Adaptive Population-based Simulated Annealing for Uncertain Resource Constrained Job Scheduling

Dhananjay Thiruvady, Su Nguyen, Yuan Sun, Fatemeh Shiri, Nayyar Zaidi, Xiaodong Li Fellow, IEEE

Abstract—Transporting ore from mines to ports is of significant interest in mining supply chains. These operations are commonly associated with growing costs and a lack of resources. Large mining companies are interested in optimally allocating their resources to reduce operational costs. This problem has been previously investigated in the literature as resource constrained job scheduling (RCJS). While a number of optimisation methods have been proposed to tackle the deterministic problem, the uncertainty associated with resource availability, an inevitable challenge in mining operations, has received less attention. RCJS with uncertainty is a hard combinatorial optimisation problem that cannot be solved efficiently with existing optimisation methods. This study proposes an adaptive population-based simulated annealing algorithm that can overcome the limitations of existing methods for RCJS with uncertainty including the pre-mature convergence, the excessive number of hyper-parameters, and the inefficiency in coping with different uncertainty levels. This new algorithm is designed to effectively balance exploration and exploitation, by using a population, modifying the cooling schedule in the Metropolis-Hastings algorithm, and using an adaptive mechanism to select perturbation operators. The results show that the proposed algorithm outperforms existing methods across a wide range of benchmark RCJS instances and uncertainty levels. Moreover, new best known solutions are discovered for all but one problem instance across all uncertainty levels.

Index Terms—Resource Constrained Job Scheduling; Uncertainty; Simulated Annealing; Adaptive Perturbation

I. INTRODUCTION

The mining supply chain in Australia is one of the largest in the world, where a significant proportion of the minerals mined are exported overseas. Mining is major contributor to the Australian economy, with the exporting minerals contributing around $AUD 200 billion per annum to the economy, employing 240,000 individuals directly, and around 850,000 individuals indirectly [1]. Due to its role in the export of raw materials, Australia has developed a reputation in building mining technologies and is one of the largest suppliers of these technologies and software, worldwide [1].

There are numerous problems within the mining supply chain. Of significant interest, is the transportation of ore from mines to ports. In particular, transporting ore from multiple mining sites to a port where resources (trains and trucks) must be shared, is a key problem. If this problem is solved effectively, it can lead to large increases in throughput, thereby leading to substantial gains in revenue for the mining companies. In an abstract form, this problem can be formulated as a resource constrained job scheduling (RCJS) problem. The transportation of ore in batches can be viewed as jobs, while different transport modes (trains or trucks), can be viewed as resources. In the real setting, certain batches are required to be loaded on ships before others, and this aspect can be modelled by precedences between jobs. The timely arrival of ore at the ports is very important, as batches of ore that unnecessarily wait will lead to demurrage costs. This aspect is incorporated as a weighted tardiness for each job (different batches of ore being delayed lead to different costs), and the aim is to ensure the total weighted tardiness across all jobs is kept at minimum.

In the literature, this problem is known as RCJS with a shared central resource [2], [3], [4]. The complexity of RCJS makes it challenging for optimisation algorithms, and hence, several approaches have been proposed to tackle it. These include metaheuristics and matheuristics [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. The problem was introduced by [4], who proposed a Lagrangian relaxation based heuristic, which was able to find good solutions and lower bounds. Singh and Weiskircher [2], [3] consider a similar problem to RCJS that is modelled as machine scheduling with fractional shared resources. They propose collaborative approaches via agent-based modelling, which prove to be effective in tackling these problems. Ernst and Singh [5] propose a Lagrangian relaxation based matheuristic that employs particle swarm optimisation, which showed improvements in solution quality across the RCJS benchmark dataset. Thiruvady et al. [6] consider a variant of RCJS with hard due dates, and they show that an ant colony optimisation (ACO) and constraint programming hybrid can efficiently find feasible and high quality solutions. The study by Thiruvady et al. [7] proposes a column generation approach for RCJS, which finds excellent solutions and lower bounds. An inherent issue is the time requirements in solving this problem, and hence, parallel implementations of metaheuristics and matheuristics have been attempted [8], [9]. The best known solutions for RCJS have been obtained by Nguyen et al. [11] and Blum et al. [14]. Nguyen et al. [11] develops a hybrid of differential evolution and column generation, and they show excellent results on small and medium sized problems. Blum et al. [14] devise a biased random key genetic algorithm, which produces excellent results, especially for large problem instances. In the interest of finding good solutions quickly, studies have shown that genetic programming can be effective for RCJS [10], [12], [15]. Finally, Thiruvady et al. [13] propose Merge search and CMSA for RCJS.

RCJS under uncertainty (RCJSU) is a variant of RCJS that takes into consideration different practical random sources such as job release times, job processing times or the avail-
ability of resources. Of vital importance in RCJS, and crucial in real-world settings, is the uncertain resource availabilities. That is, if trains and trucks breakdown or tracks and roads need maintenance leading to delays in the ore reaching ports, there can be substantial losses incurred. Therefore, planners need a solution method that can address disruptions due to random events to ensure that the solutions obtained can ensure smooth operations either by incorporating uncertainty into optimisation problems, reoptimisation, or reactive dispatching rules. Thiruvady et al. [16] developed the first RCJSU formulation, and proposed a surrogate-assisted ant colony system (SACS) to find robust solutions. The results show that SACS outperforms solution methods based on mixed integer programming (MIP), meta-heuristics and surrogate-assisted evolutionary algorithms. Further analyses show that SACS finds good solutions more quickly compared to standard ACO algorithms.

Although SACS has shown promising results, there are some key limitations. Firstly, surrogate models help SACS address the challenge of expensive solution evaluations, but the proposed algorithm tends to converge prematurely, especially for large instances. Therefore, SACS (as well as other existing algorithms in the study of Thiruvady et al. [16]) can waste a lot of time exploring the search space without making significant progress. Secondly, the proposed method has many pre-defined hyper-parameters such as the learning rate of ant colony systems (ACS) and pre-selection rate for the surrogate models. Because of this issue, the algorithm can have trouble in solving a wide range of instances without major modifications (e.g. reoptimise hyper-parameters). Finally, the performance of SACS is not consistent across different uncertainty levels and different problem sizes (i.e., number of machines). For medium problem sizes and large uncertainty levels, SACS does not show significantly superior performance compared to standard ACS.

