Construction of factory schedules using reverse simulation

A. P. Reynolds and G. P. McKeown

School of Computing Sciences, University of East Anglia, Norwich, UK

Abstract. A real-world, multi-stage, industrial scheduling problem is presented. An algorithm is described that converts a sequence of jobs into a complete schedule. Backward simulation is used to determine minimum storage requirements when scheduling each job, and to calculate the minimum amount of delay required. Combining this algorithm with a metaheuristic, such as simulated annealing, results in an effective algorithm for schedule optimization.

Keywords: scheduling; simulated annealing; simulation

1 Introduction

There has been much research into the planning and scheduling of operations within manufacturing plants and research continues to this day. Such plants may produce many different types of product, such as food [1], clothing [2], steel [3] or in the case of this paper, detergent. While dozens of off-the-shelf scheduling tools are available today, Mazziotti and Horne Jr. [2] note that their underlying capabilities may not meet the needs of systems requiring scheduling and suggest the creation of simulation-based schedulers.

The real-world scheduling problem discussed here differs from the more classical scheduling problems in a number of respects. The product to be scheduled is continuous in nature. Machines within the plant may process the product in batches, but more frequently operate continuously. Storage units must be modelled explicitly. Earlier research led to the production of an approach combining simulated annealing with a simulation of the plant [4, 5] that produced high quality results for some factories. In this paper we improve on these results by replacing the plant simulation with a schedule constructor that uses reverse simulations to determine how to schedule individual jobs.

This paper presents an algorithm for the construction of complete schedules from simple solutions manipulated by metaheuristics. Section 2 describes the type of problem to be solved. Section 3 briefly describes the approaches taken in...
the literature, focusing on those based on metaheuristics. A top-level description of the schedule construction algorithm is presented in section 4, which is followed by more detailed description of the techniques used in sections 5–7. Section 8 describes how this algorithm is combined with simulated annealing to produce an effective schedule optimizer, with results presented in section 9.

2 An industrial scheduling problem

2.1 Plant layout

The factories to be scheduled consist of a number of stages. Processing stages alternate with storage stages, as shown in figure 1. The first stage produces the bulk of the product from raw materials. These base products are then modified in different ways by processing stages later in the plant, to form product variants. Processing stages may also handle packaging of product variants into different types of container.

![Plant Layout Diagram]

Each stage contains a number of processing or storage units. There need not be complete physical connectivity between consecutive stages, for example, see the first three stages in figure 1. In addition, there may be constraints on the number of units that a given unit may feed or be fed from simultaneously,
regardless of the number of physical connections. For example, it may be possible to connect a storage unit to each of the downstream processing units, but it may be constrained to feed only one at a time.

The examples considered in this paper are all three stage plants, though we plan to extend the algorithm to handle factories with five or more stages.

### 2.2 Products

In classical scheduling problems, each product may only be handled by one machine at a time. In contrast, the product manufactured by the factories under consideration in this paper is continuous in nature. Furthermore, most processing units produce product continuously, rather than in batches. Therefore, it is possible, and indeed desirable, to have one product being handled by more than one machine at a time. At any moment in time, a base product may be being produced by more than one processing unit in the first stage of the plant, while also being fed through the storage and other processing stages to the last stage of the plant.

Not only do the products change as they move through the plant, the number of different product types handled also increases towards the last stage of the plant. Each base product may be used to make more than one product variant. These may be packed in many different types of container to produce the final products. Furthermore, in the some stages of the plant the quantity of product may also increase as extra ingredients are added.

The scheduling problem requires the operation of the resources of the plant to be organized so as to produce enough of each product to meet the demand for the specified period. The demand for each final product must be provided in the description of the problem instance.

### 2.3 Units

Each processing stage of a plant consists of a number of processing units. Each individual unit may be able to handle only a subset of the products in that plant stage. Two or more machines may be able to handle the same product. These machines need not be identical; they may process the product at different rates and they may handle different subsets of the products. If two machines can handle a task, the task can be assigned to either one of the machines, or, since the product is continuous, shared over both.

Processing units are modelled either as continuous processing units or as batch processing units. Continuous processing units must, at all times, either run at full speed, or be idle. Batch units may not be used as temporary storage vessels; once the batch is completed, it must be transferred to a downstream unit.

Storage stages consist of either a number of storage units, each of which may contain only one product at a time, or a flexible storage unit that may contain any combination of products.
2.4 Objective

A valid schedule must meet the demand for all products by the end of a specified time period. In the case studies, the time period is usually one working week, consisting of either 144 hours or 120 hours, depending on the pattern of work shifts. In this paper, this constraint is not modelled directly. Instead, the makespan (the time from the start of the week to the end of the last operation in the schedule) is used as the objective function, and is minimized.

2.5 Other features and constraints

Setup times Sequence dependent setup times may occur on all units except flexible storage.

Minimum run lengths Once a task starts on a final stage processing unit, it should not be interrupted or delayed until the task is finished. Here, a task is the production of all of the associated product assigned to the unit. On other units, it is desirable to reduce the number of small run lengths on a machine. A machine schedule with many interruptions is difficult to implement on the factory floor.

Maintenance periods Each unit may require maintenance. During maintenance periods the unit cannot process any product, or perform a setup operation.

Product sequence constraints Final stage units may be constrained to process products in only certain orders. For example, there may be the constraint that a white product may not be packed immediately after a black one, or that the products must be ordered according to the colour sequence white–blue–black.

3 The application of metaheuristics in the literature

Techniques applied to this problem include mixed integer linear and non-linear programming [6–8] and metaheuristics including simulated annealing [4, 5, 9] and genetic algorithms [10]. In all of the approaches that use metaheuristics, the metaheuristics create a sequence or sequences of jobs. This sequence is then used by a schedule builder, incorporating problem specific algorithms, that completes the remainder of the schedule. These approaches differ primarily in the algorithms used by the schedule builders.

