Solving the Repeated Partition Scheduling Problem Using Metaheuristics

Zvi Goldstein
Zvi Drezner
California State University Fullerton

In this paper we solve the repeated partition scheduling problem by the three metaheuristics: simulated annealing, tabu search, and genetic algorithms. The procedures are described and particular parameters for each of the procedures are established. Computational experiments on twenty-two test problems show that the best procedure is the tabu search.

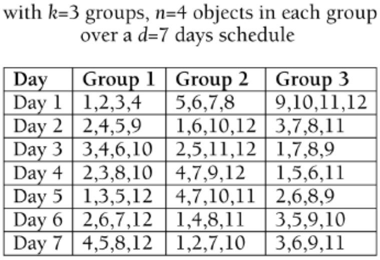
The repeated partition scheduling (RPS) problem was presented in Drezner (1998). The problem is based on a group of objects (persons) who repeat scheduled meetings in smaller groups for several days. A group of nk objects is to be divided into k groups of n members each. Different partitions are scheduled for different days and the complete schedule spans over d days. Our objective is that each pair of objects meets about the same number of times.

Applications for tournament scheduling, personnel assignment, production scheduling, advertising, experimental design, and others are described in Drezner (1998). The problem was motivated by a golf tournament organizer. Twelve players are participating in a tournament for seven days. They are divided into three groups of four players each. The grouping usually changes from one day to another. It is important that each player meets every other player (i.e. be with him/her in the same group) at least once. Also, it is desirable that each pair meets about the same number of times. For this particular problem there are sixty-six pairs and 126 pair-wise meetings throughout the seven days. The best possible solution is having sixty pairs meet twice and six pairs meet once. However, such a solution has not been found yet.

After a long manual effort for some years, the organizer found a solution with 12 pairs meeting three times, thirty-six pairs meeting twice, and eighteen pairs meeting once. By randomly generating computerized solutions and applying crude improvement procedures a solution with nine pairs meeting three times was found. The best known solution (see Drezner (1998) and the example described below) involves only three pairs meeting three times, fifty-four pairs meeting twice, and nine pairs meeting once, quite an improvement compared to the original schedule.

An example of a solution is given in the following table. Note that this is actually the best known solution. It was implemented by the organizer for the golf problem where twelve players are playing a tournament over a seven-day period with three groups playing each day.

Table 1: *Solution to the Problem*

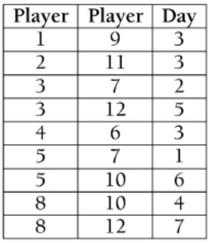


with k=3 groups, n=4 objects in each group
over a d=7 days schedule

Day	Group 1	Group 2	Group 3
Day 1	1,2,3,4	5,6,7,8	9,10,11,12
Day 2	2,4,5,9	1,6,10,12	3,7,8,11
Day 3	3,4,6,10	2,5,11,12	1,7,8,9
Day 4	2,3,8,10	4,7,9,12	1,5,6,11
Day 5	1,3,5,12	4,7,10,11	2,6,8,9
Day 6	2,6,7,12	1,4,8,11	3,5,9,10
Day 7	4,5,8,12	1,2,7,10	3,6,9,11

The table demonstrates the characteristics of the problem. Every day all twelve players are assigned to groups with no repetition. If we check all sixty-six pairs we find only three pairs that meet three times. Players 3 and 10 meet on days 3, 4, and 6; players 5 and 12 meet on days 3, 5, and 7; and players 7 and 8 meet on days 1, 2, and 3. All other sixty-three pairs meet either once or twice. Nine pairs meet once and fifty-four pairs meet twice. The nine pairs meeting only once are listed in Table 2 below.

Table 2: *Pairs Who Meet Only Once*



Player	Player	Day
1	9	3
2	11	3
3	7	2
3	12	5
4	6	3
5	7	1
5	10	6
8	10	4
8	12	7

An identical formulation can be applied to workers in a very stressful environment. The task requires dividing the workers into k groups of n workers to perform d sets of k activities each. If two workers are assigned to the same team too many times, they may experience negative outcomes, for example, they may "get on each other's nerves." The objective is therefore to arrange a schedule such that every pair meets about the same number of times. When twelve workers need to be divided into three groups of four workers in each group for seven activities, then the solution in Table 1 can be used for such a scheduling problem.

Another application is planning a schedule for *nk* machines (or workers or types of resource) that requires dk tasks to be completed in one day. Each task requires *n* machines, so *k* tasks can be performed simultaneously and *d* periods are required for the completion of the whole project. The schedule is planned ahead of time so no changes are possible during the day. If one machine is out of order for a task (or a worker does not show up or a resource is unavailable), the task can be completed but some extra cost is incurred. However, if two machines are out of order for the same task, the task cannot be completed, which adds a significant cost to the operation. We would therefore wish to design a schedule such that the number of times that each pair of machines is assigned to the same task is about the same. If a certain pair of machines is assigned to the same task many times, then the operation will suffer if this particular pair of machines happens to be out of order. Again, when twelve machines need to be assigned to three parallel tasks (four machines per task) for seven periods, the solution in Table 1 can be used for the best schedule.

Consider testing nk drugs for possible harmful interactions between pairs of drugs. In order to check all combinations, one needs to perform *nk(nk-1)/2* experiments by testing all possible pairs of drugs. However, this might take too long and be too expensive. Assume that *k* labs are available for such experiments, *n* different drugs can be administered simultaneously in each experiment, and the labs can be booked for d experiments each. A testing scheme needs to be designed such that in each of the *d* days the nk drugs are divided among the *k* labs, *n* drugs to each. It is necessary that each pair of drugs is tested at least once. Furthermore, it is desirable that every pair of drugs is in the same group about the same number of times.

Yet another application is the scheduling of commercials for broadcasting. Assume that *nk* commercials are available for broadcasting; only *n* slots are available in a particular program; and you wish to test these commercials and find the best combination of *n* commercials. The effectiveness of each individual commercial depends on the other commercials selected for showing. Some pairs of commercials work well together, while some pairs do not. Information about the interactions between various commercials is not available. The resources to check all possible selections are not available. Thus, *dk* groups of people are available for testing combinations of *n* commercials, and there are *k* rooms available for simultaneous showing. We would like to select d partitions of the *nk* commercials into *k* groups of *n* commercials each. Our objective is to have each pair of commercials represented in these groups about the same number of times so that the groups will evenly cover the spectrum of possible pairings.

The RPS problem is closely related to the well known Oberwolfach problem (Hell, Kotzig & Rosa, 1975; Alspach, 1996; Liu & Lick, 2003) named after the resort research center in Oberwolfach, Germany. Participants are invited to a week-long conference and dine at tables which accommodate the same number of persons. The seating is pre-arranged and the objective is to have every pair of participants seated at the same table about the same number of times. In the Oberwolfach problem pairs are considered matched when they sit next to each other and not all pairs at the same table are matched, i.e., the seating order around the table matters. A typical Oberwolfach problem is a conference of twenty-five participants who meet for six days (twelve meals thus d=*12*) and are to be seated at five tables, with five persons at a table. A solution does exist such that every pair of participants meets exactly twice (see Table 8 below).

