International Journal of Applied Mathematical Research, 3 (4) (2014) 422-431
©Science Publishing Corporation
www.sciencepubco.com/index.php/IJAMR
doi: 10.14419/ijamr.v3i4.3378
Research Paper

On the branch and cut method for multidimentional mixed integer Knapsack problem

Mostafa Khorramizadeh*, Zahra Rakhshandehroo

Department of Mathematical Sciences, Shiraz University of Technology, Shiraz 71555-313, Iran
**Corresponding author E-mail: m.khorrami@sutech.ac.ir*

Abstract

In this paper, we examine the effect of the feasibility pump (FP) method on the branch and cut method for solving the multidimentional mixed integer knapsack problem. The feasibility pump is a heuristic method, trying to compute a feasible solution for mixed integer programming problems. Moreover, we consider two efficient strategies for using the feasibility pump in a branch and cut method and present some tables of numerical results, concerning the application and comparison of using these strategies in the branch and cut method for solving the multidimentional mixed integer knapsack problem. The numerical results indicate that for the majority of the test problems, by using the FP or the imroved version of the FP we can improve the efficiency of the branch and cut method for solving the multidimentional mixed integer knapsack problem.

Keywords: *Feasibility pump method, Branch and cut method, multidimentional mixed integer knapsack problem.*

1. Introduction

Given a collection of items having both a weight and a usefulness, the knapsack problem is to fill a bag whose capacity is constrained while maximizing the sum of the usefulness of the items contained in the bag. In the mixed integer knapsack problem the number of some items are allowed to be continuous while the number of the remaining variables must be integer. The mixed integer knapsack problem belongs to the class of NP-hard problems [5]. The original version of the knapsack problem has one constraint relating to the capacity of the bag, while in the multidimentional knapsack problem we have several constraints of this type. This problem has many applications in the mathematical sciences and engineering [5, 7]. On efficient algorithm for solving the multidimentional mixed integer knapsack problem is the branch and cut method [1, 6]. Recently, Bertacco et al. [3] presented an efficient heuristic algorithm, called the feasibiltiy pump, for finding a feasible solution of mixed integer programming problems and showed that their heuristic algorithm is able to improve the efficiency of the branch and cut method. Then, Boland et al. [2] introduced an improvement of the feasibility pump method. One important issue is the study of the effect of the feasibity pump method to a branch and cut method when applied to solve the multidimentional mixed integer knapsack problem.

In this paper we first describe original version of the FP method [3] for computing a feasible solution of the mixed integer programming problems. Then, we explain the improved version of the FP, presented in [2]. The original version of the FP tries to find the feasible solutions without taking objective function into account, while the improved version usually finds solutions with better objective value, but is more likely to fail to find a feasible solution [2]. Then, we apply three branch and cut methods to a set of multidimentional mixed integer knapsack standard test problems, taken from [1]. The first branch and cut method uses the original version of the FP, while the second branch and cut method uses the improved version of the FP. In the third branch and cut method we do not use the FP method. Finally, we present some numerical results to examine the effect of using the original version of FP and the improved version of FP on the implemented branch and cut methods for solving multidimentional mixed integer knapsack problem instances. We apply the implemented algorithms on 90 standard rest probelms. The numerical results show that for the majority of the test problem instances, by using the FP or the imroved version of the FP we can improve the efficiency of the branch and cut method for solving the multidimentional mixed integer knapsack problem.

In section 2, we describe the original and improved versions of the FP. In section 3, we study the steps of the branch and cut method. In section 4, we consider the multidimentional mixed integer knapsack problem. Finally, section 5 is devoted to the numerical results.

2. Feasibility pump method

In this section we explain the steps of the original and improved versions of the FP. Consider the following mixed integer programming problem:

$$min\{c^T x|\ Ax \leq b,\ l \leq x \leq u,\ x_j \in Z\ \forall j \in I\} \quad (MIP)$$

In the first step of the FP we round the solution x^* of the LP relaxation

$$min\{c^T x|; Ax \leq b,\ l \leq x \leq u,\ \} \quad (LP)$$

to an integer vector $\tilde{x} = [x]^I$. For every $S \subset N$ let $[x]^S$ be so that

$$[x_j]^S = \begin{cases} [x_j + 0/5] & if j \in S \\ x_j & if j \notin S. \end{cases} \tag{1}$$

