An Investigation of Iterative Routing Algorithms

Dr. Zahir Moosa Dr. Douglas Edwards

Department of Computer Science, University of Manchester
Oxford Road, Manchester, UK, M13 9PL

Abstract

This paper presents a detailed analysis of an iterative routing algorithm in which multiple passes are made through a net list, varying the region of crossover on each net after every iteration. This is in contrast to conventional iterative routing algorithms which incorporate a crossover penalty within the cost function being minimised. Experimental results are presented and, based on these results, new modifications proposed. The modified algorithm is compared to a conventional iterative algorithm and a single pass Lee router. The results illustrate the improved completion rates of iterative algorithms with respect to single pass algorithms. Finally, an element of dynamic routing (ripup and reroute) is introduced to further improve the completion rate of the modified algorithm.

1 Introduction

The inadequacies of Lee's routing algorithm [1] are well known. The completion rate has been shown to be particularly sensitive to the order in which nets are routed. In the case of cost based Lee routers, there is a further sensitivity to the particular sets of cost penalties [2] being enforced. Past research has shown that there is no optimal net ordering algorithm or combination of cost penalties that necessarily guarantees an optimal completion rate for any given routing problem [3].

The algorithm remains popular, however, due to its ability to find a path if one exists. Recent research has concentrated on developing techniques that combine the advantages of Lee's algorithm with a degree of intelligence so that there is a reduced dependence on net orders and cost penalty selections. There are two main techniques that have been used: iterative routing and dynamic routing. Iterative routers [4, 5, 6, 2, 7, 8, 9] are characterised by multiple passes through the net list. Each net is routed so as to minimise a cost function that includes a crossover penalty for *crossing* other nets. The crossover penalty is increased after each iteration till the final pass where typically no crossovers are permitted. Dynamic routers [10, 11, 2, 12, 13, 14, 15] (ripup and reroute) on the other hand reroute a set of obstacle nets in order to free routing resources for unrouted nets. A feature of this class of routers is its recursive nature caused by rerouted nets inducing as further level of obstacle nets to be rerouted. This typically goes on until either all nets have been successfully routed or the algorithm gives up.

This paper describes an iterative routing strategy based on an idea mentioned in [9] and credited to Vintr. The cost function minimised does not include a crossover penalty, but instead varies the region of crossover on a net. No published results of Vintr's routing algorithm are available (to the author's knowledge). The work presented here will seek to identify the advantages and disadvantages of Vintr's algorithm. As a result, modifications, are suggested that may be used to improve the basic algorithm. The modified Vintr's algorithm is then compared to other iterative approaches already established in literature. Based on the latter comparison, a hybrid iterative routing algorithm is proposed that incorporates the philosophies behind Vintr's algorithm, the crossover penalty type algorithms and dynamic routing algorithms.

The remainder of this paper is organised as follows. Section 2 discusses previous work done in the area of iterative routing algorithms. Section 3 describes the implementation of Vintr's algorithm. Section 4 presents an experimental evaluation of Vintr's algorithm along with its associated modifications. The performance of the algorithm is also compared with other routing algorithms. Section 5 completes the paper by offering some conclusions and a summary of future work.

2 Previous Work

Published work on iterative routing algorithms tend to differ in the nature of the crossover penalties used and the context in which they are applied. Rubin [8] uses a path cost function of the form : $(PathCost) + K_i(NumberOfCrossings)$ where K_i is the crossover penalty for the i'th pass and $K_{i+1} = nK_i$ where $K_{initial} = 3$ and n lies between 2 and 5. Moore [6] describes a similar algorithm to Rubin but generalises the crossover penalty update function so that now $K_{i+1} = f(K_i, i)$ where i is the current iteration number. Moore also suggests a time saving mechanism referred to as a partial pass. During each iteration, only those nets that are involved in intersections are ripped up and rerouted. Linsker[4, 5] introduces the concept of path history dependent penalty functions. A crossover penalty is assigned on a net by net basis instead of a constant value per pass. The idea is to penalise crossovers over "good" nets higher than those nets that are deemed "bad". Linsker goes on to extend the algorithm to cope with other types of constraints such as crosstalk and net length restrictions. Rosenburg [7] makes the observation that other iterative approaches are not truly independent of the net ordering function, since within each pass the crossover penalty contributions to some nets may already depend on nets scheduled earlier within the same pass. Rosenburg integrates the crossover penalty with a cell weighting function. Updates to all cell costs are left till the end of the iteration. In this case the crossover penalties incurred by a net are only dependent on the paths of nets determined in the previous iteration.