Note that once the number of groups *(k)*, the number of members in each group *(n)* and the number of days *(d)* are known, the problem is fully defined and no additional information is required. Therefore, once a solution for a particular set of values for *n*, *k*, and *d* is found, it can be used for all the applications with these particular values.

In Drezner (1998), descent algorithms were proposed for the solution of this problem. In a descent-type algorithm some perturbations of the solution are checked and if an improved solution is discovered, the current solution is changed to the improved one. The algorithm proceeds until no such improvement is found. Optimization techniques for such problems received a significant boost with the discovery of metaheuristics. There are three main metaheuristics: tabu searches (Glover, 1986; Glover & Laguna, 1997), simulated annealing (Kirkpatrick, Gelat, & Vecchi, 1983), and genetic algorithms (Holland, 1975; Drezner & Drezner, 2005). These techniques generally require more computer time but produce much better results than “old fashioned" descent algorithms. The increased computational requirements are not prohibitive in this era of fast computers and results are usually obtained in a reasonable computer run time.

In this paper we show how to apply these metaheuristics to the RPS problem. These metaheuristics are widely used and produced excellent algorithms for numerous optimization problems. For a review see Salhi (1998), and Pardalos and Resende (2002). In this paper we compare these three metaheuristics and the descent algorithm for the solution of the RPS problem. Computational comparisons are reported for a set of RPS problems.

Problem Definition

Notation

n is the number of objects in each group,

k is the number of groups,

d is the number of days.

P is a partition of nk objects to k groups over d days,

$m_{rs}(P)$ is the number of times pair (r, s) meets in a partition P,

$F(P)$ is the objective function for partition P.$\mathrm{F(P)} = \sum_{r,s} m_{rs}^2(\mathrm{P})$

For example, for the golf problem n=4, k=3, d=7. The solution given in Table 1 consists of three pairs meeting three times, fifty-four pairs meeting twice, and nine pairs meeting once, which leads to the objective function F(P) = 3 x 3^2 + 54 x 2^2 + 9 x 1^2 = 252. In Drezner (1998) it was shown that this objective function is equivalent to minimizing the variance among pairs' meetings and thus leading to solutions with every pair meeting about the same number of times.

The following theorem (proven in Drezner, 1998) relates a problem (which we term the single problem) to another problem (which we term the multiple problem) with the same n and k but with the number of days being an integer multiple of the single problem.

Theorem 1: Consider two problems with the same n and k. The first (single) problem is based on d days while the second (multiple) problem is based on Ld days for an integer L. The optimal value of the objective function for the multiple problem is at least as good as that of the single problem multiplied by L^2.

The average number of meetings between pairs is $\frac{d(n-1)}{nk-1}$. If this number is integer, we term the problem a *perfect* problem because it may be possible to get a solution where all pairs meet exactly the same number of times, in which case it must be optimal. The following corollary is obvious.

Corollary: For perfect problems, if the single problem has a "perfect" solution where all pairs meet the same number of times, then in the optimal solution to the multiple problem all pairs meet the same number of times.

General Description of the Metaheuristics

Suppose we have a particular schedule as our current solution. We would like to find whether there is a better schedule. Checking all other possible schedules to find a better one is prohibitive because there are too many combinations. Therefore, we select a limited set of other schedules and find whether a better solution exists among them. In other words, we check only *perturbations* of the current schedule. A perturbation of the current schedule is defined as switching two objects between two groups on the same day. The set of all perturbed schedules is called the *neighborhood* of the current schedule. One may consider other neighborhoods as well. However, we recommend this definition of a neighborhood because it has been efficient in our experiments. We also define a *move* as changing the current schedule to another one in the neighborhood.

The descent algorithm and the metaheuristics can be described by the concepts of perturbations and neighborhoods. We first describe the methods in general, and then describe the technical details of each approach as it was applied to the RPS problem.

The Descent Algorithm

A starting schedule is randomly generated and is defined as the current schedule. Each iteration, the objective function values of all schedules in the neighborhood of the current schedule, is evaluated in a random order and the first encountered improved schedule is selected as the next current schedule. One may apply a variant of this approach by evaluating all the schedules in the neighborhood and selecting the best one. The algorithm terminates when no improved schedule exists in the neighborhood.

Tabu Search

Tabu search was first suggested by Glover (1986). In its simplest form, the tabu search starts as a descent algorithm. When no improving schedule is found in the neighborhood, the best non-improving schedule is selected for the next current solution with one restriction: a move which reverses a recent move is not allowed. This is essential to prevent cycling. Other rules that prevent cycling can be used to construct the tabu list of restricted moves. A restricted move stays in the tabu list for a pre-specified number of iterations called the tabu tenure. The process is continued until some stopping criterion is met.

In order to understand the motivation for tabu search, imagine a search on a plane with many craters. One of these craters is the deepest one, and that one is the desired solution (the global optimum). The descent algorithm performs only downward moves and may land at a shallow crater (a local optimum) and not at the global one. Tabu search attempts to get out of a shallow crater in the hope of getting to a better one. Therefore, when the descent algorithm terminates at the bottom of a crater, upward moves are taken in tabu search, while sliding back into the same crater is disallowed (by restricting reverse moves) with the hope of sliding into deeper craters and eventually reaching the global optimum. The tabu tenure controls the number of iterations a move remains restricted. The tabu tenure should be not too low (in which case the search may slide back into the same crater before it climbed out of it) or too high (in which case not enough moves are allowed and the search may miss improvement opportunities into another crater).

The Simulated Annealing Procedure

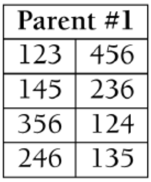
The simulated annealing procedure simulates the annealing of metals starting with a high temperature, gradually decreasing the temperature, and ending at cold temperatures. Its application to optimization problems was first suggested by Kirkpatrick et al. (1983). In each iteration a perturbation is randomly generated (and consequently one of the neighboring schedules randomly selected). If the selected neighbor is improved, it is selected as the next current solution. If the selected neighbor is non-improved, it is selected with a probability depending on the temperature and the extent of the deterioration in the value of the objective function. At the beginning, when the temperature is high, almost every move to a randomly selected neighbor is accepted regardless of its value of the objective function. As the process continues, the likelihood of selecting a non-improving neighbor decreases and towards the end of the process, when the temperature is low, a non-improving neighbor is hardly ever selected.

Borrowing the metaphor of the plane full of craters, simulated annealing is like a “bouncing” rubber ball which we hope will settle at the deepest crater because it is more difficult to get out of it. The cooling of the temperature means that the “height” of the bounce diminishes as the process continues. At the beginning, each bounce is very likely to bounce out of a crater and as the temperature is lowered, the likelihood of bouncing out of a crater diminishes (but it is still easier to bounce out of a shallow crater than a deeper one) and at the end of the process when the height of the bounce is low, the probability of bouncing out of a crater is very low.

