A Multi-Objective Evolutionary Algorithm Approach for Optimizing Part Quality Aware Assembly Job Shop Scheduling Problems

Michael H. Prince[1,2], Kristian DeHaan[1,3], and Daniel R. Tauritz[1,4]

[1] BONSAI Lab, Auburn University, Auburn AL 36830, USA
[2] mhp0009@auburn.edu
[3] kzd0054@auburn.edu
[4] dtauritz@acm.org

Abstract. Motivated by a real-world application, we consider an Assembly Job Shop Scheduling Problem (AJSSP), with three objectives: product quality, product quantity, and first product lead time. Using real-world inspection data, we demonstrate the ability to model product quality transformations during assembly jobs via genetic programming by considering the quality attributes of subparts. We investigate integrating quality transformation models into an AJSSP. Through the use of the de facto standard multi-objective evolutionary algorithm, NSGA-II, and a novel genotype to handle the constraints, we describe an evolutionary approach to optimizing all stated objectives. This approach is empirically shown to outperform random search and hill climbing in both performance and usability metrics expected to be valuable to administrators involved in plant scheduling and operations.

Keywords: Assembly job shop scheduling· Evolutionary algorithm· Genetic programming· Manufacturing· Multi-objective evolutionary algorithm

1 Introduction

The real-world scenario motivating this research involves scheduling the production of complex, low-quantity, mechanical and electrical components with exceedingly strict quality requirements. These products are constructed at a trusted foundry and often involve time-intensive additive manufacturing techniques. As a consequence, when a product fails to meet said requirements, it often must be rebuilt. This incurs a relatively significant delay due to the low quantity of products produced. Often times, an initial sample of the product must be sent off for early inspection and minor design modifications. This scenario motivates the three primary objectives for this optimization problem:

- Increasing quality of parts produced.
- Reducing time required to produce a defined quantity of parts (makespan).
- Reducing the lead-time of the first produced part.

The first contribution of this work is the modeling of quality, and integration of quality based constraints in the Assembly Job Shop Scheduling Problem (AJSSP). In assembly jobs, we propose that the quality attributes of the produced part can be determined by a non-linear combination of the quality of its subparts. With real-world intermediate inspection data, we demonstrate that accurate quality models may be constructed. We describe the techniques used to produce the quality models, the systems required to integrate this new constraint into the AJSSP, and how this constraint may be utilized to produce optimization objectives relevant to our real-world scenario.

The second contribution of this work is the formulation of a multi-objective evolutionary algorithm (MOEA) approach to optimize the AJSSP under the new constraints and objectives, particularly in the genotype design and operation. The genotype utilizes koza-style genetic programming (GP) trees to select subparts for assembly and a directly evolved schedule to control the order in which parts are assembled. This algorithm is compared against baselines to empirically gage performance. Finally, other attributes of the produced solutions, such as fitness range and distribution, are compared.

2 Background

The AJSSP is a variant of the JSSP with an added job dependency constraint. In the traditional JSSP, a series of jobs must be assigned to a limited number of machines with the typical goal of reducing makespan. The AJSSP adds an extra constraint on top of the traditional JSSP: parts produced from the jobs must be assembled to create the final product. This constraint restricts the number of valid schedules, since assembly is impossible if the sub-parts have not been manufactured. An example of this is shown in Figure 1. However, like the JSSP, the AJSSP is still considered an NP-hard problem [15]. We find that the AJSSP closely matches our real-world scenario's constraints. Additionally, the AJSSP's consideration of intermediate products and inventory provides a natural integration point for the quality modeling.

2.1 JSSP and AJSSP Literature

The JSSP is a widely studied and relevant problem to the manufacturing industry and operations research. Within the field of evolutionary algorithms (EAs), as indicated by the following surveys, the JSSP has been extended many times to consider multiple objectives [10]. Similarly, many different constraints have been considered such as resource limitations and uncertain processing time [9]. While some works, such as Al-Hinai et al. [2] and Wang et al. [18], consider machine disruptions impacting scheduling, failed products due to quality thresholds is an area that has seen little coverage.