This paper addresses the three key limitations discussed above by developing a new adaptive population-based simulated annealing (APSA) algorithm for RCJSU. The main contributions of this paper are summarised as follows: (1) a new population-based simulated annealing algorithm with a balance between exploration and exploitation, (2) a new efficient Metropolis-Hastings algorithm extended from the simulated annealing algorithm proposed in [4], and (3) a new adaptive mechanism to update the probabilities for selecting perturbation operators to avoid premature convergence. APSA is similar to memetic algorithms, and population-based stochastic local search [17], [18] that combines evolutionary algorithms and local search heuristics; however, APSA adopts a much smaller population in which the population is improved mainly via local search and adaptive perturbations rather than crossover operators. The balance between exploration and exploitation within APSA allows the algorithms to perform efficiently against noisy and complex problems like RCJSU. Extensive experiments are conducted to validate the superior performance of the proposed algorithms for RCJSU and RCJS.

The paper is organised as follows. Section II provides a review of related work. In Section III a formulation of the RCJSU problem is provided, including an integer programming model. Section IV discusses the adaptive population-based simulated annealing algorithm proposed in this study. The experimental evaluation and associated results are presented in Section V-A. Section VI concludes the paper and presents possibilities for future work.

II. RELATED WORK

A review of the literature related to RCJS is provided here, including methods and techniques that are effective at tackling problems similar to RCJS and RCJSU.

A. Optimisation techniques for tackling RCJS

Owing to the complexity of RCJS, mathematical programming methods on their own do not suffice in finding optimal or even feasible solutions to relatively small problems. Hence, metaheuristics and matheuristics have been popular approaches for tackling the problem. Metaheuristics, including partial swarm optimisation [5], ACO [8], differential evolution [11], and genetic algorithms [14] have been known to find good solutions, albeit with large time overheads. RCJS can be modelled with graphs, and hence, ACO lends itself well to the problem. Nonetheless, past studies on RCJS have shown that ACO tends to converge slowly and gets trapped in local optima. Moreover, diversification strategies including local search (e.g., parallel implementations or niching techniques) help ACO, but the challenges still remain. Hybrid methods, especially those that can guarantee optimality, have also been explored [4], [2], [11], but the results show that only relatively small RCJS problem instances (up to 30 jobs) can be solved with these methods.

B. Scheduling on problems with uncertainty

Uncertainty is inherent in many real-world applications, and dealing with uncertainty is often challenging when devising scheduling algorithms. If uncertainty is not considered, the ensuing scheduling algorithms in the deterministic setting are often not effective at even finding feasible solutions (e.g. if a machine breaks down, a job will not be able to start at its pre-determined start time). Hence, a good understanding of the setting in which the schedules are applied is important to ensure success of the scheduling algorithms. Often, reactive scheduling strategies can be more effective in dealing with frequently occurring situations with uncertainties, since the latest information can be incorporated into the decisions. Nonetheless, if predicting the schedules is needed and knowledge of uncertainty is available, the preference is to develop robust schedules. The focus of this paper is on finding robust schedules for RCJSU, since predictable schedules are necessary for mining operations to coordinate supporting activities and to ensure that ore arrives at ports in a timely manner. Of particular interest is the timely arrival of ore at the ports, which relies greatly on the availability of resources. This is indeed the focus in the RCJSU problem and presented in the formulation in Equations (1) to (7) (see Section III).

When tackling scheduling problems with uncertainty, the aim is typically to devise methods that are capable of identifying good solutions under any uncertain scenario. These solutions are called robust solutions and the research area, robust
proactive scheduling under uncertainty. There are numerous studies focusing on this area, which we summarise here. Beck and Wilson\textsuperscript{[19]} investigate job shop scheduling, focusing on a variant which consists of stochastic processing times. Their results demonstrate that hybrids of tabu search and constraint programming with Monte Carlo simulation are effective in handling large-sized problems with uncertainty. The study by Song et al.\textsuperscript{[20]} proposes branch-and-bound algorithms that generate robust schedules on a variant of resource-constrained project scheduling with processing times that are uncertain depending on time. Identifying robust solutions in uncertain problems introduces additional complexity, and for this purpose metaheuristics such as swarm intelligence algorithms and evolutionary approaches are known to be effective. Bierwirth and Mattfield\textsuperscript{[21]} study genetic algorithms for dynamic job arrivals in classic job shop scheduling. Their results show that their approaches are superior to priority dispatching rules, which are popular in dynamic production scheduling. Wang et al.\textsuperscript{[22]} devise a surrogate-assisted multi-objective evolutionary method to tackle proactive scheduling where machines are considered to have stochastic breakdowns. To cope with the uncertainty, they propose a job compression and rescheduling strategy\textsuperscript{[23]}. Support vector regression in conjunction with surrogate models have been proposed to enhance the speed of time-intensive simulations. The study by Wang et al.\textsuperscript{[24]} provides an in-depth analysis and discussion of scheduling with machine breakdowns. Systematic reviews of scheduling in problems with uncertainty have also been conducted\textsuperscript{[25], [26]}.

The least two decades have seen numerous studies that consider uncertainty in scheduling problems similar to that of RCJS. There are several similarities to project scheduling, and many studies have considered uncertainty within this setting\textsuperscript{[27], [28], [29], [30], [31], [32], [33]}. In these studies, uncertainty is typically considered in respect to the resources, though, a few consider processing time related uncertainty. Lambrechts et al.\textsuperscript{[29]} study uncertainty with resources, and specifically modes of execution related to the activities. They develop the notion of “time buffering,” that efficiently generates robust schedules. Masmoudi and Haït\textsuperscript{[30]} consider resource levelling and resource constrained project scheduling, and they propose fuzzy modelling to tackle uncertainty. Their solution approaches are a genetic algorithm and a greedy method to identify robust schedules. Khodakarami et al.\textsuperscript{[31]} investigate Bayesian networks to identify causal structures considering uncertainty in sources and parameters of a project. They find that Bayesian networks are indeed very effective at dealing with uncertainty, including aspects such as representation, elicitation and management. Uncertainty associated with processing times has also been investigated\textsuperscript{[32], [33]}.

