AMSI
AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

National Collaboration in the Mathematical Sciences

New Theory and Algorithms for Scheduling Arc Shutdown Jobs in Networks

Patrick Andersen

February 27, 2014

This project explores a recently developed type of scheduling problem that combines the two Operations Research areas of job scheduling and network optimisation. The Arc Shutdown Scheduling Problem (ASSP) involves finding the best way of scheduling routine maintenance jobs for an infrastructure network so as to maximise the performance of the network over a time horizon. This project examines a heuristic approach to solving this problem and the issues associated with using the approach for a more difficult generalisation of the ASSP that incorporates storage facilities as part of the network.

Postal Address: 111 Barry Street
c/- The University of Melbourne
Victoria 3010 Australia

Email: enquiries@amsi.org.au
Phone: +61 3 8344 1777
Fax: +61 3 9349 4106
Web: www.amsi.org.au

Contents

Postal Address: 111 Barry Street
c/- The University of Melbourne
Victoria 3010 Australia

Email: enquiries@amsi.org.au
Phone: +61 3 8344 1777
Fax: +61 3 9349 4106
Web: www.amsi.org.au

1 Introduction

1.1 Scheduling Problems

Scheduling problems are those that involve a set of jobs or tasks that must be performed during a given time horizon. The jobs must be scheduled (i.e. allocated starting times) so as to optimise a certain objective such as minimising the makespan (i.e. completion time of the latest finishing job). These problems usually have constraints on the schedule such as resource availability constraints if the jobs use a resource, or time window constraints where the job must be scheduled after a given starting time and completed before a given due time. Scheduling problems can be difficult to solve exactly and extensive research has been done in the area in order to find good heuristics and approximation algorithms as well as efficient approaches to solving the problem exactly [7].

1.2 Network Optimisation

Network optimisation is a classic area of Operations Research and has been vigorously studied since the earliest days of the field [6]. The particular network optimisation problem studied here is the maximum flow problem which is the problem of finding the maximum amount flow that can be sent through one end of a directed graph to the other whose arcs have fixed flow capacities. The node of the graph from which flow is sent is known as the source node and the node to which flow is sent is the sink node. Networks with multiple source and sink nodes can formulated as a network with a single sink and source by creating a dummy source (sink) node that is connected via infinite capacity arcs to the true source (sink) nodes. Very efficient algorithms exist that can solve the maximum flow problem exactly. [1]

1.3 Arc Shutdown Scheduling Problem

The problem studied here, the Arc Shutdown Scheduling Problem (ASSP), involves a combination of both these areas. In the problem, jobs must be scheduled over a given time horizon. Each job corresponds to an arc in a given network and while the job is active, the capacity of the arc is affected. The objective is to schedule the jobs so as to maximise the total flow through the network over the time horizon.

1.3.1 ASSP Example

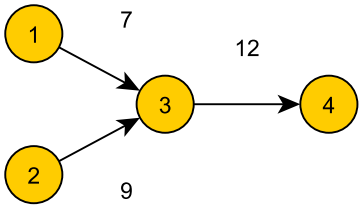
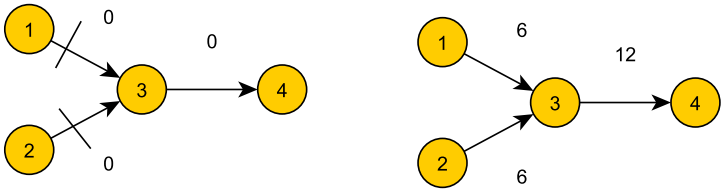
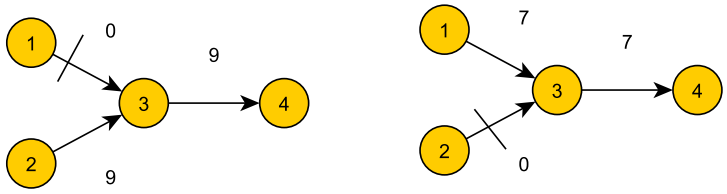


Figure 1: An example network. Flow must travel from nodes 1 and 2 to node 4. The numbers on the arcs represent the arc capacities (units of flow per day for this example).



(a) Day 1 (b) Day 2

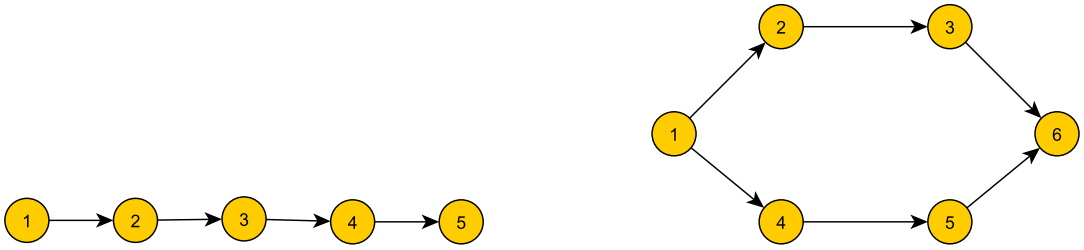
Figure 2: A possible two day, two job maintenance schedule for the network in Figure 1 where each job takes a day to complete. Here the arc numbers represent flows and crossed out arcs are those with jobs operating on them. The total flow for this schedule is 12 units.



(a) Day 1 (b) Day 2

Figure 3: An alternate maintenance schedule using the same jobs from Figure 2 scheduled on different days. Here the resultant total flow is 16 units which is an improvement from the previous schedule.

