# A Flexible Job Shop Scheduling Approach with Operators for Coal Export Terminals

Robert L. Burdett[1], Paul Corry[2], Prasad K.D.V. Yarlagada[3],Colin Eustace[4], Simon Smith[5]
[1,2] School of Mathematical Sciences, Queensland University of Technology,
[3]School of Chemistry, Physics and Mechanical Engineering, Queensland University of Technology
[4,5]AURECON Australasia, Level 14, 32 Turbot Street, Brisbane, Qld, 4000

**Abstract:** Operating a coal export terminal (CET) is a challenging task. The throughput of the terminal and the delays incurred by ships and trains is greatly affected by where material is stacked and reclaimed, and which machinery is used to perform those tasks. To improve the operation of these terminals, and to make the aforementioned decisions, a novel optimisation approach has been developed. In that approach the CET is modelled as a flexible job shop with operators (FJSOP). The optimization problem is then solved using an advanced meta-heuristic algorithm that incorporates a variety of sophisticated perturbation techniques, local improvement algorithms and pre-emption handling procedures. That level of intricacy is rarely required for more traditional scheduling problems but is a necessity for this one. A key component used in the optimisation process is a priority ordering of the activities. The priority ordering is used to create the required activity sequences, for each piece of machinery. This list dictates the order in which activities are scheduled and can be perturbed quickly and in many different ways, leading to the identification of improved solutions. The optimization approach is applied to an Australian coal terminal to demonstrate its viability and effectiveness. Numerical testing shows that problems of the size encountered by existing terminals can be handled and high quality solutions can be obtained with reasonable computing effort.

**Keywords:** Bulk material handling systems, coal export terminals, scheduling, flexible job shop with operators, meta-heuristics, priority ordering

## 1. Introduction

### 1.1 Background

Bulk material handling terminals are used throughout the world to facilitate the import and export of various types of dry bulk material, such as iron ore and coal. They provide a place to store large amounts of material, and a way to quickly reclaim that material when ships and other forms of transportation arrive to collect it. They provide loading and unloading facilities and a stockyard to assemble parcels of bulk material for export or to receive import cargo. Export material is reclaimed from stockpiles and loaded onto ships and transported to other locations where it is further processed.

This articles focus is upon coal export terminals (CET) however, iron ore terminals have similar infrastructure and are also highly relevant to this focus. Presently there are over 100 CET operating globally. In Australia ten CET are currently operating and these are positioned on the coast of Queensland and New South Wales within reach of inland mining sites. The ten CET jointly handle approximately 500 million tonnes of coal with a value of over 50 billion dollars per year.

Coal export terminals operate a variety of different equipment and machinery. These will be referred to as resources, in the remainder of this article. A schematic layout for a typical export CET is shown in Figure 1. The CET includes a stockyard with a rectangular array for stockpiles. Each stockpile is separate and has a different size, geometry and product that must be kept separate from other products. A typical CET also includes train unloading facilities, often referred to as dump-stations for trains to deposit material, and a number of berths for ships to dock. Yard machines are used to stack and reclaim material from stockpiles. These can be separate machines (i.e. stackers, reclaimers) or machines capable of performing both functions, namely stacker-reclaimers (SR). The type of yard machines considered here are long travelling, slewing and luffing SRs. They move adjacent to stockpiles and conveyers. They are installed on rails and can only be moved between specific stockpiles accessible in adjacent rows (a.k.a. pads) either side of the bund. Conveyers take material to and from stacker-reclaimers. Ship loaders (*SL*) are similar to stackers; they are used to fill ships and can be moved along the wharf between hatches on a ship and between the different berths. High volume CET will typically receive material from trains. A train will deliver material into a dump station, with the material then transferred through a series of conveyors to a stacker. Transfer towers act as switches to route material between connecting conveyors. Surge bins are used as a buffer between reclaiming and ship-loading. They minimise delays to reclaiming during hatch changes and to allow ship loaders to operate at a higher rate than reclaimers, while there is sufficient product in the bin.

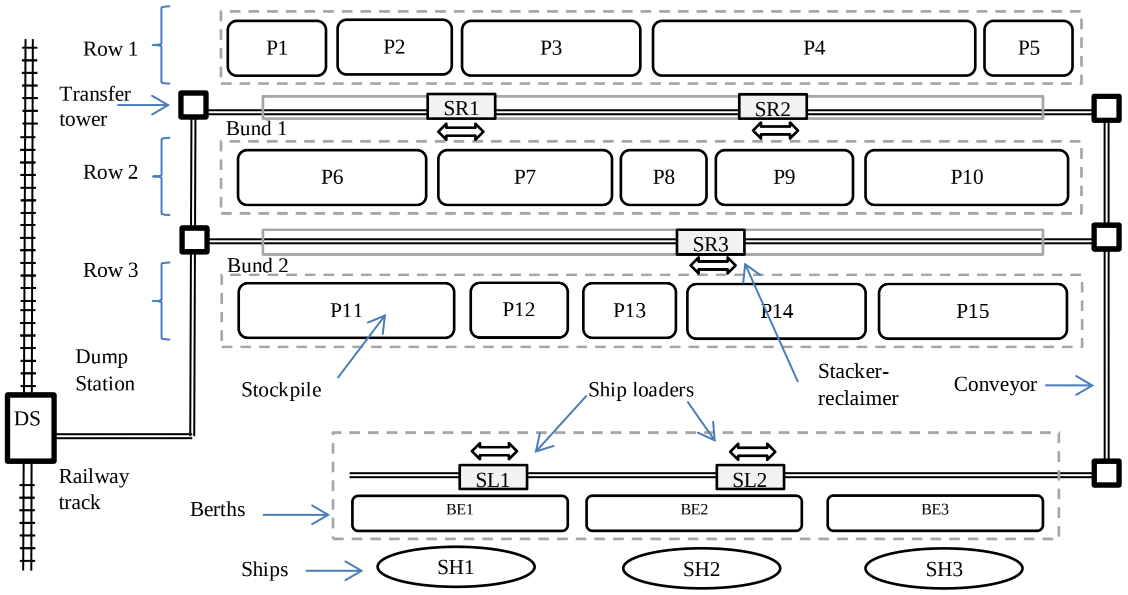

**Figure 1.** Layout of a typical CET

When operating a CET a number of important decision problems arise. In this article, an integrated sequencing and scheduling problem is considered. In that problem, there is a set of arriving ships ($SH$) and trains ($TR$); the trains bring material to the terminal and the ships collect it for export. The arrival time of the trains and ships is predictable in some circumstances, and uncertain in others. In this article we will consider two scenarios. In the first scenario (Variant 1), trains and ships will arrive at prescribed times. In the second scenario (Variant 2), trains and ships can arrive at any time. In other words, the arrival times are decisions. A third scenario (Variant 3) occurs when train and ship arrivals are random variables with a planned arrival time. That variant will not be considered in this article, but in future ones.

Regardless of the variant, the primary consideration is to determine when loading, unloading, stacking and reclaiming activities are to be performed and how those activities should be resourced from the available terminal equipment. In other words, it is necessary to choose where material is stacked after it is unloaded from trains, and which dump station and stacker-reclaimer to use. When loading ships, it is necessary to choose where material is reclaimed and which stacker-reclaimer to use. This is a sequencing problem, because the activities' start and end time is inherently linked to their position within the sequence of activities designated to each piece of equipment (i.e. the resource sequences). The activities present in each resource sequence is evidently governed by the aforementioned resource allocations. It is worth pointing out that the berth used by each ship is also a potential decision. This position is important because it affects the type of ship loaders that may be used to load a ship. Some ship loaders can only be used at one berth, while others can service several different berths. Other relevant information for sequencing and scheduling is as follows:

- Each ship has a number of holds, each potentially holding a different product. The ship requirements can be met over the scheduling period. The order in which products are to be loaded can be important and will be specified if so.
- A ship may require product from multiple stockpiles as one stockpile may not have enough material.
- Products are to be placed in specific stockpiles; each stockpile is for a product of a given type. It is common to have two stockpiles of a given product type, sometimes more depending on the terminal.
- Each train has a number of wagons and each wagon may contain a different product. We will consider trains that hold one type as this is the typical situation encountered in Australian CETs.
- In some scenarios the arrival time of trains and ships is known in advance. In other scenarios those times are decision variables and need to be identified.
- Time to stack and reclaim is a function of the tonnage. A constant stack and reclaim rate will initially be assumed. The ship loaders will operate at a constant rate, higher than the reclaim rate, but with interruptions for hatch changes.
- The maximum number of concurrent stacking activities is limited by the number of dump-stations.

The CET scheduling problem is challenging for the following reasons:

- Multiple pieces of equipment (i.e. resources) are required for each activity and they must be acquired at the same time.
- Some required resources are static while others are mobile. Mobile resources must be repositioned continuously, and the repositioning time must be taken into account
- It is permissible to pre-empt the stacking and/or reclaiming activities in some CET. This means that they can be paused and resumed at later times with the same or different terminal equipment. How best to partition an activity into pieces and when to schedule those pieces is a significant dilemma.
- Stockpile sizes vary after each stack and reclaim activity. Stockpiles must never be in deficit, nor can they be overloaded. Ensuring that the correct stockpile is assigned to each stack and reclaim activity is tricky.
- Some piles may be "resumed" during the scheduling period and replacement piles of a different product may be introduced.

## 1.2. Research Aims

It is important to operate CET effectively and to maximise their potential throughput. By more efficiently utilising their existing equipment, it may be possible, to reduce idle time and the delays to trains and ships. To achieve improved performance, advanced automated scheduling systems are likely to be of benefit, particularly as terminal throughput approaches capacity. As CETs contain unique features that make planning and scheduling difficult, existing planning and scheduling approaches have not been automated. Traditionally they have been performed manually based on experience and mental reckoning, supported by Excel spreadsheets or commercial software, like QUINTIQ (2017), that facilitates manual planning and the visualisation of data. Schedule optimisation has great potential given the challenges of manually scheduling multiple concurrent activities competing for multiple shared resources with complex consequences for future operations.

The purpose of this article is to test the premise that a generic abstraction of the terminal scheduling problem based upon a flexible job shop scheduling approach with operators (i.e. FJSOP) may be helpful and could provide schedules that lead to improved efficiency. Evidently the job-shop scheduling approach is robust and highly generic and elegantly describes the predominant features of many real world scheduling problems. In recent years, however it has also been successively adapted for a diverse range of applications with bespoke characteristics, for instance train scheduling (Burdett and Kozan, 2010) and hospital scheduling (Burdett and Kozan, 2018).

This article investigates the technical details behind the FJSOP approach in the context of CETs. The intention is to provide an approach that can be embedded within an integrated rolling horizon planning and rescheduling framework within an intelligent information management system (IIMS). To our knowledge no other paper has explicitly addressed this problem in a comparable level of detail or provided a similar generic approach. A traditional analytical approach based around the formulation and solution of an integer programming model could have been applied, however this problem is highly combinatorial, and as such, is better suited to machine scheduling and resource constrained project scheduling style approaches. It is unlikely that this problem can be solved to optimality with an analytical approach, as the search space is so large and complex.

The FJSOP is a new framework (Agnetis et al., 2014) and has only been applied to a small number of environments. It has significant potential and general applicability. The scheduling environment we are considering is fundamental and our approach, excluding features specific to CET, is applicable to problems in other domains like hospital scheduling (Burdett and Kozan, 2018). In hospital scheduling, patients are assigned bed spaces and other medical and surgical resources from a set of candidates. The patients are treated in various locations within the hospital and each surgical or medical activity requires multiple resources to be present simultaneously. That requirement is also paramount in CETs.

## 1.3. Past Research

Recent developments and trends in scheduling are now discussed. Many of these papers have motivated our current approach. Integrated scheduling of activities within CETs is a relatively new topic. To our knowledge it has not received much direct attention. Hu and Yao (2012), Babu et al. (2015), Kalinowski et al. (2017) are recent articles on CET scheduling. Babu et al. (2015) studied an integrated port and stockyard operations problem and proposed two greedy constructive algorithms. Their model consists of ship berthing, stockyard planning and train scheduling. They did not consider multiple berth and capacity constraints of SR and SL. Hu and Yao (2012) considered the scheduling of stacker reclaimers so that the maximum completion time of tasks is minimised. A mixed integer programming model was developed and solved using Genetic Algorithms. No pre-emption was included in their approach and stockpile sizes are not updated over time. In contrast our FJSOP approach does. Boland et al. (2012) developed a stockyard planning approach for cargo assembly terminals. A mathematical programming formulation was developed and solved using constructive heuristics. The objective of their model is minimization of the mean ship delay. Kalinowski et al. (2017) focused upon the scheduling of reclaiming activities as reclaiming capacity for the application of interest was lower than stacking capacity. They formulated a mixed integer programming model. The movement of reclaimers between stockpiles was not considered, but it is in this article.

Coal export terminals can be modelled as a type of flexible job shop. The flexible job shop scheduling problem (FJSP) is an extension of the classical job shop (JSP) that permits activities to be processed by a variety of candidate resources. In the FJSP, the goal is to assign each activity to a single resource, and to sequence those activities, so that a given scheduling criterion is optimized. The FJSP is a notoriously difficult combinatorial optimization problem. It is known to be NP-hard, yet is far simpler than our situation which involves multiple resource types for each activity and pre-emptions. In recent years the FJSP has been addressed in Xia and Wu (2005), Pezella et al. (2008), Junwattanakit et al. (2009), Zhang et al. (2009), Yazdani et al. (2010), Groflin et al. (2011), Chen et al. (2012), Doh et al. (2013), Gao et al. (2014, 2015). In those articles multiple objectives have been included and advanced meta-heuristics have been developed. Gao et al. (2015) also considers job reinsertions. A more complex variant with multiple process plans is considered by Doh et al. (2013). They propose a priority scheduling approach that utilizes two priority rules. Although quite practical, their approach is just a heuristic and does not target optimal solutions, nor can it be used to refine an existing schedule. In retrospection, past research highlights the need to separate the machine allocation decisions from the operations scheduling decisions, and to solve one sub problem before another. It is interesting to note that FJSP with pre-emptions is rarely considered (Zhang and Yang 2016). Hence our article, which does, is somewhat atypical.

It is important to mention that the requirement for each activity to be assisted by a single human operator has resulted in the creation of a variant problem, namely the job shop scheduling problem with operators (JSOP). It is also known to be NP-hard. In the JSOP there are a finite number of human operators that can be assigned. The JSOP has been addressed in only a few articles, namely Agnetis et al. (2011, 2014), Escamilla et al. (2012), Mencia et al. (2015). Agnetis et al. (2011) introduced this problem originally. Escamilla et al. (2012) introduced a three stage process to provide robust solutions to the JSOP. Human operators are ignored initially and the underlying JSP problem is solved. Operators are then introduced and buffers are placed in the operators sequences. Buffers are placed among the operations not involved in a critical path. Agnetis et al. (2014) later modelled operators with different experience levels and applied heuristics to solve the problem. Mencia et al. (2015) developed Memetic algorithms to solve the JSOP and introduced a new neighbourhood structure. To the best of our knowledge, no more than two operator types have ever been considered. In this article we consider four.

Our problem is an extension of the FJSP and JSOP and is evidently NP-hard. We denote this extension as FJSOP. The FJSOP however does not capture fully the complexities of CET scheduling. The incorporation of many additional advanced scheduling constraints is a necessity in this research. In recent years advanced scheduling constraints have successfully been added to many traditional job shop and flow shops. Noteworthy examples include Burdett and Kozan (2009a, 2009b, 2009c, 2010a, 2010b, 2018), Corry and Kozan (2004)), Groflin et al. (2011), Zeng et al. (2014), Zeng et al. (2014), Sauvey and Trabelsi (2015). Burdett and Kozan (2018) established a generic FJSP scheduling framework that can be used for a wide range of realistic scheduling environments. In their approach activity and machine setup times, transfer times between activities, blocking limitations and no wait conditions, timing and occupancy restrictions, buffering for robustness, fixed activities and sequences, release times and strict deadlines have all been included. The approach taken in this article has been motivated by the work of Burdett and Kozan (2018). Our approach however is an improvement, as we include additional technical conditions specific to CET, for instance, resourcing restrictions, pile sizing and collision detection. We also include pre-emptions and operators; these were not considered previously.

The remainder of the paper is organized as follows. In Section 2 the main technical developments and details of the FJSOP approach are introduced. For instance we discuss how to schedule activities using a disjunctive graph, we discuss restrictions that affect resourcing, we discuss how pile sizes are computed and how violations are identified and penalised, how activities are pre-empted and the effect of fixed sequences that may not be altered. An advanced meta-heuristic technique is discussed in Section 3. Therein a description of different perturbation and refinement operators is provided and the solution representation used. A case study and numerical investigations are then reported in Section 4. Concluding remarks are finally made in Section 5.