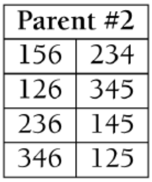
Genetic Algorithms

Genetic algorithms simulate the natural evolution of species. The application of genetic algorithms for optimization problems was first suggested by Holland (1975). The basic approach is as follows. We start with a population of schedules that can be randomly generated or obtained by other means. Each iteration, two parents are randomly selected from the population, and a crossover operator is used to produce an offspring. The offspring is created by taking part of the schedule from the first parent and combining it with the remaining part of the schedule from the second parent. The value of the objective function of the offspring is calculated and if it is better than that of the worst population member, it replaces that member. This way the size of the population remains constant. Some genetic algorithms use sporadic mutations. A mutation consists of randomly selecting a member of the population and considering a random perturbation of this member. We opted to apply the descent algorithm to the offspring and consider the improved offspring for inclusion in the population. This variant is called a hybrid-genetic algorithm or a memetic algorithm (Moscato, 2002). Since all mutations are considered by the descent algorithm, no mutations are suggested for our procedure. The evolution process is continued until a stopping criterion is met and the best member of the last population is selected as the solution. For a review of genetic algorithms see Drezner and Drezner (2005).

The most important part of genetic algorithms is the merging of two parents to produce an offspring. A good merging procedure is essential for a successful performance of a genetic algorithm. Most genetic algorithms employ a crossover operator to generate an offspring. To illustrate the process consider a problem with six people, two groups, and three people in each group, meeting for four days. The following are two arbitrary schedules (parents).



Parent #1	
123	456
145	236
356	124
246	135



Parent #2	
156	234
126	345
236	145
346	125

The crossover point is between days 2 and 3. We select, for example, the first two days from parent #1 and the last two days from parent #2 and get offspring #1. If we select the first two days from parent #2 and the last two days from parent #1 we get offspring #2.

Offspring #1	
123	456
145	236
236	145
346	125

Offspring #2	
156	234
126	345
356	124
246	135

Note that the order of the four days does not affect the value of the objective function. It is therefore possible to scramble the order of the days in each parent before determining the crossover point.

Two schedules are randomly selected and one offspring constructed. If the offspring has a good value of the objective function (better than the value of the objective function of the worst population member) it replaces it. Otherwise, the population remains unaltered. This completes one generation. The population evolves and keeps improving because "bad" schedules are removed from the population and are replaced by a "good" schedule. The process continues until a pre-specified number of generations have been performed.

Selecting Test Problems

The applications described in the introduction may have various values for the number of groups, the number of objects in each group, and the number of days. The golf scheduling problem has a specific set of parameters. However, the other applications may have various values for each particular instance. We therefore selected a set of problems to demonstrate the efficiency of the algorithms proposed in this paper. Such problems may well occur in practical applications.

One consideration in selecting the test problems was to select sets of single and multiple problems so that Theorem 1 and the Corollary can be verified. We first selected the two problems presented in Drezner (1998) (k=3, n=4, d=7; k=4, n=3, d=11) their multiples, and additional test problems, most of them being perfect problems. We restricted the problem size to d≤25. The set of 22 problems is depicted in all six tables below.

Applying the Metaheuristics for Our Problem

To illustrate the metaheuristic algorithms we consider a simple problem of only four people divided into two groups of two people each over a three-day period. There exists a simple solution where each pair meets exactly once:

1 2	3 4
1 3	2 4
2 3	1 4

A move is defined as exchanging two people between groups on the same day. For example, exchanging 1 and 3 on the first day (making the partition on the first day to be 3,2 and 1,4) and keeping the rest of the schedule the same is a move. The neighborhood is the set of all possible moves. For each day there are 4 possible exchanges for a total of twelve possible moves. Therefore, the cardinality of the neighborhood is twelve moves.

The Descent Algorithm

No specific parameters are required for the descent algorithm. We generate a random schedule and perform the first move that improves the value of the objective function. The descent algorithm proceeds until no move improves the value of the objective function. In the example above there are twelve possible moves. If at least one of them improves the value of the objective function it is executed and the twelve possible moves on the improved schedule are evaluated. If none of the twelve moves improves the value of the objective function, the algorithm terminates.

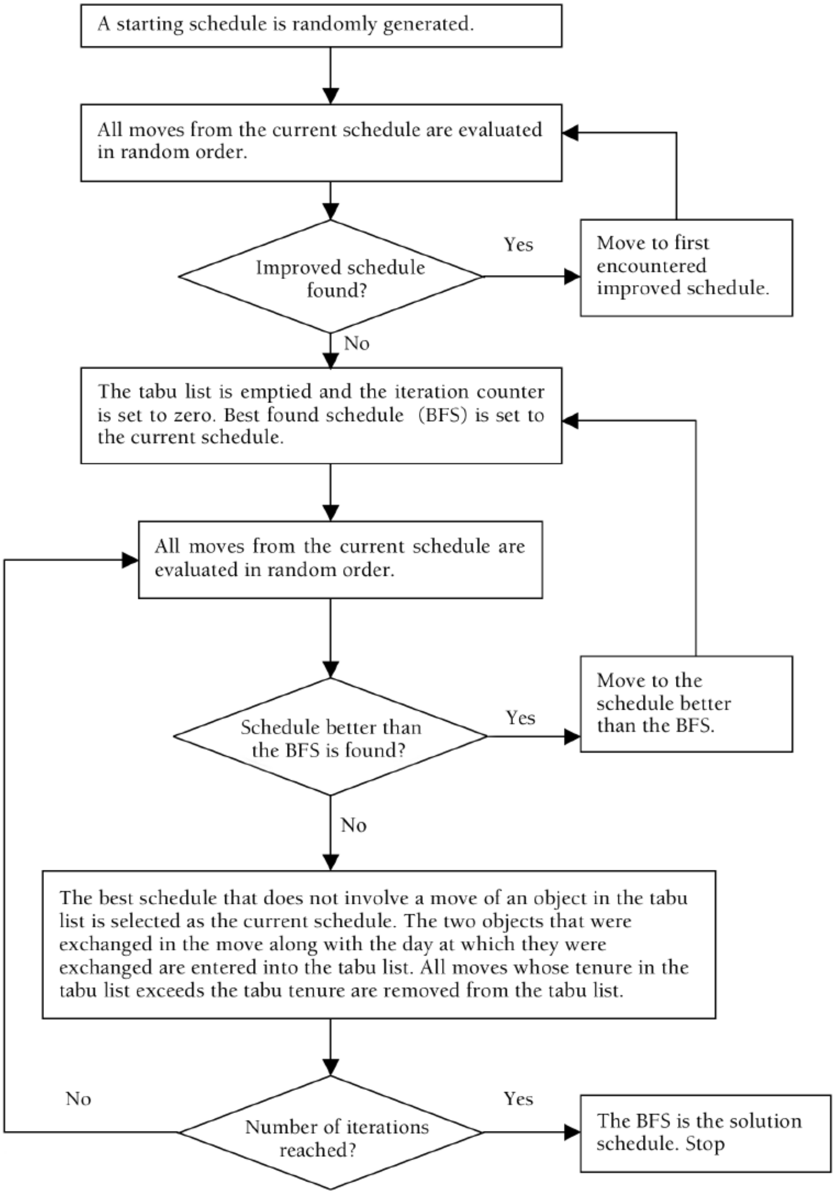
The Tabu Search

Some preliminary definitions:

- The tabu list consists of tabu moves. A tabu move is an object that was recently moved on a specific day. The tabu list consists of at most *nkd* entries for the *nk* different objects on *d* different days. For the example above, if persons 1 and 3 were exchanged on day 1, the entries person 1-day 1 and person 3-day 1 are entered into the tabu list. This means that in the next several iterations an exchange involving person 1 or person 3 on day 1 is not allowed. This is essential for an efficient search to prevent the repetition of the same exchange which would lead back to the same solution (sliding back into the same crater).
- The tabu tenure is the number of iterations that a tabu move remains on the tabu list. Once its tenure is over, the entry is removed from the tabu list. We selected the tabu tenure as 10 percent of the number of possible tabu moves: 0.1nkd. That means that 10 percent of the possible moves are disallowed because they involve persons that were recently moved on a particular day. For our particular example the tabu tenure is too small to be effective (0.1*nkd*=1.2). However, for the test problems this tabu tenure was effective.

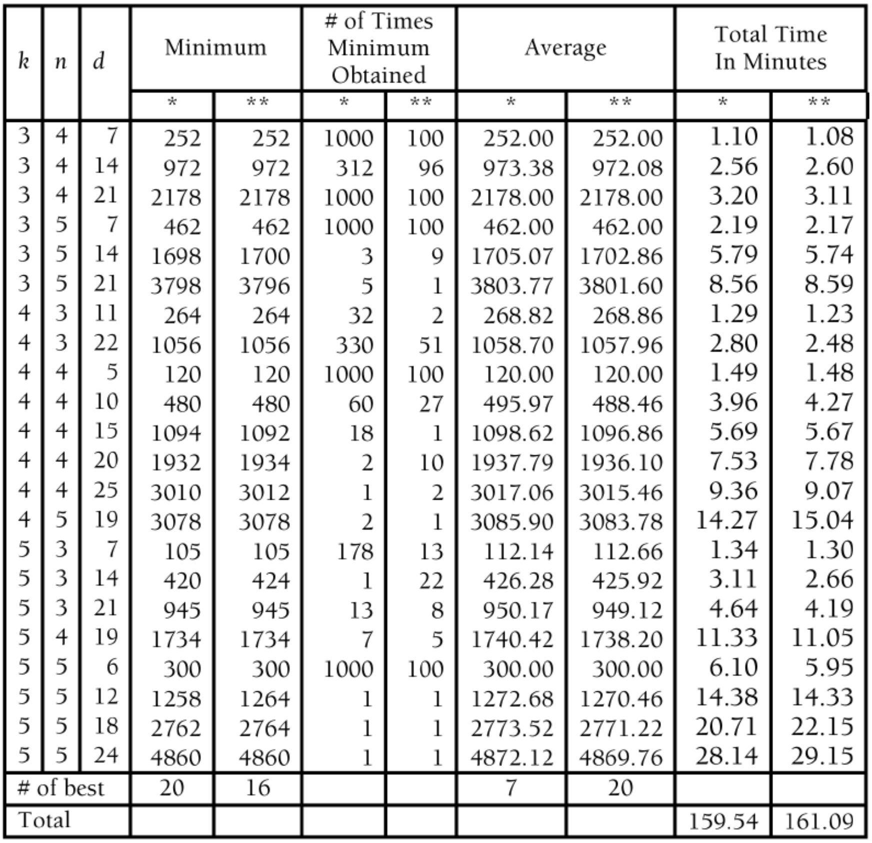
Extensive experiments lead to the following tabu search procedure. Note that the first phase of the tabu search is the descent algorithm.

The Tabu Search Procedure


A starting schedule is randomly generated.
All moves from the current schedule are evaluated in random order.
Improved schedule found?
Yes
Move to first encountered improved schedule.
No
The tabu list is emptied and the iteration counter is set to zero. Best found schedule (BFS) is set to the current schedule.
All moves from the current schedule are evaluated in random order.
Schedule better than the BFS is found?
Yes
Move to the schedule better than the BFS.
No
The best schedule that does not involve a move of an object in the tabu list is selected as the current schedule. The two objects that were exchanged in the move along with the day at which they were exchanged are entered into the tabu list. All moves whose tenure in the tabu list exceeds the tabu tenure are removed from the tabu list.
No
Number of iterations reached?
Yes
The BFS is the solution schedule. Stop

We report experiments with a stopping rule of 3,000 iterations with 1,000 runs, and 30,000 iterations with only 100 runs. The results are summarized in Table 3. The preferred approach is stopping after 3,000 iterations with 1,000 runs.

Table 3: *Comparison Between Two Values for the Number of Iterations in the Tabu Search*



k	*n*	*d*	Minimum		# of Times Minimum Obtained		Average		Total Time In Minutes	
			*	**	*	**	*	**	*	**
3	4	7	252	252	1000	100	252.00	252.00	1.10	1.08
3	4	14	972	972	312	96	973.38	972.08	2.56	2.60
3	4	21	2178	2178	1000	100	2178.00	2178.00	3.20	3.11
3	5	7	462	462	1000	100	462.00	462.00	2.19	2.17
3	5	14	1698	1700	3	9	1705.07	1702.86	5.79	5.74
3	5	21	3798	3796	5	1	3803.77	3801.60	8.56	8.59
4	3	11	264	264	32	2	268.82	268.86	1.29	1.23
4	3	22	1056	1056	330	51	1058.70	1057.96	2.80	2.48
4	4	5	120	120	1000	100	120.00	120.00	1.49	1.48
4	4	10	480	480	60	27	495.97	488.46	3.96	4.27
4	4	15	1094	1092	18	1	1098.62	1096.86	5.69	5.67
4	4	20	1932	1934	2	10	1937.79	1936.10	7.53	7.78
4	4	25	3010	3012	1	2	3017.06	3015.46	9.36	9.07
4	5	19	3078	3078	2	1	3085.90	3083.78	14.27	15.04
5	3	7	105	105	178	13	112.14	112.66	1.34	1.30
5	3	14	420	424	1	22	426.28	425.92	3.11	2.66
5	3	21	945	945	13	8	950.17	949.12	4.64	4.19
5	4	19	1734	1734	7	5	1740.42	1738.20	11.33	11.05
5	5	6	300	300	1000	100	300.00	300.00	6.10	5.95
5	5	12	1258	1264	1	1	1272.68	1270.46	14.38	14.33
5	5	18	2762	2764	1	1	2773.52	2771.22	20.71	22.15
5	5	24	4860	4860	1	1	4872.12	4869.76	28.14	29.15
# of best			20	16			7	20		
Total									159.54	161.09