Presently, research into the AJSSP is significantly less common than research into the JSSP. However, some approaches have been attempted. Lu et

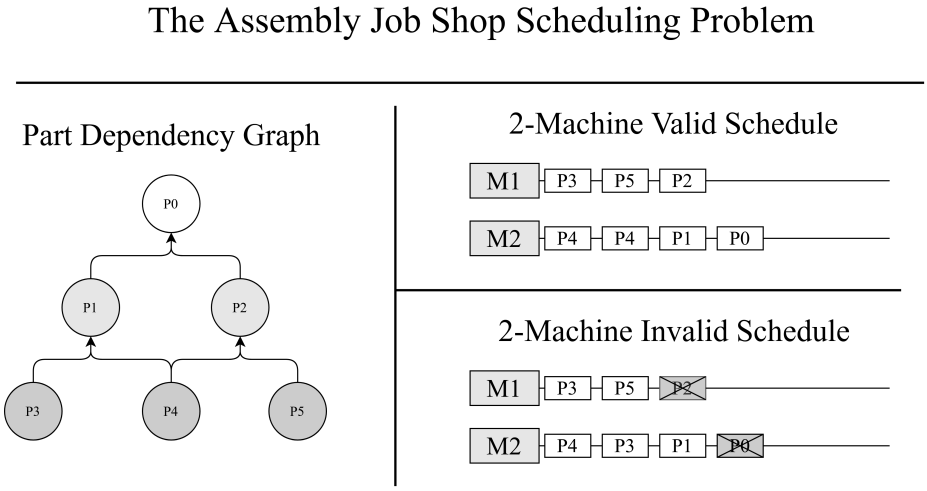

Fig. 1. A visualization of the AJSSP. In this part dependency graph, at a minimum, seven assembly steps must be taken to produce a single output product. Six of those manufacture unique parts, but P4 must be assembled twice for parts P1 and P2. In the invalid schedule, P4 is only assembled once and therefore P2 and P0 can't be produced.

al. explores the use of combining order review and release mechanisms with dispatching rules [12]. Thiagarajan et al. also considers the use of dispatch rules along with balancing multiple objectives via a weighted sum [17]. In the evolutionary space, Wong et al. compares genetic algorithm and particle swarm approaches using a chromosome to directly encode job sequencing [20]. Previous work from this group [3,19], considers different genetic algorithm techniques to solve the AJSSP and a resource constrained AJSSP respectively. Lv et al. combines the previously mentioned dispatch rules with Koza-style genetic programming (GP) [13]. Pareto-front multi-objective optimization appears to be particularly sparse in this space. In a recent work, Zhang et al. use ant colony optimization to simultaneously optimize makespan, tardiness, and workload [22].

2.2 Evolutionary Methods

Our problem necessitates two separate optimization methods: one to create the transformation models and one to optimize the scheduling and dynamic part selection. For creating part transformation models, GP was selected. GP has had success modeling non-linear scientific processes [4]. By evolving Koza-style trees, GP can create accurate and human-interpretable mathematical models. Traditional mathematical techniques struggle with non-linear and multivariate modeling since it is difficult to design model structure beforehand. Neural network techniques, while accurate, are exceedingly difficult to interpret once models have been trained [21]. For these reasons, GP carries the characteristics valued

by small-scale, high-consequence manufacturing, where not only accuracy, but interpretability, is desired.

The scheduling and part selection problem being considered motivates the use of a MOEA. Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [6] is one of the de facto MOEAs. It has seen much success in the manufacturing industry [1, 8]. When given a multi-objective optimization, the optimal solution can be represented as a set of points describing the tradeoff between objectives. This set is called the Pareto-optimal solution. EAs, due to their population based approach, are uniquely suited to exploring and discovering points along the Pareto-optimal solution. NSGA-II sets itself apart from other evolutionary methods through the use of non-dominated sorting combined with a clustering analysis during survival selection. In applications like the ones presented in this work, it may be beneficial for the end users to see the full range of options and tradeoffs when more than one objective is present.

3 Modeling of Quality Transformations

At small-scale, high-consequence production facilities, due to the quality requirements, subparts entering assembly and machining processes are often inspected before assembly begins. While the current inspection process weeds out defective parts at a single stage, we were curious if this system could be used to predict quality at a granular level, potentially identifying and preventing compounding errors as higher level assemblies are manufactured.

In order to construct a proof of concept, we obtained 70 instances of real-world product data, including the inspected attributes of all subparts and final assemblies. Initial analysis of the data showed that non-linear transformations were required in order to capture the relationships between the part and assembly attributes. The following sections detail the processes required to produce these models.

3.1 Data Sanitization

In some examples, input part data were missing. Data holes like these are common in real-world data. Normal data cleaning methods such as replacing gaps with the mean of existing data [16] were not used due to the small number of examples available. Instead, data gaps in input data were replaced with zeros. This allows the model's section pressure to favor features that are available in all examples, while retaining the ability to leverage the incomplete data.