Charalambous et al. [10] split the demand for each product into ‘sublots’, and each sublot is scheduled individually. A sophisticated genetic algorithm manipulates a single sequence of these sublots. In their later paper [9], a sophisticated variant of simulated annealing is used instead. The algorithm that schedules each sublot starts at the final processing stage of the plant, calculating and ranking candidate time intervals for the scheduling of the sublot in that stage. The best is chosen before considering the previous processing stage in the same way. If at any point the sublot cannot be scheduled in a section of the plant, backtracking
is used to select alternative candidate time intervals in the later stages of the plant. During the scheduling of a sublot on a processing unit, only continuous time intervals are considered. It will be seen that this explains the necessity of splitting the demand into sublots.

Reynolds et al. [4, 5] do not split the demand for each product in this way. The total amount of product assigned to a final stage unit is considered to be a job. Simulated annealing manipulates job sequences for each final stage unit and may also control the allocation of product to these units. A full schedule for the plant is created from these sequences by running a simulation of the plant, using problem specific heuristics to determine the sequence of operations on first stage units and the use of storage units. During this simulation, units are permitted to run at less than full speed if units are delayed by either lack of required base product or lack of storage. However, postprocessing code can be applied to correct this without changing the schedule makespan. This results in jobs being split by idle periods in the resulting schedule.

Neither of the approaches described above can guarantee that a task on final stage unit is processed without delay or interruption. This paper describes an approach that can provide this guarantee. Furthermore, this new approach will be shown to have other advantages over the approaches in the literature.

4 Approach to schedule construction

4.1 Overview

In each of the approaches described in literature, a schedule builder is provided with a list or lists of jobs. Here, the schedule builder is given one list of jobs that is produced by the metaheuristic. (In practice, the metaheuristic manipulates a list of jobs for each final stage unit. However, the use of a job prioritization scheme converts these lists into just one list for the plant.) As far as the schedule builder is concerned, each job can be processed by only one final stage unit, although there may be a choice of units for earlier stages.

Note that the term ‘job’ will refer to the processing of product at all stages of production. We will use the term ‘task’ to refer to the work required at a given stage of the plant.

The schedule builder schedules one job at a time, although it will be seen that some changes to previously scheduled jobs may be made. When scheduling a task on a final stage unit, the task must be placed after those that have already been scheduled. It will be shown that this facilitates the easy handling of product sequence constraints, when present. Tasks in the earlier stages of the plant may make full use of any gaps that are present in the partially constructed schedule. This contrasts with the approach of Reynolds et al. [4, 5], where the plant simulation progresses strictly forward in time, eliminating the possibility of utilizing such gaps.

The processing of the job should be completed as early as possible within the schedule. Ideally, the final stage task is scheduled immediately after the previous
task on the unit, taking into account the required setup times. However, since it may be impossible to supply sufficient product to the final stage unit quickly enough, delays may be necessary. Remember that once a task is started by a final stage unit it must continue without delay until completion. Therefore the start of the task is delayed to the earliest point where it is known that sufficient product will be fed to the unit.

The top level of the job scheduling algorithm is shown in figure 2. Detailed descriptions of the major parts of the algorithm are given in the sections listed in the comments.

Let $M$ be the set of first stage units capable of producing base for job $j$; Let delay $d := 0$; 
for each $m \in M$ 
  Perform a reverse simulation to obtain profile $p(m, d)$; 
endfor 
while all profiles are invalid 
  Find the minimum amount of delay, $new_d \geq d$, that results in a major change to a profile and let $m'$ be the first stage unit whose profile changes; 
  $d := new_d$; 
  Recalculate the new profile $p(m', d)$; 
endwhile 
Create a job schedule from one of the valid profiles; 

Fig. 2. Scheduling one job: top-level pseudo-code.

4.2 Producing base product for multiple jobs

In a plant that does not produce continuous product, tasks in later stages of the plant must wait for the completion of tasks in the earlier stages. As such, it makes little sense for a task in the first stage of the plant to make base product for more than one job at a time. A sequence of tasks, producing base product for each of the jobs, would take the same time and would release much of the base product earlier.

If a plant produces continuous product, it may often be useful to have a first stage unit produce product for more than one job simultaneously. For example, consider the plant shown in figure 3. Here the plant must produce 50 tonnes of two different products from one base product. Production rates and job sizes are shown. Note that if both final stage units are running, they consume product as quickly as it can be produced. Assuming that the storage unit can feed both units simultaneously, the plant can complete its tasks in five hours, but only
if the first stage unit produces base product for both jobs simultaneously. If it produces base product for one job at a time, seven and a half hours are required, since manufacture of product destined for the second job does not start until two and a half hours into the schedule.

During schedule construction, the possibility that a first stage job might usefully produce base product for two or more jobs at a time is taken into consideration. Further details are given in section 5.3.

5 Reverse simulation

Determining whether a job can be scheduled without any delay, using a predetermined first stage unit to create the base product, involves using knowledge of the availability of the first stage unit to calculate the minimum storage requirements at each moment in time. If sufficient storage space can be found and if enough base product can be produced, then the job can be scheduled without delays.

The algorithm that determines whether the job can be scheduled in this way uses a simulation that proceeds backwards from the end of the job.

- At the point in the schedule where the job finishes (point $E$), all the base product associated with the job should have been used, and hence no storage space is required.
- Consider a point, $P$, slightly earlier in the schedule.
- The amount of base product converted into completed product between this point and the end of the job is easily calculated.
- The availability of the first stage unit is known. The amount of base product that can be produced in this period is easily calculated.
- If the amount of base product that can be made by the first stage unit equals or exceeds that required, there need not be any product in storage at point $P$. However, a storage unit may be required to pass product between the
stages. Between points P and E the first stage unit is assumed (for the time being) to process product at the same rate as the final stage unit, so that the amount of product in storage is zero between the two points.

