4-2020

Efficient Algorithms for Unrelated Parallel Machine Scheduling Considering Time of Use Pricing and Demand Charges

Brady Collea
byc1625@rit.edu

Rochester Institute of Technology

Department of Industrial and Systems Engineering

Kate Gleason College of Engineering

Master of Science in Industrial and Systems Engineering

Thesis

Efficient Algorithms for Unrelated Parallel Machine Scheduling Considering Time of Use Pricing and Demand Charges

Brady Collea

April 2020

Committee Members:

Katie McConky, PhD – Industrial and Systems Engineering, RIT
Dr. Yunbo Zhang, PhD – Industrial and Systems Engineering, RIT

Acknowledgements

I would like to thank Dr. Katie McConky, my thesis advisor, for her endless guidance, support, and feedback over the last two years. Simply put, this work could not have been completed without her.

I would also like to thank the RIT Society of Software Engineers for welcoming me into their lab space and providing knowledge about programming topics beyond my depth which helped me complete this work.

My thanks go out to the RIT Research Computing lab as well. Although I was unable to include those results in this work, the assistance of the lab staff when I was working on the server was friendly, knowledgeable, and showed that the staff highly valued my student experience.

Thank you to my parents for providing me the opportunity to pursue this degree and for the countless sacrifices I do not even know about that they have made for me.

Lastly, I would like to thank my girlfriend, Lilly Bokun, for being there to support me for my entire college career. She encouraged me to take on this work and provided the academic and emotional support to see me through it.

Abstract

There is an ever-increasing focus on sustainability and energy consumption worldwide. Manufacturing is one of the major areas where energy reduction is not only environmentally beneficial, but also incredibly financially beneficial. These industrial consumers pay for their electricity according to prices that fluctuate throughout the day. These price fluctuations are in place to shift consumption away from "peak" times, when electricity is in the highest demand. In addition to this consumption cost, industrial consumers are charged according to their highest level of demand in a given window of time in the form of demand charges. This paper presents multiple solution methods to solve a parallel machine shop scheduling problem to minimize the total energy cost of the production schedule under Time of Use (TOU) and demand charge pricing. The greedy heuristic and genetic algorithm developed are designed to provide efficient solutions to this problem. The results of these methods are compared to a previously developed integer program (IP) solved using CPLEX with respect to the quality of the solution and the computational time required to solve it. Findings of these tests show that the greedy heuristic handles the test problems with only a small optimality gap to the genetic algorithm and optimal IP solution. The largest test problems could not be solved by the genetic algorithm in the provided time period due to difficulty generating an initial solution population. However, when successful the genetic algorithm performed comparably to the CPLEX solver in terms solution quality yet provided faster solve times.

Table of Contents

List of Tables

List of Figures

1. Introduction

A changing society and climate have led to a substantial shift towards focusing on sustainability in manufacturing. Both utility providers and users have been searching for the most effective ways to conserve energy in a cost effective manner. This combined effort has led to the use of time based energy use pricing policies such as time of use (TOU) and real time pricing (RTP) (U.S. Department of Energy, 2019). These different policies encourage users to shift their electricity usage away from 'peak' hours when the demand for electricity is at its highest. TOU pricing does this by assigning specific hours, usually during the mid-day and afternoon, as peak periods where the cost of electricity is higher than usual. An important distinction between TOU and RTP is that TOU prices are predetermined and do not vary from day to day (California Public Utilities Commission, 2019). By comparison, RTP changes every hour in reaction to the prices of electricity on the wholesale market. RTP also can be implemented through two different program types (Nezamoddini and Wang, 2017). A one-part program applies hourly prices to the consumer's entire electricity consumption, while a two-part program only applies hourly charges if the consumer deviates from their historical usage pattern. While this fluid response is a promising concept, Nezamoddini and Wang (2017) showed that the savings when using RTP are dependent on which program is used and how it is implemented.

In addition to these varying price consumption schemes, users with high amounts of energy consumption are also charged for their maximum demand in a given time period. This is known as a demand charge (Dieziger, 2000). Demand charges are independent of the time spent consuming the peak demand. If a large amount of energy is consumed for only 15 minutes, the demand charge is for that 15 minute period. This ends up causing demand charges to account for a large percentage of the total energy cost in a manufacturing process. Therefore, it is imperative to consider this demand charge when scheduling production.

Addressing scheduling problems has always required innovative methods to obtain solutions in a reasonable amount of time. Batista Abikarram et al. (2019) presented an integer program (IP) which addressed the scheduling problem with demand charges. However, the IP struggled to scale up and solve large test instances. An IP can become very slow when working with larger problems due to the complexity and exponential increase in problem size when increasing the number of machines or jobs. For example, consider a problem where a traditional 8

hour workday is broken into 15 minute periods, resulting in 32 periods where a job could be placed. Additionally, consider that the problem is attempting to schedule 2 jobs that each take one period, on only two machines. In this tiny problem, both jobs could be on either machine 1 or machine 2, with 64 choices for job 1 and 63 for job 2. That would create 2*64*63 possible solutions. Altogether, this problem has 8064 possible solutions. It's very easy to see that with hundreds of jobs and tens of machines, this scheduling problem becomes enormous and very taxing to solve.

Several solution approaches have looked towards nature for inspiration to solve these giant problems where IPs struggle. These solutions can be classified as evolutionary algorithms (EAs) (Corne & Lones, 2018). EAs are very flexible and are not problem specific. This allows them to be applied to many different problems in different areas of research. One of the most well-known types of evolutionary algorithms is a genetic algorithm (GA). A GA uses crossover and mutation, along with natural selection to bring a random population of solutions towards the optimal solution. Another creative approach is known as ant colony optimization (Dorigo & Di Caro, 1999). As its name suggests, ACO is modeled after real life ants, which work together towards the optimal solution, leading other ants towards promising areas of the solution space and away from those that have been previously explored and proven to be low quality. Lastly, Particle Swarm Optimization (PSO) draws inspiration from the flocking behavior of fish or birds in the wild (Kennedy & Eberhart, 1995). Each member of the population keeps track of what the best member's solution is in addition to its own to move the population towards the best solution. These different methods have all been used effectively to address scheduling problems which are similar to the one presented in this thesis.

The rest of this work is formatted in the following way. First, a brief problem statement is given explaining the difficulty of solving this problem, and what stands to be gained from successfully doing so. Afterwards, an extensive literature review of energy aware systems and their solutions is given. This is followed by the methodology and results of a preliminary version of the GA and heuristic. Afterwards, methodology regarding improvements to the GA and another round of testing results are shown. Lastly, conclusions and future work are discussed.

2. Problem statement

The inclusion of TOU pricing and demand charges makes creating schedules which minimize energy cost very difficult. It is nearly impossible to reliably assign, sequence, and place

jobs in a cost effective manner without some sort of structured solution method. Providing efficient and scalable solutions is even more challenging. Currently, little research exists addressing peak demand and factoring demand charges into the total energy cost. As a result, industrial consumers who must pay demand charges can be at a disadvantage.

This research focuses on a parallel machine system that includes both TOU pricing and demand charges. It has been proven that the parallel machine problem with TOU pricing is NP-hard (Ding et al., 2016), so it can be assumed that the inclusion of demand charges creates a problem at least as NP-hard. Due to the hardness of the problem, both a heuristic and a genetic algorithm are developed to provide solutions. These methods should provide optimal or near optimal solutions in a much shorter time span than a traditional integer program. In addition, both solution methods should be more scalable than the integer program developed in Batista Abikarram et al. (2019).

Successfully completing this research may be able to provide industrial consumers that pay demand charges with a tool to create energy aware and cost effective schedules quickly. This tool could be used in a day ahead fashion to craft schedules for future production runs. If the TOU pricing is known beyond one day, then this research may be used to provide longer term production schedules as a result of the scalability of the heuristic and genetic algorithm. Lastly, successfully completing this research may provide a base for other research considering demand charges in alternate shop systems.

3. Literature review

Scheduling problems considering energy demand have been approached in several different ways over roughly the past decade. These problems vary in three main aspects; problem setup, problem objective, and solution method. The major distinction in problem setups are whether the system consists of a single machine, parallel machines, or some other setup such as a job shop or flow shop. Regarding the problem objective, some problems consider only energy consumption, while others take the associated energy cost into account as well. Very few problems consider only energy cost or consumption though. Instead, they present some bi-objective combining makespan or tardiness with energy cost or consumption. Different pricing schemes exist within the problems that consider cost as well, such as time of use or real time pricing. Lastly,

solution methods vary greatly from integer programs to heuristics to metaheuristics. All of these methods vary in their solution quality, efficiency, and scalability.

In this section, relevant literature examining the various manufacturing situations above will be introduced. First, cases which worked with a single machine will be reviewed. Second, literature considering parallel machine systems aiming to minimize energy consumption will be introduced, followed by those which consider cost as well. In the next section, literature which does not consider a parallel machine environment will be discussed. Finally, special cases which handle demand leveling or peak demands will be introduced, including the mathematical model to which this thesis is most closely related. Additionally, a description of the differentiating factors between the reviewed texts and the problem this thesis will be addressing is provided.

3.1 Single machine energy aware systems

Although a single machine system was not considered in this thesis, the solution methods used can possibly be adapted and expanded to a parallel machine environment, and it was therefore important to review these cases. Che et al. (2016) worked with a system with single jobs that had to be completed before a shared due date. The objective was to minimize the energy cost of scheduling those jobs subject to TOU pricing. To solve this, an efficient greedy insertion heuristic was implemented. This heuristic attempted to schedule the jobs by attempting to insert the jobs with the highest energy demands into the lowest possible cost intervals. If the job could not fit into the idling time available, the scenarios where adjacent jobs were shifted to allow the new job to enter in the desired period were evaluated. If shifting proved to be less expensive, the job would insert. If shifting was not beneficial, the job would then be inserted into the next lowest cost interval. This approach allowed for 5000 jobs to be scheduled successfully given an appropriately long planning horizon. Another heuristic was proposed to address a single machine system with batched jobs with the aim to minimize the total energy cost and the makespan. Wang et al. (2016) solved this problem by creating a two-stage heuristic. The first stage grouped the jobs into batches, while the second stage determined the processing order of the batches. The largest instance solved consisted of 6 different job types and three jobs of each type. These 18 jobs were batched and processed given a 100 time period window. A third solution to a single machine system was provided by Cheng et al. (2017). This system also dealt with batches with the same bi-objective as Wang et al. (2016) subject to TOU pricing. The machines in this system also consumed energy

when turning on, but not when shutting off which added a different element to the formulation. The heuristic in this paper simplified the bi-objective problem into a series of single objective optimization problems, which were then solved with preference given to the goal of minimizing the total energy cost. Cheng et al. (2017) found that their solution could solve up to 5000 batches in a reasonable amount of time and was much more computationally efficient than the CPLEX solver.