Typically a good solution is one in which jobs on arcs in series (arcs in a connected chain) are scheduled as close together as possible, and jobs on arcs in parallel (arcs that are part of different paths through the network) are scheduled as far apart as possible. This is due to arcs in series creating a bottleneck so if one arc is inactive, the whole chain can't pass any flow. Parallel arcs provide multiple routes of flow to the sink so if a job is scheduled on one arc, the other parallel arc can still be used to send flow through the network.



(a) Arcs in series. (b) Arcs in parallel.

Figure 4: For the arcs in series, if a job is scheduled on any arc, then no flow can pass through the chain, hence it would be prudent to schedule other jobs on the chain at the same time while the flow is already being blocked. For the arcs in parallel, we have more than route through the network so it would be inadvisable to schedule jobs together so as to block the multiple routes simultaneously.

Postal Address: 111 Barry Street
c/- The University of Melbourne
Victoria 3010 Australia

Email: enquiries@amsi.org.au
Phone: +61 3 8344 1777
Fax: +61 3 9349 4106
Web: www.amsi.org.au

2 Background

The ASSP first arose during the process of modelling the capacity of the Hunter Valley Coal Chain (HVCC). The HVCC is the system of logistics facilities - principally a network of rail track extending 450km inland to cover 30 coal load points in the Hunter Valley, and three coal handling terminals - which enables coal mined by producers in the Hunter Valley to be transported, assembled, and loaded onto ships for export at the port of Newcastle [4]. As of 2013, the HVCC was delivering around 140 million tonnes of coal per year with the port of Newcastle exporting more coal by volume than any other facility in the world [5].

All the key assets (e.g. rail track sections, coal stacking machinery, terminal conveyor systems) of the HVCC have to undergo regular preventive maintenance, planned well in advance. While undergoing maintenance, an asset cannot function to deliver coal, which reduces the capacity of the system. Different maintenance schedules can result in different system capacities so astute scheduling of maintenance activities can improve the overall throughput of the network.

By modelling the HVCC as a combinatorial network [3] (see Figure 5), the problem of finding the best maintenance job schedule so as to maximise the network's coal throughput is the same the ASSP. The arcs of the graph represent the infrastructure components of the coal chain and their capacities are the amount of coal that can be passed through per time unit.

Scheduling a job on an arc reduces its capacity to zero for the job's duration and the best job schedule is the one that maximises the total flow of the network for a given time horizon.

3 Modelling the ASSP

3.1 Network

To model the basic ASSP, we start with a given network $N = (V, E)$ with $S, S' \in V$ as the source node and sink node respectively and with all arcs in E having given capacities. Our objective will involve maximising the flow from S to S' over series of subgraphs of N where scheduling decisions affect which subgraphs will be used.

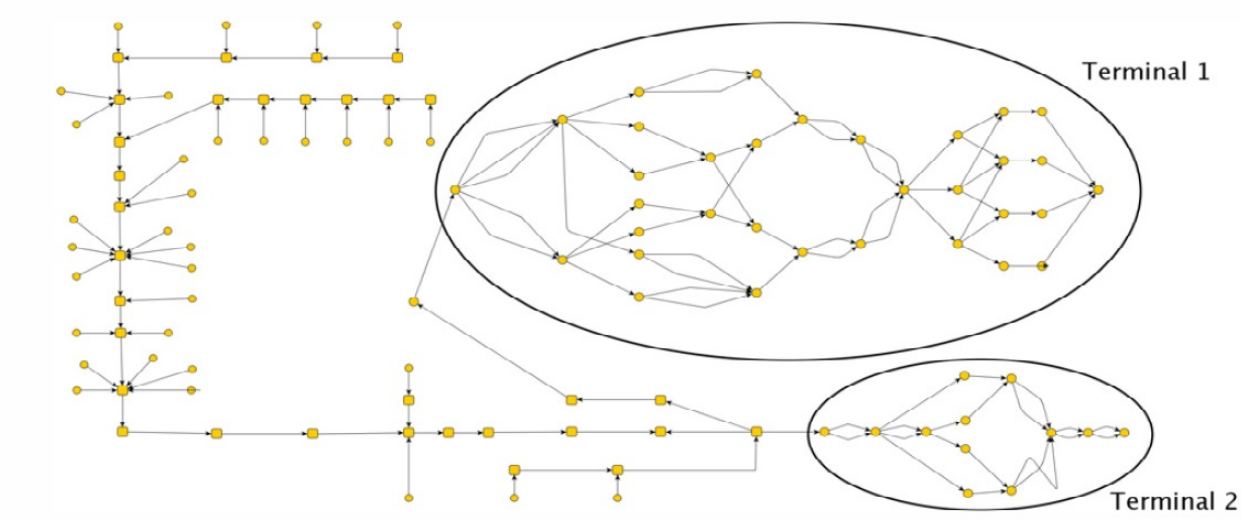


Figure 5: The HVCC network. The circled parts of the network represent the flow of coal through terminal handling equipment (the ends of the terminals are the sink nodes). The rest represents the rail network, sourcing coal from 33 coal load points (the source nodes) [3].

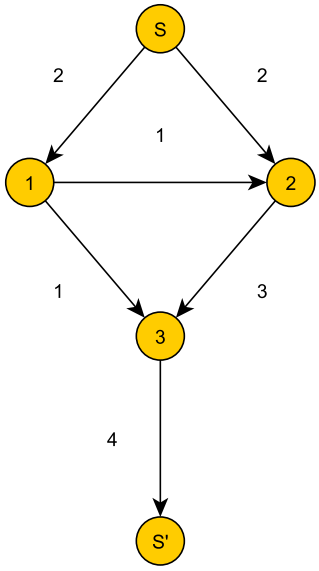


Figure 6: An example network N where the numbers on the arcs represent capacities.