- If the amount of base product that can be made in this period is less than that required, this amount must be supplemented by quantities in storage. During this period the storage unit slowly empties, since the final stage unit consumes base faster than it is produced.

By continuing to consider points earlier in the schedule, it is possible to calculate the minimum storage requirements at each moment in the schedule. If the storage space is available, the job can be scheduled. However, if there is insufficient storage, or if the simulation proceeds beyond the start of the schedule, the job must be delayed.

Notice that calculating the minimum storage requirements by performing a simulation forward in time is difficult. If such a simulation were used, it would be difficult to determine how long to delay the start of the first stage tasks. Waiting too long would cause the final stage task to be interrupted due to lack of base product, whereas starting too soon would lead to extra storage requirements.

5.1 Continuous production of base product

The backward simulation works on the assumption that processing rates are constant between events. The simulation repeatedly performs the following tasks:

- Find the next event in the simulation. As the simulation works backwards in time, this event occurs earlier in the schedule.
- Calculate the amount of base produced and consumed at the time of the event.
- Determine whether a violation of constraints occurs at the event, due either to insufficient storage being available or the simulation proceeding past the start of the schedule.
- Change the processing rates of the units, depending on the type of the event.

The result of the simulation is a profile of the amounts of base produced and consumed during the lifetime of the job. This profile provides a provisional schedule for the job that minimizes storage requirements by postponing the production of base product for as long as possible without delaying the final stage task.

This is illustrated with an example. Suppose that, before the reverse simulation is performed, the following facts are known:

- The selected first stage unit produces base product at 40T/hr.
- The final stage unit consumes base product at 20T/hr.
- The job requires 130T of base product.
- Storage units each have a capacity of 30T.
- Some higher priority jobs have already been scheduled, resulting in the selected first stage unit being unavailable for the production of base for the new job between 2.5hrs and 5.5hrs, and also between 8hrs and 9hrs.
The final stage unit is busy with other jobs or setup operations until 3.5hrs from the start of the schedule.

If the final stage task is scheduled without delay, it will end 10hrs after the start of the schedule. The reverse simulation will indicate whether it is possible to provide the base product required to do this.

The results of the simulation are shown in the profile of figure 4. This example shows many of the events that can occur during such a simulation. The arrows show the direction in which the profile is constructed by the reverse simulation. Since the desired timing of the final stage task is known in advance, the graph of the amount of base product consumed can be placed as shown in figure 4. The simulation then calculates the amount of base produced from the start of the job to each point in time on the graph. Storage requirements (base produced minus base consumed) is minimized, but must always be non-negative. This results in the arrowed graph in figure 4. The events that occur during the construction of this simulation profile are as follows.

**Job end** The simulation starts from the end of the job. At this point in time the first stage unit is available. As the final stage task nears completion, the first stage unit produces the required base. This base product is immediately consumed by the final stage unit, thus minimizing the need for storage. However, the first stage unit cannot produce base at full speed as this would
result in the graph of the amount of base produced falling below that of the amount consumed. Therefore the first stage unit runs at only 20T/hr.

**First stage unit unavailable** Once the simulation reaches the point 9hrs into the schedule, the first stage unit becomes unavailable. Since no base product is produced between 8hrs and 9hrs, product must be present in storage at the point 8hrs into the schedule.

**First stage unit available** Once the simulation reaches the point 8hrs into the schedule, the first stage unit becomes available again. Before this point in the schedule, the first stage unit runs at full speed in order to fill storage with base product to be used between 8hrs and 9hrs into the schedule.

**Slow first stage unit** Once the simulation reaches the point 7hrs into the schedule, the amount of base produced equals the amount consumed. Since the amount of base consumed must be no greater than that produced, the first stage unit must run no faster than the final stage unit.

**Storage unit required** After the simulation reaches the 5.5hrs mark, the first stage unit becomes unavailable again. Once the 4hrs mark is reached, the amount of base produced must exceed the amount consumed by at least 30T. Therefore an extra storage unit is required to store the base product.

**Job start** Processing on the final stage unit is scheduled to start at the 3.5hrs mark. The amount of base product consumed by the job at this point must be zero.

**Silo unnecessary** After the mixer becomes available, the amount of base product consumed decreases as the simulation continues to move backwards through the schedule. This leads to the amount of storage required falling below the 30T mark, making the extra storage unit unnecessary.

**Finished** Finally, when the amount of base product consumed also reaches zero, the start of the first stage task has been reached. This is also the end of the simulation.

Notice that figure 4 provides a schedule for the production of base product that minimizes the requirement for storage by delaying the production of base product for as long as possible, while still meeting the base requirements of the job. This schedule is provisional only, since it may contain periods when units process product at less than full speed. Furthermore, it may be possible to improve on the schedule to allow greater flexibility when scheduling subsequent jobs. Section 7 discusses how improved schedules are produced from simulation profiles.

(Turning figure 4 upside down gives an alternative interpretation of the profile. If the final stage unit feeds base product into the storage units, which is then fed to the first stage unit, figure 4, when turned upside-down, gives a schedule where the first stage unit always runs as fast as possible.)

### 5.2 Batch production of base product

An example of a simulation profile where the first stage unit produces product in batches is shown in figure 5. Such simulation profiles include two new types
of event — *start batch dump* and *end batch dump*. These occur whenever a batch of product is placed into storage. Two events are required for each batch dump since the storage requirements change between the events, potentially requiring the insertion of a ‘storage unit unnecessary’ event.