A genetic algorithm was another approach taken to solve a single machine problem (Shrouf et al., 2014). The system in this paper allowed for machines to turn on, off, or idle, with each operation requiring a certain amount of time. The objective was solely to minimize the energy cost of the schedule, again working with time of use pricing. A crossover rate of 60% and mutation rate of 15% were chosen, as well as the use of a 30% elitism rate in this genetic algorithm. This algorithm solved a problem with 60 jobs fitting into a 135 period schedule, and did so with a computational time of only 12 seconds. A different genetic algorithm was proposed by Yildirim and Mouzon (2012). They attempted to minimize makespan and energy consumption in a single machine environment. Since only consumption was considered, no pricing scheme was provided about electricity cost. Their multi-objective genetic algorithm was tasked with deciding when to start jobs and when to power up or power down the machine. They tested different parameter sets of the algorithm and were able to obtain a solution for a 50 job schedule. Yildirim examined a very similar problem again in 2019 (Rubaiee & Yildirim, 2019). This time, the objective was to minimize makespan and energy cost, incorporating time of use pricing into the formulation. It is also important to note that job preemption was allowed in this system. An ant colony optimization algorithm was implemented in which the artificial ants decided when the machine should be processing and when it should be off. The ants did not determine which jobs went into which time interval, since preemption allowed for jobs to be broken up within the designated process times. This approach was able to scale to solve a problem with 100 jobs in the system.

3.2 Parallel machine systems considering energy consumption

Expanding to a parallel machine system, one solution suggested the use of a two stage solution (Angel et al., 2012). The machines in the system were unrelated and had different processing capabilities, as well as different options for processing speed. The jobs in the problem were single jobs and, since the objective was to minimize makespan and energy consumption, had

no due dates. The solution detailed had a linear program provide the optimal machine for each job to be processed on. That solution was then passed into an algorithm which sequenced the jobs as soon as possible. Although no test instances are given, Angel et al. (2012) detail theorems for proof of concept.

Several evolutionary algorithms were reviewed which attempted to solve energy consumption problems. One well known genetic algorithm, NSGA-II, was applied to an identical parallel machine scheduling problem by Wang et al. (2018). This genetic algorithm utilized three different mutation operators, which each served to diversify the population in different ways to avoid local optima. Solutions using the genetic algorithm were compared to two different heuristic approaches in terms of solution quality. The largest test instances showed that NSGA-II got closer to the optimal/near optimal Pareto fronts 100% of the time. Agrawal and Rao (2014) used three different methods to solve a problem with single jobs that all had their own due dates. The objective was only to minimize energy consumption. A standard genetic algorithm was one of the methods mentioned. A second solution method was cellular automata in tandem with a genetic algorithm (CA + GA). This created initially good schedules by selecting nodes from the program graphs that were used to represent the machine system. These solutions were evolved according to elite rules to get to the best solution. The elite rules themselves were found through a genetic algorithm which selected the best rules to use. In addition to these two algorithms, a heuristic was developed for comparison purposes. Agrawal and Rao (2014) determined that the heuristic provided the quickest solutions, but were not optimal, while the genetic algorithm provided strong but not always optimal solutions if given enough computational resources. The CA + GA provided the best schedules for the given systems, but was the most computationally intense.

Another set of three genetic algorithms was provided by Hong & Fei (2017). There was a basic genetic algorithm, with crossover and mutation. The Adaptive GA had a mutation rate which changed depending on the quality of the candidate solution. Lastly, the improved GA incorporated altered crossover and altered mutation operations. A comparison of the three was provided regarding solution quality, with the improved GA yielding the best solution for a problem with 3 machines and 200 jobs. The system in Hong & Fei (2017) was slightly different. Jobs were two stages instead of one and had no due dates. The machines in the problem also had capacities of 5 jobs at a time instead of one.

A different kind of evolutionary algorithm employed in a parallel machine process was particle swarm optimization (PSO) (Fang & Lin, 2013). In this case, jobs could be processed at different speeds. The objective was to minimize tardiness and energy consumption. If speeds were higher, time could be conserved at the expense of higher energy consumption rates. The PSO algorithm was able to provide a solution for a 5 machine environment with 50 jobs. The ant colony optimization mentioned previously was also applied to a parallel machine problem (Liang et al., 2015). Minimizing the tardiness and energy consumption were the two objectives in this paper. The machines in the system required time and power during setup, which is also a requirement for the problem in this thesis. The machines could also idle and turn off. The decision of when to idle versus when to turn off was made by calculating a breakeven duration for the machines. If the time in between jobs exceeded this duration, the machine would turn off, otherwise it would idle for the interval between jobs. The artificial ants picked which machine to assign a job to, and then selected the job for that machine which would minimize tardiness. After all jobs were assigned, the ants re-evaluated their decisions to see if any improvements could be made. If necessary, jobs could be swapped between machines or two jobs could switch start points on the same machine. The largest test instance this ACO solved was a 10 machine and 100 job problem.

3.3 Parallel machine systems considering energy cost

There have been several solutions proposed to minimize energy cost in parallel machine systems. This objective aligns more directly with the work in this thesis as opposed to energy consumption. However, the literature in this section does not consider a demand charge in the energy cost calculation, nor any objective related to demand leveling.

Che et al. (2017) proposed a mixed integer linear program (MILP) to solve small instances, as well as a two stage heuristic to handle larger problems. The first stage of the heuristic assigned the jobs onto different machines to minimize the energy consumption required to complete the jobs while allowing preemption. Afterwards, the jobs were adjusted accordingly to no longer allow preemption while ordering the jobs. The performance of this heuristic was compared to the CPLEX solver with excellent results. The largest test instance solved was 20 machines with 200 jobs and was completed in under a second.

Multi-stage processes were a common occurrence across the reviewed literature, suggesting that attempting to solve a parallel machine problem in one step leads to

overcomplication. Van Den Dooren et al. (2017) used a window occupation method to assign jobs onto the machines combined with a late acceptance hill climbing algorithm (LAHC) to solve the parallel machine scheduling problem. The problem Van Den Dooren et al. (2017) examined allowed machines to turn on, off, or idle, with all of those operations requiring energy. The price of the electricity was also figured using time of use pricing. The window occupation method analyzed each job's due date minus release date to get the available window that job could be scheduled in. Next, the percentage the job's required processing time occupied of that window set the priority of jobs for assignment. The LAHC kept track of the most recent solutions in a list, which was then updated using the current solution and those in the immediate neighborhood. The use of the list in the LAHC allowed the algorithm to escape local optima. This approach successfully solved the largest test instance provided of 100 machines and 5000 jobs. Ding et al. (2016) examined an unrelated parallel machine system with different processing capabilities and attempted to minimize the energy cost of scheduling jobs in a time of use setting. The jobs in the system were independent, but shared a due date set by a makespan constraint. Ding et al. (2016) proposed a solution consisting of a column generation heuristic embedded within a two-loop solving method. This method was tested on several different instances of different sizes to analyze the scalability of the solution method, and successfully solved a problem of 20 machines with 200 jobs. A wider comparison of heuristic approaches was carried out by Li et al. (2016) which compared ten different heuristics and solutions. The heuristics were created with the bi-objective of minimizing tardiness and energy cost of individual jobs. The machines in the system had three possible states; turning on and waiting to process, idling, and processing. The turn on/warm up time also consumed energy. After examining these different methods, the best heuristic was able to solve a problem consisting of 12 machines and 200 jobs.

Evolutionary algorithms have also been used to address cost minimization for parallel machine problems. Zhou et al. (2018) worked with unrelated parallel machines to minimize both makespan and energy cost. The machines had different processing rates for the batches of jobs in the system. The cost of scheduling these batches was determined by a TOU pricing scheme. Zhou et al. (2018) developed a differential evolutionary algorithm (MODDE) to solve their problem. The differential evolutionary algorithm still used crossover and mutation operators to conduct the guided search like traditional genetic algorithms. A heuristic based on the shortest processing time principle was used to initially assign batches to the different machines. This MODDE was

compared to two other well known GAs in terms of computational time and the Pareto fronts that were generated. Moon et al. (2013) also addressed the energy cost minimization problem using a genetic algorithm. Moon et al. (2013) worked within a system where the unrelated machines had different processing capabilities, but did not mention any machine states such as turning on or off. The objective of the genetic algorithm was to simultaneously minimize makespan and the TOU energy cost of the schedule. Moon et al. (2013) developed a hybrid genetic algorithm and used a blank job insertion algorithm to accomplish this objective. Instead of idling being designated as its own state, that time was represented using blank jobs in the solution. The genetic algorithm used in this paper did not include killing off members of the population, but rather replaced the weakest members with the offspring created in the previous generation. This ensured that the quality of the population increased every generation. Otherwise, killing and randomly refilling the population may have diluted the quality of the candidates. Moon et al. (2013) tested their solution to analyze how well it performed with different test instances. Twenty machines processing 65 jobs was the largest test instance that the genetic algorithm solved successfully.

3.4 Other energy aware shop systems

The following section contains a brief review of problems that dealt with machine shop environments other than parallel machines. Although these systems are very different than a parallel machine system, many of the same solution approaches have been used when attempting to achieve energy related objectives. The main purpose of reviewing these papers was to assess whether aspects of the solution methods could be adapted for this thesis.

One alternative shop setup is a flow shop, where a job must go through a series of operations to be completed. Fazli Khalaf and Wang (2018) analyzed a flow shop that considered not only electricity use provided by utilities, but also the use and storage of renewable energy to supplement the power needed to complete the jobs. To capture these added elements, Fazli Khalaf and Wang (2018) formulated the problem in two stages. The first stage created a schedule for the next day based on the day ahead forecast of energy demand and prices, and the second stage adjusted those schedules based on real time uncertainties in the pricing. This use of real time pricing was a distinguishing feature of this paper, since most shop environments considered only time of use pricing policies. The mathematical model developed to solve this problem was proven on case studies of various sizes up to 10 machines.

