# Genetic Algorithms versus Tabu Search for Instruction Scheduling

Steven J. Beaty *
NCR Microelectronics
2057 Vermont
Fort Collins, Colorado 80525
Steve.Beaty@ftcollins.ncr.com
beaty@longs.lance.colostate.edu

## Abstract

Most scheduling problems require either exponential time or space to generate an optimal answer [7]. Instruction scheduling is an instance of a general scheduling problem and Dewitt [8] uses this fact to show instruction scheduling is a NP-complete problem. This paper applies Genetic Algorithms, Tabu Search, and list scheduling to the instruction scheduling problem and compares the results obtained by each.

## 1 Introduction

Sequencing is defined by Ashour [1] as being "concerned with the arrangements and permutations in which a set of jobs under consideration are performed on all machines." That is, what is the order the jobs will be performed; what is the priority of each job? Sequencing thereby ranks the jobs to be executed. Baker [2] states "scheduling is the allocation of resources over time to perform a collection of tasks." Scheduling usually places already ordered jobs into slots, often accounting for conflicts in resource usage. The combined sequencing/scheduling (order/place) process produces the desired outcome: jobs placed on machines capable of performing the desired tasks in the correct order at a correct time.

Instruction scheduling (IS) involves the placement of atomic machine operations into machine instructions. A data dependence DAG (DDD) is often used to describe the necessary operations and their order. The nodes in a DDD contain the operations, and the edges denote a partial order on the nodes. This partial order is used to guarantee both program dataflow and machine resource requirements. The edges of a DDD do not constrain the order nodes are scheduled, only the order they appear in the final schedule. The solution space may be viewed as an incomplete $n$-dimensional hypercube, where $n$ is the number of operations to be performed. Each operation might be executed at a variety of locations in the code, and each dimension represents the range of instructions that operation can be placed. IS is complicated by both the inherent dataflow ordering between operations in the source code and the complexities of the architecture of the target machine. The architecture may have complex timings between operations, a number of different field encodings, and a limited number of resources that can perform any given operation.

Most existing IS methods rely on heuristics to remove the examination of parts of the search space that appear fruitless. Using heuristics can be difficult when attempting to arrive at an efficient yet efficacious scheduler. This difficulty is compounded by several factors.

- The heuristics generally must be regenerated for each machine targeted.
- The heuristics themselves are not in a form easily understood by humans, thus making it difficult for humans to correctly choose and modify a scheduler's behavior.
- It is possible that the heuristics do not address an issue that has great influence on the final code.
- Heuristics that work well for one ordering of operations may not work well for another.
- Heuristics are also picked before the execution of the instruction scheduling routine and remain static throughout. They have no ability to learn from previous runs or to take advantage of anomalous situations existing in specific situations that lead to shorter code sequences.

With these difficulties in generating schedulers, several stochastic methods have been attempted to solve the IS problem. In Jacobs et al. [15], the Metropolis Monte Carlo

* the author is an affiliate faculty member at Colorado State University

technique was used. De Gloria and Faraboschi [11, 12] used a Boltzmann Machine approach to good effect. In [4, 3], Genetic Algorithms were shown to produce good results.

List scheduling (LS) is a general [7] scheduling method often used for instruction scheduling [10]. LS builds a ready set that contains all jobs that are not waiting on the results of another job. In a DDD, this is represented as nodes with no unscheduled predecessors. In finding the ready set, LS performs a topological sort of a DDD, thereby reducing the search space of the scheduling problem and increasing the chances of finding a valid schedule. List scheduling has an implicit heuristic: scheduling nodes with no predecessors results in valid orderings more often than scheduling nodes with predecessors. As with all heuristics, there are instances where this assumption does not hold.

A difficulty with using LS in combination with stochastic methods is it requires time $O(n^2)$ [17, 10], where $n$ is the number of nodes in the graph. Most stochastic methods require the evaluation of numerous of node orderings, creating a desire for a scheduling method with less time complexity. It is unnecessary to use the topological ordering of list scheduling if it can be replaced by a strong method of sequencing. Lookahead scheduling [5] addresses these concerns by providing a fast method to scheduling operations, without incurring the overhead of list scheduling.

## 2 Genetic Algorithms

Genetic Algorithms have been used successfully to perform TSP [21, 22], job shop [22], and flow shop [6, 19] optimization problems. Encouraging results from these problems drove the use of GAs for instruction scheduling.

