Journal of Heuristics, 6: 457–480 (2000)

New and "Stronger" Job-Shop Neighbourhoods: A Focus on the Method of Nowicki and Smutnicki (1996)

ANANT SINGH JAIN
i2 Technologies UK, The Priory, Stomp Road, Burnham, Buckinghamshire, England, UK, SL1 7LL
email: anant_jain@i2.com

BALASUBRAMANIAN RANGASWAMY
Optimisation and Logistics Modelling Group, US WEST Advanced Technologies, 4001 Discovery Drive, Boulder, CO 80303, USA
email: brangas@advtech.uswest.com

SHEIK MEERAN*
Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, UK, DD1 4HN

Abstract

Examination of the job-shop scheduling literature uncovers a striking trend. As methods for the deterministic job-shop problem have gradually improved over the years, they have come to rely on neighbourhoods for selecting moves that are more and more constrained. We document this phenomenon by focusing on the approach of Nowicki and Smutnicki (*Management Science*, 1996, 42(6), 797–813), noted for proposing and implementing the most restrictive neighbourhood in the literature. The Nowicki and Smutnicki (NS) method which exploits its neighbourhood by a tabu search strategy, is widely recognised as the most effective procedure for obtaining high quality solutions in a relatively short time. Accordingly, we analyse the contribution of the method's neighbourhood structure to its overall effectiveness. Our findings show, surprisingly, that the NS neighbourhood causes the method's choice of an initialisation procedure to have an important influence on the best solution the method is able to find. By contrast, the method's choice of a strategy to generate a critical path has a negligible influence. Empirical testing further discloses that over 99.7% of the moves chosen from this neighborhood (by the NS rules) are disimproving—regardless of the initial solution procedure or the critical path generation procedure employed. We discuss implications of these findings for developing new and more effective job-shop algorithms.

Key Words: neighborhood, tabu search, job-shop, scheduling

1. Introduction

A neighbourhood, $N(x)$ is a function which defines a simple transition from a solution x to another solution by inducing a change that typically may be viewed as a small perturbation (Glover and Laguna, 1997).[1] Each solution $x' \in N(x)$ can be reached directly from x by a

*Present address: School of Manufacturing and Mechanical Engineering, University of Birmingham, UK, B152 TT, UK. Email: s.meeran@bham.ac.uk

single predefined partial transformation of x called a move, and x is said to move to x' when such an transition is performed. (The term *solution* is conceived in the sense of one that satisfies certain structural requirements of a problem at hand, but it is not necessary that all requirements qualify as feasible.) In the majority of neighbourhood searches a symmetry is assumed to hold where x' is a neighbour of x if and only if x is a neighbour of x'. A neighbourhood function, more formally, is a mapping $N : \mathcal{R} \rightarrow 2^{\mathcal{R}}$ which associates a set of solutions $\mathcal{R}(x)$ with each solution $x \in \mathcal{R}$ obtained by a move (Glover and Laguna 1997). The objective of these strategies is to progressively perturb the current configuration through a succession of neighbours in order to direct the search to an improved solution. Improvement is sought at each step by standard *ascent* methods, or in some (possibly) larger number of steps by more advanced methods. In cases where solutions may involve infeasibilities, improvement is often defined relative to a modified objective that penalises such infeasibilites.

This work focuses on the application of neighbourhood search to the problem of job shop scheduling. The deterministic job shop (DJS) problem provides a useful domain for analysis because it is an important model in scheduling theory and is known to be extremely difficult to solve. It serves as a proving ground for new algorithmic ideas and is strongly motivated by practical requirements. In addition a great deal of previous research has been done that provides a basis for comparisons among alternative approaches, old and new.

1.1. The deterministic job shop problem

The DJS problem considered here is of the type given in French (1982), which consists of a finite set $\mathcal{J}$ of n jobs $\{\mathcal{J}_i\}_{i=1}^{n}$ to be processed on a finite set $\mathcal{M}$ of m machines $\{\mathcal{M}_k\}_{k=1}^{m}$. Each job $\mathcal{J}_i$ must be processed on every machine and consists of a chain or complex of m_i operations $O_{i1}, O_{i2}, \ldots, O_{im_i}$, which have to be scheduled in a predetermined order, a requirement called a *precedence constraint.* There are N operations in total, $N = \sum_{i=1}^{n} m_i \cdot O_{ik}$ is the operation of job $\mathcal{J}_i$ which has to be processed on machine $\mathcal{M}_k$ for an uninterrupted processing time period τ_{ik} and no operation may be pre-empted, i.e., interrupted and then completed at a later date. (However a simpler notation is used in this paper by allocating each operation with a unique number from 1 to (nm). The following scheme is applied in this indexing of operations. The first operation of $\mathcal{J}_1$ is 1, the first operation of $\mathcal{J}_2$ is 2, ... the first operation of $\mathcal{J}_n$ is n, the second operation of $\mathcal{J}_1$ is $(n+1)$, ... and the last operation of $\mathcal{J}_n$ is (nm). More formally if $\mathcal{M}_k$ is the kth machine required by O_{ik}, with respect to precedence constraints, then its operation number is $n(k-1)+i$ (c.f. figure 1).) Each job has its own individual flow pattern through the machines which is independent of the other jobs. The problem is defined by *capacity constraints* which stipulate that each machine can process only one job and each job can be processed by only one machine at a time. The duration in which all operations for all jobs are completed is referred to as the makespan $C_{\max}$. The objective of the scheduler is to determine starting times for each operation, $t_{ik} \geq 0$, in order to minimise the makespan while satisfying all the precedence and capacity constraints. That is, the goal maybe expressed as determining $C^*_{\max}$, where

$$C^*_{\max} = \min(C_{\max}) = \min_{\text{feasible schedules}} (\max(t_{ik} + \tau_{ik}) : \forall \mathcal{J}_i \in \mathcal{J},\ \mathcal{M}_k \in \mathcal{M}).$$

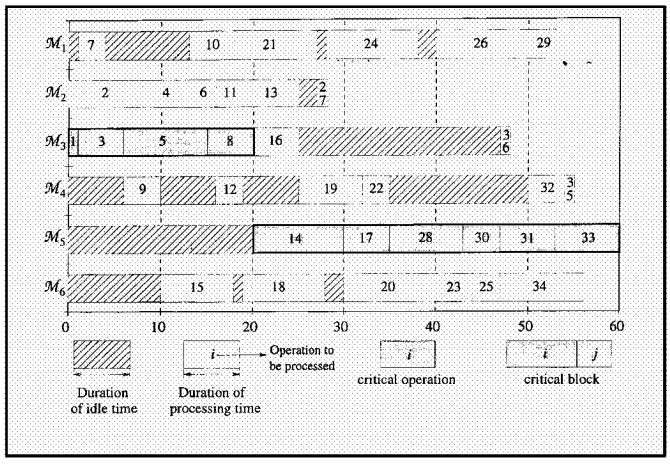

Figure 1. Illustration of important concepts.

Note the dimensionality of each Π_j instance is specified as $n \times m$. N is often assumed to be nm provided that $m_i = m$ for each job $\mathcal{J}_i \in \mathcal{J}$ and that every job has to be processed exactly once on each machine. In a more general statement of the job-shop problem, machine repetitions (or machine absence) are allowed in the given order of the job $\mathcal{J}_i \in \mathcal{J}$, and thus m_i may be greater (or smaller) than m. The main focus of this work is given to the case of $m_i = m$-non pre-emptive operations per job $\mathcal{J}_i$, unless otherwise stated.

A great deal of research has been focused on solving the DJS problem, over the last forty years, resulting in a wide variety of approaches. Recently, much work has been directed on hybrid methods to solve DJS, as a single technique cannot solve this stubborn problem. This has concentrated the research effort on techniques that combine myopic problem specific methods, which apply a neighbourhood scheme, and a meta-strategy which guides the search out of local optima. Such hybrid techniques are known as iterated local search algorithms or meta-heuristics and the approaches of Thomsen (1997), Balas and Vazacopoulos (1998) and Jain (1998) are currently providing the best results. A detailed review of local search neighbourhoods and meta-heuristics is given in Blazewicz et al. (1996) and Jain and Meeran (1999).