Postal Address: 111 Barry Street
c/- The University of Melbourne
Victoria 3010 Australia

Email: enquiries@amsi.org.au
Phone: +61 3 8344 1777
Fax: +61 3 9349 4106
Web: www.amsi.org.au

3.2 Job List

We also start with a given set of jobs J. Each job $j \in J$ has:

- The arc $a \in E$ on which the job is to be performed
- A processing time p_j
- A release time r_j (earliest possible starting time)
- A due time d_j (latest possible start time $= d_j - p_j$)

If we have a feasible schedule, then each job $j \in J$ will have a start time $S_j \in [r_j, d_j - p_j]$ and will finish at time $t = S_j + p_j$
These S_j's are the decision variables for the problem.

3.3 Time Slicing

Let the length of the time horizon be T.

Given a feasible schedule, we model the change in the system over time using time-slicing. A time slice is an interval of time for which the state of the network (the arcs that are currently active) is the same for all times within the interval.

Since the state of the network only changes when a job starts or finishes, we can find the time slices by ordering the set $\{S_j, S_j + p_j : j \in J\} \cup \{0, T\}$ into time points $\{t_0 < t_1 < \cdots < t_{m-1} < t_m\}$ where t_0 is smallest time point (0) and t_m is the largest time point (T) to get the time slices: $[t_0, t_1], [t_1, t_2], \ldots, [t_{m-1}, t_m]$
$[t_{i-1}, t_i]$ is numbered time slice i for $i = 1, \ldots, m$ and the length of time slice i is $l_i = t_i - t_{i-1}$

3.4 Objective Function

Let z_i be the maximum flow of the network during time slice i (this can be found easily through solving a linear program for maximum flow or through other efficient max flow algorithms [1])

The objective value of the system (the total flow of the network over the time horizon) given a feasible schedule is $\sum_{i=1}^{m} l_i z_i$

4 Local Job Search Heuristic

Whilst it's possible to model the ASSP as a mixed integer program, like with other types of scheduling problems, this formulation turns out to be computationally expensive to solve [3]. Thus heuristic algorithms are the chosen approach to finding good solutions to the problem.

One possible heuristic that can be employed to potentially find a good solution from a starting feasible schedule is the Local Job Search. This works by fixing all jobs in the schedule except for a single job j. The starting time of this job is moved within its feasible range $[r_j, d_j - p_j]$ to the time which gives the maximum objective value. The process is then repeated for all jobs $j \in J$.

The result of moving the job j can be seen by describing the objective value as a function of S_j.

4.1 Notation

We assume all following intervals are discrete intervals. i.e. $[a, b] \equiv [a, b] \cap \mathbb{N}$

Let $\bar{J}' = \{S_i, S_i + p_i : i \neq j\} \cup \{r_j, d_j - p_j\}$
$= \{t_0 < t_1 < ... < t_m\}$

Let $\bar{J}^* = \{t_0^* < t_1^* < ... < t_m^*\}$ where $t_i^* = t_i - p_j$ for $t_i \in \bar{J}$

Let $J' = \bar{J}' \cap [r_j, d_j - p_j]$

Let $J^* = \bar{J}^* \cap [r_j, d_j - p_j]$

Let z_i^+ be the maximum flow of the network at time slice i without job j active.
Let z_i^- be the maximum flow of the network at time slice i with job j active.

(z_i^+ and z_i^- are assumed to have been precomputed using an efficient max flow algorithm.)

Also, we will use the convention that if we have $a > b$, then $\sum_{i=a}^{b} f(i) = 0$

All other notation used below is defined in the previous section.

4.2 Objective Value as Function of Start Time of Moving Job

Let $f : [r_j, d_j - p_j] \to \mathbb{R}$ be the objective value of the schedule as a function of the starting time of the moving job j. I.e $f(S_j)$ = the objective value of the schedule where job j starts at time S_j.
For $S_j \in [t_{p-1}, t_p] \cap [t^*_{q-1}, t^*_q]$ where $t_{p-1}, t_p \in J'$ and $t^*_{q-1}, t^*_q \in J^*$,

$$\begin{aligned} f(S_j) = &\sum_{i=1}^{p-1} l_i z_i^+ + (S_j - t_{p-1}) z_p^+ \\ &+ (1 - \delta_{pq})[(t_p - S_j) z_p^- + \sum_{i=p+1}^{q-1} l_i z_i^- + (S_j - t^*_{q-1}) z_q^-] \\ &+ \delta_{pq} p_j z_p^- + (t^*_q - S_j) z_q^+ + \sum_{i=q+1}^{m} l_i z_i^+ \end{aligned}$$

where $\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$

4.3 Properties of $f(S_j)$

1. f is a continuous piecewise linear function where the breaks in f occur when S_j or $S_j + p_j$ move into a new time slice. Thus the breakpoints of f are the points sets $J' = \{t_0 < t_1 < ... < t_m\} \cap [r_j, d_j - p_j]$ and $J^* = \{t_0 - p_j < t_1 - p_j < ... < t_m - p_j\} \cap [r_j, d_j - p_j]$. Hence it is possible to derive all of f from finding the value of f at these breakpoints.

2. The starting time for job j that gives the maximum objective value occurs at one of the breakpoints. If all data is integer then the maximum objective value will be integer.