3.2 Evolving Models

Due to the non-linearity of the data, and a desire to produce models with high readability, GP was selected to model the transformations. Koza style tree-based GP [14] is employed, with the tree output representing a single expected quality attribute. A standard set of binary functions and terminals, detailed in Table 1,

were used to evolve the models. Quality attributes are injected into the model via the attribute terminal.

Table 1. GP Primitives

Primitive Type	**Primitives**
Binary Functions	+, -, *, /
Terminals	Constant, Quality Attribute

The fitness of each genotype is calculated by comparing the model's outputs with the expected data on several real-world parts. A granular and fair accuracy calculation must be constructed to handle variations in part attribute values. Our accuracy calculation scales the accuracy relative to the attribute size. This equation is described in Eq. 1 with α representing the expected value and β representing the model's value. When determining the fitness of a model, accuracy is calculated for multiple parts. The smallest of these values is selected as the fitness. This method was chosen over mean or mode to prevent the optimization algorithm from ignoring outliers.

$$fitness = \log(\frac{1}{\frac{|\alpha-\beta|}{|\alpha|}}) \tag{1}$$

The evolution process is detailed in Algorithm 1. The initial population is generated using the ramped half and half method, a classic GP initialization technique [11]. Each solution is evaluated and assigned a fitness at creatio n time. Unlike Koza-style GP, the children are created through mutation *or* recombination. First, parents are chosen from the population. During recombination, two parents are chosen and a random node from each tree is swapped. During mutation, a random node is selected and regenerated from the single parent. A strategy parameter, $MutationRecombinationRatio$, controls the ratio of children generated by recombination and mutation. The children are merged with the population. Then, using a selection method, the surviving solutions are chosen and move on to the next generation. In order to support fair comparison against baselines, the algorithm is terminated after a set amount of fitness evaluations. Within a few rounds of hand-tuning, the algorithm was able to reach a sufficiently high fitness with a 95% accuracy compared to the ground truth. As such, no more tuning was required. All strategy parameters are listed in Table 2.

4 Integrating Quality into the AJSSP

The AJSSP is well suited for integration of the quality transformation models. Under the AJSSP, the series of assemblies naturally provides a scenario where

```
Result: Quality Tranformation Model
while Evaluations < Max Evaluations do
    Children = [];
    for i ← 0 to λ do
        if U(0..1) < Mutation Recombination Ratio then
            Parent1 = Parent Selection(Population);
            Children.append(Sub-TreeMutation(Parent1));
        else
            Parent1 = Parent Selection(Population);
            Parent2 = Parent Selection(Population);
            Children.append(Sub-Tree Recombination(Parent1, Parent2));
        end
    end
    Population = Population + Children;
    Population = Survival Selection(Population);
    Evaluations += λ;
end
```

Algorithm 1: The primary loop of the part modeling EA. It is assumed that $MaxEvaluations - \mu$ is evenly divisible by λ for clean termination. Of note, children are created by either recombination or mutation. In the case of mutation the second selected parent is unused.

Table 2. GP Configuration

Strategy Parameter	**Value**
Population Management	$\mu + \lambda$
μ	200
λ	50
Parent Selection	Fitness Proportionate Selection
Survival Selection	k-Tournament Selection (k=20)
Max Depth	10
Mutation And Recombination Operators	Koza-style
Mutation Recombination Ratio	0.2
Initialization	Ramped Half-and-Half (md=5, d=10)
Termination	Max Evaluations
Max Evaluations	10,000

compounding quality shifts can occur. The AJSSP framework also allows for the tracking of final product assembly time and quantity.

Rather than counting the number of parts available at a given timestep or using a validation method to check schedule correctness, we choose to perform something more akin to a factory floor simulation. Each part produced has its quality calculated on the fly with the transformation model. For this AJSSP, all machines producing the same assembly share the same quality transformation model. If a part has multiple quality attributes, multiple models are used to produced the qualities of the assembly, one for each attribute. After a part is produced, it is placed in a "bin" with other parts of the same type. When an assembly job is run, the sub-parts required for the assembly are removed from the bins, and their attributes are fed into the assembly's quality transformation model(s). This system of bins and part tracking allows for part quality to be modeled and recorded throughout the problem, from the lowest level sub-assemblies to the final product.

At the lowest level, base parts must exist to create the initial assemblies. At the modeled facility, many of the lowest level parts are ordered ahead in bulk or are stockpiled on site. To mirror this situation, at the beginning of the AJSSP, a large amount of "supplied parts" are placed in their respective bins. For the supplied parts, in order to maintain consistency, the number of supplied parts and the attributes of said parts are the same for all experiments. These parts were created with a Gaussian distribution of part qualities.

