TABU SEARCH FOR A CLASS OF SCHEDULING PROBLEMS (Preprint)

Edward L. Mooney
Industrial and Management Engineering Department
Montana State University
Bozeman, Montana 59717-0384

Ronald L. Rardin
School of Industrial Engineering
Purdue University
West Lafayette, Indiana 47907

June 1992

Abstract

Scheduling problems are often modeled as resource *constrained* problems in which critical resource assignments to tasks are known and the best assignment of resource time must be made subject to these constraints. Generalization to *resource scheduling*, where resource assignments are chosen concurrently with times results in a problem which is much more difficult. A simplified model of the general resource scheduling model is possible, however, in which tasks must be assigned a single *primary* resource, subject to constraints resulting from preassignment of *secondary*, or auxiliary, resources.

This paper describes extensions and enhancements of Tabu Search for the special case of the resource scheduling problem described above. The class of problems is further restricted to those where it is reasonable to enumerate both feasible time and primary resource assignments. Potential applications include shift oriented production and manpower scheduling problems as well as course scheduling where classrooms (instructors) are primary and instructors (rooms) and students are secondary resources.

The underlying model is a type of quadratic multiple choice problem which we call multiple choice quadratic vertex packing (MCQVP). Results for strategic oscillation and biased candidate sampling strategies are shown for reasonably sized real and randomly generated, synthetic, problem instances. These strategies are compared with other variations using consistent measures of solution time and quality developed for this study.

1 Introduction

Scheduling has been defined as the "allocation of resources over time to perform a collection of tasks" [BAKE74]. Scheduling is important in the design and management of a variety of systems with differing tasks and resources. Some examples include production systems, construction projects, education and health delivery systems, transportation systems, and so on.

While each of these systems utilizes different resource classes, assigning times to activities requiring those resources remains the fundamental scheduling problem. It is the time element that distinguishes scheduling from other resource allocation problems. However, there are, in general, two related decisions required as part of the scheduling process:

1. Assignment of time for accomplishing each task
2. Selection (assignment) of the necessary resources

Time assignment requires dividing tasks among resources and choosing starting times. It must consider relations among tasks such as precedence requirements as well as the availability of limited resources necessary for performing the tasks. Resource assignment in general must consider the availability of specific resource units as well as the suitability of a particular unit for a particular task. The interplay between the task time and resource assignment decisions is therefore at the heart of scheduling.

We refer to scheduling problems which require simultaneous resource assignment as *resource scheduling problems*. This distinguishes such problems from the component (resource constrained) scheduling and resource assignment problems. Furthermore, we define *primary* and *secondary* resources to be those which are scheduled and known a priori respectively. Secondary resources therefore constrain the schedule but may not be chosen. Finally, problems are classified by the number of units of each resource type available for assignment (primary resources) or allocation (secondary resources) as well as the number of units which may be required by a task (one or many). Specifically, we differentiate single unit resources (or resource types) from those where there is a finite number of units, as uncapacitated and capacitated resources respectively.

This paper investigates a family of local improvement heuristics addressed to a special case of the resource scheduling problem which occurs often in practice. The case of interest assumes N_s tasks are to be scheduled on a set $\mathcal{R}$ of single unit primary resources where each task, s, requires at most a single unit to be assigned from a subset, $\mathcal{R}_s \subset \mathcal{R}$. Similarly, an arbitrary set of secondary unit resources may constrain the schedule where at most one unit of each resource may be required by a task.

Mathematically, the problem may be formulated as quadratic integer program which we call multiple choice quadratic vertex packing (MCQVP). Assume there are "choices" $x_j \in \mathcal{X}$, $j \in \{1, 2..N_x\}$, where choice x_j corresponds to a unique task, primary resource, time triple, $[s(j), r(j), t(j)]$, and the choice set for task s is $\mathcal{X}_s$. Now, given assignment "value", c_j, for picking x_j for task $s(j)$ and d_{jk}, the "cost" of not satisfying a second order preference requiring both (or neither) x_j and x_k at once we want to

$$\text{maximize} \qquad Z = \sum_j c_j x_j - \sum_j \sum_k d_{jk} x_j x_k \tag{1}$$

subject to:

$$\text{(MCQVP)} \qquad \sum_{j:x_j \in \mathcal{X}_s} x_j = 1 \quad \text{for all } s \tag{2}$$

$$x_j + x_k \leq 1 \quad \text{for all conflicting choices } x_j,\ x_k,\ j \neq k \tag{3}$$

$$x_j \in \{0, 1\} \tag{4}$$

where task $s(j)$ is assigned resource $r(j)$ for a time interval $t(j)$ if $x_j = 1$.

Equation (1) represents the desire to maximize first and second order assignment preferences for both resource and time assignments. For example, d_{jk} may be a penalty for not meeting desired relations between task times such as one task occurring before or after another. A "soft" conflict occurs when such preferences relations are not met. In the problems considered here, d_{jk} is the number of these conflicts between choices x_j and x_k.

Soft conflicts contrast with "hard" conflicts which are precluded from feasible schedules by constraints of the form of equation (3). Hard conflicts arise when two tasks are scheduled simultaneously with the same resource (primary or secondary), or possibly when required precedence relations are not honored. Obviously, the vertex packing constraints of (3) could be represented as high priced "soft" conflicts (preferences), and the search algorithm treats them similarly. We choose to separate them in the model, however, in order to maintain the difference which exists in real problems between physically required relations and desired ones.

The MCQVP problem is a generalization of graph coloring. This can be shown by transforming colorability into an equivalent decision version of MCQVP. MCQVP must be NP-hard because colorability is NP-complete [AHO74]. Hence, realistic sized instances will not likely yield to exact solution methods, suggesting the need for heuristics which will reliably provide good solutions.

2 Rationale

Local search algorithms have several inherent advantages, among them implementation ease and an intuitive appeal. Also, the ability to both generate solutions and improve existing

ones is attractive. Finally, local search algorithms have been shown to be both effective and efficient on a wide range of problems.

Figure 1 illustrates the general approach. As shown, given a problem instance and a method for obtaining an initial solution, $\mathbf{x}^0$, local search (LS) produces a series of complete solutions. The algorithm terminates according to predetermined stopping rules which may be stated in terms of number of iterations, solution improvement, or computation time. The results reported here were based on a stopping rule stated in terms of the total amount of cpu time allowed for the search. This rule allows a consistent comparison of algorithm performance but may be modified in practice.

Step 1: Get an initial solution and initialize the search

Step 2: Select a set of candidate moves

Step 3: Select the next move

- 3.1 Select a candidate for evaluation
- 3.2 Apply evaluation function
- 3.3 Check admissibility (tabu, feasibility, ...)
- 3.4 Apply selection rule (best, first improvement, ...)
- 3.5 Repeat 3.1 through 3.4 until all moves are compared

Step 4: Update the solution status if a move is found, otherwise restart at a new point

Step 5: Repeat steps 2 through 4 until done

Figure 1: Local Search (LS) Algorithm

A *move*, m, is a function that maps a trial solution at iteration $i-1$, $\mathbf{x}^{i-1}$, to a new solution, $\mathbf{x}^i$, at iteration i as follows

$$m : \mathbf{x}^{i-1} \mapsto \mathbf{x}^i$$

where $\mathbf{x}^i$ is a solution at the end of iteration i in the *neighborhood* of $\mathbf{x}^{i-1}$.

Local search algorithms for a given problem and neighborhood are differentiated by move selection strategies (step 3 in Figure 1). In general, a move is selected from a set of candidates within the set of *admissible* moves. Admissible moves are obtained by applying various restriction tests (e.g. tabu, feasibility, ...) to the neighborhood set. The candidate move set

may be comprised of the entire set of admissible moves or a subset obtained by sampling or other means. Moves are chosen by comparing candidates using an evaluation function and applying a selection rule.

All admissible moves must result in solutions which preserve feasibility of the multiple choice constraints of equation (2). Feasibility of the conflict constraints in (3) is not required for interim solutions in general. Notice that an initial solution which satisfies (2) can be easily found by arbitrarily choosing exactly one assignment per choice set. Any exchange move is admissible according to this criteria, given such an initial solution.