## 2. The CET Scheduling Framework

This section introduces a generic scheduling framework for CET. As a starting point it is first necessary to formally define the different components of a CET. A CET is described by the following tuple: $(P, ROW, BUND, SR, SL, B, DS)$. Each stockpile $p \in P$ has an id, size, maximum size, location and row, i.e. $p = \big(id, lv, \overline{lv}, (left, right), row\big)$. Stacker-reclaimers $sr \in SR$ have an id, stack rate, reclaim rate, speed and are positioned on a particular bund, i.e. $sr = (id, \lambda^{+}, \lambda^{-}, spd, bund)$. Ship-loaders $sl \in SL$ have an id, loading rate and a speed, i.e. $sl = (id, \lambda^{+}, spd) \in SL$. Each bund $bund \in BUND$ has an id and a list of the rows that are immediately accessible, i.e. $bund = (id, rows)$. Each berth $b \in B$ has an id and a location, i.e. $b = (id, (left, right))$. Each of the aforementioned sets describes a particular type of resource. These can be grouped as static, or mobile, i.e. $R^{\mathrm{s}} = P \cup B \cup DS \cup BUND \cup ROW$ and $R^{\mathrm{m}} = SR \cup SL$.

The CET can be modelled as a flexible job shop with operators (FJSOP). The FJSOP is formally defined as follows. A given set of jobs $J = \{J_1, J_2, \dots\}$ must be processed. Each job $J_i \in J$ has a list of activities and an activity precedence network. We denote these respectively as $J_i = (id, A_i, PRE_i)$ where $PRE_i = \{(a, a') | a, a' \in A_i\}$. The details present in $PRE_i$ define specific precedence relations of the form $a \prec a'$. These precedence relations describe which activities should be performed serially or concurrently. Each activity $a \in A_i$ has an id, type, list of required resources (i.e. requisites) and a size, i.e. $a = (id, \psi, C, sz, p)$. The type and size are problem specific and will be discussed in due course. Each activity $a \in A_i$ has a processing time $p_a$ measured in minutes. That processing time depends on the resources assigned or else is defined upfront as static. An activity may require a number of different resource types. For each resource type there may be multiple options. The complete set of feasible resource selections for each activity can be quite extensive. We denote $C_a[z]$ as the zth set of candidate resources for an activity $a$. It is worth pointing out that $C_a[z] \subset R$ and $C_a = (C_a[1], C_a[2], \dots)$.

For this problem we need to identify three things: a set of sequences, a set of resource assignments and a set of activity timings. The activity timings are denoted by $start_a$ and $end_a$. The set of resources assigned to each activity is denoted as $R_a = (R_a[1], R_a[2], \dots)$. Evidently, $R_a \subset R$ and $R_a[z] \in C_a[z]$. For instance if $C_a[1] = \{SR1, SR2\}$, $C_a[2] = \{P1, P2, P3\}$ then $R_a = (SR1, P2)$ is a valid resource selection because $SR1 \in C_a[1]$ and $P2 \in C_a[2]$. The sequence of activities assigned to resource $r$ is denoted by $\sigma_r$. The kth activity is hence $\sigma_r[k]$. It is important to point out that an activity can occur in multiple sequences when multiple resources are needed to perform it. For instance if $R_a = (r1, r2, r3)$ then $\exists k1, k2, k3 | \sigma_{r1}[k1] = \sigma_{r2}[k2] = \sigma_{r3}[k3] = a$.

Specific details about CET are now explained. The first thing to point out is that trains and ships are deemed jobs with one or more independent stacking and reclaiming activities. The type and quantity of material present on trains and required by ships is known. It is sufficient to define a single activity for each product held by trains and ships. Each activity requires a set of resources in order to be performed. Evidently some of the resources are mobile, and some are static. Stockpiles for instance are static resources where stacking and reclaiming activities occur. Stacker-reclaimers and ship loaders are mobile resources and are regarded as "operators". They can be repositioned to various locations, depending upon the structure of the CET. The trains and ships are not modelled as resources per se, but could be. They are assumed to be present whenever an activity of a particular type is performed. Different activities may be modelled and we suggest the following:

- **STACK:** This activity describes train unloading and the subsequent movement of material to a stockpile. It requires a stacker-reclaimer, a stockpile, and a dump station requisite.
- **RECLAIM:** This activity describes the retrieval of material from a stockpile and its subsequent movement to a berth. It requires a stacker-reclaimer, a stockpile, and a berth requisite. The berth requisite ensures that reclaiming activities do not begin until a ship has berthed and is ready to load. It may be unnecessary to explicitly model berths in scenarios where ship loaders do not switch between berths.
- **LOAD:** This activity describes the movement of material into a ship and requires a berth, ship loader and a stockpile requisite. Load activities are tightly coupled to reclaim activities. Consequently a combined "RECLAIM_LOAD" activity is defined.
- **RECLAIM_LOAD:** This activity combines RECLAIM and LOAD activities into one and makes the former activities superfluous. This combined activity avoids a number of problems, the first and foremost being the need to have overlapping synchronised activities of a job. It also bypasses the issue of simultaneously having to pre-empt a LOAD activity if the RECLAIM activity is pre-

empted or vice versa. This activity requires four resources, namely two mobile resources (i.e. a SR and a SL) and two static locations (i.e. a berth and a stockpile).

These activities are denoted by the following sets: $(STK, REC, LD, RECLD)$. Other activities may be necessary and can easily be incorporated.

Every activity $a \in A$ must have at least one static resource. This resource is described as the location of the activity, namely $l_a$. If $a \in STK$ or $a \in REC$ then $l_a \in P$. If $a \in LOAD$ then $l_a \in B$. For $RECLD$ activities however, there are two locations. When referring to the reclaiming location, $l_a \in P$. If referring to the load location, $l_a \in B$.

If the processing time of CET activities is not fixed, then it is computed dynamically as $p_a = 60(sz_a/rate_a)$. The rate measured in tonnes per hour, is governed by the resources assigned, i.e. $rate_a = \mathbf{F}(R_a)$. As there are different types of activities, this function has a number of clauses:

$$\mathbf{F}(R_a) = \begin{cases} \lambda_r^- \mid a \in REC, r \in R_a \cap SR \\ \lambda_r^+ \mid a \in STK, r \in R_a \cap SR \\ \lambda_r^+ \mid a \in LD, r \in R_a \cap SL \\ \min(\lambda_r^-, \lambda_{r'}^+) \mid a \in RECLD, r \in R_a \cap SR, r' \in R_a \cap SL \end{cases} \tag{1}$$

As mentioned earlier, it is necessary to determine the activity timings, resources sequences and resource assignments that optimize the performance. The performance however may be judged in many different ways. A sensible scheduling objective is to process the arriving trains and ships as quickly as possible and to minimise activities weighted flow-time. This is calculated in the following way:

$$FlowTime = \omega_1 \textstyle\sum_{J_i \in TR} flow_i + \omega_2 \sum_{J_i \in SH} flow_i \tag{2}$$
$$flow_i = \max_{\forall a \in A_i}(end_a) - rlt_i \quad \forall J_i \in J \tag{3}$$

The following parameters are reflective of practical considerations $(\omega_1, \omega_2) = (10,1)$ and prioritize the processing of arriving trains. A variant decision problem relaxes the train and ship arrival times. The problem is to process all of the trains and ships as quickly as possible and to minimise the schedule duration computed as $\max_{a \in A}(end_a)$. The necessary arrival times of trains and ships is provided as a by-product.

### 2.1. Scheduling Activities

A disjunctive graph is used to efficiently schedule the different activities in the CET. The disjunctive graph is initialised with conjunctive arcs. Disjunctive arcs are later added and removed as needed. Correct activity timings can only be obtained by evaluating the graph via a suitable longest path algorithm. A mathematical description of the required activity on node graph can be found in Appendix A. In summary each activity $a \in A$ has a single node in the graph. Those nodes are assigned a weight equal to the current processing time of the activity, namely $p_a$. In contrast, each resource $r \in R$ must have two nodes, i.e. a source $so_r$ and a sink $si_r$. There is an arc from the source node of the disjunctive graph (i.e. $so$) to the source node of each resource. Similarly there is an arc from the sink node of each resource to the sink node of the disjunctive graph (i.e. $si$). Other arcs are added to enforce the fixed conjunctive relationships between activities of a job and the variable disjunctive precedence prescribed by the current sequencing decisions (i.e. the resource sequences). As each resource has a chosen sequence of activities, there is an arc from the source node of the resource to the first activity in the sequence and an arc from the last activity in the sequence to the sink node of the resource. Other arcs are required for each pairs of activities within the resource sequence. The ready time of a resource is introduced as an arc weight. It is placed on the arc from the source node to the resource's source node. Additional arcs $(so, a)$ may be added that link the source node to each activity node. Release times can then be added as arc weights. Deadlines may be imposed for different activities. Deadlines are enforced by adding reverse (i.e. backward) arcs (Burdett and Kozan, 2018). These reverse arcs have a weight of $-dln_a$, originate at the activity and end at the source node of the DJG. The presence of reverse arcs causes cycles with length less than or equal to zero. Violation of the deadline however results in a positive length cycle. Provided that positive length cycles are not present, a feasible schedule is obtainable using the Bellman Ford algorithm.

It is necessary to point out that processing times must be computed for specific resource assignments and when those resource assignments change, new processing times must be computed. The nodes and arc weights in the disjunctive graph must then be updated.

## 2.2. Resourcing Restrictions

The CET activities require multiple resources and different pairings of resources may be assigned. Some pairings however are invalid and must be restricted. Stacking and reclaiming activities for instance cannot be assigned particular SR and stockpile pairings. Stacker-reclaimers can only service certain stockpiles within a stockyard. In Figure 1 for instance, SR1 and SR2 cannot be assigned to stockpile P11 – P15 however SR1 and SR2 can be assigned to stockpile P1–P10.

It is necessary to record both valid and invalid pairings. A resource pairing graph (*RPG*) and a resource pairing conflict graph (*RPCG*) are introduced. These graphs contain a node for each resource and contain arcs linking resources that can or cannot be paired. For instance in Figure 1's CET the RPG and RPCG arcs respectively are as follows:

$$arc_{RPG} = \{\{SR1, SR2\} \times \{P1, P2, \dots, P10\}\} \cup \{\{SR3\} \times \{P6, P7, \dots, P15\}\} \quad (4)$$
$$arc_{RPCG} = \{\{SR1, SR2\} \times \{P11, P12, \dots, P15\} \cup \{SR3\} \times \{P1, P2, \dots, P5\}\} \quad (5)$$

The pairings can be input explicitly but automatic generation is evidently preferable via equation (6) and (7).

$$arc_{RPG} = \cup_{b \in BUND}\{(sr, p) | sr \in SR, bund_{sr} = b, p \in P_b\} \quad (6)$$
$$arc_{RPCG} = \cup_{b \in BUND}\{(sr, p) | sr \in SR, bund_b = b, p \in P \backslash P_b\} \quad (7)$$

On bund *b* the piles accessible to stacker reclaimers is $P_b$. This set is determined automatically by aggregating piles from adjacent rows. Given $P_w$ is the set of stockpiles in row $w$, and $row_b$ is the set of accessible rows for each bund $b \in BUND$, the stockpiles accessible to bund *b* is $P_b = \cup_{w \in row_b} P_w$. The stockpiles inaccessible to bund *b* is $P \backslash P_b$ and the rows inaccessible to bund *b* is $ROW \backslash row_b$.

Some pairings identified in (6) however must be removed when there are multiple SRs on a bund. For instance, in Figure 2 it is evident that SR2 cannot be at P1, because SR1 is positioned to the left of it. Similarly SR1 cannot be at P5, because SR2 is positioned to the right of it. These invalid assignments must be removed and lead to a reduced decision space. In Figure 2 the feasible and infeasible positions for SR is shown in Table 1. If an additional SR is added to the right of SR2, then the feasible positions are given in Table 2.

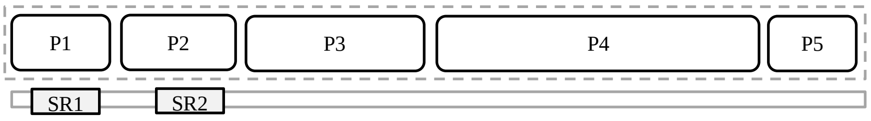

**Figure 2.** Two stacker-reclaimers servicing five stockpiles

**Table 1.** Valid positions for 2 SRs

| | P1 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|
| SR1 | ✓ | ✓ | ✓ | ✓ | × |
| SR2 | × | ✓ | ✓ | ✓ | ✓ |

**Table 2.** Valid positions for three SRs

| | P1 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|
| SR1 | ✓ | ✓ | ✓ | × | × |
| SR2 | × | ✓ | ✓ | ✓ | × |
| SR3 | × | × | ✓ | ✓ | ✓ |

The above logic can be generalised for any number of SR and stockpiles. It can also be applied to SL and berths. Given an ordered list of stacker reclaimer, namely $sr$, and an ordered list of piles $p$, the valid and invalid positions respectively are as follows:

$$\{(sr[i], p[k]) | i \in [1, N], k \in [i, M - N + i]\} \quad (8)$$
$$\{(sr[i], p[k]) | i \in [1, N], k \in [1, i - 1], [M - N + i, M]\} \quad (9)$$

Where $N = |sr|$ and $M = |p|$. Equation (8) and (9) are implemented as sub routines for automatically determining valid and invalid pairings and updating the RPG and RPCG.

Within the FJSOP approach it is necessary to check the validity of the resource assignments that are made. A complete list of valid assignments $\tilde{R}_a$ may be precomputed upfront by performing a Cartesian product of the resource candidates and then subtracting invalid assignments:

$$\tilde{R}_a \subseteq \prod_{z \in [1,|C_a|]} C_a [z] = \{(r_1, r_2, \dots,) | r_z \in C_a[z] \;\; \forall z \in [1, |C_a|]\} \quad (10)$$

For example, if $C_a = (\{P1, P2, P11, P12\}, \{SR1, SR2, SR3\})$ then $\tilde{R}_a \subseteq C_a[1] \times C_a[2]$. According to Figure 1, the invalid assignments are: $(P1, SR2)$, $(P1, SR3)$, $(P2, SR3)$, $(P11, SR1)$, $(P11, SR2)$, $(P12, SR1)$, $(P12, SR2)$. Hence, $\tilde{R}_a = \{(P1, SR1), (P2, SR1), (P2, SR2), (P11, SR3), (P12, SR3)\}$.

The maximum number of assignment solutions for activity $a$ is $|\tilde{R}_a| \leq \prod_{z \in [1,|C_a|]} |C_a[z]|$. The search space therefore contains $\prod_{a \in A} |\tilde{R}_a|$ resourcing options. For $a \in STK$, there are three requisites. If we assume that there are usually two or three piles, two to four SRs, and two DS candidates, then $|\tilde{R}_a| \in [8,24]$. For $a \in RECLD$, there are four requisites. If we assume that usually there are two to three piles, two to four SR, two SL and two BE candidates, then $|\tilde{R}_a| \in [16,48]$. A typical CET scenario might have about 50 trains and 4 ships, i.e. 54 activities. This leads to a vast number of resourcing alternatives: $8^{50} 16^4 = 9.35 \times 10^{49} \leq \prod_{a \in A} |\tilde{R}_a| \leq 24^{50} 48^4 = 5.44 \times 10^{75}$.

### 2.3. Stockpile Profiles

Stockpiles sizes must be updated after stacking and reclaiming activities. By introducing node functions, stockpile size updating may be performed when evaluating the disjunctive graph. Alternatively, it may be performed after, and that is the approach taken in this paper. In retrospect, the main reason to apply node functions (Corry and Kozan, 2004) would be if the activity duration is dependent on both the current size of the stockpile and the amount of material transferred. An update of the activity duration could also be performed by the node function before the remainder of the graph is evaluated.

A mathematical description of how pile profiles, deficits and overloads are determined in our approach can be found in Appendix B. Stockpile profiles denoted by $\Omega_p$ are needed in order to identity the presence or absence of stockpile sizing errors. Each profile contains an ordered list of tuples describing the size of the pile at discrete time points, i.e. $\Omega_p = \{(t, sz)\}$. Events are added to the profile whenever a stacking and reclaiming activity begins or ends. Further events are added whenever the pile reaches its lower or upper bound. The lower bound is zero by default. The profile shown in Figure 3 has 20 events and demonstrates the different scenarios that may occur. The shaded areas are periods of infeasibility.