Genetic algorithms were not restricted to parallel machine systems, as Zhang and Chiong (2016) used one to solve an energy aware job shop scheduling problem. This problem considered a joint objective of minimizing the total tardiness of jobs in the system and the total energy consumption. No mention was made of cost or utility pricing scheme. Zhang and Chiong (2016) compared the solutions and efficiency of two genetic algorithms for this system. Their multi-objective hybrid genetic algorithm (MOHGA) included elitism in the process, as well as a mutation operator which acted on only a certain percent of the best candidates. The MOHGA effectively solved a system with 10 machines and 10 jobs, where each job had its own specific routing throughout the network of machines.

3.5 Problems considering peak demand load

While the energy aware parallel machine shop scheduling problem has been studied extensively, not much literature exists which considers peak demand. Minimizing peak demand adds another element to the objective of the problem. By requiring demand leveling across the periods of the day, the solution must attempt to manage the tradeoff between the hourly energy price in a TOU scheme with the demand charge.

Nattaf et al. (2015) considered an unrelated parallel machine system processing batches of jobs, and which also had scheduled maintenance. The proposed solution to this problem was a series of linear programs whose objective was to minimize both total energy consumption as well as consumption during peak periods of the day. Nattaf et al. (2015) did not consider energy pricing or demand charges, just the total energy used in the pre-defined set of peak periods during the day. Initially, jobs were batched and placed onto the least busy machine until all jobs were placed. Next, that initial setup was passed into a mixed integer linear program (MILP) which sequenced the jobs and placed them in a specific window without considering the peak demand. Lastly, the schedule produced by the first MILP was passed into another MILP which finalized the schedule, attempting to minimize the peak consumption.

Rager et al. (2015) worked within a parallel machine environment to level the demand across all periods of the day. This, in turn, would reduce a demand charge, but no demand charge was directly accounted for in the energy cost calculations. Additionally, there was no fluctuation in electricity price throughout the day in this problem. The jobs in this problem were all subject to the same due date at the end of the day and were independent of one another. Rager et al. (2015)

used multiple evolutionary algorithms to solve this problem and compared the solution quality of several small test instances to the Gurobi solver. Four machines were present in the largest test instance mentioned.

The work that is most closely related to this thesis was provided by Batista Abikarram et al. (2019). Within an identical parallel machine shop, single jobs must be processed by a shared due date. The identical machines could be in one of five states, the same states used in this thesis. Additionally, Batista Abikarram et al. (2019) considered a direct demand charge tacked onto the total energy cost based on the consumption. The energy pricing in this paper is also based on a TOU scheme. The objective of the problem was to minimize the total energy cost of the schedule, taking both the energy consumption cost and demand charge into consideration. To solve this problem, an integer program was proposed, which is included in its entirety below. The solution methods developed in this thesis are directly compared with this integer program. The largest test instance solved was either 16 machines with 4 jobs per machine or 5 machines with 15 jobs per machine. However, the first configuration took days to solve and a more reasonable problem size for this program was found to be 13 machines with up to 4 jobs on each machine. This 52 job configuration took 11.3 hours to solve.

Sets

N: Independent jobs in system

M: Set of machines

P: Periods of the schedule

S: set of possible states of each machine

TS: set of transitional states that require time to be completed (subset of S)

T_s: set of transitions allowed from each state s

Pr: set of states that process parts (subset of S)

Parameters

Pt_j: Processing time of job j

Ep_p: Energy price per kWp at period p $\frac{\$}{\text{kWp}}$ $(kWp = kilowatt\ period)$

Ec_s: Energy demand (kW) of any machine in state s

Tt_s: Time required to be spent in a transitional state s ∈ TS

De: Demand charge $\frac{\$}{\text{kW}}$

Rw: Number of periods for peak load rolling window

Variables

w_{isp} $\begin{cases}1, & \text{Machine i is in state s during period p} \\ 0, & \text{O.W.}\end{cases}$

x_{ijp} $\begin{cases}1, & \text{Machine i is processing job j during period p} \\ 0, & \text{O.W.}\end{cases}$

y_{ijp} $\begin{cases}1, & \text{Machine i begins processing job j during period p} \\ 0, & \text{O.W.}\end{cases}$

z_{isp} $\begin{cases}1, & \text{Machine i begins transition states s during period p} \\ 0, & \text{O.W.}\end{cases}$

d maximum average demand in a Rw period interval

Parallel Machine Demand and Consumption Charge Model

$$Minimize\ Cost = (De * d) + \sum_{i \in M} \sum_{s \in S} \sum_{p \in P} \left(w_{isp} Ec_s Ep_p\right) \quad (1)$$

Subject to:

$$\sum_{j \in N} x_{ijp} = \sum_{s \in Pr} w_{isp}\,, \quad \forall\, i \in M, p \in P \quad (2)$$

$$\sum_{s \in S} w_{isp} = 1, \quad \forall\, i \in M, p \in P \quad (3)$$

$$w_{i00} = 1, \quad \forall\, i \in M \quad (4)$$

$$w_{isp} \leq \sum_{k \in T_s} w_{ikp+1}\,, \quad \forall\, i \in M, s \in S, p \in \{0..(|P| - 2\} \quad (5)$$

$$\sum_{j \in N} x_{ijp} \leq 1, \quad \forall\, i \in M, p \in P \quad (6)$$

$$\sum_{i \in M} \sum_{p \in P} y_{ijp} = 1, \quad \forall\, j \in N \quad (7)$$

$$Pt_j * y_{ijp} \leq \sum_{t=p}^{p+Pt_j-1} x_{ijt}\,, \quad \forall\, i \in M, j \in N, p \in \{0..\left(|P| - Pt_j\right)\} \quad (8)$$

$$\frac{1}{Rw}\sum_{i\in M}\sum_{s\in S}\sum_{t=p}^{p+Rw-1} w_{ist} * Ec_s \leq d, \quad \forall\, p \in \{0..|P|-Rw\}: Rw \geq 1 \tag{9}$$

$$w_{isp} - w_{isp-1} \leq z_{isp}, \quad \forall\, i \in M, s \in TS, p \in P: p \geq 1, Tt_s \geq 2 \tag{10}$$

$$Tt_s * z_{isp} \leq \sum_{t=p}^{p+Tt_s-1} w_{ist}\,, \quad \forall\, i \in M, s \in TS, p \in \{1..(|P|-Tt_s)\}: Tt_s \geq 1 \tag{11}$$

$$\sum_{q\in(0,4)} w_{iqp} = 1, \quad \forall\, i \in M, p \in \{(|P|-Tt_4)..|P|\} \tag{12}$$

Figure 1: Mathematical Model presented by Batista Abikarram et al. (2019)

3.6 Differentiating elements of this thesis

Although almost every aspect of this thesis can be found in the above literature, there is no research which currently has the same combination of factors that this thesis does. Batista Abikarram et al. (2019) examined a very similar problem, but this thesis will expand upon the model provided in that work. Incorporating functionality to work with unrelated parallel machines instead of identical machines adds an element to this thesis not present in that paper. In addition, this thesis addresses the scalability issues presented by Batista Abikarram et al. (2019), allowing for optimal or near optimal solutions to be found in a reasonable amount of time for problems much larger than those that were tested previously.

3.7 Summary

The following table presents the major elements of the above literature review. There is very little literature which addresses peak demand or demand leveling, and those that do are currently not dealing with the exact system this thesis analyzes.

Table 1: Literature review summary

Key
ACO: Ant Colony Optimization, GA: Genetic Algorithm, PSO: Particle Swarm Optimization, L/IP: Linear or Integer Program
SM: Single Machine, PM: Parallel Machines

Source	System	Demand Charge	Solution Method	Largest Problem (Machines/Jobs)
Che et al. (2016)	SM	No	Heuristic	1/5000
Wang et al. (2016)	SM	No	Heuristic	1/18
Cheng et al. (2017)	SM	No	Heuristic	1/5000
Shrouf et al. (2014)	SM	No	GA	1/60
Yildirim and Mouzon (2012)	SM	No	GA	1/60
Rubaiee and Yildirim (2019)	SM	No	ACO	1/100
Angel et al. (2012)	PM	No	L/IP	Not Provided
Wang et al. (2018)	PM	No	GA	Not Provided
Agrawal and Rao (2014)	PM	No	GA	Not Provided
Hong and Fei (2017)	PM	No	GA	3/200
Fang and Lin (2013)	PM	No	PSO	5/50
Liang et al. (2015)	PM	No	ACO	10/100
Che et al. (2017)	PM	No	Heuristic	20/200
Van Den Dooren et al. (2017)	PM	No	LAHC Algorithm	100/5000
Ding et al. (2016)	PM	No	Heuristic	20/200
Li et al. (2016)	PM	No	Heuristic	12/200
Zhou et al. (2018)	PM	No	MODDE Algorithm	Not Provided
Moon et al. (2013)	PM	No	GA	20/65
Fazli Khalaf and Wang (2018)	Flow Shop	No	Heuristic	10/ n/a
Zhang and Chiong (2016)	Job Shop	No	GA	10/10
Nattaf et al. (2015)	PM	Yes	L/IP	Not Provided
Rager et al. (2015)	PM	Yes	GA	4/Not Provided
Batista Abikarram et al. (2019)	PM	Yes	L/IP	13/52

4. Preliminary algorithm and testing

A first round of testing was conducted comparing the heuristic, IP, and preliminary version of the genetic algorithm. The methodology of the heuristic and the GA are discussed. There is also a subsection of the methodology which explains the construction of the test problems and the testing procedure for the preliminary testing experiments

4.1 Greedy heuristic

A greedy heuristic is proposed in this work and can be described as follows. Initially, jobs are ordered from longest to shortest processing time in descending order. Jobs are then assigned to a machine and start time in descending order, from longest job to shortest job. For each job in

the list the schedule and associated cost of assigning a job to a machine and start time combination is calculated and stored for every possible remaining insertion point. While this does require some brute force enumeration, this enumeration decreases significantly for each job that must be placed and is overall quite small compared to the solution space. Following these calculations, the job is assigned to the machine/start time combination wherever the smallest increase in cost occurs. The job is removed from the task list and the next job is evaluated until no jobs remain to be placed. The cost considers the TOU consumption cost and the possibility of incurring a larger demand charge. The calculation of the consumption cost can be seen in the second term of the objective function from equation (1). The demand charge calculation can be found in the first term of the objective function, and that first term was calculated from equation (9). A summary of the greedy heuristic scheduling process can be seen in Figure 2.

