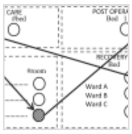
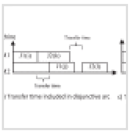
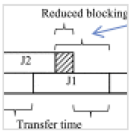
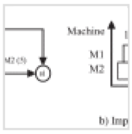
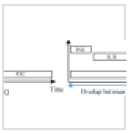
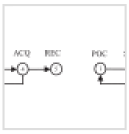
Outline

Show full outline

Figures (9)

Show all figures

Tables (17)

Show all tables

European Journal of Operational Research
Available online 30 June 2017
In Press, Corrected Proof

Innovative Applications of O.R.

An integrated approach for scheduling health care activities in a hospital

Robert L. Burdett, Erhan Kozan

Show more

https://doi.org/10.1016/j.ejor.2017.06.051

Get rights and content

Highlights

- A scheduling approach has been developed to optimise the usage of hospital treatment spaces.
- An entire hospital is describable and schedulable in one integrated approach.
- Activities in patient's clinical pathway are assigned and sequenced on different resources.
- Constructive algorithms and hybrid meta-heuristics are developed and applied to find solutions.
- Real world problems have been solved in numerical testing.

Abstract

To effectively utilise hospital beds, operating rooms (OR) and other treatment spaces, it is necessary to precisely plan patient admissions and treatments in advance. As patient treatment and recovery times are unequal and uncertain, this is not easy. In response, a sophisticated flexible job-shop scheduling (FJSS) model is introduced, whereby patients, beds, hospital wards and health care activities are respectively treated as jobs, single machines, parallel machines and operations. Our approach is novel because an entire hospital is describable and schedulable in one integrated approach. The scheduling model can be used to recompute timings after deviations, delays, postponements and cancellations. It also includes advanced conditions such as 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. To solve the FJSS problem, constructive algorithms and hybrid meta-heuristics have been developed. Our numerical testing shows that the proposed solution techniques are capable of solving problems of real world size. This outcome further highlights the value of the scheduling model and its potential for integration into actual hospital information systems.

Keywords

Scheduling; Flexible job shop; Hospital scheduling; Disjunctive graph model; Hybrid meta-heuristics

1. Introduction

Hospitals are critical elements of health care systems and access to their services is very competitive around the world. As demands have increased many hospitals have become larger. It is apparent that expansion activities cannot be performed without limit, as there are many physical and financial impediments. To achieve further improvements and to restrict cost increases, hospitals may need to operate more efficiently with the resources they already possess. To operate better, Heiser (2013) reports that hospital scheduling is beneficial as it increases surgeon productivity and revenue, decreases the cost per surgical procedure, improves staff productivity and morale, and leads to a more efficient usage of facilities. Our conversations with local health care professionals have also provided anecdotal evidence that if bed-spaces are scheduled better, patients can be placed within the home ward of a specialty to recover and may not need to be sent to other wards as outliers. When placed in the home ward of a specialty, patients receive better care from nursing staff and surgeons, are happier, and generally recover more quickly.

Hospital scheduling however is challenging. There are two types of competing clients. The elective patients are identified in advance and have specific, reasonably well known health care requirements, aka clinical pathways. Clinical pathways are standardised, evidence-based multidisciplinary management plans, which identify an appropriate sequence of clinical interventions, timeframes, milestones and expected outcomes for homogenous patient group (Queensland Health, 2017). Acute patients in contrast emerge irregularly and have dynamic conditions. Planning for known demand traditionally involves (i) the assignment of patients to resources over time, and (ii) the creation of a schedule. Planning for unknown demand is more difficult and a static plan cannot be created. Dynamic assignment and dispatching is hence necessary when acute patients arrive. Those activities inevitably corrupt the existing schedule and the pre-established plan of work. Evidently the problem of jointly considering the aforementioned demand types is very difficult.

It is important for hospitals to treat patients quickly and efficiently and to use their resources wisely. By more efficiently utilising the existing hospital resources, such as beds and other treatment spaces, it may be possible to increase the number of patients a hospital would otherwise treat, or reduce the number of resources that are required to meet given demands. To do this it is theorised that a kind of flexible job shop scheduling (FJSS) model may be helpful. This article investigates the technical details behind that approach. The intention is to embed that approach within an integrated rolling horizon hospital planning and rescheduling framework within an intelligent hospital information and management system. To our knowledge no other paper has explicitly addressed this problem in a comparable level of detail. This article builds upon prior research in Burdett and Kozan (2015) and to some extent Luscombe and Kozan (2016). In the former article, robust schedules in a similar environment were developed. That was accomplished via the insertion of strategically placed idle time (i.e. buffering). Each job in that article consisted of a single activity, and was processed in a single area, by a set of flexible machine resources. In contrast this article's system is multi-staged and each job has a full set of activities. This article does not explicitly consider the selection of buffering to handle uncertain activity times. Another article integrating buffer selection is planned in the future which will do more justice to handling uncertain processing times. In the later article Luscombe and Kozan (2016) developed job shop scheduling techniques to coordinate the activities and resources present in emergency departments.

The FJSS problem (i.e. FJSP) is an extension of the classical job shop scheduling problem. In the FJSP however each job activity can be processed by a variety of candidate machines (aka resources). The goal is to assign each activity to a machine, and to sequence those activities, so that the schedule horizon is minimal. The main purpose of formulating and solving the FJSP is to schedule the hospital's activities and to reschedule when existing plans become obsolete. This is a certain event as time elapses, no matter what plan is created. So this approach is essential. The FJSP also facilitates other functionalities which are equally if not more immediately useful to planners within hospitals. The FJSP provides the means to visualise what is going on in the hospital. It provides the capability to determine activity timings and the potential occupancy of the hospital “upfront”. Furthermore it provides the means to analyse the effect of variations in treatment times, recovery times, and other activity durations “upfront”. Used online the FJSP can notify staff of activities that are nearing completion so patients can be made ready sooner and hence free up resources quicker. Violated timing constraints and other infeasibilities can also be flagged. The FJSP can analyse alternative resource assignments and sequencing decisions and facilitates optimisation approaches that seek improved resource assignments and improved task sequencing on resources.

The FJSP approach can be used to provide a benchmark to compare current practices and to determine the operational capacity level of the hospital (Burdett, 2016; Burdett & Kozan, 2016; Burdett, Kozan, Sinnot, Cook, & Tian, 2017).

In the next section, prior research is discussed. The FJSP and the main technical developments are then introduced in Section 3. Solution techniques for the FJSP are discussed in Section 4. A case study and numerical investigations are then reported in Section 5. Concluding remarks are finally made in Section 6. For fast access and to ease understanding, the notation, parameters and variables used throughout this article are listed in Appendix A.

2. Previous research

Recent trends and features of hospital scheduling research are reviewed in this section. Machine scheduling research relevant to the planning of activities in hospitals is also examined.

2.1. Hospital scheduling approach

Many articles can be found on hospital planning and scheduling in the literature. For a comprehensive taxonomy of decision approaches, Hulshof, Kortbeek, Boucherie, Hans, and Bakker (2012) may be consulted. Javid, Jalali, and Klassen (2017) may also be consulted for comprehensive details of optimisation approaches associated with patient scheduling. In retrospect the viewpoints and solution approaches proposed in the literature vary greatly. In Carnes, Price, Levi, Dunn, Daily, and Moss (2011), it is believed, that a hospital does not have the luxury of specifying in advance, when every patient will be admitted and discharged from the hospital. Braubach, Pokahr, and Lamersdorf (2014) further judge that many scheduling approaches are unsuitable for real life application. One reason for this is that despite its general usefulness, efficient mechanisms for patient scheduling are difficult to devise due to some inherent characteristics of hospitals. For example there are many uncertainties that make precise planning ahead for even a few days very difficult, if not impossible. Patient cure remains a task that cannot be pre-planned completely in advance, even for specific clinical pathways. Hence it is necessary for patient scheduling to accommodate these characteristics and to provide a flexible and fast mechanism that can handle uncertainties at many levels. Consequently Braubach et al. (2014) propose a decentralised approach whereby patients and resources are modelled as agents with individual goals.

Scheduling policies can have a great effect on hospital performance. Cappanera, Visintin, and Banditori (2014) compared the performance of three different scheduling policies. Their scheduling objective was to maximise the number of scheduled surgeries and to balance the beds and operating room workloads. A discrete event simulator (DES) was used to identify the schedule's robustness. Their numerical investigations showed no clear distinction between the different policies and no single technique was superior in all situations.

The scheduling of elective patients has received significant attention under the alias of the surgical case scheduling problem. As these patients are visible, their treatments can in theory be planned in advance. The article by Pham and Klinkert (2008) is noteworthy and well cited. It motivates a more holistic approach to scheduling inpatients and outpatients, and coordinating hospital resources. Their work provides a strong foundation for this article's approach. Pham and Klinkert (2008) in particular proposed a resource constrained multi-mode blocking job shop framework. In their decision making problem, resources are first assigned to each activity of a job and then activities are scheduled on their assigned resources. The execution of an activity requires a set of resources and each set of resources is called a mode. Blocking occurs because there are no buffers. A mixed integer linear programing (MILP) model with makespan objective was formulated and solved using CPLEX. Their computational study however considered hypothetical scenarios of small size (i.e. 30 surgical cases) and larger instances were not solved. Another noteworthy article is Gartner and Kolisch (2014). In their approach the main decision is to decide on which day each procedure of each patient's clinical pathway should be done, taking into account the sequence of procedures as well as scarce clinical resources. A fixed admission date was assumed in their first model but this was relaxed in the second model. Both models were however embedded into a rolling horizon approach in order to cope with stochastic data. Each afternoon a decision is made as to which patients will be considered for surgery on the next day. An instance of the decision problem is then solved with the most recent data. Our approach is a more fine-grained operational approach in contrast to Gartner and Kolisch's strategic approach. It includes more technical constraints and peculiarities of hospitals. Our unit of time for scheduling is smaller and this allows more accurate and timely scheduling of activities.

Operating rooms, and theatres including multiple rooms, are an expensive resource and drive many activities within hospitals. Master surgery scheduling otherwise known as surgical block scheduling is consequently an important way to coordinate and plan hospital's activities. This decision problem has been considered numerously in the literature and recent trends are to integrate it with other decision problems. Chow, Puterman, Salehirad, Huang, and Atkins (2011) for instance combined a surgical schedule optimiser

(SSO) (i.e. an MIP model) with a bed utilisation simulator (BUS). The approach is used to level surgical bed occupancy and to reduce ward congestion and reduce peak bed occupancies. Fugener, Hans, Kolisch, Kortbeek, and Vanberkel (2014) recently developed a stochastic analytical model for master surgery scheduling (MSS). The purpose of the MSS problem is to assign block operating room time to different surgical specialities. The downstream resources are included as additional costs. Different measures were proposed to quantify these costs. Their approach includes downstream resources, such as intensive care units (ICU) and wards. That article provides a step towards the development of an integrated approach for a whole hospital. Their feedback and experience is another source of motivation for this article's approach. Spratt and Kozan (2016) addressed the master surgical scheduling problem and the surgical case assignment problem that occurs in Australian public hospitals. They developed an integrated solution approach and formulated a mixed integer nonlinear programming model. This integrated decision problem was solved using a variety of hybrid meta-heuristics.

2.2. Machine scheduling research

As indicated in Pham and Klinkert (2008) hospital scheduling is analogous to solving flexible job shop (FJSP) and flexible flow shop (FFSP) scheduling problems. They are notoriously difficult combinatorial optimisation problem and are NP-hard. We briefly discuss recent approaches and their impact on this research. Pezella, Morganti, and Ciaschetti (2008) and Chen, Wu, Chen, and Chen (2012) solve FJSP using Genetic Algorithm (GA). In the later article sequence dependant setup times are included. Yazdani, Amiri, and Zandieh (2010) developed a parallel variable neighbourhood search (PVNS) algorithm for the FJSP with makespan objective. Changes to both the assignment and sequencing are made. Groflin, Pham, and Burgy (2011) considered a flexible blocking job shop (FBJSP) with sequence dependant transfers and setup times. A Tabu Search (TS) approach was primarily advocated and tested. Job insertion and feasibility issues were discussed in depth and incorporated. The authors report the need to reduce their computation times for large practical problems. Sauvey and Trabelsi (2015) considered identical machines and included mixed blocking constraints. Their approach has two parts—machine selection and operations scheduling. This style of approach is evidently superior to others. Xia and Wu (2005) and Zhang, Shao, and Gao (2009) solve multi-objective versions. In the former article, a hybrid Simulated Annealing (SA) and Particle Swarm (PSO) algorithm is implemented. PSO and TS are combined in the later. In both articles three objectives were included. These are aggregated into one objective using a weighted sum approach. Jungwattanakit, Reodecha, Chaovalitwongse, and Werner (2009) have considered flexible flow shop problems with setup times, dual criteria and unrelated parallel machines. Their objective is a convex sum of makespan and the number of tardy jobs. They implemented SA, GA and TS meta-heuristics to solve problem instances with up to 50 jobs and 20 stages. They also implemented and tested a variety of heuristics, and constructive algorithms.

The incorporation of blocking and no-wait conditions is a necessity in this research. These conditions are important additions to flow and job shop scheduling problems (Burdett & Kozan, 2001a, 2001b, 2003). The following are particularly noteworthy articles: Burdett and Kozan (2009, 2010a, 2010b), Liu and Kozan (2011, 2012, 2016) and Kozan and Liu (2012). They apply various constructive and meta-heuristic techniques to solve train sequencing and scheduling problems as JSSP with blocking, no-wait conditions and capacitated buffers. Hayasaka and Hino (2012) proposed a solution correction approach to correct infeasibilities caused by exchanging two consecutive activities on the same machine of a blocking job shop schedule. Their approach involves the introduction of reverse conjunctive arcs. In their view swaps are not counted as a factor that causes infeasible schedules. Zeng et al. (2014) considered the creation of schedules for a difficult industrial application. In their approach the problem is represented as a job shop problem with blocking constraints and transfer times between machines. Jobs are transferred between machines using a limited number of automatic guided vehicles (AGV) and blocking with swapping is also permissible. The problem was first formulated as a MINLP problem. Due to the computational intractability of that model, a two stage heuristic algorithm was proposed instead. As blocking constraints are difficult to represent in a disjunctive graph model some custom algorithms are devised to schedule activities. These are based upon adjustments to the standard disjunctive graph model.

Our analysis of past research suggests that more work is still required on this topic. Previous test instances are relatively small compared to real life applications. Most reported test cases have about 15 jobs and 10 machines but real hospitals have hundreds on any given day or week. Additional complexities and advanced features unique to health care seem not to have been included in one holistic approach, although they have been considered individually in the scheduling literature (as shown above) for some time. The incorporation and integration of multiple scheduling objectives is also limited. Hospitals evidently have many competing entities and criteria.

3. Hospital scheduling approach

This section describes a generic integrated hospital planning and rescheduling framework. An overview of the main details is summarised in Table 1.

Table 1. Details of the integrated hospital planning and rescheduling framework.