Moradi and Shadrokh\textsuperscript{[32]} develop a problem-specific heuristic to deal with uncertainty, and the study by Chakraborty et al.\textsuperscript{[33]} propose several Branch & Cut heuristics for project scheduling.\textsuperscript{[1]}

\textsuperscript{2}Benchmark datasets obtained from the project scheduling problem library (PSPLIB)\textsuperscript{[34]}

\textsuperscript{3}We refer the reader to the study by Thiruvady et al.\textsuperscript{[16]} for a detailed discussion of uncertainty in respect to this problem

\section*{C. Hybrid optimisation methods}

Local search and population-based methods\textsuperscript{[35]} have been extensively investigated in the literature to tackle challenging scheduling problems. While local search heuristics are best known for their ability to find local optima efficiently, population-based optimisation methods such as genetic algorithms are better known for their exploration ability. Empirical studies have demonstrated the need to maintain a balance between exploration and exploitation when dealing with difficult combinatorial optimisation problems such as job shop scheduling\textsuperscript{[36]}. Such hybrid methods are commonly referred to as memetic algorithms\textsuperscript{[17]} or population-based stochastic local search algorithms\textsuperscript{[18]} in the literature. While hybrid algorithms are effective in combinatorial optimisation, there are still a few shortcomings. Firstly, design choices to develop competitive hybrid algorithms are difficult to make and can be sensitive to different problem subsets. Secondly, the performance of hybrid algorithms depend strongly on an “ideal” setting of the algorithm parameters.

Hybrid algorithms have also been developed for RCJS. The study that introduced the problem\textsuperscript{[4]} develop a Lagrangian relaxation heuristic, which combines MIP and simulated annealing. Following this, Thiruvady et al.\textsuperscript{[7]} show that a column generation and ACO hybrid is very effective on this problem by finding good lower bounds. Parallel implementations of ACO\textsuperscript{[8]} have also been effective, by finding high-quality solutions quickly.\textsuperscript{[11]} propose a hybrid algorithm combining differential evolution and local search with excellent results. The study by Thiruvady et al.\textsuperscript{[13]} show that parallel matheuristics based on MIP and ACO can effectively tackle this problem.

Thiruvady et al.\textsuperscript{[16]} has developed SACS, a sophisticated algorithm combining ACO, surrogate modelling, and local search to deal with RCJSU. The experiments show that SACS can find excellent solutions for small problems thereby outperforming other meta-heuristics. However, further analyses show that SACS tends to converge prematurely and is inconsistent when working with different problem subsets. Moreover, SACS has many hyper-parameters (for ACO, local search, surrogate models), for which the ideal settings are difficult to find.

\section*{III. Problem Formulation}

The formal definition of RCJSU as an integer program (IP) was provided by Thiruvady et al.\textsuperscript{[16]}. Past studies have considered uncertainty in different ways, including resource limits\textsuperscript{[29]} or uncertainty in job/task processing times\textsuperscript{[33]}. However, in this study the focus is on uncertainty derived from the variability in resources, which in turn affects the processing times of jobs/tasks.\textsuperscript{[3]} The problem is NP-hard as it is at least as hard as its deterministic counterpart, which is indeed shown to be NP-hard\textsuperscript{[4]}. This IP formulation extends the one of Singh and Ernst\textsuperscript{[4]}, proposed for RCJS. Uncertainty is introduced by considering several uncertain resource levels,
that is, some proportion of the original resource limit is used for each uncertainty sample (details in Section V-A).

We are given several jobs \( J = \{1, \ldots, n\} \), which are associated with the following information. Each job \( i \) must execute on machine \( m_i \) (\( M = \{1, \ldots, l\} \)), has a release time \( r_i \), processing time \( p_i \), due time \( d_i \), weight \( w_i \) and resource usage \( g_i \). Jobs on the same machine cannot execute concurrently, or in other words, only one job may execute at one time. A time horizon is given, consisting of discrete points \( T = \{0, \ldots, D\} \), and \( D \) is chosen to be sufficiently large to ensure a feasible solution will always be found. Precendences may exist between any two jobs \( i \in J \) and \( j \in J \) that execute on the same machine. This is represented as \( i \rightarrow j \), where job \( i \) completes before job \( j \) starts.

A central resource constraint exists across all machines, where the cumulative resource usage of jobs executing at any time point must be at most \( G \). When considering uncertainty, \( G \) is allowed to vary. This leads to defining a sample, which has available a proportion of the resource \( G_s \), and is an instance of the problem. As an input, the total number of samples \( u \) is provided, and the resource limit is \( G_s \) applies to the \( s \)th sample \( U = \{1, \ldots, u_S\} \).

The IP can be defined as follows. Let \( z_{sjt} \) be a binary variable, which indicates in sample \( s \), job \( j \) completes or has completed by time \( t \), if \( z_{sjt} = 1 \). By this definition, a job stays complete once completed. Compared to RCJS, the RCJSU IP formulation requires additional space complexity, which consists of \( O(n^3) \) variables, and each set of constraints with an additional dimension owing to the samples. The IP formulation is:

\[
\min_{s \in U} \sum_{s \in U} \sum_{j \in J} \sum_{t \in T} \sum_{i \in \{1, \ldots, t_{\text{max}}\}} w_{jt} \times (z_{sjt} - z_{sjt-1})
\]

s.t.

\[
\begin{align*}
z_{sjt_{\text{max}}} &= 1 & \forall s \in U, \forall j \in J \\
z_{sjt} - z_{sjt-1} &\geq 0 & \forall j \in J, \forall s \in U, \forall t \in T, \forall e_j \in \{1, \ldots, t_{\text{max}}\} \\
z_{sjt} &= 0 & \forall t \in T, \forall s \in U, \forall j \in J \\
z_{sb} - z_{sa,t-p_b} &\leq 0 & \forall (a, b) \in C, \forall s \in U, \forall t \in T \\
\sum_{j \in J} z_{sj,t+p_j} - z_{sjt} &\leq 1 & \forall i \in M, \forall s \in U, \forall t \in T \\
\sum_{j \in J} g_j \cdot (z_{sj,t+p_j} - z_{sjt}) &\leq G_s & \forall s \in U, \forall t \in T
\end{align*}
\]

Equation (1), the objective function, states that the average total weighted tardiness (TWT) across all samples must be minimised. Constraints (2) ensure that all jobs complete. Once a job completes it stays complete, which is enforced by Constraints (3). Constraints (4) ensure that all jobs satisfy release times and the precedences are imposed by Constraints (5). In this constraint, \( e_j = r_j + p_j - 1 \). Constraints (6) require that only one job can execute on one machine at every time point. Constraints (7) ensure that the resource limits, considering each sample, are satisfied.