3. Flow in time slices before r_j and after d_j are unaffected. I.e. the z_i values for the timeslices i into which job j is not permitted to move are unaffected by the movement of S_j. This means that the function could be modified to not include these flow values as part of its aggregate and the Local Job Search would still yield the same result.

4. If the given schedule was optimal, the maximum values of $f(S_j)$ for all the jobs would occur at the given schedule start times (the Local Job Search wouldn't

change the solution). This implies the optimal starting times in an exact solution of ASSP will be integer if all data is integer.

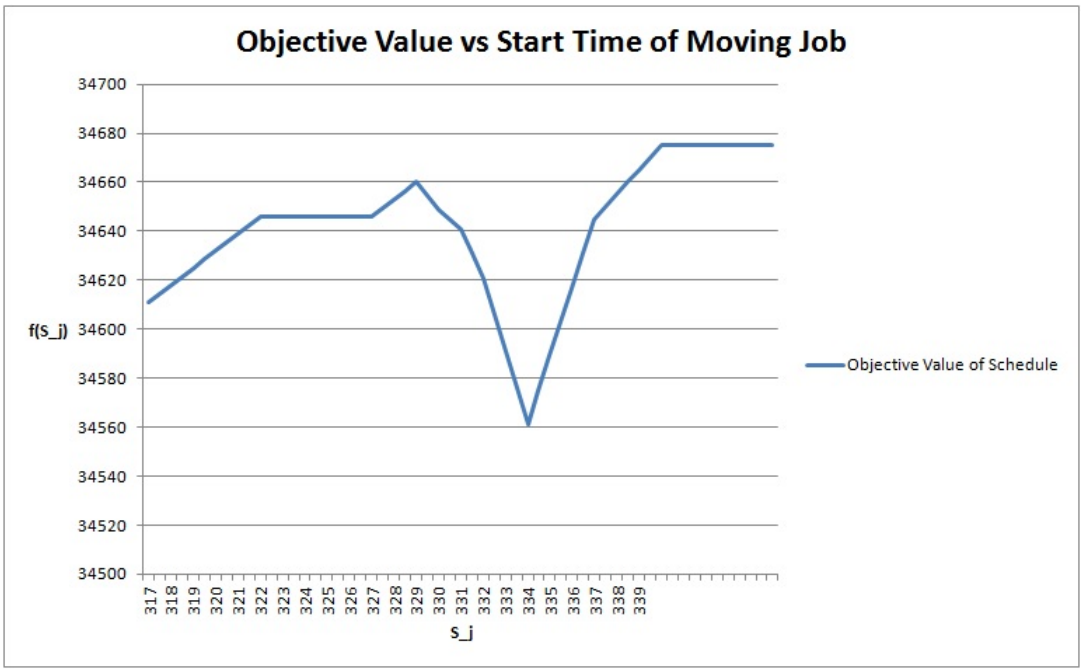


Figure 7: An example plot of $f(S_j)$ generated using an instance which contained a network of 12 nodes and 32 arcs and a job list of 310 jobs over a time horizon of 1000 units. This plot demonstrates another property of f, non-convexity, since it can be seen there is a local maximum that isn't global.

5 ASSP with Storage Nodes

We now consider a more general (and difficult) version of the problem: scheduling jobs on networks with storage nodes.

5.1 Definition

A storage node is a component of the network where flow is held onto for a period of time before being passed on to the rest of the network. Each storage node in the

network has a given fixed capacity
In the HVCC example, storage nodes can be thought of as coal stockpiles where coal can be kept whilst network is currently at capacity and passed on when there is some free capacity later.

5.2 Storage Node Properties

5.2.1 Time Slice Interconnectedness

In order to characterise the objective value of a given schedule in the case with storage, we can no longer think of the system as a collection of separate networks for each time slice, but instead as one single network where the individual time slice networks are connected via arcs between consecutive copies of the storage nodes.
I.e. instead of having a collection of disconnected networks for each time slice, each with its own source and sink, we now have one connected network with a universal source and sink and with an arc going from every storage node in every time slice other than the last one to the corresponding storage node in the next time slice. The capacities of these arcs are the corresponding storage capacities (see Figure 8 and Figure 9 for illustration).

5.2.2 Forward Propagation from Job Movement

Without storage nodes we had the condition that moving a job within a given range will not affect flow in time slices outside that range. With storage nodes, that is no longer true as changing the active jobs in a time slice can now affect arbitrarily many time slices in the future as Figure 8 and Figure 9 demonstrate.

5.2.3 Non-integer Maxima

Another property of the storage network is that if all our data is integer, the optimal schedule for the network does not necessarily have integer values (unlike the case with no storage). This can best be demonstrated in the example in Figure 10 below [2]:

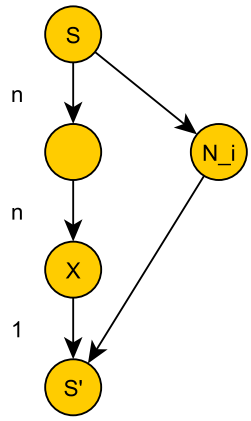
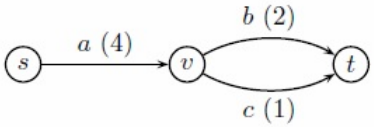


Figure 8: An example network with storage node X and capacities on arcs. Here N_i represents a subnetwork that will change in each different time slice i in terms of active jobs (needed for example since something in the network must change between time slices).



j	a_j	r_j	d_j	p_j
1	a	6	7	1
2	b	3	5	2
3	c	0	5	5
4	a	0	6	3

Figure 10: An example network with source s and sink t and a corresponding job list table. The numbers on the arcs represent the capacities and node v is the storage node with capacity 3.