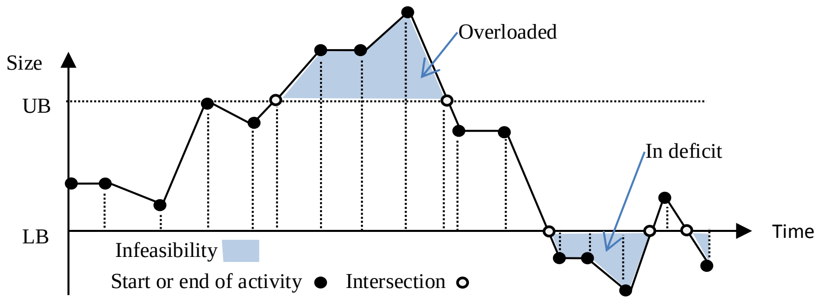

**Figure 3.** Stockpile profile and infeasibilities

The overall level of infeasibility is computed as follows:

$$Infeasibility = \sum_{p \in P}\left(\sum_{z=2,\ldots,|\Omega_p|} \mathbf{G^1}(\Omega_p[z-1], \Omega_p[z]) + \mathbf{G^2}\left(\Omega_p[z-1], \Omega_p[z]\right)\right) \quad (11)$$

$$\mathbf{G^1}\big((\alpha_t, \alpha_{sz}), (\beta_t, \beta_{sz})\big) = -(\beta_t - \alpha_t).(\alpha_{sz} + \beta_{sz} - 2LB)/2 \quad \text{for } \alpha_{sz}, \beta_{sz} \le LB \quad (12)$$

$$\mathbf{G^2}\big((\alpha_t, \alpha_{sz}), (\beta_t, \beta_{sz})\big) = (\beta_t - \alpha_t).(\alpha_{sz} + \beta_{sz} - 2UB)/2 \quad \text{for } \alpha_{sz}, \beta_{sz} \ge UB \quad (13)$$

The violations are otherwise zero if $LB < \alpha_{sz}, \beta_{sz} < UB$. Stockpile errors cannot be allowed in practice. One option is to restrict them from occurring; another is to permit them, and to penalise any occurrences in the objective function. It is worthwhile recording deficits and overloads as shown in Appendix B, because this information can be used to perform corrections. A deficit can be avoided by swapping an "offending" reclaim activity with a stacking activity. Similarly, an offending stack activity that causes an overload can be swapped with a reclaim activity. An offending reclaim activity that causes a deficit could be divided into two parts. The offending part can be postponed and the non-offending part can be left as is.

## 2.4. Pre-emptions

In this section, we consider pre-emptions and describe our approach to include them. Pre-emptions are important as they allow resources to be better utilised. Pre-emptions can be used to reduce delays incurred by trains and ships in CET. All stacking and reclaiming activities can in theory be pre-empted. As train operations are so critical in Australian CET, stacking activities are rarely pre-empted. Hence, we only consider the pre-emption of reclaiming activities here. The set of pre-emptible (i.e. dividable) activities is hence $A^{\div} = RECLD$.

When undertaking the pre-emption of an activity $a \in A^{\div}$, we have chosen to create an additional (i.e. child) activity within the associated job, say $J_i$. The child activity is a "bona-fide" activity with all the characteristics and attributes of previously described activities. Every time a pre-emption is instigated, the set of activities, namely $A$, is enlarged. Child activities are assigned the same resources as the parent activity initially. A set of completely different resources may be assigned later. Child activities must also be added to the activity precedence network of the job, namely $PRE_i$. The following statement describes the pre-emption of activity $a$ by an amount $\Delta$:

$$\mathbf{Pre}(a, \Delta) \Rightarrow (a, a') \text{ s.t. } sz_a = sz_a - \Delta, sz_{a'} = \Delta \text{ where } \Delta \in [\tau, sz_a - \tau] \quad (14)$$

The amount that is assigned to the child is restricted. In practice it is not desirable to partition the activity below a given tonnage $\tau$. This minimum tonnage is used as the basis of a discretisation of pre-emption sizing, such that the number of ways to pre-empt an activity is $\lfloor sz_a/\tau \rfloor$. Any remainder below $\tau$ can be arbitrarily assigned to any of the $\lfloor sz_a/\tau \rfloor$ pieces. For instance, if $sz_a = 11.25$ tonnes and $\tau = 0.5$ tonnes there are potentially 22 pieces. The remainder of 0.25 is less than $\tau$ and can be redistributed to any of those pieces.

At the point of creation, the child activity $a'$ should be labelled appropriately and "linkages" between the parent and child should be established:

$$parent_{a'} = a;\ if(child_a \neq NULL) \text{ then } child_{a'} = child_a \text{ and } parent_{child_a} = a'; \\ child_a = a'; \tag{15}$$

These linkages are important when adding new pre-emptions, or removing existing ones, and to ensure correct updating of the disjunctive graph and resource sequences. The labelling of child activities is a practical consideration, but is important for understanding solutions. The selection of a poor naming convention can lead to a number of implementation issues.

Every $a \in A_i$ has a list of successor activities. Those lists must be revised when activities are pre-empted. The parent's successor list is assigned to the child, and the parent's successor list is updated to contain only the child, i.e. $succ_c = succ_a$ and $succ_a = c$.

When activity $a$ is pre-empted and a child activity $a'$ is created, the conjunctive arcs within the disjunctive graph must be revised as follows:

$$E^{\mathrm{C}} = E^{\mathrm{C}} \cup \{(a, a')\} \cup \{(a', a^*) | \forall (a, a^*) \in PRE_i\} \backslash \{(a, a^*) | (a, a^*) \in PRE_i\} \tag{16}$$

For every successor of $a$, namely $a^*$, the existing arc $(a, a^*)$ must be removed, and replaced with $(a', a^*)$. Every time an activity is pre-empted, the disjunctive arcs must also be revised:

$$E^{\mathrm{D}} = E^{\mathrm{D}} \cup \bigcup_{r \in R_a} \{(a', \sigma_r[k+1]) \cup (a, a')\} \setminus \bigcup_{r \in R_a} \{(a, \sigma_r[k+1])\} \tag{17}$$

It is noteworthy to mention that *k* is the position of activity *a* in sequence $\sigma_r$, i.e. $\sigma_r[k] = a$.

To undo a pre-emption (i.e. to reset a pre-empted activity) the reverse of the operations described in (16) and (17) are made. The resources assigned to the child activity are discarded when the pre-emption is undone, i.e. the resources assigned to the parent are retained. The other processes described also need to be reversed when a child activity is merged with its parent. Successor lists are returned to the parent and parent-child linkages are revised.

## 2.5. Advanced Features

The key features of our FJSOP approach for CET scheduling have been detailed in previous sections. There are a number of other practical considerations that need to be incorporated, like double width stockpiles, changing stockpile layouts over time, collisions between mobile resources like stacker-reclaimers and ship-loaders. These additional complexities however may not be present in every instance. Further details are provided in Appendix C, D and E for interested readers.

Fixed resource assignments are a possibility that should also be considered. A typical example is the assignment of ships upfront to specific berths or the assignment of SRs to specific stockpiles. The set of fixed resource assignments should be defined upfront as follows: $FIXED = \{(a, r) | a \in A, r \in R\}$. The assignments declared in this set are made and a fixed assignment status is recorded for the requisite record involved.

Fixed sequences may also be specified for specific resources. The set of fixed sequences is defined upfront as follows: $FIXED\_SEQUENCE = \{(r, \sigma_r^{\mathrm{F}}) | r \in R, \sigma_r^{\mathrm{F}} \subset A\}$ where $\sigma_r^{\mathrm{F}}$ is an ordered set of activities. A resource with a fixed sequence is recorded as such via a Boolean parameter and the sequence is assigned, i.e. $\sigma_r = \sigma_r^{\mathrm{F}}$. The declaration of a fixed sequence also implies fixed resource assignments. For instance $\forall a \in \sigma_r^{\mathrm{F}}$ the assignment of resource *r* to *a* is made.

It is assumed that the sequence is complete. In other words, the sequence is not a partial sequence and it cannot be altered, except by pre-empting current activities. Ignoring peculiarities induced by pre-emption, fixed sequences result in the presence of fixed disjunctive arcs. The pre-emption of an activity within a fixed sequence results in the need to remove the current disjunctive arcs and to add different ones.

## 3. Meta Heuristic Approach

The FJSOP is an inherently complex and challenging decision problem and unique features present in CET make it even more challenging. Many different solution approaches can be taken to solve this problem. Traditional mixed integer programming (MIP) methods may be formulated however they are unlikely to be effective on full sized instances. Pre-emption is particularly challenging to handle in a MIP setting, greatly increasing the number of variables and constraints. Since commercial applications are time-critical in terms of solution time, constructive algorithms and meta-heuristics are the preferred method (Gao et al., 2014).

The choice of meta-heuristic is important but the consideration of how to effectively perturb a solution is more so. Within a feasible solution, the perturbation of a single resource assignment is highly problematic, as resource pairing infeasibilities can be created easily. Consider the resource selection (SR1, P1, DMP1) and the stockyard shown in Figure 1. If SR1 is replaced with SR3, the resulting assignment (SR3, P1, DMP1) is invalid as SR3 cannot ever reach P1. Many other types of infeasible pairings can also be created. If an attempt is made to perturb an activity from its current position within a resource sequence, positive length cycles are easily created within the disjunctive graph and a feasible schedule is not forthcoming. As each activity is present in multiple resource sequences, some type of compound move is definitely needed. To the best of our knowledge, compound moves have only been applied successfully in train scheduling applications (Burdett and Kozan, 2010a) and are somewhat rare.

In response a very different approach has been developed. We propose a new hybrid Simulated Annealing (HSA) meta-heuristic using a suite of perturbation and improvement operators. A standout feature is the use of a priority ordering chromosome and a constructive algorithm to decode it. The constructive algorithm operates in a deterministic way and is also used to perform resource assignments. It does a good job of assigning resources to activities but does not guarantee an optimal resource assignment. For that a dedicated resourcing chromosome is needed. More will be said about the constructive algorithm in due course. A Simulated Annealing inspired approach has been developed, because the need for computational speed is important. Of all the meta-heuristics available, it is arguably the fastest. This solution technique has very robust performance, and is well tested on “hard” optimization problems, for instance in Burdett and Kozan (2009c, 2010a, 2014, 2018), and many others. Meta-heuristics that manipulate a population of solutions are quite applicable and provide high quality solutions. Unfortunately they are significantly slower in comparison (Burdett and Kozan, 2014).

The priority ordering is simply an ordered list of activities. It is used in the optimisation process but is discarded upon completion. Perturbations of the priority ordering are used as a way to obtain high quality schedules. Our choice of this solution chromosome significantly simplifies the solution process and foreseeable complications with compound moves are avoided. Without the priority ordering chromosome and the constructive algorithm that evaluates it, it is necessary to explicitly manipulate the activities within each resource sequence. The primary solution chromosome would otherwise be a very large number of separate sequences. In a typical scenario like the one shown in Figure 1, there at least 25 resources and each one of those has a sequence of activities. In contrast, the priority ordering is a single sequence, albeit of much greater size. All of the CETs activities are present in this sequence, and consequently there are $|A|!$ solutions to choose from without consideration of pre-empted activities. In a typical scenario with 50 trains and 4 ships there are at least $54! = 2.31 \times 10^{71}$ orderings to choose from. Pre-empted activities will cause the priority ordering to grow and contract. The number of priority orderings could be as many as $(|A| + \sum_{a \in RECLD} \lfloor sz_a/\tau \rfloor)!$ but this value depends greatly on the size of each activity.

***Constructive Algorithm***: The main details of the constructive algorithm are provided in Appendix F. The constructive algorithm handles pre-emptions, fixed sequences, resource tokens and stockpile sizes. Its purpose is to create the resource sequences from a priority ordering of the stacking and reclaiming activities. The position of each activity within the priority ordering, dictates when it is granted permission to acquire the resources that it needs. The algorithm is iterative. At each step an activity is scheduled as soon as its resources are deemed available. That activity is simply appended to the partial sequences of the acquired resources. Consequently no complex insertions into resource sequences, is required. After an activity is scheduled, the availability of the associated resources is updated, i.e. as the activities planned completion time. If a prior resource assignment is not provided, the algorithm chooses which resources to assign based upon their earliest availability. When choosing resources for a particular activity $a$, the set of valid resource selections $\tilde{R}_a$ is considered. The resource selection with the best score is chosen. The score has two parts; the primary part is the earliest availability:

$$ES_v = \max_{r \in R_v}(free_r, free_\tau) \;\; \forall v \in \left[1, \left|\tilde{R}_a\right|\right] \;\; \text{where } \tau = token_r \tag{18}$$

Tokens are additional resources that are required. These special resources are introduced to restrict multiple access and will be discussed in detail later. If a token $\tau$ is required by resource $r$ then it is also included in the calculations. If the resource is a stockpile (i.e. $r \in P$), then the level of the stockpile is checked with the size of the activity. If a deficit or overload is identified, the extent of the violation is penalised by $\lambda$. Hence the final score is as follows:

$$score_v = ES_v + \lambda \times viol_v \;\; \forall v \in \left[1, \left|\tilde{R}_a\right|\right] \tag{19}$$

***Initial Priority Ordering***: To obtain an initial priority ordering, a topological sort of the disjunctive graph is performed. At this point in time there are no disjunctive arcs except those implied by fixed sequences. The topological sort ensures that precedence constraints between the activities within each job are not violated, and by definition provides a list of activities.

***Control Structure***: The HSA is an extended version of Simulated Annealing. There is an outer loop and an inner loop. The outer loop starts at temperature $t = tI$ and is terminated when the temperature reaches $t = tF$. At the end of each temperature step the temperature is updated as follows: $t = t \times tRed$ where $tRed \in (0,1)$. The inner loop is terminated after a given number of state changes (i.e. $iter$) at the given temperature. State changes are accepted using the standard metropolis function (Van Laarhoven et al., 1992).

***Perturbations and Refinements***: We have introduced several perturbation strategies and several local improvement operations to the HSA. These are applied to the current priority ordering and are performed sequentially within the inner loop. They are required in order to make quicker progress in this difficult multifaceted decision space. A precedent for such an approach can be found in Burdett and Kozan (2014). These strategies make feasible moves. The only infeasibilities allowed are pile level violations and SR collisions. They are resolved explicitly or else sifted out during the search. Infeasibilities are penalised and added to the objective. The penalty is variable and is increased as the search proceeds in the following way: $\lambda(t) = e^{\alpha.n}$ where $\alpha = 2\ln\left(\lambda^{mid}\right)/T$ and $t$ is the temperature step (i.e. counter). The number of temperature steps is $T = \left\lceil \ln\left(\frac{tF}{tI}\right) / \ln(tRed) \right\rceil$ and the penalty midway is $\lambda^{mid}$.

As the priority ordering is a sequence, shifts and reversions are particularly useful. At the start of every temperature step, one of these is chosen with equal probability. Perturbations are accepted in the traditional way, according to the extent of the improvement or deterioration and the current temperature of the HSA. The shift procedure moves an arbitrarily chosen activity in position $u$ to position $v$. The position to which it may be moved, is bounded by the position of predecessor and successor activities of the associated job. A chosen shift may not occur if precedence constraints defined by fixed sequences are violated. Infeasibilities are identified by checking for an activities membership in a fixed sequence. For instance if the result of (20) and (21) is true then the chosen shift is rejected. If the chosen shift is infeasible the HSA does not evaluate another at the current iteration; the algorithm moves on.

$$\bigwedge_{\forall r \in FIXED\_SEQEUNCE} \bigwedge_{k \in [u+1,v]} (order[k] \in \sigma_r) \qquad \text{for } u < v; \qquad (20)$$
$$\bigwedge_{\forall r \in FIXED\_SEQUENCE} \bigwedge_{k \in [v,u-1]} (order[k] \in \sigma_r) \qquad \text{for } u > v; \qquad (21)$$

A reversion takes an arbitrarily chosen sub sequence of the activities and reverses it. The size of the sub sequence is selectable. Numerical testing indicates a size of 2 to 5 activities is best. Reversions can easily violate the conjunctive precedence constraints designated within jobs. These infeasibilities frequently occur as reclaiming activities are broken into smaller and smaller pieces. Our reversion method therefore performs an auto correction. The auto correction is facilitated by first storing the activity sequence of each job within the current priority ordering. Let $\phi_k^i$ be a tuple that describes the *k*th activity of job $J_i$ that has occurred and its' position within the priority ordering ***before*** the reversal. Hence $\phi_k^i = (\phi_k^i[1], \phi_k^i[2])$ and $\phi_k^i[1] \in A_i$ and $\phi_k^i[2] \in \mathbb{Z}$. After the reversal is performed the new activity sequence of each job is recorded. Let $\psi_k^i$ be a tuple that describes the *k*th activity seen of job $J_i$ and its' position ***after*** the reversal. Hence $\psi_k^i = (\psi_k^i[1], \psi_k^i[2])$, $\psi_k^i[1] \in A_j$ and $\psi_k^i[2] \in \mathbb{Z}$. The original activity sequence of each job is corrected (i.e. reinstated) in the following way:

$$order[\psi_k^i[2]] = \phi_k^i[1] \quad \forall J_i \in J; \forall k \in K_i \qquad (22)$$

For example: Consider a sub sequence of activities: $[5^{(3)}, 6^{(1)}, 2^{(2)}, 3^{(2)}, 7^{(1)},]$. The associated job is written as a superscript. Therefore: $\phi^1 = [(6,2),(7,5)]$, $\phi^2 = [(2,3),(3,4)]$ and $\phi^3 = [(5,1)]$. After a reversion, the sub sequence is $[7^{(1)}, 3^{(2)}, 2^{(2)}, 6^{(1)}, 5^{(3)}]$ and $\psi^1 = [(7,1),(6,4)]$, $\psi^2 = [(3,2),(2,3)]$ and $\psi^3 = [(5,5)]$. Four changes are necessary to reinstate the original conjunctive relations, namely: $order[1] = 6$, $order[4] = 7$, $order[2] = 2$, $order[3] = 3$. The final sequence is $[6^{(1)}, 2^{(2)}, 3^{(2)}, 7^{(1)}, 5^{(3)}]$.