The GENITOR GA program, developed by Whitley [20, 23], was used for these studies. It has some differences with "standard" GAs that appear to increase performance. It does not replace the entire population with each generation. Instead it probabilistically chooses two parents to reform into two offspring. Recombination and mutation occur, then one of the offspring is discarded randomly. The remaining offspring is placed in the population according to its fitness in relation to the rest of the strings. The lowest-valued string is discarded. This keeps high-valued strings within the population, directly accumulating high-performance hyperplanes. It also bases the reproductive opportunity upon rank with the population, not upon a string's fitness value in comparison with the average of the population, reducing the impact of selective pressure fluctuation. It also reduces the importance of choosing a proper evaluation function for fitness in that the difference in the fitness function between two adjacent strings is irrelevant.

An evaluation function that ranks the fitness of a string in the population must be produced. Choosing a proper function, i.e., one that represents a string's relative worth in the population without inordinate bias, is important. For instruction scheduling, a minimization problem, the result of the evaluation function must reflect the length of the final schedule that a member of the population generates. A difficulty encountered is that not all members will produce valid final schedules. Failures will occur when a conflict arises (e.g. timing, resource, or field) due to the order of scheduling the operations. It is not surprising that certain orders will fail to produce valid schedules for a given DDD; the impact of ordering on the production of valid schedules is emphasised in all previous instruction scheduling methods.

After consideration, the evaluation function selected performs a "worst-case" evaluation when a string fails to produce a valid schedule. This evaluation is produced by assuming all unscheduled operations have no parallelism available in them, necessitating their serial placement. The calculation of the evaluation function is then trivial; it is the number of instructions that contain operations so far, plus the length of the path containing the serial ordering of all the unscheduled operations. This produces a good estimate in the event of schedule failure; those schedules with more operations placed will receive a better evaluation. It also produces an exact evaluation in the presence of a valid schedule.

Six different recombination operators were studied. These are described in Starkweather et al. [19] and include two order crossovers, partially mapped crossover, cycle crossover, position-based crossover, and edge recombination. Starkweather et al. demonstrate that each operator will perform differently for each problem domain. The performance difference can be measured in the speed of convergence to a good solution. For example, edge recombination finds good solutions more rapidly on the TSP while performing more poorly than the others on scheduling problems.

The number of generations should be related to the relative difficulty of producing an optimal schedule for a given DDD. DDDs with a few simple operations do not require as many generations to find good schedules as do those with many complex operations. For these experiments, the number of generations is $n^2$, where $n$ is the number of operations. The size of the genetic population is $n$. The strings in the population are of strings of non-repeating integers. This representation is consistent with those used in the TSP and shop scheduling problems previously mentioned. All strings are randomly initialized. No effort is made to optimize GA parameters for IS in this study. The selection bias is 1.5. There is no mutation, adaptive or otherwise.

```
tabu_search ()
{
    for (i = 1; i <= # iterations; i++)
    {
        value = best_move ();
        make best_move;
        make best_move tabu;
        if (value < global_best)
        {
            global_best = value
        }
    }
}
```

Figure 1: Tabu Search

## 3 Tabu Search

Tabu search (TS) is an optimization method that uses a form of short-term memory used to keep a search from becoming trapped in a local minima. A tabu list is formed that keeps track of recent solutions. At each iteration in the optimization process, solutions are checked against the tabu list. A solution that is on the list will not be chosen for the next iteration (unless it overrules its tabu condition by what is called an aspiration condition.) The tabu list forms the core of tabu search and keeps the process from cycling in one neighborhood of the solution space.

At each iteration, a steepest-descent solution that does not violate the tabu condition is chosen. If no non-tabu improving solution exists, the best non-improving solution is taken. The combination of memory and gradient descent allows for diversification and intensification of the search. Local minima in the search space are avoided while good areas are well explored.

Two bits of pseudo-code will show the basic of the TS method used here. The first, in Figure 1, is the overhead procedure. It controls the number of iterations, updating of the best solution so far, and controls the tabu list. The second, in Figure 2, finds the next move in the search by a swapping procedure. All the possible swaps in the sequence are tried, and the best non-tabu swap is chosen. The routine shown finds the best move from the current location in the search space to a neighboring position. An alternative is to find the first location in the neighborhood that is an improvement over the current one. These two possibilities are termed *best improving* and *first improving* respectively.

TS has been effectively used for a number of problems related to the IS problem. Glover and McMillan [13] used it for employee scheduling, Eck [9] studied Job shop scheduling, and Laguna et al. [16] applied TS to machine scheduling. The success in these areas helped motivate

```
best_move ()
{
    for (i = 1; i < n; i++)
    {
        for (j = i + 1; j <= n; j++)
        {
            swap (sequence[i,j]);
            value = evaluate (sequence);
            if (tabu[i][j] &&
                value > global_best)
            {
                continue;
            }
            if (value < best_so_far)
            {
                best_so_far = value;
                best_move = [i,j];
            }
        }
    }
    return best_so_far;
}
```

Figure 2: Best Move

this study of instruction scheduling.