In the table, the only job that can be moved is job 4. The total capacity of the arcs going out of v is limited by 2, 0 and 3 for the time intervals [0,3], [3,5] and [5,7], respectively. This implies an upper bound of 12 for the total throughput, and this bound is achieved if and only if job 4 starts at time $S_4 = 3/2$.
To see this note that in order to use all the capacity of arc b for the interval [0,3], the total flow on arc a over this interval has to be at least 6, which implies $S_4 \geq 3/2$. On the other hand, it follows from $S_4 + (6 - S_4 + p_4)) = 3 = 12/4$ that in order to achieve a total throughput of 12 the arc a has to be at capacity for the whole interval [0, S_4], and due to the storage capacity this is only possible if $S_4 \leq 3/2$.

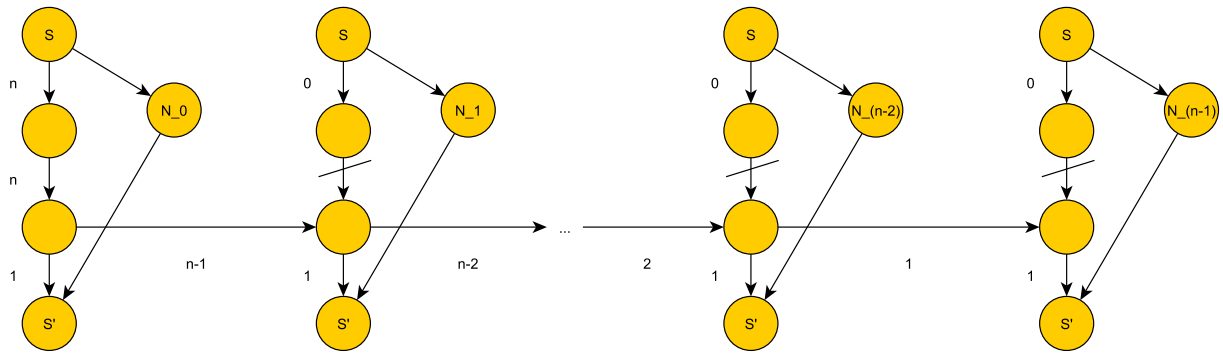


Figure 9: The network from Figure 8 with the numbers on the arcs representing flow values for some job schedule where we have a different network component for each time slice. If we were to assign a job to one of the arcs of capacity n in the first time slice, the storage flow throughout the rest of the time slices would be reduced to zero, affecting $n - 1$ time slices in the future.

5.2.4 Additional Breakpoints

An immediate consequence of the previous property is that we must have additional breakpoints for the piecewise linear function in Section 4 other than those already characterised. This means that in order to derive the function f, we can no longer use the function value of the breakpoints alone without finding some way of characterising these new breakpoints. We can generate these functions numerically by incrementing S_j and computing the total flow of the resultant interconnected time slice network (which is the approach used for Figure 7 and Figure 11), however this method can be quite computationally expensive and runs the risk of there being some approximation error. If the breakpoints can be characterised, then the Local Job Search heuristic would be a very viable heuristic to use in order to find a good solution for the storage node case.

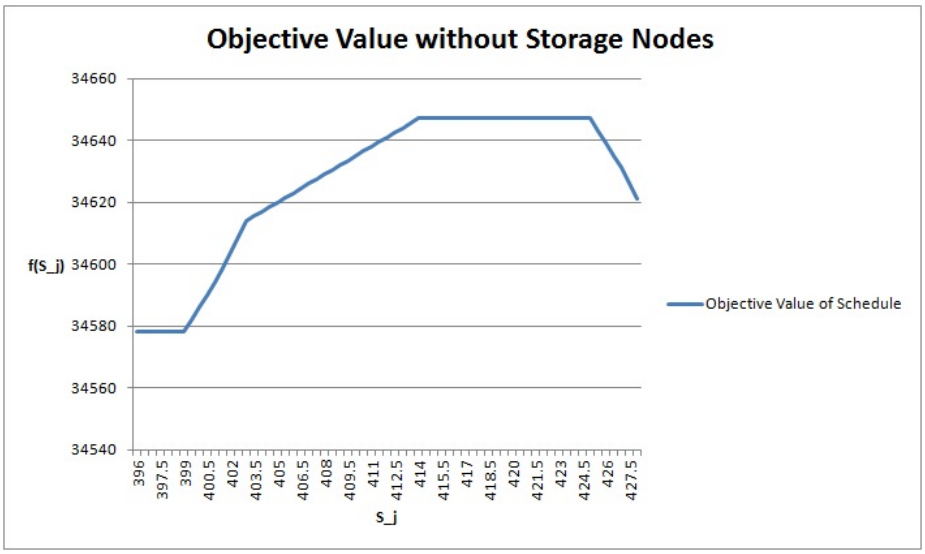


Figure 11: A plot of $f(S_j)$ generated using the same instances as in Figure 7 but with a different moving job.