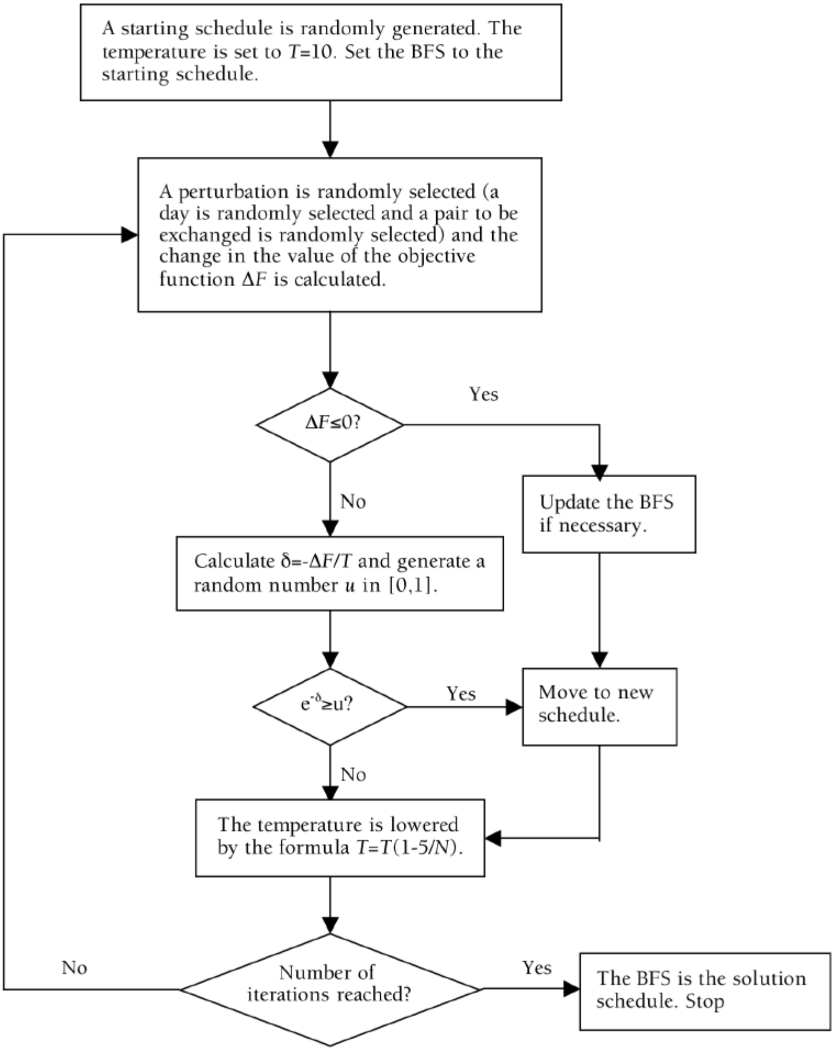
* 3,000 iterations, 1,000 runs.
** 30,000 iterations, 100 runs.

The Simulated Annealing

In our example above there are 12 possible moves. One of them is randomly selected. For example, suppose that day 2 is selected (each day is selected with probability of 1/3), and person 3 from group 1 is selected (each person in the group has a probability of 1/2 to be selected) and person 2 from group 2 is selected (also a probability of 1/2). Day 2 schedule is now 1,2 and 3,4 and days 1 and 3 keep the same schedule. Note that each move has a probability of 1/12 (the product of the above probabilities) to be selected.

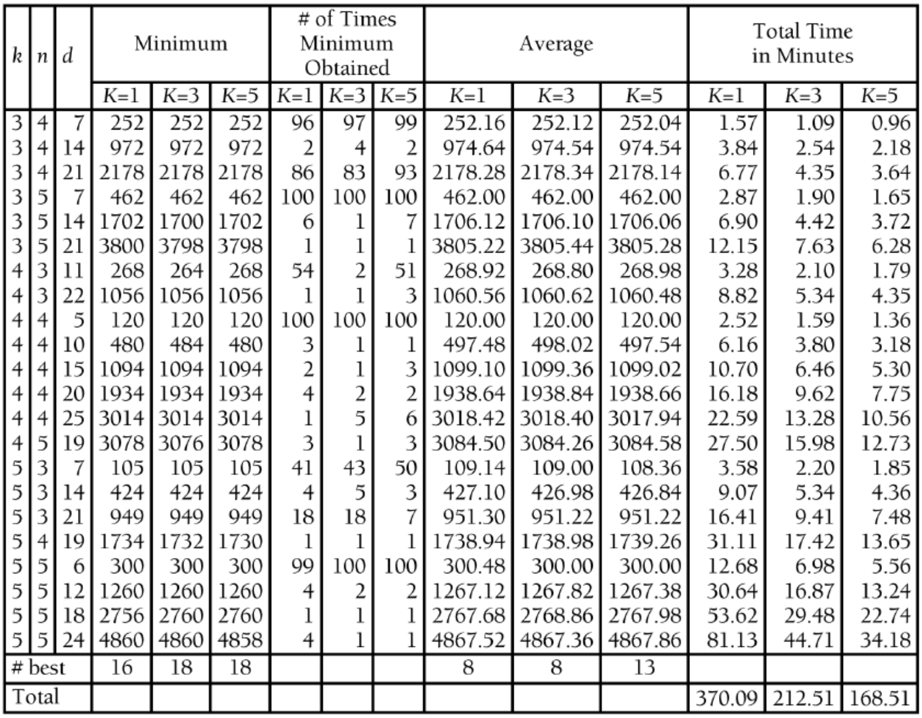
We experimented with many variants of the parameters for the simulated annealing procedure. The following parameters were found to be the most effective: An initial temperature of ten is selected. The number of iterations N was set to 2,000 times the number of possible perturbations or: $N = 1000n^2k(k = 1)d$. The procedure is:

The Simulated Annealing Procedure


A starting schedule is randomly generated. The temperature is set to T=10. Set the BFS to the starting schedule.
A perturbation is randomly selected (a day is randomly selected and a pair to be exchanged is randomly selected) and the change in the value of the objective function ΔF is calculated.
$\Delta F \leq 0$?
Yes
No
Update the BFS if necessary.
Calculate $\delta = -\Delta F/T$ and generate a random number u in [0,1].
$e^{-\delta} \geq u$?
Yes
Move to new schedule.
No
The temperature is lowered by the formula $T=T(1-5/N)$.
No
Number of iterations reached?
Yes
The BFS is the solution schedule. Stop

Following Drezner and Salhi's (2002) suggestion we also tried the SA-K simulated annealing procedure. In the SA-K procedure for a given integer $K \geq 1$, the best of K randomly generated neighbors is selected each iteration. In order to have the same number of objective function evaluations in the procedure, the number of iterations is divided by K. Note that SA-1 is the "standard" simulated annealing procedure. We report results for SA-1, SA-3, and SA-5. The comparison between different K's with one-hundred replications for each problem is given in Table 4. The best results overall for the set of twenty-two test problems was provided by K=5.