The speed and availability of the units and the size and positioning of the final stage task are the same as in the previous example. However, in this example the first stage unit produces base product in batches of 15T every 0.375hrs. Unit availability is indicated beneath the profile, along with the provisional placement of the batches. Notice the ‘first stage unit unavailable’ events occur 0.375hrs later than the actual points where the first stage unit becomes unavailable (and hence earlier in the simulation). These events merely indicate that batches may not be placed in storage, since there is insufficient time beforehand to process the batch. For clarity, events associated with the placement of batches into storage are not labeled.

### 5.3 Using one unit to produce base for more than one job at a time

Jobs are presented to the schedule construction algorithm one at a time. However, it has been shown that it may be useful to use one first stage unit to
produce the base product for two or more jobs simultaneously. When this is useful, the construction algorithm attempts to combine the new job with one that is already scheduled. No additional delay to the scheduled job is permitted. However, the timing of the tasks of this job on all but the final stage of the plant will change if the attempt to schedule the jobs together is successful.

In order to be able to do this, the backward simulation must be able to handle the processing of more than one job. When performing this parallel scheduling, two further event types will occur in the simulation profiles — the start and the end of these fixed tasks.

5.4 Storage unit selection and profile validity

At the start of the simulation, and whenever the simulation hits a ‘storage unit required’ event, an attempt is made to find a suitable storage unit to use. If no such unit can be found, a ‘storage violation’ is recorded at the event. Once such a violation is recorded, no further ‘storage unit required’ events may occur until after the next ‘storage unit unnecessary’ event. The ‘storage unit unnecessary’ event indicates that, once again, the space available in the selected storage units is sufficient to store the base product.

If more than one storage unit can be found, the unit that remains available for longest, as the simulation proceeds backwards through the scheduling window, is selected. This increases the chance that the storage unit will still be available at the ‘storage unit unnecessary’ event. If the storage unit becomes unavailable before then a storage violation is recorded.

The presence of a violation indicates that the job cannot be scheduled at the selected time. The violation may be either a storage violation or a time violation. A time violation occurs whenever the simulation continues beyond the start of the schedule. This indicates that there is insufficient time from the start of the schedule to produce the required base product.

6 Delaying jobs

If the backward simulation reveals that the final stage task cannot be scheduled in the earliest free period on the unit, the job must be delayed. It is desirable to minimize this delay. By increasing the delay continuously and modelling the changes produced in the simulation profile it is possible to find the smallest delay required.

As the delay increases, the events move smoothly within the simulation profile. The time of each event changes. Both the amount of base produced and the amount consumed at the events also change. Section 6.1 describes how the rate of change of these quantities can be calculated at an event.

When the amount of delay reaches certain points more significant changes occur within the simulation profile. The sequence of events may change, or violations may need to be added or removed, as described in section 6.2.
6.1 Rates of change, at an event

In order to be able to determine when a major change to the simulation profile will occur — such as changes to the event sequence, or the removal or addition of violations — it is necessary to know how the time of each event varies as the delay to the job increases. It is also necessary to know how both the amount of base produced and the amount consumed varies at an event as the delay increases.

These rates of change depend on both the event type and on the circumstances in which the event occurs. The number of different combinations of event type and circumstance is large and the three rates of change must be calculated for each combination. The following example describes the calculation of the rates of change for just one event, in one circumstance.

Consider the task of determining the three rates of change at a ‘storage unit required’ event. The event in question occurs at a point where the first stage unit is unavailable and the final stage unit is consuming base product. Only one job is being scheduled. The bold graphs of figure 6 show the simulation profile. The other graphs in figure 6 show how the simulation profile changes if the delay to the job is increased by $\delta t$. It is seen that both the time of the ‘storage unit required’ event and the amounts of base produced and consumed at the event change.

To calculate the rates of change of these quantities at the event, first define the following:

**Fig. 6.** The motion of a ‘storage unit required’ event.
\( T = \) event time;
\( C = \) base consumed;
\( P = \) base produced;
\( c = \) rate at which final stage unit consumes base, when running;
\( p = \) rate at which first stage unit produces base, when running;
\( t = \) final stage task start time.

Calculate the rates of change of \( C \) and \( P \) at a time \( T \), keeping \( T \) fixed but increasing the start time \( t \) of the final stage task.

\[
\frac{\partial C}{\partial t} = -c, \quad \frac{\partial P}{\partial t} = -p.
\]

Next calculate the rates of change of \( C \) and \( P \) as \( T \) increases, keeping \( t \) fixed.

\[
\frac{\partial C}{\partial T} = c, \quad \frac{\partial P}{\partial T} = 0.
\]

Let \( S \) be the amount of storage required at the event, i.e. \( S = P - C \). Then

\[
\frac{\partial S}{\partial t} = c - p, \quad \frac{\partial S}{\partial T} = -c.
\]

Now the amount of storage required at a ‘storage unit required’ event must remain fixed. Therefore

\[
\frac{dS}{dt} = 0,
\]

i.e.

\[
\frac{\partial S}{\partial t} + \frac{\partial S}{\partial T} \frac{dT}{dt} = 0,
\]

\[
(c - p) - c \frac{dT}{dt} = 0,
\]

\[
\frac{dT}{dt} = \frac{c - p}{c}.
\]

This result is now used to calculate the rates of change of \( C \) and \( P \) at the ‘storage unit required’ event.

\[
\frac{dC}{dt} = \frac{\partial C}{\partial t} + \frac{\partial C}{\partial T} \frac{dT}{dt} = -c + c \left( \frac{c - p}{c} \right) = -p,
\]

\[
\frac{dP}{dt} = \frac{\partial P}{\partial t} + \frac{\partial P}{\partial T} \frac{dT}{dt} = -p.
\]

This approach may be used to calculate the three rates of change in any circumstance and for any event.
6.2 Event sequence and violation changes

When the delay to the job reaches certain points, more fundamental changes occur to the simulation profile. These changes may involve the reordering of the events in the profile, the addition or removal of events or the addition or removal of violations. Each such major change is handled by recalculating the simulation profile by repeating the reverse simulation, but with the new, increased delay. Note that efficiency improvements might be made by making only those modifications to the profile that are necessary, rather than recalculating from scratch.

