# Genetic Algorithms for Instruction Sequencing and Scheduling

Steven J. Beaty [*]
NCR Microelectronics
2057 Vermont
Fort Collins, Colorado
80525
(303) 226-9622
Steve.Beaty@FtCollinsCO.ncr.com
beaty@longs.lance.colostate.edu

## Abstract

*In forming a complete schedule for jobs (such as instructions on a processor capable of multiple instruction issue), two independent operations occur: sequencing of the jobs and scheduling those prioritized jobs. This paper discusses the importance of the distinction between sequencing and scheduling, and gives a number of examples based on instruction scheduling to clarify the differences. It then discusses the application of Genetic Algorithms to different types of sequencing and scheduling problems.*

## 1 Introduction

In much of the current literature the terms sequencing and scheduling are used interchangeably; indeed Coffman [Cof76] explicitly endorses this fact. The reason for the duplicity of the terms is probably due to the fact that the two operations are usually either intimately tied together or irrelevant to each other. Indeed, for many scheduling tasks sequencing is required and implicit. Four different combinations of sequencing and scheduling, and the need for explicit delineation of the two, will be addressed by this paper.

## 2 Sequencing and Scheduling

Sequencing is defined by Ashour [Ash72] 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 [Bak74] states "Scheduling is the allocation of resources over time to perform a collection of tasks." Scheduling usually places already prioritized 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. Ashour notes another reason for the confusion of terms results from the implicit assumption that each job will start as early as possible; a schedule is therefore automatically formed once the sequence is known. In certain instances, such as those representable as a flow-shop (FIFO) model, this assumption is valid. In more complex models, such as those representable as a job-shop, the assumption is not.

Both sequencing and scheduling can be complex tasks independent of each other. Sequencing itself is $O(J!^M)$ where $J$ is the number of jobs and $M$ is the number of machines [Bak74]. Even if there is only one machine, the calculation of all the possible permutations is prohibitively expensive. For this reason, sequencers are usually heuristically driven, producing a sequence that is reasoned to be superior to the others possible.

Scheduling is also a difficult problem. Brightwell and Winkler [BW90] proved that producing all of the total orders from a given partial order is #P-complete, showing that exhaustive searching by a scheduler for an optimal answer is also not advisable. For this reason, schedulers are often deterministic such that given any one sequence of jobs, the scheduler will always produce the same final schedule. Schedulers take a sequence and a partial order, and create a total order of jobs such that no schedule constraints (e.g. timing or resource) are violated.

The form of the input to the complete scheduling problem must express the partial order formed by the precedence relations between the jobs that need to be completed. Graphs are often used for this form; they are directed to express the precedence and they are acyclic as a job cannot precede itself (giving a Directed Acyclic Graph or DAG.) Trees

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

are a form of DAGs but are limited by the fact that they cannot express that a job depends on more that one directly preceding job. Often a matrix representation of the graph is used for compactness and for its analytic properties.

The following subsections will discuss the following combination of sequencing and scheduling:

1. sequencing before scheduling,
2. sequencing during scheduling,
3. scheduling without sequencing, and
4. sequencing without scheduling.

## 2.1 Sequencing then Scheduling

Probably the most familiar combination of the two operations is first prioritizing the jobs and then placing them in the final schedule. Sequencing often has a large impact on the final schedule when combined with a deterministic scheduler; the only intelligence and flexibility available in the process is in the ordering of jobs. An example of how the two tasks fit together may be found in the list scheduling algorithm. A general list scheduling algorithm may be found in Coffman; a specific instruction scheduling method will be used here for demonstration purposes. An instruction list scheduler (ILS) takes a group of required machine operations (MOs) and places them in instructions. The ILS forms a data-ready set (DRS) that contains all of the MOs that are not waiting on data produced by a previous MO. Within this set the MOs are ordered based on one or more heuristics (see [Bea91b] for a list of twenty-six such heuristics) due to the large number of permutations available in a DRS. Note that this may be viewed as a static ordering such that a global ordering of MOs can be generated, and each individual DRS can derive its order from that. Whether the DRS is ordered globally or locally, the sequence must be completed before any scheduling takes place.