What to schedule	Surgical patient's pre-operative care (POC), surgery (SUR), post-operative care (PAC), recovery (REC)
	Medical patients treatment and consultation (MED), diagnostic tasks (DIAG), rehabilitation (REH)
What to sequence	The occupancy of each bed space, treatment space and operating room
What to assign	Assign patients to bed spaces and to treatment spaces from the set of candidates
What to dispatch	Dynamically schedule acute surgical patients as they eventuate
When to reschedule	Required when parameters change or the system is disrupted. For example:
	Required when acute patients arrive and urgently need surgery
	Required when patient's length of stay or other treatment is finished early or late
Robustness	Achieved by adding strategic buffering between different activities
Multiple objectives	Maximise throughput of patients; Maximise schedule robustness

This holistic approach includes the recovery wards, operating rooms, intensive care units and the surgical care areas within a hospital. The emergency department is not included at this stage but patients originating there may be included and then scheduled when they become inpatients. This approach can be used to schedule any medical or surgical activity in the hospital. From a practical perspective that level of detail may be superfluous and excessive. Consequently, higher level activities should be concentrated upon and lower level activities should be aggregated. Their effect on the system is more critical. Anecdotal evidence suggests that nurses and other hospital personnel within the different hospital areas are adept at coordinating low level activities themselves.

A key ingredient is the FJSS model and the solution of the FJSP. The FJSP involves the assignment of forthcoming patients to different beds and operating rooms and the timing of health care treatments and other activities. The aforementioned assignments are selectable and greatly affect how the system performs. To our knowledge, most hospitals do not schedule the occupancy of beds in advance. Beds are acquired at “close call” and staff within bed management units “wheel and deal” with different specialty units as need be. Our approach is therefore a new consideration for hospitals. In contrast the operating rooms have always been scheduled. They are the engine that drives the hospital and are significantly more expensive and scarce.

Hospital schedules may be evaluated by many different criterion. The selection of a scheduling criterion does not affect the proposed scheduling model. The optimisation techniques are also unaffected unless multiple criteria are defined and Pareto optimal solutions are sought. Which single objective or collection of objectives is best for hospitals is somewhat debatable. Evidently it is important to construct schedules that utilise hospital resources efficiently. Resources should be highly utilised but not to the point where small deviations cause many delays and postponements, poor schedule performance, and schedule infeasibility. In this article, a schedule with a minimum makespan is sought. That criterion is a well-known scheduling measure. In the parallel machine environment that is considered, it balances resource usage and maximises utilisation and throughput. Making the schedule robust is equally important as minimising schedule horizon is potentially questionable in an environment with uncertain processing times. That is the topic of future work however and involves the selection of appropriate buffering. Other objectives that measure access block, patient delays, number of postponed or cancelled activities, may also be used.

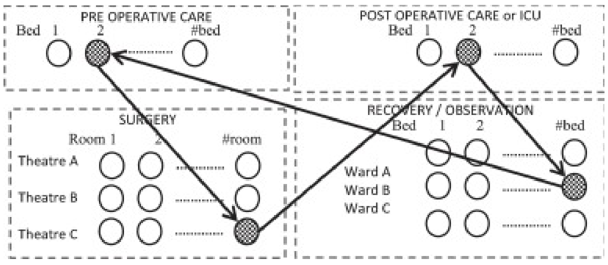
Rescheduling is needed in this environment as no level of planning can be exact when so many things can change or vary. Rescheduling is also a means to handle the implementation of less robust schedules with high utilisation. The rescheduling of activities is necessary when beds, treatment spaces and other resources are not available when planned or when patients are not ready. Rescheduling provides new activity timings and may result in the reassignment of beds and other treatment spaces. It may also result in the postponement or cancelation of previously scheduled activities if they cannot be accommodated.

3.1. Flexible job shop scheduling (FJSS)

The FJSP problem is formally defined as follows. A given set of jobs $J = \{J_1, J_2, \dots\}$ must be performed on a set of machines $M = \{M_1, M_2, \dots\}$. The processing of job J_i consists of several activities $\{o_{i,1}, o_{i,2}, \dots, o_{i,K_i}\}$ to be performed in the given order, where K_i is the number of activities in job J_i. The complete set of activities is $O = \cup_i \cup_k o_{i,k}$. The activity processing times denoted by $\rho_{i,\,k}$ depend on the machine assigned. This is denoted by $m_{i,\,k}$. The set of candidate machines that may be used to process $o_{i,\,k}$ is given by $\widetilde{M}_{i,k} \subset M$. The assignment of $o_{i,\,k}$ to a particular machine $j \in \widetilde{M}_{i,k}$ is denoted by the binary variable $a_{i,\,k,\,j}$. For instance, if $m_{i,k} = j$ then $a_{i,k,j} = 1$. A three field notation (i.e. machines / constraints / objective) is used to explain different variants of the FJSP. For example "FJ/blocking/Cmax" is a description of a FJSP with makespan objective and general blocking conditions.

To apply a FJSS approach it is necessary to treat patients as jobs, health care activities within the patient's clinical pathway as job operations, beds and treatment spaces as machines and hospital areas consisting of multiple treatment spaces as parallel machines. As the term "operation" may also be used to describe surgical activities, we do not use it to describe activities in the remainder of this article. This FJSS approach facilitates the scheduling of treatment and care activities within the many different areas of the hospital. There is no limit to the number of activities in a job, nor is there a limit to the number of times a job can visit the same machine. In other words re-entrant paths are possible. Each treatment area $w \in W$ has a collection of treatment spaces $\pi \in \Pi_w$ that may be used. In practice there is little difference between alternate wards, alternate treatment areas and alternate treatment spaces. Hence identical machines can be assumed. The possibility of non-identical machines is theoretically possible and must be incorporated.

The FJSS approach is adopted because patients ($p \in P$) can be assigned to a variety of wards and treatment areas within their clinical pathway. A basic clinical pathway for each patient is as follows: $PCP_p = \{(\phi, u, \rho, W_p, \Pi_p)\}$. Each tuple (ϕ, u, ρ, W_p, Π_p) contains the activity type ($\phi \in \Phi$), the unit performing the activity ($u \in U$), the activities expected processing time ρ, a set of permissible wards ($W_p \subseteq W$), and a set of permissible treatment spaces ($\Pi_p \subseteq \Pi$). The tuple may also include a complete list of all other resources such as doctors and nurses. An example of a patient's flexible pathway is shown in Fig. 1.



Download high-res image (205KB) Download full-size image

Fig. 1. Example of flexible pathway for a surgical patient.

If the patient starts their stay in the ward this surgical patient's visit includes an extra stage. The patient's surgery can be performed in three different operating theatres and the patient's recovery can occur in one of three different wards. A specific path is shown, including the beds assigned (i.e. filled circles). Requirements for a stay in the intensive care unit may also be added to this diagram. This type of basic pathway is predominantly the same for all surgical patients. Advanced pathways however can occur when patients need to stay in the ICU and when patients require multiple surgeries.

To apply this FJSS approach, a translation process such as the one summarised in Table 2 is required to perform the following mapping: $(W, \Pi, P) \rightarrow (J, M, O)$. This translation process does not completely describe all the technical conditions required. A number of "constructs" have therefore been created to correctly specify the necessary technical conditions. They are listed in Table 3 and will be discussed in more detail in later sections.

Table 2. Translation process.

Step	Conversion	Description
i.	$\forall w \in W$: $\forall \pi \in \Pi_w$: Convert space $\pi \rightarrow$ machine j	Convert each space to a machine
ii.	$\forall p \in P$: Convert patient $p \rightarrow$ job i	Convert each patient to a job
iii.	$\forall p \in P, \forall (\phi, u, \rho, W_p, \Pi_p) \in PCP_p$: Create activity $o_{i,k}$ & set $\rho_{i,k} = \rho$ & $\widetilde{M}_{i,k} = \Pi_p$	Convert each task to an activity

Table 3. Specification constructs.

Format	Description
$SYNCH\_1, o_{i,k}, o_{i,k'}$	Define a start-start synchronisation between specific activities of a job
$SYNCH\_2, i$	Define a finish-start synchronisation between the last and first activity of a job
RESTRICT $_{START}$, $o_{i,k}$, {*itp*}	Define a list of start time restrictions for activity $o_{i,k}$
$RESTRICT\_END, o_{i,k}, \{itp\}$	Define a list of depart time restrictions for activity $o_{i,k}$
RESTRICT, j, {itp}	Define a list of time intervals where a machine is off limits, i.e. closed
SETUP, j, val	Define a machine setup time
$PRE\_SETUP, o_{i,k}, val$	Define a setup time before processing activity $o_{i,k}$
$POST\_SETUP, o_{i,k}, val$	Define a setup time after processing activity $o_{i,k}$
OVERLAP, $o_{i,k}$, ***val***	Define finish-start delay between activity $o_{i,k}$ and its successor in job i
$FIXED\_TIME, o_{i,k}, b, e$	Define a fixed start and end time for activity $o_{i,k}$
$FIXED\_MACHINE, o_{i,k}, j$	Define a fixed machine for activity $o_{i,k}$
$FIXED\_SEQ, j, \{o_{i,k}\}$	Define a fixed sequence of activities on machine j
BUFFER, $o_{i,k}$, ***val***	Set the buffering after activity $o_{i,k}$
$BUFFER\_LIM, o_{i,k}, val$	Define a limit to the buffering after activity $o_{i,k}$
$SAME\_MACHINE, \{o_{i,k}\}$	Set same machine requirement for activities of a job

Format	Description
$ACQUIRE, o_{i,k}, o_{i,k'}$	Set a machine acquisition requirement for $o_{i,k}$ that starts when $o_{i,k'}$ begins

The following constructs, “$BUSINESS\_START, o_{i,k}, wk$”, “$BUSINESS\_END, o_{i,k}, wk$”, and “*BUSINESS, j, wk*” have also been defined. These constructs specify, for both activities and machines, that business hours are available only for the defined number of weeks. These constructs automatically generate a full list of invalid time periods (ITP).

3.2. An alternative disjunctive graph model

Disjunctive graphs (DJG) are the basis of many successful strategies for solving job shop scheduling problems (JSSP). In a disjunctive graph $G = (V, A)$ each node $v \in V | v \notin \{so, si\}$ represents an activity that must be sequenced, and each arc $a \in A$ represents a precedence requirement. The graph has fixed conjunctive arcs (CJA) and a set of selectable disjunctive arcs (DJA). When evaluated, for instance, via a longest path algorithm, the DJG provides activity timings.

An alternative disjunctive graph model (ADGM) is now introduced for hospital applications. It includes a multitude of advanced technical conditions such as 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. The exact details of the ADGM are shown below:

$$G = (V, A = CJA \cup DJA)\,;\; V = O \cup \{so, si\}\,;\; DJA = \cup_{j \in M} dja_j; \tag{1}$$

$$CJA = \{(so, o_{i,1}, 0)\ \forall o_{i,1} \in O\} \cup \{(o_{i,k}, \mathbf{succ}[o_{i,k}], \omega_{i,k})\,|\forall o_{i,k} \in O\}\,; \tag{2}$$

$$\mathbf{succ}[o_{i,k}] = o_{i,k+1}\ \forall k \in [1, K_i - 1]\,;\ \mathbf{succ}[o_{i,K_i}] = si \tag{3}$$

$$dja_j = \cup_{k \in [1,|\sigma_j|-1]} \mathbf{F}[\sigma_{j,k}, \sigma_{j,k+1}]\ \forall j \in M \tag{4}$$

$$\mathbf{F}[\sigma_{j,k}, \sigma_{j,k+1}] = \begin{cases} (o_{i^*,k^*} = \mathbf{succ}[\sigma_{j,k}], \sigma_{j,k+1}, s_j - \rho_{i^*,k^*} - \omega_{\sigma_{j,k}}) & \text{if } (o_{i^*,k^*} \neq si) \\ (o_{i^*,k^*} = \mathbf{succ}[\sigma_{j,k}], \sigma_{j,k+1}, s_j) & \text{if } (o_{i^*,k^*} = si) \end{cases} \tag{5}$$

$$\rho_{i,k} = \sum_{\forall j \in \widetilde{M}_{i,k}} (a_{i,k,j} \cdot \rho_{i,k,j})\ \ \forall i \in J, \forall k \in [1, K_i) \tag{6}$$

$$m_{i,k} = \sum_{\forall j \in \widetilde{M}_{i,k}} (j \cdot a_{i,k,j})\ \ \forall i \in J, \forall k \in [1, K_i] \tag{7}$$

$$\omega_{i,k} = \mathbf{G}[m_{i,k}, m_{i,k+1}]\ \ \forall i \in J, \forall k \in [1, K_i) \tag{8}$$

In this description, each activity of a job is “blocking” except the last; it is “ideal”. This means that job activities cannot terminate until the location of the next activity is free/acquired. The last activity is different. Once complete the job can terminate, and leave the system.

Function **G** provides the mapping $(j, j') \to t | t \in \mathbb{R}$. Many different functions may be used and will be discussed in the next section. The sequence of activities performed on machine *j* is denoted as $\sigma_j = [\sigma_{j,1}, \sigma_{j,2}, \ldots,]$ where $\sigma_{j,k} \in O$. The disjunctive arcs in (5) are created between all pairs of adjacent activities in the machine sequences. If the activity is from the same job, then a disjunctive arc is still warranted. For instance, it ensures that a job activity does not occur in a machine sequence before predecessor activities of the same job (i.e. on that machine). This disjunctive arc may be equivalent to an existing conjunctive arc, but may not have the same arc weight. Consequently, multiple arcs are permitted between the same nodes. When removing disjunctive arcs, care must therefore be taken to remove the correct arc (i.e. an arc with a particular weight), and other occurrences of the arc must be left “in-place”. An alternative approach is to pre-identify the predecessor and successor activities of the job on the same machine, and to use this information when making sequencing decisions.

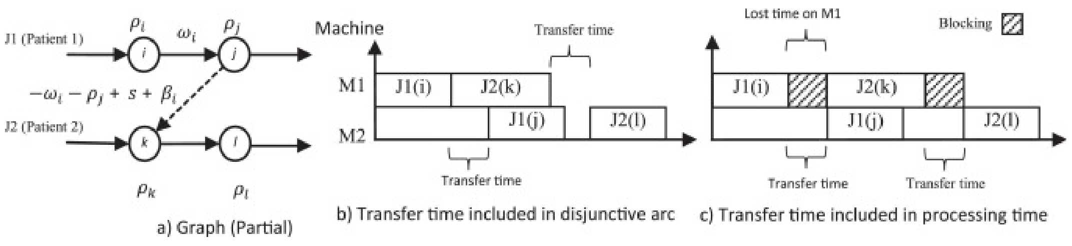
3.2.1. Transfer, setups and buffering

Hospitals are large places. Transfer times occur when patients are moved between different areas in the hospital. A variety of setup activities may need to be imposed before patients can be treated. Setup times are commonly incurred in a variety of places, for instance in ward beds and operating rooms. In this article transfers and setups are denoted by ω and s respectively. Different transfers and setups are shown in Table 4.

Table 4. Transfer time and setup time alternatives.