**Event collisions** As the delay to the job increases, the time of one event in the simulation profile may become equal to the time of another event. The result may merely be that the two jobs swap positions in the profile, possibly requiring the recalculation of the rates of change at these events. Alternatively, the two events may disappear from the simulation profile or more complex changes may occur.

**Storage unit requirement changes** As the job’s delay increases, the storage requirements at an event in the profile may require an additional storage unit to be available, or alternatively, one fewer storage unit might become necessary. Figure 7 illustrates how it may be necessary to create new storage events in the profile.

![Fig. 7. The creation of a pair of storage unit requirement events.](image)

**Storage unit availability changes** As the ‘job end’ event and ‘storage unit required’ events move as the final stage task is delayed, a storage unit that was available at the event may become unavailable. A storage unit may not have been available at the event, but one may become available as the event
moves. Also, a storage unit that is a better choice, being available further into the past, may become available. All of these changes may require the removal or addition of other events in the simulation profile.

**Other violation changes** As the job is delayed, ‘storage unit unnecessary’ events may move to a point where the storage unit selected at the corresponding ‘storage unit required’ event is still available, allowing the removal of a storage violation. Alternatively, such an event may move to a point where the storage unit becomes unavailable, requiring the addition of a storage violation. If production of base within the simulation profile starts before time zero, delaying the final stage task will eventually allow production to start at a non-negative time, within the scheduling window. When this occurs, the time violation is removed.

**First stage unit must be slowed** As the delay to the job increases, it may become necessary to slow the first stage unit at some point in the simulation profile. This usually occurs when a ‘slow first stage unit’ event collides with some other event. However, if the first stage unit is to produce base for more than one final stage task simultaneously, it may become necessary to slow the first stage unit without any such collision of events. This situation is shown in figure 8.

![Fig. 8. The addition of a ‘slow first stage unit’ event.](image)

**Batch units** If the first stage unit is a batch processor, other conditions may occur as the job is delayed, requiring changes to the simulation profile. For example, the production of batches may occur with gaps between them in the original profile. However, as the final stage task is delayed, such gaps may disappear. The collision of these batches alters the rates of change at events in the profile. Alternatively, a gap may appear that is large enough to fit in another batch, requiring changes to the simulation profile.
6.3 Keeping the delay to a minimum

In order to schedule the job with minimum delay, first it is assumed that the final stage task will be placed with no delays. A simulation profile is created for each first stage unit that may create the base product. If one of these is a valid profile, the job is scheduled without any delays. However, if violations occur in each of the simulation profiles, the delay to the final stage task is increased until one of the simulation profiles requires a major change. Then the new simulation profile is calculated, with the amount of delay to the job being just large enough to require this change to the profile. If the new profile is valid, the final stage task can be placed. Otherwise the delay to the final stage task is increased again, until another major change is required to one of the simulation profiles. By proceeding in this way, the job is scheduled with minimum delay.

7 Extracting a job schedule from a simulation profile

7.1 Continuous production of base product

Having found a valid simulation profile for the packing job, the profile provides a schedule of the production, storage and utilization of the required base product. However, there are a number of reasons for not using this schedule:

- The schedule may include periods where the first stage unit runs at less than full speed. This is not permitted in the case studies considered here.
- It makes sense to minimize the number of storage units used at any moment in time, to keep storage space free to be used by other jobs. However, since each storage unit may hold only one type of base product at a time, allocated storage units should be used in such a way as to improve the scheduling of the first stage unit.
- Ideally, the schedule for the first stage unit should not contain small idle periods that may not be used by jobs added later in the schedule construction process.
- The first stage unit should, ideally, become free as soon as possible, so that it can be used for the production of other base products.

Consider an example where the first stage unit produces base for two final stage tasks simultaneously. One consumes base product at a rate of 20T/hr, starts 5 hours into the schedule and finishes 3 hours later. The other consumes base product at 10T/hr, starts 2 hours into the schedule and finishes 4 hours later. The selected first stage unit produces base at 25T/hr, but is unavailable from a point 4 hours into the schedule for 3 hours, since it produces a different product in this period. The selected storage unit has a capacity of 60T. The left hand graph of figure 9 shows the resulting simulation profile.

Figure 9 also shows how the profile changes when the first stage unit is made to run as fast as possible from the start of the profile. The result is a schedule for the first stage unit where the only gaps are those necessary for the setup operations. The first stage unit also finishes the job as early as possible. The
amount of storage space used has increased, but the number of storage units
used has not.

Now suppose that the selected storage unit has a capacity of $22\frac{1}{4}T$. In this
case, making the first stage unit produce base as quickly as possible from the
start of the profile results in the storage unit becoming full. This delays the first
stage unit, resulting in a period of time where the unit runs at less than full
speed. Figure 10 shows how this schedule is fixed. The period where the first
stage unit is slowed is replaced by a period where it is idle followed by a period
where it runs at full speed. Longer slowed periods may be replaced by more than
just one full speed section and one idle section.

Notice that the schedule for the first stage unit can be tidied, by removing the
small processing period and replacing it at the end of the schedule. This delays
that time at which the first stage unit becomes free, but the tidier schedule may
compensate for this. However, for simplicity, no such delays are considered.