As algorithms for the DJS problem have improved, the neighbourhoods that they apply have placed more and more importance on trying to achieve an appropriate balance of multiple factors that lead to improvement (Mattfeld, 1996). Our work specifically analyses the contribution of highly restrictive neighbourhoods. To do this, we focus attention on the technique of Nowicki and Smutnicki (1996) (NS) which exploits the most restrictive neighbourhood with a tabu search strategy. We examine the outcome of applying the NS approach by initiating it with several different methods for generating starting solutions. Building on this, we further examine the effect of varying the NS strategy for generating the critical path. We also analyse the contribution of other features of the algorithm, including the types of moves it employs. Finally, we discuss the possible implications of these findings for developing new and more effective algorithms.

1.2. *Important concepts*

Some of the important concepts and terms for solving the DJS problem are as follows. Figure 1 depicts a trial solution for an illustrative problem called FT 06 (Fisher and Thompson, 1963) which contains six jobs and six machines. This problem will be the model for a series of subsequent illustrations. The trial solution depicted in figure 1, is referred to as a *lexicographic sequence*, because the operations are scheduled in order of their index. For FT 06 the lexicographic operation sequence is $< 1, 2, \ldots, 17, 18, 19, \ldots, 35, 36 >$ (c.f. figure 1).

A key component of a DJS solution is a *critical path*, which is a longest route through the operations and whose length represents the total makespan of the schedule. In figure 1 the length of the critical path is 60. Any operation on the critical path is called a *critical operation*. For any two adjacent critical operations i and j we may assume without loss of generality that if i precedes j ($i \prec j$) then the finish time of operation i is exactly equal to the start time of operation j. Figure 1 illustrates a critical path consisting of the operations $\{1, 3, 5, 8, 14, 17, 28, 30, 31, 33\}$.

It is also possible to decompose the critical path into a number of blocks. A block is a maximal sequence of adjacent critical operations that require the same machine. In figure 1 the critical path is divided into two blocks, b_1 and b_2, that are processed on machines three and five respectively. Block b_1 consists of the operations $\{1, 3, 5, 8\}$ while b_2 is made up of the operations $\{14, 17, 28, 30, 31, 33\}$. In the DJS problem each operation, i, has two immediate predecessors and successors, its job and machine predecessor ($\mathcal{JP}_i$ and $\mathcal{MP}_i$) and its job and machine successor ($\mathcal{JS}_i$ and $\mathcal{MS}_i$). For operation 17 in this example, the respective job and machine predecessors and successors are: $\mathcal{JP}_{17} = 11$, $\mathcal{MP}_{17} = 14$, $\mathcal{JS}_{17} = 23$ and $\mathcal{MS}_{17} = 28$. (Note that as the example problem has 6 jobs then the job successor of operation j has an index of $j + 6$, while its job predecessor has an index of $j - 6$ (c.f. §1.1).)

2. Constrained DJS neighbourhoods

A detailed historical sketch of DJS neighbourhoods is provided in Jain (1998). In this section we highlight some key points of that review. The first major contribution is provided by Van Laarhooven et al. (1988, 1992). Their neighbourhood structure consists only of those moves that are achieved by reversing the processing order of an adjacent pair of critical operations, subject to the condition that these operations must be processed on the same machine.[2] Further refinements have been provided by Grabowski et al. (1988) and Matsuo et al. (1988).

The most recent neighbourhood definition is by Nowicki and Smutnicki (1996) and is also based on the concept of blocks as in the case of Grabowski et al. (1988) and Matsuo et al. (1988). In this neighbourhood only one critical path is generated. Given b blocks, if $1 < l < b$, then swap the first two operations on the last block and the last two operations on the first block. However, if $l = 1$ swap only the last two operations in this block, on the other hand if $l = b$ swap the first two operations. (In the case where the first and/or last block contains only two operations then these operations are swapped. If a block contains only one operation no swap is made.) Figure 2 illustrates the NS neighbourhood of moves. As the application of different neighbourhoods leads to different collections of paths through

Table 1. The number of moves defined by each of the various neighbourhoods for FT 06.

Neighbourhood	Number of possible moves
General	20
Van Laarhooven et al. (1988, 1992)	15
Grabowski et al. (1988)	11
Matsuo et al. (1988)	10
Nowicki and Smutnicki (1996)	2

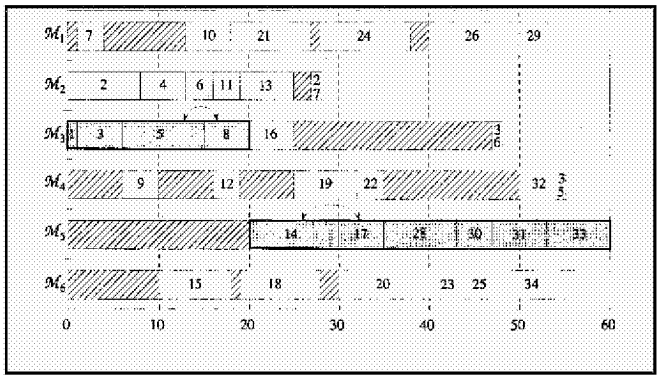

Figure 2. Neighbourhood defined by Nowicki and Smutnicki (1996).

the search space an appropriate selection becomes very important. Table 1 suggests that one of the reasons that algorithms have become more effective as well as efficient is that the neighbourhoods they define are becoming more constrained, without losing the ability to provide access to improved solutions. It also suggests that swap neighbourhoods for the DJS problem cannot be restricted any further if this ability is to be retained. The NS neighbourhood is substantially smaller than the other neighbourhoods and in this example contains only 10% of the moves defined in the most general neighbourhood, which consists of swapping an adjacent pair of operations on the same machine. The next section describes the NS algorithm, currently one of the best DJS techniques, which applies this neighbourhood as the core of its tabu search strategy. It is clearly evident that the speed of this algorithm owes significantly to the definition of its neighbourhood.

3. The technique of Nowicki and Smutnicki (1996)

The approach of Nowicki and Smutnicki (1996) exploits the neighbourhood previously described by means of a tabu search intensification strategy that couples recency-based memory with the recovery of elite solutions—an approach that provides an intensified search in the vicinity of good local optima while inducing the search to pursue different trajectories. Figure 3 provides a general overview of this procedure. In the remainder of this section a fuller description of each component function is provided.

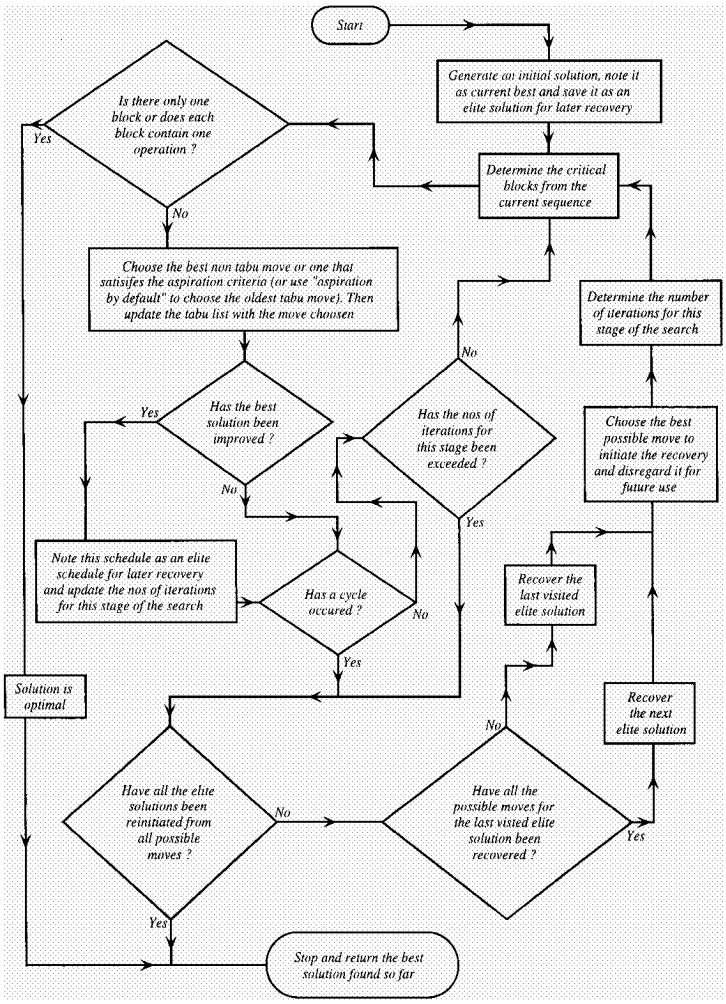

Figure 3. Generic flowchart of the NS algorithm.