Some more common AJSSP constraints were added to more closely fit the real-world scenario. First, we chose the traditional JSSP over the flexible JSSP: a machine may only run a certain subset of assembly jobs. Second, not all assembly operations take the same amount of time. Finally, it should be noted that the AJSSP simulation contains no stochastic elements. The supplied parts are identical and the machines and quality models are entirely deterministic.

4.1 Objective Calculations

The simulation environment produces three different fitnesses, one for each objective. For simplicity and visualization, each of the fitnesses are designed to be positive and ranked in increasing order. The simulation terminates if the schedule has produced a set amount of deliverable products above a specified quality threshold, emulating a typical single order of products. If a final product fails to meet the quality threshold, it is discarded. A max time is imposed to save computational resources.

The first objective, the makespan, is calculated by taking the difference between the max allowed time, and time when the schedule has produced the required quantity of parts. If the schedule fails to produce the required quantity of parts in the max time, it is assigned a fitness of zero. The next fitness monitors the first part lead time. In order to rank this fitness in increasing order, this is calculated by taking the difference between the max time and the first deliverable product that meets requirements. The last fitness measures part quality. The quality metric of the passing final products are averaged together to

form this fitness. Naturally, as a byproduct of these objectives, failed deliverable products are minimized. Creating a failed part wastes machine time that could have been used to create a part that meets requirements.

5 MOEA Solution Method

The stated problem and objectives can be broken down into two primary actions: scheduling jobs on machines, and selecting subparts to use in part creation. Since these problems are so closely related, we choose to bundle both solutions into a single genotype with two distinct components. A visualization of this genotype, as well as a single assembly operation is shown in Figure 2. The first is a schedule which describes the order and machines jobs are run on. The second component, called the part rankers, is a series of GP trees which are used to select the parts used in the jobs. With the same functions and primitives as the quality models, listed in Table 1, the GP tree uses the part attributes to generate a value. The bins are sorted using the trees' calculated values. When a part is built, the subparts with the highest values are consumed.

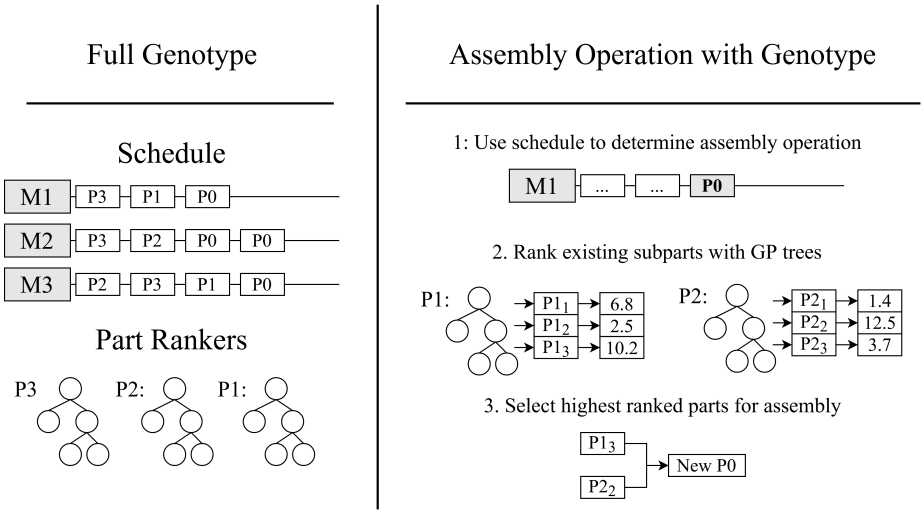

Fig. 2. A visualization of the genotype and the operation of a single assembly step. In practice, the ranking of the subparts is calculated and stored at the subparts' assembly time to prevent unnecessary re-ranking, but the displayed operation produces the same logical outcome.

5.1 Initialization

For the part rankers, unlike the modeling process, initialization is performed completely at random. At each node above the maximum depth, a random leaf

or branching node is chosen. At one less than the max depth of the tree, a random leaf node is forced. Schedule initialization is a bit more complex due to the job dependencies. In order to randomly initialize a valid schedule, at every decision point, when a machine is not running an assembly, a random assembly is selected from the set of valid assemblies. The valid set of assemblies is determined from the state of the part bins.