For the RCJS, there are efficient scheduling heuristics that are known (see Section IV-D). These heuristics typically depend on an alternative formulation, where a sequence or permutation jobs are efficiently mapped to schedules. Previous studies have used this approach for RCJS (e.g. (7)), and it has also proved beneficial for the RCJSU (10). In this study, \( \pi \) represents a sequence of jobs while \( \sigma(\pi) \) represents an efficient mapping from \( \pi \) to a feasible schedule. Therefore, \( \pi \) is effectively a solution to the problem, where given a sequence, a deterministic schedule can be efficiently constructed.

An example RCJS problem instance is shown in Table I. This example shows how a solution is constructed and what the optimal solution is. The problem consists of three jobs, with the associated information in the tables. Moreover, the resource capacity is 10 units. Considering \( \pi \), there are six potential sequences. Among the jobs, the resulting schedules depend on an alternative formulation, where a sequence or permutation of jobs are efficiently mapped to schedules. Previous approaches was simulated annealing (4). However, to tackle RCJSU, the traditional simulated annealing method has a few shortcomings, and hence, we propose an adaptive population-based simulated annealing (APSA). In particular, this method has three main differences from the traditional approach: (1) a population of solutions to track and exploit best known solutions, (2) a faster cooling schedule within the Metropolis-Hastings algorithm, and (3) adaptive probabilistic selection of the neighbourhood moves to perturb solutions in the population. Comparisons are made to ACS (ant colony system) and SACS (surrogate-assisted ant colony system) of (16), which have previously been effective on RCJSU.

### IV. Population-based Simulated Annealing

Several metaheuristics have been able to find excellent solutions to RCJS in a time efficient manner. One of these approaches was simulated annealing (4). However, to tackle RCJSU, the traditional simulated annealing method has a few shortcomings, and hence, we propose an adaptive population-based simulated annealing (APSA). In particular, this method has three main differences from the traditional approach: (1) a population of solutions to track and exploit best known solutions, (2) a faster cooling schedule within the Metropolis-Hastings algorithm, and (3) adaptive probabilistic selection of the neighbourhhood moves to perturb solutions in the population. Comparisons are made to ACS (ant colony system) and SACS (surrogate-assisted ant colony system) of (16), which have previously been effective on RCJSU.

#### A. The APSA Algorithm

Algorithm 1 shows the high-level APSA procedure for RCJSU. As input, the algorithm takes the following: (a) \( T_0 \): an initial temperature, (b) \( t_{\text{lim}} \): time limit for the execution of the algorithm, (c) \( \text{Iter} \): number of Metropolis-Hastings iterations, (d) \( s \): population size, (e) "\( \gamma \)": a parameter to control the cooling schedule and (f) "\( \rho \)" a parameter that specifies how to adapt the probabilities (\( \rho = 0.9 \) for this study). In Line 2, the population \( \Pi \) is initialised, where each solution consists of a random list of
Algorithm 1: Adaptive Population-based Simulated Annealing (APSA)

1: Input: An MMALBPS instance, \( T_0, t_{lim}, \text{Iter}, s, \gamma, \rho \)
2: \( \Pi \leftarrow \text{Initialise}(s), \pi^{bs} \leftarrow \emptyset \)
3: \( p^b \leftarrow 0.65, p^i \leftarrow 0.3, p^t \leftarrow 0.05 \)
4: while execution time < \( t_{lim} \) do
5:   for \( \pi \in \Pi \) do
6:     \( \pi^t := \text{MetropolisHastings}(\pi, T_0, \text{Iter}, \gamma) \)
7:     \( \pi^{bs} := \text{Update}(\pi) \)
8:     \( r \leftarrow \text{rand()} \)
9:     if \( r < p^b \) then
10:        \( \beta - \text{Sampling}(\pi^{bs}) \)
11:        \( p^b \leftarrow p^b \times \rho \)
12:     else if \( r < p^b + p^i \) then
13:        \( \pi := \text{SwapOps}(\pi^{ib}, m) \)
14:        \( p^i \leftarrow p^i \times \rho \)
15:     else
16:        \( \pi \leftarrow \text{RandomList()} \)
17:        \( p^t \leftarrow p^t \times \rho \)
18:     end if
19:     \( \text{Normalise}(p^b, p^i, p^t) \)
20:   end for
21: end while
22: \( \pi^f := \text{SwapAllJobPairs}(\pi^{bs}) \)
23: return \( \pi^f \)

Algorithm 2: MetropolisHastings

1: Input: \( \pi, T_0, \text{Iter}, \gamma \)
2: \( C \leftarrow \infty, T^c \leftarrow T_0, \pi^{ib} \leftarrow \pi \)
3: for \( i = 1 \) to \( \text{Iter} \) do
4:   \( \pi^{ib} \leftarrow \text{SwapJobs}(\pi^{ib}) \)
5: if \( f(\pi^{ib}) < C \) then
6:   \( C \leftarrow f(\pi^{ib}) \)
7: end if
8: \( i \leftarrow 1 \)
9: else
10: \( \Delta = f(\pi^{ib}) - C \)
11: if \( e^{-\Delta/T^c} \leq \text{rand()} \) then
12:   \( \pi^{ib} \leftarrow \pi^{ib} \)
13: end if
14: \( T^c \leftarrow T^c \times \gamma \)
15: end if
16: end for
17: return \( \pi^{ib} \)

all the tasks. The best known solution, \( \pi^{bs} \), is initialised to be empty. In Line 3, the probability distributions associated with three neighbourhood moves (discussed in the following) are initialised. The main procedure takes place between Lines 4–21, and the algorithm completes execution when the time limit is reached.

Within the main loop, the following is applied to each solution in the population. First, in Line 6, the Metropolis-Hastings algorithm is applied (Algorithm 3) using \( \pi \) as the best solution and for \( \text{Iter} \) iterations. The resulting solution is stored in \( \pi^t \), \( \pi^{bs} \) is updated to \( \pi \), if \( \pi \) is an improvement, i.e., \( f(\pi^t) < f(\pi^{bs}) \). Following this, between Lines 9–19, \( \pi \) is perturbed using a neighbourhood move (details follow in Section IV-C). As part of these steps, the probabilities associated with choosing one of the perturbation moves are adapted or updated. That is, if a move is selected, its corresponding probability reduced by an adaptation factor \( \rho = 0.9 \).