3.1. Initial solution

The initial solution can be generated in a number of ways and several different techniques are presented in the next section. The actual approach used in the NS method is based on the insertion algorithm of Werner and Winkler (1995).

3.2. Move selection

The NS approach generates a single arbitrary critical path based on a "tightest predecessor" scheme. In this scheme if $\mathcal{JP}_i$ and $\mathcal{MP}_i$ of a critical operation, i, are also critical then

Nowicki and Smutnicki choose the predecessor (from among these two alternatives) which appears first in the operation sequence. If the neighbourhood procedure only generates one critical block (i.e. all critical operations require the same machine) or if each block consists of only one operation (i.e. all operations belong to the same job) then the solution is optimal and the algorithm is stopped. This property allows the NS method to verify the optimality of the solutions it obtains for about 20% of the DJS benchmark problems.

The interesting estimation scheme proposed by Taillard (1994) which is used in some DJS procedures, and which provides a lower bound on improving moves, is not employed in the NS approach but is replaced by the more costly (but more accurate) evaluation that calculates the full makespan for each move. The choice rule of the NS method is then to select the non tabu move with the lowest makespan. The aspiration criterion used by the approach allows a move to be selected inspite of being tabu if its makespan is lower than that of the current best solution found so far.

3.3. Tabu list

The move selected at each step replaces the oldest move in the tabu list. More precisely, if the move consists of the swap $i - j$, then a swap of these same two elements is not permitted for the duration that the move is recorded as tabu. It is relevant to point out that such a definition of tabu status based on attributes of moves, rather than on attributes of solutions, is normally to be avoided, because it can permit cycling. However, because of the restricted nature of this NS neighbourhood, the restriction is equivalent to preventing specific attributes of the previous solution from being duplicated, and therefore the indicated definition is acceptable. A situation can arise where all possible moves are tabu and none of them are able to satisfy the aspiration criteria. If this is the case then the procedure uses a variant of the standard "aspiration by default" criterion that chooses the oldest member of the tabu list, as outlined in figure 4. (Although the NS procedure uses a fixed tabu list size as a basis for determining tabu tenure, the tenure can change dynamically on this particular iteration.)

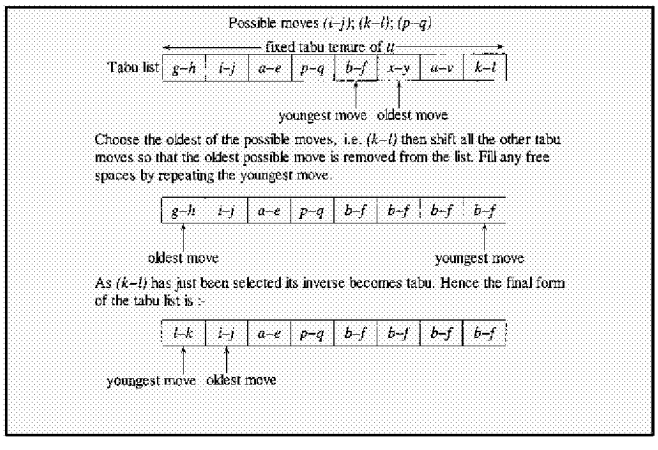

Figure 4. NS procedure when all possible moves are tabu.

3.4. *Saving solutions for elite solution recovery*

Whenever the current solution is better than the best solution obtained so far, the current solution is considered an elite solution and its attributes are saved for later recovery. Every time a new best solution is found the algorithm is run for a further *Maxiters* iterations at the current stage. (*Maxiters* is a parameter defining the number of iterations the algorithm is run after an elite solution is found.). To avoid cycling and encourage a different search trajectory the move that currently initiates the search from this best solution is forbidden to be used at a later stage to reinitiate the search from this elite solution.

3.5. *Cycle check*

After each move a check is made to ensure that the method does not cycle. A cycle is a situation in which the same set of moves is continually being repeated, i.e. the search is continually following the same path. A pseudo code of the cycle check procedure used in the NS method is provided in figure 5, where *Maxc* and *Maxd* are two parameters defining the amount of checking performed by this procedure. They are chosen to provide a balance between the amount of search performed and ensuring the existence of a cycle.

3.6. *Recovery of an elite solution*

In our prior encounter with an elite solution the search was constrained by the tabu and cycle restrictions in place at that time. When we return to the solution through the mechanism

```
Algorithm –   Cycle check

Parameters:   cycle_check []   =  Holds the most recent maxd makespans in a circular list
              cycle_position   =  The position in cycle_check to insert the next makespan
              cycle_gap        =  The difference in position within cycle_check of two matching
                                  makespans
              cycle_match      =  The number of times two makespans in cycle_check have
                                  matched

begin
    if the best solution has improved or a backtrack has been applied then
        cycle_position = cycle_gap = cycle_match = 0;
        begin for i := 1 to maxd do
            cycle_check [i] = 0;
        end
    cycle_check [cycle_position] = current makespan;
    if (cycle_gap ≠ 0) then
        if cycle_check [cycle_position + cycle_gap] = current makespan then
            cycle_match := cycle_match + 1;
        if (cycle_match >= maxc*maxd) then
            return to the main algorithm and indicate a cycle has occured;
        else    return to the main algorithm and indicate no cycle has occured;
    cycle_match = 0;
    begin for i := 1 to maxd do
        if cycle_check [cycle_position + i] = current makespan; then cycle_gap = i;
    end
    return to the main algorithm and indicate no cycle has occured
end
```

Figure 5. Pseudo algorithm for the procedure applied to detect cycles.

of elite solution recovery and perform a recombination, all such restrictions are removed, so that the search may proceed unfettered by constraints that may have disallowed a search direction that eventually leads to a better and possibly optimal solution.

If the best solution was obtained more than *Maxiters* iterations ago or a cycle has occurred then the search recovers the best elite solution, l, where $0 \leq l < Maxelite$. (*Maxelite* is the maximum number of elite solutions stored.) Recovery is applied for $Maxiters - (Recovery\_iters * [Maxelite - l])$ iterations or until a cycle is detected, where *Recovery_iters* is a parameter defining the length of the recovery procedure.

3.7. Parameter selection

Only one run of the NS algorithm is performed and the parameter values are as those chosen by Nowicki and Smutnicki (1996) for the algorithm that they call TSAB, i.e. $tt = 8$; $Maxiters = 2500$; $Maxelite = 5$; $Recovery\_iters = 400$; $maxc = 2$ and $maxd = 100$. The algorithm stops when all elite solutions on the list have been reinitiated from all possible moves or the solution is proved to be optimal because of the number and size of the critical blocks generated.

4. Initial solution methods

In this section several initial solution methods are described and their performance is evaluated. The NS method fixes all the disjunctions, except those concerning the operations to be swapped, and thereby prevents any global left shifts from occurring. Nevertheless, we consider the effect of global left shifts on these initiation techniques. A global left shift occurs when an operation can be moved earlier in the job sequence without delaying any other operation (Giffler and Thompson, 1960). If global left shifts are prevented as in the NS method, there is no guarantee of finding an optimal solution. However, such a prevention reduces the chance of quickly becoming stuck in strong local optima. This is highlighted for the problem FT 06. Without global left shifts the optimum solution of 55 is found within a few moves. However if global left shifts are permitted then the solution quickly reaches a local optimum at 57 and is unable to easily escape from it. Only after several recovery step is the optimum of 55 eventually achieved. With some parameter settings the optimum solution of 55 can never be found. In addition, perhaps surprisingly, the exclusion of global left shifts produces a neighbourhood with a larger number of moves than if such shifts are allowed. These results were first highlighted by Baker (1974) who indicates that, with an objective of $C_{\max}$, heuristic techniques that produce semi active schedules perform better than those that produce active schedules (which may seem counter intuitive). The reason for this is that heuristic methods proposed at this time contained no or very poor diversification strategies, and were therefore unable to escape from strong local minima.