### 7.2 Batch production of base product

When the first stage unit produces base product in batches, it cannot run at less
than full speed. Therefore, the simulation profiles created may be transferred
directly to the schedule. The resulting schedule for the job delays production of
base for as long as possible, in order to minimize storage requirements. However,
this process may lead to a large number of small gaps, as shown in figure 11.
Since such small gaps are unlikely to be used later in the schedule construction
process, this situation is far from ideal.

By rescheduling the first stage unit to produce base product as early in the
schedule as possible, the simulation profile can be changed to that shown in
Fig. 10. Fixing sections where the first stage unit runs slow.

Fig. 11. First stage batch unit schedule with many gaps.
Here the storage unit selected has a capacity of only 40T. This does not remove all of the gaps, but the first stage unit finishes producing base as early as possible, allowing the next task on this unit to start earlier.

Figure 12 shows two sets of consecutive batches that are delayed through lack of storage: batches 4–9 and batches 11-13. (Batch 10 is delayed by batch 9.) Each set of delayed batches is tidied using much the same process as if the first stage unit processed product continuously and was delayed through lack of storage. The delayed section is converted into a sequence of ‘stopped’ and ‘full speed’ sections, through adjusting the time at which each batch is placed in storage, or ‘dumped’.

Define:

- $G$: a set of consecutive batches delayed by lack of storage,
- $batchSize$: amount of base product in one batch,
- $batchTime$: time required to create one batch,
- $endTime$: time at which the final batch in $G$ is dumped,
- $consumed(t)$: amount of base product consumed at time $t$,
- $space(t)$: amount of allocated storage space available at time $t$.

The process of tidying the production of the batches in $G$ is described in the pseudo-code in figure 13. The result, in the example case, is shown in figure 14. Here the first stage unit finishes production as early as possible and the gaps...
\[ p := \text{the amount of base produced before } G; \]
\[ b := \text{the first batch in } G; \]
\[ s := |G| - 1; \quad // \text{Number of subsequent batches in } G \]

\begin{algorithmic}
  \While {\text{\textit{b}} \in G}
    \State // Delay batch dump for as long as possible
    \State \textit{t} \leftarrow \max \{ \text{\textit{t}} : \text{consumed} (\textit{t}) = \textit{p} \}; \quad // \text{Time at which silo becomes empty}
    \State \textit{t} \leftarrow \text{endTime} - \textit{s} \times \text{batchTime}; \quad // \text{To prevent further delay to last batch of } G
    \State \textit{t} \leftarrow \min (\textit{t}_1, \textit{t}_2);
    \EndWhile
\end{algorithmic}

// Go until a batch dump would result in overfull storage
\begin{algorithmic}
  \While {\text{\textit{b}} \in G \and \text{space} (\textit{t}) \geq \textit{p} + \text{batchSize} - \text{consumed} (\textit{t})}
    \State Dump batch \textit{b} at time \textit{t};
    \State \textit{p} \leftarrow \textit{p} + \text{batchSize}, \textit{t} \leftarrow \textit{t} + \text{batchTime};
    \State \textit{b} := \text{next batch}, \textit{s} := \textit{s} - 1;
  \EndWhile
\end{algorithmic}

\begin{figure}[h]
  \centering
  \includegraphics[width=\textwidth]{fig13}
  \caption{Tidying batch production.}
  \end{figure}

that occur, between the production of batches, are large enough to be of use later during the construction of the schedule.

8 Using simulated annealing to produce the job sequence

Using reverse simulation to schedule jobs as described above, minimizing job delay, results in a greedy algorithm for converting a permutation of jobs into a complete schedule. In order to produce a practical schedule optimizer, a method is required for optimizing this permutation of jobs. Metaheuristics, such as simulated annealing or genetic algorithms are ideally suited to this task.

Although the greedy schedule construction algorithm requires just one sequence of all the jobs that need to be scheduled, this is not the ideal solution representation for use with a metaheuristic. The potential presence of sequencing constraints on final stage units suggests a solution representation based on task sequences for each final stage unit. A single job sequence is produced from these sequences later, using a job prioritization scheme.

8.1 Solution representation

There are two types of sequencing constraints considered for final stage units:

**Enforced grouping by product attribute:** For example, all white product may need to be processed as a group of tasks, uninterrupted by tasks processing products of other colours.

**Enforced ordering by product attribute:** For example, it may be essential that no black product is processed on a unit until all the white products have been processed.
A third type of sequencing constraint may be produced by combining the two
types described above.

When a final stage unit is subject to grouping constraints, the simulated
annealing manipulates a tree of tasks for this unit. Such a tree is shown in figure
15. Here products P3, P4 and P5 must be grouped together in the production
sequence. Within this grouping, P4 and P5 must also be grouped together.

The arrangement shown also represents the product sequence P1, P2, P3,
P4, P5, P6. The simulated annealing neighbourhood operator may rearrange a
set of sibling nodes, using either a swap or a shift operation. For example, child
nodes 1 and 3 may be swapped, moving all child nodes accordingly, to produce
the tree shown in figure 16. The resulting job sequence is P3, P4, P5, P2, P1,
P6.

Ordering constraints are handled by simply preventing moves that lead to
such constraints being broken. Each node in the task tree has a list of sibling
nodes that must come before and a list of those that must come after. The
task tree representation makes it a simple matter to detect conflicting sequence
constraints and implicit sequence constraints. For example, if the task tree is as
shown in figure 15 and the case study description states that product P2 must
be processed before product P4, then product P2 must also be processed before
P3 and P5.

The task tree representation approach described is superior to using a simple
permutation and either forbidding or penalizing moves that break constraints.
The latter approach leads to a search space where the set of feasible solutions is
Fig. 15. Task tree data-structure

Fig. 16. Tree after a swap neighbourhood move.
disconnected. Two solutions could be chosen such that, to get from one to the
other, it would be necessary to pass through solutions that break the sequencing
constraints.

