Deconstructing Nowicki and Smutnicki's *i*-TSAB Tabu Search Algorithm for the Job-Shop Scheduling Problem [1,2]

Jean-Paul Watson [a] Adele E. Howe [b] L. Darrell Whitley [b]

[a] *Sandia National Laboratories*
P.O. Box 5800, MS 1110
Albuquerque, NM 87185-1110 USA

[b] *Department of Computer Science*
Colorado State University
Fort Collins, CO 80523-1873 USA

Abstract

Over the last decade and a half, tabu search algorithms for machine scheduling have gained a near-mythical reputation by consistently equaling or establishing state-of-the-art performance levels on a range of academic and real-world problems. Yet, despite these successes, remarkably little research has been devoted to developing an understanding of why tabu search is so effective on this problem class. In this paper, we report results that provide significant progress in this direction. We consider Nowicki and Smutnicki's *i*-TSAB tabu search algorithm, which represents the current state-of-the-art for the makespan-minimization form of the classical job-shop scheduling problem. Via a series of controlled experiments, we identify those components of *i*-TSAB that enable it to achieve state-of-the-art performance levels. In doing so, we expose a number of misconceptions regarding the behavior and/or benefits of tabu search and other local search metaheuristics for the job-shop problem. Our results also serve to focus future research, by identifying those specific directions that are most likely to yield further improvements in performance.

Key words:
Tabu search, metaheuristics, job-shop scheduling, empirical analysis.

[1] Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000
[2] This work was partially sponsored the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-00-1-0144. The U.S.

1 Introduction and Motivation

Over the last 10 to 15 years, tabu search has emerged as a dominant algorithmic approach for locating high-quality solutions to a wide range of machine scheduling problems [1]. In the case of the classical job-shop and permutation flow-shop scheduling problems, tabu search algorithms have consistently provided state-of-the-art performance since 1993, currently by a significant margin over the competition [2,3]. However, researchers have only recently begun to analyze the underlying causes for the superior performance of tabu search on these problems [4,5] and many fundamental questions remain unanswered.

We consider the well-known $n \times m$ static job-shop scheduling problem or JSP [6] with the makespan minimization objective; we assume familiarity with the associated concepts and terminology. The current state-of-the-art approximation algorithm for the JSP is Nowicki and Smutnicki's i-TSAB tabu search algorithm [7], which is an extension of their earlier landmark TSAB algorithm [8]. In addition to the short-term memory mechanism found in all implementations of tabu search, i-TSAB is characterized by the following algorithmic components: (1) the highly restrictive *N5* move operator, [3] (2) re-intensification of search around previously encountered high-quality solutions, and (3) diversification of search via path relinking between high-quality solutions. Despite its effectiveness, almost nothing is known about how both these and various secondary components interact to achieve state-of-the-art performance, or even if all of the components are necessary. Thus, our primary goal in this paper is to determine those components of i-TSAB that are integral to its remarkable performance, and the degree to which they share the responsibility.

Our methodological approach is to first analyze a simple version of tabu search based on the *N5* operator, and then gradually extend the analysis to more complex variants that share additional key features with i-TSAB. We begin by demonstrating that the "core" tabu search metaheuristic (i.e., lacking long-term memory) does not provide any distinct advantage over other metaheuristics for the JSP based on the *N5* operator. In particular, both iterated local search and Monte Carlo sampling provide competitive performance. Next, we show that intensification and diversification can significantly improve the performance of both tabu search and the other metaheuristics we consider, such that the resulting performance is consistently competitive with i-TSAB. Finally, we analyze the relative impact of intensification and diversification on the performance of tabu search algorithms for the JSP. Here, we find that although application of either mechanism in isolation can lead to performance

Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

[3] Our notation for move operators is taken from [6].

improvements, their mutual effect is multiplicative: both components are required to achieve state-of-the-art performance.

The remainder of this paper is organized as follows. We begin in Section 2 with an overview of the TSAB and i-TSAB tabu search algorithms for the JSP. Following a discussion of the implications of the *N5* move operator for metaheuristic design, we introduce novel tabu search, iterated local search, and Monte Carlo metaheuristics for the JSP. The relative performance of these metaheuristics is then analyzed in Section 4. A general framework for augmenting these metaheuristics with long-term memory is detailed in Section 5; the impact of such memory is analyzed in Section 6. We conclude in Section 7 with an analysis of the trade-offs between intensification and diversification on the performance of tabu search, and subsequently summarize our contributions in Section 8.

2 TSAB and i-TSAB: Algorithm Descriptions

Taillard introduced the first tabu search algorithm for the JSP in 1989 [9]. Taillard's algorithm represented a remarkable combination of simplicity and effectiveness, and spurred a wave of successor algorithms, e.g., see [10] and [11]. A significant breakthrough occurred in 1993 when Nowicki and Smutnicki developed their TSAB tabu search algorithm [8], which represented a major advance in both speed and effectiveness. TSAB is perhaps best known for its ability to locate an optimal solution to Fisher and Thompson's [12] infamous 10×10 instance in less than 30 seconds on a now-primitive hardware platform.

TSAB is a relatively a straightforward implementation of tabu search, with two key exceptions. First, TSAB is based on the powerful *N5* move operator, enabling it to focus on a very small subset of feasible neighbors at each iteration. Second, TSAB is equipped with a long-term memory that is used to periodically intensify search around previously encountered high-quality solutions. Starting from a solution obtained with Werner and Winkler's greedy insertion heuristic [13], TSAB executes a core tabu search procedure and tracks the N best or "elite" solutions encountered. If a pre-specified number of iterations pass without an improvement in the best-so-far solution, TSAB then re-initiates the core tabu search procedure from one of the elite solutions.

TSAB imposes a limit on the number of intensifications performed for each elite solution e and always selects the best elite solution that is under the imposed limit; the algorithm terminates once all possible intensifications have been exhausted. TSAB uses a fixed tabu tenure, and employs a mechanism to detect the associated cyclic search behavior [1]. When a cycle is detected, TSAB immediately performs intensification around an elite solution.

Building on their initial success, Nowicki and Smutnicki introduced an enhanced version of TSAB in 2001 [14]. The algorithm, denoted i-TSAB, currently represents the state-of-the-art in approximation algorithms for the JSP and outperforms all competitors by a significant margin [7]. Search in i-TSAB proceeds sequentially in two phases. In the *initiation* phase, i-TSAB uses TSAB to construct a set or pool of n elite solutions E. Let E_0 denote the solution obtained by Werner and Winkler's heuristic. The first elite solution E_1 is generated by applying TSAB to E_0. Subsequent elite solutions E_i, $2 \leq i \leq E_n$, are generated by applying TSAB to a solution ϕ obtained via a simple path relinking procedure [15] between E_{i-1} and E_1.

Next, i-TSAB enters the *proper work* phase. Let E^* denote the solution with the lowest makespan in E, and let E^x denote the solution that is maximally distant in terms of disjunctive graph distance from E^*. A path relinking procedure, *NIS*, is used to generate a solution ϕ that is "roughly" equi-distant (see [7] for an explanation) from both E^* and E^x. TSAB is applied to ϕ, yielding E^c, which then unconditionally replaces the solution E^x in E. The process iterates until the distance between E^* and E^x drops below a pre-specified threshold. The design of i-TSAB is predicated on the existence of a "big-valley" distribution of local optima, for which there is reasonably strong empirical evidence in the JSP [14].

In contrast to most metaheuristics for the JSP – including many implementations of tabu search - both TSAB and i-TSAB are largely deterministic. The only possible [4] sources of randomness involve tie-breaking at specific points during search, e.g., when multiple equally good non-tabu moves are available. For our purposes, such strong determinism complicates cross-comparisons of the performance of i-TSAB and TSAB with that of inherently stochastic algorithms. Specifically, it is unclear whether it is reasonable to compare the results of i-TSAB with the mean or median performance of a competing stochastic algorithm, or if some more principled statistic is more appropriate, if available.