Pairwise interchange moves were chosen for solution of MCQVP instances. These moves simultaneously complement two of the choices belonging to a multiple choice set, s, $x_{k'}$ and x_k ($s(k) = s(k') = s$). At the beginning of the exchange, x_k is the current solution variable for choice set s, ($x_k = 1$); $x_{k'}$ is the variable selected to replace x_k as the choice for set s. Because one and only one of the choices for each set can be in the solution at a time, $x_{k'} = 0$ at the beginning of the swap. Of course, $x_k = 0$ and $x_{k'} = 1$ when the exchange is complete.

Notice that there is another way to view this neighborhood. Each choice represented by some variable x_j is a choice for one and only one set, $s = s(j)$. Therefore, the set $\mathcal{X}_s = \{x_j : s(j) = s\}$ can be viewed as discrete values that some choice-set variable can take on. When viewed in this way, an exchange of choice k for k' corresponds to a 1-change in the value of the choice set variable, s.

2.1 Diversification

A local search algorithm which takes the "best improving" move at each step typically moves to a local optimum in the neighborhood of the initial solution and stops when no improving moves can be found. Such algorithms may be applied when good initial solutions are known or can be found by, for example, constructive methods to get better solutions. The chances of finding a global optimum are generally very low for hard problems, however, since such problems are usually characterized by a large number of similarly valued local optima with relatively few "good" solutions available.

An obvious strategy for improving the chances of finding a global optimum is to *diversify* the search away from local optima. Many stochastic strategies have been used for this purpose and, in general, involve sampling solutions or moves. For example, simulated annealing (SA) samples moves and accepts non-improving moves with a time-varying probability [KIRK83].

Multistart (MS) samples a new initial solution whenever a local optimum is reached with no improving moves available. Figure 2 shows a typical solution trajectory for the multistart algorithm applied to an instance of MCQVP. Restarts with a new solution appear as discontinuities. Each restart requires complete reinitialization, a costly activity relative to each local search iteration.

The graph in Figure 2 shows a shaded region delineated by two solid lines. A point is

plotted on each line corresponding to the solution at each iteration. The upper line shows the objective function (Z) value and the lower line shows this value *minus* the hard conflicts (total infeasibility). The shaded region therefore represents the degree of infeasibility at each iteration. Feasible solutions are easily identified when the two curves come together and the shaded region disappears. The first feasible and local optimums are labeled with their objective function values.

Figure 2 also illustrates two interesting aspects all of the algorithms presented here appear to share. First, feasibility appears to be achieved in most cases quite early in the search. This is likely a result of the structure of the test problems drawn from the course scheduling domain and the suitability of local search in general to their solution. A second feature is their apparent ability to simultaneously improve the objective function and feasibility. This can be explained, at least in part, by the obvious correlation between hard (constraints) and soft conflicts (quadratic objective function terms). Also, it has been observed that as the search develops moves are often available which will improve the objective function but not feasibility or the reverse.

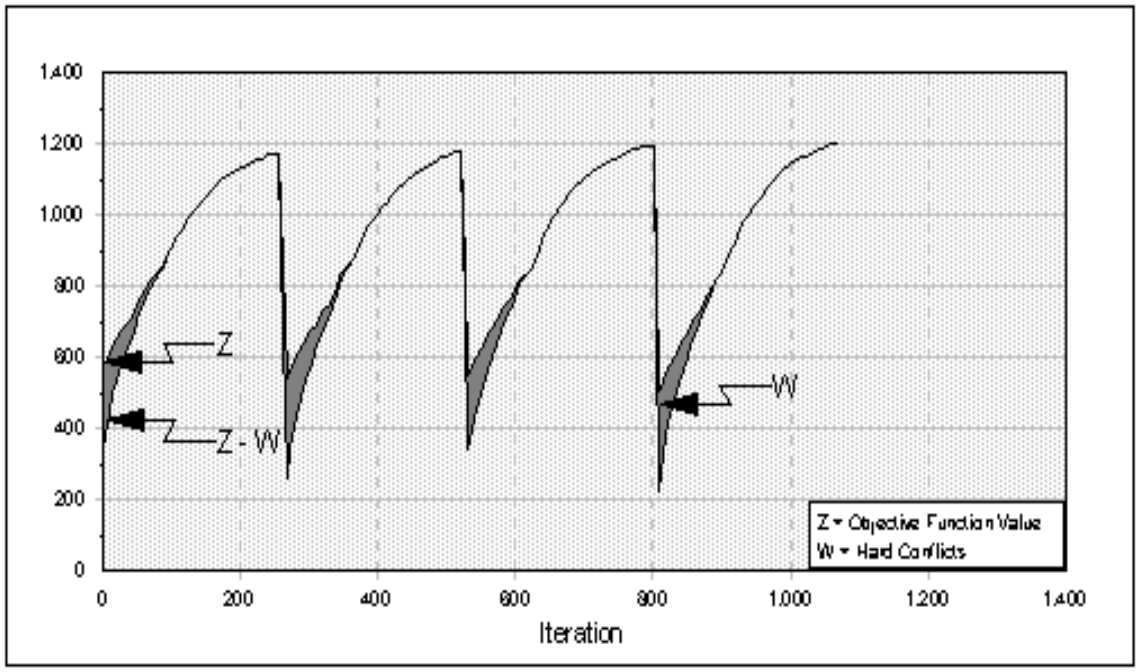

Figure 2: Multistart Trajectory

Tabu Search achieves diversification without restarting. The main mechanism for accomplishing this is the tabu list which maintains a "memory" of moves recently taken in order to prevent reversals which would cycle back to the same local optimum. This is done while maintaining a greedy move selection bias toward the best available solutions. Finally, Tabu methods attempt to balance diversification with *intensification* in promising regions to avoid missing good local optima [GLOV89D, GLOV90A].

Figure 3 illustrates a search trajectory for a simple Tabu Search algorithm (TS) using a short term memory. An aspiration criteria which allows override of tabu status for moves which lead to a better solution than ever found was employed to enhance intensification. Comparison of Figure 3 with 2 demonstrates the value of the Tabu strategy in obtaining better quality solutions without restarting the search.

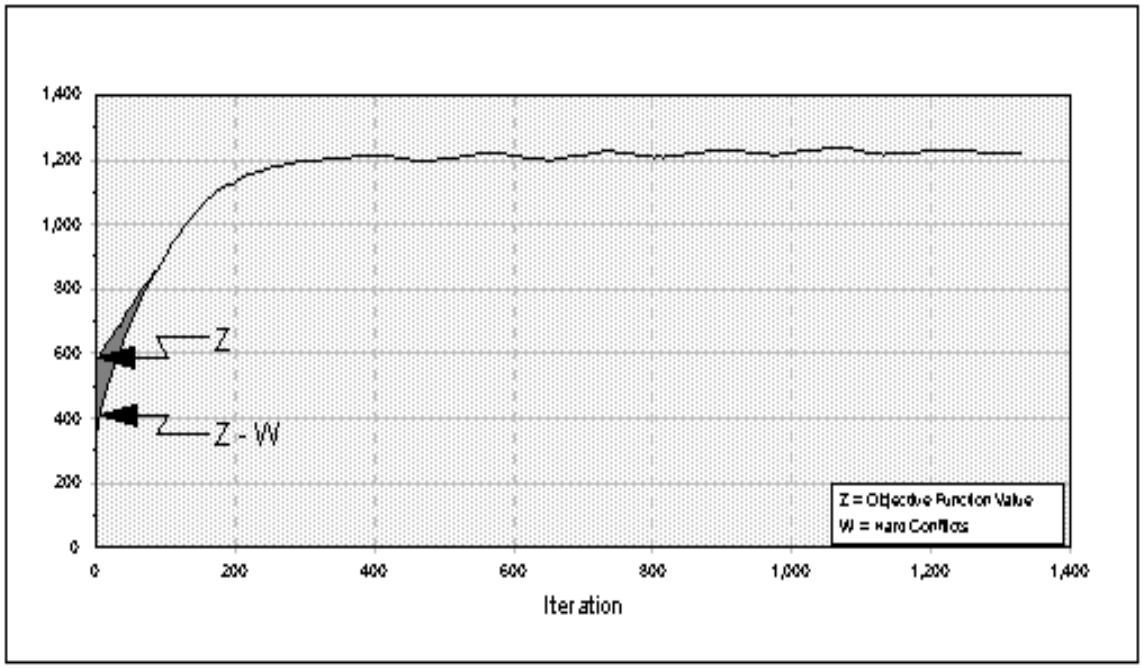

Figure 3: Tabu Search Trajectory

2.2 This Research

Close examination of the trajectory in 3 reveals a cycling tendency evidenced by periodic oscillation related to the tabu list length. This tendency appears to be a consequence of the use of a fixed length tabu list and suggests an algorithm that may become trapped in a region of the search space. Although this region is larger than a single local optimum, it may not contain the global optimum. Also, the problem becomes more pronounced as problem size increases and the number of neighbors for any solution increases exponentially.