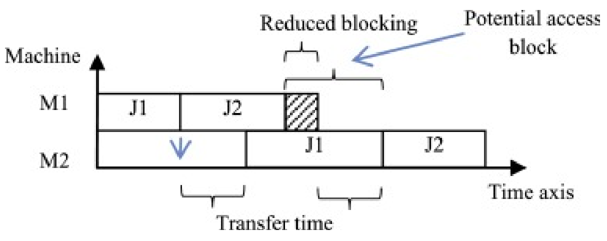
	Transfer description	**Mapping**
i.	Between specific job activities. Independent of machine assignments.	$(o_{i,k}, o_{i,k+1}) \rightarrow \omega_{i,k}$
ii.	Between specific machines. Independent of job activity.	$(j, j') \rightarrow \omega_{j,j'}$
iii.	Combination of the former.	$(o_{i,k}, m_{i,k}, o_{i,k+1}, m_{i,k+1}) \rightarrow \omega_{i,k}$
	Setup description	Mapping
iv.	For a machine. Between all activities processed there.	$j \rightarrow s_j$
v.	For an activity. Independent of machine.	$o_{i,k} \rightarrow s_{i,k}$
vi.	For an activity performed on a specific machine.	$(o_{i,k}, m_{i,k}) \rightarrow s_{i,k}$

The transfer time $\omega_{i,k}$ must be included in alternate disjunctive arcs as shown in Fig. 2a to ensure the starting time of the next job activity is correct. The transfer time should be included in the disjunctive arc and not in the node weight as lost time will occur. This is shown in Fig. 2c with relation to the former scenario in Fig. 2b. The transfer time also affects the departure time and ultimately the amount of blocking that an activity incurs. In Fig. 3, J2 leaves earlier than would otherwise have occurred in the case of zero transfer time. Hence M1 is free sooner for other jobs. If the transfer cannot strictly occur until the successor machine (i.e. M2) is ready, then the full amount of blocking will occur.



Download high-res image (190KB) Download full-size image

Fig. 2. Transfer time incorporation and effect.



Download high-res image (87KB) Download full-size image

Fig. 3. An example of reduced blocking caused by the presence of transfer times.

The buffering denoted by $\beta_{i,k}$ may be placed after every activity to provide schedule robustness when activity processing times are longer than anticipated in practice. For the purposes of this article the buffering is assumed given. The selection of buffering has been considered in Burdett and Kozan (2015).

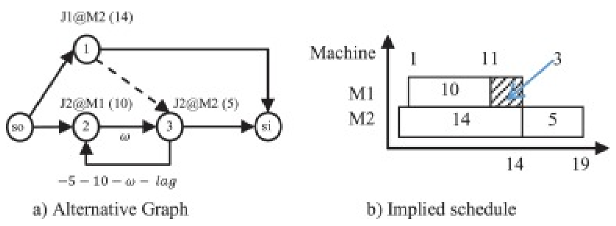
The constructs "SETUP", "PRE_SETUP", "POST_SETUP" and "BUFFER" have been provided to facilitate the aforementioned setup and buffering requirements. The construct "OVERLAP" can be used to facilitate transfers of various kinds.

3.2.2. Blocking limitations and no wait conditions

Different finish-start timing conditions (i.e. lags) need to be enforced when scheduling in hospitals as medical and surgical activities must occur promptly without much delay. Different possibilities are facilitated by altering two user input parameters, namely $lag_{i,k}$ and $type_{i,k}$. The $lag_{i,k}$ parameter describes the permissible interval of time between two successive activities of a patient job. Under normal circumstances, $lag_{i,k} \in [0, \infty]$. Scenarios where $lag_{i,k}$ is less than zero will be discussed in due course. The parameter $type_{i,k}$ describes what happens after an activity is completed. The options are as follows: $type_{i,k} \in \{ideal, blocking, no\_wait\}$. The first option allows the job to move into an intermediate "storage" area straight away. The second option forces the job to wait until released (i.e. blocking) as there is no intermediate storage. The last option implies that the job cannot be delayed.

In hospitals the possibility of blocking is always present. Once patients are admitted there is no stage in the clinical pathway where blocking cannot occur as patients must be held in actual hospital spaces that are limited and constrained. Blocking is evidently something that should be minimised because of the repercussions to patient's health and to overall system performance. Standard blocking conditions are activated when $lag_{i,k} = \infty$. The creation of alternate disjunctive arc is needed, that originate from an activity's successor, within the current job. In actuality patients must be promptly moved to a different location to further their care/treatment. If needed, successor activities may be commenced at the patient's current location; but this is not standard practice. Hence the amount of blocking is restricted. This is facilitated by setting $lag_{i,k} < \infty$.There are a number of ways to enforce the restriction on the amount of blocking at any stage in a patient's clinical pathway. A simple approach involving the penalisation of excessive blocking (i.e. as a soft constraint) is viable but that approach offers no guarantee of feasibility. A more elegant approach involving the addition of "backwards" synchronisation arcs may be used to explicitly enforce the aforementioned time lag constraints. These backward arcs were introduced in Mascis and Pacciarelli (2002). These particular modifications ensure that activities are scheduled within $lag_{i,k}$ time units of their predecessor's completion. This backward scheduling approach ensures that there is no waiting time before processing. In other words, an activity starts as soon as it arrives. This approach also makes the disjunctive graph cyclic, which adds significant extra overhead to graph evaluations. This issue will be discussed later.

A demonstration of backward arcs is shown in Fig. 4a. In that diagram two jobs J1 and J2 need to be scheduled. J2 has two activities but J1 has only a single activity. Both jobs are processed on machine M2 in the order (J1, J2). The disjunctive arc required to enforce that precedence requirement is dotted. A blocking limitation is defined for J2 of three time units (i.e. $lag = 3$). Hence a backward arc linking node 3 to node 2 is required. The Gantt chart associated with that disjunctive graph is shown in Fig. 4b.



Download high-res image (96KB) Download full-size image

Fig. 4. Demonstration of backward arc to enforce blocking limitation.

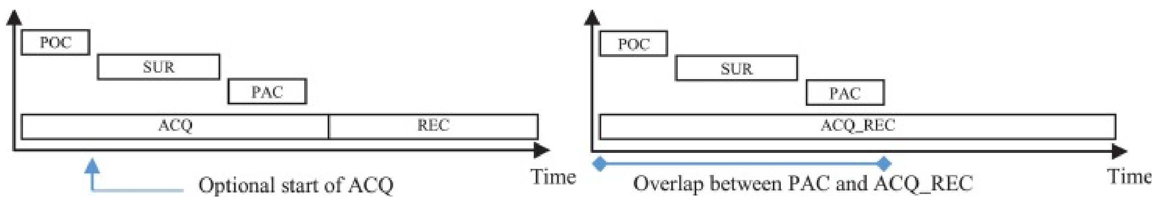
To fully eliminate blocking, job activities may be defined as no wait. The no wait condition is achieved by setting $lag_{i,k} = 0$. Defining activities as no wait may cause an inferior schedule to be produced in comparison to other scenarios. The time lag parameter is clearly irrelevant when the activity is “ideal”.

3.2.3. Ward bed acquisition (WBA)

Ward beds need to be acquired and then set aside for specific patients to occupy. The acquisition should occur when the patient begins their treatment, either at the start of POC or at the start of surgery. The acquisition time however depends on various factors but should occur before the patient arrives at the ward. This requirement is a non-standard feature and to our knowledge is not present in many scheduling applications. To facilitate this condition, the strategies shown in Table 5 are considered for a standard surgical patient job whose pathway is (POC, SUR, PAC, REC). Each strategy has different strengths and weaknesses. In essence they involve the definition of the acquisition (ACQ) as a separate job activity or as a sub component of another activity (see Fig. 5). The start of the ACQ activity must also align with the start of a predecessor activity such as the POC or SUR. Fig. 5 demonstrates the net effect of the different strategies.

Table 5. Different approaches for modelling the ward bed acquisition requirement.

Option	#Activity	Pathway	Conditions
A	5	$[POC, SUR, PAC \,\&\, ACQ, REC]$ $PAC \wedge ACQ \prec REC$	Same machine: (*ACQ*, *REC*).
B1	5	[*ACQ*, *POC*, *SUR*, *PAC*, *REC*]	Same machine: (*ACQ*, *REC*). $\rho_{ACQ} = 0$
B2	5	[*ACQ*, *POC*, *SUR*, *PAC*, *REC*]	Overlap between *ACQ* and *POC*. Same machine: (*ACQ*, *REC*). $\rho_{ACQ} = \rho_{POC} + \rho_{SUR} + \rho_{PAC} + transfers$
B3	5	[*POC*, *SUR*, *PAC*, *ACQ*, *REC*]	Overlap between *PAC* and *ACQ*. Same machine: (*ACQ*, *REC*). $\rho_{ACQ} = \rho_{POC} + \rho_{SUR} + \rho_{PAC} + transfers$
C	4	$[POC, SUR, PAC, ACQ\_REC]$	Overlap between *PAC* and *ACQ_REC*

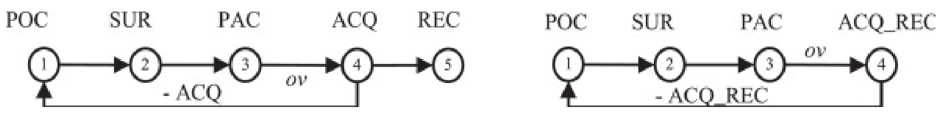


Download high-res image (121KB) Download full-size image

Fig. 5. Alternative approach: a) a five activity job, b) a four activity job.

In the aforementioned strategies the overlap is computed as $ov = -(\rho_{POC} + \rho_{SUR} + \rho_{PAC} + transfers)$. The construct "ACQUIRE" is used to define seizure activities and to compute their processing time. Except for B1, the ACQ component includes the processing times and transfers of predecessor activities. In strategy A and B the ACQ activity is separate. This is problematic as the ACQ activity must always be performed on the same machine as the recovery task. This technical condition is forthwith called a "same machine assignment" (SMA) requirement. It should be noted that in each SMA there is essentially only one assignment decision. Methods for enforcing SMA in job shop scheduling are not standard. Hence some innovations will be introduced and demonstrated later. The aforementioned construct "SAME_MACHINE" is used to define SMA requirements.

In this article B3 and C have been adopted. Option C is believed best as the "acquire" and recovery are merged into one and the SMA is not needed. In addition the disjunctive graph is smaller as there is one less activity per job. In each of these strategies the conjunctive arcs required, for a typical job are shown in Fig. 6. The reverse "synchronisation" arc ensures a start-start lag of zero between POC and ACQ or between POC and ACQ_REC. The introduction of the synchronisation arc creates a no wait job where every activity except the last is no-wait. The aforementioned construct "SYNCH_2" has been created to facilitate the synchronisation arc in strategy C. The construct "SYNCH_1" can be used to facilitate the synchronisation arc in strategy B3. These synchronisation constructs are generic. They can be integrated automatically with the acquisition construct.



Download high-res image (89KB) Download full-size image

Fig. 6. Conjunctive arcs respectively for option B3 and C.

The aforementioned approaches are effective; yet some care should be taken during implementations. When performing an analysis of processing time deviations, the overlap and ACQ processing time changes. This is because they are functions of the other activities. A basic implementation takes these values as static data. It is better that they be computed using specific functions. A revised approach is therefore suggested where the acquisition times are added on arcs and nodes where and when needed. Acquisition times in addition are not explicitly included within transfer times. The synchronisation arc weight is a function as is the overlap.

3.2.4. Deadlocks, cycles and blocking swaps

In classical JSSP's the disjunctive graph is acyclic if the solution is feasible, and the presence of cycles indicates precedence impossibilities, caused by the violation of certain conjunctive arcs. These cycles have positive length and must be restricted from occurring. In the FJSP problem however, the disjunctive graph is always cyclic because of the incorporation of backward synchronisation arcs. These cycles have a negative length in contrast to the aforementioned one whose length is positive. This distinction is important as the presence of a cycle of negative length can be handled with an appropriate scheduling algorithm; for instance, the Bellman Ford algorithm. The traditional longest path algorithm cannot be used if cycles of any type are present. The Bellman Ford algorithm in contrast is able to identify when there are cycles of positive length. It is also able to schedule correctly when there are cycles of negative length.

The Bellman Ford algorithm is primarily used to determine the shortest paths to each node from a single source in a directed graph. It is easily adapted for longest path problems. The distances are updated each iteration and become more accurate as the algorithm progresses. Every arc is considered during the iteration. This algorithm terminates if no changes are made during any iteration. Otherwise the number of iterations is the number of nodes minus one. The worst case time bound is *O*(|*V*||*E*|) but improvements can be applied to reduce calculations. Yen (1970) introduced several approaches. His first improvement reduces the number of relaxations required by only relaxing nodes if their distance has changed in the last iteration. This modification is particularly advantageous on disjunctive graph as they are quite dense.

In this FJSP some other cycles are also created, and these are valid too if their length is negative. They relate to the presence of blocking swaps. Blocking swaps are needed when two jobs need to swap places. Both jobs need to move simultaneously to resolve this type of deadlock, i.e. simultaneous swaps are needed. Deadlocks consisting of more than two jobs can also occur. In this hospital application deadlocks of this nature are unlikely because in most circumstances the flow is one way and there are many candidate treatment spaces.

3.2.5. Timing and occupancy restrictions

In hospitals there are restrictions on when activities can be performed during the day and when resources such as operating rooms can be used. These restrictions need to be included in scheduling activities and may be treated as either hard or soft constraints. Start time, completion time and occupancy restrictions (i.e. denoted STR, CTR, OTR) will now be discussed. For reference purposes they will be referred to in the following way: FJ/STR/Cmax, FJ/CTR/Cmax, FJ/OTR/Cmax. Those constraints are deemed to be strict and consequently are treated as hard constraints. Several approaches have been developed and these will be discussed in due course. Timing restrictions could alternatively be treated as soft constraints and timing violations could be introduced as a secondary objective that should be minimised. A variety of multi criteria scheduling approaches could then be utilised; a weighted sum approach being the simplest to implement. The application of multi-criteria scheduling techniques however is outside the scope of this article and a source of further research.

Correct start and end times can be achieved by applying a ***Right Shift Corrective Scheduling Policy (RSCSP)***. A RSCSP is straightforward to encode when invalid time periods (ITP) occur at the same time each week. For example:

Step 1: Determine which day the activity currently begins and ends. If the activity starts on Saturday or Sunday then postpone start time till Monday 8 am.

Step 2: Determine start time during the day. If start time occurs before 8 am, postpone till 8 am. If start time occurs after 5 pm, postpone till 8 am on the next workday (i.e. not Saturday or Sunday).