3 The *N5* Operator: Moving Beyond Tabu Search

The canonical local search neighborhood for the JSP is van Laarhoven et al.'s *N1* move operator [16]. The neighborhood of a solution under the *N1* operator consists of the set of solutions obtained by inverting the order of a pair of adjacent operations in a critical block. Many of the moves induced by *N1* operator – specifically those involving operation pairs residing completely within a critical block – cannot yield an immediate improvement in solution makespan [17].

[4] It is unclear from their descriptions whether Nowicki and Smutnicki's implementations of TSAB and i-TSAB actually use such random tie-breaking.

Nowicki and Smutnicki [8] further refine this observation by identifying two additional operation pairs whose inversion cannot reduce solution makespan: the first (respectively last) pair of operations in the critical block of machine 1 (respectively machine m), subject to restriction that the critical block size is greater than 2. These observations form the basis of the highly restrictive *N5* move operator, introduced in conjunction with the TSAB algorithm.

The *N5* operator enables TSAB to ignore a large number of provably non-improving moves, thereby significantly reducing the per-iteration run-time cost relative to tabu search algorithms based on the *N1* operator, e.g., see [18] and [11]. Intuition suggests that the *N5* operator might also significantly improve the performance of other metaheuristics for the JSP, such as simulated annealing [16] and iterated local search [19]. Yet, to date the *N5* operator has rarely been used in conjunction with metaheuristics other than tabu search (see [20] for an exception), such that this intuition remains largely unconfirmed.

The lack of broad-based studies assessing the performance of *N5*-based metaheuristics leads us to our first question regarding the effectiveness *i*-TSAB: Is it necessary to employ tabu search to achieve state-of-the-art performance? We differentiate between the long-term and short-term memory mechanisms found in tabu search, and define "core" tabu search as a tabu search lacking any long-term memory mechanism other than a simple aspiration criterion. The primary purpose of short-term memory is to guide search out of local optima [1]. By definition, all other local search metaheuristics have mechanisms to accomplish the same objective. Temporarily ignoring the issue of the effectiveness of the escape mechanism, there is then no *a priori* reason that we cannot substitute another metaheuristic for the core tabu search mechanism used by *i*-TSAB. However, in order to answer this question, it is first necessary – given the lack of previously published algorithms – to introduce one or more additional *N5*-based metaheuristics for the JSP.

3.1 Implications of N5 for Metaheuristic Design

The ability of the *N5* operator to ignore provably non-improving neighbors comes with a price: the induced search space is disconnected, such that there does not always exist a path from an arbitrary solution to an optimal solution. Thus, no metaheuristic based strictly on the *N5* operator can guarantee that an optimal solution will eventually be located, independent of the run-time. Despite the obvious drawback, the implications of this fact for metaheuristic design are poorly understood.

In preliminary experimentation with the *N5*-based metaheuristics introduced below, we observed that search invariably became restricted to a small cluster of solutions S; in no case did we observe $|S| > 10$. The topology of these clusters is such that they form a sub-graph that is isolated from the rest of the search space, i.e., once search enters the sub-graph, it can never escape. For example, consider a solution s with a single critical block of size ≥ 2 residing on machine 1, such that $|N5(s)| = 1$; the sole neighbor s' is obtained by inverting the order of the last pair of operations (o_i, o_j) in the critical block. Let $C_{max}(s)$ denote the makespan of a solution s and suppose that (1) $C_{max}(s) = C_{max}(s')$ and (2) s' also contains a single critical block of size ≥ 2 – on machine 1 – with (o_j, o_i) serving as the last two operations. Once any *N5*-based metaheuristic encounters either s or s', search will forever cycle. We refer to such isolated clusters of solutions as *traps*. Although it may appear contrived, this example and related generalizations are frequently observed in the benchmark instances we consider in Sections 4 through 7.

Given the induced traps, *N5*-based metaheuristics must be able to (1) detect when search is restricted to a trap and (2) initiate some form of escape mechanism once a trap is detected. TSAB detects traps via an explicit cycle checking mechanism. When a cycle is detected, search is immediately re-initiated from an elite solution, if available. While satisfying the aforementioned criteria, this approach entangles the concepts of trap detection and escape with the long-term memory mechanism of intensification – ultimately making it impossible to control for the effect of long-term memory on algorithm performance. To achieve effective experimental control, we consider an alternative approach to detecting and escaping from traps.

To detect a trap, we monitor the *mobility* of search over time. Mobility [21] is defined as the distance between two solutions visited some k number of iterations apart. We define the distance between two solutions as the well-known disjunctive graph distance for the JSP [22]. The metaheuristics we introduce below generally maintain high search mobility over a wide range of parameter settings. Consequently, if mobility falls below some relatively small threshold, then search is likely contained in a trap, and an escape sequence can be initiated. Instead of re-starting search from some previously encountered elite solution, we instead use the *N1* operator to apply a small perturbation to the current solution. Because the *N1* operator induces a connected search space, the mechanism is able to remain nearby the current solution while simultaneously moving search out of the trap.

3.2 The CMF Framework

The metaheuristics introduced in Sections 3.4 through 3.6 are expressed in terms of a framework that implements the trap detection and escape mechanisms and other common algorithmic features. The framework is denoted

```
function CMF<Metaheuristic T>
    param MaxIters                                  // search duration
    param MTCI                                      // mobility trap check interval
    param M_thrs                                    // mobility threshold
    param MWL_nom                                   // nominal mobility walk length
    param MWL_inc                                   // mobility walk length increment
    s = genRandomLocalOptimum()
    s* = s
    lastMTCISol=s
    declare verifyTrapEscape = false
    declare MWL_cur = 0
    T.initialize()
    for i = 1 to MaxIters do
        declare generateTrapEscape = false
        if i mod MTCI then
            declare mobility = distance(s,lastMTCISol)
            if (verifyTrapEscape == true) and (mobility > M_thrs) then
                verifyTrapEscape=false
            else if (verifyTrapEscape == true) and (mobility ≤ M_thrs) then
                MWL_cur = MWL_cur + MWL_inc
                generateTrapEscape = true
            else if (verifyTrapEscape == false) and (mobility ≤ M_thrs) then
                MWL_cur = MWL_nom
                generateTrapEscape = true
                verifyTrapEscape = true
            lastMTCISol = s
        if generateTrapEscape == true then
            s = randomWalk(s,N1,⌈MWL_cur⌉)
        else
            s = T.nextSolution(s)
        if C_max(s) < C_max(s*) then
            s* = s
    return s*
```

Fig. 1. Pseudo-code for the Core Metaheuristics Framework (CMF).

CMF for *C*ore *M*etaheuristic *F*ramework; pseudo-code is provided in Figure 1. CMF assumes the availability of a metaheuristic T, with an interface that implicitly supports both initialization (the *initialize* method) and execution of individual iterations (the *nextSolution* method). Search begins from a random local optimum, which we obtain by applying steepest-descent to a random semi-active solution; the latter is constructed using the procedure described in [20]. At the highest level, CMF simply executes MaxIters iterations of the metaheuristic T and returns the best overall solution. To facilitate trap detection, CMF monitors the mobility of search every MTCI iterations. If the mobility is not greater than some minimal mobility threshold M_{thrs}, then CMF performs a random walk under *N1* of length MWL_{nom} from the current solution; otherwise, T is used to select a neighbor of the current solution s. If the mobility is still not super-threshold after another MTCI iterations, the random walk length is increased by a factor of MWL_{inc}, and another random walk is performed. The process continues, with monotonically increasing walk lengths, until mobility is super-threshold. We note that a similar adaptive process is embodied in reactive tabu search metaheuristic [21].

3.3 Computation of Neighboring Solution Makespans

Independent of move operator, the run-time of local search metaheuristics for the JSP is typically dominated by the cost of computing the makespans of neighboring solutions at each iteration of search. To perform these computations, we implement a set of remarkably efficient and highly specialized techniques introduced by Nowicki and Smutnicki. Due to their complexity, we do not detail these algorithms here; rather, we refer the reader to [7].