5.2 Mutation and Recombination

The two different components of the genotype must be mutated in separate ways. For the GP trees, a random set of trees are mutated, controlled by the $TreeMutationRate$ parameter. For each tree that undergoes mutation, a random node is selected and regenerated using the initialization technique. The schedule is mutated by selecting a random decision point and regenerating the schedule through the initialization technique. The $ScheduleMutationRate$ parameter determines the likelihood that the schedule will undergo mutation if the genotype is selected for mutation.

Likewise, different techniques must be used to perform recombination for the two different components. For the GP trees, in recombination, controlled by $TreeSwappingRecombinationRatio$, either the whole tree can be swapped, or a random node of one tree may be swapped with a node of another tree. Similar to mutation, a random subset of trees is selected for recombination via the $TreeRecombinatoinRate$.

Crossover of the schedule always occurs during recombination. For schedule recombination, a random time is selected as the crossover point. The first schedule is copied up to that point. From then on, the second schedule is used as a guide to complete the new schedule. If the next decision of the guide schedule is possible, then it is added. However, if it is not, due to subpart dependencies, a random decision is selected from the set of available decisions, similar to initialization.

5.3 Evolutionary Process

These genotypes are systematically evolved in order to search for new solutions. First, an initial population of genotypes is initialized and ranked based on Pareto optimality as in NSGA-II. The lowest set split during truncation is first sorted via NSGA-II fitness clustering, and the least clustered members are selected for survival. The highest ranking members of the population are selected into a parent set for mutation and recombination. From this set, λ child genotypes are created. Similar to the modeling evolutionary process, $MutationRecombinationRatio$ determines the ratio of children created through recombination or mutation. These children are added to the population and all members are re-ranked. Finally, the population is culled via truncation. From there the next set of parents are selected, and the process repeats.

All of the MOEA configuration parameters, including the parameters noted above in the mutation and recombination section, are listed in Table 3. The

Table 3. MOEA Configuration

Strategy Parameter	**Value**
μ	500
λ	500
Population Management	$\mu + \lambda$
Parent Selection	Fitness Proportionate Selection
Survival Selection	Truncation with NSGA-II Clustering
Mutation Recombination Ratio	0.24
Schedule Mutation Rate	0.88
Tree Mutation And Recombination Operators	Koza-style
Max Tree Depth	6
Tree Mutation Rate	0.93
Tree Swapping Recombination Ratio	0.26
Tree Recombination Rate	0.82
Tree Initialization	Random
Termination	Max Evaluations
Max Evaluations	50,000

parameters were optimized with random search. A set of parameters is run five times to capture consistency. The optimal config, observing both relative performance and consistency, was hand-picked. We can observe that, in general, the parameters favored high rates of change, picking above 0.8 for schedule, ranking mutation, and recombination rates. The algorithm, in general preferred recombination to mutation. When given the option to swap trees entirely between genotypes, the algorithm landed on traditional node-based crossover most of the time. Other than the high rate of change, the discovered parameters are fairly typical.

6 Scheduling Baselines

In order to evaluate our algorithm's performance, we constructed two baselines using the same genotype configuration as our EA. Random search was performed by repeatedly initializing the genotype structures. These solutions were ranked by Pareto non-domination. For the second baseline we implemented a form of hill climbing designed to operate in a multi-objective environment. The hill climber is seeded with a single initialization of the genotype. At each step, a new genotype is generated via the mutation operator, re-evaluated, and added to the population using the mutation method described in the above section. Afterwards, all genotypes are ranked via Pareto non-domination. Any members not in the Pareto frontier are removed from the population. In order to save on compute resources, the Pareto front is shuffled and truncated if it grows to more than 500 members. The next member to be mutated is selected from the remaining population.

7 Modeling Experiments and Results

Since both the modeling GP and baseline were run 30 times, they can be statistically compared with the student's t-test. K-fold cross validation, specifically 30-fold cross validation with a 20% / 80% test-train split, was used to prevent overfitting. First, by conducting a F-test, we determine that the variances are unequal between the two algorithms. Next, a two-tailed t-test was run assuming unequal variances. The t-test confers a confidence interval beyond 0.99 showing a statically significant difference in performance. With GP's higher mean of 95.06%, compared to the random search mean of 68.79%, in the test set, we can conclude that GP outperformed the baseline. Full details of the calculation can be found in Table 4. In the vast majority of instances, GP was able to find more accurate models than the random search baseline.