The first initialisation technique we analyse is a simple lexicographic solution (*lexico*) (c.f. §1.2). We also apply a selection of priority dispatch rules. Priority dispatch rules are based on the technique of Giffler and Thompson (1960) where each operation is given

Table 2. Preference assignments of the 10 priority dispatch rules applied in the initial solution analysis.

Rule	Priority assignment
Spt/Twkr	Smallest ratio of the processing time to total work remaining
Lrm	Longest remaining work excluding the operation under consideration
Mwkr	Most work remaining to be done
Mopr	Most number of operations remaining
Spt/Twk	Smallest ratio of processing time to total work
Fcfs	The operation that arrived the earliest is processed first (first come first served)
Lso	Longest subsequent operation
Spt	Shortest processing time—Also known as Sio (Shortest imminent operation) rule
Fhalf	More than one half of the total number of operations remaining, i.e. first half preferred
Ninq	Next operation is on the machine with the fewest number of operations waiting

a priority and the objective is to select the operation with the greatest priority from a set of conflicting operations. Despite the fact that many priority dispatch rules are available (Panwalker and Iskander, 1977) only ten rules have been applied here. They have been selected based on the results of Chang et al. (1996), who prove that on their 3750 problem test bed these ten rules, out of 42 analysed, perform best with respect to the criterion of makespan minimisation.[3] The ten rules selected are presented in Table 2 in order of decreasing performance as proved by Chang et al. (1996). Also included in the analysis is a rule that selects an operation from the conflict set at random (*random*) and a technique that randomly combines all of these rules (*combo*). Due to the nature of these heuristics, each rule has been run ten times on each problem in order to get valid results. *Combo* has been applied twice. The first application employs *combo* in the same way as all the other rules, for ten runs ($combo_{10}$), while the second application employs *combo* as in Lawrence (1984) by generating 200 non improving runs ($combo_{200}$).

The final initial solution method analysed is the insertion algorithm (*insert*) of Werner and Winkler (1995). Table 3 presents the average results of these initial solution methods when left shifts are not considered, and Table 4 presents corresponding results when left shifts are considered. The mean relative error (MRE) is calculated from the best known lower bound (LB_{BEST}), and the upper bound (UB_{SOL}), which is given by the makespan of the best solution found by the particular method, using the "relative deviation" formula $MRE = (UB_{SOL} - LB_{BEST})/LB_{BEST}$. All experiments have been performed on the complete test bed of 242 DJS benchmark problem using an IBM RS/6000 − 390 with 256 Mb of RAM running AIX 3.2 for OS2. To allow algorithmic comparisons between the results presented here and those of other methods, the Computer Independent CPU time (CI-CPU) (Vaessens et al. 1996) is given and is based on our interpretations of the work of Dongarra (1998). Here $TF \times CPU = CI\text{-}CPU$, where TF is the transformation factor. Note that the TF of this machine is estimated to be 19.

Due to the deterministic nature of *lexico* and *insert* only one run of each of these heuristics is required. Taking *mwkr* as an example, when left shifts are allowed the deviation of the best solution achieved over ten runs averages 28%, while the deviation of the average solution

Table 3. Results of the initial solution methods when left shifts are not allowed.

	MRE (%)				Time				Runs	
Method	Best	μ	E_Best	N_Best	*B_Run*	μ	Σ	Σ *CI-CPU*	Best	Σ
lexico	*51.985*	51.985	2	0	*0.051*	*0.051*	0.051	0.977	*1*	*1*
Spt/Twkr	32.069	36.684	3	0	0.018	0.025	0.253	4.808	*4.868*	*10*
Lrm	28.066	30.755	2	0	0.018	0.025	0.251	4.763	*4.281*	*10*
Mwkr	29.469	31.984	1	0	0.019	0.025	0.249	4.724	*4.252*	*10*
Mopr	30.935	35.204	8	0	0.020	0.026	0.261	4.960	*5.467*	*10*
Spt/Twk	50.786	60.295	0	0	0.018	0.024	0.245	4.649	*5.025*	*10*
Fcfs	33.365	34.622	4	0	0.018	0.025	0.247	4.699	*3.178*	*10*
Lso	49.856	59.714	0	0	0.017	0.025	0.250	4.754	*5.421*	*10*
Spt	53.545	64.077	0	0	0.016	0.025	0.248	4.713	*5.062*	*10*
Fhalf	30.935	35.204	8	0	0.020	0.026	0.260	4.949	*5.467*	*10*
Ninq	52.228	64.310	0	0	0.019	0.026	0.262	4.983	*5.368*	*10*
Random	40.850	51.854	0	0	0.019	0.025	0.254	4.821	*5.632*	*10*
$combo_{10}$	31.262	38.565	4	0	0.020	0.027	0.273	5.186	*5.893*	*10*
$combo_{200}$	26.331	38.623	7	0	0.027	0.027	9.475	180.023	143.748	343.748
insert	*22.392*	22.392	4	0	*5.802*	*5.802*	5.802	110.242	*1*	*1*

μ = mean; Σ = Total; *E_Best* = The number of times the solution equals the best known upper bound; *N_Best* = The number of times the solution improves the best known upper bound; *B_Run* = Time taken by the run that found the best solution.

Table 4. Results of the initial solution methods when left shifts are allowed[a].

	MRE (%)				Time				Runs	
Method	Best	μ	E_Best	N_Best	*B_Run*	μ	Σ	Σ *CI-CPU*	Best	Σ
lexico	*35.122*	35.122	3	0	0.055	0.055	0.055	1.036	*1*	*1*
Spt/Twkr	28.487	32.434	3	0	0.020	0.028	0.281	5.347	*4.847*	*10*
Lrm	26.774	28.986	2	0	0.021	0.028	0.284	5.403	*4.521*	*10*
Mwkr	28.216	30.405	2	0	0.022	0.028	0.281	5.347	*4.240*	*10*
Mopr	29.765	33.771	8	0	0.023	0.030	0.297	5.640	*5.145*	*10*
Spt/Twk	39.582	46.368	0	0	0.021	0.027	0.269	5.103	*5.017*	*10*
Fcfs	32.402	33.564	4	0	0.022	0.028	0.280	5.314	*3.145*	*10*
Lso	39.195	46.427	0	0	0.021	0.028	0.278	5.282	*5.496*	*10*
Spt	43.407	51.715	0	0	0.020	0.027	0.272	5.177	*4.901*	*10*
Fhalf	29.765	33.771	8	0	0.023	0.030	0.302	5.732	*5.145*	*10*
Ninq	41.647	51.057	0	0	0.022	0.029	0.289	5.490	*5.430*	*10*
Random	34.877	44.099	1	0	0.021	0.029	0.290	5.515	*5.326*	*10*
$combo_{10}$	28.389	34.293	4	0	0.024	0.030	0.301	5.720	*6.017*	*10*
$combo_{200}$	23.974	34.358	10	0	0.030	0.030	10.500	199.497	151.674	351.674
insert	*20.625*	20.625	6	0	5.856	5.856	5.856	111.266	*1*	*1*

[a]The key is as previous tables.

over these runs averages 30%. Two of the solutions matched the best known upper bounds while no solution is able to improve any of the best known upper bounds. The average time by the run that found the best solution is 0.022 secs with the average time of a run being 0.028 secs. Not surprisingly the average time to perform the ten runs is 0.28 secs (5.347 CI-CPU secs). On average the best solution comes at run 4.24 and the rule is run ten times on each problem.