3.4 The Baseline: Simple Tabu Search

The baseline algorithm in our analysis is a simple *N5*-based tabu search metaheuristic for the JSP, which we denote by STS for *S*imple *T*abu *S*earch. As in TSAB, the short-term memory in STS stores the TT_{cur} most recently swapped pairs of critical operations, enabling STS to escape from local optima by preventing the original order of these operation pairs from being quickly re-established. Following Taillard [18], the tabu list size TT_{cur} in STS is dynamically updated every TUI iterations by sampling uniformly from the interval $[\mathrm{TT}_{min},\mathrm{TT}_{max}]$. All other features of STS are borrowed from TSAB, with the exception of the cycle detection mechanism and the initial solution; in STS, search is initiated from a random local optimum supplied by CMF.

3.5 Iterated Local Search and the *N5* Operator

The second novel *N5*-based metaheuristic we introduce is based on iterated local search, or ILS [23]. ILS has been successfully applied to a wide range of NP-hard problems, achieving state-of-the-art performance in a number of cases. In contrast to other well-known metaheuristics for local search, ILS uses two move operators: a small-step operator N_s and a large-step operator N_l. Typically, greedy descent with respect to N_s is used to transform an initial solution s into a local optimum s'. The N_l operator is then used to generate a perturbation s'' of s'; greedy descent is subsequently applied to s'' (i.e., with s'' serving as s in the next iteration), and the process repeats. Ideally, N_l moves are large enough to yield s'' such that when greedy descent is applied to s'', the result is a new local optimum $s^* \neq s'$.

To date, only Lourenco has introduced implementations of ILS for the JSP [24,19], although Jain [25] used the notion of a large-step operator in a complex multi-level search framework for the JSP. Despite promising initial results, Lourenco's algorithms in particular and ILS algorithms for the JSP in general have received little attention from the broader research community.

In ILS, the strength of the perturbations required by the N_l operator is a function of attractor basin strength in the problem under consideration. In the JSP, attractor basins of local optima are surprisingly weak; Watson [26] shows that the local optima of random JSP instances can be escaped with high probability simply by accepting one or two dis-improving moves and re-initiating greedy descent. This result provides a straightforward operational definition for an N_l operator, which in turn forms the basis for a new *N5*-based ILS metaheuristic for the JSP. The resulting metaheuristic is denoted IJAR, for *I*terated *J*ump *A*nd *R*edescend.

Search in IJAR proceeds via a sequence of iterations, starting from a random local optimum supplied by CMF. In a typical iteration, IJAR accepts a random sequence of at most k monotonically dis-improving neighbors from the current solution s, where k is a user-specified parameter. The "at most" qualifier is required to handle rare circumstances in which it is not possible to accept k dis-improving moves from a local optimum, e.g., when the optimum is both locally minimal and maximal. Randomized next-descent is then used to transform the resulting solution s' into a local optimum s'', which then serves as the s in the next iteration. If search consistently locates new local optima, IJAR continues to perform nominal iterations. However, in practice IJAR occasionally encounters local optima for which the ascent-descent mechanism either cannot guarantee escape or does so with only very low probability. Consequently, IJAR is equipped with mechanisms to both detect these situations and initiate an appropriate recovery.

To detect search stagnation, IJAR analyzes the makespan of solutions generated OTCI (Optima Trap Check Interval) iterations apart. Let s and s^- respectively denote the current solution and the solution observed OTCI iterations prior. If $C_{max}(s) = C_{max}(s^-)$, search is likely either permanently trapped in a local optimum or has a very low probability of escape. In this case, IJAR enters an optimum escape mode or OEM. While in OEM, random walks – with respect to the *N5* operator – of monotonically increasingly length are substituted for the usual ascent-descent process. The initial length and growth rate of the random walk are respectively determined by the OWEL$_{nom}$ and OWEL$_{inc}$ parameters, respectively. A fully detailed description of IJAR, including pseudo-code, is provided in [26]. We conclude by observing that although IJAR is strictly a variant of ILS, it also exhibits strong similarities to Hansen and Jaumard's steepest-ascent/mildest-descent metaheuristic [27]. Further details regarding the design and development of IJAR are reported in [26].

3.6 *Monte Carlo Sampling and the N5 Operator*

The third novel *N5*-based metaheuristic we introduce is based on Monte Carlo or Metropolis sampling, which serves as the basis of the well-known simulated annealing metaheuristic [28]; run temperatures (see below) are fixed in Monte Carlo sampling, while they are variable in simulated annealing. Most computational comparisons of metaheuristics for the JSP conclude that simulated annealing is not competitive with tabu search [29,30]. Specifically, although simulated annealing is capable of locating high-quality solutions, tabu search can generally locate equally good solutions in far lower run-times. However, we hypothesize that it may be possible to exploit the relative weakness of local optima attractor basins in the JSP to develop high-performance metaheuristics based on Monte Carlo sampling. In particular, we observe that if attractor basins are consistently weak (in the specific sense described above in Section 3.5), simulated annealing is highly inefficient, as search at all but low temperatures is likely unnecessary. Monte Carlo sampling at a fixed low temperature may be sufficient to yield effective search.

We denote our Monte Carlo metaheuristic by RMC, for *R*ejectionless *M*onte *C*arlo. RMC begins from a random local optimum supplied by CMF. In any given iteration of RMC, the well-known Metropolis [31] acceptance criterion is applied to each *N5* neighbor of the current solution s in turn; the visitation sequence is randomized. If a neighbor is accepted, the corresponding move is immediately performed. If none of the neighbors is accepted, then a single neighbor is accepted according to a weighting scheme defined by the Metropolis acceptance criterion. Let s denote the current solution, and let s' denote a neighbor $N5(s)$ of s. We define $\Delta = C_{max}(s') - C_{max}(s)$. Under the Metropolis criterion, the probability $P(s'|s)$ of accepting a move from s to s' is given as

$$P(s, s') = \begin{cases} e^{\frac{-\Delta}{T}}/L & \text{for } \Delta > 0 \\ 1/L & \text{for } \Delta \leq 0 \end{cases}$$

where $L = |N5(s)|$. A single neighbor s' is then accepted according to the probabilities given by $P(s, s')/\sum_k P(s, s_k)$. This acceptance scheme forms the basis of rejectionless Monte Carlo sampling [32]. We intentionally avoid a pure rejectionless acceptance criterion in order to avoid evaluation of the full neighborhood of s at each iteration of search.

The sole parameter in RMC is the search temperature T. Instead of directly specifying T, we use a simple reverse-annealing procedure to determine the lowest temperature T for which the proportion of strictly "uphill" (i.e., disimproving) moves that are accepted is at least UAG $\in$ [0..1]. The procedure we use was introduced by Yamada et al. [33], although the general idea was developed much earlier, e.g., see [34]. In general, UAG is empirically far less problem-dependent than T. Fixed values of UAG yield reasonable performance over a wide range of problem instances, while T must be tuned to achieve similar performance levels. A fully detailed description of RMC, including pseudo-code, can be found in [26].

4 Experiment 1: The Impact of the Core Metaheuristic

We now quantify both the absolute and relative performance of STS, IJAR, and RMC on a range of widely studied benchmark instances. Our goal is to answer two specific research questions: (1) Are other metaheuristics competitive with tabu search on the JSP, or is there an intrinsic advantage to the use of tabu search as the core metaheuristic in i-TSAB? and (2) To what degree do i-TSAB's long-term memory mechanisms improve the performance of the core tabu search metaheuristic? As in Section 3, we define a "core" metaheuristic as a metaheuristic whose components are strictly dedicated to escaping local optima, i.e., it lacks long-term memory mechanisms. Both questions have received little or no attention in the literature. The question of whether other metaheuristics can achieve competitive performance is currently open due to the lack of *N5*-based alternatives to tabu search. Similarly, reliable performance statistics for a core *N5*-based tabu search algorithm have not been previously reported; in particular, Nowicki and Smutnicki [7] only contrast the performance of i-TSAB and TSAB.