## 2.2 Scheduling then Sequencing

There are instances when scheduling may partially precede sequencing. This is advisable when the formation of a partial schedule should change the priorities of the remaining jobs to be placed. This interaction is a two-way street: the sequence changes based on the schedule, and the final schedule is changed due to the re-prioritizing of the jobs. An initial sequence is needed to start the scheduling process. An example is the distribution of memory operations in a RISC architecture. The penalty of the load or store delay may be best offset with MOs of other types instead of a long string of memory operations. The memory pipeline can be stalled for the memory delay time between each successive load or store, and no useful work could be done. In contrast, if other (e.g. arithmetic) MOs can be scheduled that do not depend on a memory operation, they may complete earlier and shorten the overall length of the instruction sequence. The placement of memory operations cannot be known until the schedule is at least partially constructed. The schedule may then be examined and the priority of the various non-scheduled MOs modified to better reflect the partially constructed final schedule.

## 2.3 Scheduling without Sequencing

It is possible to have scheduling without having sequencing associated. This type of problem occurs when the order of placement has no impact on the final schedule; i.e. all orders of job placement produce identical final schedules. For the production of valid schedules, the scheduler must either backtrack in the presence of a resource conflict or there must be no such conflicts possible during job placement. Note that this is not the same as having the sequence become the schedule; that assumes there are no conflicts that must be examined and resolved. The scheduling step is necessary to ensure the correct placement of jobs. An example of scheduling without sequencing is lookahead instruction scheduling [Bea91b]. Here, a number of MOs may be constrained to specific instructions; if an MO cannot be placed in its specified instruction the entire schedule fails. The process that constrains the MOs basically determines an earliest and latest start time for all MOs in a Data Dependency DAG (DDD). When the earliest start time equals the latest start time for an MO, calculating its priority compared to other MOs is irrelevant: it can only be placed in one slot. If that slot is unavailable, the schedule generated so far is deemed invalid. Lookahead scheduling schedules all completely constrained MOs at each point during scheduling, thereby decreasing schedule generation time without any deleterious effects.

## 2.4 Sequencing without Scheduling

It is also possible to have sequencing without the need for scheduling. Baker [Bak74] calls this the "pure sequencing problem" and notes it is a specialized scheduling problem. A simple example is having a independent set of jobs and a single machine that can complete all tasks by itself. A critical point is there are no resource conflicts possible (one resource, one stream of jobs), and hence no need to form a schedule distinct from the sequence.

Within this model, it is possible that the order of tasks is irrelevant (e.g. all operations have equivalent setup and/or takedown times) so all final orders generated from the partial order produce schedules of equivalent length. This model is similar to processors with no form of parallelism and whose instructions all take one clock cycle [HL85]. The final schedule of instructions is usually based on the order of source code as this is a "natural" ordering.

It is also possible that operation costs vary for different orders of execution (e.g. operations having differing setup/takedown times.) The Traveling Salesperson Problem (TSP) is example of this type of problem. The sequence generated is the order cities are visited and there can be no resource conflict, but the distance between any two cities vary by the route chosen between them. If the cities and the connecting roads are viewed as resources, only one resource may be consumed at a time and there can be no contention (for there is only one consumer.)

### 2.5 Summary

In this section four combinations of sequencing and scheduling were discussed. One case not discussed is scheduling then sequencing. Once a schedule is formed, there is no reason to determine priorities on the jobs in the completed schedule.

## 3 Genetic Algorithms for Sequencing and Scheduling

For a thorough introduction to genetic algorithms (GAs) please see Goldberg [Gol89]. Briefly, GAs are a robust adaptive optimization technique based on a biological paradigm. They perform efficient search on poorly-defined spaces by maintaining an ordered pool of strings that represent regions in the search space. New strings are produced from existing strings using the genetic-based operators of recombination and mutation. Combining these operators with natural selection efficiently uses the hyperplane information found in the problem to guide the search. The searches are not greatly influenced by local optima or non-continuous functions.

A number of different genetic recombination operators (GROs) have been developed; each operates differently and is useful under different circumstances. The current set include operators that emphasize

- adjacency,
- relative position, or
- absolute position

within the parent strings when recombining [SMWM92]. GAs have been used in various sequencing and scheduling problems, and their efficiency for each is directly related to the type of recombination operator used. In pure sequencing problems, the GROs that emphasize adjacency find better solutions faster than those that emphasize order; for scheduling problems, the opposite occurs. Intuitively, this makes sense. In pure sequencing, there is no resource contention and the order of the sequence becomes the schedule. In scheduling problems, where resource contention exists, the relative ordering of the jobs has a large impact on the resource usage. Indeed, preserving adjacency in the sequence can form poor schedules. For example, if job $\alpha$ must precede job $\beta$ and both use the same (expensive) resource, keeping them adjacent in the sequence might disallow a scheduler placing any other tasks until both $\alpha$ and $\beta$ complete. If, however, $\alpha$ precedes but is not adjacent to $\beta$ any intervening jobs in the sequence may be scheduled.