The aforementioned approach however is not generic, and not well suited to situations where ITP occur at any time and differ on each day of every week. A generic approach consequently needs to analyse a set of chronologically ordered ITP, where at worst, every activity may have a different list of ITP. In our proposed approach the aforementioned constructs “RESTRICT_START” and “RESTRICT_END” are used to specify the list of ITP. To implement these constructs, two strategies have been considered. The first utilises functions on nodes. In the disjunctive graph, the start and end time at each node, may need to be corrected. The function shown in (9) is used to perform those corrections. This function is written generically and *t* is any uncorrected value of time that may be input. The start time is first corrected and then the end time. The function's output is then recorded in place of the original aforementioned values.

$$\mathbf{Correct}\,(t) = \begin{cases} t & \text{if } \nexists z \in [1, |ITP|]\,|t_b^z \le t < t_e^z \\ t_e^z & \text{if } \exists z \in [1, |ITP|]\,|t_b^z \le t < t_e^z \end{cases} \tag{9}$$

Eq. (9) implies that the input time value is not altered if no ITP intervals can be found that bound it. Otherwise, the time is altered to that interval's upper value. Evaluating this function may be computationally expensive if the number of ITP is large, as many intervals must be checked. The aforementioned approach does not handle situations where machine occupancy is instead restricted in a given ITP. This is because an activity may overlap the ITP yet both the start and end times may occur correctly outside the ITP. Operating rooms are a typical example where occupancy restrictions occur. In many hospitals operating rooms only operate during business hours (i.e. Monday–Friday, 8 am–5 pm). In terms of elapsed time the set of ITP for a typical week is as follows: $ITP = \{[0,8], [17,24], [24,32], [41,48], [48,56], [65,72], [72,80], [89,96], [96,104], [113,120], [120, 144], [144,168]\}$. The last two intervals are for the whole of Saturday and Sunday. Upon closer scrutiny, the set of ITP can be reduced by merging the evening ITP with the morning ITP, and merging Saturday and Sunday. Hence: $ITP = \{[0,8], [17,32], [41,56], [65,80], [89,104], [113,168]\}$.

To incorporate occupancy restrictions the aforementioned construct "RESTRICT" is used. Several strategies are available to enforce occupancy restrictions. The choice of which method to use relies to some extent on whether ITPs are regular or not. If they are then the first strategy is more apt. The first strategy involves the application of *correction algorithms*. If an activity overlaps an ITP that occurs in $[t_b, t_e]$, the activity must be scheduled earlier or else its start time must be postponed to the end of the ITP, i.e. $b_{i,k} = t_e$. An overlap occurs when $t_b \in (b_{i,k}, c_{i,k}) \vee b_{i,k} \in (t_b, t_e)$. In other words, an ITP starts within $(b_{i,k}, c_{i,k})$ or else the activity begins within the ITP (t_b, t_e). Two algorithms are introduced. Their exact details are shown below:

The first algorithm determines if an activity is scheduled within an ITP. After the first "if" statement, we know that $b_{i,k} < t_e$. Hence if the activity occurs before t_b then we know that no overlap occurs. Otherwise an overlap does occur. The search terminates when the first if statement is not met, as the current ITP occurs before all later ITP. Algorithm 2 is necessary to correct an activity. This iterative algorithm utilises Algorithm 1. It corrects an activity if necessary. It then checks future ITPs too. It continues to correct an activity until an appropriate "valid time interval" is found. If an activity takes longer than all available time periods, then the activity cannot be feasibly scheduled.

Algorithm 1. **Check_ITP**(z, $b_{i,k}$, $e_{i,k}$, ITP).

```
{
  while ($z < |ITP|$) {
    $[t_b, t_e] = ITP_z$ ;
    if ($b_{i,k} \geq t_e$) $z = z + 1$;  // Start time occurs after the current ITP ends. Go to next ITP.
    else if ($c_{i,k} \leq t_b$) return NULL; // Status: feasible. The end time occurs before the ITP.
    else return $ITP_z$; // Status: infeasible. Return the ITP that overlaps with this activity.
  }
}
```

Algorithm 2. **CorrectActivity** ($\rho_{i,k}$, $b_{i,k}$, $c_{i,k}$, ITP).

```
{
  $z = 0$ ;
  do {
    $[t_b, t_e] = \text{Check\_ITP}$ ($z$, $b_{i,k}$, $e_{i,k}$, $ITP$);
    if ($\nexists[t_b, t_e]$) $z = |ITP|$ ; // Stop no correction required
    else {$b_{i,k} = t_e$;  $c_{i,k} = b_{i,k} + \rho_{i,k}$; } // Perform correction
  } while($z < |ITP|$);
}
```

An alternative approach is to utilise dummy activities. For each unavailable period a dummy activity can be created with a release time (*rlt*), deadline (*dln*) and processing time equal to the difference, i.e. $\rho = dln - rlt$. Each ITP activity must be assigned a fixed machine status. Hence it may not be assigned to other machines, and its candidate list must be of size one. On a particular machine several ITP can be grouped as an entire job. Each ITP activity is defined as ideal, and not blocking. Hence the source of disjunctive arcs involving ITP activity is from the activities node and not the successor. In other words, traditional disjunctive arcs are necessary. In the disjunctive graph the release time is facilitated by an arc from the source node to the activity with weight *rlt*. This timing constraint is strict and cannot be violated. The deadline is facilitated by an arc from the activity to the source, of weight "$-dln$". This causes a cycle of length zero. Violation of the deadline results in a positive length cycle. Hence provided that positive cycles are restricted, a feasible schedule is obtainable. This approach is transparent, classical and theoretically elegant. It is relatively easy to implement. To schedule activities the Bellman Ford algorithm is again required. The main downside of this approach is that many additional dummy activities could be created. The exact number of dummy jobs is difficult to determine, because the duration of the schedule may not be known with certainty. Hence an appropriate upper bound is needed.

3.2.6. Fixed activities and sequences

A number of decisions may already be fixed when planning and scheduling activities are activated. These fixed decisions affect all other decision making activities. The aforementioned constructs "FIXED_TIME", "FIXED_MACHINE" and "FIXED_SEQ" have been provided for those scenarios. The first construct fixes the start and end time of an activity. This construct causes a release time and a deadline to be added. In the disjunctive graph the release time is enforced by a conjunctive arc from the source to activity $o_{i,k}$ of weight $rlt_{i,k}$. The deadline is enforced by an arc from the activity to the source of weight $-dln_{i,k}$. These arcs allow other activities to be correctly sequenced before and after this fixed activity.

The second construct fixes the machine assignment for activity $o_{i,k}$. A Boolean flag is altered by this construct. That flag is used in a variety of solution techniques to differentiate between those activities that can or cannot be re-assigned to alternative machines. This construct can be used to remove machines from $\widetilde{M}_{i,k}$, the machine candidate list for $o_{i,k}$. Hence machine re-assignments will not be possible for that activity. The third construct is used to specify fixed sequences. Each activity in the fixed sequence is given a fixed position and is defined as having a fixed machine. In other words this construct also calls the second construct.

In practice the scheduling of surgeries in operating rooms is predominantly restricted by the MSS. Consequently patients of particular types can only be operated upon in specific rooms within specific intervals of time (i.e. blocks). If this assumption is relaxed then patient's surgeries can be scheduled anytime during business hours, Monday–Friday. In theory the situations shown in Table 6 could occur. This table describes what actions are required.

Table 6. Operating room possibilities and actions.

Scenario	OR sequence	Surgery timings	Action
i.	Given	Given	Apply construct FIXED_SEQ and FIXED_TIME
ii.	Given	Selectable	Apply construct FIXED_SEQ
iii.	Selectable	Selectable	none

If the patient is assigned to a specific block, but the order of the patients within the block is free, then the FIXED_TIME construct can be called with the block start and end time. In other words a release time and a deadline are introduced. Fixed sequences pose some unique difficulties for solution creation. These will be discussed in a later section.

4. Solution techniques

Solution techniques to solve the FJSP are proposed in this section. In Section 4.1 the assignment of machines to activities is separately considered. The joint sequencing and scheduling problem is then addressed in Section 4.2.

4.1. Static machine assignment problem (MAP)

As this FJSP problem is so difficult to solve, it is anticipated that a separate approach that predetermines where activities are to be performed, may be more beneficial and effective. This approach would be applied prior to sequencing and scheduling activities and would provide a better starting point for those activities. The following mixed integer programming (MIP) mathematical model is proposed:

$$Minimise \sum_{g \in G} H_g \tag{10}$$

Subject to:

$$\sum_{\forall o_{i,k} \in O_j^2} (a_{i,k,j} . \rho_{i,k,j}) \leq H_g - \tau_j - \overline{s}_j \, \forall g \in G, \forall j \in M_g; \text{ [Machine Utilisation]} \tag{11}$$

$$\sum_{\forall j \in \widetilde{M}_{i,k}} (a_{i,k,j}) = 1 \, \forall o_{i,k} \in O \text{ [Each activity is assigned a machine]} \tag{12}$$

$$a_{i,k,j} = 0 \forall (i,k) \in O, \forall j \in M | j \notin \widetilde{M}_{i,k} \text{ [Invalid assignment]} \tag{13}$$

$$a_{i,k,j} \in \{0,1\} \, \forall (i,k) \in O, \forall j \in \widetilde{M}_{i,k} \text{ [Binary decision]} \tag{14}$$

The notation used in this model has been defined earlier. This model balances the workload of machines within different groups and creates a starting solution whereby the horizon of each group, namely H_g, is minimal. This is accomplished via constraint (11). Each group generally refers to a type of activity, i.e. $g \equiv \phi$ and $G \equiv \Phi$. Constraint (12) ensures that each activity is assigned a single machine from the list of candidates. Some assignments are known to be invalid upfront and are “zeroed” in constraint (13). In this approach machines are grouped in order to identify a better balance. If groupings are not created, then some machines will unduly influence the solution. For example, patient length of stay in ward beds greatly overshadows the processing requirements elsewhere in the hospital. This model can be divided into $|G|$ sub-problems as the assignment to machines within groups is separate.

This model does not take into account a patients severity of illness and other stochastic factors that may affect this assignment in practice. Sequence dependent setups that occur between activities are also disregarded. This is because a sequence is not provided or even considered by this approach. The model can however be modified if a static sequence independent setup s_j is given. In that case the total time lost to setups is as follows:

$$\overline{s}_j = s_j \left(\sum_{\forall o_{i,k} \in O_j^2} a_{i,k,j} - 1 \right) \tag{15}$$

The summation term is the number of assigned activities, and the number of setups is one less than that number. If the first job activity also needs a setup then the minus one should be removed.

Although simple this assignment model has been found to be computationally intractable. Numerical investigations, for instance using OPL Studio, have shown that only small instances can be solved quickly.

Consequently a hybrid Simulated Annealing (HSA) approach is developed. Before the HSA is applied a fast heuristic is applied to provide a starting solution of high quality. Our fast heuristic iteratively assigns activities to machines. At each step, the list of candidates for the current activity is inspected and the machine that is least utilised is assigned. That machine's workload is then updated accordingly. This greedy heuristic provides high quality solutions with little if any computational overhead. That solution is then refined within HSA using two operators.

For the MAP a random swap perturbation has been found to be effective. In each step of the HSA, and for each group, the application of the swap operator is necessary. The swap mechanism changes the workload of two machines within a group. This is accomplished by randomly selecting two activities that are performed on those machines and reassigning them to the other machine. The workload of those machines is altered for benefit or detriment. The HSA cooling strategy dictates what level of detriment is permitted. An alternative strategy is to shift single activities. Numerical testing has shown that approach to be inferior.

After the swap perturbation operator has been applied, an improvement step should be applied to improve the convergence and solution quality. Our numerical testing indicates that an improvement operator is very effective and should be applied to each group. This novel feature separates our HSA from traditional versions of SA. The machines that are most and least utilised are selected. Their workloads are then decreased and increased respectively. There are many possible ways to rebalance the workload of those machines. Our approach is straight forward yet effective. A single activity is selected randomly from the machine with the greatest workload. That activity is transferred to the other machine if the target workload is not exceeded. The target workload is defined as the average workload of the two machines, i.e. $(wkld_1 + wkld_2)/2$. If the transfer is permitted, then the horizon of the group is decreased. This improvement operator is well suited to making small improvements, particularly after random perturbations have been made elsewhere. This algorithm is less effective when the difference between the most and least utilised machines is small and the activity processing times are larger than that difference.

4.2. Scheduling activities

This section describes our approach to scheduling the individual activities of each job. Constructive algorithms and meta-heuristics are utilised for that activity.

4.2.1. Constructing a solution

An iterative constructive insertion algorithm (INSALG) was first developed and provides a starting solution for our meta-heuristics. This algorithm inserts jobs iteratively. The activities within a job are inserted with respect to the magnitude of their processing time, i.e. from the largest to the smallest. This means that more influential activities are inserted and positioned first. Each activity is arbitrarily assigned to a machine (i.e. INSALG_RAND) or else assigned a machine via the solution of the MAP (i.e. INSALG_MAP). Each activity is trialled in each position of the current partial sequence for the assigned machine. This approach provides good quality "robust" solutions because many partial sequences of activities are evaluated (see Burdett & Kozan, 2010b). This approach is made effective by adding nodes and conjunctive arcs iteratively. Hence at early stages, the disjunctive graph is empty and graph evaluations take minimal computational effort.

A second type of insertion algorithm (INSAPP) was also implemented. In that approach an arbitrary sequence of jobs is again given. Although jobs and their activities are iteratively inserted, an analysis of alternative insertion positions is not performed. Consequently job activities are just appended. For each job activity, a list of the candidate machines that are free is collated. The first free candidate machine is selected and the activity is appended to that machines sequence. The INSAPP has attributes of a dispatching algorithm, as scheduling activities are performed over time. This approach is very fast, and is well suited to real life application and to problem instances of excessive size. This approach however is not a search algorithm per se. The INSAPP algorithm is also a greedy approach and will provide locally optimal solutions at best and may not provide a solution as robust as the one provided by the INSALG.

4.2.2. Meta-heuristics

To improve an existing solution different meta-heuristics were considered. As there is a need to repetitively evaluate the disjunctive graph using the Bellman Ford algorithm and that algorithm has high computational overheads, population based strategies such as Evolutionary Algorithms (EA) and Particle Swarm Optimisation (PSO) were avoided. Population based strategies inevitably incur higher CPU times as a population of solutions must typically be manipulated and evaluated (Burdett & Kozan, 2003). In contrast Simulated Annealing (SA), Tabu Search (TS) and other related methods do not. Preliminary numerical testing demonstrated that traditional SA is not effective on a problem like this, which has a more complex decision space. Insufficient progress was made and convergence was poor. A superior hybrid SA (HSA) was therefore created. It addresses some of the weaknesses that were observed. The perturbation operators summarised in Table 7 are an important and noteworthy feature of our HSA algorithm. They are needed to effectively search the decision space. The control parameters used in the HSA are otherwise the same as traditional SA. A traditional cooling policy was also adopted in the HSA.

Table 7. Perturbation strategies implemented and tested.

Name	Abrev.	Description
Operation Re-insertion	OPRINS	Re-sequence an arbitrarily selected activity within the current machine.
		Variants: random or greedy.
Operation Re-assignment	OPRASS	Reassign an activity to another machine and re-sequence it within that machine if possible.
		Variant: look at all candidate machines.
Job Re-insertion	JRINS	Re-sequence each activity of an arbitrarily selected job, within the currently assigned machines.
		Variant: assign each activity to a different machine.

The idea behind these strategies is the re-insertion of a selected activity and the evaluation of feasible insertion positions, to judge where the re-insertion should occur. In this application, moving activities within sequences is difficult due to the very strict timing conditions that exist. Hence only small changes are often feasible. If an activity is to be moved too far left or right, then other mechanisms like the job re-insertion are needed. The OPRASS involves the application of the OPRINS algorithm. The OPRINS can be sped up by reducing the range of the evaluated insertion positions, i.e. the neighbourhood size. The disjunctive graph is not evaluated as many times using this policy.