Table 4: *Comparison Between Three Versions of the Simulated Annealing*

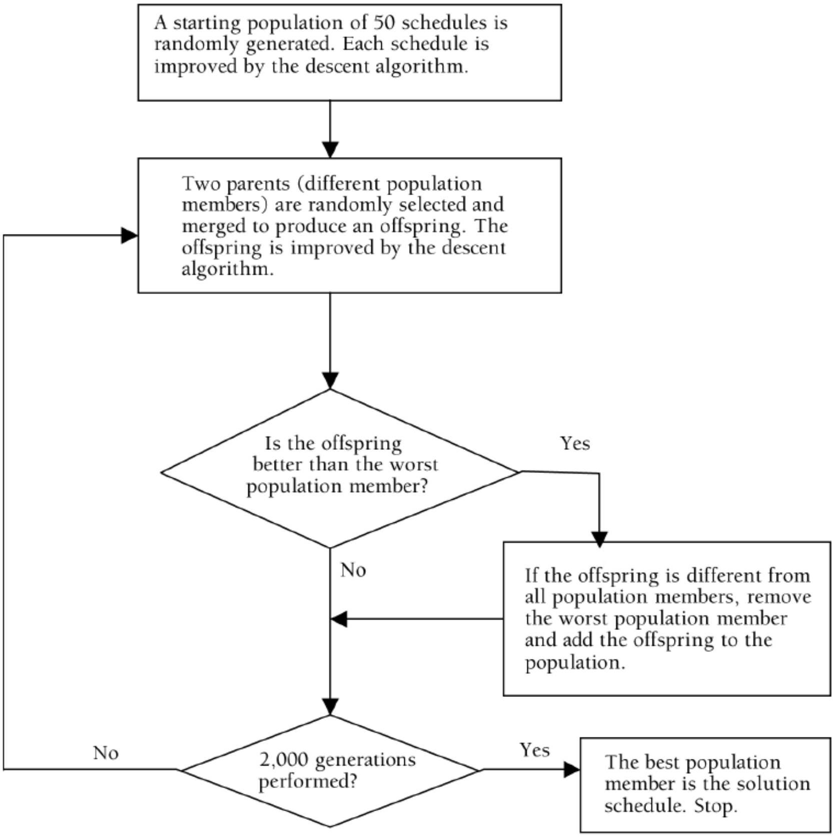


k	*n*	*d*	Minimum			# of Times Minimum Obtained			Average			Total Time in Minutes		
			K=1	*K*=3	*K*=5	*K*=1	*K*=3	*K*=5	*K*=1	*K*=3	*K*=5	*K*=1	*K*=3	*K*=5
3	4	7	252	252	252	96	97	99	252.16	252.12	252.04	1.57	1.09	0.96
3	4	14	972	972	972	2	4	2	974.64	974.54	974.54	3.84	2.54	2.18
3	4	21	2178	2178	2178	86	83	93	2178.28	2178.34	2178.14	6.77	4.35	3.64
3	5	7	462	462	462	100	100	100	462.00	462.00	462.00	2.87	1.90	1.65
3	5	14	1702	1700	1702	6	1	7	1706.12	1706.10	1706.06	6.90	4.42	3.72
3	5	21	3800	3798	3798	1	1	1	3805.22	3805.44	3805.28	12.15	7.63	6.28
4	3	11	268	264	268	54	2	51	268.92	268.80	268.98	3.28	2.10	1.79
4	3	22	1056	1056	1056	1	1	3	1060.56	1060.62	1060.48	8.82	5.34	4.35
4	4	5	120	120	120	100	100	100	120.00	120.00	120.00	2.52	1.59	1.36
4	4	10	480	484	480	3	1	1	497.48	498.02	497.54	6.16	3.80	3.18
4	4	15	1094	1094	1094	2	1	3	1099.10	1099.36	1099.02	10.70	6.46	5.30
4	4	20	1934	1934	1934	4	2	2	1938.64	1938.84	1938.66	16.18	9.62	7.75
4	4	25	3014	3014	3014	1	5	6	3018.42	3018.40	3017.94	22.59	13.28	10.56
4	5	19	3078	3076	3078	3	1	3	3084.50	3084.26	3084.58	27.50	15.98	12.73
5	3	7	105	105	105	41	43	50	109.14	109.00	108.36	3.58	2.20	1.85
5	3	14	424	424	424	4	5	3	427.10	426.98	426.84	9.07	5.34	4.36
5	3	21	949	949	949	18	18	7	951.30	951.22	951.22	16.41	9.41	7.48
5	4	19	1734	1732	1730	1	1	1	1738.94	1738.98	1739.26	31.11	17.42	13.65
5	5	6	300	300	300	99	100	100	300.48	300.00	300.00	12.68	6.98	5.56
5	5	12	1260	1260	1260	4	2	2	1267.12	1267.82	1267.38	30.64	16.87	13.24
5	5	18	2756	2760	2760	1	1	1	2767.68	2768.86	2767.98	53.62	29.48	22.74
5	5	24	4860	4860	4858	4	1	1	4867.52	4867.36	4867.86	81.13	44.71	34.18
# best			16	18	18				8	8	13			
Total												370.09	212.51	168.51

The Genetic Algorithm

Following experiments with various strategies, the following genetic algorithm is proposed:

The Genetic Algorithm


A starting population of 50 schedules is randomly generated. Each schedule is improved by the descent algorithm.
Two parents (different population members) are randomly selected and merged to produce an offspring. The offspring is improved by the descent algorithm.
Is the offspring better than the worst population member?
Yes
If the offspring is different from all population members, remove the worst population member and add the offspring to the population.
No
No
2,000 generations performed?
Yes
The best population member is the solution schedule. Stop.

Two variants of the merging procedure are reported. In the first one, termed the "fixed" procedure, the offspring is generated by selecting days 1, 3, 5,… from the first parent, and days 2, 4, 6,… from the second parent. This is the odd-even rule. In the "random" procedure, the order of the days in both parents is scrambled and the schedule is then constructed by the odd-even rule. In Table 5 we compare the two approaches on the twenty-two test problems with one-hundred runs each. The preferred approach is the random procedure.

Table 5: *Comparison Between Random and Fixed Merge of Parents in the Genetic Algorithm*

k	*n*	*d*	Minimum		# of Times Minimum Obtained		Average		Total Time in Minutes	
			Rand.	Fixed	Rand.	Fixed	Rand.	Fixed	Rand.	Fixed
3	4	7	252	252	100	100	252.00	252.00	0.44	0.44
3	4	14	972	972	12	14	973.76	973.78	1.27	1.21
3	4	21	2178	2178	100	100	2178.00	2178.00	2.35	2.20
3	5	7	462	462	100	100	462.00	462.00	1.09	1.13
3	5	14	1700	1702	1	4	1705.66	1706.28	3.56	3.50
3	5	21	3802	3802	6	7	3805.08	3805.32	6.62	6.43
4	3	11	268	264	82	2	268.36	268.36	0.60	0.60
4	3	22	1056	1060	1	83	1060.02	1060.34	1.72	1.68
4	4	5	120	120	100	100	120.00	120.00	0.71	0.70
4	4	10	480	480	2	2	497.94	497.92	2.04	1.92
4	4	15	1098	1096	13	2	1100.24	1100.54	3.87	3.63
4	4	20	1936	1936	1	2	1939.74	1940.16	6.08	5.64
4	4	25	3016	3016	1	3	3019.56	3019.70	8.55	7.87
4	5	19	3082	3082	1	1	3087.94	3088.62	16.08	15.71
5	3	7	105	105	17	13	111.14	111.66	0.60	0.63
5	3	14	426	426	12	9	428.10	428.46	1.79	1.80
5	3	21	949	951	1	15	952.52	952.94	3.33	3.31
5	4	19	1738	1740	1	1	1743.54	1744.32	12.20	11.41
5	5	6	300	300	100	100	300.00	300.00	5.66	5.68
5	5	12	1268	1268	1	1	1275.24	1275.76	16.89	16.57
5	5	18	2772	2772	5	6	2777.08	2777.26	32.81	31.96
5	5	24	4870	4866	1	1	4876.74	4877.34	51.68	50.00
# best			19	18			21	7		
Total									179.94	174.02

Computational Experiments

We report the results by all four heuristic procedures on the set of twenty-two test problems. Programs were coded in Microsoft Fortran PowerStation 4.0 and run on a Pentium IV 2.8GHz desktop PC with 256 MB RAM.