4.1 Experimental Methodology

We quantify the performance of each algorithm relative to a subset of Taillard's benchmark instances for the JSP, specifically those labeled `ta01` through `ta50` [35]. This subset contains 10 instances of each of the following problem sizes: 15×15, 20×15, 20×20, 30×15, and 30×20. The optimal makespans are known for 17 of these 50 instances. However, many of these "solved" instances are by no means easy, as even state-of-the-art algorithms often have difficulty consistently locating optimal solutions. Taillard's 20×20 and 30×20 instances are widely regarded as the most difficult random (i.e., as opposed to workflow or flowshop) JSP benchmarks available. With the exception of `ta62`, all three algorithms can quickly locate optimal solutions to Taillard's larger benchmark instances. Consequently, we do not consider these instances in our analysis.

Independent of problem size, our implementations of STS, IJAR, and RMC each devote approximately 98% of the total run-time to computing the makespans of neighboring solutions (see Section 3.3). The code for performing such

Table 1
Run-time cost-per-iteration multipliers for IJAR and RMC relative to STS.

	Problem Size				
	15×15	20×15	20×20	30×15	30×20
IJAR	0.33	0.33	0.31	0.30	0.31
RMC	1.37	1.40	1.41	1.37	1.43

computations is shared by all three algorithms. Consequently, by limiting trials of each algorithm to either a fixed CPU time or a time-equivalent number of iterations, reasonably fair comparisons can be achieved: any (in)efficiencies present in our codes are transmitted to all algorithms. Further, most algorithmic or implementation enhancements to the routines for computing neighboring solution makespans benefit all local search metaheuristics for the JSP, albeit not necessarily to the same extent.

We execute 10 independent trials of each algorithm on each of the 50 benchmark instances and record the makespan of the best solution located during each trial. We treat STS as the baseline for comparison, and allocate a total of 50 million iterations to each trial. Via extensive empirical testing, we have computed estimates of the per-iteration run-time costs of both IJAR and RMC relative to STS. The resulting *cost-per-iteration* multipliers are shown in Table 1. The data indicate that individual iterations of IJAR are roughly three times more costly than a single iteration of STS, due to the complexity of the ascent/re-descent phase. In contrast, iterations of RMC are significantly cheaper than those of STS, primarily because RMC avoids evaluation of the entire *N5* neighborhood of a solution at each iteration. Via the combination of cost-per-iteration multipliers and a common experimental platform (see below), individual trials of RMC and IJAR are allocated a number of iterations that is approximately equivalent in terms of total run-time to 50 million iterations of STS. All code is written in C++ using the Standard Template Library, and compiled using the GNU gcc 3.3.1 compiler with level 3 optimization. All trials of our algorithms are executed on 3.0 GHz Pentium IV hardware with 2GB of memory, running Linux 2.4.20-8.

In using cost-per-iteration multipliers, our goal is to allocate equivalent CPU times to each trial of STS, IJAR, and RMC. We avoid direct time-limited trials primarily in order to enable replicability [36], as the number of iterations is a platform and implementation-independent performance measure. We selected the baseline computational cost of 50 million iterations of STS to enable us to directly contrast our results with those obtained by Nowicki and Smutnicki [37], who also report the performance of *i*-TSAB on the same benchmarks after 50 million iterations. Both STS and *i*-TSAB use the same algorithms to compute the makespans of neighboring solutions, which dominate the respective run-times. Given identical *implementations* of these underlying algorithms, the actual run-times of *i*-TSAB and STS (and by inference RMC and IJAR) would therefore be equivalent.

4.2 Parameter Settings

Values for all algorithmic parameters were selected based on limited experimentation. With the few noted exceptions, we made no exhaustive effort to determine "optimal" settings; rather, the indicated values yield reasonable performance on the benchmark instances under consideration. The performance of all our algorithms is somewhat sensitive to the ratio of the mobility trap check interval MTCI to the mobility threshold M_{thrs}. For both STS and IJAR, we let MTCI$=50$; for RMC, MTCI$=1000$. The value of M_{thrs} is fixed to 10, independent of algorithm. In contrast, performance is largely insensitive to the choice of the nominal walk length MWL_{nom} and walk length increment MWL_{inc}. Here, we fix $\text{MWL}_{nom} = 5$ and $\text{MWL}_{inc} = 0.25$.

An empirically effective tabu list size for TSAB on Taillard's benchmarks is 8 [8]. We also found that similar sizes resulted in the best overall performance of STS. Specifically, we let $\text{TT}_{min} = 6$ and $\text{TT}_{max} = 10$. Performance is far less sensitive to the choice of update interval; we fix TUI$=15$. For IJAR, we obtained the strongest performance on all benchmark instances using $k = 1$. Based on empirical quantification of attractor basin strength in the JSP [26], we fix OTCI=20, $\text{OEWL}_{nom} = 5$, and $\text{OEWL}_{inc} = 1$. The sole parameter in RMC is the ratio of uphill-to-accepted moves (UAG), used to compute the run temperature T. Based on a limited analysis of performance on smaller problem instances, we found that UAG=0.10 yields good overall performance.

4.3 Results: Qualitative Observations

We contrast the performance of both IJAR and RMC relative to STS. Let $C_{max}(i, X)$ denote the makespan of the best solution obtained by a given algorithm during trial i on instance X. Let $C^*_{max}(X)$ denote either the optimal makespan of instance X, if known, or the largest known lower bound on the optimal makespan. Up-to-date values of $C^*_{max}(X)$ can be obtained from Taillard's web site [38]; we use values reported as of January 1, 2004. Let $P(X)$ denote the quality of solutions obtained by a given algorithm on instance X; $P(X)$ represents either the best or mean makespan observed over 10 trials, depending on the context. Algorithm performance on instance X is quantified as the percent excess RE(X)$=\frac{P(X)-C^*_{max}(X)}{C^*_{max}(X)} * 100$ of $P(X)$ relative to $C^*_{max}(X)$. We then respectively define b-RE(X) and m-RE(X) as the value of RE(X) computed using the best and mean makespans observed over the 10 trials. Finally, for each problem group, we let b-MRE and m-MRE respectively denote

Table 2
Performance of core metaheuristics on Taillard's benchmark instances.

Problem	RMC		IJAR		STS		*i*-TSAB	PEZ	BAL*
Group	b-MRE	m-MRE	b-MRE	m-MRE	b-MRE	m-MRE			
ta01-10	0.06	0.15	0.12	0.22	0.09	0.18	0.11	0.45	0.16
ta11-20	2.75	2.93	2.91	3.13	2.94	3.13	2.81	3.47	2.81
ta21-30	5.79	5.94	6.05	6.27	5.87	6.04	5.68	6.52	6.10
ta31-40	0.88	1.01	0.96	1.09	0.87	1.00	0.78	1.92	0.80
ta41-50	4.79	5.09	5.09	5.37	4.85	5.07	4.7	6.04	5.20

the mean b-RE(X) and m-RE(X) observed across the member instances X.

The results for our three algorithms are shown in Table 2. We additionally cite the original performance statistics reported for (1) *i*-TSAB [7], (2) Pezzella and Merelli's tabu search algorithm [39], and (3) Balas and Vazacopoulos' guided local search algorithm [40]. Together, these algorithms dominate all others reported in the literature to date [7]. As indicated in Section 2, the *i*-TSAB statistics were obtained using individual trials. For `ta21-ta50`, results were reported after 50 million iterations; for `ta01-ta10` and `ta11-ta20`, results were reported after 5 and 20 million iterations, respectively. The statistics for Pezzella and Merelli's algorithm, denoted PEZ, are based on individual trials. Results for Balas and Vazacopoulos' algorithm, denoted BAL*, are based on the best solutions obtained over a wide range of trials and parameter settings. Statistics for these three algorithms are taken directly from [7]. We make no attempt to compare the run-times expended to obtain results for *i*-TSAB relative to PEZ and BAL*; results for the latter two algorithms are included simply for purposes of qualitative comparison. However, as indicated previously, the computational effort required for both our algorithms and *i*-TSAB are directly comparable.

