

A tabu search based heuristic for the 0/1 Multiconstrained Knapsack Problem

Johan Oppen
Molde College, 6411 Molde, Norway
Johan.Oppen@hiMolde.no

Solveig Irene Gruner
Molde College, 6411 Molde, Norway
Solveig.I.Gruner@hiMolde.no

Arne Løkketangen
Molde College, 6411 Molde, Norway
Arne.Lokketangen@hiMolde.no

Abstract

We describe a reasonably simple method for solving the 0-1 Multiconstrained Knapsack Problem (0/1 MKP), and report quite good results. We base our method on tabu search, where moves that contain recently used elements get a penalty. In addition we concentrate on strategic oscillation about the feasibility boundary.

1 Introduction

The 0-1 Multiconstrained Knapsack Problem (0/1 MKP) is a Discrete Optimization Problem (DOP) which has a very simple structure and is easy to understand. Nevertheless, some types of instances can be very hard to solve to proven optimum.

A lot of work has been done to develop good heuristics for this problem, using various techniques. These include simulated annealing, tabu search and genetic algorithms. All these approaches have given good results. We have chosen to focus on tabu search, and base our implementation on the work of Glover and Kochenberger [2]. We try to simplify their methods in order to find out if it is possible to get good results while keeping the algorithms simple. The main ideas in [2] is a tabu search with strategic oscillation about the feasibility boundary and the use of critical events.

2 Problem description

The 0-1 Multiconstrained Knapsack Problem (0/1 MKP) can be formulated

$$\begin{aligned} \text{Max} \quad & \sum_{j=1}^n c_j x_j \\ \text{s.t.} \quad & \\ & \sum_{j=1}^n a_{ij} x_j \leq b_i, \forall i \in \{1, 2, \dots, m\} \\ & x \in \{0, 1\} \end{aligned}$$

n is the number of items, and m is the number of knapsack constraints. The c values can be interpreted as the value of including the different items, the a values can be interpreted as measures of weight or volume for each item, and the b values are limits for each constraint.

The x values represent the items, which are given the value 1 if they are included in the solution, 0 otherwise. The problem is then to find the best combination of items to include in the solution, not violating any of the constraints. There are 2^n different solutions to the problem, some of these are infeasible as they violate one or more of the constraints. The problem is known to be NP-hard and a lot of work has been done to find good algorithms to solve it.

3 Tabu search

Tabu Search is a metaheuristic based on local search, which in addition allows non-improving moves. Tabu Search makes use of a memory structure, it “remembers” attributes of earlier visited solutions in order to avoid parts of the solution space. Solutions or some of their attributes are considered “tabu” or illegal for some time to keep the search from going in loops, perhaps returning to the same local optimum. In addition, frequency based long term memory is used to lead the search into new regions of the solution space (diversification). It is also common to ignore the tabu criteria if a new best solution can be reached (aspiration).

For a more thorough presentation of Tabu Search, see ch 3 in Reeves (1993) [1].

3.1 Search Implementation

Our implementation of the search is based on the methods of Glover and Kochenberger [2]. We have implemented their main idea, a strategic oscillation process that navigates both sides of the feasibility boundary.

- We start with an empty solution, which ensures that we have a feasible *starting solution*.
- A *move* is the flip of a variable, which means assigning the opposite value to a variable. This is analogous to taking items into the knapsack (0 becomes 1) and removing items from the sack (1 becomes 0).
- The *neighborhood* of a solution is the set of all possible moves that can be made, the *neighborhood size* is $|n|$, the number of variables. It is also trivial to show that the optimal solution is never more than n flips away from the current solution.
- *Move evaluation* is based on surrogate constraints and tabu penalties, and the best move is always chosen. This means that we always search the whole neighborhood before we decide what move to make.