Numerical testing to date indicates the best strategy is a combination of OPRASS and JRINS. The JRINS is performed less often (i.e. say 30% of the time) as it is more “disruptive” and computationally arduous. Job re-insertions however are a form of compound move and are quite necessary for making larger changes in the solution. The perturbation operation is applied every iteration of the HSA approach and at some steps, both OPRASS and JRINS are applied.

A number of procedures are used within the OPRINS, OPRASS and JRINS algorithms. The most noteworthy are summarised in Table 8.

Table 8. Key procedures that facilitate OPRINS, OPRASS, JRINS.

Procedure	Description
make_insertion	This procedure inserts an un-inserted activity if it can. Evaluations are performed to compare different insertion possibilities. This procedure utilises “find_valid_position”.
find_valid_position	This procedure determines valid insertion positions for an activity.

Procedure	Description
	Output: a list of valid positions.
find_nearest_position	When re-inserting an activity on a different machine, this procedure finds the nearest position where insertion is most likely to feasibly occur.
evaluate_positions	This procedure determines valid insertion positions and evaluates their merit. Output: a sorted list of insertion positions.
test_insertion	This procedure tests the insertion of an activity in a given position of a given sequence. Output: feasibility status.
get_changes_insert	
get_changes_remove	These procedures determine which disjunctive arcs should be added and removed when an activity is inserted or removed from a sequence. Output: a list of added arcs and a list of removed arcs.
insert_operation	
uninsert_operation	These procedure insert and un-insert activities from a given position in a sequence. Output: the altered disjunctive graph and sequences.
evaluate	This procedure evaluates the current solution. The disjunctive graph is first evaluated using Bellman Ford. If valid, activities are then scheduled. The schedule is then evaluated and key performance indicators (kpi) are computed. Output: a schedule and the kpi's of the schedule.

4.3. Non standard features

Some features of this problem pose added difficulties for scheduling. A number of changes to core algorithms are required:

- *Fixed* activities*:* A reduced set of activities are available for selection in the OPRASS and OPRINS. Activities that have fixed machines must be excluded from the OPRASS and activities with fixed positions must be excluded from the OPRINS. When applying the JRINS algorithm, activities that have fixed positions must be bypassed.
- *Same machine assignments*: The presence of SMA means that some activity cannot be re-assigned independently. Multiple activities must be re-sequenced together. In actuality, this amounts to an iterative re-insertion according to an arbitrary ordering of those activities.
- *INSALG:* The *INSALG* is used to build sequences that have not been predefined. If some sequences are per chance given (i.e. there are fixed sequences), a modified INSALG is needed. The precedence relations implied by those fixed sequences must be initialised at some point in the construction process. To evaluate fixed sequences, the associated activity nodes must be present in the disjunctive graph. Inevitably this may involve other activities of the job, and these are not performed on the machine with the fixed sequence. If those activities are not present, many of the disjunctive arcs required to enforce the fixed sequence cannot be added. One solution is to insert un-sequenced activities iteratively. An alternative is to add all nodes and conjunctive arcs upfront. The computational overhead is significantly increased as the full graph must always be evaluated. Another approach is to add pre-sequenced components iteratively as well. Pre-sequenced activities must be inserted according to the fixed sequences. In theory this is easy to do, i.e. by iterating through the fixed sequence, only taking into account those activities that have been inserted thus far.

5. Case study and numerical investigations

This section evaluates the effectiveness of our approaches on a large suite of realistic sized test problems. All numerical experiments have been run on a quad core Dell personal computer (PC) with a 2.6 gigahertz processor and 16 gigabyte memory under Windows 7. The FJSS approach has been coded in C++ for added speed and robustness. The main procedures constitute about 5000 lines of code. All of the constructs defined in Table 3, except the buffering, have been used in our test problems. Our test problems can be found at QUT Eprints and QUT Data Finder.

5.1. Arbitrary benchmarks

The first set of problem instances has been randomly generated. In total there are 24 test problems (Ex 1 –24). The number of patient jobs was selected as 50, 100, 150 or 200. Each patient job is given four activities, namely (POC, SUR, PAC, REC). The following scenarios were selected and describe the number of candidate machines available to each job activity: {[5,2,5,10], [5,3,5,10], [5,4,5,20], [5,5,5,20], [5,10,5,20], [5,10,5,30]}. The number of machines in each test problem is 22, 23, 34, 35, 40, and 50. The number of machines is dictated by the number of candidate machines available to each job activity. For instance, the sum of the candidates gives the total number of machines.

In these test problems, start time restrictions are placed on the first two activities of each job, namely a business time requirement. The processing times of each activity are in minutes. They have been arbitrarily selected for POC, SUR, PAC and REC activities from the following ranges respectively: {[15,60], [60,300], [30,300], [1440,7200]}. These ranges are indicative of typical POC, SUR, PAC and REC activities in hospitals. We assume that processing times do not depend on the bed-space and operating room assigned. Post setups and transfers of 15 minutes have been imposed.

The optimal makespan is not known for these instances. The following lower bounds are hence derived to provide an idea:

$$\begin{aligned} LB &= \max_{\phi\in\Phi}(lb_\phi) \text{ where } lb_\phi = \frac{\sum_{\forall i\in J}\rho_{i,\phi}}{|M_\phi|\,.\,T_\phi} \text{ and} \\ M_\phi &= \{j\in M | fn_j = \phi\} \end{aligned} \tag{16}$$

A lower bound is computed for each of the four activities (i.e. $\Phi = \{\text{poc}, \text{sur}, \text{pac}, \text{rec}\}$). It is computed as the sum of the processing times divided by the number of machines available to perform the activity. That value is then divided by T_ϕ the number of hours that task ϕ may be performed per day. Recovery tasks typically occur 24 hours per day but surgery and pre and post-operative care can only occur for 8 hours per day. The lower bound provides the number of days of processing required. The activity with the greatest processing requirement is then used as an approximation of the makespan. As the recovery tasks far outweigh the other activities the lower bound can be approximated by lb_{rec}.

The application of the constructive algorithms is shown in Tables 9–11. Each test problem is solved 30 times to understand the expected behaviour of the constructive algorithms. The CPU column refers to the average time. The timing restrictions were handled using the *RSCSP*.

Table 9. Solutions produced by the constructive algorithms for *FJ*|*No STR*|*Cmax*.

#,|*J*|,|*M*|	**LB**	**INSAPP [No STR]**		**INSALG_RAND [No STR]**			**INSALG_MAP [No STR]**		
		avg, [min,max] (days)	**Stdev (days)**	**avg, [min,max] (days)**	**Stdev (days)**	**CPU (seconds)**	**avg, [min,max] (days)**	**Stdev (days)**	**CPU (seconds)**
1,50,22	14.82	19.12, [17.86,19.8]	0.51	28.94, [24.3,34.61]	3.14	0	20.1, [18.82,22.14]	0.93	0.07
2100,22	32.24	37.82, [36.43,39.04]	0.66	55.56, [44.7,72.11]	5.92	14.03	40.41, [38.48,43.02]	1.14	14.23
3150,22	44.29	51.52, [50.49,52.88]	0.59	75.53, [63.03,92.83]	6.72	81.2	56.11, [53.38,62.6]	1.89	82.3
4200,22	60.81		0.68		7.54	268.03		1.43	270.83

#,|*J*|,|*M*|	**LB**	**INSAPP [No STR]**		**INSALG_RAND [No STR]**			**INSALG_MAP [No STR]**		
		avg, [min,max] (days)	**Stdev (days)**	**avg, [min,max] (days)**	**Stdev (days)**	**CPU (seconds)**	**avg, [min,max] (days)**	**Stdev (days)**	**CPU (seconds)**
		69.69, [68.34,70.75]		95.22, [83.81,116.11]			74.37, [71.91,78.2]		
5,50,23	15.37	19.07, [18.08,20.2]	0.54	31.36, [23.68,45.58]	4.82	0	20.08, [18.85,21.93]	0.85	0
6100,23	29.89	35.33, [34.49,36.5]	0.5	51.29, [44.33,69.63]	5.62	11.47	38.3, [36.46,40.66]	1.29	11.3
7150,23	44.18	51.14, [50.08,52.2]	0.57	75.69, [64.39,103.35]	9.25	65.5	54.51, [51.97,56.55]	1.14	65.5
8200,23	61.27	70.03, [68.73,71.37]	0.56	95.23, [82.53,129.25]	8.52	223.2	74.81, [72.5,78.52]	1.15	216.73
9,50,34	6.79	10.18, [9.38,10.93]	0.48	20.41, [15.43,33.58]	4.2	0	10.17, [8.71,11.84]	0.79	0
10,100,34	14.99	19.11, [18.18,20]	0.44	35.27,[28.89,50]	4.67	9.43	21.58, [19.76,24.2]	1.21	9.17
11,15,34	22.09	27.11, [25.98,28.19]	0.55	46.47, [36.69,60.13]	5.51	56.47	29.85, [28.06,32.24]	1.1	56.53
12,200,34	30.57	36.37, [35.4,37.16]	0.47	59.63, [50.68,69.82]	5.15	196.27	41.2, [38.69,43.93]	1.29	188
13,50,35	7.34	10.67, [9.91,11.54]	0.42	20.23, [15.87,25.65]	2.73	0	10.93, [9.23,13.32]	1.19	0
14,100,35	15.41	19.78, [19.05,20.85]	0.46	36.14, [27.25,51.09]	5.11	8.83	22.35, [20.13,26.73]	1.5	8.47
15,150,35	23.31	28.28, [27.44,29.17]	0.44	48.67, [41.39,57.94]	4.42	52.03	32.25, [29.97,35.81]	1.36	51.1
16,200,35	32.23	38, [37.22,38.79]	0.37	62.81, [51.36,76.43]	6.32	175.47	43.44, [39.13,47.68]	1.66	174.43
17,50,40	8.23	11.87, [11.04,12.87]	0.46	22.48, [17.94,30.41]	3.11	0	11.91, [9.87,14.53]	1.13	0
18,100,40	15.46	19.88, [19.05,20.96]	0.55	35.27, [26.06,41.72]	4.05	6.73	22.31, [19.79,25.93]	1.5	6.47
19,150,40	23.91	28.88, [28.08,29.48]	0.43	50.64, [39.26,63.43]	5.86	42.17	32.97, [30.86,38]	1.49	40.53
20,200,40	30.48	36.14, [35.32,36.98]	0.51	57.97, [47.09,72.78]	5.38	144.13	40.26, [38.71,43.25]	1.27	144.63
21,50,50	5.04	8.79,[7.9,9.44]	0.44	19.45, [12.4,26.29]	3.73	0	7.49,[6.76,9.85]	0.76	0
22,100,50	10.02	13.93, [13.14,14.55]	0.38	28.74, [24.35,35.99]	3.05	6.57	15.42, [13.71,17.9]	1.14	5.93
23,150,50	14.18	18.71, [17.77,19.51]	0.42	36,[27.75,52.19]	5.37	42	21.46, [19.48,25.28]	1.61	39.73

#,|*J*|,|*M*|	LB	INSAPP [No STR]		INSALG_RAND [No STR]			INSALG_MAP [No STR]		
		avg, [min,max] (days)	Stdev (days)	avg, [min,max] (days)	Stdev (days)	CPU (seconds)	avg, [min,max] (days)	Stdev (days)	CPU (seconds)
24,200,50	19.27	24.14, [23.34,25.15]	0.43	43.78, [36.35,54.47]	4.58	144.77	27.3, [25.14,29.47]	1.2	141.5

Table 10. Solutions produced by the constructive algorithms for *FJ*\|*STR*\|*Cmax*.

#,|*J*|,|*M*|	INSAPP [STR]		INSALG_MAP [STR]		
	avg, [min,max] (days)	Stdev (days)	avg, [min,max] (days)	Stdev (days)	CPU (minutes)
1,50,22	23.91,[20.64,26.49]	1.64	27.61,[23.83,33.97]	3.32	0.048
2100,22	47.57,[46.01,48.94]	0.61	58.42,[47.09,71.22]	5.87	1.368
3150,22	64.87,[61.75,68.44]	2.11	88.23,[68.38,102.45]	8.02	9.848
4200,22	88.69,[85.12,91.18]	1.59	119.26,[109.67,134.4]	6	39.497
5,50,23	23.25,[20.66,26.26]	1.36	26.26,[23.05,32.66]	2.09	0.033
6100,23	42.39,[40.58,45.35]	1.14	50.84,[45.11,59.51]	4.06	0.93
7150,23	62.04,[59.64,64.6]	1.24	80.11,[68.37,89.33]	6.44	6.871
8200,23	69.74,[68.54,70.93]	0.65	74.99,[70.95,79.35]	1.78	3.573
9,50,34	12.68,[11.82,13.44]	0.42	14.64,[11.99,19.59]	2.15	0.017
10,100,34	23.57,[21.41,26.65]	1.35	36.16,[28.08,45.7]	4.83	0.752
11,15,34	33.87,[32.6,34.75]	0.54	60.38,[46.16,75.5]	6.57	5.731
12,200,34	46.55,[43.34,47.74]	1.3	78,[65.73,91.09]	7.38	25.971
13,50,35	12.76,[12.37,13.57]	0.27	15.49,[11.8,19.85]	2.22	0.016
14,100,35	23.99,[22.25,26.58]	1.41	36.12,[28.48,52.66]	4.78	0.671
15,150,35	34.02,[32.96,35.57]	0.61	53.48,[40.13,67.36]	6.88	4.686
16,200,35	47.51,[46.78,49.3]	0.49	78.39,[60.84,99.69]	7.87	23.151
17,50,40	13.14,[12.49,13.8]	0.42	14.79,[12.55,19.45]	2.12	0
18,100,40	24.11,[21.33,26.75]	1.41	33.22,[26.08,41.87]	4.49	0.469
19,150,40	35.04,[33.56,37.34]	0.83	52.37,[40.22,67.27]	6.27	3.947
20,200,40	44.69,[43.05,47.57]	1.46	70.79,[53.59,87.48]	7.11	18.911
21,50,50	10.35,[8.81,12.58]	1.23	13.16,[10.8,17.8]	2.01	0
22,100,50	18.33,[15.82,19.58]	1.05	26.27,[19.24,32.94]	3.68	0.378
23,150,50	23.26,[22.45,25.71]	0.82	41.88,[35.59,54.13]	4.76	3.202
24,200,50	30.12,[28.31,33.42]	0.94	58.46,[47.53,68.25]	5.38	17.193

Table 11. Solutions produced by the constructive algorithms for *FJ*\|*OR FXSEQ*\|*Cmax*.