This tendency for enhanced algorithms using only short term memory to cycle at frequencies related to the memory length suggests that additional diversification is necessary for larger problem sizes. The purpose of this research was to test this hypothesis by developing additional diversification strategies to complement the basic Tabu methods. A secondary research question concerned the validity of the tenets underlying Tabu Search: aggressive exploration and balance between diversification and intensification.

Other researchers have also observed the need for additional diversification and have proposed a lengthy list of possible approaches. Examples include the use of long term memory, multiple and dynamic tabu lists, strategic oscillation, and probabilistic strategies. Others include candidate lists and frequency based Tabu Search [GLOV89A, GLOV89B, GLOV90A]. Target analysis is a relatively recent entry in the field of techniques which have been proposed [GLOV89C, LAGU90A]. These strategies have been demonstrated to improve Tabu Search performance in various problem settings but, with a few exceptions, have not been rigorously studied yet.

A number of algorithms incorporating various diversification strategies were evaluated in order to compare these strategies and verify the efficacy of Tabu search in this problem domain. All strategies were implemented as variations in the candidate move set selection, move evaluation, and admissibility test steps shown in Figure 1. A requirement of all algorithms was that parameters must be easily tuned for a particular class of problems.

Several diversification strategies were considered in addition to MS. They included multiple tabu lists, long term memory, strategic oscillation, and various stochastic approaches. Preliminary experimentation identified two promising classes for further study:

- Strategic Oscillation
- Stochastic Tabu

Tabu Search with strategic oscillation (TSO) as explored here involves alternately emphasizing feasibility and objective function during move selection. This is consistent with the principles outlined above in that the search is aggressively moved into "good" (minimum infeasibility or maximum object value) regions away from local optima. Also, the diversification-intensification balance is controlled by a single, easily set parameter - the period of the oscillator.

Two stochastic Tabu Search variations were tried. Both strategies required sampling choice sets for move evaluation. In the random Tabu Search (RTS) variation, a random sample of choice sets was chosen at each step for evaluation.

The dynamic biased sampling method (DBS) selectively sampled choice sets for move evaluation based on a dynamic priority. The priority, by reflecting both the time since a set was checked (recency) and the number of times exchanges were made (frequency), eliminates the need for a tabu list to implement the short-term memory functions. This approach is consistent with the basic Tabu principle of aggressive move selection but, by incorporating a kind of "stochastic aspiration", also allows for the sort of serendipity typical of human memories.

Figure 4 shows the search trajectory for the dynamic biased sampling algorithm with strategic oscillation (DBSO) run on the same problem as shown in Figures 2 and 3 for the MS and TS algorithms. As before, at any iteration the shaded area corresponds to the total hard conflicts in the solution. The dotted curve shows how the weight given to feasibility

oscillates over time, driving the search toward feasible solutions when it is high and ignoring feasibility entirely when it reaches zero.

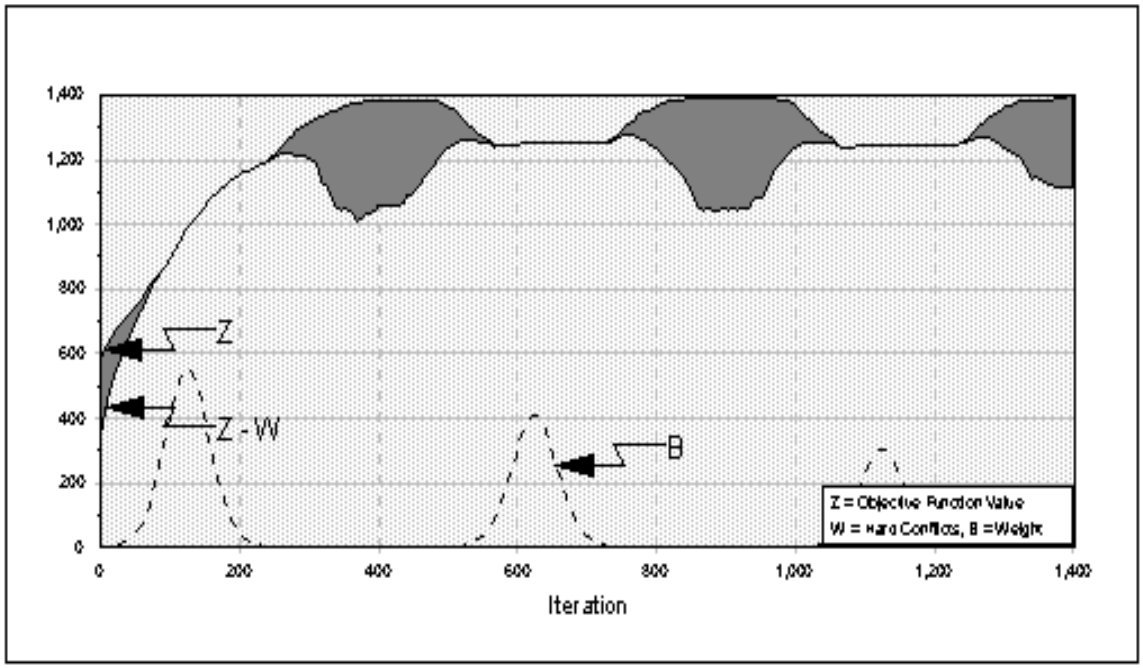

Figure 4: Dynamic Biased Search With Oscillation Trajectory

The idea is to periodically drive the search into "good" regions where good regions contain either super-optimal or feasible solutions. It appears that the best period for the oscillator is closely linked to the number of choice sets (between 2 and 2.5 times that number). Although relatively small penalty values are usually adequate to find a feasible local optimum, a large amplitude oscillation is often necessary to move the search to a new local optimum. This seems to allow the search to move to (i.e. intensify around) a local optimum before diversifying to another promising region (see Figure 2).

Figure 4 clearly demonstrates the validity of the approach for this sample problem. We present results which further support these claims in the remainder of this paper. Test results are given for both synthetic and real problems following a detailed description of the methods used.

3 Research Methods

An experimental approach was used to explore MCQVP search algorithms within the domain of university course scheduling problems. Specifically, we consider the problem of choosing rooms and times for each section of each course to be taught. Sections are modeled as multiple choice sets in MCQVP, and choice values are a function of room and/or time fit or

preference. Pairs of sections which cannot be taught at the same time "interact" through common instructor and/or student assignments, and they are assumed to be known.

The algorithms were restricted to three classes of local search: a "strawman" local improvement algorithm, Tabu and stochastic Tabu. The basic approach was to identify promising algorithm variations and screen them on a representative set of problems. The best variants were then rigorously tested using random and real problems drawn from the university course scheduling domain.

The research was accomplished in three phases:

1. Problem factor identification
2. Algorithm screening
3. Final testing and analysis

In the first phase the general problem structure was analyzed using real scheduling problems, and a random problem generator was developed to embed those characteristics in synthetic test problems. A large set of problems was then run against representatives of the three algorithm classes. The purpose of these runs was identification of the problem characteristics with real impact on algorithm ranking, based on the larger problem set.

Problem factors were further reduced in the next phase as a large number of algorithm variations were tried and eliminated from further consideration. It was also determined that two levels of each problem factor were sufficient to correctly rank the algorithms. Additional effort was also devoted to calibrating algorithm parameters for final testing and analysis.

Final experiments were conducted using both real and new synthetic problems. Multiple "runs" of each algorithm were made for each problem from different starting points. Problem, algorithm, and final experimental testing and analysis details are given below.

3.1 Test Problems

Real university scheduling problems, derivatives of the real problems (pseudo real problems), and randomly generated synthetic problems were used for final algorithm testing and evaluation. A total of 6 real, 6 pseudo real, and 96 synthetic problems were run against seven algorithm variations.