Just like shifts, reversions may also violate fixed sequences. A reversion between *u* and *v* is infeasible and must be rejected if (23) is true:

$$\bigwedge_{\forall r \in FIXED\_SEQENCE} \bigwedge_{k \in [u,v]} (order[k] \in \sigma_r) \qquad (23)$$

The aforementioned perturbations are valuable but their application is unguided and unpredictable, and may result in slow convergence in more complex integrated decision spaces. Two local refinement procedures are hence suggested. These are applied next within the inner loop. As more computational effort is required to apply these, they applied less frequently, and with equal probability. The first refinement iteratively tests the value of all pairwise exchanges (i.e. shifts of one position). The algorithm starts at the first pair of activities and then proceeds on to other pairs. At any point in time, if an exchange provides an improvement it is automatically accepted, otherwise it is rejected. This algorithm sifts activities within the priority ordering, from right to left, and is particularly useful when jobs have release times. This algorithm is geared towards moving activities with earlier release times. An exchange that violates fixed sequences is rejected according to equation (24).

$$\bigwedge_{\forall r \in FXD} (order[k] \in \sigma_r) \wedge (order[k+1] \in \sigma_r) \qquad (24)$$

The second refinement involves the re-insertion of a single arbitrarily selected activity. All positions are evaluated and the best is accepted if an improvement occurs. Otherwise no shift is made. The re-insertion position is restricted by predecessor and successor activities of the associated job. The validity of the re-insertion is also checked by (20) and (21).

***HSA Modes***: The HSA can be run in one of two modes. Mode 1 (aka "NO_PREEMPT") does not facilitate and handle pre-emptions. It runs faster but may provide inferior solutions if pre-emptions are strictly necessary to achieve the optimal solution. Mode 2 (aka "PREEMPT") has full pre-emption handling. Numerical testing indicates that the application of mode 1 first can sometimes make it easier for mode 2 to obtain a higher quality solution. Mode 3 is defined as the application of mode 1 first, followed by the application of mode 2.

***Making Pre-emptions***: Pre-emptions lead to significant improvements in solution quality and can be handled in several ways. The development of perturbation operators that arbitrarily partition "pre-emptible" activities into smaller pieces is one possibility. A complementary approach that undoes prior pre-emptions is also possible. An alternative approach is to automatically make pre-emptions when beneficial. The advantage of that approach is evident. It can be used to directly "polish" the current solution and can be applied each iteration (i.e. temperature step) to great effect. This is the approach we have developed. The "polish pre-empt" procedure is applied directly after the aforementioned perturbation and refinement procedures.

To polish the solution it is necessary to analyse the pre-emptible activities ($A^{\div}$) within the current priority ordering in the order that they appear. Our algorithm considers how big the child activity should be and where it should be inserted within the current priority ordering. All feasible insertion positions are analysed. Evidently there are fewer positions to consider when activities are positioned later in the priority ordering. The child's size is restricted to the range $[\tau, sz_a - \tau]$. In order to break up the parent activity it is necessary that $sz_a \geq 2\tau$. There are many ways to divide an activity into smaller pieces. We evaluate all multiples of $\delta$ where $\delta = (sz_a - 2\tau)/N$ and choose the best. Consequently the computational effort is $O(N|A^{\div}|)$. Evidently the CPU time and the solution quality can vary considerably depending on how many bits $N$ are entertained.

***Updating Fixed Sequences***: After activity pre-emptions are performed it may be necessary to repair fixed sequences. The order of those activities cannot be changed, however it is possible to partition them into smaller pieces in many different ways. Consequently the fixed sequence can expand and contract as the search progresses, and needs to be continually maintained to accurately reflect the currently pre-empted activities. The correction is facilitated using a copy of the original fixed sequences. To perform the correction it is necessary to discard the current fixed sequences as some child activities may no longer exist and to add in the current activities.

***Removing Redundant Pre-emptions***: It is worthwhile to remove redundant pre-emptions from the final solution. A pre-emption is regarded as redundant if the resources assigned to the child activities are the same as those assigned to the parent and the start time of the child activity equals the completion time of the parent. To remove redundant pre-emptions, the redundant child is merged with the parent, and ceases to exist. The child of the child is then analysed if it exists. This process is applied to all pre-empted activities, to reduce the length of the priority ordering, and the computational effort required to manipulate it. A potential disadvantage is that there is more chance of falling into local optima and there are fewer opportunities to move reclaiming activities around. There needs to be a balance between keeping and removing redundant pre-emptions. Numerical testing indicates that removing redundant pre-emptions 10% of the time generally leads to better solutions. The removal of redundant pre-emptions is the last activity performed within the inner loop of the HSA.

***Solution Evaluation***: The steps required to evaluate a solution are now discussed. Given a set of resource allocations and a priority ordering, the aforementioned constructive algorithm is used to construct the sequences. The disjunctive graph is then updated using the new sequences. Additional temporary disjunctive arcs are added to enforce dynamic stockpile precedence. These continually change as activities are reassigned to different piles and cannot be ascertained upfront. These arcs are removed when no longer needed. The Bellman Ford algorithm is then applied to the disjunctive graph to determine earliest start times for each activity. Afterwards the activities are scheduled and key performance indicators are computed. Stockpile level profiles and infeasibilities are then determined. The objective function is penalised if need be. Finally the presence of stacker-reclaimer collisions are identified. If any are detected, the objective function is again penalised. Appendix E provides full details of collision detection.

## 4. Numerical Investigations

This section tests the effectiveness of the proposed FJSOP framework and the HSA solution approach. Our scheduling procedures have been coded in C++. All numerical experiments have been run on a quad core Dell personal computer (PC) with a 2.6 Ghz processor and 16 GB memory under Windows 7. Scheduling metrics are shown in minutes and CPU times in seconds. All values are rounded up to the nearest integer.

A coal terminal in Australia is considered. The full details of the case study can be found in Appendix G. The stockyard of this CET has thirty stockpiles P01 – P30 and these are positioned as shown in Figure 4 (i.e. a top down view). The x-axis represents the chainage in that figure. The CET has four bunds, BU01 - BU04, and five pads, R01 – R05. Each bund has a single SR, namely SR01 – SR04. The stack and reclaim rate for SRs is respectively 6780 and 7360 tonnes/hr. They move at 2.4 km/h. As one SR occupies each row there is no possibility for collisions to occur. There are three berths and two ship loaders. The SL move at 1.8 km/h. SL01 is not permitted to access BE03 and SL02 is not permitted to access BE01. Hence there is no possibility for collisions to occur between SL. Two dump stations exist for trains to unload coal, namely DS01 and DS02. Four ships SH01 – SH04 and 52 trains, TR01 – TR52, need to be scheduled. Every train has DS01 and DS02 as requisites. In addition every stack and reclaim activity has SR01-SR04 as requisites and every reclaim has SL01 and SL02 as requisites. The stockpile requisites for each stack and reclaim are shown in Appendix G. It is not permitted to pre-empt stacking activities as trains have a tight time window for unloading. SH01 and SH03 require BE01 and SH02 and SH04 require BE02. For this study, ship berthing sequences are given and an allocation of ships to berths has been established. The docking sequences for ships are as follows: BE01: (SH01, SH03); BE02: (SH02, SH04).

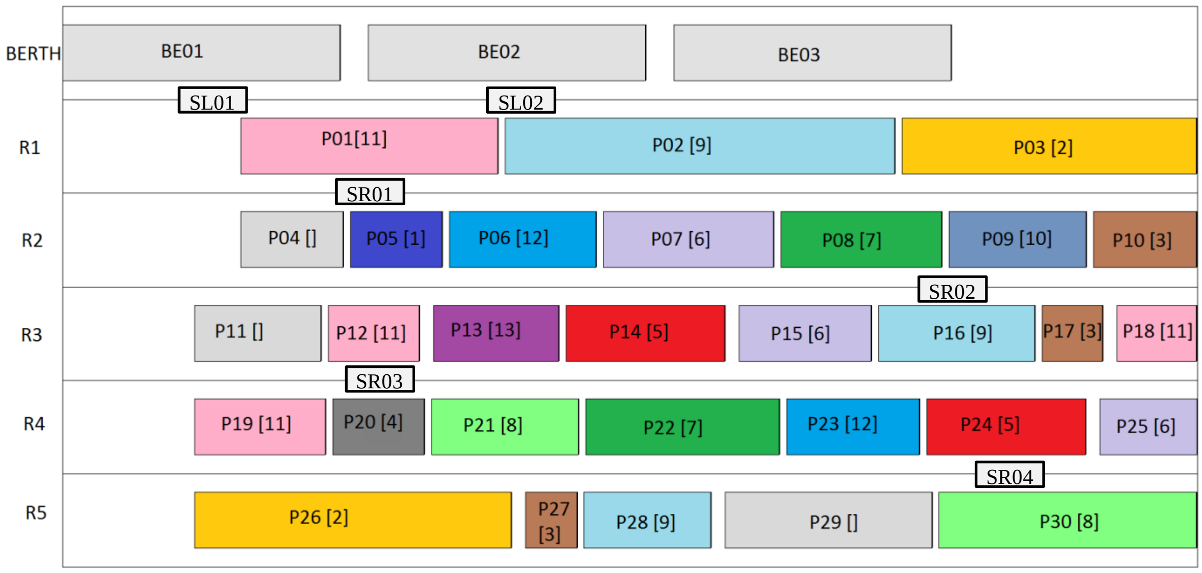

**Figure 4**. Schematic of the CET including SR/SL starting positions and products in "[]"

## 4.1. Release Times – Fixed Train and Ship Arrivals

The arrival of trains and ships at specific times is considered here. A weighted flow time objective is assigned and stockpile level violations are added as a penalty. The HSA was first applied using mode 1 which has no pre-emption handling. The application of the HSA with mode 2 and 3 (i.e. that uses both modes sequentially) was then evaluated. Ten runs were performed and the summarised results that contrast the application of the three different strategies are shown in Table 2. It is evident that mode 1 (i.e. NO_PREEMPT) is most variable and produces the worst solutions. That approach however is very fast. Without pre-emption handling all of the solutions obtained using that mode had one or two pile sizing errors. It is noteworthy to mention that mode 1 and mode 3 produced feasible solutions in all runs and that the CPU requirements and level of variability are less when mode 1 is run first. When used as the starting point for mode 2, better solutions can often be obtained than otherwise. The time spent evaluating the priority ordering is around 80% of the total time spent running the HSA. Hence 20% of the time is spent perturbing a solution. Within the time spent evaluating the priority ordering, around 37% is spent evaluating the disjunctive graph and the remainder (i.e. 63%) is spent constructing the sequences.

**Table 2.** Aggregated HSA results [RLT + WFLOW objective / $tI$ = 1000, $iter$ = 300]

| MODE | WFLOW | | | | CMAX | | | | CPU (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | min | avg | max | stdev | min | avg | max | stdev | min | avg | max | stdev |
| 1 | 78593 | 115115 | 151927 | 27092 | 9249 | 9699 | 10087 | 282 | 155 | 163 | 170 | 6 |
| 2 | 45163 | 47961 | 54380 | 2653 | 8798 | 8840 | 9020 | 80 | 1922 | 2148 | 2382 | 140 |
| 3 | 45044 | 48562 | 53132 | 2621 | 8798 | 8853 | 9033 | 92 | 1887 | 2003 | 2207 | 113 |

To investigate the full effect of the iteration number and starting temperature on the performance of the HSA, more tests were performed and these are summarised in Table 3.

**Table 3.** Aggregated HSA results for mode 2 [RLT + WFLOW objective]

| | | WFLOW | | | | CMAX | | | | CPU (sec) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $tI$ | $iter$ | min | avg | max | stdev | min | avg | max | stdev | min | avg | max | stdev | $\Delta^1$ | $\Delta^2$ |
| 1 | 100 | 56078 | 66397 | 97039 | 11964 | 8811 | 9023 | 9285 | 190 | 179 | 236 | 295 | 40 | 16 | 43 |
| 10 | 100 | 45097 | 53136 | 62602 | 4043 | 8798 | 8925 | 9158 | 269 | 347 | 398 | 443 | 68 | 16 | 28 |
| 100 | 100 | 48093 | 54647 | 65940 | 2829 | 8823 | 9083 | 9211 | 130 | 357 | 483 | 609 | 87 | 12 | 28 |
| 1000 | 100 | 47679 | 53766 | 64178 | 5820 | 8798 | 9072 | 9387 | 153 | 455 | 648 | 779 | 36 | 12 | 26 |
| 1 | 200 | 50134 | 55302 | 61520 | 5279 | 8798 | 9007 | 9548 | 219 | 379 | 522 | 595 | 72 | 10 | 19 |
| 10 | 200 | 46091 | 52542 | 59847 | 2305 | 8798 | 8992 | 9396 | 123 | 697 | 808 | 910 | 102 | 13 | 23 |
| 100 | 200 | 44948 | 50212 | 53823 | 6332 | 8798 | 8896 | 9173 | 131 | 935 | 1065 | 1190 | 96 | 11 | 17 |
| 1000 | 200 | 47204 | 51532 | 56779 | 2458 | 8798 | 8891 | 9079 | 146 | 1208 | 1381 | 1579 | 93 | 9 | 17 |
| 1 | 300 | 47035 | 51696 | 55749 | 1306 | 8798 | 8915 | 9196 | 153 | 710 | 851 | 956 | 155 | 10 | 16 |
| 10 | 300 | 46630 | 49874 | 53947 | 5387 | 8823 | 8877 | 9158 | 171 | 1022 | 1217 | 1370 | 101 | 7 | 14 |
| 100 | 300 | 46602 | 48443 | 50699 | 3643 | 8798 | 8907 | 9158 | 108 | 1467 | 1672 | 2022 | 115 | 4 | 9 |
| 1000 | 300 | 45163 | 47961 | 54380 | 2653 | 8798 | 8840 | 9020 | 80 | 1922 | 2148 | 2382 | 140 | 6 | 17 |

In those tests we chose to apply mode 2 as it is the primary mode of the HSA. The weighted flow time and the makespan of the schedule are shown. All of the aforementioned are important descriptors of the relative merit of a schedule. Table 3 demonstrates that increasing the number of iterations from 100 to 200 doubles the CPU time, but a further increase to 300 is about 1.5 times the time for 200 iterations. This reduction occurs because the solution converges faster with increased iterations, and refinements that cause infeasibilities or large changes to the objective are restricted sooner. If they are permitted, then more effort is required to refine the solution in later iterations. In earlier iterations, there is great freedom of movement; there are more insertion positions to evaluate and more pre-emptions to consider.

The level of improvement varies and depends whether best, average or worst case behaviour of the HSA is considered. The increased number of iterations predominantly results in superior solutions. In terms of the average and worst case it is always better to have a higher number of iterations. The best solution however is not necessarily obtained with the highest starting temperature. There is some evidence that a starting temperature of 100 is better than 1000. The difference between the average solution and the best solution (i.e. column $\Delta^1$) ranges from 3.8 - 15.54 percent. On average the improvement is at least 9.96%. The different between the best and worse solution (i.e. column $\Delta^2$) ranges from 8 - 42% but on average is 12.4%. The best solution obtained is shown in Figure 5.

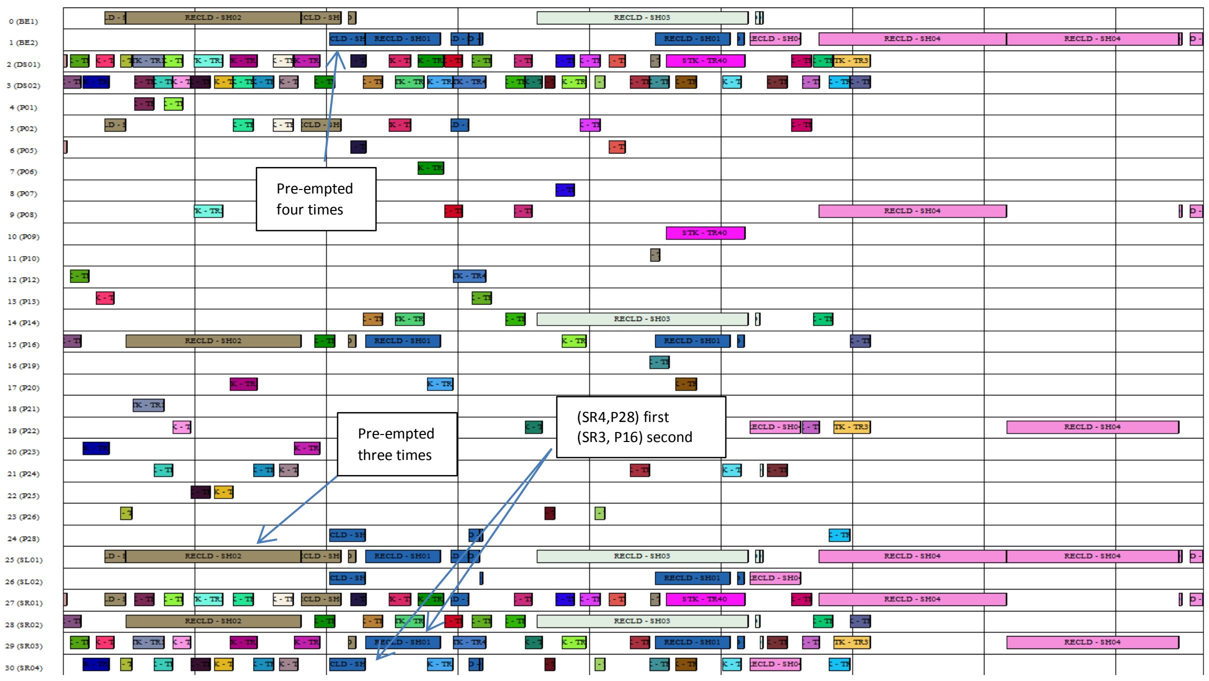

**Figure 5.** Gantt chart of the best solution [RLT case]

The vertical time lines are drawn every 6 hours and each row refers to a particular resource. Each train and ship has a different colour for its activities. This solution exhibits features one would expect in an efficient schedule. Most trains are processed straight away and the DS are running fairly constantly. Each SR is highly utilised and the idle periods are relatively small.