#,|*J*|,|*M*|	INSALG_MAP [No STR]			INSALG_MAP [STR]		
	avg, [min,max] (days)	Stdev (days)	CPU (minutes)	avg, [min,max] (days)	Stdev (days)	CPU (minutes)
1,50,22	38.235,[31.71,45.244]	4.344	0	48.61,[41.04,59.3]	5.28	0.15
2100,22	88.373,[73.419,99.363]	8.44	0.215	105.79,[88.3123.14]	7.98	4.442
3150,22	128.365,[114.326,142.894]	8.158	1.192	151.55,[132.28,168.86]	9	25.932
4200,22	170.955,[159.704,187.468]	8.361	4.053	212.1,[188.32,235.05]	14.53	92.395
5,50,23	33.895,[25.959,44.012]	5.609	0	42.02,[32.08,58.97]	5.54	0.128
6100,23	75.956,[68.116,81.061]	4.38	0.192	90,[76.23,106.23]	6.75	3.748
7150,23	114.576,[104.379,125.299]	6.378	1.165	131.93,[120.62,151.62]	9.04	25.01
8200,23	158.19,[143.172,180.09]	10.983	3.898	154.13,[146.56,165.01]	6.18	3.738
9,50,34	20.383,[18.342,23.837]	1.646	0	24.92,[21.32,28.32]	1.97	0.096
10,100,34	44.683,[40.471,54.283]	3.848	0.157	56.83,[41.61,69.61]	6.17	2.91
11,15,34	66.906,[59.453,71.779]	3.506	1.043	81.78,[69.83,95.35]	7.47	19.535
12,200,34	94.161,[80.324,117.402]	10.29	3.152	114.95,[106.86,124.86]	6.06	70.74
13,50,35	16.975,[14.649,20.611]	1.91	0	22.93,[18.73,28.1]	2.71	0.075
14,100,35	43.222,[39.697,46.549]	2.037	0.152	54.51,[46.56,68.85]	5.26	2.718
15,150,35	67.686,[58.088,79.495]	6.61	0.98	81.41,[75.76,89.05]	4.05	18.073
16,200,35	93.445,[82.722,101.81]	6.714	3.292	112.24,[97.66,124.22]	7.66	67.408
17,50,40	16.508,[13.797,19.405]	1.822	0	19.48,[15.45,24.5]	1.91	0.046
18,100,40	39.889,[35.546,45.315]	3.151	0.135	44,[37.38,54.73]	4.33	2.242
19,150,40	54.375,[49.013,57.761]	2.332	0.927	67.81,[60.46,75.68]	5.23	15.495
20,200,40	73.546,[68.512,77.174]	3.075	3.118	86.76,[81.56,90.08]	2.7	59.072
21,50,50	11.309,[8.968,15.039]	1.678	0	13.88,[10.4,16.47]	1.77	0.037
22,100,50	26.379,[24.041,28.186]	1.263	0.128	32.45,[26.79,39.29]	3.02	1.958
23,150,50	39.862,[33.573,45.449]	3.552	0.857	48.17,[42.2,52.69]	3.68	14.687
24,200,50	53.184,[49.063,60.443]	3.442	2.928	65.65,[55.2,74.86]	5.12	55.208

In summary the INSAPP was clearly the faster of the two constructive algorithms. It completed almost immediately on these test problems. As expected the INSALG time requirements were much larger but still acceptable. For instance, on the time unrestricted cases (i.e. FJ/No STR/Cmax), the CPU time did not exceed 5 minutes. On the time restricted cases (i.e. FJ/STR/Cmax) the time required on problems with less than 200 jobs, was 10 minutes or less. Only on the larger 200 job problems did the CPU times increase greatly. They varied from 20 to 40 minutes. It is anticipated that the time requirements of the INSALG do not cause any great issues for practical application, but improvements that reduce those times are possible and desirable.

Good quality solutions were obtained by both algorithms on the whole. On the time restricted case that is difficult to prove, however on the time unrestricted cases, the solutions are evidently quite close to the lower bound approximations. The INSAPP was clearly superior to the INSALG; at best the INSALG is competitive. Its performance however deteriorates as the number of jobs increases. This is particularly noticeable on the time restricted cases. The INSALG was less able to create good solutions when there are 150 or 200 jobs. For cases with fewer jobs, it performed better.

The random assignment of machines to activities resulted in particularly poor INSALG solutions. The creation of a balanced assignment first greatly improved the solution quality. The solutions obtained then were quite comparable to the INSAPP. Hence the application of the INSALG_RAND was disregarded in later numerical experiments. In contrast the INSAPP automatically balances the machine utilisation, because the first free machine is always selected. That approach mimics real life practices. The order in which jobs are inserted is arbitrary and greatly affects the solution quality. The performance of the INSAPP and INSALG can therefore be quite variable. The INSAPP however has less variation of the two approaches.

As the processing time of the recovery activity is so much larger, the scheduling of the other job activities will have minimal impact on the solution where there are no timing restrictions. Hence those activities could in theory be ignored and scheduled later.

The HSA was applied next. The best of the INSAPP and INSALG solutions were used as starting solutions. The results are summarised in Tables 12 and 13. The HSA was applied with an initial temperature of 1, final temperature of 0.01, temperature reduction 0.9, and 200 evaluations per temperature. The lower bound in (16) is shown and used to judge the quality of No STR solutions. The No STR solutions are used as a lower bond to judge the quality of the STR solutions.

Table 12. Solutions produced by HSA for *FJ*|*No STR*|*Cmax* and *FJ*|*STR*|*Cmax*.

#,|*J*|,|*M*|	**LB**	**START: INSAPP [No STR]**			**START: INSALG_MAP [No STR]**			**START: INSAPP [STR]**			**START: INSALG_MAP [STR]**		
		CMAX (days)	**IMPR (days)**	**CPU (hours)**	**CMAX (days)**	**IMPR (days)**	**CPU (hours)**	**CMAX (days)**	**IMPR (days)**	**CPU (hours)**	**CMAX (days)**	**IMPR (days)**	**CPU (hours)**
1,50,22	14.82	17.378	0.478	0.048	*16.795	2.02	0.048	19.629	1.008	0.252	19.89	3.936	0.25
2100,22	32.24	36.042	0.384	0.506	36.302	2.178	0.5	42.501	3.509	2.609	*42.188	4.906	2.62
3150,22	44.29	50.204	0.289	1.745	*50.095	3.287	1.909	59.331	2.417	9.949	60.939	7.442	9.11
4200,22	60.81	68.134	0.206	4.475	69.155	2.758	4.473	79.46	5.655	32.968	82.388	27.279	30.6
5,50,23	15.37	18.084	0	0.042	*17.407	1.442	0.038	20.008	0.655	0.23	20.219	2.831	0.23
6100,23	29.89	34.49	0	0.446	34.588	1.872	0.426	38.594	1.983	2.094	39.919	5.194	2.25
7150,23	44.18	50.081	0	1.689	*49.452	2.515	1.591	56.852	2.783	8.188	60.278	8.089	9.07
8200,23	61.27	68.73	0	3.333	69.589	2.912	3.768	68.405	0.133	3.883	69.388	1.562	3.77
9,50,34	6.79	8.542	0.835	0.034	*8.169	0.544	0.037	10.928	0.89	0.157	*10.055	1.937	0.16
10,100,34	14.99	17.535	0.649	0.388	18.428	1.33	0.402	20.521	0.888	1.633	21.985	6.092	1.7
11,15,34	22.09	25.953	0.025	1.453	26.188	1.872	1.477	32.078	0.522	7.521	33.082	13.083	6.8
12,200,34	30.57	35.126	0.276	3.565	37.117	1.574	3.866	42.344	1.0	21.726	45.938	19.793	21.24
13,50,35	7.34	9.297	0.612	0.033	*9.076	0.151	0.034	11.797	0.574	0.144	*11.651	0.146	0.14
14,100,35	15.41	18.834	0.221	0.366	*18.716	1.417	0.378	21.41	0.842	1.661	21.807	6.671	1.6
15,150,35	23.31	27.111	0.33	1.33	27.527	2.442	1.378	32.741	0.215	5.757	33.13	7	6.67
16,200,35	32.23	37.016	0.205	3.443	37.488	1.646	3.559	44.425	2.359	18.347	46.683	14.16	19.34

#,|*J*|,|*M*|	LB	START: INSAPP [No STR]			START: INSALG_MAP [No STR]			START: INSAPP [STR]			START: INSALG_MAP [STR]		
		CMAX days)	IMPR (days)	CPU (hours)	CMAX (days)	IMPR (days)	CPU (hours)	CMAX (days)	IMPR (days)	CPU (hours)	CMAX (days)	IMPR (days)	CPU (hours)
17,50,40	8.23	10.061	0.981	0.027	*9.789	0.085	0.03	12.289	0.2	0.118	*11.732	0.815	0.12
18,100,40	15.46	18.222	0.828	0.334	18.515	1.278	0.336	20.469	0.86	1.292	21.117	4.966	1.31
19,150,40	23.91	27.666	0.412	1.236	29.449	1.407	1.207	33.406	0.156	5.567	33.651	6.568	5.75
20,200,40	30.48	35.292	0.024	3.093	35.619	3.086	3.063	41.867	1.183	16.805	46.492	7.098	16.64
21,50,50	5.04	6.758	1.138	0.033	*6.539	0.218	0.029	7.883	0.931	0.101	8.488	2.311	0.1
22,100,50	10.02	12.91	0.229	0.339	*12.313	1.399	0.323	15.147	0.672	1.264	15.692	3.55	1.25
23,150,50	14.18	17.306	0.462	1.21	17.72	1.765	1.215	22.116	0.338	5.049	23.403	12.187	4.98
24,200,50	19.27	23.015	0.324	3.021	23.259	1.88	3.048	27.363	0.951	16.161	32.453	15.076	15.59

Table 13. Solutions produced by HSA for *FJ|OR FXSEQ|Cmax*.

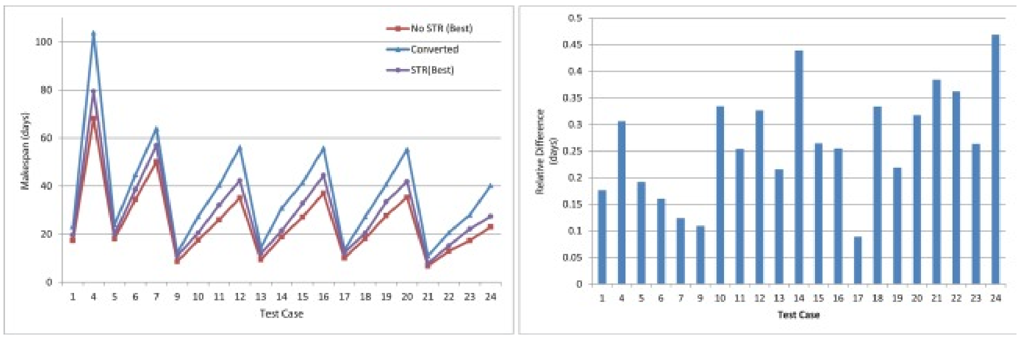
#,|*J*|,|*M*|	START: INSALG_MAP [No STR]				START: INSALG_MAP [STR]			
	INSALG (days)	CMAX (days)	IMPR (days)	CPU (hours)	INSALG (days)	CMAX (days)	IMPR (days)	CPU (hours)
1,50,22	31.71	19.33	12.38	0.04	41.04	24.24	16.8	0.62
2100,22	73.419	41.37	32.05	0.34	88.3	68.14	20.16	7.81
3150,22	114.326	58.68	55.64	1.16	132.28	104.98	27.3	16.69
4200,22	159.704	79.96	79.75	2.73	200.32	141.32	59	63.49
5,50,23	25.959	18.75	7.21	0.04	32.08	24.27	7.81	0.58
6100,23	68.116	39.17	28.95	0.34	76.23	57.23	19	6.3
7150,23	104.379	57.05	47.33	1.21	120.62	97.62	23	23.91
8200,23	143.172	77.83	65.35	2.84	146.56	83.26	63.3	2.16
9,50,34	18.342	10.27	8.07	0.04	21.32	12.51	8.81	0.42
10,100,34	40.471	20.52	19.95	0.37	41.61	33.4	8.21	5.15
11,15,34	59.453	30.65	28.8	1.23	69.83	56.41	13.42	22.13
12,200,34	80.324	48.29	32.04	1.27	113.2	103.2	106.86	31.93
13,50,35	14.649	10.16	4.49	0.03	18.73	12.57	6.16	0.43
14,100,35	39.697	20.5	19.2	0.39	46.56	34.68	11.88	4.77
15,150,35	58.088	32.86	25.23	0.53	75.76	56.76	19	21.05
16,200,35	82.722	49.27	33.46	1.27	97.66	82.31	15.35	51.05
17,50,40	13.797	10.45	3.35	0.03	15.45	12.62	2.84	0.29
18,100,40	35.546	19.23	16.31	0.33	37.38	28.45	8.93	4.04
19,150,40	49.013	32.42	16.59	0.58	60.46	46.14	14.32	19.12

#,|*J*|,|*M*|	**START: INSALG_MAP [No STR]**				**START: INSALG_MAP [STR]**			
	INSALG (days)	**CMAX (days)**	**IMPR (days)**	**CPU (hours)**	**INSALG (days)**	**CMAX (days)**	**IMPR (days)**	**CPU (hours)**
20,200,40	68.512	43.1	25.41	1.28	81.56	61.08	20.48	52.1
21,50,50	8.968	7.12	1.85	0.03	10.4	8.32	2.08	0.24
22,100,50	24.041	13.48	10.56	0.35	26.79	19.64	7.15	3.57
23,150,50	33.573	21.68	11.89	0.58	42.2	34.09	8.1	17.92
24,200,50	49.063	29.42	19.65	1.46	55.2	47.2	8	46.04

On average the HSA can improve the INSAPP starting solutions by 0.37 days, yet it can improve the INSALG solutions by 1.71 days. It is interesting that the HSA was able to find better solutions on many occasions if the starting solution was provided by the INSALG and not the INSAPP. These are indicated by a * in Table 12. A number of starting solutions provided by the INSAPP could also not be improved upon at all by the HSA. It was known that these INSAPP solutions were not optimal. We believe that this occurs because the INSAPP provides solutions that are more highly compressed. They are quite sensitive to change and evidently become less refinable. A robust but less efficient schedule however can often be improved more easily. Also, the INSAPP solutions are in some sense locally optimal. To backtrack out of that part of the decision space to another is more difficult. In contrast the INSALG produces solution with more robust underlying features. In all likelihood they are closer to the region of the global optimum.

In summary the CPU time requirements for the HSA ranged from several minutes to 2 days. The CPU requirements are much larger when the number of jobs is 150 and 200. The addition of the time restrictions, again inflate the CPU time and take significantly more time to solve. The HSA CPU time was quite prohibitive because the evaluation procedure requires potentially as many iterations as there are nodes.

The conversion of unrestricted solutions to restricted solutions was investigated as a means of reducing the computational overhead. The idea is that a simpler and faster solution approach can be used in practice, if a good solution of the *FJ*|*No STR*|*Cmax* problem is also a good solution of the *FJ*|*STR*|*Cmax* problem. The results are summarised in Fig. 7. They indicate that *FJ*|*No STR*|*Cmax* solutions when converted to *FJ*|*STR*|*Cmax* solutions are of reasonable quality. However dedicated *FJ*|*STR*|*Cmax* solutions are still significantly better. The *FJ*|*STR*|*Cmax* solutions are on average 26.6% better than the converted *FJ*|*No STR*|*Cmax* solutions. The min and max relative differences are [0.089, 0.47] with a standard deviation of 0.104.