3.1.1 Real and Pseudo Real Problems

Real problems were obtained from departments and campuses of Purdue University. General characteristics for both the real and pseudo real problems are summarized in Table 1. For these problems choices correspond to slots consisting of a classroom and time. Solution of the

Table 1: Real Test Problem Characteristics

Problem	Sections	Choices	Section Interaction Density	Pattern Overlap Density	Choice Conflict Density
R1	67	731	.0660	.0444	.0114
R2	196	5,707	.0322	.1109	.0054
R3	162	5,348	.0225	.1109	.0056
R4	362	25,166	.0185	.0803	.0048
R5	137	5,784	.0331	.0736	.0047
R6	492	17,313	.0032	.0732	.0024

problem results in a slot, or choice, being assigned to each section. The number of sections and choices are indicative of the problem size.

Time pattern overlap, section interaction, and choice conflict densities shown in Table 1 are given as a fraction of time pattern, section, and choice pairs respectively. Time patterns identify days and times for each meeting of a section. Two patterns *overlap* if they have any common time. Sections *interact* if they cannot be scheduled at the same time (e.g. due to common instructors or students). Therefore, if a pair of choices correspond to the same room, or their sections interact, their choices will *conflict* if their time patterns overlap.

All problems except R5 and R6 were departments. R5 is a set of "core", or highly interacting courses at a small campus. This explains the slightly lower conflict densities compared with similar sized departments. Problem R6 is a large lecture room scheduling problem which crosses several curricula.

In general, the problems represent a range of room sizes and preference structures. For example, R1, R4, and R6 have time preferences expressed for most sections, while the remaining problems have none. Similarly, student conflicts for R1 through R4 were determined by experts, while R5 and R6 were derived from joint course enrollment history. It is also noteworthy that the choice conflict density decreases with problem size. This is the natural consequence of the tendency for conflicts to cluster with section subsets as the problem size increases.

The objective function coefficients for the problems are not shown in Table 1. In general, the first order preferences (corresponding to the linear term in equation (1) reflected room and time pattern "fit" with the class size and instructor preferences respectively and ranged between 0 and 10. For problems R2, R3, and R5 this range was 0 to 5 reflecting a lack of preferences for times. Second order preferences ("soft" conflicts) corresponding to the quadratic term in (1) were generally very small.

Six pseudo real problems were created from the real problems by setting all linear coefficients in equation (1) (corresponding to first order preferences) to a value of one. This effec-

tively removed the linear term from the objective function (equation (1)). Since relatively few "soft" conflicts existed, this nearly removed the objective function from consideration and provided an additional set of problems for comparison.

3.1.2 Synthetic Problems

Ninety-six synthetic problems were obtained for final testing and evaluation using a random problem generator. The generator is designed to model the domain from which the real problems were drawn. This is done by generating a set of "choices" corresponding to resource-time combinations for each choice set representing tasks (e.g. sections to be taught).

Choices for each choice set, s, are obtained by first sampling the number of rooms and times, $N_r(s)$ and $N_t(s)$, and then creating $N_x(s) = N_r(s)N_t(s)$ choices. Resource and time indexes for individual choice sets are sampled from Poisson distributions with parameters $N_r/2$ and $N_t/2$ respectively where N_r and N_t are the number of available resources and times. $N_r(s)$ and $N_t(s)$ are both sampled from binomial distributions. Distribution parameters were based on $N_r(s)$ and $N_t(s)$ maximum ($Max[N_r(s)]$ and $Max[N_t(s)]$), and expected ($E[N_r(s)]$ and $E[N_t(s)]$) values.

A sampling population of $N_t = 200$ time patterns was assumed for all cases. Values of $Max[N_t(s)] = 20$ and $E[N_t(s)] = 10$ were also used. Likewise, $Max[N_r(s)] = 4$ and $E[N_r(s)] = 2$ were assumed. The size of the sampling population for rooms, N_r, varied with problem size and the case considered, however. Small (large) N_r values, by forcing high (low) average room use, resulted in high (low) hard conflict levels.

Objective function and constraint matrix coefficients were also generated randomly. Choice "values" corresponding to coefficients in the linear term of the MCQVP objective function (equation (1)) averaged 5 for all problems with the range varied for different problem cases.

Hard choice conflicts result when two choices share a common primary resource or their respective choice sets "interact" (e.g. share a common secondary resource), and the choice times "overlap". Choice set clusters are used to model the tendency for organizations to be formed around a common set of resources and, in some sense, concentrate the potential for resource conflict. For example, universities organize into departments with instructors and many of the students in the department and, often, with a set of rooms under their control. We reflect this in synthetic problems by making the probability that two choice sets interact depend on whether they belong to the same or different clusters.

Similarly, soft conflicts occur when two choices' tasks have a "soft" paired interaction preference (e.g. back-to-back scheduling) and their times don't satisfy this requirement. The generator samples attributes for pairs of tasks and times from joint distributions specified on input. The cost of not satisfying paired preferences (of incurring soft conflicts) is sampled from a binomial distribution.

Forty-eight problem cases were considered by varying the level of each of the following five problem factors:

1. Number of choice sets (tasks)
2. Number of choice set clusters
3. Linear choice coefficient (value) range
4. Hard conflict density
5. Soft conflict density

The levels of each factor are shown in Table 2. As shown, three levels were used for the number of choice sets and the other four factors were set to two levels each for a total of 48 combinations.

Table 2: Synthetic Problem Factors

Choice Sets	Factor Level	Clusters	Value Range	Conflict Density	
				Hard	Soft
100	LOW	1	2	.006	.000
	HIGH	3	10	.037	.011
200	LOW	1	2	.004	.000
	HIGH	5	10	.018	.005
500	LOW	1	2	.002	.000
	HIGH	10	10	.011	.002

Comparison of the values in Table 2 with those in Table 1 shows that the conflict densities of the generated problems roughly predict those of the six real instances available. High levels are actually somewhat higher than those of real data but decrease with problem size as do the real problem densities.

Soft and hard conflict levels in the synthetic problems are a result of the sampling process and are a function of paired choice set and paired time pattern attributes. In conjunction with high values for N_r, low hard conflict levels were obtained by specifying overlap and choice set interaction densities of .05 and .03 for all problem sizes. High hard conflict levels were obtained with densities of .30 and .25 and low values for N_r.

The low level was set to zero because most of the real problems had very few soft conflicts, while the high level was set large enough to encompass likely values. The average soft conflict density for pairs of choices is also shown in Table 2. The 96 test problems resulted when each of the 48 factor combinations was replicated. All problems were generated with one feasible solution embedded.

The characteristics of individual problem instances varied considerably from the averages shown in Table 2. Maximum hard conflict densities were nearly twice the average values shown, for example. Detailed problem characteristics are provided in [MOON91].

3.2 Algorithm Variations

The algorithms considered in this study are all variations of the one shown in Figure 1 that were developed for evaluating the effectiveness of strategic oscillation and stochastic diversification methods. Input consists of an instance of MCQVP including definition of the multiple choice sets (tasks) and the variables (choices) for each set. First and second order (soft conflict) preference values are also input for each choice as are all of the conflicting pairs of choices.

Output is a set of assignments, one for each choice set. These assignments yield the maximum value for the minimum number of violated vertex packing (conflict) constraints found. Other output documents the search and results.

Hundreds of variations on the algorithm in Figure 1 were tried during the course of this study. All variations use pairwise exchange moves within a single choice set. This guarantees satisfaction of the multiple choice constraints. Most incorporated differences in implementing Steps 2 and 3.

The seven variants shown in Table 3 were selected for further study. As shown, differences are restricted to methods used to identify candidate move sets, select moves from among the candidates, and restrict admissible moves through the use of tabu lists.

Table 3: Algorithm Variant Features

Alg	Candidate Move Set Selection	Move Selection Rule	Move Evaluation Function
MS	All	Best improving	Lexicographic
TS	All	Best non-tabu	Lexicographic
TSO	All	Best non-tabu	Oscillating B
RTS	Random sample	Best non-tabu in sample	Lexicographic
RTSO	Random sample	Best non-tabu in sample	Oscillating B
DBS	Biased sample	Best non-tabu in sample	Lexicographic
DBSO	Biased sample	Best non-tabu in sample	Oscillating B

Variations include "strawman" multistart (MS), Tabu Search (TS), and Tabu Search with strategic oscillation (TSO). Random Tabu Search (RTS) and random Tabu Search with oscillation (RTSO) evaluate moves associated with 30 percent of the choice sets selected randomly at each iteration but in other ways are identical to TS and TSO respectively.