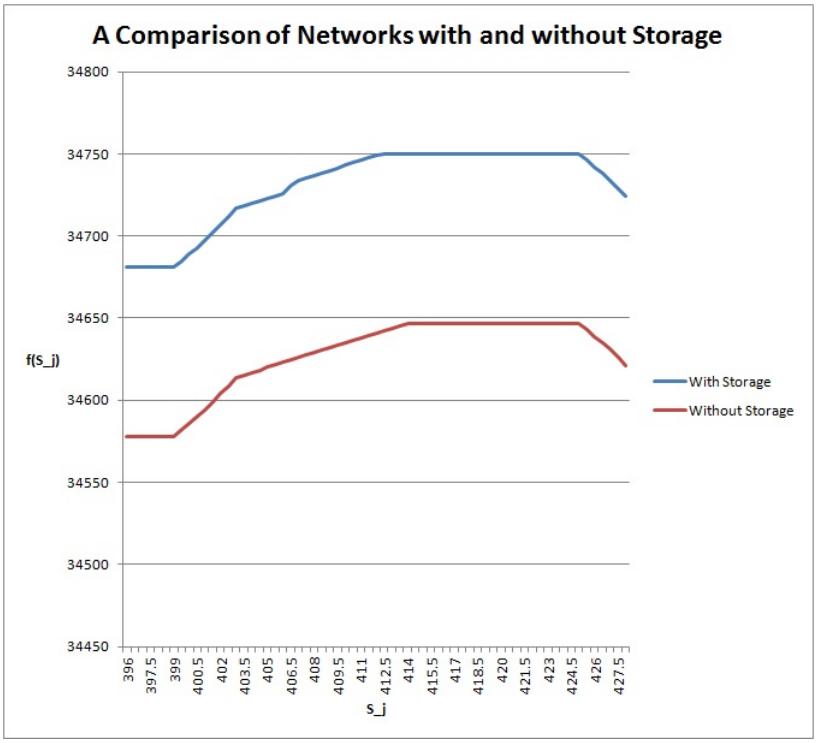


Figure 13: A single graph comparison of the plots from Figure 11 and Figure 12

Postal Address: 111 Barry Street
c/- The University of Melbourne
Victoria 3010 Australia

Email: enquiries@amsi.org.au
Phone: +61 3 8344 1777
Fax: +61 3 9349 4106
Web: www.amsi.org.au

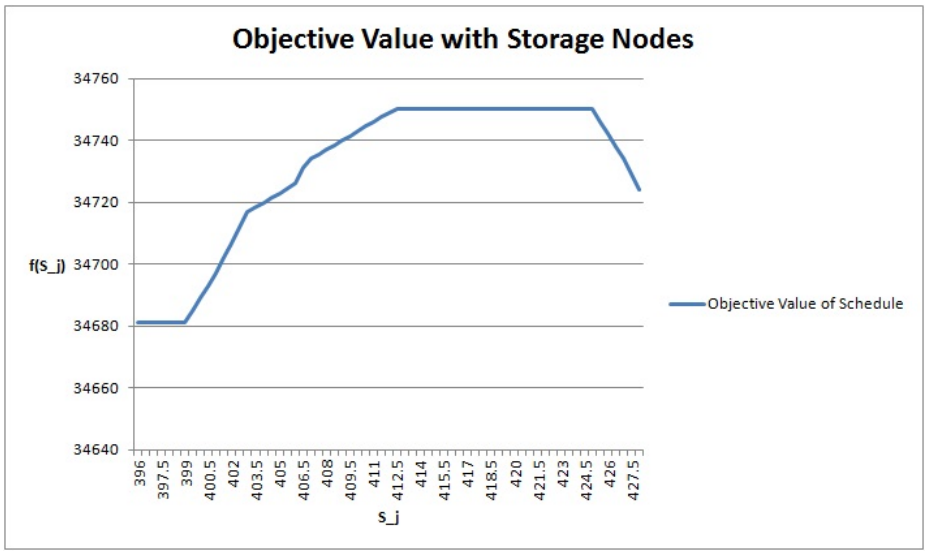


Figure 12: The same instances and moving job as in Figure 11 but with a storage node of capacity 5 as part of the network. It can be seen that there is an extra pair of breakpoints occurring in this plot.

6 In-out Networks

6.1 Definition

In order to assist in understanding the features of the storage problem, particularly the additional breakpoints, we consider a specific type of network; the in-out network.

An in-out network is one in which there are only three nodes; a source node a storage node and a sink node. An arc (known as the inbound arc) connects the source node to the storage node and another arc (known as the outbound arc) connects the storage node to the sink node.

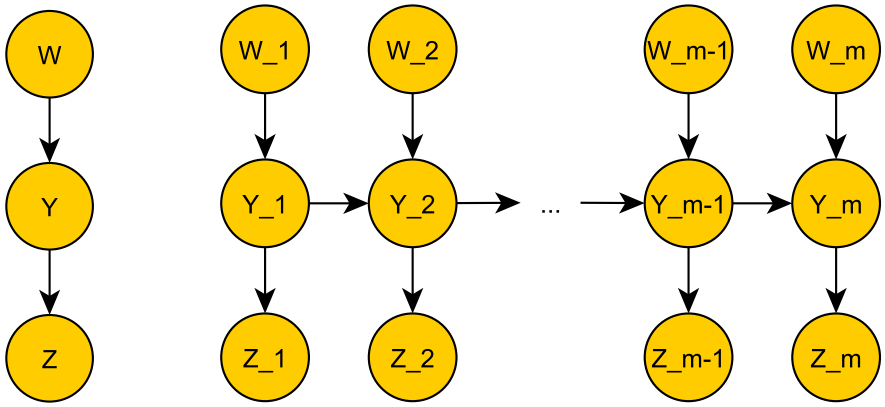


Figure 14: The in-out network with source W, sink Z and storage node Y and the network divided into interconnected time slices.