We first consider the performance of STS relative to IJAR and RMC. Discounting a few exceptional instances, STS consistently outperforms IJAR in terms of both b-MRE and m-MRE. However, the difference is typically minimal, and is only 0.24 in the worst case. Similar results hold when comparing RMC and STS, although here the roles are reversed: RMC consistently *outperforms* STS. As with IJAR versus STS, the difference is typically minimal and is only 0.19 in the worst case. Overall, the results indicate that tabu search is *not* inherently superior to other metaheuristics for the JSP: straightforward implementations of iterated local search and Monte Carlo sampling provide quantitatively similar performance for the same computational effort.

Next, we compare the performance of STS with that of *i*-TSAB. STS is very similar to the core tabu search component found in *i*-TSAB. By contrasting the performance of STS and *i*-TSAB, we approximately control for the impact of long-term memory on the performance of *i*-TSAB. The results reported in

Table 3
The number of problem instances for which STS fails to outperform IJAR and RMC.

	IJAR		RMC	
Problem Group	@50M iters	@250M iters	@50M iters	@250M iters
ta01-10	9	10	10	10
ta11-20	7	9	9	10
ta21-30	4	10	10	10
ta31-40	4	10	9	10
ta41-50	2	9	8	10

Table 2 indicate that *i*-TSAB consistently outperforms the m-MRE obtained by simple *N5*-based tabu search, with a worst-case difference of 0.37. Recall that the computational effort for the two algorithms is equivalent (at 50 million iterations apiece) for the largest three problem groups, while *i*-TSAB uses far fewer iterations to achieve stronger results for the smallest two problem groups. The fact that *i*-TSAB outperforms STS is not unexpected, given the extensive empirical evidence regarding the benefits of long-term memory [1]. However, it is surprising that tabu search can yield such strong performance without long-term memory.

Finally, we contrast the performance of our algorithms with that of PEZ and BAL*, the two closest competitors to *i*-TSAB. All three algorithms outperform PEZ in terms of m-MRE. Both STS and RMC are competitive with BAL* in terms of their m-MRE, and frequently outperform BAL* in terms of b-MRE. Although the run-times are not directly comparable, the results do reinforce our previous observation that simple metaheuristics based on the *N5* operator can yield exceptionally strong performance.

4.4 Results: Quantitative Analysis

We now subject the hypothesis that STS offers no distinct advantage over either IJAR or RMC to statistical verification, via a series of non-parametric, two-sample, unpaired Wilcoxon tests. [5] Our null hypothesis posits that STS yields no better solutions than those obtained by either IJAR or RMC; the alternative hypothesis states that STS outperforms either IJAR or RMC. In the columns of Table 3 labeled “@50M iters”, we report the number of instances in each problem group for which we *fail* to reject the null hypothesis at $p \leq 0.05$ under the Wilcoxon test, given the computational equivalent of 50 million iterations of STS (i.e., using the multipliers specified in Table 1).

[5] All Wilcoxon tests are performed using the exact Wilcoxon test found in the *exactRankTests* library of the freely available *R* statistical software package. All tests are one-sided.

STS provides statistically significant performance improvements over RMC on only 4 of the 50 instances. In contrast, STS holds a statistically significant edge over IJAR on most of the larger problem instances.

The imposition of hard limits on the number of iterations allocated to algorithmic trials naturally raises the question of whether a competing algorithm might achieve equal performance with modest increases in computational effort. To address this issue, we re-consider those instance/algorithm pairs in Table 3 for which STS provided better performance at $p \leq 0.05$. For each such combination, we execute the corresponding algorithm on the given problem instance for an equivalent of 250 million iterations of STS, determined using the multipliers shown in Table 1. We then compute the p-values relative to our original null hypothesis, substituting the solutions obtained during the extended runs for their original values, when performed. The results are reported in Table 3 under the columns labeled "@250M iters". For all but two of the instances, a five-fold increase in computational effort achieves statistically insignificant differences in performance of IJAR and RMC relative to STS.

Recall that the 50 million factor derives from our original experimental allocation to runs of STS, which in turn was taken to mirror the computational effort required to achieve reported results for i-TSAB. Although somewhat arbitrary, we selected a multiplicative factor of five to generate the results shown in Table 3 for the following reason. In practice, implementation details and parameter settings necessarily impact the effectiveness of any given algorithm. Thus, we argue that if the performance of two algorithms is statistically indistinguishable given similar computational effort (e.g., within a factor of five), then there is little justification by which to base any claim of superiority of one algorithm over the other. Further, this mode of analysis allows us to directly quantify, or at least bound, the relative effectiveness of different metaheuristics for the JSP in terms of the computational effort required to achieve a given performance target.

The historical dominance of tabu search algorithms for the JSP, e.g., as documented in [6] and [41], would seem to provide confirmatory evidence for the hypothesis that tabu search holds an inherent advantage over other metaheuristics for the JSP. However, our results conclusively establish that this is in fact not the case: iterated local search and Monte Carlo sampling can provide equivalent performance. More specifically, both RMC and IJAR yield performance that is statistically equivalent to STS given a ratio of no more than 5:1 in terms of the computational effort required. Further, in the case of RMC, performance is almost always statistically indistinguishable given equivalent computational effort. At *worst*, all three algorithms are within a small constant factor of one another in terms of mean performance. Finally, due to our experimental objectives, we have avoided any discussion of the frequency with which RMC and IJAR *outperform* STS; rather, we observe that such differences are often statistically significant, particularly for RMC.

5 A Framework for Analyzing Intensification and Diversification

We now consider the impact of long-term memory mechanisms on the performance of core metaheuristics for the JSP. To facilitate this new analysis, we introduce a general algorithmic framework for integrating intensification and diversification - two prominent features of i-TSAB - into each of the algorithms analyzed in Section 4. The framework is denoted IDMF for Intensification-Diversification Metaheuristics Framework.

IDMF is expressed in terms of a variant of the CMF framework, which in turn assumes the existence of a core metaheuristic T. The variant, denoted CMF', is identical to CMF with the exception of the initial solution and the termination criterion. CMF' is designed to intensify search in the region near a pre-specified solution s_{init}. Search executes for X iterations of the metaheuristic T after each update of the best-so-far solution s^* (specific to a given intensification), including the initial assignment. Initially, $X = X_a$. After each subsequent update of s^*, $X = X_b$. The mechanism, borrowed directly from TSAB, allows search to continue as long as improvements to s^* are being located. Typically, $X_a > X_b$, allowing search to initially proceed for an extended period without a change in s^*. As in i-TSAB, both X_a and X_b are user-specified parameters; in practice, particular values are tied to one or more problem groups.

Mirroring i-TSAB, IDMF proceeds in two phases. In the first phase, CMF' is applied to random local optima (see Section 3.2), yielding the initial elements of a pool of elite solutions E. The first phases of IDMF and i-TSAB differ in three key respects. First, intensification and diversification are absent in the initial phase of IDMF. In contrast, i-TSAB uses a combination of TSAB and path relinking to generate the initial elements of E. Second, the initial elite solutions in IDMF are generated by applying CMF' to relatively poor-quality solutions. In i-TSAB, the initial elite solutions are generated by applying TSAB to either the solution resulting from Werner and Winkler's heuristic or solutions obtained via path relinking between extant elite solutions. Third, in IDMF the $e \in E$ are generated independently, while in i-TSAB they are linked via the process described in Section 2. As a result of these differences, the initial elite solutions under IDMF generally possess much higher makespans than those of i-TSAB.

The second phase of IDMF consists of a series of iterations in which intensification and diversification procedures are applied to the solutions in E with the respective probabilities p_i and p_d at each iteration, subject to $p_i + p_d = 1$. Under intensification, CMF' is initiated from a random elite solution $x \in E$.