The typical convergence of the HSA is shown in Figure 6 for the three different starting temperatures. The second chart is a magnified portion. The best solution and the current solution at each temperature step are shown. These charts show very rapid convergence over the first 20 temperature steps. They also clearly show that the higher temperature is very much needed, to obtain the best possible solution. A low temperature also works particularly well. The current solution at each temperature step fluctuates greatly, and this is in part due to the acceptance of infeasible solutions whose occurrence must be penalised. It is important to remember that our penalty is increased as the search progresses, and is largest towards the end.

### 4.2. No Release Times – Flexible Train and Ship Arrivals

The arrival of trains and ships at any time is considered here. Consequently the sequence and timing of their arrivals are decision variables, rather than fixed as in previous examples. A weighted flow objective was first considered. When there are no release times however it makes more sense to minimise the duration of the schedule and not to prioritise trains over ships. Hence a makespan objective was selected with $(\omega_1, \omega_2) = (1,1)$. Stockpile level violations are again added as a penalty. Ten runs were performed and the results are summarised in Table 4. In comparison to Section 4.1 results, the makespan is almost one third. Mode 2 produced the best solution but on average mode 3 is best; there is less variation in the solution quality when mode 1 is run first. To investigate the full effect of the iteration number and starting temperature, more tests were again performed and these are summarised in Table 5. The best solution obtained is shown in Figure 7.

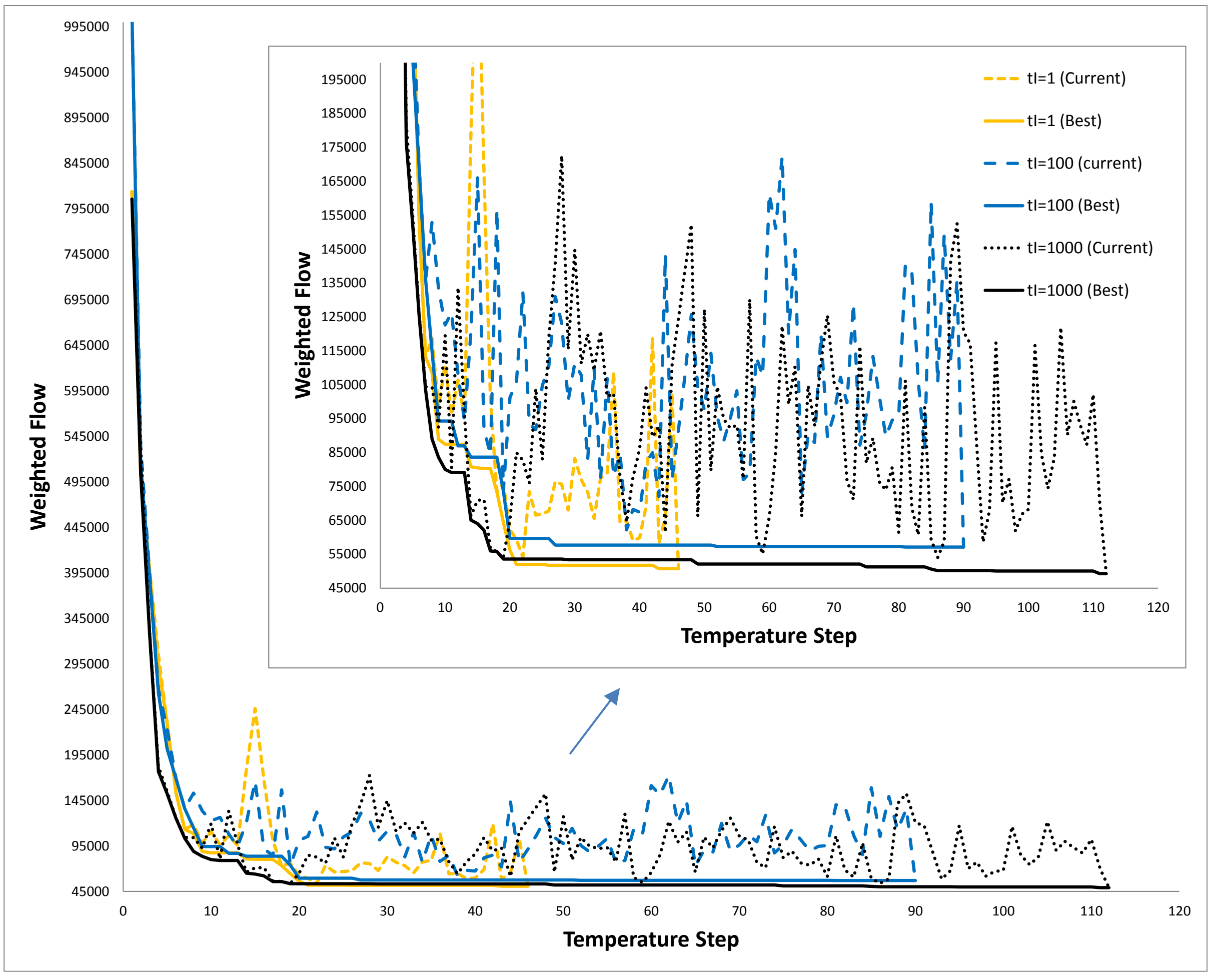

**Figure 6.** Convergence of the HSA [RLT CASE / iter = 100]

**Table 4.** Aggregated HSA results [No RLT + CMAX objective/ $tI$ = 1000, $iter$ = 300]

| MODE | WFLOW | | | | CMAX | | | | CPU (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | min | avg | max | stdev | min | avg | max | stdev | min | avg | max | stdev |
| 1 | 54150 | 59133 | 70055 | 4687 | 2687 | 3012 | 3407 | 246 | 153 | 157 | 161 | 3 |
| 2 | 53787 | 56143 | 58897 | 1815 | <u>1891</u> | 1932 | 1981 | 30 | 2897 | 3257 | 4168 | 476 |
| 3 | 53183 | 55928 | 58122 | 1751 | 1898 | 1932 | 1975 | 25 | 2604 | 3065 | 3609 | 295 |

**Table 5.** Aggregated HSA results for mode 2 [No RLT + CMAX objective]

| | | WFLOW | | | | CMAX | | | | CPU (sec) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $tI$ | $iter$ | min | avg | max | stdev | min | avg | max | stdev | min | avg | max | stdev | $\Delta^1$ | $\Delta^2$ |
| **1** | 100 | 54439 | 57669 | 60839 | 2175 | 1920 | 1977 | 2027 | 42 | 529 | 682 | 862 | 87 | 2.9 | 5.29 |
| **10** | 100 | 55270 | 58011 | 62637 | 2344 | 1913 | 1977 | 2031 | 43 | 843 | 1007 | 1185 | 111 | 2.74 | 4.11 |
| **100** | 100 | 55158 | 56924 | 57799 | 766 | 1904 | 1967 | 2054 | 47 | 1323 | 1496 | 1681 | 145 | 4.07 | 15.33 |
| **1000** | 100 | 54948 | 56995 | 60532 | 1850 | 1891 | 1955 | 2033 | 44 | 1469 | 1770 | 2041 | 177 | 3.25 | 5.79 |
| **1** | 200 | 53851 | 57215 | 61298 | 2184 | 1903 | 1957 | 1985 | 26 | 1272 | 1428 | 1620 | 122 | 2.92 | 4.4 |
| **10** | 200 | 52627 | 56112 | 58883 | 1609 | 1896 | 1953 | 1983 | 34 | 1804 | 2173 | 2706 | 306 | 3.42 | 6.22 |
| **100** | 200 | 54408 | 57023 | 60168 | 1929 | 1903 | 1950 | 1993 | 33 | 2344 | 2675 | 3004 | 197 | 3.17 | 7.31 |
| **1000** | 200 | 54970 | 57220 | 59866 | 1682 | 1909 | 1950 | 1982 | 28 | 2598 | 3252 | 3712 | 414 | 2.41 | 4.51 |
| **1** | 300 | 54466 | 57364 | 63312 | 2988 | 1931 | 2013 | 2281 | 101 | 1730 | 2001 | 2296 | 173 | 3.43 | 6.47 |
| **10** | 300 | 52946 | 55726 | 57862 | 1610 | 1881 | 1948 | 2006 | 43 | 2223 | 2598 | 3048 | 284 | 3.27 | 6.98 |
| **100** | 300 | 52281 | 55830 | 58819 | 1887 | 1869 | 1936 | 1999 | 43 | 2147 | 2804 | 3448 | 486 | 2.1 | 3.67 |
| **1000** | 300 | 53787 | 56143 | 58897 | 1815 | 1891 | 1932 | 1981 | 30 | 2897 | 3257 | 4168 | 476 | 2.1 | 4.56 |

The difference between the average solution and the best ranges from 2.1 – 4.1%, but is around 2.98% on average. The difference between the best solution and the worst ranges from 3.67 – 15.33% but is about 6.22% on average. These ranges are much smaller than encountered when there are release times. Evidently, the problem is easier to solve without release times. A higher iteration number still tends to be best but a significant difference between 100, 200 or 300 iterations is not easy to distinguish. Table 5 also shows little evidence that the starting temperature is that significant here. Hence, lower starting temperatures and iteration counts seem to be sufficient. Evidently the search space is more constrained and complex when release times are present, and the HSA needs more freedom initially, and hence a higher starting temperature. The typical convergence of the HSA is shown in Figure 8 for the three different starting temperatures. The best solution and the current solution at each temperature step is again shown.

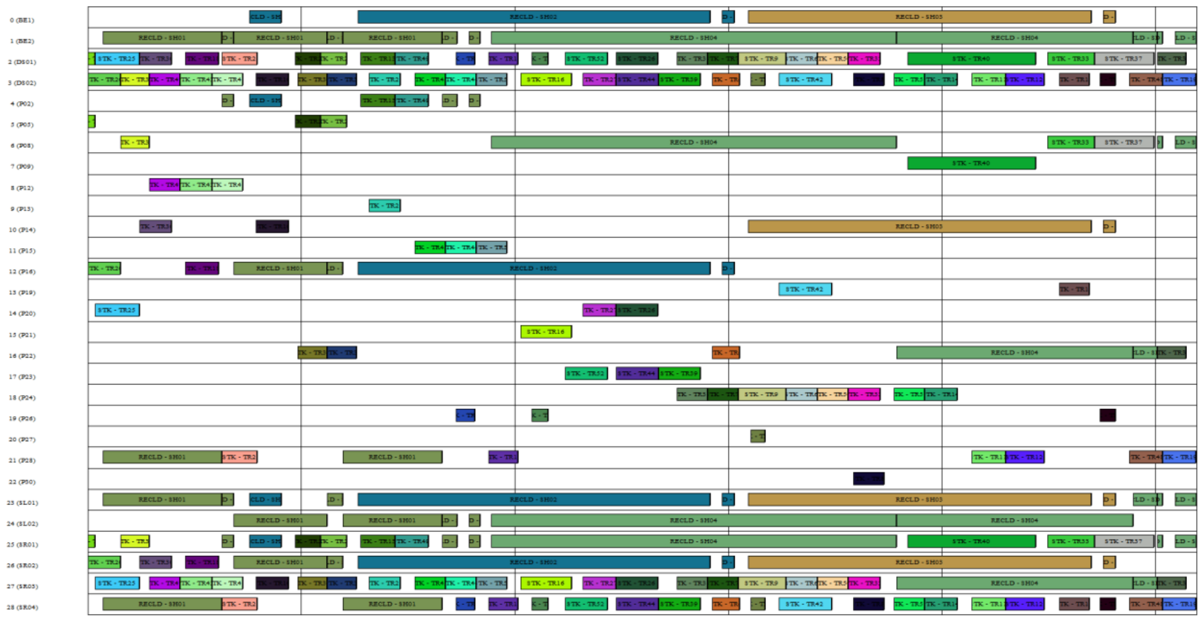

**Figure 7.** Gantt chart of the best solution for the No RLT scenario

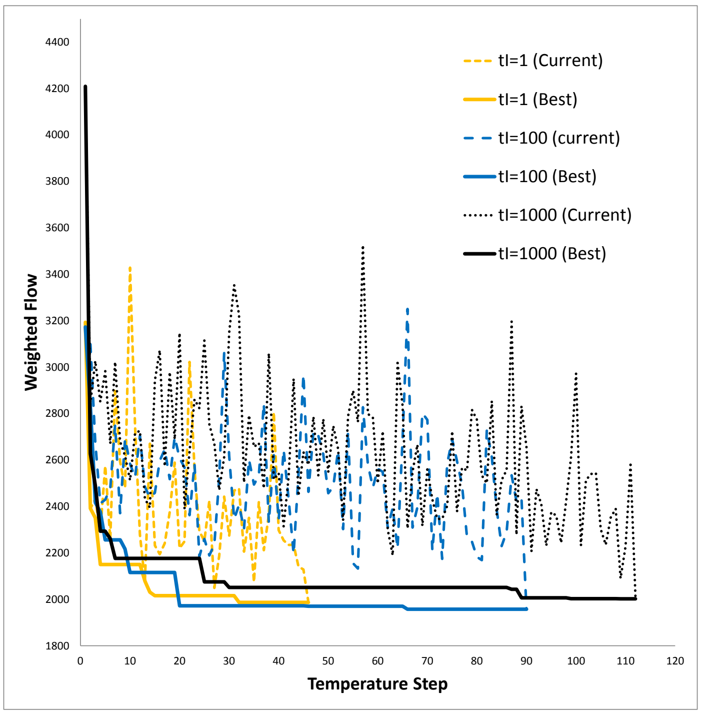

**Figure 8.** Convergence of the HSA [NO RLT CASE/ iter = 100]

## 4.3. Double Width Stockpiles

The treatment of stockpiles on R2, R3 and R4 as double stockpiles is considered here. The division of these stockpiles into two parts results in an additional 22 stockpiles and the requirement for 22 tokens. The HSA was applied ten times to this scenario and the results are summarised in Table 6 and 7. The best solutions for the two scenarios are shown in Figure 9 and 10. A close inspection of the solutions shows that the single side access constraint has been enforced. In comparison to Table 2 and Table 4 results, the effect of the double stockpiles has been a slight increase across all metrics. In retrospection this is to be expected as double width stockpiles amount to imposing additional constraints to scheduling. The CPU time to apply the HSA is greater as there are more piles to assign work to. There is an increase of roughly 16-20% on average for the RLT case. For the No RLT case the CPU time is significantly greater at 33-69%.