Finally, Tabu ideas are implemented in the dynamic biased sampling (DBS) and DBS with oscillation (DBSO) algorithms without the use of tabu lists.

Details concerning these strategies are given below. Mechanics common to all variants are given in [MOON90] along with derivations of move selection and search status update equations. Results obtained running each algorithm for the test problems are reported in a later section.

3.2.1 Candidate Move Set Selection

Two strategies are used for selecting candidate move sets in algorithm variations:

1. Include all moves in the neighborhood
2. Include all moves for a sample of the choice sets

The first approach was used in the MS, TS, and TSO variations, with one of two sampling methods used in the rest.

Evaluation of all the pairwise interchange moves possible across all of the multiple choice sets seems desirable. However, sampling move subsets has at least two potential benefits. First, since fewer moves need to be evaluated, more iterations should be possible over a fixed time, depending on sample size and sampling efficiency.

A second potential sampling benefit is diversification of the search into new regions of the search space. The degree of diversification is a function of the sample size. For example, if only one move were sampled at a time as in simulated annealing (see [KIRK83]), the solution space would be covered evenly. The tradeoffs here are that the search may miss good local optima and a good deal of time can be wasted evaluating moves in poor regions with too much random diversification. Results obtained with two different sampling methods are presented. A uniform sample of choice sets was drawn at each iteration in the RTS and RTSO variations. A more complicated priority based method was used in DBS and DBSO. Sample sizes equal to 30 percent of the number of choice sets (tasks) were found during calibration runs to provide the best results.

The idea of the *dynamic, biased sampling* approach used in DBS and DBSO is to combine random and strategic diversification techniques. These algorithms select choice sets for move evaluation with a bias toward high priority choice sets. The priority reflects both the time since a choice set has been sampled and the number of times it has been selected for an exchange. Choice sets which have not been checked for a long time and have had their assignments changed the fewest times are favored for move evaluation. Thus, both exchange frequency and evaluation recency are considered.

A lowest value first (LVF) choice set queue based on the priority is maintained during the search. A truncated geometric distribution is used for sampling from this queue. As implemented, the distribution shape parameter, p, was moved from a value of $.28/N_s$ to

$1.4/N_s$ during the first $4N_s$ iterations where N_s is the total number of choice sets. This has the effect of beginning the search with relatively small bias in the sampling and increasing the degree of biasing as the algorithm "learns" which sections to favor. It also appears to make performance less sensitive to the parameter value.

3.2.2 Move Evaluation and Selection

All algorithm variations exchange a choice, x_j, for the current choice, x_k, for some choice set, $s(k) = s(j)$ at each iteration where $s = s(k)$ is a member of the set of candidate choices. Move selection requires a search for the best exchange available among the candidate choice sets and subject to restrictions on admissibility. With the exception of multistart (MS), all algorithm variations presented here choose the best move available in the candidate move set chosen for evaluation. The multistart algorithm uses a "best improving" rule and restarts the search at a random solution when no improving moves are available.

In all algorithm variations exchanges are compared according to their relative change in hard conflict feasibility ($C_{xx}^{(2)}$) and objective function value (Z). The net improvement expected if x_j enters the solution is given by

$$r_j = \bar{Z}(j) - B\bar{C}_{xx}^{(2)}(j) \tag{5}$$

where,

$$\bar{Z}(j) = \bar{C}_x(j) - \bar{C}_{xx}^{(1)}(j) \tag{6}$$

and

$$\begin{aligned} \bar{Z}(j) &= \text{The increase in } Z \text{ if } x_j \text{ enters} \\ \bar{C}_x(j) &= \text{Increase in linear part of } Z \text{ if } x_j \text{ enters} \\ \bar{C}_{xx}^{(1)}(j) &= \text{The increase in soft conflicts if } x_j \text{ enters} \\ \bar{C}_{xx}^{(2)}(j) &= \text{The increase in hard conflicts if } x_j \text{ enters} \\ B &= \text{hard conflict (constraint) multiplier} \end{aligned}$$

Parameter B is set to reflect the relative importance of hard conflict feasibility and objective function value. In general, B can vary between 0 and B_{max} where B_{max} is "large" enough so that hard conflicts dominate move comparison, with changes in Z affecting move selection only when no reduction in hard conflicts is possible.

The lexicographic move evaluation function requires that tradeoffs between feasibility and objective function improvements are resolved in favor of feasibility. That is, the objective function is used to break ties between moves which will improve feasibility the most. The function is used in the MS, TS, RTS, and DBS algorithms and is equivalent to setting $B = B_{max}$ for all evaluations.

Algorithm variations incorporating strategic oscillation (TSO, RTSO, and DBSO) dynamically vary B between 0 and B_{max} according to

$$\ln(B) = B_{max} e^{-b/i} \sin(4\pi^2 i/p^2) \tag{7}$$

The sinusoidal function in (7) has period, p, amplitude, B_{max}, and damping time constant, $1/b$ parameterized. Parameter values $B_{max} = 3N_s$ and $p = 2.5N_s$ were found experimentally to provide good overall performance during calibration runs made with a separate set of test problems. The parameter b, was set close to zero for all runs to ensure mimimal damping. This function was chosen as the most natural and straight forward implementation of the oscillation strategy.

At each iteration $\bar{C}_x(j)$ and $\bar{C}_{xx}^{(p)}(j)$ need only be updated for the choice sets involved in the interchange. Efficient ($O(N_x)$) $\bar{\mathbf{C}}_x$ update following an exchange is straightforward. If possible, the $O(N_s N_x)$ calculations needed for $\bar{\mathbf{C}}_{xx}^{(p)}$ initialization should be avoided during update. In fact, these values may also be updated in $O(N_x)$ time as shown by Mooney and Rardin [MOON90].

3.2.3 Tabu Move Restrictions

Tabu move restrictions are used in several of the variations to increase diversification while allowing the search to intensify around good local optima [GLOV90A]. A single tabu list was used in these variations, with the "entering" and "leaving" choices involved in an exchange placed on the list.

Several alternative tabu, aspiration criteria and list lengths were tried during algorithm screening and calibration. The tabu lists used consist of simple fixed length circular queues upon which both the entering and leaving solution variables were stored for each exchange made.

A tabu criteria which prevents entering choices from leaving while the move is on the tabu list was found to be most effective. This amounts to blocking all moves for the choice set chosen for an exchange during the move's tenure on the list. A tabu list length equal to 40 percent of the number of choice sets was found experimentally to produce the best results. The aspiration criteria allows override of the tabu status if a move produces the best solution found to date.

3.3 Performance Measures

Heuristic performance evaluation requires a number of measures and necessarily has a statistical element. Performance measures presented here address the two dimensions of search algorithm performance:

1. Quality of the best solution found
2. Time to find the best solution

Solution quality measures indicate how good the best solution is in terms of feasibility and objective function value. The time required (of a maximum allowed) to find the best solution

is indicative of overall search efficiency but can be evaluated only in the context of solution quality. Other measures document general algorithm characteristics.

Data for computing the measures was obtained by running each algorithm for a problem dependent search time. Run times were obtained on an IBM RISC 6000 model 520 workstation. Programs were compiled with a relatively new version of the "cc" compiler. Code optimization was not used due to the unreliability of the resulting programs.

Absolute and relative algorithm performance measures are used in order to give the most complete picture possible. The measures are computed for each algorithm by first obtaining average performance across starting points for each problem. These values are then averaged across problems for each algorithm. Varying problem sizes mandate the use of ratios for most measures.

Absolute measures are independent of the experiment in the sense that computation does not require knowledge of other algorithms' performance. The absolute measures used here are average search iterations per second, diversification level, minimum hard conflicts found, and the fraction of total search time required to reach the minimum hard conflicts and to find the best solution. Iterations per second gives an idea of the efficiency of alternative move identification and selection approaches. Average minimum hard conflicts is indicative of the solution quality. Time fractions measure the efficiency of an algorithm in arriving at the best solutions it found, but can be evaluated only in the context of overall solution quality.