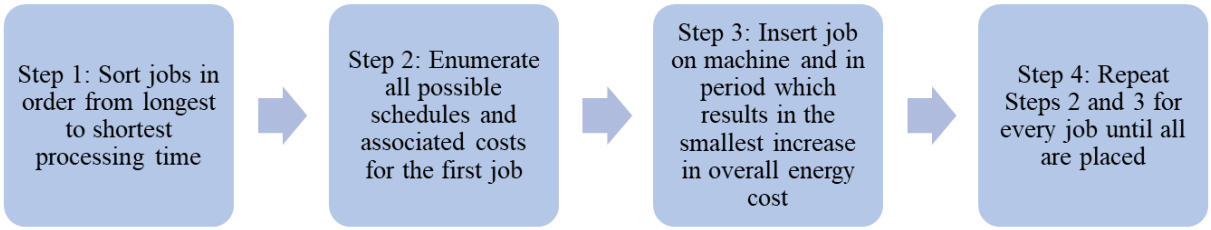


Figure 2: Greedy heuristic flowchart

4.2 Preliminary genetic algorithm

A genetic algorithm solves optimization problems by mimicking the behavior of natural selection. A population of solutions is generated and the candidates are evaluated for their fitness, which is the total electricity charges including TOU and demand charges in this case. Fit candidates remain in the population and are crossed with one another or mutated to get new solutions that may improve the fitness of the population. Weaker candidates may be killed and replaced with new candidates, or replaced by the offspring and mutations of the fit candidates. This cycle repeats with new generations until a near optimal solution is found.

This genetic algorithm was constructed using a specified population size, desired mutation rate and desired crossover rate. Given these parameters, a population was initialized and underwent various genetic operations. Population creation was done by sorting the jobs from longest to shortest processing time and attempting to place a job on a random machine in a random period.

A check was done during this process to force jobs to avoid preempting if possible. For this genetic algorithm, more focus was placed on the mutation of the single candidates than the crossover of pairs of candidates. Crossover only occurred if the jobs on one machine in one candidate matched the jobs on a machine in a different candidate. The schedules for the machines with matching job sets were then swapped, as illustrated in Figure 3(e). These same changes could be achieved by a combination of the mutation operators, thus the focus on mutation.

Following the mutation of a candidate or the creation of child candidates, these new members were compared to the weakest members of the current generation's population. Any new member with a more optimal cost replaced the weakest member in the population. The stopping criteria for the algorithm was when the best member of the population was unchanged for a predetermined number of generations.

This initial algorithm contained 4 mutation operators which are seen in Figure 3. The 1J1M operator randomly selected one job on one machine and shifted it a random number of periods in the day based on the amount of available time surrounding the selected job's initial schedule position. This was done by randomly selecting a machine, followed by a job on that machine to shift. Each machine and job were equally likely to be selected. A float between 0 and 1 determined whether the job would be shifted earlier or later in the day, with 0.5 and higher being a shift later in the day. This method was used to determine forward or backwards shifting in all of the mutation operators which shifted jobs. Lastly, available periods ahead of or behind the job were determined as necessary, and the job shifted a random number of periods within that availability. If no periods were available before or after the job, the mutation attempt was skipped. Figure 3(a) shows Job 2 being shifted one period earlier in the day.

The AJ1M operator took all the jobs on a single machine and shifted them either one period forward or backward. In Figure 3(b) all the jobs on Machine 1 are shifted one period later in the day. The AJAM operator acted in a similar manner to AJ1M but moved all the jobs in the whole system forward or back one period. In Figure 3(c) all jobs shown are shifted one period later. If the mutation created as a result of this shifting was infeasible, it was not inserted into the population.

Lastly, the S1J operator selected a job on one machine and attempted to move it onto a different machine. The machine to pull from was not selected at random. S1J pulled from the

machine with the least number of jobs to clear a machine if possible and squeeze its jobs onto other machines in the system. This in turn could lower the demand charge. The machine to move the job onto was whichever machine in the system had the longest single block of idle time available. If that block was longer than the job processing time, the job would insert without hesitation at the beginning of the idle block. However, if (as in the example shown in Figure 3(d)), the idle block was shorter than the job processing time, jobs on the destination machine would be shifted around to allow for the insertion of the incoming job. The shifting was done by inserting the job into the beginning of the longest idle period on the destination machine. Jobs before this time could remain untouched, but subsequent jobs were pushed later in the day to accommodate the incoming job.

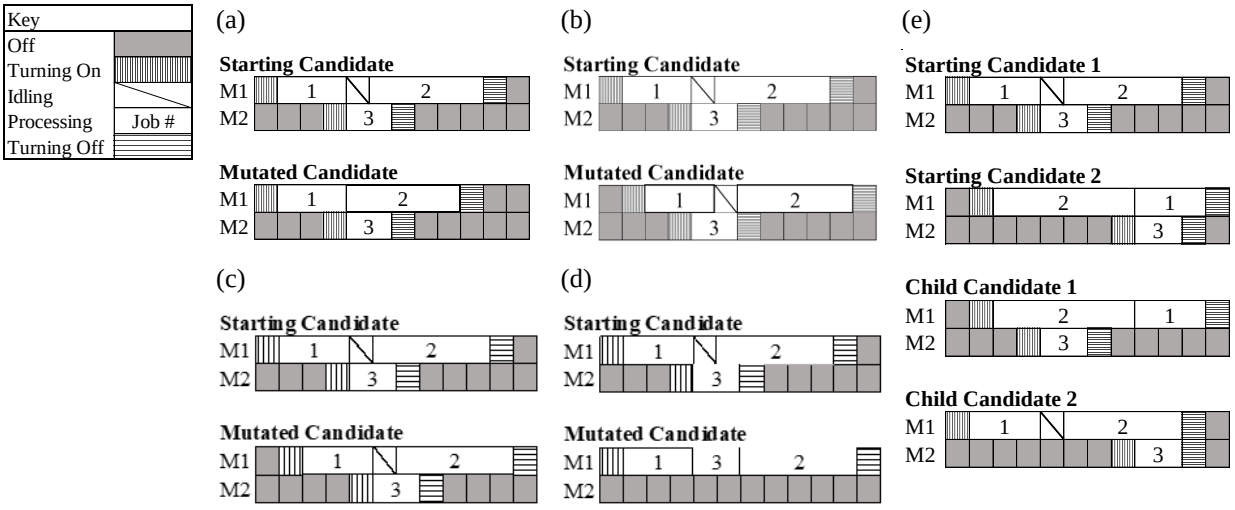


Figure 3: **(a)** Shift one job on one machine (1J1M) **(b)** Shift all jobs on one machine (AJ1M) **(c)** Shift all jobs on all machines (AJAM) **(d)** Swap one job from one machine to another (S1J) **(e)** Crossover operator

Algorithms 1,2, and 3 detail the crossover and mutation functions, as well as the overall structure of the genetic algorithm.

Algorithm 1: Crossover function

```
popsize = 200
cross_rate = 0.1
mut_rate = 0.7
crossover function:
        select (popsize*cross_rate) random pairs of candidates (parent1, parent2)
        for each pair:
                check if any machines on parent1 have the same set of jobs as any machine on parent2
                if yes:
                        schedule1 = schedule of machine from parent1 with jobsetA
                        schedule2 = schedule of machine from parent2 with jobsetA
                        create child1 by replacing schedule2 with schedule1
                        create child2 by replacing schedule1 with schedule2
                        compare both child candidate's cost to the weakest member of the population
                        if child has lower cost:
                                replace weak member with child candidate
                else:
                        continue to next pair
```

Algorithm 2: Mutation function

```
popsize = 200
cross_rate = 0.1
mut_rate = 0.7
mutation function:
        take (popsize*mut_rate) random candidates
        for each candidate:
                randomly choose mutation operator to perform
                create mutation
                compare mutation's cost to the weakest member of the population
                        if mutation has lower cost:
                                replace weak member with mutation
```

Algorithm 3: GA main function

```
popsize = 200
cross_rate = 0.1
mut_rate = 0.7
create a popsize population of candidates
sort population from lowest to highest total energy cost
best_member = first population member
generations_without_improvement = 0
while best member cost does not equal 0:
        if generations_without_improvement = 100:
                return best_member
        crossover function
        mutation function
        sort population from lowest to highest total energy cost
        current_best = first population member
        if current_best cost is better than best_member cost:
                best_member = current_best
                generations_without_improvement = 0
        generations_without_improvement += 1
```

4.3 Test problems and experimental design

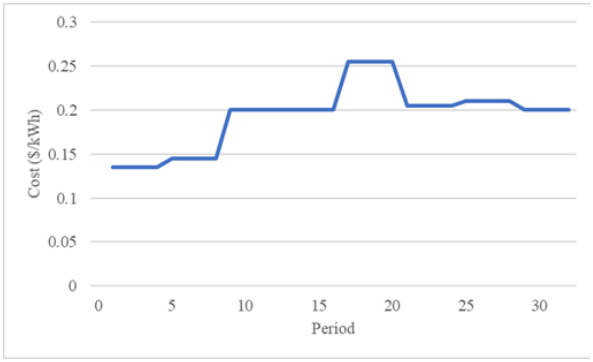
A schedule with 32 periods available on each machine per day was used for testing, and all jobs must be completed within the available timeframe. Machines in the system could be in five possible states. These states and energy consumption rates can be seen in Table 2.