**Table 6.** Results for Variant 1 [RLT + WFLOW objective]

| MODE | WFLOW | | | | CMAX | | | | CPU (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | min | avg | max | stdev | min | avg | max | stdev | min | avg | max | stdev |
| 1 | 231815 | 447055 | 654219 | 121090 | 10048 | 10388 | 10765 | 254 | 185 | 189 | 191 | 3 |
| 2 | 50402 | 59277 | 77544 | 7435 | 8811 | 8981 | 9158 | 138 | 2381 | 2506 | 2720 | 132 |
| 3 | 48590 | 60462 | 79741 | 8763 | 8829 | 9127 | 9505 | 196 | 2075 | 2398 | 2747 | 232 |

**Table 7.** Results for Variant 2 [No RLT + CMAX objective]

| MODE | WFLOW | | | | CMAX | | | | CPU (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | min | avg | max | stdev | min | avg | max | stdev | min | avg | max | stdev |
| 1 | 55212 | 65024 | 73275 | 6091 | 2689 | 3251 | 4379 | 523 | 278 | 323 | 357 | 36 |
| 2 | 56186 | 60180 | 62803 | 1929 | 2029 | 2091 | 2145 | 41 | 4125 | 5499 | 6090 | 555 |
| 3 | 55158 | 60596 | 63845 | 2904 | 2034 | 2108 | 2166 | 44 | 3789 | 4105 | 5157 | 383 |

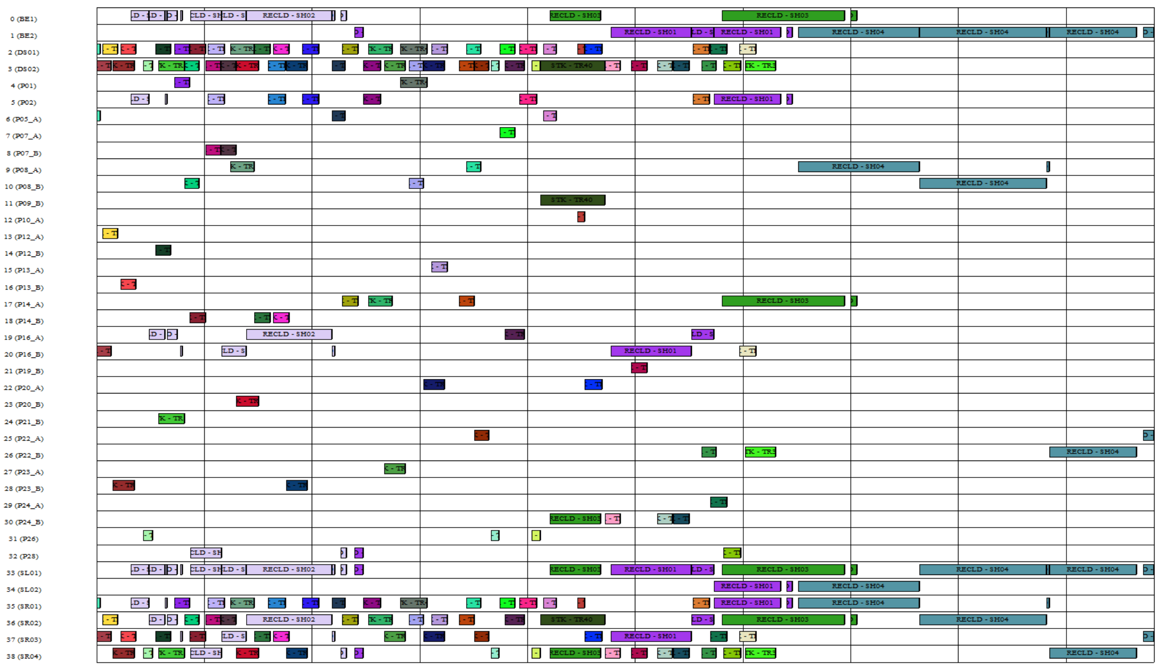
**Figure 9.** Gantt chart of the best solution for the RLT - double width scenario

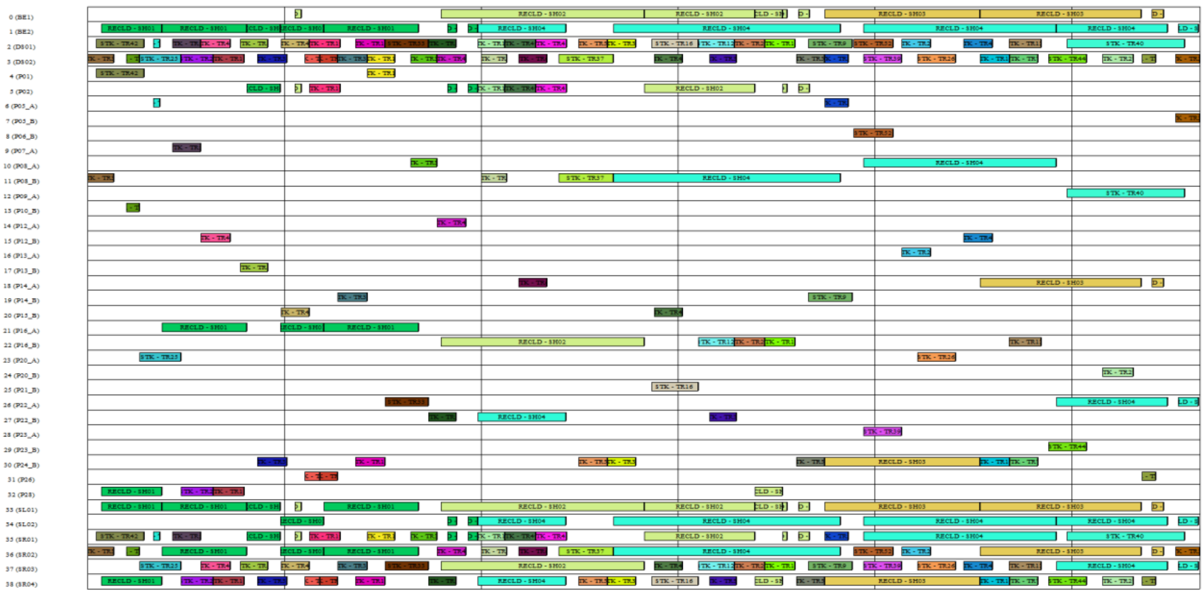
**Figure 10.** Gantt chart of the best solution for the No RLT - double width scenario

### 4.4. Dynamic Stockpiles

In this section our test instance is extended to include dynamic stockpiles. In this scenario the release times for ships and trains have been removed and there are no fixed berthing sequences assumed. In our scenario stockpile P09 and P13 are selected to be replaced. Figure 11 shows the layout after the new stockpiles are added.

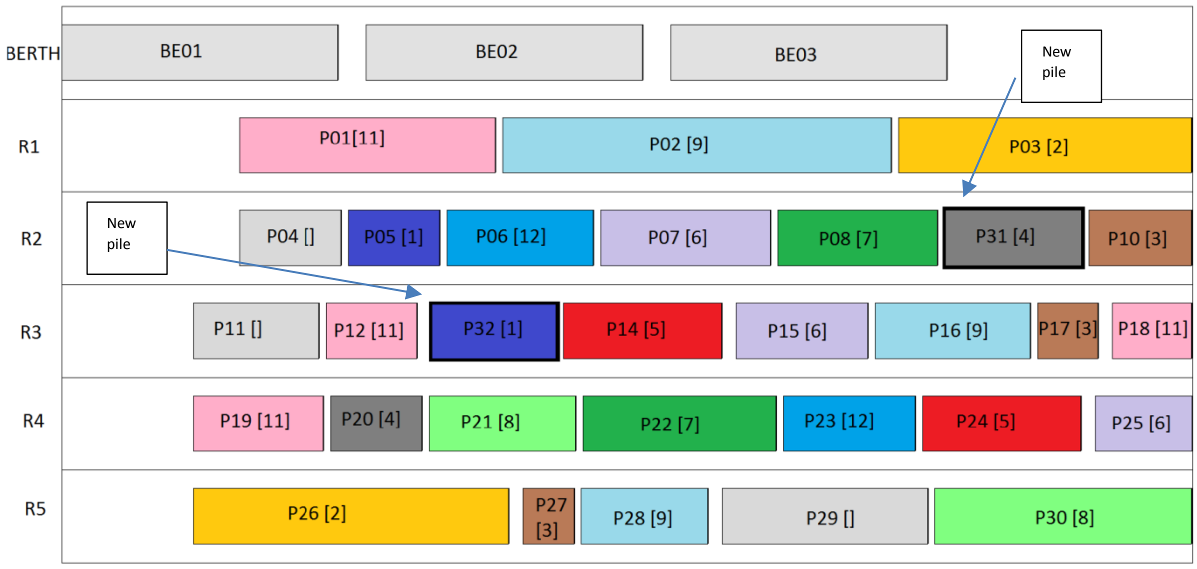

**Figure 11.** Schematic of the CET after P31 and P32 are added and (P09 and P13) are fully reclaimed

The replacement stockpiles are P31 (i.e. P9[REP]) and P32 (i.e. P13[REP]) with product 4 and 1 respectively. Two trains with 10000 tonnes each of product 1 and four trains with 10000 tonnes each of product 4 are also added to bring material to the new stockpiles. Two additional ships are added to remove the material in P09 and P13. The first has a requirement of 29480 of product 10 and the second has a requirement of 60760 of product 13. The following stockpile precedence P09 $\prec$ P31 and P13 $\prec$ P32 are added. These are then converted into additional disjunctive arcs. They continuously change depending on which activities have been assigned to P09, P31, P13 and P32. The HSA was reapplied ten times and the results are summarised in Table 8. The best solution is shown in Figure 12. The makespan of our solution is 3542.21. The weighted flow is 540195.98 and the flow is 64613.1. The most important feature of this solution is that the activities performed on P09 are all finished before the first activity is performed on P09[REP]. This is also true of P13. The later activities performed on P13[REP] are all performed after the last activity on P13. The stockpile precedence mechanism has enforced the relationship between the original stockpile P09 and P13 and their later replacements.

**Table 8.** Results for dynamic pile replacement scenario

| MODE | WFLOW | | | | CMAX | | | | CPU (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | min | avg | max | stdev | min | avg | max | stdev | min | avg | max | stdev |
| 2 | 540196 | 585605 | 722361 | 50942 | 2984 | 3457 | 4163 | 445 | 1760 | 3009 | 4965 | 919 |

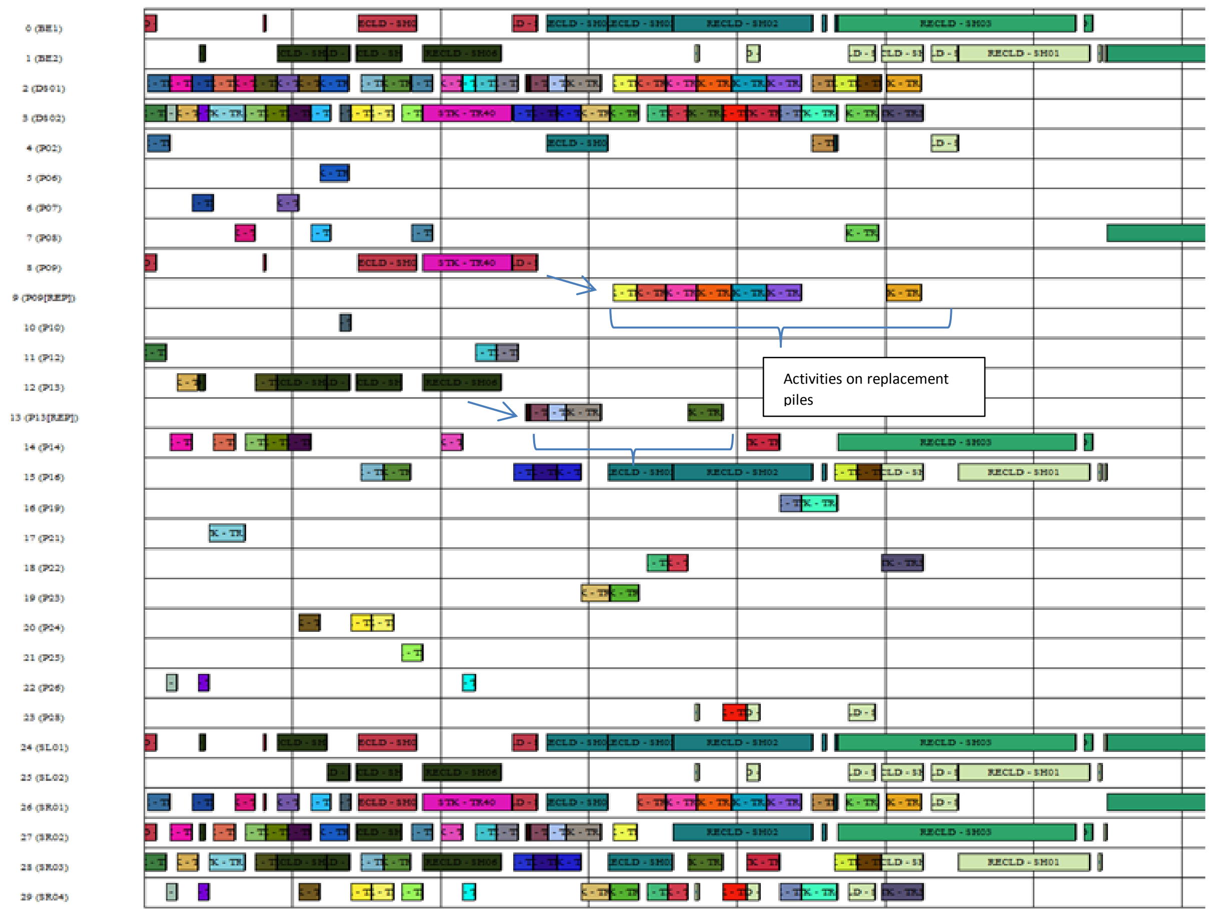

**Figure 12.** Gantt chart of the solution for the stockpile replacement scenario

## 5. Conclusions

In this article a new optimization approach has been developed for coal export terminals and other bulk material handling terminals. Our integrated approach schedules the stacking and reclaiming activities associated with the planned arrival of ships and trains. It also assigns resources to perform the aforementioned activities.

To increase the system’s output and to reduce delays incurred by trains and ships, we treat this integrated decision problem as a flexible job shop with operators (FJSOP) and apply a hybrid meta-heuristic. The application of a FJSOP approach is a relatively new avenue. We believe this article provides the first documented FJSOP approach for CET. In this scheduling problem, each activity can have three or four requisite resources. To the best of our knowledge, no more than two operator types have ever been considered in past papers. Our approach is also significant because it can be used without change, to solve a vast array of scheduling problems in other domains (i.e. for instance health). It is a generic scheduling platform and incorporates a wide range of advanced scheduling constraints.

The incorporation of many additional advanced scheduling constraints is a necessity in this research. Our meta-heuristic is novel as a multitude of advanced features relevant to CET are incorporated, namely accessibility and resourcing restrictions, multiple resource requirements per activity, pre-emptions, stockpile size limitations, dynamic stockpiles, double width stockpiles, proximity and collision detection. These features may occur in other problems, but have not been addressed as comprehensively within one integrated approach before. The meta-heuristic also employs several carefully selected perturbation strategies and several local improvement algorithms. The solution chromosome in our meta-heuristic is a priority ordering of the different activities. It describes the order in which activities are scheduled and is translated into the set of resource sequences using a constructive algorithm. Our choice of solution chromosome greatly simplifies scheduling and results in a significantly reduced search space. The use of a priority ordering is a significant contribution of this article and to our knowledge is an approach rarely taken. We anticipate that a priority ordering could be successfully used in many other multi-sequence scheduling problems. Without our priority ordering chromosome it is necessary to take a more traditional approach whereby resource sequences are explicitly perturbed. Our numerical investigations demonstrate that this is quite difficult, particularly when there are many resource requisites. In fact, the complexity increases as the number of resource requisites increases. In this article we have four. A small change in any one sequence can easily cause positive length cycles to be created. This means that it is very difficult to move from one feasible solution to another without the application of special compound moves or cycle correction devices – both of which are computationally demanding and complex to implement. The net effect is poor convergence and solution quality. Our priority ordering chromosome however is very easy to refine and can easily be translated to produce all the resource sequences. Our approach is also comparatively simpler to implement.

Our solution approach was applied to a real life test problem and numerical investigations demonstrate the advanced features present in CET can be handled. The best solutions can be obtained within 30 minutes on average if a larger number of iterations (i.e. 300) are applied. Further computing time however may be allowed to provide even better solution quality.

In formulating, implementing and testing our scheduling approach we have found that this integrated scheduling problem is deceptively difficult. The pre-emption of ship loading activities is by far the hardest feature as is the interplay between the train and ship jobs. It is quite difficult to choose the correct number of pieces and to insert these pieces intelligently within the schedule. Changing the resource assignment can easily disrupt the quality of the solution during the search – either beneficially or detrimentally. The scheduling of train unloading activities is comparatively easier, particularly when considered independently. To facilitate pre-emptions, the selection of a poor labelling convention can lead to a number of implementation issues. Performing pre-emptions and performing all the necessary accounting is also awkward to encode. At the end of the search the best solution may contain activities which no longer exist plus a set of new activities. There are also a few peculiarities induced by pre-emption and fixed sequences. Fixed sequences actually expand and contract as the search progresses. This occurs when pre-emptions are done or undone.

In future research we will consider how to reduce the computing time further. The assignment of resources is currently handled by our constructive algorithm. Although effective, an alternative strategy is to have a dedicated resource allocation chromosome. How best to manipulate this chromosome in conjunction to the priority ordering chromosome should be further investigated. Also to be determined is whether superior solutions can be obtained with less computational effort. The incorporation of multiple performance metrics, and the development of a multi-criteria optimization approach are also planned. Our optimization approach can test different stockyard configurations and thus may be used to facilitate strategic planning of the stockyard over time, i.e. the “Stockpile Placement Problem (SPP)”. The Variant 3 problem discussed in the introduction, with uncertain ship and train arrivals was not considered in this article, and warrants further attention in the future.

This articles approach was developed for CET and as such, it is necessary to comment upon the implications to managers of CET. This articles approach is a new consideration for staff of CETs and will require them to invest in the development of new IT systems. In our opinion, the FJSOP approach has the potential to increase the overall efficiency and productivity of CET, but only if it is integrated into an appropriate information management system (IMS). It will be necessary to inform the IMS when every activity begins and ends within the terminal. This would enable the scheduling model to identify problems in advance and suggest better ways to operate. The quality of the solutions produced by our optimization approach is important, but how close they are to optimality is predominantly of academic interest. It is debatable whether the determination of optimal schedules is of interest to CET managers and planning staff. To our understanding, the most important requirement for industry is the capability to quickly determine improved schedules, at short notice, particularly in the event of breakdowns, delays and other events.

## REFERENCES

**Agnetis. A., Murgia, G., Sbrilli, S. (2014).** A job shop scheduling problem with human operators in handicraft production. *International Journal of Production Research*, 52(13), 3820-3831.