- *Span* is a parameter that indicates the amplitude of the oscillation about the feasibility boundary, measured in number of variables flipped to 1 when proceeding from the boundary into the infeasible region and in the numbers of variables flipped to 0 when proceeding from the boundary into the feasible region. We have chosen span to be a random integer between 1 and 6.
- The implementation is split into two phases, constructive and destructive. In the constructive phase the variables are set to 1 until we reach span, and in the destructive phase the variables are set to 0 until we reach span. We define an *iteration* to be a constructive phase followed by a destructive phase. The iterations will thus contain a different numbers of moves each, depending both on the number of moves needed to reach the boundary between the feasible and infeasible region, and the value of the parameter span.
- When we reach the boundary and just before we step into the infeasible region, we search for a new best solution. The same procedure is repeated when we move from infeasibility to feasibility. These are the *critical events* of the search.
- The main idea of tabu search is to use information about earlier solutions visited. We use both recency and frequency information. A variable that has been flipped recently gets a penalty to avoid it from getting flipped again for a period. *Tabu tenure* is the number of moves the variable keeps a penalty. This means that it is not forbidden to flip the variable, it is just more costly if it has been flipped recently. The frequency information is used to lead the search into new areas of the solution space where we have not been before (diversification).
- The tabu list contains information about which variables have been part of the last feasible solutions found at critical events, the tabu tenure is the length of this list. Variables that have been part of these solutions get a penalty, depending on the number of solutions they have been part of, to make it harder for them to be included in the solution and easier to be dropped from the solution. When a new feasible solution is found at a critical event, this is added to the tabu list while the last solution in the list is removed.

3.2 Test cases

To test our implementation, we have used the same set of 57 test cases as Glover and Kochenberger[2]. To our knowledge, these are currently not available at “OR-library”. They have been made available to us by our lecturer, professor Arne Løkketangen at Molde College.

We have used 4 of these 57 test cases for preliminary testing to find values for the parameters span and tabu tenure. These are FLEI, PET6, PB4 and WEISH30, and we have chosen these because they differ quite much in number of variables and constraints.

3.3 Starting Solution

We have tried to start the search with a random generated solution, which leads to a starting solution that can be either feasible or infeasible. We combined this approach with the use of restarts during the search to lead the search into new regions of the solution space. This was then tried on a few instances, but did not perform as well as if we leave the diversification process to the tabu search itself by using frequency information.

3.4 Move Evaluation

The move evaluation consists of two parts, surrogate constraints and tabu penalty. These are used in combination in order to find the best variable to flip.

3.4.1 Surrogate constraints

First we compute:

$$b'_i = b_i - \sum_{j=1}^n (a_{ij} : \text{for } j \text{ with } x_j = 1)$$

This is done for all the rows in the constraint set, and can be seen as the slack for the different constraints. Violated constraints lead to a negative b'_i , constraints where the left hand side and the right hand side are equal give a $b'_i = 0$, and constraints with a positive b'_i are not fully utilized. Based on the values of b'_i we compute the weights:

$$\begin{aligned} \text{If } b'_i > 0, \text{ set } w_i &= 1/b'_i \\ \text{If } b'_i \leq 0, \text{ set } w_i &= 2 + |b'_i| \end{aligned}$$

We can then put together the surrogate constraints:

$$\sum_{j=1}^n s_j x_j \leq s_0$$

where

$$s_j = \sum_{i=1}^m w_i a_{ij}$$

and

$$s_0 = \sum_{i=1}^m w_i b_i$$

Here, s_j is the sum of the weights of all the constraint rows multiplied with the constraint coefficients for the column j . This is the only part of the surrogate constraint that is used to choose which variable to flip.

To choose which variable to flip to 1 in the constructive phase, we find the variable that satisfy

$$\max(c_j / s_j : x_j = 0)$$

To choose which variable to flip to 0 in the destructive phase, we find the variable that satisfy

$$\min(c_j / s_j : x_j = 1)$$

3.4.2 Tabu penalty

The tabu penalty for each variable consists of two parts, a penalty term for recency and another for frequency. The penalty term for recency is computed as follows: first we look in the tabu list to find the number of the last feasible critical event solutions the variable has been part of. This is then multiplied with the constant penR, which is the largest column sum from the normalized constraint matrix. The penalty term for frequency is computed like this:

$$\text{penF} = \text{penR}/(C*\text{curIter});$$

Here, curIter is the iteration counter, C is the constant 10 000. Glover and Kochenberger[2] recommend this value for the constant C, and we follow their advise. penF is then multiplied with a counter that shows how many times the variable in question has been part of a critical event solution.

We have also considered changing the value for penR, perhaps in connection with the value of tabu tenure. We discuss this further in 3.7.3.