Diversification levels are important to this study as one of the goals was to determine the validity of the hypothesis that more diversification would be beneficial. The index which is presented here is based on the idea that an algorithm that diversifies the search must cover the search space more or less evenly. One way to interpret this is to say that such an algorithm would perform the same number of exchanges, on average, for each of the N_s choice sets.

We therefore desire a diversification index, δ, that could be computed for a particular search which would have a value of 1 if "perfect" diversification were achieved, and 0 if none were obtained. These cases might correspond, for example, to completely random sampling, and a case where all exchanges were made in one choice set, respectively.

Now, let e_s be the number of exchanges performed for choice set, s, through some iteration, i, where

$$\sum_s e_s = i$$

We let

$$\delta = 1 - \sqrt{\frac{N_s \sum_s e_s^2 - i^2}{i^2(N_s - 1)}} \tag{8}$$

Notice that $1 - \delta$ is equal to the sample standard deviation of the number of exchanges per section times $1/i$. An interpretation of δ as a measure of how evenly the search covers the search space therefore seems reasonable. Also, it is easy to verify that if $e_s = K$ for all s,

where K is some constant, $\delta = 1$. Similarly, if $e_s = i$ for some s and 0 for the rest, $\delta = 0$ as required by the definition.

This measure clearly applies only to diversification in the choice set space and neglects explicit measurement of the coverage of choices within a choice set. We believe this is valid because of observations made during this study that suggest that choice space diversification is the primary issue. Specifically, tabu criteria which prevent current choices from leaving the solution are much more effective than those which prevent choices from entering. Also, observed search trajectories show little evidence of choice cycling when adequate diversification is achieved for choice sets.

Relative measures presented here include the average objective function value attained when a minimum conflict solution was first found, ZF, and when the best solution was found, ZB. These values are expressed as a fraction of the maximum solution found (averaged over starting points) by any algorithm for the problem, Zmaxp. As such, they can only be interpreted in the context of the experiment involving the other algorithms.

4 Results and Conclusions

Results obtained by each of the seven algorithms described in section 3.2 are summarized in Table 4 for the synthetic problems. Tables 5 and 6 present results obtained using real test problem data. Columns in these tables correspond to the measures described in the previous section of this paper. Three rows showing average, minimum and maximum values (of averages over starting points) of each measure are shown for each algorithm. These values document expected, best, and worst case performance.

4.1 Synthetic Problems

The values in Table 4 are summarized for a total of 96 synthetic problems (48 cases, 2 replicates each). Starting points for each case and replicate were randomly sampled. All algorithms were initialized at the same starting points. The 100, 200 and 500 choice set problems described in section 3.1, are represented. Each algorithm was run for 3, 7, and 40 minutes of search time and averaged .13, .51, and 3.17 minutes for initialization for 100, 200 and 500 choice sets respectively.

Referring to Table 4, it can be seen that the total iterations possible in a fixed time is the lowest for MS. The deterministic Tabu methods (TS, TSO) are much faster, reflecting the savings from not restarting. The sampled methods are still faster, although this doesn't always result in a better solution. With the exception of dynamic biased sampling (DBS), all algorithms attained relatively high diversification. As expected, the algorithms incorporating strategic oscillation (TSO, RTSO, DBSO) consistently resulted in higher diversification than their non-oscillating counterparts (TS, RTS, DBS).

Table 4: Synthetic Problem Results

Alg.	Value	Iter/ Sec	Dvrs Indx	Min. Hard Con.	ZF/ Zmaxp	IterF/ Niter	ZB/ Zmaxp	IterB/ Niter
MS	avg	2.60	.985	0.0	.777	.062	.986	.535
	min	.85	.977	0.0	.485	.015	.939	.217
	max	5.09	.991	0.0	.983	.282	.999	.964
TS	avg	3.29	.937	0.0	.777	.050	.995	.552
	min	1.03	.895	0.0	.485	.012	.963	.204
	max	6.46	.967	0.0	.984	.194	1.000	1.000
TSO	avg	3.22	.957	0.0	.781	.052	.999	.567
	min	1.00	.903	0.0	.491	.013	.971	.273
	max	6.30	.986	0.0	.992	.317	1.000	.926
RTS	avg	3.55	.945	0.0	.780	.046	.995	.550
	min	7.11	.913	0.0	.486	.013	.970	.243
	max	6.90	.968	0.0	.987	.158	1.000	.935
RTSO	avg	3.51	.960	0.0	.785	.052	.997	.522
	min	1.10	.917	0.0	.507	.014	.961	.224
	max	6.84	.987	0.0	.994	.593	1.000	.852
DBS	avg	3.51	.797	0.0	.782	.047	.986	.763
	min	1.05	.778	0.0	.499	.014	.940	.211
	max	6.90	.823	0.0	.978	.133	.999	.995
DBSO	avg	3.46	.893	0.0	.785	.048	.997	.665
	min	1.04	.813	0.0	.509	.015	.936	.169
	max	6.80	.956	0.0	.976	.135	1.000	.998

All of the algorithms were able to obtain feasible solutions for all of the synthetic problems. These initial feasible solutions (ZF) had objective functions of about 78 percent of the best found, and they were found in about 5 percent of the total run time on average. Considerable variation in the best and worst case objective function and timing occurred, thus indicating variation in problem structure and the quality of the starting points.

Similarly, all of the algorithms returned, on average, best objective function values (ZB) greater than 98 percent of the best found. The best and worst case for ZB differed much less than those of ZF, ranging between approximately 94 and 97 percent. However, the times to achieve the best solutions found varied significantly between algorithms with average values between 53.5 and 76.3 percent of total search time. Overall, the TSO algorithm appeared to perform the best with average and minimum ZB values within 99.9 and 97.1 percent of the best found and search times to find its best solution second only to RTSO.

It is important to consider how well the algorithms did in absolute terms given the consistently high relative values shown in Table 4. A simple bound can be computed for this

purpose by summing the maximum coefficients for each choice set in the linear term of the objective function. When this was done for all of the synthetic problems, it was found that the average best value found, ZB, was 94 percent of this bound. This may indicate partly that the problems were relatively "easy" in terms of soft and hard conflicts. It certainly indicates that very good solutions are being returned, especially by the better performing algorithms.

Figure 5 provides a graphical view of the relative solution quality and search time performance for each of the algorithms. Scatter plots are shown for each problem size class as well as the combined set. Notice that markers for the oscillating algorithms are filled versions of those for their non-oscillating counterparts. The axes are solution time and quality "rank" indexes which vary from a zero (worst) to one (best). One can easily visualize solution time-quality tradeoffs where they exist.