**Agnetis. A., Flamini. M., Nicosia. G., Pacifici. A. (2011).** A job-shop problem with one additional resource type. *Journal of Scheduling*, 14(3):225–237, 2011.

**Babu, S.A.K.I., Pratap, S., Lahoti, G., Fernande, K.J., Tiwari, M.K., Mount, M., Xiong, Yu.** (2015). Minimizing delay of ships in bulk terminals by simultaneous ship scheduling, stockyard planning and train scheduling. *Maritime Economics and Logistics*, 17 (4), 464-492.

**Boland, N., Gulczynski, D. and Savelsbergh, M. (2012)** A stockyard planning problem. *European Journal of Transportation and Logistics,* 1(3): 197–236.

**Burdett, R.L. and Kozan E. (2009a)**. Scheduling trains on parallel lines with crossover points. *Journal of Intelligent Transportation Systems: Technology, Planning, and Operations, 13*(5), 171-187

**Burdett, R.L., Kozan E. (2009b)**. Techniques for restricting multiple overtaking conflicts and performing compound moves when train scheduling. *Mathematical and Computer Modelling*, 50(1-2), 314-328.

**Burdett R.L. and Kozan E. (2009c).** Techniques for inserting additional trains into existing timetables. *Transportation Research B,* 43(8), 821-836.

**Burdett, R.L., Kozan E. (2010a).** A sequencing approach for train timetabling. *OR Spectrum*, 32(1), 163-193.

**Burdett, R.L., Kozan E. (2010b).** Development of a disjunctive graph model and framework for constructing new train schedules. *European Journal of Operational Research*, 200(1), 85-98.

**Burdett R.L. and Kozan E. (2014)**. An integrated approach for earthwork allocation, sequencing and routing. *European Journal of Operational Research*, 238, 741-759.

**Burdett, R.L., Kozan, E. (2018).** An integrated approach for scheduling health care activities in a hospital. *European Journal of Operational Research*, 264(2), 756-773.

**Corry, P., Kozan, E. (2004).** Job scheduling with technical constraints. *Journal of the Operational Research Society*. 55 (2), 160-169.

**Chen, J.C., Wu, C.C., Chen. C.W., Chen, K.H. (2012).** Flexible job shop scheduling with parallel machines using genetic algorithm and grouping genetic algorithm. *Expert Systems with Applications*, 39, 10016-10021.

**Corry, P., Kozan, E. (2004**). Job scheduling with technical constraints. *Journal of the Operational Research Society*, 55, 160-169.

**Doh, H.H., Yu, J.M., Kim, J.S., Lee, D.H., Nam, S.H. (2013).** A priority scheduling approach for flexible job shops with multiple process plans. *International Journal of Production Research*, 51(12), 3748-3764.

**Escamilla, J., Rodriguez-Molins M., Salido, M.A., Sierra, M.R., Mencia, C., Barber, F. (2012).** Robust solutions to job-shop scheduling problems with operators. 2012 IEEE 24th International Conference on Tools with Artificial Intelligence, 299 – 306.

**Gao, K.Z., Suganthan, P.N., Pan, Q.K., Chua, T.J., Cai, T.X., Chong, C.S. (2014).** Pareto-based grouping discrete harmony search algorithm for multi-objective flexible job shop scheduling. Information Sciences, 289, 76-90.

**Gao, K.Z., Suganthan, P.N., Tasgetiren, M.F., Pan, Q.K., Sun, Q.Q. (2015).** Effective ensembles of heuristics for scheduling flexible job shop problem with new job insertion. *Computers and Industrial Engineering*, 90, 107-117.

**Groflin, H., Pham, D.N., Burgy R. (2011).** The flexible blocking job shop with transfer and set-up times. J Comb Optim, 22, 121-144.

**Hu, D., Yao, Z. (2012).** Stacker-reclaimer scheduling in a dry bulk terminal. *International Journal of Computer Integrated Manufacturing*, 25(11), 1047-1058.

**Kalinowski, T., Kapoor, R., Savelsbergh, M.W.P. (2017).** Scheduling reclaimers serving a stack pad at a coal terminal. *Journal of Scheduling*, 20, 85-101.

**Mencia R., Sierra, M.R., Mencia, C., Varela, R. (2015).** Memetic algorithms for the job shop scheduling problem with operators. *Applied Soft Computing*, 34, 94-105.

**Luscombe, R., Kozan, E. (2016).** Dynamic resource allocation to improve emergency department efficiency in real time. *European Journal of Operational Research*, 255(1), 593-603.

**Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P., Werner, F. (2009).** A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *Computers and Operations Research*, 36, 358-378.

**Pezella, F., Morganti, G., Ciaschetti, G. (2008).** A genetic algorithm for the flexible job-shop scheduling problem*. Computers and Operations Research*, 35, 3202-3212.

**Quintiq (2017).** Case Study: Port Kembla Coal Terminal improves performance in six months with Quintiq. http://www.quintiq.com/industries/ports-and-terminals.html.

**Sauvey C., Trabelsi, W. (2015).** Hybrid job shop scheduling with mixed blocking constraints between operations. Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference. Date 8-11 Sept. 2015, page 1-8. DOI: 10.1109/ETFA.2015.7301538.

**Stack Overflow (2017)**. https://stackoverflow.com/questions/19085937/finding-intervals-of-a-set-that-are-overlapping

**Van Laarhoven P. J. M., Aarts E. H. L., Lenstra J. K. (1992).** Job shop scheduling by simulated annealing. *Operations Research*, 40(1), 113-125.

**Xia, W., Wu, Z. (2005).** An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers and Industrial Engineering*, 48, 409-425.

**Yazdani, M., Amiri, M., Zandieh, M. (2010).** Flexible job-shop scheduling with parallel variable neighbourhood search algorithm. *Expert Systems with Applications*, 37, 678-687.

**Zeng, C., Tang, J., Yan, C. (2014).** Scheduling of no buffer job shop cells with blocking constraints and automated guided vehicles. *Applied Soft Computing*, 24, 1033-1046.

**Zhang, G., Shao, X., Gao, P. L.L. (2009).** An effective hybrid particle swarm optimization algorithm for multi-objective flexible job shop scheduling problem. *Computers and Industrial Engineering*. 56, 1309-1318.

**Zhang, J., Yang, J. (2016).** Flexible job-shop scheduling with flexible workdays, pre-emption, overlapping in operations and satisfaction criteria: an industrial application. *International Journal of Production Research*, 54(16), 4894-4918.

## Appendix A. Disjunctive Graph Model

$$DJG = (V, E = E^C \cup E^D) \tag{A1}$$

$$V = \{so, si\} \cup [\cup_{r\in R}\{so_r, si_r\}] \cup [\cup_{a\in A}\{a\}] \tag{A2}$$

$$E^{\mathrm{C}} = [\cup_{r\in R}\{(so, so_r, rdy_r)\}] \cup [\cup_{r\in R}\{(si_r, si, 0)\}] \cup \left[\cup_{J_i\in J} \cup_{(a,a')\in PRE_i}\{(a, a', \omega_{a,a'})\}\right] \\ \cup \left[\cup_{a\in A|rlt_a>0}\{(so, a, rlt_a)\}\right] \quad \text{(A3)}$$

$$E^{\mathrm{D}} = \left[\cup_{r\in R}\{(so_r, \sigma_r[1], setup_{r,a})\}\right] \cup \left[\cup_{r\in R} \cup_{k=1,..,|\sigma_r|-1}\{\mathbf{DJA}(r, \sigma_r[k], \sigma_r[k+1])\}\right] \\ \cup \left[\cup_{r\in R}\{(\sigma_{r,|\sigma_r|}, si_r, 0)\}\right] \quad \text{(A4)}$$

$$\mathbf{DJA}(r, a, a') = \begin{cases} (a, a', setup_{r,a,a'}) \text{ if } a \in ideal \cup nowait \\ (a^*, a', setup_{r,a,a'} - p_{a^*} - \omega_{a,a^*})\ if\ a \in block;\ a^* = \mathbf{succ}(a) \end{cases} \quad \text{(A5)}$$

The conjunctive and disjunctive arcs of the graph are respectively denoted by $E^C$and $E^D$. Every activity has a release time ($rlt_a$) and every resource has a ready time ($rdy_r$). By default these are zero if none are provided. A finish start lag may be required between two activities of a job. This is denoted by $\omega_{a,a'}$ and needs to be included on conjunctive arcs. Every job has a set of activities; the successor activity of any activity $a \in A_i$ is given by the function $\mathbf{succ}(a)$. The disjunctive arcs are generated by the function $\mathbf{DJA}(r, a, a')$. This function generates an appropriate disjunctive arc for the precedence $a \prec a'$. The first arc is needed when activity $a$ is ideal or it is no-wait. If there is a no intermediate storage (i.e. blocking) condition with its successor(s) then the second arc is needed. Two types of setup need to be included. The setup incurred by resource $r$ between activity $a$ and $a'$ is denoted as $setup_{r,a,a'}$. The setup incurred by resource $r$ before processing activity $a$ is denoted $setup_{r,a}$. Sequence dependent setups are calculated in the following way for mobile resources like stacker reclaimer and ship loaders:

$$setup_{r,a,a'} = fixed + \begin{cases} 0\ if\ l_a = l_{a'} \\ \frac{60.\mathbf{D}(l_a,l_{a'})}{spd_r}\ if\ l_a \neq l_{a'} \end{cases};\ setup_{r,a} = \begin{cases} 0 \text{ if } l_a = orig_r \\ \frac{60.\mathbf{D}(orig_r,l_a)}{spd_r} \end{cases} \quad \text{(A6)}$$

The location of the two activities is used to compute the distance travelled. The resources speed is then used to compute the travel time. A fixed time component is also included. If there is no prior activity, the second equation is used, which involves the resources original location (i.e. an input parameter).

**Appendix B. Computing pile profiles**

```
$\forall p \in P$:      // For each pile
{
    $sz_p = sz_p^I$;  // Assign initial pile size
    $\forall k \in [1, |\sigma_p|]$: // For each activity assigned to pile p
    {
        $a = \sigma_p[k]$; $\lambda = 1$ if $a \in STK$ else $-1$ if $a \in REC \cup RECLD$
        $\Omega_p = \Omega_p + \{(start_a, sz_p)\}$;
        If $(sz_p < 0)$ {
            $\delta = sz_a$;
            If$(sz_p + \lambda.sz_a > 0)${ $t = start_a - sz_p/rate_a$; $\Omega_p = \Omega_p + \{(t, 0)\}$; $\delta = -sz_p$;}
            **RecordDeficit**$(a, start_a, \delta)$;
        }
        If $(sz_p \geq 0)$ {
            If$(sz_p + \lambda.sz_a < 0)$ { $t = start_a + sz_p/rate_a$; $\Omega_p = \Omega_p + \{(t, 0)\}$;
            **RecordDeficit**$(a, t, sz_a - sz_p)$; }
        }
        If $(sz_p \leq UB_p)$ {
            If$(sz_p + \lambda.sz_a > UB_p)$ {
             $t = start_a + (UB_p - sz_p)/rate_a$; $\Omega_p = \Omega_p + \{(t, UB_p)\}$;
            **RecordOverload**$(a, t, sz_p + sz_a - UB_p\ )$;}
            }
            If $(sz_p > UB_p)$ {
                $\delta = sz_a;$
                If$(sz_p + \lambda.sz_a < UB_p)$ { $t = start_a + (sz_p - UB_p)/rate_a;$
                $\Omega_p = \Omega_p + \{(t, UB_p)\}; \delta = sz_p - UB_p\}$
                **RecordOverload**$(a, start_a, \delta);$
            }
            $sz_p = sz_p + \lambda.sz_a; \Omega_p = \Omega_p + \{(end_a, sz_p)\};$
        }
        $\Omega_p = \Omega_p + \{(cmax, sz_p)\};$ // Record level at schedule completion
}
```

## Appendix C. Double width stockpiles

Stockpiles are generally accessible from both sides of a bund. When they are not, it is because the pads are unusually deep. In those circumstances stockpiles are described as double width. One way to approach this situation is to treat such piles as two piles of the same product, see Figure C1.

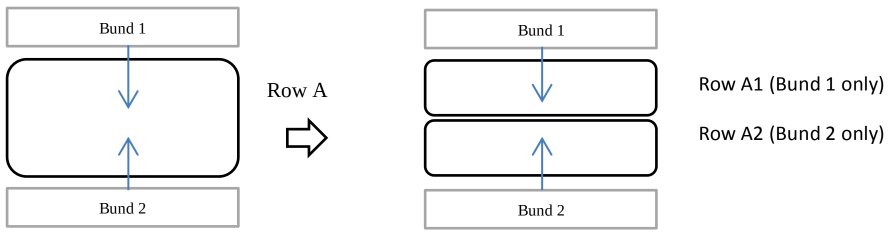

**Figure C1.** Traditional single width pile versus double width pile and division

It is then necessary to define a new row. It is important to note that every other double width pile (i.e. along that row) should also be divided in the same way. Each row is then accessible by only one bund and hence only by the SRs on that bund. A CET may contain both single and double width stockpiles. In that situation, the pad is still treated as two separate rows. The single width piles are maintained but any double width piles must be divided.

When double width stockpiles are present, an important consideration is whether two SRs can access the two sides at the same time. Concurrent access is not allowed due to potential collisions and problems with material slumping onto stacker-reclaimers. To ensure that only one side is used at any time (i.e. single side access) a dummy resource, called a token, is introduced. The idea is that this token must be acquired in order to use each side. Evidently, both sides cannot acquire this resource simultaneously. Tokens are a subset of the resources (i.e. $TOK \subset R$) and must be defined upfront. It is important to note that these additional resources have their own sequence of activities. In addition, it is necessary to point out that in the context of CET, the tokens are a requisite of the stockpile and not the activity. Hence the token chosen depends only on which stockpile is chosen.

## Appendix D. Dynamic creation and destruction of stockpiles

Over time existing stockpiles may be fully reclaimed and new stockpiles with different dimension and product may be created in their place. For example, in Figure D1, there is a future requirement for the creation of stockpile P6 and P7 in place of P2, P3 and P4. The new stockpiles are assumed empty when created.

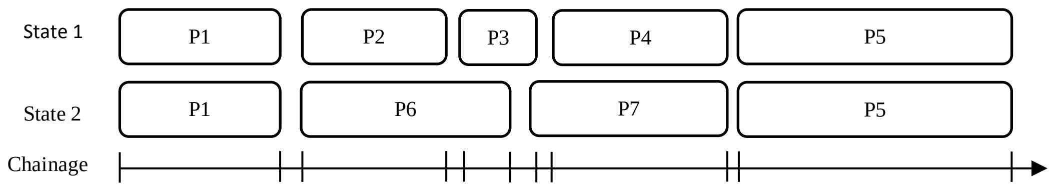

**Figure D1.** Changing stockpile configurations

It is necessary to know the stockpiles and their locations upfront to handle these types of scenarios. Stockpile precedence conditions are introduced to ensure that activities assigned to earlier piles are completed before activities assigned to later piles. In the context of Figure D1, it is evident that any activity assigned to stockpile P6 and P7 must occur after those activities assigned to P2, P3 and P4. This is equivalent to the following conditions: $\mathrm{P6} \succ \{\mathrm{P2}, \mathrm{P3}\}$ and $\mathrm{P7} \succ \{\mathrm{P3}, \mathrm{P4}\}$. These precedence statements are translated into new precedence relations between the activities assigned to those stockpiles. Without loss of generality it is necessary to generate set $\{(a, a') | a \in \sigma_p, a' \in \sigma_{p'}\}$ for any stockpile precedence $p \prec p'$ and to add disjunctive arcs for each pair in that set. Here $\sigma_p$ and $\sigma_{p'}$ are the current sequences for stockpile $p$ and $p'$. These disjunctive arcs are dynamic as the activities assigned to those stockpiles are variable. They can only be identified after activities are assigned to stockpiles. At that point in time they are added to the disjunctive graph. Once the disjunctive graph evaluation is complete, they are removed. An example is shown below:

**Example:** Given $\sigma_{\mathrm{P2}} = (a_1, a_2)$; $\sigma_{\mathrm{P3}} = (a_3)$; $\sigma_{\mathrm{P4}} = (a_4, a_5, a_6)$; $\sigma_{\mathrm{P6}} = (a_7, a_8)$; $\sigma_{\mathrm{P7}} = (a_9)$ and the stockpile precedence $\{\mathrm{P2}, \mathrm{P3}\} \prec \mathrm{P6}$ and $\{\mathrm{P3}, \mathrm{P4}\} \prec \mathrm{P7}$, DJA are required for the following pairs:

$$(\sigma_{\mathrm{P2}} \cup \sigma_{\mathrm{P3}}) \times \sigma_{P6} = \{a_1, a_2, a_3\} \times \{a_7, a_8\} = \{(a_1, a_7), (a_1, a_8), (a_2, a_7), (a_2, a_8), (a_3, a_7), (a_3, a_8)\}$$
$$(\sigma_{\mathrm{P3}} \cup \sigma_{\mathrm{P4}}) \times \sigma_{P7} = \{a_3, a_4, a_5, a_6\} \times \{a_9\} = \{(a_3, a_9), (a_4, a_9), (a_5, a_9), (a_6, a_9)\}$$

It is worth pointing out that every reclaim and stack activity has a set of candidate stockpiles and from those a single stockpile is allocated to the activity. In theory, those candidate stockpiles will change over time when existing stockpiles vanish and others take their place. There is no need however to change candidate stockpiles, but a complete list should be defined upfront however. For example, if an activity is assigned to stockpile P6 and that activity is sequenced incorrectly, say before activities assigned to P2 and P3, then a positive length cycle will occur. Hence that activity must be reassigned to another stockpile if possible or else re-sequenced to a later time. If an activity is sequenced correctly then an issue does not arise.