3.5 Choosing values for the parameter span

Glover and Kochenberger [2] change the value of the span parameter by increasing and decreasing its value systematically within an interval. They recommend [1,7] to be this interval, and the different values are used for a different number of iterations each.

We have implemented this idea, and it certainly worked well. To keep our search method simple, we decided to simplify the choice of values for the span parameter. We have run the 4 cases for preliminary testing with fixed values for span in the range from 1 to 7. We also tried assigning random values to span for each iteration, which turned out to be a much better choice than using a fixed value. We found that random numbers in the interval [1,6] seemed to be the best.

3.6 Critical Events

In the constructive phase, we sooner or later reach the point where choosing the best move results in a new solution that is infeasible. When this is about to happen, we first search for another variable to flip that results in a new best solution that is feasible. This means that we compute the objective value and check the original constraints for each possible move from the current solution. This is done before the move chosen by the move evaluation procedure is taken. If a new best solution is found, we keep the value of this new best solution.

We then take the move chosen by the move evaluation, which brings the search into the infeasible region. Before proceeding with the next move we search for a variable to flip from 1 to 0 in order to get a new best solution that is feasible. In the destructive phase a similar but opposite procedure is performed at the critical event.

The way these critical events are handled means that we make use of no aspiration criteria by choosing a move if it leads to a new best solution.

3.7 Choosing values for tabu tenure

Glover and Kochenberger [2] use two parameters connected with the use of recency information, one is the length of the tabu list, the other is the number of moves in the beginning of each phase (constructive/destructive) where the tabu list is taken into consideration. This means that a lot of moves are made where no recency or frequency information is used.

Following our main idea of keeping the search simple, we use only the tabu tenure parameter. This means that every move is evaluated by taking the tabu information into consideration. We have tried some different ways of choosing values for tabu tenure:

1. Tabu tenure is a constant, this constant is used for all instances.
2. Tabu tenure is a function of problem size. This means either a function of the number of variables, the number of constraint rows, or both.
3. Tabu tenure is chosen randomly for each iteration, or a number of iterations. We then need an interval for tabu tenure.

3.7.1 Constant value for tabu tenure

We have tested constant tabu tenure values from 1 to 100 on our 4 test cases. 0 has not been tried as a value for tabu tenure, for two reasons. First, this would mean that we leave the concept of tabu search, making no use of recency information. It would also require rewriting some of the code for our implementation, and we have chosen not to do so for the time being. We find optimum in all our four test cases, but with different values for tabu tenure. For three of the test cases the “good” values for tabu tenure that gives optimum is spread out over almost the whole interval. For FLEI we find optimum with all values except 1, for PET6 we find optimum with the values 2, 3, 5, 8, 9 and 50, and for PB4 we find optimum with the values 2, 4, 6, 8, 10, 20, 30, 40, 50, 60, 70 and 80. For WEISH30 we find optimum only with the values 2 and 3, in this case 1 is the value with the best average performance. A simple statistical test shows that it is impossible to find one single value for tabu tenure that is better than others. For our four test cases, it does not seem necessary to use values larger than 10 for tabu tenure. The average value of the objective by the value of tabu tenure is shown in Figure 3.1.

Our conclusion is that if we find no other way of choosing good values for tabu tenure, we should use all the values in the interval from 1 to 10. This means that we hope that our four test cases are representative of all the 57 in the way that no instances need a higher value of tabu tenure than 10 to find optimum, if optimum can be found by our method. In this way we are sure that all the values are used the same number of times, which would not necessarily been the situation if we used randomness to find values for tabu tenure. As we could not find any function of problem size that performed well as tabu tenure, we will use the values in the interval [1,10].