When left shifts are allowed solution quality improves, the number of best known upper bounds matched increases and the best solution is achieved slightly faster on average. However the time required is slightly greater than when left shifts are not allowed. The effect of left shifts is most noticeable on the *lexico* solutions, where the MRE of the best solution improves from 52% to 35%. *Lexico* achieves solutions of similar quality to the dispatch rules but requires twice as long per run.

The results show that most of the priority dispatch rules are quite similar in their performance, running very fast, but achieving poor solutions. (For all practical purposes *mopr* and *fhalf* are the same.) The reason for the discrepancy between the results here and those achieved by Chang et al. (1996) is because a greater variety of problems are considered here, many of which are both larger and harder than those used in that earlier study.

The best results are clearly achieved by $combo_{200}$ and *insert*, although these techniques require substantially more CPU time, on average two orders of magnitude more, than the other methods. $Combo_{200}$ is quite costly computationally and on average takes nearly twice as long as *insert*, although it is able to match 40% more best known upper bounds. Of particular note is the number of best solutions achieved by *mopr* and *fhalf*. They are able to find eight such solutions even when left shifts are not considered. This is more than *insert* can achieve even when left shifts are considered. However *insert* is only run once on each problem while the dispatch rules are run 10 times. Nevertheless these results clearly highlight the problem specific nature of dispatch rules, disclosing that each of these rules does well on some problems and badly on others, while *insert*, by contrast, performs consistently (if not impressively) over the whole problem set. Among the dispatch rules, for example, the MRE of *mopr* and *fhalf* can sometimes be as high as 74%.

Of these techniques, we select *lexico*, *lrm*, *mwkr*, *fcfs*, *spt* and *insert* to initialise the NS method. Table 5 shows the average MRE of these techniques when applied for only one run with no left shifts, as done in the NS approach. While *lexico*, *lrm*, *mwkr* and *fcfs* are quite similar *insert* is clearly the best starting method and *spt* the worst.

Table 5. The solution quality of the initialisation methods as they are incorporated in NS[a].

	Chosen starting method to initialise the NS procedure					
	lexico	*lrm*	*mwkr*	*fcfs*	*spt*	*insert*
MRE (%)	51.985	38.278	38.993	41.726	83.546	22.392
E_Best	2	0	1	3	0	4
N_Best	0	0	0	0	0	0

[a]The key is as previous tables.

5. Analysis of NS

5.1. *Varying the initial solution*

Table 6 presents the average results on all the benchmark problems when the NS method is initiated from each of the chosen initial solutions. These results are obtained when NS is applied until *Maxiters* have been performed with no improvement in the best schedule. Here no solution recovery or cycle checking is done. Table 7 presents the average results

Table 6. Applying the ns method with various initial solutions and no solution recovery or cycle checking.

	lexico	*lrm*	*mwkr*	*fcfs*	*spt*	*insert*
MRE (%)	10.0662	9.152	9.090	9.703	13.060	7.869
E_Best	86	76	83	83	23	92
N_Best	38	37	37	36	7	46
Iter_Best	7038.731	6587.826	6619.430	7254.748	9121.401	6251.954
Σ *Iter*	9097.194	8695.595	8697.426	9321.302	11487.157	8352.322
CPU time	58.277	58.233	56.845	60.532	156.411	76.263
Time_Iter	0.00527	0.00569	0.00559	0.00581	0.00971	0.0162
Time_Best	47.835	47.135	45.871	49.780	132.593	63.270
%_Best (%)	68.583	67.308	67.651	67.218	69.166	64.385
CI-CPU time	1107.250	1106.430	1080.055	1150.107	2971.802	1449.002

Iter_Best = The iteration in which the best solution is found on average for each benchmark problem; Σ *Iter* = The total number of iterations performed on average for each benchmark problem; *Time_Iter* = The time in seconds to perform one iteration on average for each benchmark problem; *Time_Best* = The time in seconds to find the best solution on average for each benchmark problem; *%_Best* = The percentage of the total time taken to find the best solution on average for each benchmark problem.

Table 7. Applying the full NS procedure with various initial solutions[a].

	lexico	*lrm*	*mwkr*	*fcfs*	*spt*	*insert*
MRE (%)	7.639	7.005	6.841	7.453	8.184	6.190
E_Best	106	106	110	105	71	123
N_Best	46	49	55	46	42	64
Iter_Best	37102.103	37959.174	40495.781	39434.050	60724.397	37470.285
Σ *Iter*	59468.475	60706.434	63084.037	61591.971	93809.434	61781.331
CPU time	348.943	376.919	386.728	363.200	969.675	405.517
Time_Iter	0.00548	0.00571	0.00577	0.00595	0.00702	0.0157
Time_Best	228.574	252.454	262.139	245.757	676.907	264.192
%_Best (%)	61.639	60.745	60.297	60.174	58.103	57.026
CI-CPU time	6629.920	7161.468	7347.825	6900.805	18423.827	7704.815

[a]The key is as previous tables.

of the full NS algorithm, i.e. solution recovery, cycle checking, etc. are all applied. All results in this subsection are obtained using the tightest predecessor critical path approach.

For illustrative purposes consider the initial solution generated by *mwkr*. When no solution recovery is applied the average deviation of the final solution is about 9% from LB_{BEST}. 83 best known upper bounds are matched and 37 new best upper bounds are achieved. (Note all of the new best solutions achieved in this work are for the DMU benchmark problems (Demirkol et al., 1998) which have not as yet been exposed to the full set of DJS algorithms.) The best solution is found on average at iteration 6619 and the total number of iterations is 8697 on average. Each problem is solved in 57 secs (1080 CI-CPU secs) on average and the average time to perform one iteration is 5.59 ms. The average time to achieve the best solution is 46 secs and on average 68% of the total time is required to find the best solution. The discrepancy between this percentage and the ratio of *Time_Best* to *CPU time* is due to the fact that about 20% of problems are proved to be optimal by the termination criteria, and 70% of these latter problems are solved within a second.

Without solution recovery, the solution quality deviates from optimality by 8 to 13%, while with solution recovery the deviation is improved to be from 6 to 8%. The results suggest that solution recovery reduces the deviation from optimality, but in general the quality of the final solution appears to be more strongly determined by the initial solution. (However, as we will see, this picture is slightly misleading, because solution recovery appears able to compensate somewhat for a situation where an initial solution is not very good.) *Spt* is clearly the poorest method (nearly four times worse than *insert* and twice as bad as the remaining techniques). Excluding *insert* the other techniques have similar MRE values and this perhaps explains why they produce similar results when used as starting procedures for the NS method. *Lexico* is the weakest of these four methods, according to its MRE values, and also yields slightly poorer final results for NS in comparison with *mwkr*, *fcfs* and *lrm*. *Insert* is clearly the best initial technique with respect to solution quality. When applied with solution recovery it is able to attain the best known upper bound for over half of the benchmark problems and is able to improve the best known upper bound on 80% of the DMU instances. The percentage of the total time taken to find the best solution is less with *insert*, but the time per iteration and the total time required to perform *insert* is greater than required by the other methods.

The NS method can be considered to be composed of three phases. The initial solution phase (*ini*), the phase without recovery (*n_rec*) and the phase with recovery (*rec*). Table 8 presents the number of best solutions equalled and surpassed and the improvement in solution quality achieved by one phase with respect to another. The improvement in solution quality is determined by $(phase_i - phase_j)/phase_i$, where $MRE(phase_i) > MRE(phase_j)$.

Tables 6 through 8 illustrate that the *n_rec* phase provides substantial improvement on the results achieved from the *ini* phase in a relatively short period of time as *n_rec* is able to improve the solution quality of *mwkr* by 21% in 57 secs on average. The *rec* phase allows for considerable fine tuning to get very good solutions. However a great deal of time is required in this phase. With *mwkr*, for example, a further 1.9% improvement in solution quality requires an additional 330 secs. Therefore, depending on the requirements of the user, a trade off has to be made between the *rec* and *n_rec* phases. A user who wants quite good solutions quickly can use only the *ini* and *n_rec* phases. However if solution quality is more important than time, then all three phases can be run. It should be noted that even

Table 8. Contribution of each phase of the algorithm to the overall solution.