Every solution obtained (by applying any neighbourhood move, etc.) requires the computation of its objective value. As mentioned earlier, a solution is represented by a sequence of the tasks. For each sequence of tasks \( \pi \), a feasible schedule \( (\sigma(\pi)) \) is obtained by constructing \( u_s \) schedules corresponding to each uncertainty sample. Note, each schedule is resource feasible, ensuring that at most \( G_s \) resources are used at each time point for sample \( s \). The result is that each sample has its own TWT. The objective value of \( \pi \) is computed as the average of the TWTs across all samples.

B. The Metropolis-Hastings Algorithm

The Metropolis-Hastings method relies on the cooling schedule to ensure a gradual convergence to high-quality regions of the search space. However, due to the overheads introduced by uncertainty, the traditional gradual reduction in temperature will not suffice, and hence we propose a modified version of the method for RCJSU.

Algorithm 2 shows the Metropolis-Hastings procedure. It takes as input: (a) \( \pi \): current best solution, (b) \( T_0 \): initial temperature, (c) \( \text{Iter} \): number of iterations that the algorithm will execute for and (d) \( \gamma \): a parameter to determine the cooling schedule. \( C \) represents the best known objective value, and is set to a large number. The temperature \( T^c \) is set to the initial temperature \( T_0 \), and the best solution \( \pi^{ib} \) is initialised to \( \pi \). The algorithm executes for \( \text{Iter} \) iterations between Lines 3–16. Using \( \pi^{ib} \), the neighbourhood move of swapping a pair of jobs is applied. That is, two jobs within the sequence are selected at random and swapped (see Figure 2). If the new solution found \( \pi^{ib} \) is an improvement, \( C \) is updated to the objective value of \( \pi^{ib} \). Moreover, \( \pi^{ib} \) is updated to \( \pi^{ib} \) and the iteration count is reset. Otherwise, a move to the non-improving solution \( \pi^{ib} \) is still chosen with probability \( e^{-\Delta/T^c} \). The cooling schedule takes place in Line 14.

C. Adaptive Perturbation

Key to the success of APSA is perturbing the best known solution, via three neighbourhood moves. In particular, when applied carefully at different stages of the search, the overall APSA algorithm is able to achieve excellent trade-offs between intensification and diversification.

The first neighbourhood move is \( \beta \)-sampling, which was first studied in the context of project scheduling [37]. The basic idea underlying the method is that consecutive jobs or tasks with a sequence are inter-dependent, and moving whole sub-sequences (maintaining certain dependencies) can lead to significant improvements in solutions. For the purposes of the current study, starting at a randomly selected position in the sequence, a sub-sequence of fixed length (5 jobs) is moved to the back of the permutation.

Figure 1 provides an illustration of this method. \( \pi \) consists of 11 jobs. A position \( p \in 1, \ldots, n - 5 \) is chosen randomly,
The aim of the scheduling heuristic is to generate a resource feasible schedule $\sigma(\pi)$ from the sequence $\pi$. For this purpose, the serial scheduling scheme is employed [38], which is

$$\frac{dp_b}{dt} = 0.1p_b\left(\frac{p_j^2}{1 - 0.1p_j} + \frac{p_r^2}{1 - 0.1p_r} - p_b(p_j + p_r)\right),$$  \hspace{1cm} (11)

$$\frac{dp_j}{dt} = 0.1p_j\left(\frac{p_r^2}{1 - 0.1p_r} + \frac{p_b^2}{1 - 0.1p_b} - p_j(p_b + p_r)\right),$$  \hspace{1cm} (12)

$$\frac{dp_r}{dt} = 0.1p_r\left(\frac{p_b^2}{1 - 0.1p_b} + \frac{p_j^2}{1 - 0.1p_j} - p_r(p_b + p_j)\right).$$  \hspace{1cm} (13)

**Theorem 1.** For any initial condition of $p_b^0$, $p_j^0$, $p_r^0 \in [0, 1]$ and $p_b^0 + p_j^0 + p_r^0 = 1$, the probability values converge to $p_b = p_j = p_r = 1/3$.

**Proof.** At equilibrium of the differential equations (11)-(13), $\frac{dp_b}{dt} = \frac{dp_j}{dt} = \frac{dp_r}{dt} = 0$. Solving this system of equations with $p_b + p_j + p_r = 1$, we can obtain the following seven equilibria from the differential equations:

- $(p_b = 0, p_j = 0, p_r = 1)$,
- $(p_b = 0, p_j = 1, p_r = 0)$,
- $(p_b = 1, p_j = 0, p_r = 0)$,
- $(p_b = 0, p_j = 1/2, p_r = 1/2)$,
- $(p_b = 1, p_j = 0, p_r = 1)$,
- $(p_b = 1/2, p_j = 1/2, p_r = 0)$,
- $(p_b = 1/3, p_j = 1/3, p_r = 1/3)$.

However, the first six equilibria are unstable, in the sense that a small disturbance will perturb the system away from these equilibria. The last equilibrium $(p_b = 1/3, p_j = 1/3, p_r = 1/3)$ is the only stable equilibrium. Hence, after a sufficient number of time steps, the probability values will converge to this stable equilibrium.

Figure 3 shows a simulation of how the probabilities associated with each neighbourhood move (Algorithm 1) evolve across 200 simulations. We see that three probabilities converge to almost equal levels by 125 simulations.

### D. Scheduling Heuristic

The aim of the scheduling heuristic is to generate a resource feasible schedule $\sigma(\pi)$ from the sequence $\pi$. For this purpose, the serial scheduling scheme is employed [38], which is
adapted from previous studies on RCJS [4, 6, 7, 8]. Note, Thiruvady et al. [16] showed that this scheduling scheme can indeed be effective for RCJSU.

The waiting list also consists of some waiting jobs partially completed with a few jobs allocated to each machine. The height of a machine represents the amount of resources available, while a job’s height represents its resource requirement. A colour implies a job’s machine, i.e., jobs of one machine are of different colours. The memory limit was 2 GB, which proved to be sufficient for a higher overall TWTs.

Fig. 4: An example of a problem with three machines and 15 jobs. From π, jobs are selected in order (left to right) and scheduled on machines that they belong to (e.g. orange jobs on m2). Those jobs that have predecessors that are not yet scheduled will be put on the waiting list. (a) Job 8 requires Job 2 to be scheduled before it can be scheduled. (b) Job 8 is scheduled once Job 2 has been scheduled, and this will be done as early as possible considering resources.

