \{q_1, q_2, \ldots, q_M\}$. The vectors to be merged can be determined by finding q_i with minimal merge cost (*FindMinimimumDistance*). After the merge (*MergeClusters*), the pointers are updated (*UpdatePointers*), and the process is repeated until the size of the codebook is reduced to M.

3.4. Mutations

Mutations are generated by replacing a randomly chosen code vector by a randomly chosen training vector. This method is denoted as *random swap*, and it is the neighborhood method used in the local search algorithm by Fränti et al. (1998). The use of mutations is not necessary in our algorithm. It slows down the search whereas we aim at fast convergence. In the long run, however, mutations can be used for increasing genetic variation in the population. The purpose is to discover new search paths when the population becomes too homogenous for the crossover to achieve significant improvement anymore. Effectively, the mutations simulate local search by making small modifications to the current solution. If the inclusion of the mutations is vital, it implies that the crossover is not well-defined and the algorithm actually implements a parallel local search algorithm.

3.5. Local optimization by GLA

The result of the crossover can practically always be fine-tuned by the GLA. It can be iterated until the nearest local minimum is reached although a few iterations are usually sufficient for making the new solution competitive. The inclusion of the GLA was shown to be vital in the previous GA implementations (Fränti et al., 1997) because heuristic crossover can rarely produce solutions that are competitive with the parent solutions. The heuristic methods therefore relies on the use of the GLA, and the effect of the crossover is mainly to create different starting points for the GLA.

The PNN crossover, on the other hand, can produce competitive solutions as such. The use of the GLA is therefore not necessary although still recommended because of its extra benefit in fine-tuning of the solution. The inclusion of the GLA can be rather time-consuming because there are several candidate solutions to be processed. Most of the computation in the GLA originates from the calculation of $N \cdot M$ vector distances. Fortunately, a large proportion of these distance calculations can be avoided using the grouping technique introduced by Kaukoranta et al. (1999). The grouping technique is very effective when applied for solutions that are already of good quality, which is the case after the PNN crossover.

4. Test results

We generated three training sets: *Bridge*, *Miss America*, and *House*, see Fig. 4. The vectors in the first set (*Bridge*) are 4×4 pixel blocks from the image. The second set (*Miss America*) has been obtained by subtracting two subsequent image frames of the original video image sequence, and then constructing 4×4 spatial pixel blocks from the residuals. Only the first two frames have been used. The third data set (*House*) consists of color values of the *RGB* image. Applications of this kind of data sets are found in image and video image coding (*Bridge*, *Miss America*), and in color image quantization (*House*). The size of the codebook is fixed to $M = 256$ throughout the experiments.

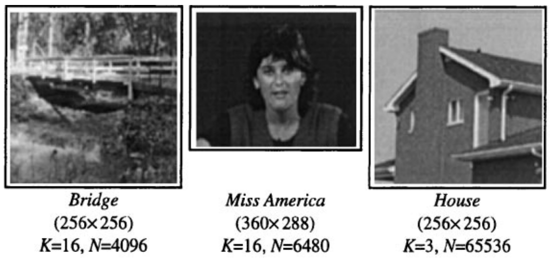
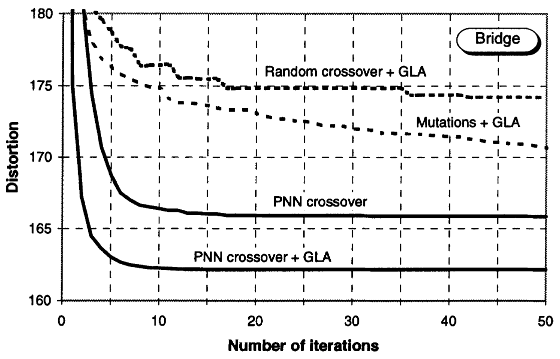


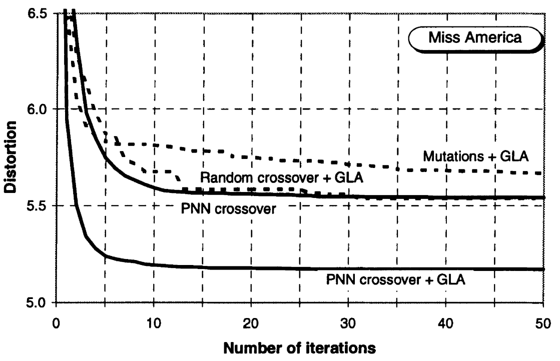
Fig. 4. Sources for the training sets.

Table 1
Running times (min:s) of the GA with the PNN crossover

	Bridge	*Miss America*	*House*
Previous GA	332	536	805
Proposed GA	13	7	6
SR	8	21	41
GLA	0.12	0.25	0.50

We study first the population size (S) and the number of iterations (T). Experiments show that improvement can appear during a long time but most remarkable improvement is obtained during the first few iterations only. The later modifica-





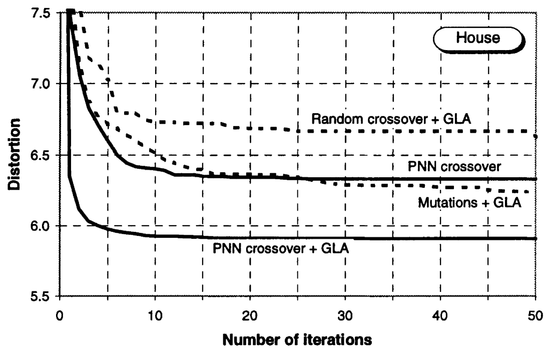


Fig. 5. Distortion performance of the GA as a function of time. The parameters were set up as ($S = 16$, $T = 50$) for *Bridge*, ($S = 5$, $T = 50$) for *Miss America*, and ($S = 6$, $T = 50$) for *House*.

tions are more or less fine-tuning of the solution and further improvement remains marginal. It is therefore reasonable to stop when the algorithm first time fails to improve. Using this stopping criterion, we performed the GA with all population sizes from $S = 2$ to 32. Two GLA iterations were applied for all solutions but no mutations were performed.