6.2 Notation

For a given schedule and time slicing:

$$\begin{aligned} u_i^{\text{in}} &= \text{in capacity in slice } i \\ u_i^{\text{out}} &= \text{out capacity in slice } i \\ u_{\text{storage}} &= \text{storage capacity} \\ m &= \text{number of timeslices} \end{aligned}$$

$$\begin{aligned} z_i &= \text{outflow in timeslice } i \\ y_i &= \text{storage flow from timeslice } i \text{ to timeslice } i+1 \\ w_i &= \text{inflow in timeslice } i \end{aligned}$$

$$\text{Total outflow} = \sum_{i=1}^{m} z_i$$

6.3 Greedy Flow Algorithm

One way to find the maximum flow in the in-out network is to use the Greedy Flow Algorithm. This algorithm allocates flow so that at any given time slice, as much

flow as possible sourced from the current time slice's in-flow and previous time slice's storage flow is sent as out-flow for the current time slice. Of the remaining in-flow and storage flow, as much of it as possible is sent to the next time slice as storage flow.

6.3.1 Mathematical Formulation

Let $y_0 = 0$. Then the Greedy Flow Algorithm will iteratively allocate flow in such a way that:

$$z_i = \min\{u_i^{\text{in}} + y_{i-1}, u_i^{\text{out}}\}$$

$$y_i = \max\{\min\{u_i^{\text{in}} + y_{i-1} - u_i^{\text{out}}, u_{\text{storage}}, \sum_{j=i+1}^{m} u_j^{\text{out}}\}, 0\}$$

$$w_i = y_i + z_i - y_{i-1}$$

6.3.2 Optimality

Proposition: The total outflow, $\sum_{i=1}^{n} z_i$, is a maximum flow of the network.

Proof: Suppose we have an in-out network $N = (V, E)$ with m time slices. Let the nodes v_i^{in}, v_i^{out}, $v_i^{\text{s}} \in V$ be the source node, sink node, and storage respectively for time slice i. I.e. there is an arc $(v_i^{\text{in}}, v_i^{\text{s}}) \in E$ with capacity u_i^{in} $\forall i \in [1, m]$, an arc $(v_i^{\text{s}}, v_i^{\text{out}}) \in E$ with capacity u_i^{out} $\forall i \in [1, m]$, and an arc $(v_i^{\text{s}}, v_{i+1}^{\text{s}}) \in E$ with capacity u_{storage} $\forall i \in [1, m-1]$.

Let $S \in V$ be a common source node and add arcs $(S, v_i^{\text{in}}) \in E$ $\forall i \in [1, m]$ to N with infinite capacity, and let $S' \in V$ be a common sink node and add arcs $(v_i^{\text{out}}, S') \in E$ $\forall i \in [1, m]$ to N, also with infinite capacity.

Suppose we have a feasible flow for N that has been allocated according the Greedy Algorithm, i.e. the total outflow is $\sum_{i=1}^{m} z_i$, and assume that this is not a maximum flow for N. This implies that in the residual network, there exists a simple directed path from S to S'. Due to the structure of N, this path must begin with the initial sub-path $(S, v_j^{\text{in}}, v_j^{\text{s}})$ for some $j \in [1, m]$ and end with the final sub-path $(v_k^{\text{s}}, v_k^{\text{out}}, S')$ for some $k \in [1, m]$. Hence the existence of a path implies that there is unused in capacity in time slice j and unused out capacity in time slice k. There are two possible cases for simple directed paths from S to S':

Case (1): $j \geq k$
The path from S to S' must be of the form $(S, v_j^{\text{in}}, v_j^{\text{s}}, v_{j-1}^{\text{s}}, v_{j-2}^{\text{s}}, ..., v_{k+1}^{\text{s}}, v_k^{\text{s}}, v_k^{\text{out}}, S')$. (Note: the path cannot visit nodes outside time slice j or k other than S, S', and storage nodes v_i^{s} where $k < i < j$ without resulting in a repetition of visited nodes.) The existence of this path implies that in the current flow allocation, flow is being sent via storage from time slice k to time slice j. However since there must also be unused out capacity in time slice k, this contradicts the definition of z_i since the algorithm would always choose to send flow out in the current time slice before any flow is sent to future time slices via storage. Hence this path cannot exist.

Case (2): $j < k$
The path from S to S' must be of the form $(S, v_j^{\text{in}}, v_j^{\text{s}}, v_{j+1}^{\text{s}}, v_{j+2}^{\text{s}}, ..., v_{k-1}^{\text{s}}, v_k^{\text{s}}, v_k^{\text{out}}, S')$. (Note: the path cannot visit nodes outside time slice j or k other than S, S', and storage nodes v_i^{s} where $j < i < k$ without resulting in a repetition of visited nodes.) The existence of this path implies that there is available storage capacity in the time slices j to $k-1$ such that additional flow stored at time slice j can eventually become additional outflow in time slice k. However since there is unused in capacity in time slice j, this contradicts the definition of y_i since if there is remaining inflow in a time slice after the algorithm has sent out as much outflow as the out capacity of the time slice will allow, then it sends as much of the remainder as possible as storage flow to future time slices. Hence this path cannot exist.