Vintr's algorithm differs significantly from other iterative approaches in that the region of crossover on a connection is varied from iteration to iteration, rather than the penalty for crossing the connection. During each iteration, a net path is determined using a standard path finding algorithm, such as Lee's algorithm. Only a part of the net is then actually implemented on the routing plane so that the connection will now appear as two stubs emanating from the terminals at the endpoints of the net. The proportion of the net implemented increases from iteration to iteration till the final iteration where the entire net is implemented. Vintr's routing algorithm attempts to address the terminal isolation problem[10] by having protruding tails from each terminal that attempt to prevent blockages in the vicinity of terminals. This algorithm will be described in greater detail in the next section.

3 Vintr's Algorithm

Paths for nets are determined using any standard path finding algorithm (e.g. Lee's algorithm) but instead of implementing the entire net, only part of the net is actually implemented from each end. Another way of looking at this would be to regard all nets as being comprised of 3 distinct parts, two fixed non-crossable tail nets emanating from both the endpoints of the net, and a non-fixed crossable middle portion. The proportion of a net that is fixed is dependent on the number of passes completed, and will be regarded as the cutoff value for the relevant pass. Each time a net is rerouted, the existing tail nets are first ripped up from the routing surface. A new path is then determined which may or may not differ from the previously determined path due to the presence of tails belonging to other nets. The cutoff value is iteratively increased until the final

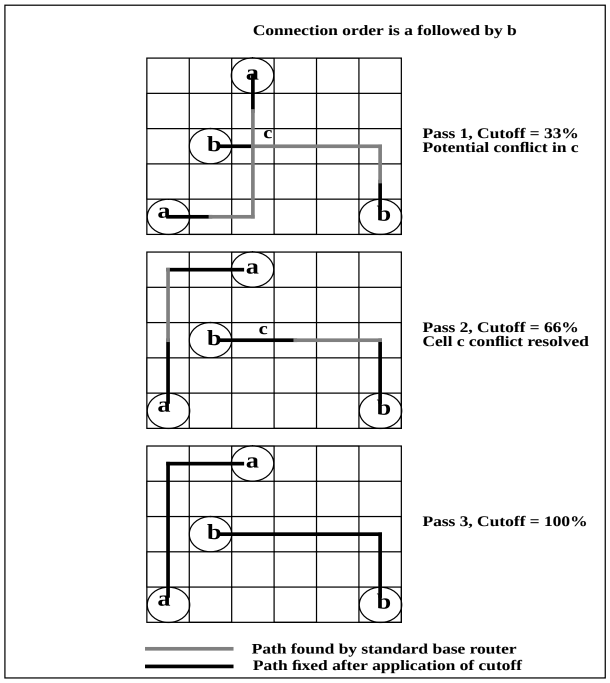

Figure 1: **llustration of Vintr's algorithm**

iteration in which the entire connection is fixed and non-crossable (i.e. a cutoff value of 100%). An example illustrating an application of Vintr's algorithm is shown in Figure 1.

Vintr's algorithm attempts to minimise the extent of terminal isolation. The terminal isolation problem has been identified by Dees [10] as a major cause of routing failures. Terminal isolation occurs when a terminal (pin) becomes blocked from the rest of its net as a result of tracking features belonging to other nets, in the vicinity of the terminal. To a small extent the problem may be solved by complex cell weighting strategies and the iterative routing techniques described in the previous section. However, this is not usually enough to prevent the problem. The main advantage of Vintr's algorithm and its derivatives is that the fixed tails protruding from each terminal has the effect of reducing the probability of terminal isolation whilst still allowing interaction of nets.

The rest of this paper will refer to the set of cutoff values for a particular run of the algorithm as the cutoff set for the corresponding invocation of the router. Thus 2 iterations with cutoff values of 50 and 100 will be denoted as:

$$identifier = [50, 100]$$

Any references to $identifier$ will indicate the corresponding cutoff parameter set.

4 Results

Vintr's algorithm has been implemented in the C programming language on a SUN 4 Sparc workstation. A diverse range of commercially fabricated PCB designs have been used to test the algorithm. All CPU times are quoted in seconds.

