(David)

The flexible job shop scheduling algorithm is an extension of the job shop scheduling problem: an NP-hard algorithm that attempts to assign jobs to resources in the most optimized fashion.

The traditional algorithm takes n jobs and m machines and finds the most optimal way to run all the jobs through any one individual machine for every machine. For example, the algorithm may find that the most optimal way to run three jobs through machine 1 is to first run job 1, then job 3, then job 2.

However, Machine 2 would run the same jobs most optimally by first running job 2, then job 1, then job 3. But this type of problem is not very representative of a real-world situation. Any standard shop such as a manufacturing plant or a kitchen will have machines that cannot do some of the jobs. Also one would want to run multiple jobs or tasks on multiple machines in parallel.

The flexible job shop scheduling algorithm takes several jobs that can be completed by one or more machines. It must determine what the most optimal way to choose a task for a machine and sequence is such that the cost of running the entire process is minimized.

We go into two extensions relevant to the algorithm. First, this algorithm uses evolutionary approximation methods to approximate the answer. Rather than explain how Bayesian and Evolutionary optimization works, there will be a very fast and condensed example of how the algorithm would run the task of optimizing making a Thanksgiving meal. Secondly, we extend the problem by showing a critical flaw that was found in the published paper. The goal of optimization through approximation was not achieved.

(Mitch)

Similar to the flexible Job-Shop-Scheduling Problem, we have “machines” that can do multiple jobs. For example, a sous-chef can chop food, stir food, and roll out dough. Again like our job-shop scheduling problem, we have tasks that must be completed first. For example, onions must be chopped before they are added to the stuffing and cooked. We need to cook the potatoes before they are mashed. (Our example slides)

We begin our example of the Hybrid Evolutionary Algorithm by generating many different orderings: (orderings slide)

Randomly group orderings: (grouping slide)

Loop through the following steps:

Select the best individual in each group and each population (fitness/selection slide)

Update each individual by “pushing” them towards the best individual (pushing slide)

Select the best Bayesian Network Structure: (network slide)

Regroup orderings using Bayesian Network structure (grouping slide)

We were originally very excited by this algorithm because of the combination of Machine Learning techniques. However, we recently noticed an issue with this algorithm. The authors of our primary source were reporting that they improved the number of generations to convergence. However, most people report improving run-time in addition to or instead of reporting the number of generations. So we took another look at their results table. Here we have outlined the run-time for the Hybrid Evolutionary Algorithm in red and the run-time for a Particle Swarm Optimization algorithm in blue. You can see that the Hybrid Evolutionary Algorithm run-time is always bigger by almost an order of magnitude. (First Analysis and Critique Slide)

Here, we've taken their run-time and generation numbers and calculated a mean time per generation for each algorithm. This is only for the smallest problem size they reported, but you can see how much bigger the mean time per generation is for the Hybrid Evolutionary Algorithm. The improvement in generations was only by one to three hundred generations, and the huge difference in generation time means this algorithm shows an enormous decrease in performance over other established approaches. (Mean Time per Generation Slide)

This long generation time is probably due to the fact that the algorithm is dependent on learning a Bayesian Network. Learning a Bayesian has been shown to be NP-Complete. (Bayesian Network slide)

Here are some of the ways the authors could have shown this algorithm was worthwhile.

Prove their algorithm was faster than a specific solution

Prove their algorithm produced a better approximation than other evolutionary approaches

Prove their design had better concurrency and potential for parallelization

Prove an approximation lower-bound through Bayesian Network Selection reduction (Necessary Fixes slide)