Therefore there is no simple directed path from S to S' in the residual network of N.
Therefore, by contradiction, the total outflow as given by the Greedy Algorithm, $\sum_{i=1}^{m} z_i$, is a maximum flow. □

6.4 Upper Bound on Number of Time Slices Affected by Job Movement

Assuming we were allocating flow using the Greedy Flow Algorithm, we can get an upper bound for the number of time slices affected by a job movement. For a job movement to an earlier start time, this can be found by evaluating the new storage flow y_i of the last time slice i the job was active in before it was moved and determining, based on the available out capacity of future time slices, how many time slices it would take to consume this storage flow.
E.g. if we increase the capacity of an inbound arc i by δ, then a naive upper bound

for the number of time slices affected is the smallest number k such that:
$\sum_{j=i}^{i+k} p(j) \geq \min\{u_{\text{storage}}, u_i^{\text{in}} + \delta - u_i^{\text{out}}\}$

where:

$$p(i) = \begin{cases} u_i^{\text{out}} - (u_i^{\text{in}} + \delta) & \text{if } u_i^{\text{out}} > (u_i^{\text{in}} + \delta) \\ 0 & \text{otherwise} \end{cases}$$

and

$$p(j) = \begin{cases} u_j^{\text{out}} - u_j^{\text{in}} & \text{if } u_j^{\text{out}} > u_j^{\text{in}} \\ 0 & \text{otherwise} \end{cases}$$

for $j > i$

A similar process can be used to determine an upper bound for a later start time job movement by computing the y_i for last time slice the job will be active in once moved, to determine how many time slices would be affected if this flow y_i was decreased.

7 Conclusion and Future Work

We as of yet have been unable to properly characterise the breakpoints of the storage case function however by working with the in-out networks and continuing with computer experiments, we hope that this will lead to some insight into the nature of this problem. The end goal of this is to be able to have a good formulation of a piecewise linear objective function of starting time that can be used for the Local Job Search in the case of storage nodes so that the effectiveness of this heuristic can be explored for the storage node problem.

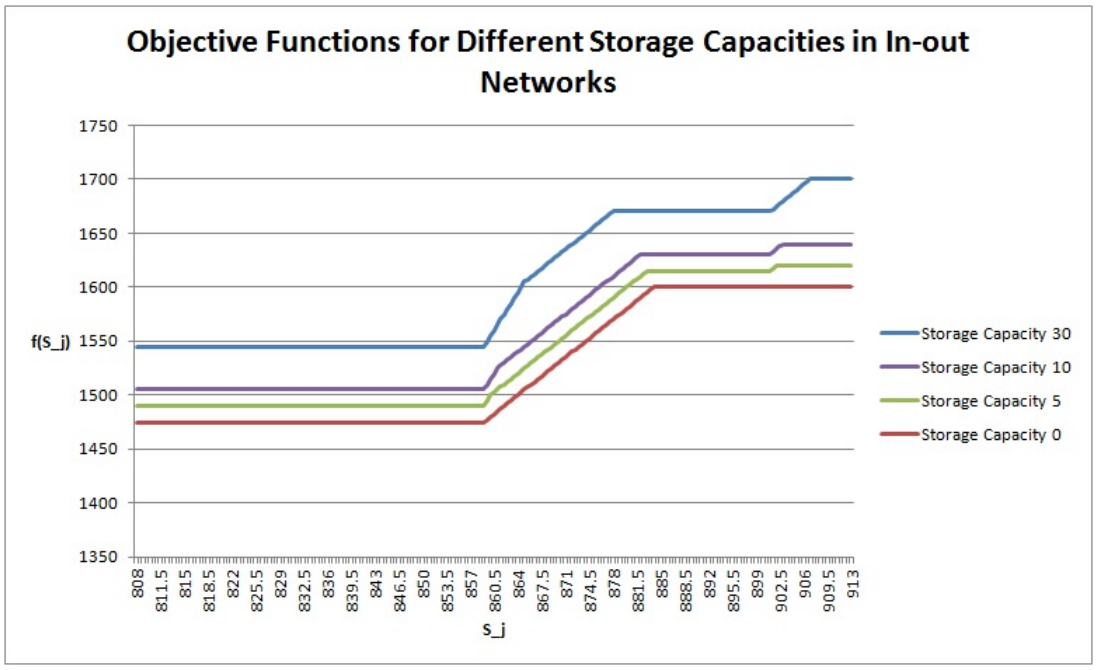


Figure 15: A comparison of objective functions for different storage capacities using the same in-out network with an in capacity 10 units and an out capacity of 5 units, and a job list of 5 jobs. This kind of result will hopefully be useful in the future when it comes to characterising the storage case breakpoints.

References

[1] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows*. Prentice Hall, Upper Saddle River, NJ, 1st edition.

[2] Boland, N., Kalinowski, T., and Kaur, S. (2013a). Scheduling network maintenance jobs with release dates and deadlines to maximize total flow over time: Bounds and solution strategies (manuscript).

[3] Boland, N., Kalinowski, T., Waterer, H., and Zheng, L. (2014). Scheduling arc maintenance jobs in a network to maximize total flow over time. *Discrete Applied Mathematics*, 163:34–52.

[4] Boland, N., McGowan, B., Mendes, A., and Rigterink, F. (2013b). Modelling the capacity of hunter valley coal chain chain to support capacity alignment of main-

tenance. In Piantadosi, J., Anderssen, R., and Boland, J., editors, *MODSIM2013, 20th International Congress on Modelling and Simulation*. Modelling and Simulation Society of Australia and New Zealand.

[5] Boland, N. and Savelsbergh, M. (2012). Optimizing the hunter valley coal chain. In Gurnani, H., Mehrotra, A., and Ray, S., editors, *Supply Chain Disruptions: Theory and Practice of Managing Risk*, pages 275–302. Springer, London.

[6] Ford, L. R. and Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–405.

[7] Pinedo, M. L. (2012). *Scheduling: Theories, Algorithms and Systems*. Springer, New York, NY, 4th edition.

Acknowledgements

The University of Newcastle

Professor Natashia Boland

Dr Thomas Kalinowski

Simranjit Kaur

Australian Mathematical Sciences Institute