4.1 Vintr's Algorithm - VinBasic and VinPenalty

Vintr's algorithm, as outlined by Soukup [9], only considers the implementation of nets from the endpoints (terminals). The middle portions of all nets are discarded and disregarded during routing. Nets, as a result, are now free to cross the potential paths of other connections unpenalised. This form of Vintr's will be referred to as VinBasic.

A second version of Vintr's algorithm, VinPenalty, increases the degree of lookahead, by penalising any expansions across potential paths of other nets The crossover penalty for using cells on the non-fixed parts of other nets is set very high in order to discourage crossovers as much as possible. The difference obtained by the two routers is quite significant and this is reflected in the overall results shown in Table1 which shows a comparison between VinPenalty and VinBasic for a number of boards with the following cutoff set:

$$P2 = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$$

Note that $P2$ is in no way meant to indicate the optimal parameter set. The issue of optimal parameter sets will be discussed later in this section. For now $P2$ will be used for comparison purposes. The results in Table 1 show quite clearly the better completion rate obtained by VinPenalty compared to that obtained by VinBasic (in a comparable execution time).

Table 1: **Comparison of** VinBasic **and** VinPenalty

Router		VinBasic		VinPenalty	
Board	**Connections**	**Fails**	**Time**	**Fails**	**Time**
jeval2	1040	162	566	51	554
jbaseb	634	91	964	60	842
jmain1	604	129	936	110	982
jmarc2	1035	126	4295	103	4599
jepnwb	2081	8	6661	2	6292
jdipmb	3314	52	4647	6	4884
jmarc4	204	22	102	11	94
jresbd	1561	221	4250	213	4654
jepbrb	2546	27	7419	19	7992
jdipbd	2432	88	8659	67	8325
jcontb	1685	278	1731	248	1761
jemcpb	1485	131	12420	82	13483
jtrmem	3151	456	3062	353	2525
jwheel	2431	343	12942	290	13071
jmain2	1621	48	2664	3	2397
jtimvb	2102	307	14985	284	12919
jfocbd	2733	488	12669	466	13923
jnmlkb	1805	340	2925	304	2801
jwireb	1886	279	13493	253	13089
jaclbd	1906	312	3822	278	3564

4.2 Selection of Cutoff Parameters

The basic operation of Vintr's algorithm is to set up $CutoffTable[0, \ldots, n-1]$ where n is the number of iterations to be performed. All the nets on the board are routed in turn. The final cutoff value $CutoffTable[n-1]$ is set to 100 to ensure that the final iteration does not contain any intersections on completion. The initial cutoff value, $CutoffTable[0]$, and the subsequent rate of increase of successive cutoff values plays a crucial role in guiding the iterative process to its final solution.

High starting values for $CutoffTable[0]$ will not prevent the terminal isolation problem. Simulations with different values of initial cutoff lengths suggest that long tail nets are still liable to contribute to failures due to blockage of terminals belonging to other unrouted nets. Any intersections that still remain tend to get resolved by means of long detour paths that may adversely affect the routing of other nets. Low starting values on the other hand allow conflicts to be resolved with shorter detour paths. More importantly, all nets are given a chance to compete for routing resources before the higher cutoff lengths start eliminating some of the nets. With high initial cutoff values, it is not unusual for a net to be eliminated (because of failure) before it has had a chance to compete for routing resources.

The other important factor in determining the quality of the final solution is the rate of increase of successive cutoff values. High rates of increase of cutoff values reduce the chances of the iterative process from escaping from any local minima encountered. Eliminating intersections tends to become increasingly dependent on the ordering algorithm in such cases. A net that suddenly finds its path blocked might not have a chance to get routed elsewhere due to the lack of routing space. Even if it does get rerouted successfully, the detours taken may be so preposterous that other nets routed after it may be affected in a similar manner. A low rate of increase, on the other hand, is impractical so far as CPU effort is concerned. Several iterations may take place without any significant change in the layout of all the connection paths.