Download high-res image (258KB) Download full-size image

Fig. 7. A comparison of converted FJ/No STR solutions to existing FJ/STR solutions.

In the aforementioned numerical investigations timing restrictions were enforced by the RSCSP. The handling of occupancy restrictions via the definition of dummy jobs and dummy activities with strict release times and deadlines was however also investigated. In that scenario the INSAPP cannot be used because it just appends activities and cannot evaluate alternative insertion positions. Hence solutions must be constructed using the INSALG. For Ex 1 the application of the INSALG resulted in a solution with a makespan of 51.416 days. To obtain a feasible starting solution it was necessary to append some jobs to the back of the schedule. We found that to insert them directly within the existing schedule, when there are fixed activities, is too intricate for the INSALG. Consequently it may be better to add those types of dummy jobs, after an initial feasible schedule has been determined in other ways. The HSA was also applied. On Ex 1 it took 693 seconds (0.1925 hours) in comparison to the 900 seconds (i.e. 0.25 hours) previously taken by the HSA with initial solution produced INSALG and occupancy restrictions not modelled by dummy jobs and dummy operations. This improvement (i.e. a reduction of 300 seconds) is quite considerable. Evidently the computational overhead of using node functions to correct timing violations is greater than the addition of dummy jobs. Unfortunately the solution quality was not as good. For instance the makespan was 23.4 days in comparison to 19.89 days obtained previously. The reason for this is twofold. As the starting solution was so much larger than previous starting solutions, it was unlikely that the final solution would be as good, and more computational effort would need to be expended to reach the same quality of solution. The second reason is that when there is fixed activities, it is very difficult to perturb the schedule without making it infeasible. A simple perturbation for instance easily causes excessive blocking. This is not allowed in hospital applications. In essence the HSA performs badly because multiple changes are needed to improve the solution. A single alteration only makes the solution worse. In conclusion the development of strategies that perform compound moves seems to be necessary. In retrospection it is also evident that the right shift correction is effective because perturbations that cause multiple timing violations can still be made. The node functions in the disjunctive graph automatically correct these violations and return the schedule to feasibility. In contrast, this does not occur when there are fixed activities for invalid time periods. The non-fixed activities have to be intricately moved around the fixed activities to maintain schedule feasibility. This restricts the effectiveness of the HSA or any other meta-heuristic scheme.

When patient admissions and surgeries are pre-planned, then there are many fixed activities in the problem. It is beneficial to simplify the problem in this situation. For instance the fixed surgery activities should be removed and only the post-op and recovery activity should be included in the problem. The post-op activity can then be given a release time equal to the designated (i.e. predicted) end of surgery time.

A visual inspection of the Gantt charts of the associated schedules shows some inherent properties of good and bad schedules. For instance, any scheduling decisions that result in the discharge of patients on weekends, also results in an unoccupied bed for the remainder of the weekend. As operating rooms are closed (except for some emergency surgeries), those aforementioned bed-spaces cannot be refilled until at least Monday lunchtime, and remain un-utilised. The best solutions to the FJSP make sure patients are discharged after the weekend period or if necessary late in the weekend. Evidently how patients are scheduled around the weekend period is very important if patient throughput is to be maximised. From an economic perspective an unutilised bed is not necessarily bad. The bed-space can be formally closed, in which case it does not require nursing staff to be in attendance. Hence ward operating costs could be reduced.

5.2. Hospital case study

The case study presented here is based upon a large metropolitan, tertiary referral and university and college affiliated teaching hospital in Brisbane, Qld, Australia. It provides acute and elective adult medical and surgical care and emergency department services. This hospital has 600–800 beds distributed over more than 30 wards. It has 20 operating rooms, the majority running only during business hours (i.e. 8 am–5 pm). There are 18 surgical specialities (i.e. surgical units). The average treatment times for different surgical specialties are shown in Table 14. These values were obtained from ORMIS and HBCIS records.

Table 14. Average treatment times (in hours). Note: Standard deviations are in brackets.

Unit	POC	OP	PAC	REC
1	3.12	1.65 [1.1]	4.81 [3.287]	34.57 [72.075]
2	2.6	1.92 [0.929]	4.45 [2.553]	19.19 [40.819]
3	2.02	3.44 [2.805]	4.97 [3.079]	44.18 [93.272]
4	1.02	4.1 [1.478]	5.07 [1.992]	33.92 [54.441]
5	2.66	2.94 [3.499]	4.94 [3.397]	22.51 [313.503]
6	2.55	1.7 [1.375]	5.01 [2.796]	15.19 [31.872]
7	0.27	1.1 [0.72]	3.98 [1.927]	26.69 [100.133]
8	2.69	3.12 [1.972]	4.93 [2.906]	44.96 [98.199]
9	2.32	1.04 [0.709]	5.93 [3.274]	46.16 [97.517]
10	2.02	2.87 [1.816]	4.86 [2.893]	45.47 [87.398]
11	2.66	0.74 [0.537]	4.86 [1.596]	14.79 [140.785]
12	2.63	2.21 [1.494]	5.58 [3.402]	41.6 [100.85]
13	2.73	1.38 [1.34]	5.11 [2.345]	32.98 [101.352]
14	2.26	1.74 [1.1]	4.4 [2.752]	29.46 [62.003]
15	0.12	4.19 [1.93]	6.83 [5.314]	33.12 [68.624]
16	2.47	2.1 [1.817]	4.73 [2.475]	30.16 [64.598]
17	2.38	1.48 [1.357]	4.43 [2.487]	21.01 [51.926]
18	2.29	2.61 [1.337]	4.58 [2.512]	48.04 [103.611]

To demonstrate our approach a scenario with 129 resources and 180 patients was chosen. The 180 patients are divided equally across the 18 surgical units. In other words there are ten patients per surgical specialty: J1–J10 are specialty 1, J11–J20 are specialty 2,…., J171–J180 are specialty 18. The standard deviations in Table 14 have been used to generate the different activity processing times for each job. In our test instance there are two surgical care areas for pre and post-operative care; each has 20 beds. There are 14 recovery wards (i.e. W1–W14). As each ward has 20 plus beds, it is evident that there is excessive capacity for our test instance. Hence it is appropriate to reduce the number of beds in each ward. We have chosen five beds per ward. We have considered 19 of the 20 OR's. One OR is set aside for emergencies. The surgical units that use each of the recovery wards is as follows:

W1 (18), W2 (5, 6, 13), W3 (12, 15), W4 (12), W5 (10), W6 (4), W7 (4), W8 (17), W9 (9, 14), W10 (1, 8), W11 (1, 2, 13, 16), W12 (3), W13 (11), W14 (7).

For the OR FXSEQ scenario, patients from each surgical specialty are processed on a single OR in the order that they are defined. Hence: J1–J10 on OR1, J11–J20 on OR2,…, J171–J180 on OR18. Patient activities on other machines can be selected from the candidate list and are not fixed.

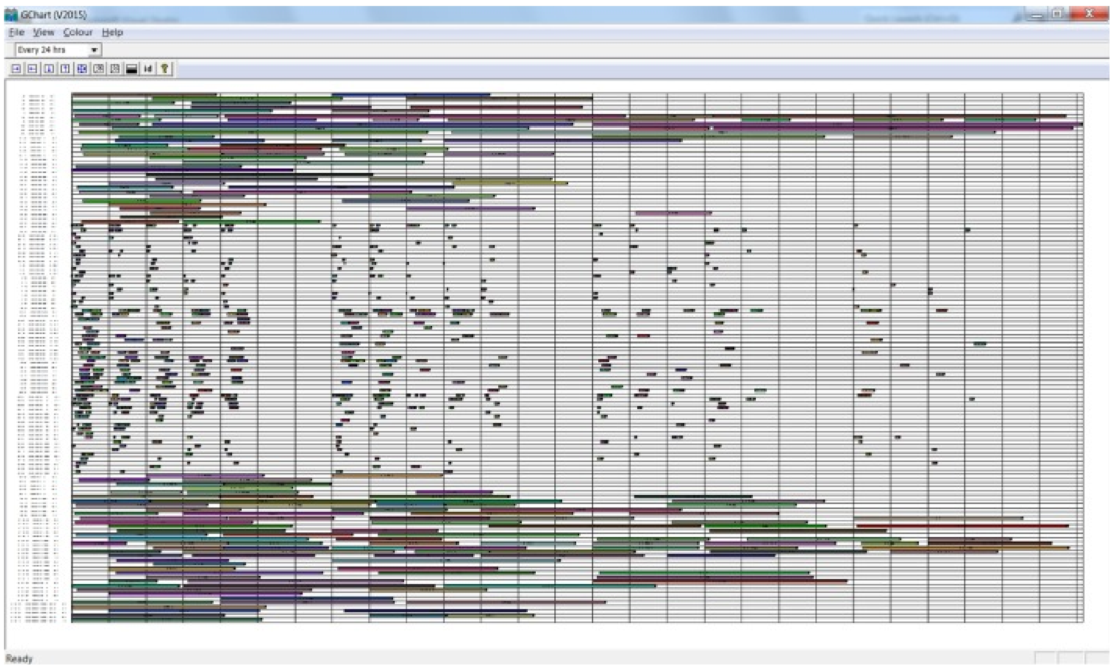
Our numerical investigation here is similar to that performed in Section 5.1. The results are summarised in Table 15. The INSAPP is not appropriate when there are fixed sequences. Hence “not applied” is shown in Table 15 in some of the cells.

Table 15. Solutions of FJ/No OR FXSEQ/Cmax and FJ|OR FXSEQ|Cmax.

STR	OR FXSEQ	INSAPP	HSA		INSALG_MAP	HSA	
		avg,[min,max], stdev, cpu	Cmax (days)	CPU (min)	avg,[min,max], stdev, cpu	Cmax (days)	CPU (min)
No	No	30.7,[25.91,35.12],2.18,0	25.77	45.9	25.56,[24.33,34.5],1.825,12.2	24.13	43.87
Yes	No	33.88, [28.91,38.91],2.75,0	27.47	90.57	40.59, [32,95,51.86],5.08,40.23	30.47	103.57
No	Yes	Not applied			45.42,[33.39,58.05],7.88,15.7	33.02	44.27
Yes	Yes	Not applied			52.62, [40.17,74.27],8.02,43.53	34.47	122.22

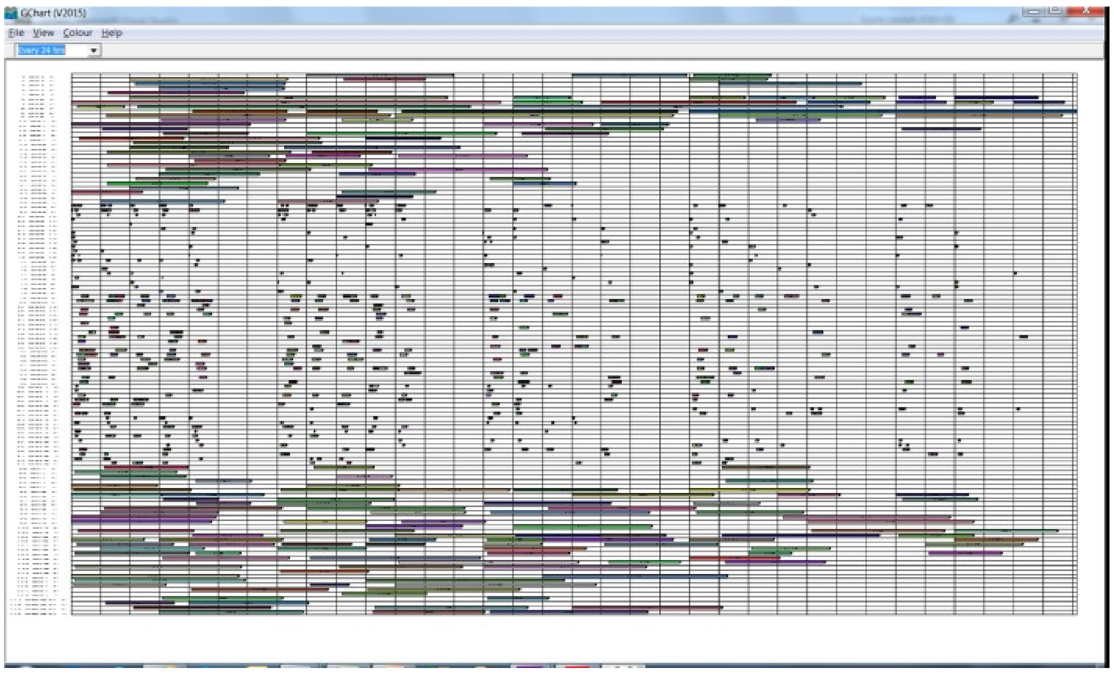
Table 15 demonstrates that the HSA results are far superior to the starting solutions and timing restrictions inflate CPU times considerably. The fixed OR sequences can be facilitated easily and do not cause CPU times to increase. It is evident that the schedule makespan is larger for the OR FXSEQ scenarios. This result provides further evidence of how OR activities have a large effect on hospital occupancy, the schedule horizon, and patient throughput. Freedom to jointly manipulate the OR sequences and the bed assignments is necessary if any improvement is to be realised.

The solution to the *FJ|STR, No FXSEQ|Cmax* is shown in Fig. 8. The *FJ|STR, OR FXSEQ|Cmax* solution is shown in Fig. 9. From these Gantt charts it is evident that beds in W2, W3 and W11 are more heavily utilised because more than one surgical speciality uses them. These wards dictate the makespan of the schedule. Our numerical testing shows that careful planning of the OR's greatly affects ward occupancies and the hospital's performance. Visual inspection of the solutions shows that further improvements are possible. These improvements may be achieved by the reapplication of HSA. The use of a compound perturbation operator however seems necessary to achieve further improvements with surety and finesse.



Download high-res image (1MB) Download full-size image

Fig. 8. Solution of *FJ|STR, No FXSEQ |Cmax*.



Download high-res image (1MB) Download full-size image

Fig. 9. Solution of FJ|STR, OR FXSEQ|Cmax.

6. Conclusions

This article considers how health care activities can be scheduled in hospitals and how beds and operating rooms can be utilised. In response a flexible job shop scheduling (FJSS) approach has been developed. This approach assigns patients to beds and other treatment spaces and sequences and schedules those patient's activities in the assigned locations. An approach such as this is greatly needed because most hospitals do not plan in advance how beds and other treatment spaces are going to be used. They also do not have a scheduling model that can immediately recompute activity timings and identify the hospital's state of occupancy into the near future. Without a scheduling model such as the one proposed, they cannot perform capacity checking and can easily overbook critical hospital resources such as treatment spaces. By far the most important benefit of the FJSP is knowing when activities should finish. The scheduling model can notify staff when to prepare patients to depart and hence free up resources quicker.

This article's approach is a new consideration for hospitals and will require them to invest in the development of new IT systems. In our opinion the FJSP approach has the potential to increase the overall efficiency and productivity of an entire hospital but only if it is integrated into an appropriate hospital information and management system (HIMS). It will be necessary to inform the HIMS when every activity begins and ends, and when every patient arrives and departs from the different locations within the hospital. This would enable the scheduling model to identify problems in advance and suggest better ways to operate.