Table 4. Statistical Analysis Results

Test or Parameter	**GP**	**Random Search**
Samples	30	30
Mean Fitness	95.06%	68.78%
σ	1.4%	0.04%
T-Test, Unequal Variances, 2-Tailed	$1 - p = 4.7 \times 10^{-}8$	

8 Factory Simulation Experiments and Results

Due to the high computational cost, it was infeasible to compare the three algorithms run to convergence. Instead, in order to perform a fair empirical comparison, we opted to provide each of the algorithms the maximum feasible computational time: 50,000 evaluations. Each algorithm was run 30 times with different seeds to capture consistency and provide a more robust comparison. All algorithms were run with the same factory and assembly configuration. This configuration consists of a single deliverable part, D, which is constructed from two subparts: A and B. These subparts are constructed from supplied parts. A is constructed from supplied part SPA and B is constructed from supplied part SPB. The factory has three machines: $M1$, $M2$, and $M3$. $M1$ can produce parts D and A. $M2$ can produce A and B. $M3$ can produce D and B. For this proof of concept, and due to the nature of parts created at our facility, the part models used in our experiments were randomly generated.

8.1 Empirical Comparison

We first compare the performance of the three algorithms by observing Pareto-dominance between fronts. Each run was compared against another through a

series of binary comparisons. The comparisons are performed by combining the Pareto-fronts of two runs and re-sorting the genotypes. Since each algorithm was run 30 times, 900 sets of comparisons are performed for each permutation of the comparison. If only one run's solutions appear in the most dominant front, we declare that run to have dominated the other. If both runs' solutions appear in the most dominant front, then we declare the two runs to have tied. When the algorithms tie, we note the number of elements which contributed to the final best Pareto front. If more elements of that front were contributed by a single algorithm, the comparison is labeled a "greater tie". If both algorithms contributed equally to the final front, the result is labeled a "true tie". For each algorithm we performed several runs and compared each run individually. For example, if each algorithm had 30 runs, we recorded 900 different comparisons. The results of these comparisons can be seen in Table 5.

Table 5. Empirical Comparison of Algorithms

Alg A	**Alg B**	**A Doms**	**B Doms**	**A Greater Ties**	**B Greater Ties**	**True Ties**
MOEA	Random	710	0	190	0	0
MOEA	Hill	641	0	259	0	0
Hill	Random	190	61	186	423	40

From the results gathered shown in Table 5 we can see that a significant number of solutions tied, but MOEA performed demonstrably better than the baselines. Looking at the complete front dominations, the MOEA solver outperformed random search and hill climbing around roughly 75% of the time. In the case of a tie, the MOEA always contributed more members to the combined Pareto front. Comparing the two baselines, we see an interesting patten. While the hill climber is able to dominate random search more often, random search generally contributed more members to ties. We will explore this more when we discuss aspects of the solution quality. The distribution of fitnesses, shown in Figure 3 backs up the performance metrics. Observing the top row of plots, the best-performing fitnesses in a run, the MOEA was able to consistently produce solutions with better time savings and part qualities. All three algorithms were able to find the best possible lead time, which was a single optimal permutation of jobs to produce the first part.

8.2 Comparison of Solution Quality

With multiple objectives, more metrics and figures may be derived than front domination. To gather more empirical data about solution qualities, the size of the front and the range of solution fitnesses were chosen as metrics . The data is aggregated in Table 6. The MOEA method was, on average, able to produce solutions with five times the members of both random search and hill climbing.

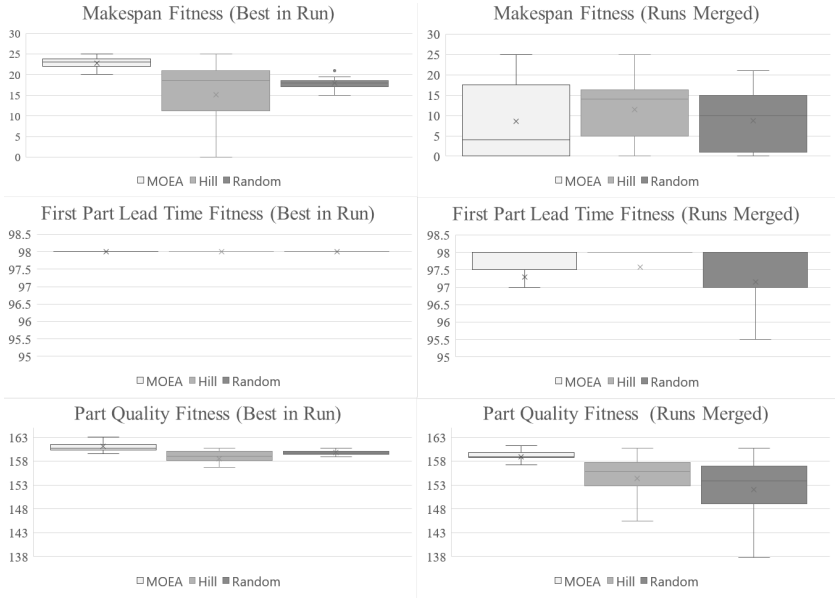

Fig. 3. The distributions of the fitnesses. The top series of charts display the distributions of the best fitnesses for each of the 30 runs. The bottom charts display the fitness distributions of all 30 runs' members merged.