When a final stage task may be assigned to more than one unit, it is also
necessary to include a representation of the allocation of tasks to units. However,
in the case studies presented here, each final stage task may be assigned to only
one final stage unit. Examples of problems where task allocation is required may
be found in [11].

8.2 Job prioritization

A task sequence for each final stage unit can be extracted from the solution repre-
sentation manipulated by the SA. However, the schedule construction algorithm
requires just one list of final stage tasks. To select the next job to be scheduled,
the highest priority unscheduled task is chosen from the task sequences for the
individual machines. The task priority is simply the minimum time required by
the final stage unit to complete the task and all subsequent tasks. This includes
all setup operations and maintenance periods.

9 Results

Two real world case studies illustrate the advantage of the schedule construction
approach described here over those in the literature. The first is the case study
used in the papers of both Reynolds et al. [4, 5] and Charalambous et al. [10,
9]. The three stage plant consists of two first stage processing units, five storage
units and five final stage processing units. All processing units process product
continuously, with the exception of one first stage unit that manufactures base
product in batches. Each storage unit may feed product to only one final stage
unit at a time, and receive product from only one first stage unit at a time.
Furthermore, each final stage unit may be fed by only one storage unit at a
time. Optimal schedules for this case study have a makespan of 113.75 hours.
Schedules must be completed within 120 hours.

The second case study involves a similar plant but with only four final stage
units. All processing units process product continuously. A notable feature of
this case study is the presence of a very large job, with a final stage task that
requires 101.25 hours of processing time. A lower bound on the optimal makespan
is 106.39 hours. Production must be completed within 120 hours.

More detail of these and other case studies may be found in [11], although
for full details, the reader should contact Unilever plc.

All experiments consisted of forty runs of the simulated annealing, allowing
160 CPU seconds per run (6400 CPU seconds per experiment) on a DEC Al-
phastation 500 with a 400 MHz processor. Experiments were performed with
a wide range of initial starting temperatures and a number of different cooling
schedules.
9.1 Case study one

Experiments on the first case study were performed with starting temperatures ranging from 1 to 20. The temperature was multiplied by 0.9985 every 20 iterations. This resulted in a final temperature approximately one tenth of the starting temperature. Task sequences on final stage units were changed using the swap neighbourhood operation only.

Figure 17 shows the mean result of the forty runs, for each starting temperature.

All runs produced a schedule with a makespan of either 113.75 hours (the optimum) or 114.97 hours. The critical final stage unit in this plant produces two different product types: dilute and concentrated. A large setup operation (36 hours) is required when changing between product types. Only 1 hour is required when changing between two products of the same type. In the best solutions, the critical unit produces all products of one type and then produces all the products of the other type, to prevent the occurrence of more than one large setup period.

Dilute product is produced in the first stage unit that operates continuously. Hence if the critical unit starts with dilute products, it need not wait to start processing, as product can be routed to it immediately. Concentrated products are produced in the first stage unit that produces batches. If the critical unit starts with concentrated products, it must wait for 1.22 hours for the first batch to be completed. This is precisely the difference between the two makespans.

If the critical unit processes concentrated product first in the current solution, and the current solution has a makespan of 114.97 hours, the only way in which the product ordering can be changed to that of an optimal solution is by first...
introducing a second large setup operation. Hence the simulated annealing must accept a schedule with a makespan of at least 148.75 hours. This does not happen if the temperature is low.

There is an alternative to using a high temperature to enable this change in the schedule to occur. In order to remove the possibility that two large setup operations will occur on the critical unit, sequence constraints can be added. These enforce the grouping of dilute products together in the production sequence, and the grouping of the concentrated products. The neighbourhood operators described in section 8.1 enable the ordering of the groups of product to be changed, switching the group of dilute products with the group of concentrated products. This operation means that the simulated annealing can move from the solutions with a makespan of 114.97 hours to the optimal solutions without the introduction of extra long setup operations on the critical unit. Results with an initial temperature of 20 are much the same as when this constraint is not added. However, optimal solutions also occur in every run when the initial temperature is only 1.

The mean time taken for a run to find either an optimal solution or a solution with makespan of 114.97 hours was 1.5 seconds.

A comparison with results in the literature Charalambous and Hindi [10] applied a genetic algorithm to this case study, and were able to obtain optimal solutions. However, their run lengths were of 6 hours on a Pentium 150 (roughly equivalent to 1 hour 50 minutes on the AlphaStation 400), rather than the 160 CPU second runs in the experiments reported here. Furthermore, in their initial experiments, no constraints on the number of connections that each unit has open at a time were imposed. When each unit is permitted to feed only one unit at a time, and be fed by one unit at a time, the quality of the schedules produced drops. The best schedule produced in these circumstances is reported as having a makespan of 118.95 hours.

Constraints on the number of connections a unit may have open at once cause no difficulties to the algorithm described here. All the results reported already take into account the presence of such constraints.

Charalambous and Hindi also considered a second way to make this case study more of a challenge. Halving the storage capacity of each of the storage units results in their algorithm failing to produce an optimal schedule. However, the performance of the algorithm of this paper is not greatly affected by this reduction in available storage. The only noticeable change in the quality of the results is that the average time taken to find a schedule with a makespan of either 113.75 or 114.97 increases from 1.5 seconds to 3.5 seconds. Indeed, it is possible to further reduce storage capacity, by removing one of the storage units, and still obtain these schedules in an average time of 5.4 seconds.