Table 2: Machine states and energy consumption rates

Machine State	Off	Turning On	Idle	Processing Job	Turning Off
Energy Consumption Rate	0 kW	5 kW	2 kW	4 kW	1 kW

To test the 3 solutions, test instances were created where the number of machines, utilization rate, volatility of TOU pricing scheme, and the size of the demand charge were manipulated. The number of available machines was varied from 3 to 8 machines. For each number of machines, a 2^4 experiment was created and replicated twice, resulting in 32 problems for each number of machines and a total of 192 test instances. The first factor was the utilization rate of the system, tested at 70% and 90%. The other factors were the volatility of the TOU pricing scheme, the size of the demand charge at either 12 or 20 $/kW, and the number of jobs in the system. Pricing scheme 1 had consumption costs per period ranging from 0.135 to 0.255 $/kW, while pricing scheme 2 ranged from 0.135 to 2.55 $/kW. The two pricing schemes can be seen in Figure 4. Pricing scheme 2 existed to force the algorithms to avoid peak demand times as much as possible.

Pricing Scheme 1



Pricing Scheme 2

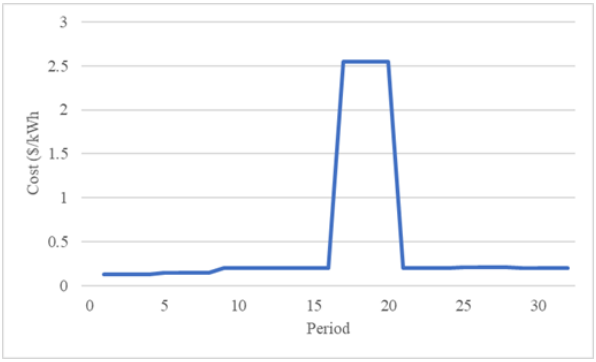


Figure 4: Preliminary pricing schemes

Since scalability was a primary metric, as the number of machines increased, the number of jobs in the system also was increased. For 3 machines, 10 and 20 job configurations were tested. For 4 machines, 20 and 30 jobs were used. This pattern continued until the largest system tested

was 8 machines with either 60 or 70 total jobs. When creating replications, the same number of machines and jobs were used, but the processing times of the jobs were generated randomly to sum to the desired utilization rate. This meant that each replication was identical except for the processing times of the jobs. Additionally, problems in this paper included only identical parallel machine systems to allow for a comparison to the IP, although unrelated machine capabilities were programmed into both the heuristic and genetic algorithm. Three methods were used to solve each of the 192 test problems: the greedy heuristic, the genetic algorithm, and the IP formulation provided in Batista Abikarram et al. (2019). The genetic algorithm used a specified population of 200, with a 10% crossover rate and a 70% mutation rate. The stopping condition for the genetic algorithm was set at 100 generations without improvement.

The integer program tests used CPLEX release version 12.8 on Windows 10 PC with a 6 core CPU and 16GB RAM. Pyomo Version 5.6 and Python Version 3.7 were used to interface with the CPLEX solver. Each method was given a maximum of 10 minutes to find a solution. If the genetic algorithm could not find an initial solution within 10 minutes, the program was terminated. When running the integer program, the CPLEX solver returned either an optimal solution in the permitted time or the current best solution if one was available after 10 minutes. If a solution was not available after 10 minutes, the program was terminated.

4.4 Preliminary results

Table 3 details the quality of the solutions and time required by CPLEX, the heuristic, and the genetic algorithm. “Optimal Test Instances” is the number of test instances solved to optimality in under 10 minutes, while “Total Solutions Provided” is the number of problems for which any solution was returned. Solutions returned by the greedy heuristic and genetic algorithm are considered “solved to optimality” even though it is expected they provide sub-optimal solutions. CPLEX was able to solve the smallest number of problems to optimality due to the ten minute cap on solve time. To provide a fair comparison, the optimality gap and average solve time presented in Table 3 only considered the 60 problems which all three methods solved to optimality. The “Average Solution Gap” compared only the candidate solutions provided by CPLEX to the genetic algorithm and heuristic for test instances that all three returned a solution, a total of 85 instances.

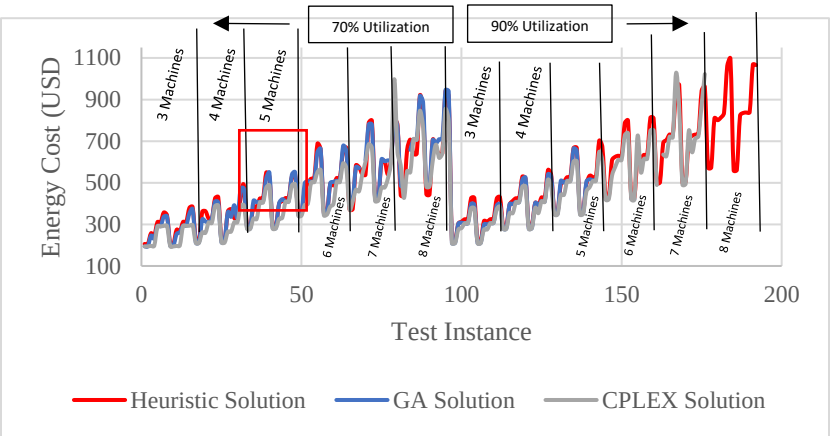
Table 3: Solution quality comparison between CPLEX, heuristic, and genetic algorithm

Solution Method	Optimal Test Instances	Average Optimality Gap	Average Solve Time (s)	Total Solutions Provided	Average Solution Gap
CPLEX	60	—	151.08	174	—
Greedy Heuristic	192	10.33%	0.79	192	9.91%
Genetic Algorithm	145	6.78%	58.77	145	8.58%

CPLEX was able to provide many more candidate solutions than fully optimized ones. An additional 118 tests provided candidate solutions compared to just 60 where an optimal solution could be identified. However, there were still 14 problems which CPLEX could not initialize within the 10 minutes to provide any solution. If the genetic algorithm found an initial solution, the algorithm was always able to run to completion before the 10 minute time limit. However, the GA could not initialize a starting population in order to provide any answer for 47 of the test instances. The greedy heuristic successfully solved all 192 test problems.

The greedy heuristic was the most efficient by far in terms of solve time compared to both the genetic algorithm and the integer program. This was expected, as the primary trade-off for the heuristic is solution time versus solution quality. The genetic algorithm was more efficient in solve time than CPLEX for the problems which the CPLEX solver completed. A comparison of the genetic algorithm, CPLEX, and the heuristic for solution quality across all test instances can be seen in Figure 5.

(a)



(b)

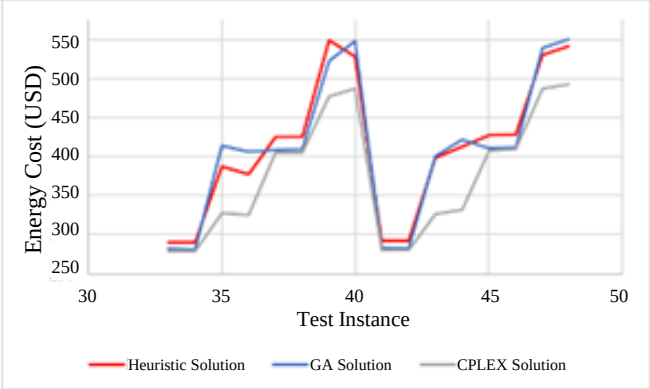


Figure 5: **(a)** Comparison of solution quality between CPLEX, GA, and heuristic for all tests **(b)** Sub-section displaying the 5 machine and 70% utilization tests

CPLEX was the best when solved to optimality but could not provide an optimal solution for any test instances beyond 6 machines and 50 jobs. The CPLEX solutions were particularly better than the GA and heuristic when the demand charge was larger, as seen at the peaks of the mountains in Figure 5(b). The valleys on the graph indicate that the algorithms performed very similarly to CPLEX when the demand charge was small. Candidate solutions were provided sporadically up to the max problem size. The GA reached the max problem size of 8 machines and 70 jobs, but only at the lower utilization rate. This drawback could be attributed to the random nature of the candidate creation in the genetic algorithm. When there was little flexibility in the available time in the system, the GA struggled to generate an initial population. Solutions provided by the greedy heuristic were comparable to the GA, occasionally providing better solutions. Based on these preliminary results, several improvement requirements for the GA were identified.

5. Final testing

Further and more structured testing was conducted following the creation of a revised genetic algorithm which was created to improve the performance and address issues of the preliminary genetic algorithm. No changes were made to the design of the greedy heuristic. The first improvement to the GA was an additional mutation operator which swapped two jobs on different machines of a single candidate (S2J). This complemented and expanded upon the S1J operator from the preliminary GA. In addition, the creation of candidates was altered to be less random by making use of the greedy heuristic to help create starting solutions. Lastly, the final genetic algorithm considered parameter tuning of the population size, crossover rate, mutation rate, and probabilities of each mutation operator.

5.1 Methodology

The adjustments to the genetic algorithm are discussed in this section. There is also a section which outlines the test problems and experimental design used for final testing.

5.1.1 Final genetic algorithm – population initialization

The primary reason the preliminary GA failed was that an initial population could not be created as a result of the size and complexity of the candidates. The preliminary GA relied solely on random placements of the jobs on the machines when creating candidates. While this allowed for the most diverse population, the system was too complex at higher utilization rates and randomly generating feasible solutions was nearly impossible. The revised genetic algorithm used permutations of the greedy heuristic to create the initial population. This change guaranteed that all candidates created were feasible. The original greedy heuristic solution was theoretically included in the population, which meant the final GA solution should be at least as good as the heuristic. Pseudocode of the new population creation method can be seen below.

Algorithm 4: Population creation method

```
while population is not full:
    if creating the first candidate:
        create candidate using greedy heuristic
        add candidate to the population
    else:
        create candidate using permutation of greedy heuristic
        if the candidate is feasible:
            add candidate to the population
return population
```

The creation of the heuristic permutations is explained in Figure 6. While this method did not create a very diverse population, this was a trade-off made to ensure feasibility. Group sizes were set at 2 and the percent of longest jobs was set at 25%. Testing was done to maximize the group sizes and the percent of longest jobs to insert unchanged, but problems using parameters other than those selected still had issues creating an initial population.