For this study, two different resequencing operators were applied. The first swapped two machine operations in the sequence and evaluated the resulting sequence. The second performed an insertion procedure by removing one operation from the sequence, shifting the remaining elements to fill in the open spot up a certain point and then placing the removed operation into that spot. In some cases, the insertion procedure may prove more suited for a sequencing task if the sequence is almost completely optimized. For example, if an optimal sequence is 1 2 3 4 5 6 7 8 9, and the current solution is 1 9 2 3 4 5 6 7 8, the swap procedure would take more iterations to arrive at the optimal solution. This can occur in IS when the last node is a branch instruction and must be placed last in the block.

There are a number of different types of information that can be kept on the tabu list. For example, when a operation is moved from one position in the sequence to another, one could make moving that operation back to its original position tabu. One could keep the relative position information for each operation. The total sequence could be saved. In these experiments, the contents of the tabu list for the two procedures was different. For the swap procedure, the list contained pairs of operations that had been swapped recently. This kept operations from reversing their current relative positions. Insertion is more difficult to express as a relative condition as each insertion

changes the position of many different operations. For this case, the tabu list contained the actual permutations from the recently effective evaluations.

The same evaluation function (lookahead scheduling) used with GAs, was used for TS. The first improving move scheme was employed. The tabu list size was of length seven for both the swap and insert procedures. Various other lengths were studied (e.g. length $n$, where $n$ was the number of MOs) and found to produce very similar results. This suggests that the local minima neighborhoods are fairly small for this instance of the IS problem. The number of evaluations was limited to $n^2$ as in the GA approach. A running average was kept in order to compare directly with the GA results.

# 4 Comparisons and Conclusions

A number of different programs were run through the compiler in order to compare the effectiveness of GAs, TS, and list scheduling. The compiler was targeted to produce code for the IBM RS/6000 architecture [14]. A representative example is shown in Figure 3. This graph represents the major block found in the forth Lawrence Livermore kernel [18]. The `EDGE_RECOMB`, `ORDER1`, `ORDER2`, `PMX`, `CYCLE`, and `POSITION` lines are the six genetic operators. The `LIST` line represents the list scheduling result. Note it is drawn to give a reference, it requires only one evaluation to compute. The `INSERT` and `SWAP` lines are the two tabu operators. Both GAs and tabu search worked well for finding good solutions to IS problems.

Genetic operators emphasizing order converged faster than those emphasizing adjacency. This comes as no surprise; all previously effective methods for IS also emphasize order. This evidence does however shed additional light on the nature of of the instruction scheduling process by providing more controlled, empirical evidence. The ordering of the placement of nodes by the genetic algorithm mirrors the approach used by human coders. The nodes with the greatest impact on final schedule length are placed first, with those having lesser impact placed later. The order of placement that ensures validity is also reflected.

In this study, both the swap and insert operators demonstrated very similar behavior, pointing to the fact that absolute order is not of ultimate importance for this particular IS problem. Both were able to avoid local minima and find competitive solutions. It took tabu search longer to find the better solutions than the best genetic operators. This may be a reflection of the fact that GAs are more suited to the IS problem. It could also be that the genetic technique used is more highly "evolved" having been used for a number of sequencing problems before.

# References


[1] S. Ashour. *Sequencing Theory*. Springer-Verlag, New York, 1972.

[2] K. R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley and Sons, Inc., New York, 1974.

[3] S. Beaty. Genetic algorithms and instruction scheduling. In *Proceedings of the 24th Microprogramming Workshop (MICRO-24)*, Albuquerque, NM, November 1991.

[4] S.J. Beaty. *Instruction Scheduling Using Genetic Algorithms*. PhD thesis, Mechanical Engineering Department, Colorado State University, Fort Collins, Colorado, 1991.

[5] Steven J. Beaty. Lookahead scheduling. In *Proceedings of the 25th Annual International Symposium on Microarchitecture (Micro-25)*, pages 256–259, Portland, Oregon, December 1992.

[6] Gary A. Cleveland and Stephen F. Smith. Using genetic algorithms to schedule flow shop releases. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[7] E.G Coffman. *Computer and Job-Shop Scheduling Theory*. Jon Wiley & Sons, New York, 1976.

[8] D.J. DeWitt. *A Machine-Independent Approach to the Production of Optimal Horizontal Microcode*. PhD thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, 1976.

[9] B.T. Eck. Good solutions to job shop scheduling problems via tabu search. Technical report, Department of Industrial Engineering and Operations Research, Columbia University, New York, May 1989.

[10] F. Gasperoni. Compilation techniques for vliw architectures. Technical report, Courant Institute of Mathematical Sciences, New York University, March 1989.