Table 6. Comparison of Solution Quality

Algorithm	**Front Size**	**Makespan**	**Part Quality**	**First Part**
MOEA	49.6	18.4	5.98	4.5
Random Search	11.0	17.93	22.47	10.0
Hill Climbing	9.06	14.36	12.33	1.0

*Makespan, Part Quality, and First Part are the mean ranges of their respective fitnesses.

In this context, more solutions would provide greater decision options to the end users of this optimization. This may also explain the discrepancy between the best and merged fitness distributions of solutions produced by the MOEA. The MOEA was able to generate Pareto fronts of wider rangers, bringing down the averages of the merged solutions. The MOEA had the greatest range in time, but only by a small amount. Random search showed significantly more range in part quality and first part lead time. Considering random search's raw performance, we suspect this is caused by the poorer quality of solutions produced by random search. As solutions approach their maximums, observed in the best in run lead times, the general range of non-dominated solutions tends to close.

9 Conclusion

Inspired by a real-world scenario, we have introduced a new method of modeling quality through the use of intermediate inspection data. Using GP, real-world assembly quality attributes may be reliably and accurately predicted. Confirmed by statistical analysis, GP outperforms random search when generating these models.

With these models, we presented a method of integrating quality tracking into the AJSSP problem. With granular quality tracking, we can observe the effect of compounding errors during the manufacturing process. Additionally, this quality modeling enables novel objectives in the AJSSP. We can simulate the effect of final product requirements affecting manufacturing time. We also gain the objective of maximizing general product quality.

With the new AJSSP formulation, we developed a MOEA method to optimize this environment using a combined genotype of GP to manage sub-assemblies and a direct representation of a schedule to control job sequence. Through the use of NSGA-II, all objectives of the environment can be simultaneously optimized to produce a Pareto-front of non-dominant solutions. The MOEA method was compared to both random search and hill climbing. This method was shown to outperform both baselines in raw performance and usability metrics.

10 Future Work

This work may be extended in a variety of directions. The AJSSP may be extended to delivering multiple product orders simultaneously. Each time a new product is added, the system must consider three additional fitnesses. Early experiments with multiple products shows that NSGA-II is not able to optimize well due to the abundance of non-dominant solutions. Variations and modifications on NSGA-II [7] have been shown to be effective at solving many-objective problems, especially the non-dominant solution issue. Another alternative may be to swap out the NSGA-II optimizer with the NSGA-III optimizer [5], specifically designed to handle such issues.

While this work primarily operates in the empirical domain, further work could be done to create a more rigorous mathematical analysis of the quality-aware AJSSP problem and optimization method. Such a model may provide critical insights into provably optimal solutions and novel solution methods.

This algorithm can be used to analyze how adding or removing machines or suppliers affects the various objectives, as well as the tradeoff between objectives. While the current work may help with short to medium term planning through scheduling, observing how the optimization changes under factory modifications may help inform medium to long term decisions such as purchasing equipment, hiring operators, or expanding floor space. Taking this concept into the adversarial space, this manipulation of factory structure may be linked to another EA forming a competitive co-evolution problem. One agent is trying to maximize factory objectives while the other is attempting to minimize said objectives by sabotaging select parts, machines, or other factory processes. Through this proposed system, critical manufacturing components may be identified.

11 Acknowledgements

This work is funded by the Department of Energy's Kansas City National Security Campus, operated by Honeywell Federal Manufacturing & Technologies, LLC, under contract number DE-NA0002839.