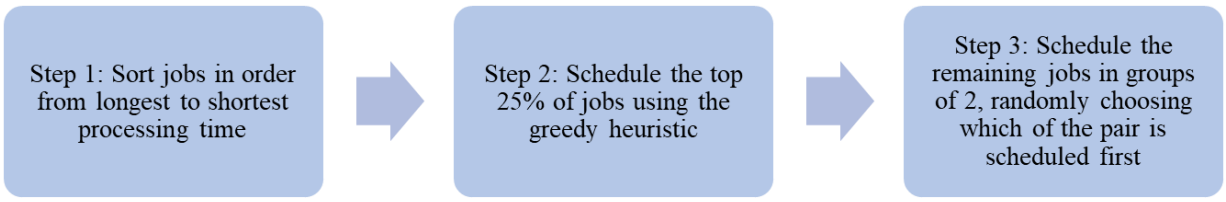


Figure 6: Flowchart of the population creation using shuffled versions of the heuristic

5.1.2 Final genetic algorithm – mutation adjustments

The S2J operator randomly selected a job on one machine and a different job from another machine and attempted to swap the two. The shorter of the two jobs could always insert into the new machine without issue, while the longer job may have required shifting jobs on the destination machine to make space. This was done by aggregating the total idle time on the destination machine, placing the job to be swapped in the earliest period of idle time, and shifting the remaining jobs to later in the day. This guaranteed that all available idle time could be used if needed, which was not a guarantee present in the preliminary GA. Figure 7 shows the S2J operator and can be seen below. Jobs 3 and 4 are swapped. Note that Job 4 simply inserts in the same start period as Job 3 was previously, but Job 3 gets placed in the first idle period and pushes Jobs 1 and 2 one period later in the day. If there was not enough space on one of the machines to perform the swap, the mutation attempt was concluded without success.

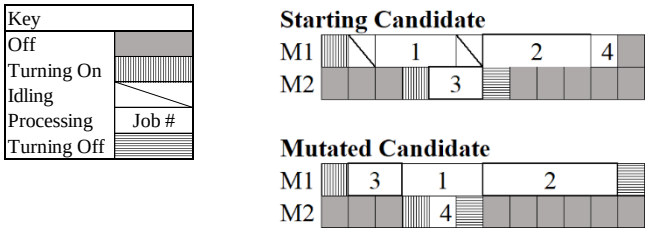


Figure 7: Swap 2 Jobs (S2J) operator

5.1.3 Final genetic algorithm – parameter tuning

The preliminary GA used predetermined parameters for the population size, crossover and mutation rates, and mutation probabilities. Population size, crossover rate, and mutation rate were tested in a 2^k experiment on a small set of test instances. Population size was set at either 20 or 200, crossover at 10% or 30%, and mutation at 30% or 70%. Dependent variables were the total energy cost found by the GA and the solve time in seconds. The problems used for this experiment were either 3 machines with 10 total jobs or 6 machines with 50 total jobs. Utilization rates of both 70% and 90% were used. The associated p-values of those factors can be seen in Table 4.

Table 4: Associated p-values of GA parameters

	Total Energy Cost	Solve Time (s)
Population Size	0.570	7.12E-08
Crossover Rate	0.907	0.695
Mutation Rate	0.733	0.790

The effect of population size on the solve time was shown to be the only significant factor. As a result, a population size of 20 was chosen for the final GA to provide faster solve times without sacrificing optimality. The crossover and mutation rates were kept from the preliminary GA at 10% and 70%, respectively.

Following this experiment, the population size, crossover rate, and mutation rate were held constant at the chosen values specified above and further tuning was done within the mutation rate to determine the specific probabilities of the five mutation operators. Three different sets of mutation probabilities were tested using the same test problems as the parameter tuning. The mutation probability sets were choosing all five operators at an equal probability, probabilities which were expected to produce better results, or probabilities which was purposefully chosen to produce poor results. Table 5 details the probability sets of the mutation tuning.

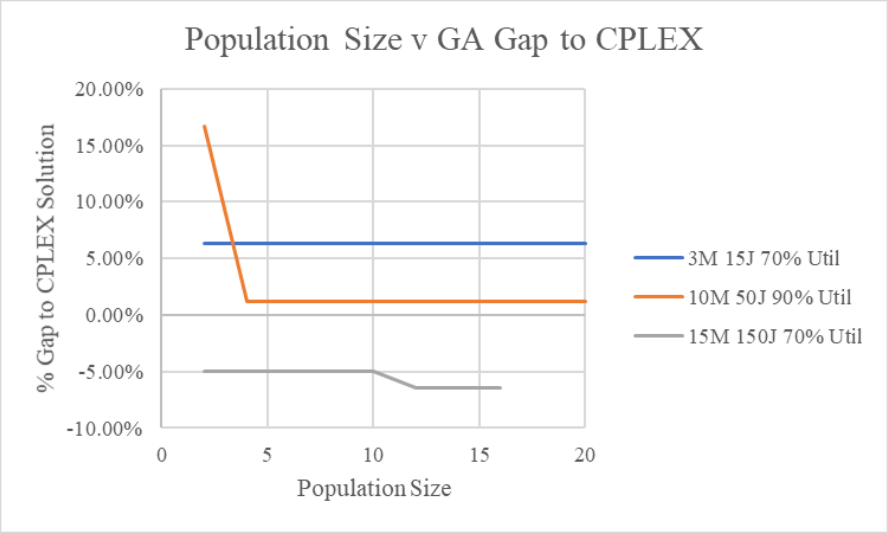
Table 5: Mutation operator probabilities for different parameter sets

Mutation Operator	All Even	Better Expectation	Worse Expectation
S2J	20%	30%	10%
S1J	20%	40%	15%
1J1M	20%	10%	50%
AJ1M	20%	5%	15%
AJAM	20%	5%	10%

Despite the expectations when creating the mutation sets, the associated p-value of the mutation set on the GA optimality was 0.992, indicating little to no influence on the optimality. Even probabilities for each mutation operator were used in the final GA testing.

Retrospectively, testing was done to assess the lower limit of population size that improved the quality of the solution and to verify that large population sizes were too time intensive to consider. Testing was done on a small, medium, and large problem at various starting population sizes. The population sizes tested were [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100], while the plots of the population size against solution quality and time can be seen in Figure 8. For all three problems, population sizes over 20 gave no improvement over the solution with 20 members and have been excluded from the solution quality plot. The smallest problem saw no improvement in the solution regardless of the population size. The largest problem saw an improvement when the population increased from 10 to 12 members, but not with any additional increases in size.

The solve time was shown to be a linear function based on the population size with the slope increasing as problem size increased. Population sizes over 16 could not be created within 30 minutes for the largest problem. Taking these two plots into consideration, this suggested that the ideal population size was around 15 members. The value of 20 which was used in testing was a fair number to use but may have been slightly larger than necessary.



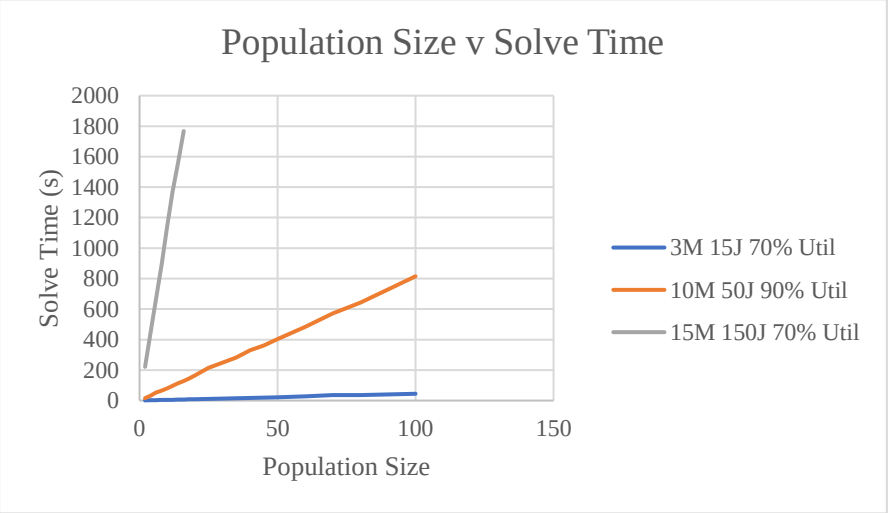


Figure 8: Population size testing results

5.2 Test problems and experimental design

For final testing, the pricing scheme was held constant using pricing scheme 1 from the preliminary tests, and the demand charge was held constant at $12/kW. The focus of this work was to test the solve time and scalability of the alternatives to the integer program from Batista Abikarram et al. (2019). It was shown in that work that these two factors did not significantly affect the model's solve time or ability to reach optimality. Focus for this work was placed on the machine-job configuration, and the utilization rate of the system. The number of the machines was set at 3, 5, 10, 15, and 20, while the number of jobs was set at either five or ten times the number of machines in the system. This meant that the smallest problem tested was 3 machines

and 15 total jobs and the largest problem was 20 machines and 200 total jobs. Utilization rate was set at 70% or 90%. This resulted in 20 different problem setups, which were each replicated 10 times for a total of 200 test problems. Replications of a problem setup were made by varying the lengths of the individual jobs in the system while still meeting the desired utilization. Problems were run in a very similar fashion to the preliminary testing. These problems were run using CPLEX release version 12.8 on Windows 10 PC with a 4 core CPU and 16GB RAM. Pyomo Version 5.8 and Python Version 3.8 were used to interface with the CPLEX solver. Each method was given a maximum of 30 minutes to find a solution. If the GA could not find a starting solution in that time, it was terminated. When running the integer program, the CPLEX solver returned either an optimal solution in the permitted time or a partially optimized solution after 30 minutes. If no solution was available after 30 minutes, it was terminated as well.

5.3 Results

Table 6 details the quality of the solutions and time required by CPLEX, the heuristic, and the genetic algorithm using the same metrics as the preliminary testing phase. CPLEX was again able to solve the fewest problems to optimality in the allotted time. Recall that "Optimal Test Instances" is the number of test instances solved to optimality in under 30 minutes by CPLEX, while "Total Solutions Provided" is the number of problems for which any solution was returned. In addition, the optimality gap and average solve time presented in Table 6 only considered the 71 problems which all three methods solved to optimality. The "Average Solution Gap" compared the 122 test instances in which all three methods returned a solution.