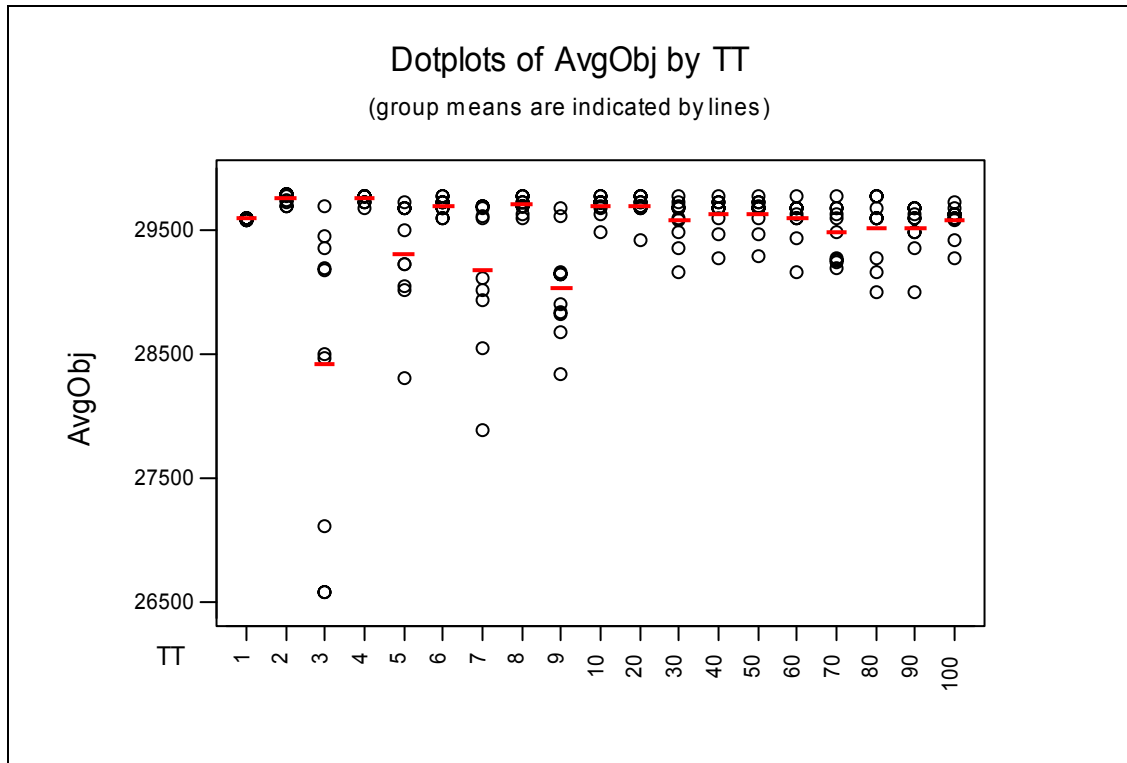


Figure 3.1

3.8 Number of iterations

To find a suitable number of iterations for the search, we also used the PET6 case as our only test case. We ran this instance 10 times with each of the 10 chosen values for tabu tenure, for a number of 100 000 iterations each time. Running more than 100 000 iterations seems to be out of the question, as this would require an unacceptable amount of time.

Optimum were found in 31 of the 100 runs. In 20 of these, optimum were found after more than 20 000 iterations. 8 times optimum were found after more than 50 000 iterations, and 2 times optimum were found after more than 90 000 iterations. The average number of iterations needed to reach the best value found in each run was 29 861. This shows that we often find the best solution after a large number of iterations, and we therefore think it can be worthwhile to use 100 000 iterations.

3.9 Differences to the approach of Glover and Kochenberger

These are the main differences between what we have implemented and the approach of Glover and Kochenberger [2] :

- We do not oscillate in a systematic and deterministic way about the feasibility boundary, but draw a random number between 1 and 6 to determine how far into the feasible or infeasible region to move in each iteration.

- We do not change the value of the tabu tenure during the search like Glover and Kochenberger do, but keep this value fixed throughout the search. Instead, we run the search multiple times, each time with a different value of tabu tenure.
- We do not differ between the “easy” and the “hard” cases, but give them all the same number of iterations. Glover and Kochenberger use less iterations to solve the “easy” ones, and then put more effort into solving the “hard” ones by modifying parameters and increasing the number of iterations.

4 Computational results

In the appendix of this paper, we present a table containing a summary of our computational results. In the following section we go more thoroughly into these results.

4.1 About the testing

As we make use of random numbers for the parameter span, we have run each test case with 10 different random seeds. This is done to get some amount of statistical significance, but it is not trivial to find how many different random seeds are needed to get reliable results. Using only 6 different values for the parameter span, we think 10 different seeds should be enough. The results also show that if we find optimum with a certain value for tabu tenure, we often find optimum with all the 10 different seeds. The decision of using tabu tenure values in the interval $[1,10]$ leads to a total number of runs for each test case of 100. In each run we do 100 000 iterations.