Simulations showed the following parameter set, $P3$, gave a good compromise between routing quality and CPU time:

$$P3 = [2, 4, 6, 8, 10, 20, 30, 40, 50, 60, 70, 80, 90, 92, 94, 96, 98, 100]$$

Initially, small increments are used to steer nets towards optimal paths without implementing excessively long tails from the net terminals. This increment is then increased to 10 till the 90% cutoff limit is reached when the increment is then decreased again in an attempt to resolve all the

Table 2: **Comparison of *P*2 and *P*3 for** VinPenalty

Parameter Set		*P2*		*P3*	
Board	**Connections**	**Fails**	**Time**	**Fails**	**Time**
jeval2	1040	51	554	48	918
jbaseb	634	60	842	44	1536
jmain1	604	110	982	92	1456
jmarc2	1035	103	4599	85	6977
jepnwb	2081	2	6292	2	10815
jdipmb	3314	6	4884	3	7505
jmarc4	204	11	94	7	164
jresbd	1561	213	4654	150	7111
jepbrb	2546	19	7992	1	11932
jdipbd	2432	67	8325	26	13894
jcontb	1685	248	1761	190	2738
jwheel	2431	290	13071	247	20847
jmain2	1621	3	2397	1	4138
jtimvb	2102	284	12919	250	21284
jfocbd	2733	466	13923	325	20711
jnmlkb	1805	304	2801	245	4671
jwireb	1886	253	13089	193	21972
jaclbd	1906	278	3564	163	5774

remaining intersections. The results obtained using *P*3 on VinPenalty are tabulated in Table 2. For comparison, the results obtained routing the same boards using the parameter set, *P*2 are also shown. The improvement using *P*3 compared to *P*2 is apparent, though this has been achieved at the expense of additional CPU time.

4.3 Comparison of VinPenalty with Lee's algorithm and other iterative routers

This section compares the relative performance of Vintr's algorithm (VinPenalty using *P*3) with a Lee (a standard Lee router) [16, 2] and RubinMoore (an iterative router based on algorithms suggested by Rubin and Moore). RubinMoore was selected as a basis for comparison because, like VinPenalty, a constant crossover penalty is applied during each iteration. RubinMoore can easily be adapted from the Vintr's algorithm. The crossover penalty is varied after each iteration and the cutoff lengths left fixed at 0 for every iteration. This ensures that the full extent of each net is crossable. At the end of the final iteration the routing will invariably contain crossovers. Nets are then removed starting with the net containing the most intersections, and so on until there are no design violations left. The removed nets are then rerouted to see if any alternative intersection free path exists. An initial crossover penalty of 0 is used which is incremented by 1 after every iteration till a maximum crossover penalty of 15 (16 iterations).

Table 3 compares the results obtained using RubinMoore, VinPenalty and Lee. The boards are listed in order of estimated complexity (where the complexity is an estimated degree of difficulty in routing the board).

Examination of the results, shows both RubinMoore and VinPenalty showing an improvement over Lee for the majority of the boards with RubinMoore obtaining the more consistent results. It is noticeable that VinPenalty obtains the best results, of the three, for the relatively low complexity boards (see Table3). As the complexity of the boards increase, RubinMoore starts consistently obtaining the better results (results for *jdipbd* onwards). This can be explained by considering the semi-greedy nature (gradually consuming routing resources with no crossovers) of Vintr's algorithm and its variants. This does not cause too much of a problem with light to medium congested boards. The semi-greedy problem tends to manifest itself in heavily congested test boards. Practical situations in which this problem is most apparent are those in which a design has been reduced to fewer layers when it clearly needs more layers to get routed, or even in relatively simple designs in which there is a heavy concentration of connections in a limited area of the board. In these situations too many (initially all) nets are being allowed to compete for routing resources even though some of them will eventually fail. This may cause several excessive detours initially, and when the routing space is eventually freed, it may be to late to take advantage of it. One would expect the routing quality in such situations to be severely degraded, and even worse than Lee at times (see results for *jdipbd* onwards in Table 3). This problem does not affect RubinMoore because of its non-greedy nature. It can be concluded that Vintr's algorithm works best when resolving discrete conflicts on sparse to medium density boards.

4.4 An Integrated Approach - VinHybrid

Based on results presented in Table 3, the behaviour of Vintr's algorithm can be proposed as follows:

- For low to medium congested boards, the outgrowing tail nets from all the terminals serve to prevent the terminal isolation problem. The results indicate that Vintr's algorithm under these circumstances gives better results than both Lee and a conventional iterative router RubinMoore.
- For those boards that are heavily congested, or have specific areas congested, the outgrowing tail nets tend to hinder other connections as the tails get longer and longer and more routing resources are unnecessarily consumed in a greedy manner. The number of intersections tends to increase rather than get resolved. In these situations Vintr's algorithm gives worse results than RubinMoore and at times even Lee gives better results depending on the extent of congestion on the board.