	(n_rec – ini)		(rec – ini)		(rec – n_rec)		% Improvement in solution quality		
	E_Best	*N_Best*	*E_Best*	*N_Best*	*E_Best*	*N_Best*	*n_rec/ini*	*rec/ini*	*rec/n_rec*
lexico	84	38	104	46	20	8	26.756	28.195	1.993
lrm	76	37	106	49	30	12	20.609	22.040	1.829
mwkr	82	37	109	55	27	18	20.960	22.444	1.924
fcfs	80	36	102	46	22	10	21.879	23.325	1.893
spt	23	7	71	42	48	36	38.189	40.772	4.048
insert	88	46	119	64	31	18	11.592	12.896	1.488

$(phase_j - phase_i)$ = The difference in the number of best upper bounds achieved between phases i and j.
$(phase_j/phase_i)$ = The percentage improvement in solution quality achieved by phase j with respect to phase i.

though *rec* only contributes an additional 1.9% improvement in solution quality for *mwkr*, this translates into matching an additional 27 best upper bounds and improving 18 of these best bounds. In general, the better the initial solution the smaller the improvement made by solution recovery, only 1.5% in the case of *insert*. The most notable contribution of *rec* with respect to *n_rec* is made when the solution is initialised by *spt*. The resulting final solution nevertheless is comparable to those obtained by the other methods. This seems to suggest that if the initial solution is poor then the application of *rec* can still get reasonably good solutions . That is, to some extent the recovery process can overcome limitations of a poor start, effectively converting it, after some number of iterations, into a good one. Hence the greater number of iterations performed by *spt* in comparison with the other methods. (The conclusions with regards to the initial solution is due to the termination criteria set by NS (c.f. §3.7). If the algorithm is run for a longer period, then solution recovery would be able to compensate for a bad start. Even under the current parameter settings the results of *spt* support this.)

To prove whether the initial solution has an effect on the overall solution the Wilcoxon Signed-Rank test shall be used. The Wilcoxon Signed-Rank text is a non parametric procedure used with two related variables to test the hypothesis that the two variables have the same distribution. It makes no assumptions about the shapes of the distributions of the two variables. This test takes into account information about the magnitude of differences within pairs and gives more weight to pairs that show large differences than to pairs that show small differences. The test statistic is based on the ranks of the absolute values of the differences between the two variables. (For more details see Lehmann 1975.) Table 9 indicates the Wilcoxon Signed-Rank values for the six initial solution methods. The results show that no two of the initial solution methods have the same distribution (although there is a 0.056 probability that the solutions from *lrm* and *mwkr* come from the same distribution). Hence it can be concluded that these six initialisation methods provide different initial solutions from which NS can commence.

Table 10 shows the results of the Wilcoxon Signed-Rank Test on the final solutions of NS when it is initiated from the six chosen methods. The solutions initiated by *lexico* and *fcfs* are the only ones that show any similarity (at a level below 0.45), and hence we conclude

Table 9. Wilcoxon Signed-Rank Test on the six initial solution methods chosen to initiate NS.

	lexico	*lrm*	*mwkr*	*fcfs*	*spt*	*insert*
lexico	1.000	0.000	0.000	0.000	0.000	0.000
lrm	0.000	1.000	0.056	0.000	0.000	0.000
mwkr	0.000	0.056	1.000	0.000	0.000	0.000
fcfs	0.000	0.000	0.000	1.000	0.000	0.000
spt	0.000	0.000	0.000	0.000	1.000	0.000
insert	0.000	0.000	0.000	0.000	0.000	1.000

Table 10. Wilcoxon Signed-Rank Test on the final solution of NS initiated from one of the six initialisation techniques

	lexico	*lrm*	*mwkr*	*fcfs*	*spt*	*insert*
lexico	1.000	0.002	0.000	0.449	0.000	0.000
lrm	0.002	1.000	0.092	0.003	0.000	0.000
mwkr	0.000	0.092	1.000	0.000	0.000	0.003
fcfs	0.449	0.003	0.000	1.000	0.000	0.000
spt	0.000	0.000	0.000	0.000	1.000	0.000
insert	0.000	0.000	0.003	0.000	0.000	1.000

that varying the initial solution does have an effect on the overall solution (given the current cutoff point for stopping the solution process). The results in Tables 6 through 8 suggest that in general the better the initial solution method (by the current termination criteria), the better the final solution of NS and the greater the number of previously known best solutions matched and surpassed. Hence as the starting solution has an effect on the overall solution, this is the reason Nowicki and Smutnicki (1996) applied *insert*.

5.2. *Varying the critical path*

This section analyses the effect of slightly varying the neighbourhood of moves on the overall NS solution when initiated from a *lexico* and an *insert* solution. If both $\mathcal{JP}_i$ and $\mathcal{MP}_i$ are critical then priority is given to 1. tightest predecessor (*t-pred*) as defined by Nowicki and Smutnicki (1996) (§3.2) 2. $\mathcal{JP}_i$ (*job*) 3. $\mathcal{MP}_i$ (*mach*) and 4. the predecessor with the smallest head (*head*). In these experiments the full NS procedure is applied, with solution recovery and cycle checking and is initiated from a *lexico* and an *insert* solution. (It should be noted that 33.9% and 36.8% of the solutions generated have multiple critical paths when initiated from *lexico* and *insert* respectively. Thereby suggesting that the method of generating the critical path has an important influence on the neighbourhood produced.)

Table 11 presents the results of the four different critical path generation techniques when initiated by *lexico* and *insert*. The *t-pred* values are as presented in Table 7. The results

Table 11. Effects of varying the critical path methods with 2 different initial solutions[a].

	Lexico initial solution				Insert initial solution			
	t-pred	*job*	*mach*	*head*	*t-pred*	*job*	*mach*	*head*
MRE (%)	7.639	7.704	7.748	7.585	6.190	6.214	6.226	6.215
rec/ini (%)	28.195	28.164	28.139	28.232	12.896	12.877	12.864	12.878
E_Best	106	107	103	107	123	119	117	118
N_Best	46	46	43	45	64	64	63	64
Iter_Best	37102.103	39660.231	38940.310	38627.388	37470.285	38207.603	35616.756	35624.818
Σ *Iter*	59468.475	62277.471	61836.120	60916.426	61781.331	63731.562	59323.112	59496.368
CPU time	348.943	355.398	354.542	357.245	405.517	421.064	391.919	400.810
Time_Iter	0.00548	0.00522	0.00524	0.00526	0.01570	0.01542	0.01555	0.01563
Time_Best	228.574	240.602	237.370	242.398	264.192	268.574	252.594	259.885
%_Best (%)	61.639	61.080	60.763	61.117	57.026	55.965	56.416	56.004
CI-CPU time	6629.920	6752.571	6736.291	6787.662	7704.815	8000.208	7446.470	7615.398

[a]The key is as previous tables.

Table 12. Wilcoxon Signed-Rank Test on the 4 critical path methods with 2 different initial solutions.

	Lexico initial solution				Insert initial solution			
	t-pred	*job*	*mach*	*head*	*t-pred*	*job*	*mach*	*head*
t-pred	1.000	0.585	0.240	0.591	1.000	0.414	0.759	0.488
job	0.585	1.000	0.763	0.818	0.414	1.000	0.947	0.989
mach	0.240	0.763	1.000	0.613	0.759	0.947	1.000	0.918
head	0.591	0.818	0.613	1.000	0.488	0.989	0.918	1.000

show that the strategy of generating the critical path has no real effect on the overall solution quality. The results of the strategies initiated by *insert* are better than those initiated by *lexico*. Although *t-pred* requires slightly more time per iteration than the other methods it can be considered the best strategy on the basis of solution quality, followed by *job*, *head* and *mach*. However due to the variable performance of these methods, this ordering is not consistent over all problems. For example in the case of LA 3 (Lawrence, 1984) the tightest predecessor technique yields a makespan of 604, while *job*, *mach* and *head* strategies produce makespans of 597, 615 and 604 respectively. Hence for this instance generating the critical path by giving preference to the job predecessor would be the preferred approach.