In Table 6 the minimum value of the objective function and the number of times found for each of the heuristics are depicted. The best known results are marked in boldface. In Table 7 we report the average value of the objective function for all runs and the run times in minutes for all runs. The best average solution among the four heuristics is marked in boldface. Solving the set of twenty-two test problems the specified number of replications took less than three hours of computer time for each of the heuristics.

Table 6: *Comparison of Best Solutions by Various Metaheuristics*

k	*n*	*d*	Decent		Tabu		S.A.		Genetic	
			100,000 Runs		1,000 Runs		100 Runs		100 Runs	
			Min	Times found	Min	Times found	Min	Times found	Min	Times found
3	4	7	**252**	3759	**252**	1000	**252**	99	**252**	100
3	4	14	**972**	7	**972**	312	**972**	2	**972**	12
3	4	21	**2178**	719	**2178**	1000	**2178**	93	**2178**	100
3	5	7	**462**	5630	**462**	1000	**462**	100	**462**	100
3	5	14	1702	1	**1698**	3	1702	7	1700	1
3	5	21	3802	3	3798	5	3798	1	3802	6
4	3	11	268	117	**264**	32	268	51	268	82
4	3	22	**1056**	3	**1056**	330	**1056**	3	**1056**	1
4	4	5	**120**	38618	**120**	1000	**120**	100	**120**	100
4	4	10	484	1	**480**	60	**480**	1	**480**	2
4	4	15	1098	2	1094	18	1094	3	1098	13
4	4	20	1936	1	**1932**	2	1934	2	1936	1
4	4	25	3016	2	**3010**	1	3014	6	3016	1
4	5	19	3084	3	3078	2	3078	3	3082	1
5	3	7	**105**	9	**105**	178	**105**	50	**105**	17
5	3	14	424	1	**420**	1	424	3	426	12
5	3	21	949	1	**945**	13	949	7	949	1
5	4	19	1740	2	1734	7	**1730**	1	1738	1
5	5	6	**300**	4248	**300**	1000	**300**	100	**300**	100
5	5	12	1268	1	**1258**	1	1260	2	1268	1
5	5	18	2772	4	2762	1	2760	1	2772	5
5	5	24	4872	2	4860	1	**4858**	1	4870	1
# best			8		16		11		9	

In Table 8 we report the best possible theoretical result and the best known solution for each problem (including those obtained by non-selected variants of the metaheuristics reported in Tables 3-5). Theorem 1 and the corollary may be used to confirm an optimal solution for multiple problems. However, for six multiple perfect problems the optimal solution was not obtained by the proposed heuristics. Theorem 1 and the corollary are indeed confirmed by these results.

The results in Tables 6 and 7 rank the heuristic algorithms as Tabu search being the best, simulated annealing the second best, then the genetic algorithm and the descent heuristic a clear last. Tabu search is the fastest among the three metaheuristics for the total number of trials run.

Discussion

This study looks at the problem of a group of people (objects) needing to be partitioned into smaller groups a given number of times. For example, twelve golf players play a tournament over seven days and each day they are partitioned into three groups of four players each. The repeated partition scheduling (RPS) problem requires

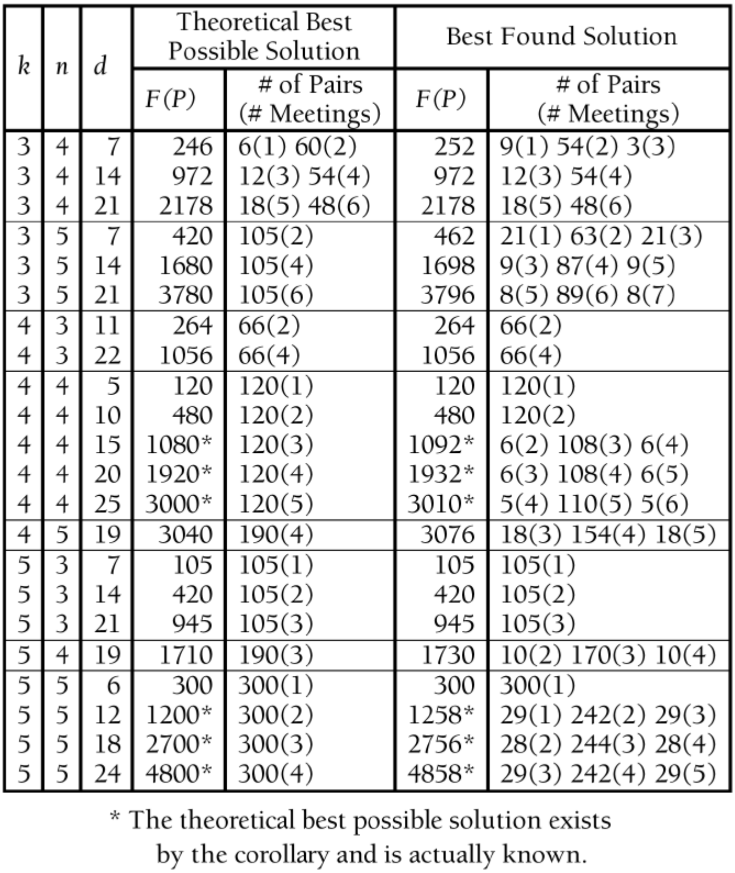
Table 7: *Comparison of Average Solutions by Various Metaheuristics*