Table 6: Solution comparison form final testing

Solution Method	Optimal Solutions	Average Optimality Gap	Average Solve Time (s)	Total Solutions Provided	Average Solution Gap
CPLEX	71	—	459.92	122	—
Greedy Heuristic	200	4.53%	0.93	200	2.53%
Genetic Algorithm	180	2.58%	28.39	180	1.20%

The greedy heuristic performed efficiently and effectively again, completing all 200 problems with no issue. The heuristic solved problems much larger than the integer program could accomplish. When CPLEX did find an optimal solution, the heuristic found a solution on average nearly 500 times faster with only a 4.53% gap to the CPLEX solution The longest solve time for

the heuristic occurred in problems with 20 machines, 200 jobs, and 70% system utilization, which took 148 seconds on average.

The revised genetic algorithm performed much better than the preliminary GA in terms of solution quality. If the integer program was solved to optimality, the GA found a solution which sacrificed 2.58% optimality but was found 16 times faster. The 20 problems that the GA could not solve were the problems which had 20 machines and 200 total jobs. CPLEX could not solve these either; only the greedy heuristic was able to solve that sub-section of problems in the given time period.

A very high percentage of the GA solve time could be attributed to the population initialization. This can be seen in Table 7.

Table 7: % of GA runtime attributed to population initialization

Average	Max	Min
97.01%	99.25%	91.38%

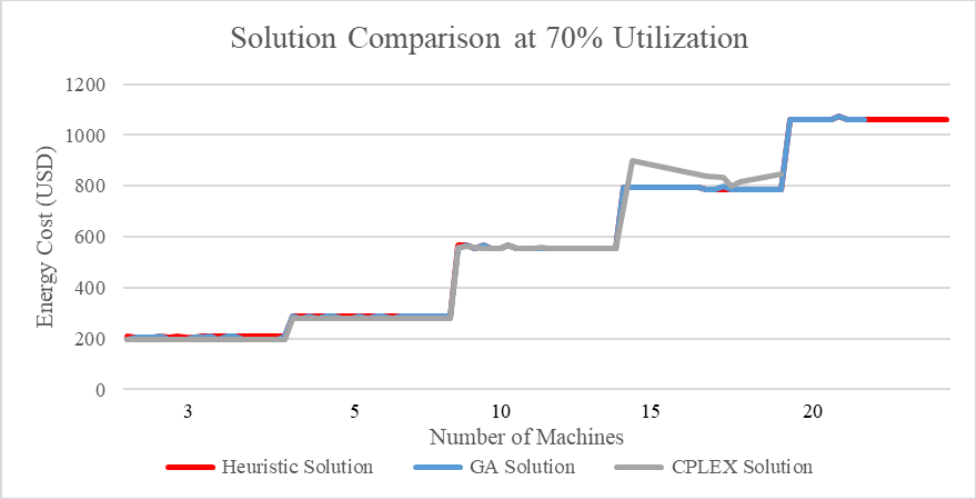
At a minimum, the population creation accounted for 91% of the GA runtime and was over 99% in other cases. For larger problems, the population creation took over 30 minutes. However, if given enough time to create an initial population, the GA could always run to completion. Once the population was created, the algorithm itself worked quickly. By using the heuristic to create the starting population, that population began with a higher fitness than in the preliminary GA. This led to reaching the stopping criteria of 100 generations without improvement in fewer total generations. Another benefit of this technique was that the final GA did not struggle in the 90% utilization problems as the preliminary GA did. Overall, the final genetic algorithm outperformed the preliminary GA. This can be seen in Table 8.

Table 8: Comparison of preliminary and final GA

	Average Optimality Gap to CPLEX	Average Solve Time (s)	% of Test Problems solved	Largest Problem Solved
Preliminary GA	6.78%	58.77	75.52	8 Machines, 70 total jobs, 70% utilization
Final GA	2.58%	28.39	90.00	15 machines, 150 total jobs, 90%utilization

Figure 9 provides a graphical representation of the three different solutions with respect to their quality.

(a)



(b)

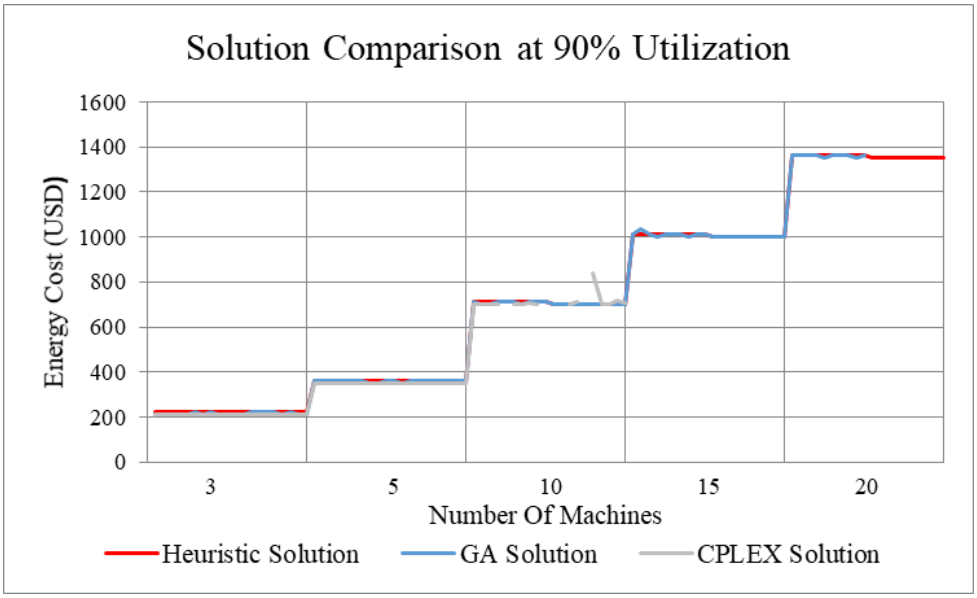


Figure 9: **(a)** Comparison of solution quality between CPLEX, GA, and heuristic at 70% utilization **(b)** Solution quality comparison at 90% utilization

In terms of solution quality, the greedy heuristic and GA performed very comparably to CPLEX. There were 69 instances when the GA was within less than 1% of the CPLEX solution. It can be seen in Figure 9(a) that sometimes the heuristic and GA produced better solutions than

the partially optimized CPLEX solution. This occurred 21 and 41 times for the heuristic and GA, respectively. Combined, the GA solved a total of 110 problems equal to or better than CPLEX in the allotted time.

While the GA was running, mutations of candidates were analyzed to determine how far from optimal that feasible solutions could be. Some mutations in the GA population were as much as 18% from the optimal solution in a 3 machine and 15 job problem, and 15% off in a 5 machine and 50 job setup, the largest setup which CPLEX solved to optimality. However, these mutations were still relatively strong as they were mutations of the heuristic solution. The true optimality gap of a feasible schedule could easily be quite larger. Therefore, it was very promising that the solutions from the final GA and heuristic were both regularly matching or exceeding CPLEX.

A few other interesting observations were made based on Figure 9. The plateaus in the graphs suggest that even though the test problems varied the number and lengths of jobs in the system for each level of machines, they ended up creating identical or near identical optimal costs. This suggests that the system utilization rate had a larger effect on the energy cost. Intuitively, this makes sense because the consumption cost will always be less if the system is used 20% less of the time. In addition, whether the number of jobs was 5 or 10 times the number of machines in the system did not significantly affect the consumption cost. The jobs were taking up the same amount of time in the day overall.

The performance of the GA and heuristic were also compared directly to one another. Table 9 shows that the GA improved on the heuristic solution 44 times, while 136 times it could not lower the total energy cost from the heuristic solution. The largest problem where the GA improved the solution more than 1% was 15 machines and 150 total jobs at 90% utilization. This suggests that for problems with 16 machines or more, the heuristic should be used. The genetic algorithm is most suitable for problems with 15 or fewer machines.

There were also 4 test instances when the heuristic performed better than the GA. This happened as a result of the parsing function which brought the heuristic solution into the GA. All of the job assignment information was parsed directly, but not when the machines would turn on at the beginning of the day. Addressing that element was left to a separate function in the GA code, which did not have that logic programmed correctly. As a result, machines in the GA were

unnecessarily turning on in the first period of the day, leading to higher costs than the heuristic solution which was passed in. Fixing this issue has been left for future work.

Table 9: Comparison of final GA and greedy heuristic

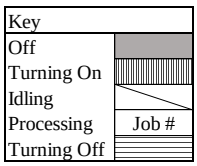
Average Solution Gap from Heuristic to GA	Problems with identical solutions	Problems where GA improved solution	Largest Problem GA improved solution	Problems Heuristic had lower cost
0.91%	136	44	15M, 150J, 90% Util	4

Lastly, the GA and heuristic were tested on a problem with 5 unrelated machines and 25 jobs to prove their unrelated machine capabilities. The machines in the problem consumed energy at different rates in the various machine states. These consumption rates are shown in Table 10. Machines 3 and 4 consumed the most energy, while machines 1 and 5 were the most energy efficient.

Table 10: Unrelated machine consumption rates

Machine	Off	Turning On	Idle	Processing Job	Turning Off
1	0 kW	4 kW	2 kW	3 kW	1 kW
2	0 kW	5 kW	2 kW	4 kW	1 kW
3	0 kW	7 kW	3 kW	6 kW	1 kW
4	0 kW	7 kW	3 kW	6 kW	1 kW
5	0 kW	4 kW	2 kW	3 kW	1 kW

Figure 10 visualizes the difference in schedules when the machines were identical versus unrelated. Figure 10(a) represents the identical machine setup and Figure 10(b) shows the unrelated machine setup. The heuristic scheduled virtually the same sets of jobs on each machine in the identical and unrelated setups, which shows its capability to pick the lowest cost increase when scheduling. There was no clear priority when scheduling Figure 10(a), as the heuristic simply picked a random machine to start with. However, there was a clear strategy in Figure 10(b). The schedule output shows that the greedy heuristic was able to successfully prioritize placing jobs on the machines with the lower energy consumption rates first.