There is also the possibility that stockpiles are expanded or contracted at future times. In Figure D2, pile P2 increases in dimension and P4 decreases. To avoid confusion P2 and P4 may be defined as new stockpiles P6 and P7, and the aforementioned approach may be taken. At the point of creation however, the amount of material in P6 will need to be equal to the current size of P4.

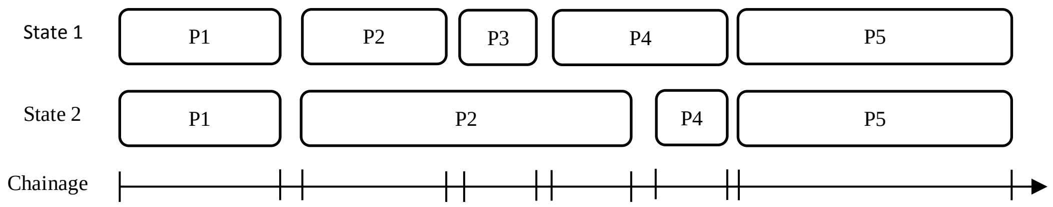

**Figure D2.** Inflating and contracting stockpiles

## Appendix E. Proximity restrictions and maintaining separation

More than one SR may be located on each bund within the stockyard of a CET. This is problematic for scheduling, as two SR cannot be in the same location at the same time, nor can they be positioned within a given proximity of each other. In addition, SR cannot pass through each other.

To schedule SR correctly and to maintain a safe separation, a number of avenues are possible. One approach is to compute in advance, the position of each SR over time. This position profile can then be compared to other nearby SR. Conflicts can be identified and adjustments can be made. Algorithms for doing this are well tested, for example in the train scheduling domain. To determine the position of SRs it is necessary to divide the bund into smaller regions as demonstrated in Figure E1, and to analyse the occupancy of those regions over time.

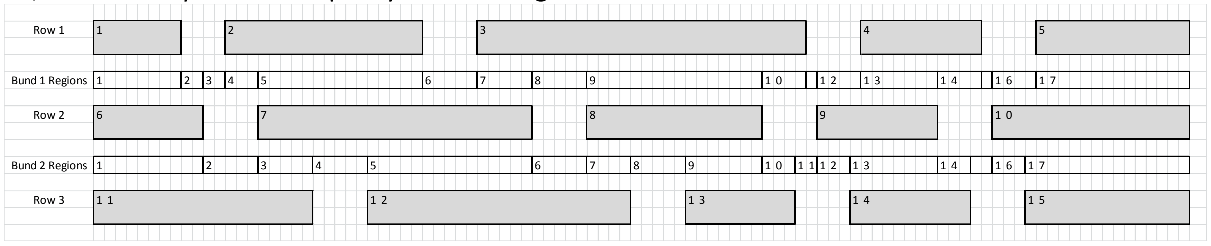

**Figure E1.** Demonstration of regions required

***Creating Regions***: It is necessary to create a list for each bund. For each stockpile in each row, it is necessary to append the stockpiles left and right position to that list. After which it is necessary to sort the list in ascending order of magnitude. A region should be created for each pair of adjacent points, provided they have different values. As a last step it is necessary to add stockpile references to each region and vice versa. The set of regions is defined for each $b \in BUND$ as $REG_b$ and the occupancy of region $reg \in REG_b$ as $OCC_{reg}$.

***Single and Multiple Row Scenarios***: Every region associated with a stockpile will have a valid sequence of occupancy due to the presence of stockpile sequences. Regions between stockpiles (i.e. gaps) however do not. In single row scenarios, to determine the presence or absence of collisions and other proximity violations, it is sufficient to determine the occupancy of the gaps. When a bund accesses two rows of stockpiles, the process is more involved. The determination of gap occupancy is insufficient as one SR can be accessing a stockpile on one side and another SR can be accessing a stockpile on the other side. In multiple row scenarios the occupancy of all regions must be computed.

***Determining Region Occupancy***: The occupancy of each region is determined via Algorithm 1. The first step is to build a path for each SR. The SR path is an ordered list of tuples. Each tuple contains a time interval and a reference to the region occupied, i.e. $path_r = [(itl_1, reg_1), (itl_2, reg_2), \ldots,]$ where $r \in SR$ and $itl = [t_1, t_2]$. **The time intervals of stacking and reclaiming are taken into account as well as SR repositioning between stockpiles.** After these paths are constructed, the occupancy of each region is computed. To do this, the intervals within SR paths are grouped by region. In other words, the time intervals of each SR are assigned to the appropriate region list. Each region is then analysed for overlapping intervals. A count of the number of overlapping intervals is made. Any overlapping interval implies the presence of multiple SR. A count of zero implies that no collisions occur. To determine overlapping intervals an efficient O(N + N log (N)) algorithm is used. Here N refers to the number of time intervals. This algorithm is shown in Algorithm 2 and is a variant of one found at StackOverflow (2017). It first creates a list of time points. A time point occurs for the start and end of each interval of occupancy. Each time point is assigned a 1 or -1 respectively depending upon whether the time point is the start or end of the interval. This list is then sorted chronologically, a process which breaks up the domain into unique non overlapping intervals. The last step is to count the occupancy of each interval occurring between adjacent points. This is achieved by traversing the list and incrementing a counter of the aforementioned 1 and -1 values attached to each pt. The counter is decremented when an original interval ends and incremented when an original interval begins. The compare function considers the first value in the pairs and then the second if need be.

The sequence of resources (and hence activities) occupying each region can be constructed by attaching additional information with the time interval, i.e. a reference to the resource.

```
Algorithm 1: CheckViolation()
{
    1. $\forall sr \in SR$: $path_{sr}$ = MakePath(sr); UpdateRegion($path_{sr}$);
    2. $BUND' = \{b \in BUND \mid |SR_b| > 1\}$; // Note: $SR_b = \{sr \mid sr \in SR, bund_{sr} = b\}$
    3. $\forall b \in BUND'$:
        3a. $count_b = 0$;
        3b. $\forall reg \in REG_b$: $count_b$ += CountOverlaps($OCC_{reg}$,2);
}
```

```
Algorithm 2. CountOverlaps($itl, \alpha$)
{
    1. $pts = \cup_{\forall [t_1,t_2] \in itl} \{(t_1, 1) + (t_2, -1)\}$; // Create a list of tuples
    2. $pts.sort(\mathbf{compare})$; // Sort the tuples. Prioritize by earliest time and interval start
    3. $itl = \emptyset$; // Discard the original list
    4. $sum = 0$; // Initialise counter for # of open intervals
    5. $n = 0$; // Initialise counter for number of intervals with occupancy greater than 1
    6. for($i \in [1, |pts|]$) // Iterate through the sorted list. Note: $pts = [(a_1, b_1), (a_2, b_2), \ldots, ]$
        6a. $sum$ += $b_i$; // Current occupancy
        6b. if($a_i \neq a_{i+1}$ and $sum > \alpha$) { $itl$ += $[a_i, a_{i+1}, sum]$; $n = n + 1$; } // Record interval
    7. return $n$;
}
```

```
Algorithm 3: compare($(a, b), (c, d)$) { if($a = c$) return ($b > d$); else return ($a < c$); }
```

**Appendix F. Details of the constructive algorithm**

```
Algorithm 4: ConstructByPriority($order$)
{
    1. $level_p = sz_p \quad \forall p \in P$; // Record initial stockpile level
    2. $free_r = rdy_r \quad \forall r \in R$; // Record resource ready times
    3. For $k = 1, .., |order|$ // Iterate through activities in the priority ordering
        3a. $a = order[k]$; // Next activity
        3b. $earliest\_start$ = AcquireResourcesAsap($a, free, level$);
        3c. $\forall r \in R_a$: // For each resource assigned
            If ($r \in P$) $level_r$ += $sz_a$; // Resource is a stockpile. Update the amount.
            If ($\sigma_r \notin FIXED\_SEQUENCE$) $\sigma_r = \sigma_r + a$; // Append to sequence
            $free_r = earliest\_start + p_a$; // Update the free time
            If there is a token associated with the resource:
                $free_{tok} = earliest\_start + p_a$; // Update the tokens ready time
    }
    Convert sequences to disjunctive arcs and evaluate the solution;
}
```

## Appendix G. Case study details

**Table G1.** Stockpile Information

| P | LEFT (metres) | RIGHT (metres) | SIZE (tonnes) | ROW | PROD |
|---|---|---|---|---|---|
| **01** | 342.5 | 620.8 | 8550 | R1 | 11 |
| **02** | 628.3 | 1049.1 | 9230 | R1 | 9 |
| **03** | 1056.6 | 1374.9 | 28070 | R1 | 2 |
| **04** | 342.5 | 453.9 | 14680 | R2 | - |
| **05** | 461.4 | 560.3 | 40944.2 | R2 | 1 |
| **06** | 567.8 | 726.7 | 26320 | R2 | 12 |
| **07** | 734.2 | 918.1 | 43200 | R2 | 6 |
| **08** | 925.6 | 1099.5 | 78900 | R2 | 7 |
| **09** | 1107 | 1255.9 | 22110 | R2 | 10 |
| **10** | 1263.4 | 1374.8 | 3990 | R2 | 2 |
| **11** | 292.5 | 429.5 | 7340 | R3 | - |
| **12** | 437 | 535.5 | 17100 | R3 | 11 |
| **13** | 550.5 | 686.5 | 43400 | R3 | 13 |
| **14** | 694 | 865.4 | 70080 | R3 | 5 |
| **15** | 880.4 | 1023.5 | 34560 | R3 | 6 |
| **16** | 1031 | 1200.5 | 105489 | R3 | 9 |
| **17** | 1208 | 1273.3 | 11970 | R3 | 3 |
| **18** | 1288.3 | 1375 | 51300 | R3 | 11 |
| **19** | 292.5 | 434.2 | 25650 | R4 | 11 |
| **20** | 441.7 | 540.9 | 29800 | R4 | 4 |
| **21** | 548.4 | 707.6 | 47340 | R4 | 8 |
| **22** | 715.1 | 924.3 | 47340 | R4 | 7 |
| **23** | 931.8 | 1076 | 32900 | R4 | 12 |
| **24** | 1083.5 | 1254.9 | 96360 | R4 | 5 |
| **25** | 1269.9 | 1375.2 | 0 | R4 | 6 |
| **26** | 292.5 | 634.8 | 12030 | R5 | 2 |
| **27** | 649.8 | 705.8 | 3990 | R5 | 3 |
| **28** | 713.3 | 850.6 | 36920 | R5 | 9 |
| **29** | 865.6 | 1089.1 | 18140 | R5 | - |
| **30** | 1096.6 | 1374.9 | 0 | R5 | 8 |

**Table G2.** Berth Information

| BE | LEFT (metres) | RIGHT (metres) |
|---|---|---|
| **01** | 150 | 450 |
| **02** | 480 | 780 |
| **03** | 810 | 1110 |

**Table G3.** Ship Information

| SH | #ACT | SIZE (tonnes) | RLT | PILE REQ | PROD |
|---|---|---|---|---|---|
| **01** | 2 | (79310,2530) | 5875 | P02,P16,P28 | 9 |
| **02** | 2 | (73230,2230) | 6489.4 | P02,P16,P28 | 9 |
| **03** | 2 | (70950,2630) | 7056.4 | P14,P24 | 5 |
| **04** | 2 | (138850,4460) | 7609 | P08,P22 | 7 |

**Table G4.** Train Information

| TR | SIZE (ton) | TIME (min) | RLT | PILE REQ | PROD |
|---|---|---|---|---|---|
| **01** | 995.79 | 12.26 | 5760 | P05 | 1 |
| **02** | 5270.99 | 49.32 | 5760 | P02,P16,P28 | 9 |
| **03** | 8550 | 51.06 | 5780.7 | P01,P12,P18,P19 | 11 |
| **04** | 6580 | 71.64 | 5815.2 | P06,P23 | 12 |
| **05** | 8680 | 51.24 | 5840.16 | P13,P30 | 13 |
| **06** | 4010 | 32.1 | 5916.66 | P03,P26 | 2 |
| **07** | 8550 | 52.2 | 5956.98 | P01,P12,P18,P19 | 11 |
| **08** | 7890 | 86.52 | 5950.92 | P21 | 8 |
| **09** | 8760 | 51.78 | 5969.64 | P14,P24 | 5 |
| **10** | 8550 | 52.26 | 6017.7 | P01,P12,P18,P19 | 11 |
| **11** | 7890 | 50.16 | 6029.76 | P08,P22 | 7 |
| **12** | 8640 | 51.78 | 6095.4 | P07,P15,P25 | 6 |
| **13** | 9230 | 56.46 | 6131.64 | P02,P16,P28 | 9 |
| **14** | 7890 | 79.86 | 6107.94 | P08,P22 | 7 |
| **15** | 8640 | 52.08 | 6173.88 | P07,P15,P25 | 6 |
| **16** | 7450 | 75.36 | 6217.86 | P20 | 4 |
| **17** | 8760 | 53.82 | 6270.96 | P14,P24 | 5 |
| **18** | 9230 | 56.82 | 6334.62 | P02,P16,P28 | 9 |
| **19** | 8760 | 52.38 | 6351.96 | P14,P24 | 5 |
| **20** | 6580 | 70.68 | 6393.84 | P06,P23 | 12 |
| **21** | 9230 | 55.86 | 6448.8 | P02,P16,P28 | 9 |
| **22** | 6990 | 43.56 | 6546.9 | P05 | 1 |
| **23** | 8760 | 54.3 | 6581.4 | P14,P24 | 5 |
| **24** | 9230 | 59.1 | 6652.14 | P02,P16,P28 | 9 |
| **25** | 8760 | 80.7 | 6668.34 | P14,P24 | 5 |
| **26** | 6580 | 70.86 | 6721.32 | P06,P23 | 12 |
| **27** | 8550 | 88.92 | 6775.56 | P01,P12,P18,P19 | 11 |
| **28** | 7370 | 215.94 | 6692.22 | P09 | 10 |
| **29** | 7890 | 49.44 | 6804 | P08,P22 | 7 |
| **30** | 7450 | 70.08 | 6757.98 | P20 | 4 |
| **31** | 8680 | 53.76 | 6878.88 | P13,P30 | 13 |
| **32** | 7890 | 48.66 | 6995.46 | P08,P22 | 7 |
| **33** | 8760 | 51.96 | 6971.82 | P14,P24 | 5 |
| **34** | 4010 | 26.1 | 7078.98 | P03,P26 | 2 |
| **35** | 7890 | 47.34 | 7023.24 | P08,P22 | 7 |
| **36** | 8640 | 51.78 | 7107.72 | P07,P15,P25 | 6 |
| **37** | 9230 | 65.34 | 7126.02 | P02,P16,P28 | 9 |
| **38** | 4010 | 27.54 | 7215.66 | P03,P26 | 2 |
| **39** | 9230 | 56.34 | 7170.24 | P02,P16,P28 | 9 |
| **40** | 6990 | 43.86 | 7252.26 | P05 | 1 |
| **41** | 8760 | 52.2 | 7312.44 | P14,P24 | 5 |
| **42** | 3990 | 24.36 | 7368.42 | P10,P17,P27 | 3 |
| **43** | 8550 | 53.58 | 7356.96 | P01,P12,P18,P19 | 11 |
| **44** | 7450 | 56.46 | 7381.44 | P20 | 4 |
| **45** | 8760 | 51.6 | 7565.04 | P14,P24 | 5 |
| **46** | 8760 | 54.06 | 7687.14 | P14,P24 | 5 |
| **47** | 9230 | 56.04 | 7753.02 | P02,P16,P28 | 9 |
| **48** | 7890 | 48.6 | 7782.48 | P08,P22 | 7 |
| **49** | 8760 | 54.24 | 7812.6 | P14,P24 | 5 |
| **50** | 9230 | 58.14 | 7855.14 | P02,P16,P28 | 9 |
| **51** | 7890 | 100.08 | 7869 | P08,P22 | 7 |
| **52** | 9230 | 55.26 | 7908.36 | P02,P16,P28 | 9 |

**Table G5.** Product on hand: Row 1 – from piles, Row 2 – from ships; Row 3 – total amount;

| Type 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40944.2 | 40100 | 19950 | 29800 | 166440 | 77760 | 126240 | 65480 | 151639 | 22110 | 102600 | 59220 | 43400 |
| 14975.79 | 12030 | 3990 | 22350 | 87600 | 25920 | 55230 | 7890 | 88340.99 | 7370 | 42750 | 19740 | 17360 |
| 55919.99 | 52130 | 23940 | 52150 | 254040 | 103680 | 181470 | 73370 | 239979.99 | 29480 | 145350 | 78960 | 60760 |