In a later paper, Charalambous et al. [9] apply simulated annealing to the Pozzilli case study. Less detail of the time requirement is provided, although the authors state that “even for complex systems, it does not exceed an hour on a high performance PC”. Using a standard SA did not produce optimal solutions.
However, their modifications to the SA algorithm allowed optimal solutions to be obtained, provided the constraints on simultaneous connectivity are ignored. When these constraints are added, or storage availability is reduced, optimal solutions cannot be found.

In both of the papers by Charalambous et al. [10, 9], demand for individual products are broken into sublots. This means that the constraint that final stage tasks, once started, process at full speed until completion is unlikely to be met. Furthermore, implementation of task sequencing constraints on final stage units would be difficult.

The simulation based approach of Reynolds et al. ([4, 5] and improved further in [11]) is able to produce schedules of the same quality as those produced by the algorithm of this paper, and requires a similar amount of time. However, the simulation based approach gives no guarantee that final stage tasks, once started, proceed without interruption until completion. The second case study will more clearly demonstrate the superiority of the construction based approach of this paper.

9.2 Case study two

Experiments on the second case study were performed with starting temperatures ranging from 0.1 to 10. The temperature was multiplied by 0.99 every 20 iterations, again resulting in a final temperature approximately one tenth of the starting temperature. Task sequences on final stage units were changed using the swap neighbourhood operation only.

Figure 18 shows the mean result of the forty runs, for each starting temperature.

![Graph](image-url)

**Fig. 18.** Results for case study two.
On average, the best results are obtained with a starting temperature of 2. The mean result obtained with this starting temperature was 110.56 hours, with the best being 108.70 hours. (A schedule with a makespan of 108.40 hours was obtained in one of the runs with a starting temperature of 0.9.)

Performing the same number of experiments, and using similar settings, the simulation approach of Reynolds et al. produced the results outlined in figure 19. It can be seen that the construction based algorithm of this paper significantly outperforms the simulation based approach of Reynolds et al. The results obtained by using the simulation based approach can be improved considerably by splitting the large final stage task into a number of sub-tasks, in much the same way as Charalambous et al. do. However, even splitting this task into nine equal sized pieces results in a mean makespan of 111.25 hours. Also, splitting the task in this way makes it highly likely that, once started, the task will be interrupted by other tasks or idle periods.

9.3 Additional case studies

The two case studies described above were chosen to illustrate the advantages of the approach over those taken by Charalambous and Hindi [10, 8] and Reynolds et al. [4, 5]. The algorithm has also been applied to real world case studies with additional features such as maintenance periods and task sequence constraints on the final stage units. These case studies proved to be less challenging than both case study one and two. Setup times on the critical units were independent of the products being produced and the case studies typically had one unit that had much more work to do than the others. Optimal solutions were obtained in no more than a few seconds.
Extending the approach to deal with plants with more stages may be more of a challenge and is a matter for further research.

10 Conclusions

In the problem instances we have studied, it is typically the case that first stage units run faster than final stage units. When a first stage task starts, storage requirements increase until the task is stopped or paused. If neither the final stage task nor the first stage task associated with a job are broken up into smaller tasks, these storage requirements can become very large. In the second case study, scheduling the large job in this way would require a minimum of 271 tonnes of storage space. The plant as a whole only has 150 tonnes of storage space.

The approach of Charalambous and Hindi is to schedule jobs in unbroken time periods on both first stage and final stage units. As a result, the jobs must be broken into ‘sublots’, to prevent the blocking of units within the plant. Using sublots in this way introduces new problems:

– If sublots are processed on a machine in a random order, a large number of setup operations are required. Therefore, most solutions in the search space will be of poor quality.
– Even if sublots are sequenced on final stage units in such a way as to prevent one task (consisting of a set of sublots) being interrupted by another, it is difficult to ensure that, once a task is started, it is not interrupted by idle periods. Schedules where final stage units are repeatedly stopped and restarted are difficult to implement on the factory floor.
– If sublots are small, the search space will be large. However, using larger sublots increases the likelihood that excessive demand will be placed on the storage section of the plant.

The simulation approach of Reynolds et al. does not use sublots. Instead, first stage units may be slowed during the simulation of the plant. Postprocessing is used to convert each slowed period into a sequence of periods where the unit either operates at full speed or is idle. This produces better results on the first case study. However, the second case study illustrates a problem with this technique. Although, by slowing the first stage unit, pressure on the storage section of the plant can be reduced, the result is inefficient use of the first stage unit. For example, a first stage unit manufactures base product for the large job of the second case study and has access to one storage unit. It produces product at full speed for 3.70 hours until the storage unit becomes full. Then is produces product at a reduced rate for 90.05 hours, making a total of 93.75 hours. At full speed and with sufficient storage available, the unit could produce this base product in 33.47 hours. More than 60 hours of machine time has been wasted. If the job had access to all the storage units in the plant, the result would still be over 30 hours of wasted machine time, plus the fact that the other jobs in the plant would have to be halted due to lack of storage.
The approach described in this paper allows a first stage unit to produce the base product for this job intermittently, producing gaps in the schedule that are used to produce different base products. In this way, pressure on the storage section of the plant can be minimized without resulting in inefficient use of first stage units.

The result is an algorithm that produces better results than those achieved by Charalambous et al. [10, 9] and those obtained using the simulation approach of Reynolds et al. [4, 5]. Furthermore, the algorithm guarantees that final stage tasks, once started, are neither interrupted nor delayed in any way until completion. The same cannot be said of the other approaches in the literature.

Note that the simulated annealing algorithm used to obtain these results was little more than the standard algorithm, whereas Charalambous et al. used more sophisticated variants of genetic algorithms and simulated annealing. This paper illustrates that, when using metaheuristics to solve this problem, the algorithm used to convert job sequences into complete schedules is of much greater importance than the details of the metaheuristic used.

References