V. EXPERIMENTAL SETTING

We investigate APSA on RCJSU, and compare its performance to current state-of-the-art approaches, including ACS and SACS. Additionally, we investigate APSA against standard simulated annealing (SA) with the same settings, and the SA of Singh and Ernst [4]. Finally, we examine APSA’s performance on the deterministic problem (RCJS), and we compare APSA to ACS, SACS and a column generation and ACO hybrid by Thiruvady et al. [7]. The experiments are conducted on problem instances from the literature. While this data was originally created for the RCJS, each problem instance is modified to generate the data for RCJSU. Specifically, there are 10 samples generated for each problem instance, where each one has resource limits in the range [U_{min}, U_1 \times G]. U_{min} is chosen to be the largest resource required by any job, i.e. \max_{j \in J} y_j, which ensures a resource feasible schedule can always be found. Additionally, the multiplier U_1 \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} is used to select a proportion of the resources. Typically, a smaller value (e.g. 0.5 or 0.6) means that the problem instance is tightly constrained, where the schedules will need larger completion times, leading to a higher overall TWTs.

The parameter settings for APSA were selected through tuning by hand and by using past studies as a guide. We refer to Algorithms 1 and 2. The time limit t_{lim} is set to 600 seconds. The initial temperature T_0 was set to 1500 [4]. Due to uncertainty and the overheads required to construct schedules with a number of samples, the parameters within the Metropolis-Hastings algorithm have been set to allow for much quicker cooling schedule. This is achieved by allowing only 1000 iterations Iter = 1000 and \gamma = 0.5. The population size was found through tuning by hand s = 10. The subsequence size used within \beta-sampling of length 5 jobs, which is obtained from [4]. Moreover, in Line 14 of Algorithm 1, there are m = rand() * n applications of job swapping, where rand() is a random number between 0 and 1. The parameter settings for ACS and SACS were obtained from [16], to which we refer the reader for complete details.

We carry out experiments on all problem instances considering all uncertainty levels. There are 25 repetitions of for each algorithm, with each one given 10 minutes of wall clock time. The memory limit was 2 GB, which proved to be sufficient for all algorithms. The algorithms were implemented in C++ with GCC-5.4.0. All experiments were carried out on Monash University’s Campus Cluster – MonARCH.

The IP was implemented in Gurobi 9.0.1 [39]. It was run for 10 minutes per problem instance and allowed a larger amount of memory of 10GB. Across all the runs, only one solution was found for one problem instance at the uncertainty of 0.8. This is not surprising given the size of the model, which is unsuitable in practical settings.

https://github.com/andreas-ernst/Mathprog-ORlib/blob/master/data/RCJS_Instances.zip consists of problem instances for RCJS. The resource usage within these instances is modified to generate problem instances for RCJSU.

https://confluence.apps.monash.edu/display/monarch/MonARCH+Home
A. Results

The results that follow are reported as a percentage difference from the average solution quality over 25 runs for each algorithm to the best solution found by any algorithm. Consider a problem instance and let \( B^* \) be the best solution obtained by any algorithm for that problem instance. Let \( A_k \) be the best solution found by APSA for instance \( k \). Then the percentage difference for APSA is \( \frac{A_k - B^*}{A_k} \).

Tables II and III show the results of ACS, SACS and APSA for low and high uncertainty levels \( UL \), respectively. The tables show the best result obtained by any algorithm (which is highlighted in bold if it is found by APSA) for each uncertainty level, followed by the percentage differences of each algorithm to the corresponding best solution. The best result found by any algorithm is highlighted in bold, and those results that are statistically significant (determined using a Wilcoxon ranked-sum test) are marked with a ‘*’.

To summarise the performance of the algorithms across the problem instances, the last row of the tables report the number of times (# best) the algorithm finds the best solution.

The results in both tables show that APSA finds best known solutions for all problem instances and uncertainty levels (except instance 15–2 at uncertainty level 0.9), and the best average results across all problem instances for all uncertainty levels. Looking closely, we see that the differences between the algorithms are smallest for the smallest problems (3 machines) and largest problems (20 machines). Everywhere else, APSA has significant advantages. For the problem instances with 3 machines, this is not surprising, because it is expected that all algorithms find solutions that are close to optimal. However, APSA has the advantage that, for every run, it finds a solution close to the best. On investigating the individual runs on the large problem instances, we find that there are relatively few iterations conducted (< 10). This leads to APSA not converging sufficiently well, thereby not finding as large improvements are seen on other smaller problem instances.

The results in Tables II and III are summarised in Figure 6. The figure shows the performance of each algorithm (ACS, SACS and APSA) by uncertainty levels. The bars represent the average of percentage difference to best across all instances in an uncertainty level. As expected, APSA clearly outperforms the other two methods, with a slight advantage with increasing uncertainty levels.

B. Convergence Characteristics of APSA

APSA outperforms SACS and ACS across all problem instances for all uncertainty levels, due to its ability to diversify well. In particular, APSA’s ability to stop getting stuck in local optima leads to substantial gains.

In the following, we examine the convergence of APSA, and compare it to ACS and SACS. We choose four problem instances, which are sufficiently diverse in size to allow making somewhat general inferences about APSA. The results are shown in Figure 6, where each sub-figure corresponds to one problem instance. The plots show time in seconds on the x-axis and TWT (logarithmic scale) on the y-axis.

C. Comparisons of APSA to the original SA (OSA)

The original SA (OSA) algorithm [4] can be adapted to tackle the RCJSU problem. The main difference is that every time a new solution is created, a new objective needs to be computed, which involves creating multiple schedules corresponding to the samples. Otherwise, all the settings for the algorithm remain the same.

Table IV shows the results for APSA and OSA considering the uncertainty levels 0.6 and 0.9. Other than the smallest problem instances (3 machines), APSA clearly is superior, finding better average solutions (> 85%) across both uncertainty levels.

D. Comparison of APSA with SA

The novel components of APSA including the population of solutions, adaptive probabilistic selection and quicker cooling schedule clearly lead to significant gains over traditional OSA. In this section, we aim to understand the impacts of the population on APSA. This essentially involves an implementation of APSA with a population size of 1.

The results can be seen in Table V, where comparisons are made for the uncertainty levels 0.6 and 0.9. We see that there is a clear advantage of the population, and the advantage increases with the higher uncertainty levels.