k	*n*	*d*	Decent		Tabu		S.A.		Genetic	
			100,000 Runs		1,000 Runs		100 Runs		100 Runs	
			Ave	Time (min)	Ave	Time (min)	Ave	Time (min)	Ave	Time (min)
3	4	7	259.71	0.31	**252.00**	1.10	252.04	0.96	**252.00**	0.44
3	4	14	981.92	0.92	**973.38**	2.56	974.54	2.18	973.76	1.27
3	4	21	2184.26	1.74	**2178.00**	3.20	2178.14	3.64	**2178.00**	2.35
3	5	7	466.72	0.77	**462.00**	2.19	**462.00**	1.65	**462.00**	1.09
3	5	14	1719.37	2.63	**1705.07**	5.79	1706.06	3.72	1705.66	3.56
3	5	21	3818.48	5.04	**3803.77**	8.56	3805.28	6.28	3805.08	6.62
4	3	11	276.40	0.45	268.82	1.29	268.98	1.79	**268.36**	0.60
4	3	22	1067.76	1.35	**1058.70**	2.80	1060.48	4.35	1060.02	1.72
4	4	5	136.02	0.49	**120.00**	1.49	**120.00**	1.36	**120.00**	0.71
4	4	10	512.35	1.44	**495.97**	3.96	497.54	3.18	497.94	2.04
4	4	15	1112.93	2.83	**1098.62**	5.69	1099.02	5.30	1100.24	3.87
4	4	20	1952.62	4.52	**1937.79**	7.53	1938.66	7.75	1939.74	6.08
4	4	25	3032.13	6.45	**3017.06**	9.36	3017.94	10.56	3019.56	8.55
4	5	19	3105.97	12.39	3085.90	14.27	**3084.58**	12.73	3087.94	16.08
5	3	7	123.33	0.44	112.14	1.34	**108.36**	1.85	111.14	0.60
5	3	14	438.55	1.39	**426.28**	3.11	426.84	4.36	428.10	1.79
5	3	21	963.02	2.67	**950.17**	4.64	951.22	7.48	952.52	3.33
5	4	19	1759.45	9.13	1740.42	11.33	**1739.26**	13.65	1743.54	12.20
5	5	6	372.66	3.81	**300.00**	6.10	**300.00**	5.56	**300.00**	5.66
5	5	12	1296.41	12.53	1272.68	14.38	**1267.38**	13.24	1275.24	16.89
5	5	18	2798.95	25.20	2773.52	20.71	**2767.98**	22.74	2777.08	32.81
5	5	24	4899.07	40.47	4872.12	28.14	**4867.86**	34.18	4876.74	51.68
# best			0		15		9		6	
Total				136.97		159.54		168.51		179.94

that each pair of objects be in the same group about the same number of times. Applications include tournament scheduling where an equitable partition makes the game more enjoyable and fair, personnel assignment where an equitable solution enhances good relations between group members, production scheduling so that unexpected break down of machines will have the least effect on the operation, advertising and experimental design where it may be too costly to measure interactions between pairs of objects and we can group the objects into groups of more than two objects each reducing the cost of the experiment, and others. For example, in personnel assignment we wish that pairs of workers will not work together in the same group for too many days so that they will not “get on each other’s nerves”. In small conferences, when participants are assigned to tables during meals, we wish that each participant will sit at the same table with other participants about the same number of times in order to facilitate meetings and conversations between participants.

The problem is a combinatorial design problem with a very large number of possible partitions. Our goal is to find the partition that is the most equitable. That means that every pair of objects is in the same group about the same number of times. It is noted that once the number of objects, the number of groups, and the number of days are given, then the problem no longer depends on the application. For example, if twelve objects need to be partitioned into three groups over a seven day period, then the solution depicted in Table 1 can be used for any application with these three values.

Table 8: *Summary of Results for the Test Problems*



k	n	d	Theoretical Best Possible Solution		Best Found Solution	
			$F(P)$	# of Pairs (# Meetings)	$F(P)$	# of Pairs (# Meetings)
3	4	7	246	6(1) 60(2)	252	9(1) 54(2) 3(3)
3	4	14	972	12(3) 54(4)	972	12(3) 54(4)
3	4	21	2178	18(5) 48(6)	2178	18(5) 48(6)
3	5	7	420	105(2)	462	21(1) 63(2) 21(3)
3	5	14	1680	105(4)	1698	9(3) 87(4) 9(5)
3	5	21	3780	105(6)	3796	8(5) 89(6) 8(7)
4	3	11	264	66(2)	264	66(2)
4	3	22	1056	66(4)	1056	66(4)
4	4	5	120	120(1)	120	120(1)
4	4	10	480	120(2)	480	120(2)
4	4	15	1080*	120(3)	1092*	6(2) 108(3) 6(4)
4	4	20	1920*	120(4)	1932*	6(3) 108(4) 6(5)
4	4	25	3000*	120(5)	3010*	5(4) 110(5) 5(6)
4	5	19	3040	190(4)	3076	18(3) 154(4) 18(5)
5	3	7	105	105(1)	105	105(1)
5	3	14	420	105(2)	420	105(2)
5	3	21	945	105(3)	945	105(3)
5	4	19	1710	190(3)	1730	10(2) 170(3) 10(4)
5	5	6	300	300(1)	300	300(1)
5	5	12	1200*	300(2)	1258*	29(1) 242(2) 29(3)
5	5	18	2700*	300(3)	2756*	28(2) 244(3) 28(4)
5	5	24	4800*	300(4)	4858*	29(3) 242(4) 29(5)

* The theoretical best possible solution exists
by the corollary and is actually known.

We solved the RPS problem using the three metaheuristics: tabu search, simulated annealing and the genetic algorithm. The recommended method is the tabu search as it provided the best solutions in the fastest computer time on our set of twenty-two test problems.

References

Alspach, B. (1996). The Oberwolfach Problem, in *CRC Handbook of Combinatorial Designs*, C.J. Colbourn and J.H. Dinitz (Eds.), CRC Press, 394-395.

Drezner, Z. (1998). On the Repeated Partition Scheduling Problem, *Journal of Business and Management*, 5: 65-77.

Drezner, T. & Drezner, Z. (2005). Genetic Algorithms: Mimicking Evolution and Natural Selection in Optimization Models, in Y. Bar-Cohen (Ed.), *Biomimetics - Biologically Inspired Technologies*, CRC Press, Boca Raton, FL, 157-175.

Drezner, Z. & Salhi, S. (2002). Using Hybrid Metaheuristics for the One-Way and Two-way Network Design Problem, *Naval Research Logistics*, 49: 449-463.

Glover, F. (1986). Future Paths for Integer Programming and Link to Artificial Intelligence, *Computers and Operations Research*, 13: 533-549.

Glover, F. & Laguna, M. (1997). *Tabu Search*, Kluwer Academic Publishers, Boston.

Hell P., Kotzig A. & Rosa, A. (1975). Some Results on the Oberwolfach Problem, *Aeq. Mathematica,* 12: 1-5.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan.

Kirkpatrick, S., Gelat, C.D. & Vecchi, M.P. (1983). Optimization by Simulated Annealing, *Science,* 220: 671-680.

Liu, J. & Lick, D.R. (2003). On λ-Fold Equipartite Oberwolfach Problem with Uniform Table Sizes, *Annals of Combinatorics*, 7: 315-323.

Moscato, P. (2002). Memetic Algorithms, in P.M. Pardalos and M.G.C. Resende (Eds.) *Handbook of Applied Optimization*, Oxford, U.K: Oxford University Press.

Pardalos, P.M. & Resende, M.G.C. (Eds.) (2002). *Handbook of Applied Optimization*, Oxford, U.K: Oxford University Press.

Salhi, S. (1998). Heuristic Search Methods, in *Modern Methods for Business Research*, G.A. Marcoulides (Ed.), Lawrence Erlbaum Associates, Mahwa, NJ.