4.2 Results

We found optimum in 55 of the 57 test cases, for 6 test cases we found optimum in the first iteration of every run.

Small values (2-3) for tabu tenure performed well for most of the test cases, and it looks like we could have reduced our interval for tabu tenure, and still got good results for most of the test cases. However, for one instance 1 turned out to be the only value for tabu tenure that lead to optimum, and for a few others 1 performed best. This indicates that 1 should be used. The upper bound of the interval could have been set to 3, as this is the highest value that seems to be needed to find optimum in any of the instances where the optimal value was found.

If we divide the test cases into subgroups, 30 of them (the WEISH cases) constitute a family of instances that show a particular behaviour when we try to solve them with our method. For almost all of them we find the best value very early in the search, often in the first iteration. Figure 4.1 shows a typical behaviour for this kind of instance.

For 4 of the WEISH instances this turns out to be the optimal solution, for the rest our method finds it quite hard, if not impossible, to find optimum. Both the cases where our method failed to find optimum is in this family, and it is also here we find the cases where 1 performed well as tabu tenure. One question is then if these cases are constructed to be difficult to solve, or if our approach is especially poor on these cases.

Anyway, it is a fact that our move evaluation function leads to a high quality solution in almost “no time”.

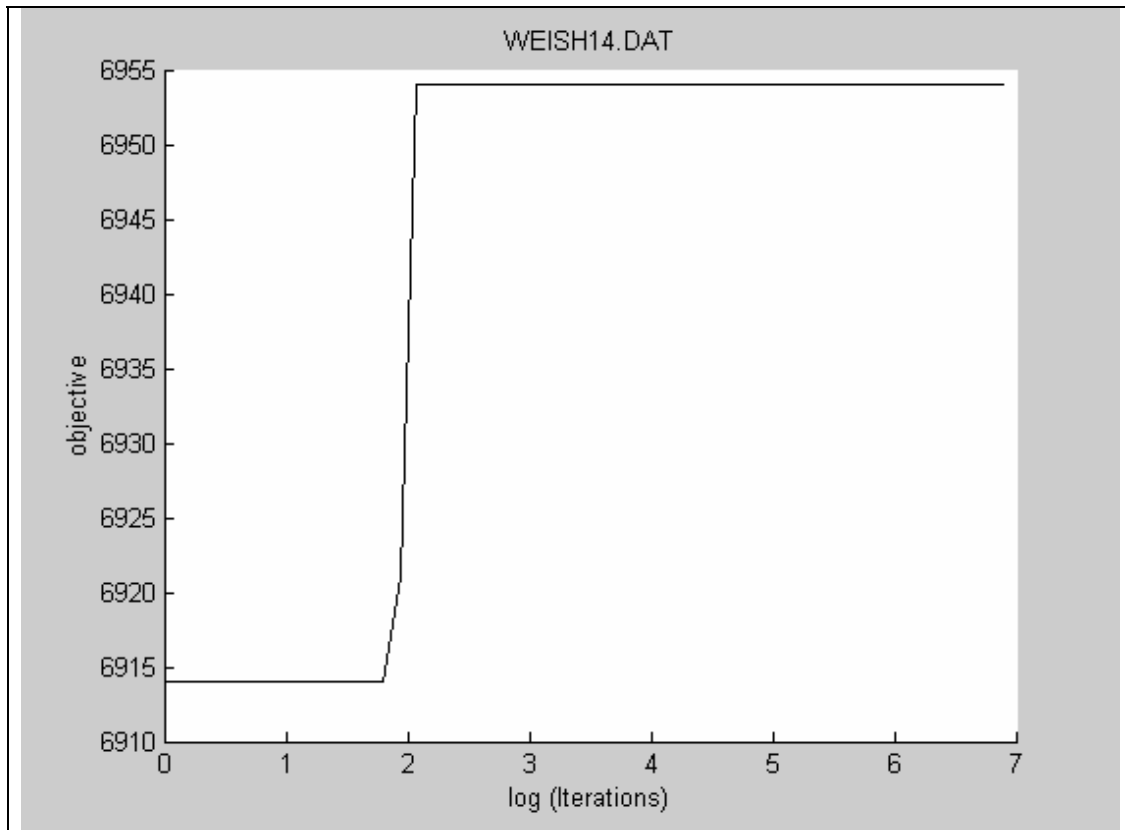


Figure 4.1

The average number of iterations needed to find the best value in each run varies from 1 to 47 657 for the 57 instances, with a mean of 14 694. The instances that need only a few iterations to the best solution found (the WEISH cases) have a major influence on this number. For most of the others, between 20 000 and 50 000 iterations are typically needed to find the best value in each run. 100 000 iterations in each run then seems quite suitable in our opinion.