*Full details of the algorithm can be found in [4], and due to space limitations, we do not provide the full algorithm or its description here.
Table II: Results of ACS, SACS and APSA shown as the percentage difference to the best solution for uncertainty levels 0.5, 0.6 and 0.7; Results in column “Best” are highlighted in bold if APSA finds the best solution; Results for each algorithm are highlighted in bold when they achieve best average results, while statistically significant results (pair-wise Wilcoxon ranked-sum test) at a 95% confidence interval are marked with a *.

| Algorithm | 0.5 | 0.6 | 0.7 |
|-----------|-----|-----|-----|
| Best | | | |
| ACS | | | |
| SACS | | | |
| APSA | | | |

Table III: Results of ACS, SACS and APSA shown as the percentage difference to the best solution for uncertainty levels 0.8, 0.9 and 1.0; Results in column “Best” are highlighted in bold if APSA finds the best solution; Results for each algorithm are highlighted in bold when they achieve best average results, while statistically significant results (pair-wise Wilcoxon ranked-sum test) at a 95% confidence interval are marked with a *.
Fig. 6: Convergence profiles of ACS, SACS and APSA on four RCJSU problem instances of different sizes. The instances chosen are 3–53, 6–28, 9–47 and 12–14. The TWTs on the y-axes are presented on a logarithmic scale.

E. Comparisons on the Deterministic Problem

APSA’s advantages obtained from the population, a quicker cooling schedule and adaptive probabilistic selection for the perturbation of solutions, make for an effective algorithm in the uncertain setting. Its performance on the deterministic problem or RCJS is worth investigating, to see if any of the novelties proposed in the algorithm can be beneficial for the deterministic problem. The results are shown in Table VI where APSA to ACS, SACS, and a column generation and ACO hybrid (CGACO) are compared.

CGACO is also run 25 times on each problem instance. A single sample was used for ACS, SACS and APSA, leading to the original resource limit (effectively RCJS). We see in Table VI that beyond small problem instances (> 4 machines), APSA has a clear advantage. It generally performs best, but without the same large gains seen on RCJSU.

VI. CONCLUSION

This study investigates an optimisation problem that originates in the mining industry, specifically transporting ore from mines to ports. Past studies have considered the problem as deterministic, however, in real settings, the resources needed to transport the ore are subject to uncertainty. The problem is also called resource constrained job scheduling with uncertainty, where for this study, we consider multiple uncertain scenarios. To tackle this problem, we propose an adaptive population-based simulated approach, which has significant advantages over previously proposed ACO based approaches. We find that using a population, increasing the rate of the cooling schedule within the Metropolis-Hastings algorithm and adaptively selecting intensification and diversification via the neighbourhood moves, proves advantageous. These modifications, compared to traditional simulated annealing, lead to finding robust solutions, and hence, best results on a benchmark dataset for RCJSU.

Despite the success of APSA on RCJSU, investigating how the approach performs on other optimisation problems with uncertainty will be of interest in future work. In particular, it is expected an adaptation of APSA is likely to be effective for problems that consist of highly constrained disjoint feasible regions, such as RCJSU.

The MIP model in Section III is clearly too complex and large for existing solvers, but decomposition approaches such as column generation or Benders’ decomposition may prove effective. The advantage of such methods is that they can provide guarantees of solution quality (via lower bounds to RCJSU), which can help to measure the quality of the solutions found by APSA.

REFERENCES

[1] M. C. of Australia, “Mining is the Largest Contributor to Australian Economy in 2019-20,” 2020, available from: https://www.minerals.org.au/news/mining-largest-contributor-australian-economy-2019-20.
TABLE IV: A comparison of PSA and the original SA (SAO) for the uncertainty levels of 0.6 and 0.9.

| Uncertainty | PSA | SAO |
|-------------|-----|-----|
| 0.6         |     |     |
| Best        |     |     |
| 1-8         | 942.64 | 949.18 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2-12        | 232.48 | 232.24 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3-20        | 213.37 | 213.25 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4-22        | 127.92 | 113.26 | 0.02 | 0.00 | 0.12 | 0.00 |
| 5-17        | 361.12 | 360.61 | 0.04 | 0.00 | 0.12 | 0.00 |
| 6-14        | 123.24 | 123.25 | 0.01 | 0.00 | 0.12 | 0.00 |
| 7-12        | 373.31 | 362.89 | 0.07 | 0.00 | 0.12 | 0.00 |
| 8-16        | 533.80 | 488.18 | 0.14 | 0.00 |
| 9-18        | 688.01 | 645.89 | 0.12 | 0.00 |
| 10-20       | 2026.29 | 1404.63 | 1.10 | 2.96 |
| 11-25       | 762.44 | 502.18 | 1.30 | 2.53 |
| 12-28       | 798.97 | 517.45 | 1.01 | 3.37 |
| 13-31       | 1025.49 | 1277.61 | 0.27 | 1.87 |
| 14-35       | 1194.27 | 1166.58 | 1.21 | 2.60 |
| 15-39       | 1191.50 | 1493.28 | 1.05 | 4.49 |
| 16-43       | 1995.45 | 1740.71 | 1.92 | 5.11 |
| 17-47       | 1250.50 | 1107.35 | 1.52 | 3.26 |
| 18-51       | 2971.52 | 2669.98 | 1.16 | 2.52 |

| 0.9         |     |     |
| Best        |     |     |
| 1-8         | 942.64 | 949.18 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2-12        | 232.48 | 232.24 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3-20        | 213.37 | 213.25 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4-22        | 127.92 | 113.26 | 0.02 | 0.00 | 0.12 | 0.00 |
| 5-17        | 361.12 | 360.61 | 0.04 | 0.00 | 0.12 | 0.00 |
| 6-14        | 123.24 | 123.25 | 0.01 | 0.00 | 0.12 | 0.00 |
| 7-12        | 373.31 | 362.89 | 0.07 | 0.00 | 0.12 | 0.00 |
| 8-16        | 533.80 | 488.18 | 0.14 | 0.00 |
| 9-18        | 688.01 | 645.89 | 0.12 | 0.00 |
| 10-20       | 2026.29 | 1404.63 | 1.10 | 2.96 |
| 11-25       | 762.44 | 502.18 | 1.30 | 2.53 |
| 12-28       | 798.97 | 517.45 | 1.01 | 3.37 |
| 13-31       | 1025.49 | 1277.61 | 0.27 | 1.87 |
| 14-35       | 1194.27 | 1166.58 | 1.21 | 2.60 |
| 15-39       | 1191.50 | 1493.28 | 1.05 | 4.49 |
| 16-43       | 1995.45 | 1740.71 | 1.92 | 5.11 |
| 17-47       | 1250.50 | 1107.35 | 1.52 | 3.26 |
| 18-51       | 2971.52 | 2669.98 | 1.16 | 2.52 |