If $C_{max}(s^*) < C_{max}(x)$, then the resulting s^* replaces x in E. Under diversification, two distinct elite solutions $x, y \in E$ are selected at random. Nowicki and Smutnicki's *NIS* path relinking procedure [7] is then used to generate a solution z roughly equi-distant in terms of disjunctive graph distance [22] from both x and y, and CMF' is initiated from z. If $C_{max}(s^*) < C_{max}(x)$, then the resulting s^* replaces x in E. IDMF terminates once the total CMF' and *NIS* iterations exceeds the limit MaxIters. For the parameter settings we consider, the relative ratio of CMF' iterations to *NIS* iterations is always greater than 100:1. Consequently, although *NIS* is based on the *N1* move operator, runs of CMF and IDMF are comparable given identical values of MaxIter.

Relative to i-TSAB, we have completely re-structured how and when intensification and diversification are performed, and have parameterized the relative frequency with which they are applied. In contrast to IDMF, intensification in i-TSAB always follows diversification, and intensification is never re-applied to the solutions in E. Further, because it is expressed via TSAB – which itself maintains a pool of elite solutions – intensification is applied in i-TSAB with much greater frequency than diversification. Finally, i-TSAB always applies the *NIS* path relinking procedure to the best elite solution $e^* \in E$ and the solution $e \in E$ that is maximally distant from e^*.

IDMF allows us to analyze the impact of long-term memory on the performance of each of the core metaheuristics considered in Section 4. The same functionality could also be achieved via a direct parameterization of i-TSAB. However, IDMF uniquely enables us to answer two additional research questions: (1) What is the relative impact of intensification and diversification on metaheuristic performance and (2) Is the full complexity of i-TSAB actually required to achieve state-of-the-art performance levels, or is there a small subset of key algorithmic features that yields equivalent performance? We answer the first question later in Section 7. The second question is answered in conjunction with our analysis of long-term memory on metaheuristic performance, described next in Section 6.

6 Experiment 2: The Impact of Long-Term Memory

We now consider the performance of the core metaheuristics analyzed in Section 3 when hybridized with IDMF; we denote the resulting algorithms by i-STS, i-IJAR, and i-RMC. Our objective to answer two open, related questions regarding the behavior metaheuristics for the JSP: (1) Can the long-term memory mechanisms commonly associated with tabu search equally improve the performance of other core metaheuristics? and (2) Can such "augmented" metaheuristics also achieve state-of-the-art performance levels? Additionally, we quantify the magnitude of the performance improvement conferred by long-term memory and identify those key components of i-TSAB that enable it to achieve state-of-the-art status.

6.1 IDMF-Specific Parameter Settings

We generally borrow the methodology and parameter settings described in Section 4. Due to both the strong similarities between IDMF and i-TSAB and the availability of tuned values of i-TSAB parameters, we directly borrow the values reported by Nowicki and Smutnicki [7] where there are direct analogs between i-TSAB and IDMF. In all trials, we fix $|E| = 8$. For i-STS, the search duration parameters X_a and X_b under 15×15 benchmark instances are set to X_a=20,000 and X_b=4,000; for larger instances, X_a=40,000 and X_b=7,000. The X_a (X_b) for i-IJAR and i-RMC are defined such that the computational effort is equivalent to X_a (X_b) iterations under i-STS, i.e., the values are computed using the multipliers shown in Table 1. Lacking any evidence to favor one mechanism over the other, we set the intensification and diversification probabilities as $p_i = p_d = 0.5$. Finally, the "maxV" parameter associated with the *NIS* path relinking procedure (see [7]) is set to 0.5. We made no attempt to tune the values of $|E|$, X_a, or X_b for specific algorithms, due to the computation cost of performing additional experiments and the fact that these parameters are known to interact [37].

6.2 Results: Core versus Augmented Metaheuristics

Based on the results presented in Table 4, we hypothesize that long-term memory mechanisms will yield significant and equal improvements in the performance of the core STS, IJAR, and RMC metaheuristics. To test this hypothesis, we compare the performance of each core metaheuristic with that of its augmented counterpart. Table 4 records both the difference in m-MRE between the augmented and core metaheuristics (specifically the m-MRE of the core metaheuristic minus the m-MRE of the augmented metaheuristic) and the number of instances in each problem group for which we fail to reject the null hypothesis of equivalent performance between the core and augmented metaheuristics at $p \leq 0.05$; the alternative hypothesis is that the augmented metaheuristic outperforms the core metaheuristic. The data indicates that in absolute terms, long-term memory improves performance of each core metaheuristic. Further, the difference is frequently statistically significant, particularly in the more difficult problem groups where there is more room for improvement relative to either optimal makespans or best-known upper bounds.

There is some evidence to suggest that the benefit of long-term memory is

Table 4
Performance differences between core metaheuristics versus those augmented with long-term memory.

	RMC vs. *i*-RMC		IJAR vs. *i*-IJAR		STS vs. *i*-STS	
Problem Group	Diff. in m-MRE	# Significant	Diff. in m-MRE	# Significant	Diff. in m-MRE	# Significant
ta01-10	0.03	5	0.13	3	0.06	2
ta11-20	0.19	8	0.14	5	0.34	8
ta21-30	0.03	9	0.42	6	0.25	8
ta31-40	0.06	5	0.08	2	0.17	6
ta41-50	0.19	9	0.42	6	0.30	8

not identical across the different metaheuristics. Recall from Table 2 that RMC and STS yield near-equivalent m-MRE on the 30×15 and 30×20 problem groups. Yet, long-term memory yields larger reductions in the m-MRE of STS than RMC, and minimally impacts the performance of RMC on the 20×20 instances. Further, the largest improvements are observed for IJAR, e.g., on the 20×20 and 30×20 problem groups. Despite these particular observations, however, it is difficult to draw any general conclusions without further analysis – in particular without further assessing the optimality of the parameter settings used in these experiments.

The results presented in Table 4 conclusively demonstrate that long-term memory significantly improves the performance of a range of core metaheuristics, and provide further indirect evidence in support of the hypothesis that the core tabu search metaheuristic is not inherently superior to other metaheuristics for the JSP. We have also for the first time directly, and under strict experimental control, quantified the benefit of long-term memory for metaheuristics for the JSP. Finally, we observe that the magnitude of the improvements shown in Table 4 are in absolute terms unexpectedly small. However, as shown below in Section 6.3, these differences are large enough to distinguish state-of-the-art metaheuristics for the JSP from second-tier competition.

6.3 Augmented Metaheuristics versus i-TSAB: Qualitative Results

We now consider the performance of the augmented metaheuristics in comparison to both each other and to *i*-TSAB. Table 5 reports the b-MRE and m-MRE of each augmented metaheuristic, in addition to (1) the mean relative error for solutions obtained by *i*-TSAB and (2) the mean relative error corresponding to either the best known upper bounds or the optimal makespans. In the latter two cases, the term "mean relative error" refers to the mean RE(X) over a problem group (see Section 4.3) for the solutions X returned by *i*-TSAB

Table 5
Performance of augmented metaheuristics on Taillard's benchmark problems.

Problem	Best	i-RMC		i-IJAR		i-STS		i-TSAB
Group	Known	b-MRE	m-MRE	b-MRE	m-MRE	b-MRE	m-MRE	
ta01-10	0.0	0.05	0.12	0.04	0.09	0.04	0.12	0.11
ta11-20	2.33	2.48	2.74	2.61	2.80	2.58	2.79	2.81
ta21-30	5.45	5.60	5.91	5.62	5.85	5.58	5.79	5.68
ta31-40	0.52	0.71	0.95	0.83	1.01	0.67	0.83	0.78
ta41-50	4.12	4.57	4.90	4.58	4.95	4.35	4.77	4.70

and the best known/optimal solutions, respectively.