An alternative approach based on the latter observations is now investigated which combines the advantages of

Table 3: **Comparison of** VinPenalty**,** RubinMoore **and** Lee

Router		VinPenalty		RubinMoore		Lee	
Board	**Connections**	**Fails**	**Time**	**Fails**	**Time**	**Fails**	**Time**
jeval2	1040	48	918	76	656	203	38
jbaseb	634	44	1536	43	1179	65	93
jmain1	604	92	1456	68	1186	140	80
jmarc2	1035	85	6977	82	5830	87	323
jepnwb	2081	2	10815	9	10573	11	546
jdipmb	3314	3	7505	18	7018	56	335
jmarc4	204	7	164	9	158	11	7
jresbd	1561	150	7111	131	8500	148	302
jepbrb	2546	1	11932	18	11779	24	602
jdipbd	2432	26	13894	34	15018	45	686
jcontb	1685	190	2738	167	2229	267	202
jwheel	2431	247	20847	130	21840	203	982
jmain2	1621	1	4138	19	4285	25	203
jtimvb	2102	250	21284	124	22695	215	1062
jfocbd	2733	325	20711	245	21257	357	917
jnmlkb	1805	245	4671	140	4632	285	273
jwireb	1886	193	21972	106	23239	188	1089
jaclbd	1906	163	5774	150	5510	219	235

Vintr's algorithm with the consistency of RubinMoore. The iterative process is now divided into 2 phases as follows:

- The first phase (Vintr's phase) varies the cutoff length with a constant crossover penalty as before. When a specified cutoff limit is reached the second phase (Rubin's phase) is entered.

- The second phase of the algorithm keeps the cutoff length constant and instead varies the crossover penalty.

The cutoff limit chosen should be long enough to reduce the probability of terminal isolation whilst still ensuring that a minimal amount of routing resources is actually fixed. This approach will now be referred to as VinHybrid in all the results that follow. VinHybrid was run on all the test boards with the following cutoff parameter set for Vintr's phase:

$$P4 = [2, 4, 6, 8, 10]$$

A constant crossover penalty of 0 was used in Vintr's phase. At the 10% cutoff point the crossover penalty is incremented by 1 till a maximum penalty of 15 (total of 20 iterations). The final conflict free layout is generated as before. The ideal cutoff limit for each board will vary, but 10% was found to give reasonably good results. The denser the routing problem, the lower the cutoff limit should be to prevent an excessive amount of routing resources being consumed greedily. Table 4 outlines the results obtained. It is noticeable VinHybrid gives better results, in most cases, than both Lee and VinPenalty no matter how heavily congested the board is. This is because the proportion of nets that get fixed is now carefully controlled.

Table 4: **Comparison of** VinPenalty**,** VinHybrid **and** Lee

Router		VinPenalty		VinHybrid		Lee	
Board	**Connections**	**Fails**	**Time**	**Fails**	**Time**	**Fails**	**Time**
jeval2	1040	48	918	93	880	203	38
jbaseb	634	44	1536	47	1540	65	93
jmain1	604	92	1456	69	1504	140	80
jmarc2	1035	85	6977	78	7593	87	323
jepnwb	2081	2	10815	5	13751	11	546
jdipmb	3314	3	7505	20	9171	56	335
jmarc4	204	7	164	11	217	11	7
jresbd	1561	150	7111	134	9991	148	302
jepbrb	2546	1	11932	23	15369	24	602
jdipbd	2432	26	13894	42	17325	45	686
jcontb	1685	190	2738	186	2955	267	202
jwheel	2431	247	20847	129	28468	203	982
jmain2	1621	1	4138	17	5609	25	203
jtimvb	2102	250	21284	131	29890	215	1062
jfocbd	2733	325	20711	246	31193	357	917
jnmlkb	1805	245	4671	153	5911	285	273
jwireb	1886	193	21972	104	30121	188	1089
jaclbd	1906	163	5774	137	7258	219	235

4.5 Incorporating Dynamic Routing

VinPenalty attempts to discourage crossovers by making the crossover penalty very large but does not prevent them altogether. A more restrictive variant has been implemented that is to a small extent dynamic (tries to reroute nets around obstacle nets). The path finding procedure is now subdivided into 2 phases:

- The first phase restricts the search domain only to those resources that are not occupied by any other net. If this phase is successful, an intersection free path has been found, otherwise the second phase of the search is entered.

- The second phase attempts to find a path penalising crossovers in the usual manner. The search domain is now extended to include any routing resource that is crossable.