The results were compared to the best previous crossover method (Fränti et al., 1997), in which the parameters were setup as $S = 45$ and $T = 50$. We found out that the smallest population size (on average) needed to bypass the previous results was $S = 16$ (*Bridge*), $S = 5$ (*Miss America*), and $S = 6$ (*House*). The corresponding number of iterations were $T = 15$ (*Bridge*), $T = 17$ (*Miss America*), and $T = 12$ (*House*). Equally good codebooks can therefore be obtained with significantly less computing efforts.

The running times of the proposed method are summarized in Table 1. The slowness of the previous implementation originates from three facts: (i) a large number of candidate codebooks are generated, (ii) all candidates are iterated by the GLA and (iii) the crossover is very slow. The new method gives significant improvement in all these cases. Firstly, the improved quality of the crossover is utilized by reducing the number of candidates by a factor of 10–40. Secondly, the GLA iterations can be performed extremely fast for the codebooks resulting from the PNN crossover. Finally, the removal of the empty clusters avoids the slowness of the PNN implementation. Overall, the proposed method reduces the running time by a factor of 25–50 in comparison to the previous implementation. The GA is still slower than the GLA but the difference is much smaller.

The convergence of the new method is illustrated in Fig. 5. Most of the improvement appear during the first few iterations. The use of mutations are therefore not needed but they can be useful in the long run. Comparative results are shown for random crossover, and a method without any crossover where the best solution is replicated and mutated. The deterministic crossover is superior both in time and quality, and it always converges very fast and more steadily than the other methods.

The distortion performance of the proposed GA method is compared with the other methods in Table 2. The "fast" refers to the discussed parameter combination, for which the quality of the codebook was equal to that of the previous GA implementation. The "best" refers to another parameter combination, in which we aim at the highest possible quality by setting the parameters as $S = 100$, $T = 500$. Mutations and ten GLA-iterations are also applied for every solution. The extra benefit remains marginal and the "fast" variant is therefore recommended.

Additional results are shown in Table 2 in the case when the test set is outside of the training set. For *Bridge*, we use 25% randomly chosen vectors as training data, and the rest 75% of the vectors are used as test data. For *Miss America*, we use different frames for training and for testing. For *House*, we apply a prequantized image (5 bits per color component) for training, and the original image (8 bits) is used for testing. The main observation is that the proposed GA performs well also outside the training set although the differences are smaller. This indicates the importance of the proper choice of the training set.

Table 2
Performance comparison of the various algorithms

	Inside training set			Outside training set		
	Bridge	*Miss America*	*House*	*Bridge*	*Miss America*	*House*
Random	251.32	8.34	12.12	277.99	9.99	11.23
GLA	179.68	5.96	7.81	251.71	8.60	9.52
PNN	169.15	5.52	6.36	250.07	8.60	8.91
SR	162.45	5.26	6.03	250.85	8.73	9.13
GA-fast	162.01	5.17	5.92	248.65	**8.52**	8.92
GA-best	**160.92**	**5.09**	**5.85**	**248.30**	**8.52**	**8.89**

5. Conclusions

A deterministic crossover method was introduced for the GA in VQ. The use of a deterministic crossover has the advantage that good results are achieved much faster. The proposed GA-based method can therefore produce high quality codebooks within a few minutes instead of several hours as required by the previous implementation. The method outperforms the comparative codebook generation methods for the tested training sets, although the difference to SR is rather small.

Acknowledgements

The work of Pasi Fränti was supported by a grant from the Academy of Finland.

References

Equitz, W.H., 1989. A new vector quantization clustering algorithm. IEEE Transactions on Acoustics, Speech and Signal Processing 37, 1568–1575.

Fränti, P., Kaukoranta, T., 1998. Fast implementation of the optimal PNN method. In: IEEE Proceedings of the International Conference on Image Processing (ICIP'98), Chicago, Illinois, USA (revised version will appear in IEEE Transactions on Image Processing).

Fränti, P., Kivijrävi, J., Kaukoranta, T., Nevalainen, O., 1997. Genetic algorithms for large scale clustering problems. The Computer Journal 40, 547–554.

Fränti, P., Kivijrävi, J., Nevalainen, O., 1998. Tabu search algorithm for codebook generation in VQ. Pattern Recognition 31, 1139–1148.

Gersho, A., Gray, R.M., 1992. Vector Quantization and Signal Compression. Kluwer Academic Publishers, Dordrecht.

Kaukoranta, T., Fränti, P., Nevalainen, O., 1999. Reduced comparison search for the exact GLA. In: Proceedings of the IEEE Data Compression Conference (DCC'99), Snowbird, Utah, USA, pp. 33–41.

Linde, Y., Buzo, A., Gray, R.M., 1980. An algorithm for vector quantizer design. IEEE Transactions on Communications 28, 84–95.

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computing 1, 67–82.

Zeger, K., Gersho, A., 1989. Stochastic relaxation algorithm for improved vector quantiser design. Electronics Letters 25, 896–898.