[11] A. De Gloria and P. Faraboschi. A boltzmann machine approach to code optimization. *Parallel Computing*, 17:969–982, December 1991.

[12] A. De Gloria, P. Faraboschi, and M. Olivieri. A non-deterministic scheduler for a software pipelining compiler. In *Proceedings of the 25th Annual International Symposium on Microarchitecture (Micro-25)*, pages 41–44, Portland, Oregon, December 1992.

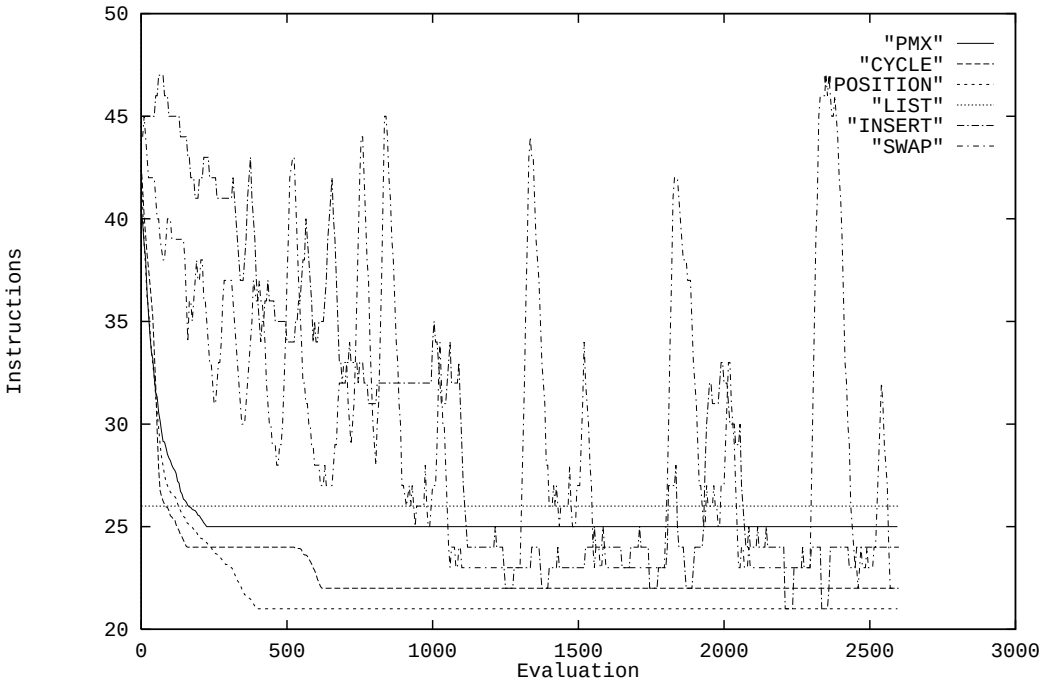


Figure 3: Comparative Results

[13] F. Glover and C. McMillan. The general employee scheduling problem: An integration of management science and artificial intelligence. *Computers and Operations Research*, 13(5):563–593, 1986.

[14] IBM. *IBM Journal of Research and Development*, January 1990.

[15] Dean Jacobs, Jan Prins, Peter Siegel, and Kenneth Wilson. Monte carlo techniques in code optimization. In *Proceedings of the 15th Annual Workshop on Microprogramming (Micro-15)*, pages 143–148, Palo Alto, California, December 1982.

[16] M Laguna, J.W. Barnes, and F. Glover. A tabu search method for scheduling jobs on parallel processors. Technical report, Department of Mechanical Engineering, University of Texas-Austin, November 1989.

[17] D. Landskov, S. Davidson, B.D. Shriver, and P.W. Mallett. Local microcode compaction techniques. *ACM Computing Surveys*, 12(3):261–294, September 1980.

[18] F.H. McMahon. The livermore fortran kernels: A computer test of numerical performance range. Technical report, Lawrence Livermore National Laboratory, December 1986.

[19] T. Starkweather, S. McDaniel, K. Mathias, C. Whitley, and D. Whitley. A comparison of genetic sequencing operators. In *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991.

[20] D. Whitley and J. Kauth. Genitor: a different genetic algorithm. In *Proceeding of the Rocky Mountain Conference on Artificial Intelligence, Denver, Co.*, pages 118–130, 1988.

[21] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[22] D. Whitley, T. Starkweather, and D. Shaner. The traveling salesman and sequence scheduling quality solution using genetic edge recombination. In L. Davis, editor, *The Genetic Algorithms Handbook*. 1990.

[23] Darrell Whitley. The GENITOR algorithm and selective pressure: Why rank - based allocation of reproductive trials is best. In *Proceeding of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.