### 3.1 Genetic Sequencing for Pure Sequencing Problems

When using GAs to perform TSP optimization, it is desirable to maintain any good subtours present in the parents as this leads to shorter overall tours in the children. A recombination operator that preserves edges, where an edge represents the connection between two cities, will exploit the most information available in the parents. In Whitley et al. [WSF89, WSS90], this preservation is achieved by making an edge map (edges are represented by adjacency in the sequence) and having a recombination operator use this map. This method does not use information on the distance between cities, only the distance of the overall tour, creating what is known as a "blind" traveling salesperson problem. The use of the overall tour length metric for the evaluation function is important because of its simplicity and applicability to many forms of sequencing and scheduling. The published results from this method are impressive [WSS90, UPvL$^+$90].

In scheduling machine usage on a flow shop floor, the flexibility is found in the sequence of jobs presented to the line. There are fixed setup, idle, and active costs for machines. A strict amount of product needs to be produced in order to meet the demand. Therefore, this optimization problem is one of sequencing the types of jobs presented to the first machine in the line, and can then be viewed as a problem similar to the TSP. The sequence is then evaluated to find the total cost which is passed back to the GA to drive the search.

In Whitley et al. [WSS90], a detailed description of an actual production line in use at Hewlett-Packard in Fort Collins, Colorado is discussed. It contains 6 workcells (groups of machines) in sequence, each performing a specific operation. Each has a single input and a single output queue. Every workcell contains two identical machines operating independently. The machines have costs associated with the various tasks they perform. There are twenty different types of products that can be produced by the line. Two different optimization approaches were tried: 1) a strict FIFO where the GA controlled the sequence of jobs presented to the line, and 2) a HYBRID where the GA attempted to optimize the initial sequence and a greedy algorithm attempted to reorder jobs in the line for maximal machine usage. The FIFO job is more difficult because it is not allowed to reorder jobs within the line. Both models try to keep all machines busy all the time. Surprisingly, the FIFO model produced better sequences, i.e. kept more machines busy more of the time resulting in lower cost. It also produced results faster than the HYBRID method. The results were not greatly different (approximately 3%) but the implication of not having to use a greedy, heuristic-based method (that requires more code and effort) is great. It is thought that FIFO worked better than HYBRID because of its ability to directly control all the global information. What appears good from a local greedy point of view is not always good from a global perspective. The FIFO is also probably a more realistic model of many other sequencing tasks.

In both of the TSP and flow shop problems, the edge recombination GRO worked best, as is expected due to the lack of resource conflicts.

### 3.2 Genetic Sequencing for Schedules

In Starkweather et al. [SMM$^+$91], the development of a system to schedule the production of beer at a large local brewery[1] is discussed. A simulator was developed that mirrored the actual production constraints of the brewery. The objective of the scheduler was the efficient allocation of orders to loading docks based on some fixed production cycle. Production occurs 24 hours a day on multiple lines, each line capable of producing a certain beer type. Different labeling and packaging possibilities exist that can complicate scheduling. Data were available on the different flow rates, start and stop times, and product type for each line as was a set of orders that needed to be filled. The goal was the reduction of average inventory (costly in many different ways) while filling as many orders as possible. The genetic algorithm manipulated strings representing the sequence beer orders were processed. Each sequence was run on the simulator to produce an evaluation, with the original genetic population being generated randomly. Results from simulations demonstrated that, for improvement, preserving relative order was more important than preserving adjacency in the population's strings. This is not surprising as adjacency has little meaning in this problem's context; items adjacent in the sequence were probably unrelated, with their completion occurring on separate production lines.

Beaty [Bea91a] discusses the results of combining instruction scheduling and genetic algorithms. Two approaches were reported.

The first approach used a GA in concert with a list scheduler. The strings in the population represented the priority of the nodes in the DDD, thereby creating the data ready sets. Most of the schedules produced were valid, and can be attributed to the power of the implicit heuristic of list scheduling: placing only data ready nodes. This combined list scheduling/genetic algorithm performed well. In simple DDDs, it easily found solutions as good as list scheduling alone. The combination also found some new best-known solutions to difficult DDDs. While this may appear surprising, consider the great lengths required to generate a set of heuristics that usually produce valid schedules. The heuristic with most impact on final schedule length, critical path, has been shown to produce erroneous results in simple cases [Bea91b]. Other heuristics, such as counting the number of restricted edges a node has, must be emphasised in an attempt to assure schedule validity. The emphasis therefore shifts (correctly) from the possibility of generating shorter schedules to the probability of producing valid schedules. The GA tends to place nodes in an order that produces shorter results, and with an evaluation function that reflects failure as a longer result, the GA will place nodes in an order that also produces valid results. A detraction from these encouraging results is the time complexity of the combined algorithms. List scheduling has a complexity of at least $O(n^2)$ [LDSM80], and this algorithm must be performed repeatedly.