The statistical evidence of the effect of the critical path strategy is given in Table 12, which provides the Wilcoxon Signed-Rank Test values for the four different critical path generation methods. The results clearly indicate that there is no real effect on the final solution if either a *job*, *mach* or *head* strategy is used. For example, when initiated by *insert* the probability that the *head* and *job* solutions are similar is 0.989. The results are not so clearly defined when *t-pred* is used, due to its random nature. This is highlighted by the fact that *t-pred* and *mach* are gauged to be similar at a 0.24 level when *lexico* gives the starting solution, but are similar at a 0.76 level when *insert* gives the starting solution. However, the results of Table 11 do not indicate a significant discrepancy between *t-pred* and the other techniques, leading to the conclusion that the strategy used to generate the critical path does not materially affect the final solution.

5.3. Types of moves in the neighbourhood of NS

We evaluate moves in the neighbourhood of NS by subdividing them into three simple classifications: an improving move (*imp*), a zero valued move (*zero*) and a disimproving move (*d_imp*). If the current best solution has a makespan of x then the makespan of an improving move, zero-valued move and disimproving move is respectively less than, equal to and greater than x. Tables 13 through 15 detail the types of moves in the neighbourhoods generated from the strategies applied in Tables 6, 7 and 11 respectively.

The solutions generated without solution recovery contain slightly more improving moves on average than those generated with solution recovery, this is because only a limited area of the solution space, in the vicinty of the initial solution, has been searched. Taking *mwkr*

Table 13. Types of moves achieved when NS is applied with various initial solutions and no solution recovery or cycle checking.

	lexico	*lrm*	*mwkr*	*fcfs*	*spt*	*insert*
Imp	572.657	484.025	479.492	471.880	986.434	269.711
Zero	538.248	491.628	497.818	472.442	694.273	397.595
D_imp	60211.293	60895.397	59916.450	63659.054	128144.368	64777.219
Σ *Moves*	61322.198	61871.050	60893.760	64603.376	129825.074	65444.525
% Imp	0.934	0.782	0.787	0.730	0.760	0.412
% Zero	0.878	0.795	0.818	0.731	0.535	0.608
% D_imp	98.188	98.423	98.395	98.539	98.705	98.980

Imp = The average number of improving moves for each benchmark problem; *Zero* = The average number of zero valued moves for each benchmark problem; *D_imp* = The average number of disimproving moves for each benchmark problem; Σ *Moves* = The average number of total moves for each benchmark problem; *% Imp* = The percentage of neighbourhood moves which are improving; *% Zero* = The percentage of neighbourhood moves which are of zero value; *% D_imp* = The percentage of neighbourhood moves which are disimproving.

Table 14. Types of moves achieved when by the full NS procedure is applied with various initial solutions[a].

	lexico	*lrm*	*mwkr*	*fcfs*	*spt*	*insert*
Imp	614.831	520.116	516.579	511.545	1077.413	295.0950
Zero	721.963	665.223	647.223	632.777	1082.463	555.587
D_imp	449228.843	469234.029	489063.182	466683.818	970395.880	517179.405
Σ *Moves*	450565.636	470419.368	490226.983	467828.141	972555.756	518030.087
% Imp	0.136	0.111	0.105	0.109	0.111	0.057
% Zero	0.161	0.141	0.132	0.135	0.111	0.107
% D_imp	99.703	99.748	99.763	99.755	99.778	99.836

[a]The key is as previous tables.

as a test case when no solution recovery is applied the total number of moves performed is 60894 on average of which 0.8% are improving, 0.8% are zero valued and 98.4% are disimproving. Unlike the previous results the ratio of moves does not appear to be dictated by the initial solution as all the techniques have very similar proportions. The results of *spt* are perhaps surprising because one would expect a greater number of improving moves. This suggests that although the solution starts off poorly, with respect to solution quality, within a few moves it has reached makespan values which are comparable to the other techniques. At this early stage in the search an improving move in *spt* might reduce the makespan by several units while in the other techniques it will be the order of one or two units. *Insert* has slightly fewer improving moves in comparison with the other techniques. This is because it produces a good initial solution thereby immediately surmounting many poor local minima. Table 15 illustrates that varying the critical path has no noticeable effect on the ratio of moves.

Table 15. Types of moves achieved by varying the critical path methods with 2 different initial solutions[a].

	Lexico initial solution				Insert initial solution			
	t-pred	*job*	*mach*	*head*	*t-pred*	*job*	*mach*	*head*
Imp	614.831	614.967	610.318	617.579	295.0950	292.017	293.488	291.855
Zero	721.963	722.314	685.434	698.711	555.587	579.384	567.525	548.566
D_imp	449228.84	468580.91	463998.93	462475.94	517179.41	529060.36	489177.61	493969.16
Σ *Moves*	450565.64	469918.19	465294.68	463792.23	518030.09	529931.76	490038.62	494809.58
% Imp	0.136	0.131	0.131	0.133	0.057	0.055	0.060	0.059
% Zero	0.161	0.153	0.147	0.151	0.107	0.109	0.116	0.111
% D_imp	99.703	99.715	99.721	99.716	99.836	99.836	99.824	99.830

[a]The key is as previous tables.

The results therefore indicate that regardless of the strategy applied the proportion of improving and zero valued moves is very small and the number of disimproving moves is never less than 99.7%. This is independent of the initial solution or the critical path strategy. These ratios have been found by totalling all the *imp*, *zero* and *d_imp* moves for all the problems and then dividing each of these by Σ *Moves* for all the problems. Note that if the ratio of *imp*, *zero* and *d_imp* moves is determined for each individual problem and then an average is taken the proportions are slightly different. This is because, as indicated in §5.1 for about 20% of the problems the termination criteria applied by NS is able to prove optimality. The majority of these problems are solved in a few seconds and as a result the proportion of improving moves is very high.

6. Implications for future algorithms

In this section we briefly discuss implications of our findings for developing new and more effective DJS algorithms. As the majority of search time is concentrated in evaluating the neighbourhood and as the majority of moves are disimproving then it can be valuable to accelerate the evaluation stage of the algorithm. Fast estimation techniques that can quickly filter out moves that have a high probability of being improving can be particularly useful. One such moves are isolated, they can be evaluated fully. Fast estimation strategies have been proposed by Taillard (1994) and Dell'Amico and Trubian (1993). However the effects of these strategies are not fully understood hence their usage has been limited up till now.

The major effect on solution quality produced by the initial solution suggests that further improvements may result from methods that are able to generate even better initial solutions. With respect to *insert* this can be achieved by incorporating a beam search methodology. Currently only the best solution is carried forward to the next stage, however by applying beam search the β best solutions are carried forward to the next stage. Both Werner and Winkler (1995) and Sabuncuoglu and Bayiz (1997) indicate that quite substantial improvements can be achieved by beam search with only a moderate increase in computing effort. In addition Sabuncuoglu and Bayiz (1997) also incorporate the notion of filter width into their beam search. Instead of evaluating each node fully the α most promising solutions are evaluated fully from which the β best solutions are carried forward to the next stage. Note the sequential fan candidate list strategy and the fan and filter method in tabu search are generalisations of beam search and filtered beam search respectively.

NS is currently one of the best DJS techniques and is able to achieve good solutions. This is due to the intensification strategy it adopts. However in order to achieve further improvements intensification is not enough. A strong diversification feature is also required to direct the search to other regions of the solution space and thereby allow the intensification strategy to evaluate these regions fully. Also by incorporating diversification local minima can be transcended thereby allowing active rather than semi active schedule generation. This should hopefully improve the chances of finding the global optimum. As indicated earlier NS generate semi active schedules as they have no purposeful diversification scheme.

If a diversification strategy is to be created instead of randomly generating an initial solution and initiating the search from there, it would be more prudent to incorporate

features of the current elite schedules to find a good diversified solution from which the search can recommence. Such an idea is applied by the large step component in the large step optimisation method and Lourenço and Zwijnenburg (1996) indicate that the link between large steps and diversification strategies is an area well worth exploring.