We first compare the performance of i-RMC and i-IJAR with that of i-STS. In terms of absolute differences in m-MRE (as opposed to statistically significant differences, which are discussed below), i-RMC either equals or outperforms i-STS on the smaller two problem groups, while i-STS outperforms i-RMC on the larger, more difficult groups. In no case does the difference exceed 0.13. Similarly, i-STS outperforms i-IJAR on all but the smallest problem instances, with a worst-case difference of 0.18. Analogous differences exist in terms of b-MRE performance. Mirroring the results in Table 2, there is some preliminary evidence to suggest that i-STS may outperform the other augmented metaheuristics. For now we simply observe that i-STS *may* hold a marginal advantage; we rigorously test this hypothesis below in Section 6.4.

Next, we compare the performance of i-STS with i-TSAB. Recall from Section 4.3 that the computational efforts are only directly comparable on the three larger problem groups. In terms of m-MRE, i-STS only slightly underperforms i-TSAB; the largest absolute difference is a minimal 0.11. Further, the difference is only 0.07 on the notoriously difficult 30×20 problem group. In terms of b-MRE, i-STS consistently outperforms i-TSAB, with the largest difference of 0.35 observed on the most difficult problem group. Overall, the performance of i-TSAB is bracketed by the b-MRE and m-MRE obtained by i-STS, and generally falls much closer to the latter than the former. Unfortunately, the deterministic nature of i-TSAB prevents us from establishing a well-defined "mean" performance metric, such that we are unable to assess whether the observed differences are statistically significant. However, even given statistically significant differences in performance, i-STS only slightly underperforms i-TSAB.

Finally, we observe that by omitting various features of i-TSAB in IDMF, e.g., the more complex mechanism to construct the initial pool of elite solutions, we implicitly hypothesized that the full complexity of i-TSAB was not required to achieve its remarkable performance levels. Even given the assumption that i-TSAB only slightly outperforms i-STS, our results indicate that state-of-the-art performance depends primarily on the presence of two key algorithmic

Table 6
The number of problem instances for which i-STS fails to statistically outperform i-IJAR and i-RMC.

	i-IJAR		i-RMC	
Problem Group	@50M	@250M	@50M	@250M
ta01-10	9	10	10	10
ta11-20	8	10	8	9
ta21-30	6	9	7	10
ta31-40	4	10	7	9
ta41-50	8	10	6	10

features: the *N5* operator and a balance of straightforward intensification and diversification. At best, the omitted features of i-TSAB yield very minimal gains in performance. We conclude by observing that even the b-MRE achieved by the augmented metaheuristics – which are generally much lower than the MRE obtained by a single run of i-TSAB – are still as much as 0.25 higher than the MRE for the best-known or optimal solutions. Thus, status as "state-of-the-art" does *not* guarantee that an algorithm can consistently locate solutions equivalent to the best known; there remains significant room for improvement in the design of metaheuristics for the JSP, e.g., through the design of more advanced long-term memory mechanisms.

6.4 Augmented Metaheuristics versus i-TSAB: Quantitative Analysis

We now revisit the hypothesis that i-STS outperforms – albeit slightly – both i-RMC and i-IJAR. Mirroring the approach taken in Section 4.4, we subject this hypothesis to statistical verification via a series of non-parametric two-sample Wilcoxon tests. The null hypothesis posits equivalent performance between i-STS and either i-RMC or i-IJAR; the alternative hypothesis posits that i-STS yields better solutions on average that either i-RMC or i-IJAR. The columns of Table 6 labeled "@50M" report the number of instances in each problem group for which we fail to reject the null hypothesis at $p \leq 0.05$, given equivalent computational effort. The results indicate that i-STS does yield stronger performance than i-RMC and i-IJAR, although only on a small number of instances. More significant differences are observed for i-IJAR than i-RMC, which is consistent with the results reported in Table 5.

To bound the efficiency of i-STS relative to the other algorithms, we again execute trials of i-RMC and i-IJAR for an equivalent of 250 million iterations of i-STS. We then compute p-values corresponding to comparisons between the i-STS trials and the extended-duration i-RMC and i-IJAR trials, where performed; the results are reported in the columns of Table 6 labeled "@250M". In all but three cases, the additional computation is sufficient to achieve sta-

Table 7
Performance of augmented metaheuristics under extended-duration trials.

Instance	Best	i-IJAR		i-RMC		i-TSAB
Group	Known	b-MRE	m-MRE	b-MRE	m-MRE	
ta01-10	0.0	0.03	0.07	0.04	0.08	0.11
ta11-20	2.33	2.39	2.62	2.52	2.69	2.81
ta21-30	5.45	5.57	5.80	5.59	5.69	5.68
ta31-40	0.52	0.63	0.86	0.72	0.86	0.78
ta41-50	4.12	4.22	4.61	4.34	4.73	4.70

Table 8
New best-known upper bounds for Taillard's benchmark instances.

Instance	New Upper Bound	Old Upper Bound
ta19	1334	1335
ta20	1350	1351
ta37	1774	1778
ta41	2014	2018
ta42	1950	1956
ta46	2019	2021
ta47	1900	1903

tistically indistinguishable performance relative to i-STS. Overall, the results indicate that the performance of i-RMC and i-IJAR is generally statistically equivalent to that of i-STS given a ratio of no more than 5:1 in terms of the computational effort required.

In Table 7, we report the b-MRE and m-MRE values resulting from the extended-duration i-RMC and i-IJAR trials. In contrasting these results with the corresponding entries for i-STS in Table 5, we observe minimal absolute differences, which is consistent with the results from our statistical analysis. Of particular note is the remarkable performance, in terms of b-MRE, achieved by i-RMC; the worst-case deviation from the best known results is only 0.12. Finally, although our research objective is the analysis of metaheuristics for the JSP, our experiments have resulted in solutions that improve over the current best-known upper bound for seven of Taillard's benchmark instances. The improved bounds are summarized in Table 8.

Given the potentially large impact of implementation details and parameter values on performance, we conclude that the performance of the augmented metaheuristics is roughly equivalent, i.e., no clear winner stands out. Most importantly, our results demonstrate that long-term memory equally benefits all of the core metaheuristics we consider, and that all of the augmented metaheuristics achieve performance that is competitive with i-STS, and by inference i-TSAB. Due to the experimental control, these results also provide additional evidence that core tabu search is not integral to the performance of i-TSAB.

Table 9
Performance of i-STS under various combinations of p_i and p_d.

		p_i / p_d				
Instance Group	STS	0.00/1.00	0.25/0.75	0.50/0.50	0.75/0.25	1.00/0.00
ta01-10	0.18	0.14 (2)	0.11 (3)	0.12 (3)	0.09 (3)	0.07 (4)
ta11-20	3.13	2.81 (7)	2.79 (8)	2.79 (8)	2.70 (8)	2.79 (8)
ta21-30	6.04	5.83 (7)	5.87 (6)	5.79 (8)	5.80 (8)	5.93 (5)
ta31-40	1.00	0.90 (3)	0.93 (4)	0.83 (6)	0.86 (5)	1.10 (0)
ta41-50	5.07	4.83 (6)	4.79 (7)	4.77 (8)	4.79 (8)	5.49 (0)

7 Experiment 3: Intensification versus Diversification

Our motivation in Section 6 for selecting $p_i = p_d = 0.5$ was simply that there is no *a priori* evidence to suggest that one mechanism should be preferred over the other. Empirically, equi-probable settings significantly improve the performance of all the algorithms we consider, and enable i-STS to achieve performance levels competitive with those of i-TSAB. However, this decision leaves two important questions unresolved: (1) Can different weightings yield improved performance? and (2) What are the relative benefits of intensification and diversification? Given the overall focus of our research, we address these questions in the context of i-STS, although the choice is in principle arbitrary.

Using the experimental methodology and parameter settings described in Section 6, we compute m-MRE statistics for i-STS under the following p_i (see Section 5): 0.0, 0.25, 0.75, and 1.0. Table 9 records the results; data for $p_i = 0.50$ and STS are included for reference. We additionally report (in parenthesis) the number of instances in each problem group for which i-STS under the given parameter setting outperforms STS at $p \leq 0.05$ under a Wilcoxon test; the null hypothesis posits identical performance, while the alternative hypothesis posits stronger performance than STS.