References

1. Ahmadi, E., Zandieh, M., Farrokh, M., Emami, S.M.: A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms. Computers & Operations Research **73**, 56–66 (2016). https://doi.org/10.1016/j.cor.2016.03.009
2. Al-Hinai, N., ElMekkawy, T.Y.: Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. International Journal of Production Economics **132**(2), 279–291 (2011). https://doi.org/10.1016/j.ijpe.2011.04.020
3. Chan, F.T., Wong, T., Chan, L.: A genetic algorithm-based approach to job shop scheduling problem with assembly stage. In: 2008 IEEE International Conference on Industrial Engineering and Engineering Management. pp. 331–335. IEEE (2008). https://doi.org/10.1109/IEEM.2008.4737885
4. Dabhi, V.K., Chaudhary, S.: Empirical modeling using genetic programming: A survey of issues and approaches. Natural Computing **14**(2), 303–330 (2015). https://doi.org/10.1007/s11047-014-9416-y
5. Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. IEEE transactions on evolutionary computation **18**(4), 577–601 (2013). https://doi.org/10.1109/TEVC.2013.2281535
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017

7. Elarbi, M., Bechikh, S., Gupta, A., Said, L.B., Ong, Y.S.: A New Decomposition-based NSGA-II for Many-Objective Optimization. IEEE transactions on systems, man, and cybernetics: systems **48**(7), 1191–1210 (2017). https://doi.org/10.1109/TSMC.2017.2654301
8. Frutos, M., Olivera, A.C., Tohmé, F.: A memetic algorithm based on a nsgaii scheme for the flexible job-shop scheduling problem. Annals of Operations Research **181**(1), 745–765 (2010). https://doi.org/10.1007/s10479-010-0751-9
9. Gao, K., Cao, Z., Zhang, L., Chen, Z., Han, Y., Pan, Q.: A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. IEEE/CAA Journal of Automatica Sinica **6**(4), 904–916 (2019). https://doi.org/10.1109/JAS.2019.1911540
10. Gen, M., Lin, L.: Multiobjective evolutionary algorithm for manufacturing scheduling problems: state-of-the-art survey. Journal of Intelligent Manufacturing **25**(5), 849–866 (2014). https://doi.org/10.1007/s10845-013-0804-4
11. Koza, J.R., Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, vol. 1. MIT press (1992)
12. Lu, H., Huang, G.Q., Yang, H.: Integrating order review/release and dispatching rules for assembly job shop scheduling using a simulation approach. International Journal of Production Research **49**(3), 647–669 (2011). https://doi.org/10.1080/00207540903524490
13. Lv, H., Han, G.: Research of assembly job shop scheduling problem based on modified genetic programming. In: 2017 10th International Symposium on Computational Intelligence and Design (ISCID). vol. 2, pp. 147–151. IEEE (2017). https://doi.org/10.1109/ISCID.2017.120
14. Poli, R., Langdon, W.B., McPhee, N.F., Koza, J.R.: A Field Guide to Genetic Programming. Lulu. com (2008)
15. Potts, C.N., Sevast'Janov, S., Strusevich, V.A., Van Wassenhove, L.N., Zwaneveld, C.M.: The two-stage assembly scheduling problem: Complexity and approximation. Operations Research **43**(2), 346–355 (1995). https://doi.org/10.1287/opre.43.2.346
16. Rahm, E., Do, H.H.: Data Cleaning: Problems and Current Approaches. IEEE Data Eng. Bull. **23**(4), 3–13 (2000)
17. Thiagarajan, S., Rajendran, C.: Scheduling in dynamic assembly job-shops to minimize the sum of weighted earliness, weighted tardiness and weighted flowtime of jobs. Computers & Industrial Engineering **49**(4), 463–503 (2005). https://doi.org/10.1016/j.cie.2005.06.005
18. Wang, Y.M., Yin, H.L., Da Qin, K.: A novel genetic algorithm for flexible job shop scheduling problems with machine disruptions. The International Journal of Advanced Manufacturing Technology **68**(5-8), 1317–1326 (2013). https://doi.org/10.1007/s00170-013-4923-z
19. Wong, T.C., Chan, F.T., Chan, L.: A resource-constrained assembly job shop scheduling problem with lot streaming technique. Computers & Industrial Engineering **57**(3), 983–995 (2009). https://doi.org/10.1016/j.cie.2009.04.002
20. Wong, T.C., Ngan, S.C.: A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop. Applied Soft Computing **13**(3), 1391–1399 (2013). https://doi.org/10.1016/j.asoc.2012.04.007
21. Zhang, Q.s., Zhu, S.C.: Visual interpretability for deep learning: a survey. Frontiers of Information Technology & Electronic Engineering **19**(1), 27–39 (2018)
22. Zhang, S., Li, X., Zhang, B., Wang, S.: Multi-objective optimisation in flexible assembly job shop scheduling using a distributed ant colony system. European Journal of Operational Research **283**(2), 441–460 (2020). https://doi.org/10.1016/j.ejor.2019.11.016