In average, we find an objective value of 99.44% of the optimal. Statistically this means that if we pick a random instance, a random value for tabu tenure on the interval [1,10] and one of the random seeds, the expected value for the objective will be 99.44% of optimum.

The number of times we find optimum varies a lot over the 57 test cases. Some of the test cases are “easy”, and in addition they are insensitive to variations in tabu tenure. Other test cases are either “hard” or they can be solved to optimum with only one or a few values for tabu tenure.

4.3 Comparisons to other methods

To get a better understanding of how our implementation performs, we have compared our results to the results obtained by Glover & Kochenberger [2] . We have also used cplex 6.6.0 to solve the problems.

Glover and Kochenberger [2] find the optimal solution in 42 of the 57 test cases in less than 1900 moves. For the other 15, they reached optimum within 100 000 moves. To find optimum in these 15 cases, they also modified parameter values on the way. We have used about 700 000 moves in each run, this number is partly governed by the random value of the parameter span.

Cplex 6.6.0 solved all the 57 test cases to proven optimum in 2 seconds. We have used a total time of 41 hours to run all the test cases 100 times each, using different values for tabu tenure and different seeds for generating random numbers. This shows that, at least for relatively small sized knapsack problems, cplex is indeed very efficient.

5 Conclusions

Our main conclusion is that it is possible to get quite good results even if we keep our search method relatively simple. We have to some extent used brute force instead of complexity to get results. This is done by using a lot of different values for tabu tenure, and by doing many iterations in each run.

We have learned that there are a lot of decisions to make in order to make good methods. Even if we reuse what is done by others, there are a lot of parameter values to be set. If we should do proper testing to find the best values for every parameter, we would need a lifetime.

The work we describe in this paper has been carried out during one semester, as part of a course in optimization methods. There are a lot of areas we would like to investigate further if we had the opportunity, which we hope to get in the future.

- We would like to look more closely into the relationship between problem size or other characteristics, and the best value for tabu tenure.
- We would also like to try using aspiration criteria by taking moves that leads to a new best solution, not just saving its value like we do now.
- Another approach is to use a full sack as starting solution, beginning with the destructive phase in each iteration.
- It would be interesting to try our implementation on larger problems to find out how our method performs, especially compared to exact methods like cplex.

References

[1] Reeves, Colin R (1993) **Modern Heuristic Techniques for Combinatorial Problems**, Blackwell Scientific Publications.

[2] Glover, Fred and Kochenberger, Gary A (1996) "Critical Event Tabu Search for Multidimensional Knapsack Problems", In I.H. Osman and J.P. Kelly, editors, *Meta Heuristics: Theory and Applications*, Kluwer Academic Publishers, pp 407-427.

[3] Hvattum, Lars Magnus, Løkketangen, Arne and Glover, Fred (2002) "Adaptive Memory Search for Boolean Optimization Problems". Submitted to the special issue of *Discrete Applied Mathematics on boolean and pseudo-boolean functions*.

[4] Glover, Fred and Laguna, Manuel (1997) **Tabu Search**, Kluwer Academic Publishers.

Appendix – Computational result table

This is an explanation of the column headers in the table.

n	Number of variables
m	Number of constraint rows
opt	Optimal value of the objective
no	Number of runs where we found optimum. Bold number in this column means best value found when we did not find optimum.
av	Average objective value found
%	Average as % of optimal value
tt	Best value(s) for tabu tenure
it	Average number of iterations to best value found in each run
*	An asterisk in this column means that optimum was found