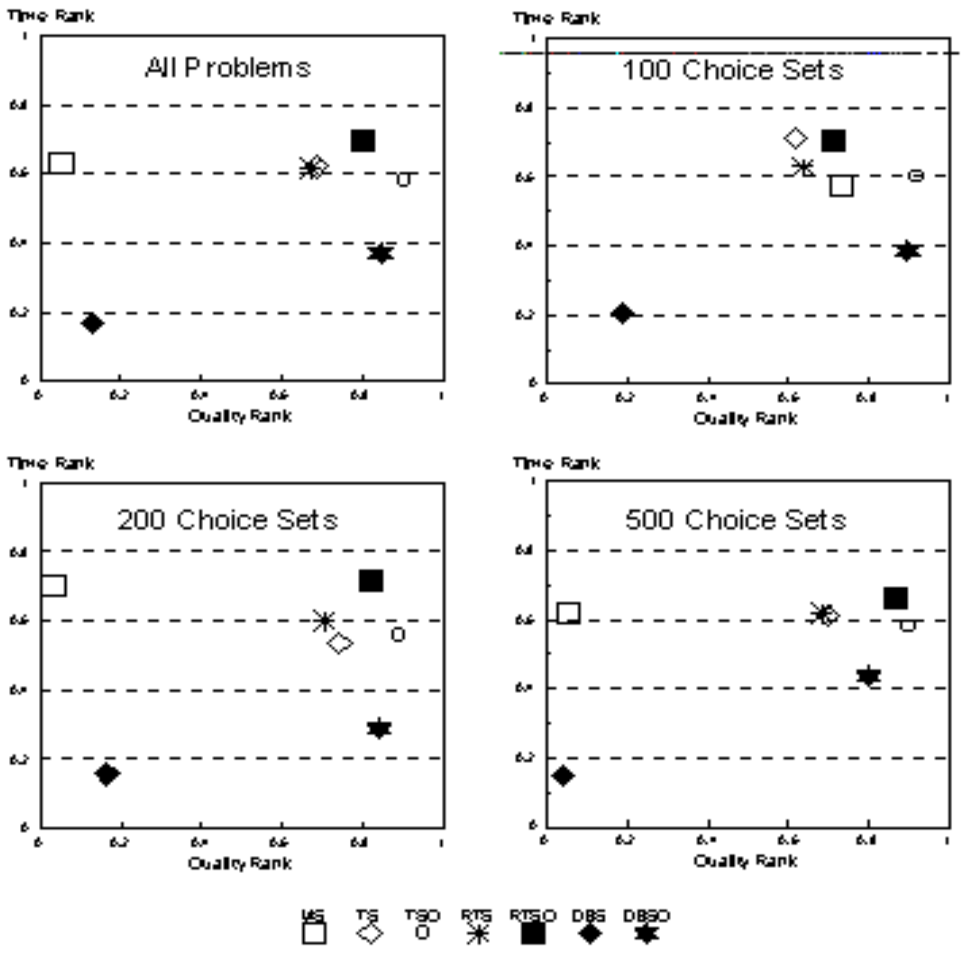

Figure 5: Synthetic Problem Algorithm Ranking

Time and quality ranks were obtained by scaling the values for ZB/Zmaxp and IterB/Niter obtained by the seven algorithms. The scale used corresponded to setting the value(s) for the

worst and best algorithms to 0 and 1 respectively. Algorithm ranks for individual problems were then averaged across all problems to obtain an overall ranking for each algorithm. The "best" algorithm (relative to the others) would appear in the upper right hand corner.

A number of results are apparent in Figure 5. First, the three algorithms featuring strategic oscillation, TSO, RTSO, and DBSO consistently return better quality solutions than their non-oscillating analogs. The picture is a little less certain for the 100 choice set problems but can be explained due to the fact that these are the easiest problems. Also, although TSO tends to produce better solutions on average, there may be a tradeoff between quality and time with RTSO. The better time rank is likely due to the effect that choice set sampling has on reducing iteration times for RTSO.

4.2 Real and Pseudo Real Problems

Two sets of runs were made using data obtained from the real university course scheduling problems described in section 3.1 as part of this study. The first set of runs was based on original data used to schedule actual course sets. The second, "pseudo real", problem set was obtained from the original problems by eliminating the linear term in the objective function.

Results obtained applying the algorithms to the original problems are summarized in Table 5. Each algorithm was run for each problem from the same 4 starting points. The starting points for each problem included 3 randomly sampled solutions and a "given" solution based on the previous year's schedule. Measures tabulated in the columns are the same as those presented for the synthetic problems.

The results in Table 5 are similar to those obtained with the synthetic problems except that DBSO performs better relative to TSO. As before, all algorithms produce excellent objective function values. Interpretation is more difficult, however, as none of the algorithms always returned a feasible solution. In fact, MS had a worst case performance of 3.5 average hard conflicts (on problem R5).

DBS had the fewest average and maximum hard conflicts (of the average across starting points). Analysis of individual problem results indicated that DBS found feasible solutions 83 percent of the time whereas DBSO found feasible solutions in 79 percent of the runs. Both were able to find feasible solutions for all problems at least once in four runs made from different starting points. However, DBSO returned slightly better objective function values on average when the minimum conflict solution (ZF) was obtained as well as overall (ZB). The similarity in the results for oscillation and non-oscillation strategies can be explained in part by the fact that complete first order preferences were available for only some of the problems and even room preferences tended to be very close together. The next best algorithm, TSO, found feasible solutions 63 percent of the time.

Table 6 shows the results for the pseudo real problems. As for the original real problems, DBS and DBSO provide virtually identical results. Notice also that the oscillation algorithms (TSO, RTSO, DBSO) all obtain diversification identical with their non-oscillating

Table 5: Real Problem Results

Alg.	Value	Iter/ Sec	Dvrs Indx	Min. Hard Con.	ZF/ Zmaxp	IterF/ Niter	ZB/ Zmaxp	IterB/ Niter
MS	avg	3.74	.939	1.2	.951	.255	.995	.454
	min	.46	.875	0.0	.813	.017	.984	.375
	max	15.06	.967	3.5	.994	.512	1.000	.544
TS	avg	5.42	.918	.6	.957	.152	.998	.326
	min	.79	.857	0.0	.813	.011	.996	.107
	max	23.24	.954	2.2	.997	.308	1.000	.747
TSO	avg	5.32	.924	.6	.957	.122	.999	.303
	min	.72	.866	0.0	.815	.012	.998	.105
	max	22.73	.958	2.0	.998	.280	1.000	.486
RTS	avg	6.80	.927	.6	.959	.152	.998	.313
	min	.86	.891	0.0	.839	.010	.996	.071
	max	28.09	.955	1.8	.996	.449	1.000	.806
RTSO	avg	6.74	.931	.6	.963	.120	.999	.360
	min	.83	.891	0.0	.829	.012	.998	.206
	max	27.80	.960	2.0	.996	.221	1.000	.562
DBS	avg	7.31	.921	.2	.954	.089	.998	.287
	min	.90	.821	0.0	.819	.010	.994	.131
	max	29.39	.959	.5	.994	.201	1.000	.741
DBSO	avg	7.25	.932	.4	.960	.106	.999	.322
	min	.88	.836	0.0	.822	.010	.996	.171
	max	29.25	.966	1.5	.998	.217	1.000	.468

counterpart (TS, RTS, DBS). Other results are nearly identical as well. This illustrates the value of the biased sampling strategy where the objective function cannot be used effectively to diversify the search.

Figure 6 illustrates the relative performance of the seven algorithms on the complete set of real problems including both original and pseudo instances. The two plots show solution time and feasibility ranks versus a "Z" rank. Average objective function ratios (ZB/Zmaxp and ZF/Zmaxp) were scaled between zero and one across algorithms for each problem to compute the Z-rank. Algorithms finding the best (maximum objective for minimum conflicts) solution were given ranks of one and those with the worst were given zero.

The left plot in Figure 6 has the same scales as those shown in Figure 5 for the synthetic problems. The axes are solution time and quality (Z) "rank" indexes. Solution time ranks were computed, similarly to the Z-ranks, by scaling the time to obtain the best solution as a fraction of total search time between zero and one for each algorithm. Ranks for individual problems were then averaged across all problems for each algorithm.

The "Z" ranks are hard to interpret since, unlike the synthetic problem runs, feasible solutions were not always found. Notice that the algorithms rank on average between approximately .25 and .57 with respect to objective function value. This indicates relative

Table 6: Pseudo Real Problem Results

Alg.	Value	Iter/ Sec	Dvrs Indx	Min. Hard Con.	ZF/ Zmaxp	Iter/F Niter	ZB/ Zmaxp	Tbest IterB/ Niter
MS	avg	3.04	.932	2.6	.895	.266	.989	.292
	min	.41	.881	0.0	.704	.020	.946	.055
	max	12.26	.965	13.5	1.000	.530	1.000	.563
TS	avg	5.61	.915	.6	.927	.125	.988	.126
	min	.38	.864	0.0	.729	.010	.946	.015
	max	24.92	.952	2.0	1.000	.302	1.000	.303
TSO	avg	5.46	.915	.6	.928	.141	.991	.182
	min	.38	.864	0.0	.729	.010	.946	.016
	max	24.18	.952	2.0	1.000	.357	1.000	.598
RTS	avg	7.06	.923	.5	.929	.121	.989	.151
	min	.67	.888	0.0	.690	.009	.946	.020
	max	29.46	.953	1.8	1.000	.264	1.000	.377
RTSO	avg	6.99	.923	.5	.929	.099	.991	.134
	min	.66	.888	0.0	.690	.010	.946	.020
	max	29.14	.953	1.8	1.000	.234	1.000	.240
DBS	avg	7.75	.949	0.0	.945	.100	.998	.136
	min	.95	.887	0.0	.729	.009	.987	.023
	max	31.24	.978	0.0	1.000	.202	1.000	.290
DBSO	avg	7.68	.949	.1	.947	.103	1.000	.129
	min	.94	.887	0.0	.729	.009	1.000	.023
	max	30.99	.978	.5	1.000	.212	1.000	.250

weak dominance of the oscillation algorithms over the others as would be expected when the pseudo problems are combined with the original problems. The MS, TS and RTS algorithms do form a cluster, however.