TABLE V: A comparison of PSA and SA for the uncertainty levels of 0.6 and 0.9.

| Uncertainty | PSA | SA |
|-------------|-----|----|
| 0.6         |     |    |
| Best        |     |    |
| 1-8         | 437.29 | 2998.54 | 1.81 | 3.21 |
| 2-12        | 6821.22 | 4923.86 | 1.25 | 2.52 |
| 3-20        | 5310.08 | 3754.28 | 2.30 | 3.13 |
| 4-22        | 7554.10 | 7144.36 | 1.64 | 2.04 |
| 5-17        | 8585.22 | 8037.26 | 1.45 | 1.41 |
| 6-14        | 7021.04 | 6473.33 | 3.05 | 3.68 |
| 7-12        | 1381.10 | 1380.81 | 1.03 | 2.21 |
| 8-16        | 23173.50 | 23158.44 | 2.59 | 2.98 |
| 9-18        | 12713.00 | 11895.5 | 1.17 | 0.10 |

| 0.9         |     |    |
| Best        |     |    |
| 1-8         | 437.29 | 2998.54 | 1.81 | 3.21 |
| 2-12        | 6821.22 | 4923.86 | 1.25 | 2.52 |
| 3-20        | 5310.08 | 3754.28 | 2.30 | 3.13 |
| 4-22        | 7554.10 | 7144.36 | 1.64 | 2.04 |
| 5-17        | 8585.22 | 8037.26 | 1.45 | 1.41 |
| 6-14        | 7021.04 | 6473.33 | 3.05 | 3.68 |
| 7-12        | 1381.10 | 1380.81 | 1.03 | 2.21 |
| 8-16        | 23173.50 | 23158.44 | 2.59 | 2.98 |
| 9-18        | 12713.00 | 11895.5 | 1.17 | 0.10 |
### TABLE VI: A comparison of CGACO, ACS, SACS and PSA on RCJS. Statistical significance was obtained using a pair-wise Wilcoxon ranked-sum test, with significant results at a 95% confidence interval highlighted in bold.

| Instance | Best | Mean | SD | Best | Mean | SD | Best | Mean | SD | Best | Mean | SD |
|----------|------|------|----|------|------|----|------|------|----|------|------|----|
| 3 - 5    | 515.68 | 520.30 | 6.16 | 505.00 | 505.00 | 0.00 | 505.00 | 505.00 | 0.00 | 505.00 | 505.00 | 0.00 |
| 3 - 6    | 149.07 | 150.04 | 1.09 | 149.07 | 149.09 | 0.08 | 149.07 | 149.11 | 0.12 | 149.07 | 149.07 | 0.00 |
| 3 - 7    | 69.36 | 69.36 | 0.00 | 69.36 | 69.36 | 0.00 | 69.36 | 69.36 | 0.00 | 69.36 | 69.36 | 0.00 |
| 3 - 8    | 23.90 | 23.90 | 0.00 | 23.90 | 24.04 | 0.09 | 23.94 | 24.00 | 0.00 | 23.81 | 23.81 | 0.00 |
| 4 - 5    | 68.59 | 71.26 | 3.01 | 67.73 | 67.73 | 0.00 | 67.73 | 67.73 | 0.00 | 67.07 | 67.06 | 0.33 |
| 6 - 6    | 47.69 | 47.98 | 0.23 | 45.96 | 45.96 | 0.00 | 45.96 | 45.96 | 0.00 | 45.96 | 45.96 | 0.00 |
| 5 - 6    | 260.15 | 275.00 | 17.65 | 256.97 | 267.40 | 3.43 | 256.69 | 262.81 | 4.29 | 252.90 | 253.35 | 0.17 |
| 6 - 7    | 177.27 | 177.27 | 0.00 | 165.18 | 166.21 | 0.92 | 168.63 | 168.79 | 0.39 | 168.63 | 168.63 | 0.00 |
| 6 - 2    | 264.99 | 276.22 | 13.75 | 259.19 | 266.35 | 5.17 | 259.12 | 266.37 | 5.43 | 250.51 | 252.63 | 2.62 |
| 6 - 10   | 1031.40 | 1031.40 | 0.00 | 874.39 | 891.58 | 9.87 | 870.83 | 887.66 | 8.29 | 813.21 | 843.86 | 6.11 |
| 6 - 28   | 241.28 | 241.28 | 0.00 | 230.04 | 239.48 | 3.35 | 255.53 | 264.04 | 5.87 | 218.37 | 222.83 | 1.91 |
| 6 - 58   | 264.33 | 268.87 | 4.72 | 249.01 | 255.63 | 3.34 | 241.04 | 251.53 | 4.52 | 238.84 | 240.65 | 0.60 |
| 7 - 5    | 486.17 | 502.07 | 20.37 | 445.18 | 460.07 | 6.51 | 445.92 | 459.88 | 5.91 | 413.08 | 426.25 | 7.87 |
| 7 - 23   | 655.52 | 658.40 | 2.55 | 587.96 | 604.99 | 10.14 | 577.88 | 595.26 | 8.20 | 553.91 | 570.05 | 8.24 |
| 7 - 47   | 543.82 | 547.21 | 11.49 | 472.24 | 502.76 | 12.45 | 470.42 | 497.73 | 11.80 | 428.51 | 453.82 | 12.17 |
| 8 - 7    | 911.22 | 1010.67 | 17.72 | 725.49 | 768.23 | 15.77 | 721.80 | 746.00 | 14.32 | 692.92 | 669.74 | 17.23 |
| 8 - 53   | 522.59 | 522.58 | 0.00 | 494.32 | 507.07 | 6.56 | 487.27 | 501.04 | 6.54 | 455.45 | 470.12 | 7.38 |
| 8 - 77   | 1630.28 | 1648.35 | 67.75 | 1281.98 | 1311.84 | 14.88 | 1264.41 | 1298.44 | 14.54 | 1208.54 | 1235.36 | 14.73 |

### 2.4.1 Confidence Interval Analysis

The confidence interval highlighted in bold indicates a significant difference in performance between the algorithms. Further analysis using a pair-wise Wilcoxon ranked-sum test confirmed the statistical significance of these results.