The application of a FJSS approach to hospital scheduling is a relatively new avenue. Our approach is significant because it can be used to solve a vast array of scheduling problems. It constitutes a generic scheduling platform. It incorporates a wide range of advanced scheduling constraints. To our knowledge many different non-standard technical conditions have for the first time been incorporated within one approach. Different scheduling problems can easily be specified using our introduced specification constructs.

The proposed FJSP was applied to a variety of test cases. We have used it to schedule a real life scenario involving the surgical activities in a large public hospital. The numerical investigations have shown that the proposed methods are able to handle the scheduling requirements of an entire hospital. The emergency department is the only area yet to be integrated. Our numerical testing indicates that to maximise throughput in some hospitals, planned discharges should not occur on weekends as new patients cannot be admitted and operated upon till Monday of the new working week. In future research it would be beneficial to compare our approach with current practice to see just how applicable our approach is. It is important to remember however that the health care system is very conservative and is not open to change without vast amounts of evidence. It is not possible to apply approaches in real life settings without ethics approval. Hospitals are vast spaces with hundreds of beds, treatments areas and patients. A small area could in theory be scheduled, but that would not prove the validity for a “holistic” integrated approach.

As the FJSP is non-deterministic polynomial time hard (NP-hard), optimal solutions cannot be obtained with any surety or confidence. A hybrid Simulated Annealing (HSA) approach and constructive algorithms were hence implemented. The solutions that were obtained by our solution approaches were of high quality as evidenced by their proximity to lower bound approximations. Upon inspection, the number of idle time periods on each ward bed is reasonably small. There are more idle periods in the time restricted test cases because of the requirement to operate OR's in business hours. Our constructive algorithms are fairly intelligent and comparable to many other approaches in the scheduling literature, where success has been reported. Those types of algorithms always provide good solutions. The HSA results are far better than the constructive algorithm solutions and that is an indication of the ability of the HSA. The HSA is an effective search algorithm as evidently poor and mediocre starting solutions can be drastically improved.

The optimality of the HSA solutions is primarily but not exclusively of academic interest. It is debatable whether their determination is of interest to hospital managers and planning staff. In our observations the bottom line to them is “Will a scheduling model or process result in improved patient care”. Some might be interested in increasing the number of patients treated. Most would be happy if a new scheduling process is more effective than current processes.

As mentioned earlier, the implementation of the scheduling model “online” is perhaps more important. Given the current computing platform, it is debatable whether near optimal solutions can be determined in real time. In our opinion the HSA needs a “boost”, but that boost comes at significant additional computational overhead. Given the excessive computational requirements of scheduling so many jobs in real life, it is anticipated that recovery activities should be concentrated upon and other activities (i.e. POC and PAC) should be dealt with later.

This article demonstrates that timing restrictions are the most difficult element of this problem. A right shift correction policy consumes more CPU time, however superior solutions can be obtained. The alternative is to define dummy activities with tight release times and deadlines. This practice is more generic but causes additional practical complexities that restrict the effectiveness of traditional meta-heuristics. This article however does demonstrate that it is possible to schedule non fixed activities around fixed activities. Further research is required to overcome the aforementioned difficulties. Our future research will also consider uncertain length of stays and will focus upon the creation of robust schedules. The insertion and handling of acute patients will also be focussed upon. The solutions obtained by our search algorithms are already quite robust. They contain a limited number of idle time periods. These can be used to absorb variations in patient's length of stay. From a practical point of view it is futile to try and obtain solutions that are nearer to optimal, as they will not be as robust.

Acknowledgements

This research was funded by the Australian Research Council (ARC) Linkage grant LP 140100394 and supported by the Princess Alexandra Hospital, Brisbane, Australia. We would like to thank Dr. Michael Sinnot, Dr. David Cook, Sean Birgan and other staff at the PAH for their considerable feedback and guidance.

Appendix A. Nomenclature

The following is a list of the notation used.

Index:

i, j, k, g: Index for jobs, machines, the stage within a job, and groups.

p, u, w, π, ϕ: Index for patient, unit, ward and treatment area, treatment space, activity type

t: Index for time. Also used as a value of time.

Sets:

$P, J, M, \overline{\overline{M}}, G, O$: Set of jobs, machines, parallel machines, groups and activities.

O_i^1 : Set of activities for job J_i. $O_i^1 = \{o_{i,k} | k \in [1, K_i]\}$

O_j^2 : Set of activity that may be processed on machine *j*. $O_j^2 = \{o_{i,k} \in O | j \in \Gamma_{i,k}\}$.

V, A: Set of nodes and arcs in the disjunctive graph. $G = (V, A)$.

W, Π, P: Set of wards, spaces and patients

Π_w: Set of spaces in ward *w*.

Φ: Set of activity types. $\Phi = \{poc, sur, pac, rec\}$

M_ϕ: Set of machines that can perform activities of type *ϕ*.

$\widetilde{M}_{i,k}$: Set of candidate machines for operation $o_{i,k}$.

Parameters:

$o_{i,k}$: A reference to the *k* th activity of job *i*.

$b_{i,k}$, $c_{i,k}$, $d_{i,k}$: Start, end and departure time of activity $o_{i,k}$.

$type_{i,k}$: The activities type, i.e. ideal, blocking, no_wait

$m_{i,k}$: The machine to which activity $o_{i,k}$ is assigned.

$\rho_{i,k}$, $\rho_{i,k,j}$: The processing time of activity $o_{i,k}$ and the value when performed on machine *j*.

$fn_{i,k}$: The purpose of activity $o_{i,k}$. $fn_{i,k} \in \Phi$

$\Gamma_{i,k}$: Set of machines that may be used to process $o_{i,k}$.

$\omega_{i,k}$: The transfer time between $o_{i,k}$ and its successor.

$lag_{i,k}$: Time lag between the completion of $o_{i,k}$ and the start of its successor.

$\beta_{i,k}$: Buffering placed after activity $o_{i,k}$.

τ_j: Ready time for machine *j*.

so, si: The source and sink node of the disjunctive graph.

$s_j, \overline{s}_j$: Individual setup time and total setup time incurred by machine *j*.

$rlt_{i,k}$, $dedln_{i,k}$: Release time and deadline for activity $o_{i,k}$.

t_b^z, t_e^z : Start and end of invalid time period *z*.

PCP_p: Clinical pathway for patient p. $PCP_p = \{(\phi, u, t)\}$

G_j: The group that machine *j* belongs to.

fn_j: The primary purpose and function of machine *j*. $fn_j \in \Phi$

Decision variables:

$a_{i,k,j}$: Binary decision for the assignment of activity $o_{i,k}$ to machine j.

σ_j, $\sigma_{j,k}$: Sequence for machine j and the kth element respectively.

$\beta_{i,k}$: Buffering after activity $o_{i,k}$ to absorb variation and to provide robustness.

H: Schedule duration, i.e. makespan.

Recommended articles Citing articles (0)

References

Braubach et al., 2014 L. Braubach, A. Pokahr, W. Lamersdorf
Negotiation-based patient scheduling in hospitals. Technologies and decision support systems
Studies in Computational Intelligence, 486 (2014), pp. 107-121

Burdett, 2016 R.L. Burdett
Optimization models for expanding a railways theoretical capacity
European Journal of Operational Research, 251 (3) (2016), pp. 783-797
Article PDF (839KB)

Burdett and Kozan, 2001a R.L. Burdett, E. Kozan
Sequencing and scheduling for non-serial permutation flow shops
E. Kozan, A. Ohuchi (Eds.), Operations research/management science at work: Applying theory in the Asia Pacific region" in the international series in operations research & management science, Kluwer Academic Publishers (2001)

Burdett and Kozan, 2001b R.L. Burdett, E. Kozan
Sequencing and scheduling in flow shops with task redistribution
Journal of the Operational Research Society, 52 (2001), pp. 1379-1389

Burdett and Kozan, 2003 R.L. Burdett, E. Kozan
Evolutionary algorithms for resource constrained non-serial mixed flow shops
International Journal of Computational Intelligence and Application, 3 (4) (2003), pp. 411-435

Burdett and Kozan, 2009 R.L. Burdett, E. Kozan
Techniques for inserting additional trains into existing timetables
Transportation Research B, 43 (8) (2009), pp. 821-836
Article PDF (427KB)

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

Burdett and Kozan, 2010b R.L. Burdett, E. Kozan
Development of a disjunctive graph model and framework for constructing new train schedules
European Journal of Operational Research, 200 (1) (2010), pp. 85-98
Article PDF (514KB)

Burdett and Kozan, 2015 R.L. Burdett, E. Kozan
Techniques to effectively buffer schedules in the face of uncertainties
Computers and Industrial Engineering, 87 (2015), pp. 16-29
Article PDF (528KB)

Burdett and Kozan, 2016 R.L. Burdett, E. Kozan
A multi-criteria approach for hospital capacity analysis
European Journal of Operational Research, 255 (2) (2016), pp. 505-521
Article PDF (3MB)

Burdett et al., 2017 R.L. Burdett, E. Kozan, M. Sinnot, D. Cook, Y.C. Tian
A mixed integer linear programing approach to perform hospital capacity assessments

Expert Systems and Applications, 77 (2017), pp. 170-188
Article PDF (2MB)

Cappanera et al., 2014 P. Cappanera, F. Visintin, C. Banditori
Comparing resource balancing criteria in master surgical scheduling: A combined optimisation-simulation approach
International Journal of Production Economics, 158 (2014), pp. 179-196
Article PDF (679KB)

Carnes et al., 2011 T. Carnes, D. Price, R. Levi, P.F. Dunn, B.J. Daily, S. Moss
An optimization framework for smoothing surgical bed census via strategic block scheduling
Manufacturing Service Operation Management (2011), pp. 488-494

Chen et al., 2012 J.C. Chen, C.C. Wu, C.W. Chen., K.H. Chen
Flexible job shop scheduling with parallel machines using genetic algorithm and grouping genetic algorithm
Expert Systems with Applications, 39 (2012), pp. 10016-10021
Article PDF (673KB)

Chow et al., 2011 V.S. Chow, M.L. Puterman, N. Salehirad, W. Huang, D. Atkins
Reducing surgical ward congestion through surgical scheduling and uncapacitated simulation
Production and Operations Management, 20 (3) (2011), pp. 418-430

Fugener et al., 2014 A. Fugener, E.W. Hans, R. Kolisch, N. Kortbeek, P.T. Vanberkel
Master surgery scheduling with consideration of multiple downstream units
European Journal of Operational Research, 239 (2014), pp. 227-236
Article PDF (888KB)

Gartner and Kolisch, 2014 D. Gartner, R. Kolisch
Scheduling the hospital-wide flow of elective patients
European Journal of Operational Research, 233 (2014), pp. 689-699
Article PDF (672KB)

Groflin et al., 2011 H. Groflin, D.N. Pham, R. Burgy
The flexible blocking job shop with transfer and set-up times
Journal of Combinatorial Optimization, 22 (2011), pp. 121-144

Hayasaka and Hino, 2012 T. Hayasaka, R. Hino
Multiple exchanges of job orders for no-buffer job shop scheduling problem
Journal of Advanced Mechanical Design, Systems, and Manufacturing, 6 (5) (2012), pp. 661-671

Heiser, 2013 R. Heiser
Using a best practice perioperative governance structure to implement better block scheduling
AORN Journal, 97 (1) (2013), pp. 125-131
Article PDF (268KB)

Hulshof et al., 2012 P.J.H. Hulshof, N. Kortbeek, R.J. Boucherie, E.W. Hans, P.J.M. Bakker
Taxonomic classification of planning decisions in health care: A structured review of the state of the art in OR/MS
Health Systems, 1 (2012), pp. 129-175

Javid et al., 2017 A.A. Javid, Z. Jalali, K.J. Klassen
Outpatient appointment systems in healthcare: A review of optimization studies
European Journal of Operational Research, 258 (1) (2017), pp. 3-34

Jungwattanakit et al., 2009 J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse, F. Werner
A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria
Computers and Operations Research, 36 (2009), pp. 358-378
Article PDF (247KB)

Kozan and Liu, 2012 E. Kozan, S.Q. Liu
A demand-responsive decision support system for coal transportation

Decision Support Systems, 54 (2012), pp. 665-680
Article PDF (2MB)

Liu and Kozan, 2011 S.Q. Liu, E. Kozan
Optimising a coal rail network under capacity constraints
Flexible Service and Manufacturing Journal, 23 (2011), pp. 90-110

Liu and Kozan, 2012 S.Q. Liu, E. Kozan
A hybrid shifting bottleneck procedure algorithm for the parallel-machine job-shop scheduling problem
Journal of the Operational Research Society, 63 (2) (2012), pp. 168-182

Liu and Kozan, 2016 S.Q. Liu, E. Kozan
Parallel-identical-machine job-shop scheduling with different stage-dependent buffering requirements
Computers and Operations Research (2016), 10.1016/j.cor.2016.04.023

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

Mascis and Pacciarelli, 2002 A. Mascis, D. Pacciarelli
Job-shop scheduling with blocking and no-wait constraints
European Journal of Operational Research, 143 (2002), pp. 498-517
Article PDF (424KB)

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

Pham and Klinkert, 2008 D.N. Pham, A. Klinkert
Surgical case scheduling as a generalised job shop scheduling problem
European Journal of Operational Research, 185 (2008), pp. 1011-1025
Article PDF (494KB)

Queensland Health 2017 Queensland Health (2017). https://www.health.qld.gov.au/improvement/pathways.

QUT Data Finder QUT Data Finder: https://researchdatafinder.qut.edu.au/display/n11006.

QUT Eprints QUT Eprints: https://eprints.qut.edu.au/107724/.

Sauvey and Trabelsi, 2015 C. Sauvey, W. Trabelsi
Hybrid job shop scheduling with mixed blocking constraints between operations
Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th conference. Date 8–11 September 2015 (2015), pp. 1-8, 10.1109/ETFA.2015.7301538

Spratt and Kozan, 2016 B. Spratt, E. Kozan
Waiting list management through master surgical schedules: A case study
Operations Research for Health Care, 10 (2016), pp. 49-64
Article PDF (1MB)

Xia and Wu, 2005 W. Xia, Z. Wu
An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems
Computers and Industrial Engineering, 48 (2005), pp. 409-425
Article PDF (264KB)

Yazdani et al., 2010 M. Yazdani, M. Amiri, M. Zandieh
Flexible job-shop scheduling with parallel variable neighbourhood search algorithm
Expert Systems with Applications, 37 (2010), pp. 678-687
Article PDF (674KB)

Yen, 1970 J.Y. Yen
An algorithm for finding shortest routes from all source nodes to a given destination in general networks
Quarterly of Applied Mathematics, 27 (1970), pp. 526-530

Zeng et al., 2014 C. Zeng, J. Tang, C. Yan
Scheduling of no buffer job shop cells with blocking constraints and automated guided vehicles
Applied Soft Computing, 24 (2014), pp. 1033-1046
Article PDF (3MB)

Zhang et al., 2009 G. Zhang, X. Shao, P.L.L. Gao
An effective hybrid particle swarm optimization algorithm for multi-objective flexible job shop scheduling problem
Computers and Industrial Engineering, 56 (2009), pp. 1309-1318
Article PDF (645KB)