(a)

Period	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
M1			1								22							12						13						9		
M2				15								17						18				4						8				23
M3				14					16					19					2				7					25				3
M4				6			11			20			5			10			21			24										
M5																																

(b)

Period	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
M1			1								22							12						13						5		
M2			14					16					19					2				7				25				3		
M3			6		11			20			9		10		21			24														
M4																																
M5				15					17							18						4					8				23	

Figure 10: (a) Schedule generated by the greedy heuristic with identical machines **(b)** Schedule generated by the greedy heuristic with unrelated machines

The energy cost found by the greedy heuristic for the unrelated machine problem was 282.84. This was also the cost found by the GA, which indicated that the GA could also properly account for the unrelated machines but could not improve the heuristic solution for the problem.

In addition to this example, 30 test problems were run to assess whether the GA could improve the heuristic solution in an unrelated machine setup. Machine levels were varied at 3,5, and 10 with the same 5x and 10x jobs and utilization rates as the final testing. Of these problems, there were 4 instances in which the GA reduced the cost. The largest problem tested was 10 machines with 100 jobs at a 70% utilization rate, and the GA was able to improve the heuristic in one of the replications at this problem size. When including unrelated machines, different parameter tuning and increased stopping criteria may be required to improve the performance of the GA

6. Conclusion and future work

This paper has presented two alternatives to an integer program that solve an identical parallel machine shop scheduling problem. The greedy heuristic and genetic algorithm developed provide the scalability and efficiency of solve time desired with minimal effect on the quality of the solution provided. The heuristic provided solutions to the same problems as the integer program nearly 500 times faster with only a 4.53% average optimality gap. The GA similarly provided answers over 16 times faster than the integer program within 2.58% of the CPLEX solution. Revisions to the genetic algorithm improved its performance in terms of both solution quality and solve time when compared to its preliminary version. The average optimality gap to CPLEX was reduced from 6.78% to 2.58% and the percent of test problems solved in the allotted time increased from 75.52% to 90%.

Future work should be dedicated to robustly testing and improving the unrelated machine capabilities of the heuristic and GA. While both methods were programmed with this capability, neither were tested properly in this paper due to the limitations of the integer program being used as a comparison. Either the two methods should be compared to one another in detail, or a third method with unrelated machine capability should be compared to the heuristic and GA presented in this work. Additionally, the heuristic and GA currently possess the ability to schedule jobs in a system where each machine may have different energy consumption rates in the different states. However, there is some literature which allows not only this element of unrelatedness between machines, but also allows the processing time of jobs to vary depending on which machine they are scheduled to. Incorporating this functionality is a logical next step for the two methods.

The other main area to address in future work is the ability of the GA to create an initial population. There is an issue with parsing the heuristic solution and the population creation takes up a very large proportion of the total GA execution time. Lowering this time could be done by reducing the number of candidates that are created initially and using mutations and child candidates to fill the remainder. The testing in this paper used a population size of 20. Under a new approach, maybe only 5 or 10 candidates are created and a ‘generation 0’ is run to reach the desired carrying population of 20 members. The issue with parsing the heuristic will require an additional function which translates it directly for the GA. The current method of translating everything but

the logical machine states still led to discrepancies between the heuristic solution and the GA's version of the same schedule.

References

Agrawal, P., & Rao, S. (2014). Energy-Aware Scheduling of Distributed Systems. *IEEE Transactions on Automation Science and Engineering, 11*(4), 1163-1175. doi:10.1109/TASE.2014.2308955

Angel, E., Bampis, E., & Kacem, F. (Nov 2012). *Energy Aware Scheduling for Unrelated Parallel Machines*. 2012 IEEE International Conference on Green Computing and Communications,

Batista Abikarram, J., McConky, K., & Proano, R. (2019). Energy cost minimization for unrelated parallel machine scheduling under real time and demand charge pricing. *Journal of Cleaner Production, 208*, 232-242. doi:10.1016/j.jclepro.2018.10.048

California Public Utilities Commission (2019). What are TOU Rates? https://www.cpuc.ca.gov/General.aspx?id=12194

Che, A., Zeng, Y., & Lyu, K. (2016). An efficient greedy insertion heuristic for energy-conscious single machine scheduling problem under time-of-use electricity tariffs. *Journal of Cleaner Production, 129*, 565-577. doi:10.1016/j.jclepro.2016.03.150

Che, A., Zhang, S., & Wu, X. (2017). Energy-conscious unrelated parallel machine scheduling under time-of-use electricity tariffs. *Journal of Cleaner Production, 156*, 688-697. doi:10.1016/j.jclepro.2017.04.018

Cheng, J., Chu, F., Liu, M., Wu, P., & Xia, W. (2017). Bi-criteria single-machine batch scheduling with machine on/off switching under time-of-use tariffs. *Computers & Industrial Engineering, 112*, 721-734. doi:10.1016/j.cie.2017.04.026

Corne, D. W., & Lones, M. A. (2018). Evolutionary Algorithms. https://arxiv.org/abs/1805.11014

Dieziger, D. (2000). Saving Money by Understanding Demand Charges on Your Electric Bill. United States Forest Service Technology and Development Publications. https://www.fs.fed.us/t-d/pubs/htmlpubs/htm00712373/

Ding, J.-Y., Song, S., Zhang, R., Chiong, R., & Wu, C. (2016). Parallel Machine Scheduling Under Time-of-Use Electricity Prices: New Models and Optimization Approaches. *IEEE Transactions on Automation Science and Engineering, 13*(2), 1138-1154. doi:10.1109/TASE.2015.2495328

Dorigo, M., & Di Caro, G. (1999). Ant Colony Optimization: A New Meta-Heuristic. Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). doi:10.1109/CEC.1999.782657

Fang, K.-T., & Lin, B. M. T. (2013). Parallel-machine scheduling to minimize tardiness penalty and power cost. *Computers & Industrial Engineering, 64*(1), 224-234. doi:10.1016/j.cie.2012.10.002

Fazli Khalaf, A., & Wang, Y. (2018). Energy-cost-aware flow shop scheduling considering intermittent renewables, energy storage, and real-time electricity pricing. *International Journal of Energy Research, 42*(12), 3928-3942. doi:10.1002/er.4130

Hong, L., & Fei, Q. (Aug 2017). *An improved genetic algorithm for a parallel machine scheduling problem with energy consideration*. 2017 13th IEEE Conference on Automation Science and Engineering (CASE). doi:10.1109/COASE.2017.8256314

Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. Proceedings of ICNN'95 - International Conference on Neural Networks. doi:10.1109/ICNN.1995.488968

Li, Z., Yang, H., Zhang, S., & Liu, G. (2016). Unrelated parallel machine scheduling problem with energy and tardiness cost. *The International Journal of Advanced Manufacturing Technology, 84*(1-4), 213-226. doi:10.1007/s00170-015-7657-2

Liang, P., Yang, H.-d., Liu, G.-s., & Guo, J.-h. (2015). An Ant Optimization Model for Unrelated Parallel Machine Scheduling with Energy Consumption and Total Tardiness. *Mathematical Problems in Engineering, 2015*, 1-8. doi:10.1155/2015/907034

Moon, J.-Y., Shin, K., & Park, J. (2013). Optimization of production scheduling with time-dependent and machine-dependent electricity cost for industrial energy efficiency. *The International Journal of Advanced Manufacturing Technology, 68*(1), 523-535. doi:10.1007/s00170-013-4749-8

Nattaf, M., Artigues, C., Lopez, P., Medina, R., Parada, V., & Pradenas, L. (Oct 2015). *A batch sizing and scheduling problem on parallel machines with different speeds, maintenance operations, setup times and energy costs*. 2015 International Conference on Industrial Engineering and Systems Management. doi:10.1109/IESM.2015.7380260

Nezamoddini, N., & Wang, Y. (2017). Real-time electricity pricing for industrial customers: Survey and case studies in the United States. *Applied Energy, 195*, 1023-1037. doi:10.1016/j.apenergy.2017.03.102

Rager, M., Gahm, C., & Denz, F. (2015). Energy-oriented scheduling based on Evolutionary Algorithms. *Computers & operations research, 54*, 218-231.

Rubaiee, S., & Yildirim, M. B. (2019). An energy-aware multiobjective ant colony algorithm to minimize total completion time and energy cost on a single-machine preemptive scheduling. *Computers & Industrial Engineering, 127*, 240-252. doi:10.1016/j.cie.2018.12.020

Shrouf, F., Ordieres-Meré, J., García-Sánchez, A., & Ortega-Mier, M. (2014). Optimizing the production scheduling of a single machine to minimize total energy consumption costs. *Journal of Cleaner Produ2015ction, 67*, 197-207. doi:10.1016/j.jclepro.2013.12.024

U.S. Department of Energy (2019). Time Based Rate Programs. https://www.smartgrid.gov/recovery_act/time_based_rate_programs.html

Van Den Dooren, D., Sys, T., Toffolo, T., Wauters, T., & Vanden Berghe, G. (2017). Multi-machine energy-aware scheduling. *EURO Journal on Computational Optimization, 5*(1), 285-307. doi:10.1007/s13675-016-0072-0

Wang, S., Liu, M., Chu, C., & Chu, F. (2016). Bi-objective optimization of a single machine batch scheduling problem with energy cost consideration. *Journal of Cleaner Production, 137*, 1205-1215. doi:10.1016/j.jclepro.2016.07.206

Wang, S., Wang, X., Yu, J., Ma, S., & Liu, M. (2018). Bi-objective identical parallel machine scheduling to minimize total energy consumption and makespan. *Journal of Cleaner Production, 193*, 424-440. doi:10.1016/j.jclepro.2018.05.056

Yildirim, M. B., & Mouzon, G. (2012). Single-Machine Sustainable Production Planning to Minimize Total Energy Consumption and Total Completion Time Using a Multiple Objective Genetic Algorithm. *IEEE Transactions on Engineering Management, 59*(4), 585-597. doi:10.1109/TEM.2011.2171055

Zhang, R., & Chiong, R. (2016). Solving the energy-efficient job shop scheduling problem: a multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption. *Journal of Cleaner Production, 112*, 3361-3375. doi:10.1016/j.jclepro.2015.09.097

Zhou, S., Li, X., Du, N., Pang, Y., & Chen, H. (2018). A multi-objective differential evolution algorithm for parallel batch processing machine scheduling considering electricity consumption cost. *Computers and Operations Research, 96*, 55-68. doi:10.1016/j.cor.2018.04.009