Another promising area for diversification is the application of multiple neighbourhoods, as it is known that the most restrictive neighbourhood is not necessarily by definition the best. Such a multisystem would apply NS as the core, because it can be searched quickly and efficiently, and then some of the other neighbourhoods, described earlier, at other levels in order to provide the necessary diversification.

With respect to intensification a point of note to improve the robustness of NS and make parameter selection more automated is that instead of making the size of the recovery list constant it should be modified accordingly for harder problems and easier problems as well as for larger problems. This should improve solution quality but will definetly affect computing times.

7. Conclusions

The inherent weakness of many search procedures is that although they allow search without excess computation, they often get trapped in a region around some local optima. Their ability to breakout of such entrapments and achieve better, ideally global optima, is based fundamentally on their neighbourhood structure. This work has tracked the history of DJS neighbourhoods and shown that they cannot be restricted any further. The tabu search strategy of Nowicki and Smutnicki (NS) (1996), one of the most effective DJS techniques, has been analysed, and the results have demonstrated that given its present termination criteria the choice of initialisation procedure has an important influence on the final solution, such that a better initial solution provides better results, while the choice of strategy to generate the critical path has a negligible influence on the final solution. Even though the NS method applies a highly constrained neighbourhood which avoids many of the non improving moves defined in the neighbourhood of Van Laarhooven et al. (1992) and Matsuo et al. (1988) it is shown here that over 99.7% of the time the algorithm is just evaluating inferior moves. Based on these findings it is suggested that move estimation schemes as well as powerful diversification strategies are required in order to develop new and more effective DJS algorithms.

Acknowledgments

This work has been sponsored by the Royal Society of Edinburgh and the Carneigie Trust for the Universities of Scotland while the first author was a visiting researcher at the University of Colorado, Boulder. The authors would like to thank Dr. Nowicki for providing us with his code from which we learnt a great deal, Professor Frank Werner and Dr. Thomas Tautenhahn for helping us greatly with our implementation of *insert* and the assistance provided by Professor Fred Glover in the preparation of this work which has greatly enhanced the quality of the paper.

Notes

1. Neighbourhoods that involve more substantial changes are sometimes fabricated, e.g. by randomised processes for creating "large-step" transformations in Markov-based methods or by strategic processes for generating *influential* transformations tabu search.
2. However Kolonko (1999) proves that the connectivity property does not imply convergence to an optimum in these neighbourhoods.
3. Ninq is actually ranked as the thirteenth best rule. However it has been chosen because the rules in position 10 to 12 are based on due dates and this analysis does not consider due dates.

References

Baker, K.R. (1974). *Introduction to Sequencing and Scheduling*. New York: John Wiley.

Balas, E. and A. Vazacopoulos. (1998). "Guided Local Search with Shifting Bottleneck for Job-Shop Scheduling." *Management Science* 44(2), 262–275.

Blazewicz, J., W. Domschke, and E. Pesch. (1996). "The Job-Shop Scheduling Problem: Conventional and New Solution Techniques." *European Journal of Operational Research* 93(1), 1–33.

Chang, Y.L., T. Sueyoshi, and R.S. Sullivan. (1996). "Ranking Dispatching Rules by Data Envelopment Analysis in a Job-Shop Environment." *IIE Transactions* 28(8), 631–642.

Dell'Amico, M. and M. Trubian. (1993). "Applying Tabu Search to the Job-Shop Scheduling Problem." *Annals of Operations Research* 41, 231–252.

Demirkol, E., S. Mehta, and R. Uzsoy. (1998)."Benchmarks for Shop Scheduling Problems." *European Journal of Operational Research* 109(1), 137–141.

Dongarra, J.J. (1998). "Performance of Various Computers Using Standard Linear Equations Software." Technical Report CS-89-85, Computer Science Department, University of Tennessee, Knoxville, TN 37996–1301, Tennessee, USA.

Fisher, H. and G.L. Thompson. (1963). "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules." In J.F. Muth and G.L. Thompson (eds.), *Industrial Scheduling*. Englewood Cliffs, New Jersey: Prentice Hall, Ch. 15, pp. 225–251.

French, S. (1982). *Sequencing and Scheduling—An Introduction to the Mathematics of the Job-Shop*. New York: Ellis Horwood, John-Wiley & Sons.

Giffler, B. and G.L. Thompson. (1960). "Algorithms for Solving Production Scheduling Problems." *Operations Research* 8, 487–503.

Glover, F. and M. Laguna. (1997). *Tabu Search*. Norwell, MA: Kluwer Academic Publishers.

Grabowski, J., E. Nowicki, and C. Smutnicki. (1988). "Block Algorithm for Scheduling Operations in a Job-Shop System." Przeglad Statystyczny XXXV, 1, 67–80, in Polish.

Jain, A.S. (1998). "A Multi-Level Hybrid Framework for the Deterministic Job-Shop Scheduling Problem." Ph.D. Thesis, Department of APEME, University of Dundee, Dundee, Scotland, UK, DD1 4HN.

Jain, A.S. and S. Meeran. (1999). "Deterministic Job-Shop Scheduling: Past, Present and Future." *European Journal of Operational Research* 113(2), 390–434.

Kolonko, M. (1999). "Some New Results on Simulated Annealing Applied to the Job Shop Scheduling Problem." *European Journal of Operational Research* 113(1), 123–136.

Lawrence, S. (1984). "Supplement to Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques." GSIA, Carnegie-Mellon University, Pittsburgh.

Lehmann, E.L. (1975). *Nonparametrics: Statistical Methods Based on Ranks*. San Francisco: Holden-Day.

Lourenço, H.R.D. and M. Zwijnenburg. (1996). "Combining the Large-Step Optimization with Tabu-Search: Application to the Job-Shop Scheduling Problem." In I.H. Osman and J.P. Kelly (eds.), *Meta-heuristics: Theory and Applications*. Kluwer Academic Publishers, pp. 219–236.

Matsuo, H., C.J. Suh, and R.S. Sullivan. (1988). "A Controlled Search Simulated Annealing Method for the General Job-Shop Scheduling Problem." Working Paper #03-04-88, Graduate School of Business, The University of Texas at Austin, Austin, Texas, USA.

Mattfeld, D.C. (1996). *Evolutionary Search and the Job Shop: Investigations on Genetic Algorithms for Production Scheduling*. Heidelberg, Germany: Physica-Verlag.

Nowicki, E. and C. Smutnicki. (1996). "A Fast Taboo Search Algorithm for the Job-Shop Problem." *Management Science* 42(6), 797–813.

Panwalkar, S.S. and W. Iskander. (1977). "A Survey of Scheduling Rules." *Operations Research* 25(1), 45–61.

Sabuncuoglu, I. and M. Bayiz. (1997). "A Beam Search Based Algorithm for the Job Shop Scheduling Problem." Research Report: IEOR-9705, Department of Industrial Engineering, Bilkent University, 06533 Ankara, Turkey.

Taillard, É. (1994). "Parallel Taboo Search Techniques for the Job-Shop Scheduling Problem." *ORSA Journal on Computing* 16(2), 108–117.

Thomsen, S. (1997). "Meta-heuristics Combined with Branch & Bound." Technical Report, Copenhagen Business School, Copenhagen, Denmark (in Danish).

Vaessens, R.J.M., E.H.L. Aarts, and J.K. Lenstra. (1996). "Job Shop Scheduling by Local Search." *INFORMS Journal on Computing* 8, 302–317.

Van Laarhoven, P.J.M., E.H.L. Aarts, and J.K. Lenstra. (1988). "Job Shop Scheduling by Simulated Annealing." Report OS-R8809, Centrum voor Wiskunde en Informatica, Amsterdam.

Van Laarhooven, P.J.M., E.H.L. Aarts, and J.K. Lenstra. (1992). "Job Shop Scheduling by Simulated Annealing." *Operations Research* 40(1), 113–125.

Werner, F. and A. Winkler. (1995). "Insertion Techniques for the Heuristic Solution of the Job-Shop Problem." *Discrete Applied Mathematics* 58(2), 191–211.