Name	n	m	opt	no	av	%	tt	it	*
FLEI	20	10	2139	90	2134.6	99.79	2 - 10	6082	*
HP1	28	4	3418	71	3411.9	99.82	2,3	26084	*
HP2	35	4	3186	13	3169.8	99.49	2,4,6	38405	*
PB1	27	4	3090	65	3082.9	99.77	2,3	23060	*
PB2	34	4	3186	30	3174.1	99.62	5	38326	*
PB3	19	2	28642	89	28514	99.55	2 - 10	6	*
PB4	29	2	95168	45	93812	98.58	2,4,8	16837	*
PB5	20	10	2139	90	2134.7	99.80	2 - 10	7726	*
PB6	40	30	776	71	767.4	98.89	2,4,9,10	17427	*
PB7	37	30	1035	19	1030.3	99.55	2,3	37403	*
PET1	6	10	3800	100	3800	100.00	All	1	*
PET2	10	10	87061	63	86812	99.71	Even	56	*
PET3	15	10	4015	100	4015	100.00	All	1	*
PET4	20	10	6120	100	6120	100.00	All	964	*
PET5	28	10	12400	100	12400	100.00	All	2813	*
PET6	39	5	10618	31	10603	99.86	2,3,7	29861	*
PET7	50	5	16537	5	16496	99.75	2	47657	*
SENTO1	60	30	7772	26	7734.1	99.51	2,3	36553	*
SENTO2	60	30	8722	20	8713.3	99.90	2	22262	*
WEING1	28	2	141278	85	141264	99.99	All \ 1,10	19652	*
WEING2	28	2	130883	86	130868	99.99	All \1	18500	*
WEING3	28	2	95677	93	95601	99.92	All \1	206	*
WEING4	28	2	119337	47	118079	98.95	2,3,4,6	14948	*
WEING5	28	2	98796	78	98713	99.92	All \ 1,3	15404	*
WEING6	28	2	130623	99	130619	99.99	All	15896	*
WEING7	105	2	1095445	25	1095258	99.98	2,3	33096	*
WEING8	105	2	624319	14	621761	99.59	2	32902	*
WEISH01	30	5	4554	100	4554	100.00	All	1	*

WEISH02	30	5	4536	71	4531.1	99.89	1 - 4	25642	*
WEISH03	30	5	4115	58	4090.2	99.40	2,3,5,6	25387	*
WEISH04	30	5	4561	100	4561	100.00	All	1	*
WEISH05	30	5	4514	100	4514	100.00	All	1	*
WEISH06	40	5	5557	38	5516.5	99.27	1,2,3	36761	*
WEISH07	40	5	5567	36	5521.9	99.19	1,2,3	20047	*
WEISH08	40	5	5605	31	5603.6	99.98	2,3,4	6968	*
WEISH09	40	5	5246	38	5186.3	98.86	1,2,3	20095	*
WEISH10	50	5	6339	28	6264.3	98.82	1,2	17903	*
WEISH11	50	5	5643	26	5474.6	97.02	2,3	21340	*
WEISH12	50	5	6339	29	6266.6	98.86	1,2,3	12191	*
WEISH13	50	5	6159	33	6018.9	97.73	1,2,3	18184	*
WEISH14	60	5	6954	24	6857.8	98.62	1,2	11359	*
WEISH15	60	5	7486	15	7407.3	98.95	2	8241	*
WEISH16	60	5	7289	33	7278.6	99.86	1,2,3	2789	*
WEISH17	60	5	8633	32	8608.8	99.72	1,2,3	11245	*
WEISH18	70	5	9580	3	9537.3	99.55	2	15890	*
WEISH19	70	5	7698	23	7686.5	99.85	1,2	4274	*
WEISH20	70	5	9450	25	9415.5	99.63	1,2	8058	*
WEISH21	70	5	9074	21	9053.7	99.78	2,3	5579	*
WEISH22	80	5	8947	1	8910.3	99.59	1	1974	*
WEISH23	80	5	8344	8306	8162	97.82	3	11686	
WEISH24	80	5	10220	25	10198	99.78	1,2	5762	*
WEISH25	80	5	9939	30	9870.1	99.31	1,2,3	7026	*
WEISH26	90	5	9584	9581	9340.3	97.46	3	13681	
WEISH27	90	5	9819	100	9819	100.00	All	1	*
WEISH28	90	5	9492	23	9329.9	98.29	1,2	8826	*
WEISH29	90	5	9410	10	9308	98.92	2	8214	*
WEISH30	90	5	11191	16	11184	99.94	2,3	6326	*