The plot on the right shows relative performance with respect to objective function and feasibility. These values were also computed averaging individual problem ranks for each algorithm. Individual problem ranks were computed by scaling the number of conflicts with minimum conflicts (found by any algorithm) having a rank value of one and the maximum producing a value of zero.

With respect to feasibility, the DBS and DBSO algorithms demonstrate a clear relative advantage. DBS found the minimum conflict solution found on average 97.4 percent of the time while DBSO was able to find it 93.8 percent of the time. The other algorithms all clustered about the 70 percent level. Overall, DBSO and DBS appear to perform the best for these problems.

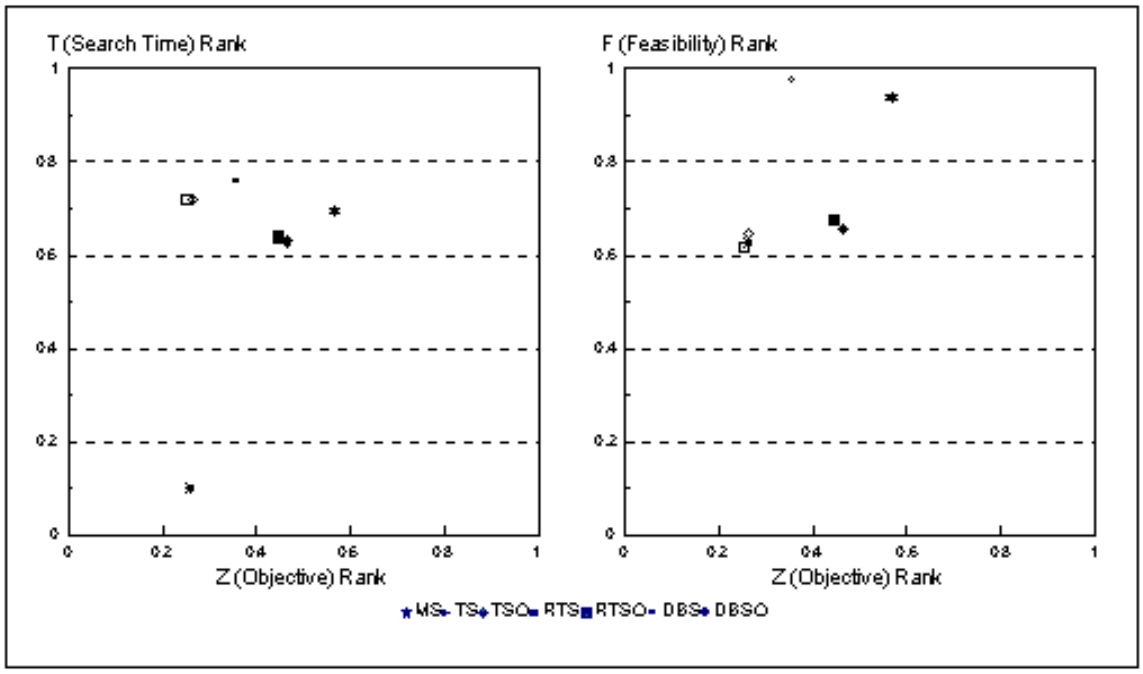

Figure 6: Real Problem Algorithm Ranking

4.3 General Conclusions

Several conclusions can be drawn as a result of this work. First, local search algorithms appear to be well suited for solving the class of problems studied here. Also, the basic Tabu Search principles are further validated by these results. Application of the principles of aggressive search bias and balance between diversification and intensification resulted in significantly better algorithms when compared to multistart. We have also shown that the use of short term memory embodied in the basic Tabu Search methodologies, coupled with greedy move selection can achieve adequate diversification for relatively small problem instances.

However, as shown in Figure 3 short term memory alone cannot move the search away from large plateaus. Indeed, reliance only on short term memory ultimately results in cycles whose period is related to the length of the tabu list. This observation led us to hypothesize the need for additional, strategic diversification. We were interested, furthermore, in developing such strategies consistent with the overall Tabu philosophy.

We have made several observations concerning diversification as a result of our experiments with strategic oscillation and stochastic Tabu algorithms. First, based on a diversification measure developed for this class of problems and local search neighborhood, it appears that a significant degree of diversification is necessary for good algorithm performance. All algorithms presented here provided diversification levels on average greater than 80 percent. This is no coincidence, since algorithms (and parameter values) which resulted in poor per-

formance were eliminated early in the study. Nonetheless, some evidence remains as shown in Table 4; one of the two poorest algorithms (DBS) averaged 79.7 percent diversification.

A second, more important, conclusion can also be made as a result of this work. Referring again to Table 4, it is shown that the multistart (MS) algorithm obtained diversification levels of 98.5 percent on average. In spite of this, however, it ranked with DBS as the poorest performer with respect to solution quality and faired worse as the problem size increased. This is likely a result of the totally random diversification strategy used. Clearly, while relatively high diversification appears to characterize the best performing algorithms, it is not sufficient to guarantee good solutions!

The evidence therefore supports the notion that effective local search heuristics require aggressive, "strategic" (i.e. goal directed) diversification. The oscillation strategy fits this definition, and the results of the experiments demonstrate the superiority of this approach. The idea is to periodically drive the search into "good" regions where good regions contain either super-optimal or feasible solutions (see Figure 4). The three variants incorporating the oscillation approach (TSO, RTSO, DBSO) consistently out performed their non-oscillating counterparts, especially with respect to feasibility (see Figures 5 and 6).

In general, it appears that either the TSO or DBSO algorithms will perform very well on the class of problems considered here. TSO performed slightly better on the synthetic problems, but DBSO appeared to be superior on the real problems. This may be due to the existence of some structure in the real problems which was not modeled exactly by the generator. The possibility of serendipity allowed by the stochastic DBSO may in some way get around this. Notice, however, that even the sampling in DBSO is consistent with an aggressive search philosophy.

Additional work can be done in many areas. Of particular interest might be other aggressive diversification strategies which do not require the long cycles that our oscillation approach does. Also, there is the possibility of a theory relating the oscillation frequency to "natural" cycles in the problems (e.g. the number of choice sets appears to be important). Finally, further testing and refinement of DBSO and TSO in other problem domains should be of interest.

References

[AHO74] Aho, V.A., Hopcroft, J.E., Ullman, J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1974, 470 pages.

[BAKE74] Baker, K.R., *Introduction to Sequencing and Scheduling*, John Wiley and Sons, 1974, 305 pages.

[GLOV89A] Glover, F., "Tabu Search — Part I", *ORSA Journal on Computing*, 1:3, Summer 1989, 190-206.

[GLOV89B] Glover, F., "Tabu Search — Part II", *ORSA Journal on Computing*, 2:1, Winter 1990, 4-31.

[GLOV89C] Glover, F. and Laguna, M., "Target Analysis to Improve a Tabu Search Method for Machine Scheduling", *Working Paper, Center for Applied Artificial Intelligence*, University of Colorado, September 1989, 23 pp.

[GLOV89D] Glover, F. and Greenberg, H.J., "New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence", *European Journal of Operational Research*, 39, 1989, 119-130.

[GLOV90A] Glover, F., "Tabu Search: A Tutorial", *Working Paper, Center for Applied Artificial Intelligence*, University of Colorado, February 1990, 60 pages.

[KIRK83] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P, "Optimization by Simulated Annealing", *Science*, 220, 671-680, 1983.

[LAGU90A] Laguna, M. and Glover, F., "On Target Analysis and Diversification in Tabu Search", *Working Paper*, Graduate Program in Operations Research, The University of Texas at Austin, April 1990, 16 pages.

[MOON90] Mooney, E.L. and Rardin, R.L., "Local Improvement Heuristics for Multiple Choice Vertex Packing Problems", *Institute for Interdisciplinary Engineering Studies Technical Report CC-90-30*, Purdue University, November 1990, 28 pages.

[MOON91] Mooney, E.L., "Tabu Search Heuristics for Resource Scheduling with Course Scheduling Applications", *Unpublished Ph.D. Dissertation*, Purdue University, 1991, 179 pages.