Table 5: **Results of two-phase path finder for** VinHybrid

Two-phase Path Finder		*False*		*True*	
Board	**Connections**	**Fails**	**Time**	**Fails**	**Time**
jeval2	1040	93	880	39	1104
jbaseb	634	47	1540	23	1961
jmain1	604	69	1504	60	1706
jmarc2	1035	78	7593	2	7502
jepnwb	2081	5	13751	0	9633
jdipmb	3314	20	9171	0	7890
jmarc4	204	11	217	5	163
jresbd	1561	134	9991	121	10128
jepbrb	2546	23	15369	0	9457
jdipbd	2432	42	17325	3	11643
jcontb	1685	186	2955	75	3103
jemcpb	1485	39	30240	16	17123
jwheel	2431	129	28468	112	29900
jmain2	1621	17	5609	0	3608
jtimvb	2102	131	29890	129	26671
jfocbd	2733	246	31193	213	29027
jnmlkb	1805	153	5911	112	5186
jwireb	1886	104	30121	82	30747
jaclbd	1906	137	7258	103	5666

The effect of the two-phase path finder is to restrict crossovers to situations where it is absolutely necessary. Table 5 summarises the results obtained when the two-phase path finder is incorporated with VinHybrid. The results show a further reduction in the number of unrouted connections when compared with the non dynamic version of VinHybrid. In some cases the improvements are quite substantial.

5 Conclusions

This paper has described a series of experiments with an iterative routing algorithm. The success of iterative routing algorithms in general over serial routing algorithms has been shown. This paper considered 2 iterative routing approaches, Vintr's algorithms and Rubin's algorithm. Both implementations (with Rubin's approach being derived from the basic Vintr's algorithm) have succeeded in obtaining a better completion rate than a single iteration of Lee for all the test cases examined, though it has been shown that Vintr's algorithm has to be carefully controlled in order to do so.

References

[1] C. Lee. An algorithm for path connections and its applications. In *IRE Transactions on Electronic Computers*, pages 346 – 365, 1961.

[2] Z. Moosa. *On Improving Maze Routing Algorithms*. PhD thesis, University of Manchester, 1992.

[3] L. Abel. On the ordering of connections for automatic wire routing. *IEEE Transactins on Computers*, pages 1227 – 1233, 1972.

[4] R. Linsker. An iterative improvement penalty function driven wire routing system. In *IBM Journal of Research and Development*, number 28, pages 613 – 624. 1984.

[5] R. Linsker. Wire routing with path history dependent penalty functions. In *IBM Technical Disclosure Bulletin*, volume 27, pages 399 – 406. 1984.

[6] A. Moore and C. Ravitz. Weighted and iterative multi-wire routing. In *IBM Technical Disclosure Bulletin*, volume 25, pages 3619 – 3628. 1982.

[7] R. Rosenburg. A new iterative supply/demand router with ripup capability. In *24'th ACM/IEEE Design Automation Conference*, pages 721 – 726, 1987.

[8] F. Rubin. An iterative technique for printed wire routing. In *11'th ACM/IEEE Design Automation Conference*, pages 308 – 313, 1974.

[9] J. Soukup. Circuit layout. *Proceedings of the IEEE*, 69(10):1281 – 1304, 1981.

[10] W.A. Dees and P.G. Karger. Automated ripup and reroute techniques. In *19'th ACM/IEEE Design Automation Conference*, pages 432 – 439, 1982.

[11] K. Kawamura, M. Umeda, and H. Shirashi. Hierarchical dynamic router. In *23'rd ACM/IEEE Design Automation Conference*, pages 803 – 809, 1986.

[12] Z. Moosa, M. Brown, and D. Edwards. An application of simulated annealing to maze routing. In *European Design Automation Conference*, pages 434 – 440, 1994.

[13] W.S. Scott and J.K. Ousterhout. Plowing: Interactive stretching and compaction in magic. In *21'st ACM/IEEE Design Automation Conference*, pages 166 – 172, 1984.

[14] H. Shin and A.S. Vincentelli. A detailed router based on incremental routing modifications : Mighty. *IEEE Transactions on CAD*, 6(6):942 – 955, 1987.

[15] I. Shirakawa and S. Futagami. A rerouting scheme for single layer printed wiring boards. *IEEE Transactions on CAD*, 2(4):267 – 271, 1983.

[16] T.D. Spiers. *A Hardware Assisted Routing System*. PhD thesis, University of Manchester, 1985.