Under pure diversification ($p_d = 1.0$), i-STS outperforms STS, with absolute differences in m-MRE ranging from 0.04 to 0.32. The relatively small margins are reflected in the p-values, which indicate that the difference is statistically significant for just over half of the benchmark instances. Similar results hold for i-STS under pure intensification ($p_i = 1.0$) on the three smaller problem groups. However, performance is actually worse than STS on the largest two problem groups: intensification alone can *degrade* the performance of the core metaheuristic. Overall, the strongest performance is obtained when both p_d and p_i are non-zero. The best results on the largest three problem groups are obtained when $p_i = p_d = 0.5$, also yielding the largest number of instances for which the performance differences between i-STS and STS are statistically significant; similar results are obtained with $p_i = 0.75$ and $p_d = 0.25$. Overall, the results demonstrate that the effect of intensification and diversification is multiplicative, with the combination yielding stronger performance than either mechanism in isolation. Further, there is some evidence for a "sweet spot" near $p_i = p_d = 0.5$, i.e., the mechanisms must be applied with near-identical frequency to achieve maximal performance.

8 Implications and Conclusions

The primary goal of our research was to explicitly identify those components of Nowicki and Smutnicki's *i*-TSAB tabu search algorithm that enable it to achieve state-of-the-art performance levels, and to quantify the overall contribution of each such component. We show that the core metaheuristic, and in particular tabu search, is *not* integral to the performance of *i*-TSAB. Rather, *i*-TSAB achieves state-of-the-art performance levels through the use of both the *N5* move operator and a balanced combination of intensification and diversification. Viewed from another standpoint, we have shown that state-of-the-art algorithms for the JSP are, at a very fundamental level, quite simple – leaving significant room for further advances. For example, more advanced methods for maintaining pools of elite solutions [42] offer the potential to significantly improve performance.

Our results emphasize the importance of long-term memory in metaheuristic performance, by demonstrating that long-term memory significantly and equally improves the performance of a range of core metaheuristics. Traditionally, long-term memory mechanisms have been associated primarily with the tabu search metaheuristic. However, given that the problem of escaping local optima is "solved", our results strongly suggest that a more integrative view of the role of long-term memory in metaheuristic design is required. In particular, we argue that metaheuristics should *only* be differentiated in terms of how they use long-term memory to guide search.

We conclude by observing that through a relatively simple series of carefully designed experiments, we are able to yield significant insights into the behavior of metaheuristics for the JSP. Much of the research on metaheuristics focuses on the development of new techniques, rather than developing a thorough understanding of the behavior of existing algorithms. As a result, misconceptions and unproven assertions regarding various metaheuristics are found throughout the literature. Yet, straightforward analysis can – if carefully performed – often resolve these issues, and place the field of metaheuristics on a more scientific foundation.

Acknowledgments

The authors would like to thank the anonymous referees for their invaluable suggestions relating to the clarification of our results and the compaction of our overall presentation.

References

[1] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, Massachusetts, 1997.

[2] E. Nowicki and C. Smutnicki. *Metaheuristic Optimization Via Memory and Evolution*, chapter Some New Ideas in TS for Job Shop Scheduling, pages 165–190. Kluwer Academic Publishers, 2005.

[3] J. Grabowski and M. Wodecki. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Technical Report 64/2002, Institute of Engineering Cybernetics, Wroclaw University of Technology, Wroclaw, Poland, 2002.

[4] J.P. Watson, J.C. Beck, A.E. Howe, and L.D. Whitley. Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence*, 143(2):189–217, 2003.

[5] Jean-Paul Watson, L. Darrell Whitley, and Adele E. Howe. Linking search space structure, run-time dynamics, and problem difficulty: A step toward demystifying tabu search. *Journal of Artificial Intelligence Research*, To Appear.

[6] J. Blażewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93(1):1–33, 1996.

[7] E. Nowicki and C. Smutnicki. New algorithm for the job shop problem. Technical report, Institute of Engineering Cybernetics, Wroclaw University of Technology, Wroclaw, Poland, 2003.

[8] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[9] É.D. Taillard. Parallel taboo search technique for the jobshop scheduling problem. Technical Report ORWP 89/11, DMA, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1989.

[10] M. Dell'Amico and M. Trubian. Applying tabu-search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252, 1993.

[11] J.W. Barnes and J.B. Chambers. Solving the job shop scheduling problem with tabu search. *IIE Transactions*, 27:257–263, 1995.

[12] H. Fisher and G.L. Thompson. *Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules*, chapter 15, pages 225–251. Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

[13] F. Werner and A. Winkler. Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics*, 58:191–211, 1995.

[14] E. Nowicki and C. Smutnicki. Some new ideas in TS for job-shop scheduling. Technical Report 50/2001, Institute of Engineering Cybernetics, Wroclaw University of Technology, Wroclaw, Poland, 2001.

[15] F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Advances and applications. In F. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[16] P.J.M. van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

[17] H. Matsuo, C.J. Suh, and R.S. Sullivan. 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, 1988.

[18] É.D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[19] H.R. Lourenço. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347–364, 1995.

[20] D.C. Mattfeld. *Evolutionary Search and the Job Shop*. Physica-Verlag, Heidelberg, 1996.

[21] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):128–140, 1994.

[22] D.C. Mattfeld, C. Bierwirth, and H. Kopfer. A search space analysis of the job shop scheduling problem. *Annals of Operations Research*, 86:441–453, 1999.

[23] H.R. Lourenco, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[24] H.R. Lourenço. *A Computational Study of the Job-Shop and the Flow-Shop Scheduling Problems*. PhD thesis, School of Operations Research and Industrial Engineering, Cornell University, 1993.

[25] A.S. Jain. *A Multi-Level Hybrid Framework for the Deterministic Job-Shop Scheduling Problem*. PhD thesis, University of Dundee, 1998.

[26] J.P. Watson. *Empirical Modeling and Analysis of Local Search Algorithms for the Job-Shop Scheduling Problem*. PhD thesis, Department of Computer Science, Colorado State University, 2003.

[27] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[28] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[29] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8(3):302–317, 1996.

[30] E.H.L. Aarts, P.J.M. van Laarhoven, J.K. Lenstra, and N.L.J. Ulder. A computational study of local search algorithms for job shop scheduling. *ORSA Journal on Computing*, 6(2):118–125, 1994.

[31] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[32] P. Salamon, P. Sibani, and R. Frost. *Facts, Conjectures, and Improvements for Simulated Annealing*. Society for Industrial and Applied Mathematics (SIAM), 2002.

[33] T. Yamada, B.E. Rosen, and R. Nakano. A simulated annealing approach to job-shop scheduling using critical block transition operators. In *IEEE International Conference on Neural Networks (ICNN '94)*, pages 4687–4692, 1994. Orlando, Florida, USA.

[34] E.H.L. Aarts and P.J.M. van Laarhoven. A new polynomial-time cooling schedule. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 206–208, 1985.

[35] É.D. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[36] D.S. Johnson. A theoretician's guide to the experimental analysis of algorithms. In M.H. Goldwasser, D.S. Johnson, and C.C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250. American Mathematical Society, Providence, 2002.

[37] E. Nowicki and C. Smutnicki. Some new tools to solve the job shop problem. Technical Report 60/2002, Institute of Engineering Cybernetics, Wroclaw University of Technology, Wroclaw, Poland, 2002.

[38] É.D. Taillard. http://ina.eivd.ch/collaborateurs/etd/default.htm. January, 2004.

[39] F. Pezzella and E. Merelli. A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120(2):297–310, 2000.

[40] E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job-shop scheduling. *Management Science*, 44(2):262–275, 1998.

[41] A.S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present, and future. *European Journal of Operational Research*, 113(2):390–434, 1999.

[42] P. Greistorfer. Experimental pool design: Input, output, and combination strategies for scatter search. In M. Resende and J. de Sousa, editors, *Metaheuristics: Computer Decision-Making*, pages 279–300. Kluwer Academic Publishers, 2003.