A second method arose when it was noticed that with the application of the absolute timing algorithm [SDX87, WA89], each node already "knew" approximately where in the final schedule it must be placed. If the order of placement could be performed intelligently, there was no reason for a top-down (list scheduling) priority in node placement. Top-down priority has been used as a pseudo-intelligent form of ordering; it certainly increase the probability of valid schedules but it cannot adapt to vagaries of individual DDDs. With genetic algorithms an intelligent, adaptable method of node sequencing is available. The scheduling mechanism therefore became using the GA to pick the order operations are placed in the schedule. This works because each operation "knows" where it can be placed. Certainly, failures can occur due to choosing an improper order, thereby creating an invalid schedule. GROs emphasizing order converged faster than those emphasizing adjacency. This is no surprise as all previously effective instruction scheduling methods also emphasize order.

In summary, both the production and instruction schedulers are related to job shop scheduling, and in both GROs that emphasized order performed significantly better than those emphasizing adjacency.

## 4 Conclusions

Sequencing and scheduling are not the same and should not be treated as such. There are at least four different combinations of the two, each useful for different sets of problems.

[1]Coors Brewery, Golden Colorado.

Using Genetic Algorithms to solve different types of sequencing and scheduling problems, it was determined that in pure sequencing problems, the GROs that emphasize adjacency find better solutions faster than those that emphasize order. Conversely, for scheduling problems, the GROs that emphasize order produced better results. Thus, the different efficiencies the genetic recombination operators produce reflect the nature of the problem.

## References

[Ash72] S. Ashour. *Sequencing Theory*. Springer-Verlag, New York, 1972.

[Bak74] K. R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley and Sons, Inc., New York, 1974.

[Bea91a] S. Beaty. "Genetic algorithms and instruction scheduling". In *Proceedings of the 24th Microprogramming Workshop (MICRO-24)*, Albuquerque, NM, November 1991.

[Bea91b] S.J. Beaty. *Instruction Scheduling Using Genetic Algorithms*. PhD thesis, Mechanical Engineering Department, Colorado State University, Fort Collins, Colorado, 1991.

[BW90] Graham Brightwell and Peter Winkler. "Counting linear extensions is #p-complete". DIMACS Technical Report 90-49, Bellcore, 445 South Street, Morristown, New Jersey, 07960, July 1990.

[Cof76] E.G Coffman. *Computer and Job-Shop Scheduling Theory*. Jon Wiley & Sons, New York, 1976.

[Gol89] David Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[HL85] T.L. Harmon and B. Lawson. *The Motorola MC 68000 Microprocessor Family*. Prentice Hall, Englewood Cliffs, NJ, 1985.

[LDSM80] D. Landskov, S. Davidson, B.D. Shriver, and P.W. Mallett. "Local microcode compaction techniques". *ACM Computing Surveys*, 12(3):261–294, September 1980.

[SDX87] B. Su, S. Ding, and J. Xia. "Microcode Compaction with Timing Constraints". In *Proceedings of the 20th Microprogramming Workshop (MICRO-20)*, Colorado Springs, CO, December 1987.

[SMM$^+$91] 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.

[SMWM92] T. Starkweather, K. Mathias, D. Whitley, and S. McDaniel. "Sequence scheduling with genetic algorithms". In *New Directions in Operations Research*. Springer-Verlag, 1992.

[UPvL$^+$90] N. Ulder, E. Pesch, P. van Laarhoven, H. Bandelt, and E. Aarts. "Improving tsp exchange heuristics by population genetics". In *Parallel Problem Solving in Nature*. Springer-Verlag, 1990.

[WA89] P. Wijaya and V.H. Allan. "Incremental foresighted local compaction". In *Proceedings of the 22nd Microprogramming Workshop (MICRO-22)*, Dublin, Ireland, August 1989.

[WSF89] D. Whitley, T. Starkweather, and D. Fuquay. "Scheduling problems and traveling salemen: The genetic edge recombination operator". In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[WSS90] D. Whitley, T. Starkweather, and D. Shaner. "The traveling saleman and sequence scheduling quality solution using genetic edge recombination". In L. Davis, editor, *The Genetic Algorithms Handbook*. 1990.