In case $\tilde{x}$ is not feasible, we try to compute a vector in the set of feasible solutions of the LP relaxatuon, that minimizes $\Delta(x, \tilde{x}) = \sum_{j\in I} |x_j - \tilde{x}_j|$. The FP replaces x^* with the new computed vector and proceed. Moreover, FP terminates if $x^* = \tilde{x}$ or a predifined iteration limit is reached. Note that the closest point in the feasible set of the LP relaxation to $\tilde{x}$ is computed by solving the followhng LP problem:

$$\begin{array}{ll} min & \Delta(x,\tilde{x}) = \sum_{\substack{j\in I \\ \tilde{x}_j = l_j}} (x_j - l_j) + \sum_{\substack{j\in I \\ \tilde{x}_j = u_j}} (u_j - x_j) + \sum_{\substack{j\in I \\ l_j < \tilde{x}_j < u_j}} d_j, \\ s.t. & Ax \leq b,\ d \geq x - \tilde{x},\ d \geq \tilde{x} - x,\ l \leq x \leq u, \end{array} \tag{2}$$

where, the variables d_j model the nonlinear term $d_j = |x_j - \tilde{x}_j|$ for integer variables x_j that are not equal to one of their bounds in the rounded solution $\tilde{x}$. During the algorithm the same sequence of integer and LP-feasible points can be visited over and over again. To overcome this difficulty, each time an integer point $\tilde{x}$ is generated that was already visited in a prior iteration, we perform a restart. In a restart a random perturbation step is executed, which shifts some of the variables randomly up or down and installs this perturbed vector as new integer point $\tilde{x}$ to continue the search. The second isue accurs if for a larg number of iterations, there is no considerable improvment in the fractional measure

$$f(x^*) := \sum_{j\in I} (f(x_j^*)) \qquad , \qquad f(x_j^*) := |x_j^* - [x_j^* + 0/5]|.$$

To deal with the second isue we perform a restart if in a certain number of iterations no larg number of improvment is observed in fractional measure. For fractional considrations, the original vergen of the FP consists of three stages. In the first stage we relax the integrality conditions on the general integer variables and perform a certain number of iterations (called a pumping round) just on the binary variables $B \subset I$. If the performance of the first stage dose not give us the feasible solution, we apply the second stage. We start the second stage from the initial point $\tilde{x}$ which is the closest point to the set of feasible solutions of the LP, that was visited during the first stage. Then, we execute a certain number of iterations on all integer variables. If no feasible solution is found by performing the second stage, we enter the third stage. In the third stage we use a point $\tilde{x}$ from the second stage closest to the set of feasible solutions of LP and solve the MIP:

$$min\{\Delta(x,\tilde{x})|\ Ax \le b,\ l \le x \le u,\ x_j \in Z\ \forall j \in I\}$$

The third stage stops if the first feasible solution is found or if a a certain iteration limit is reached.

Next we describe the improved version of the FP. The main idea of the improved FP is to find a feasible solution x of the problem for wich the objective value c^Tx is as small as possible. In [2] Boland et al. considerd this issue by gradually reducing the influence of the original objective function c^Tx and increasing the weight of the artificial objective function of the FP Δ. Indeed assume that $c \neq 0$. In the improved FP the distance function $\Delta(.,\tilde{x})$ is replaced with a convex combination of Δ and c:

$$\Delta_\alpha(x,\tilde{x}) = (1-\alpha)\Delta(x,\tilde{x}) + \alpha\frac{||\Delta||}{||c||}c^Tx, \quad \alpha \in [0,1],$$

where $||.||$ is the Euclidean norm of a vector, and Δ is the objective function vector of the original FP. In the first stage we have $||\Delta|| = \sqrt{|B|}$ and in the second stage we have $||\Delta|| = \sqrt{|I|}$. In pumping rounds α is decreased geometricly, i.e. $\alpha_{i+1} = \phi\alpha_i$ and $\alpha_0 \in [0,1]$ where $\phi < 1$ and α_i denotes the value of α_i in the ith iteration. In the improved version of the FP, we avoid cycling as follows. We remember paires $(\tilde{x},\alpha_i)$ and in the ith iteration, we perform a restart if $\tilde{x}$ was visited at itration $i' < i$ with $\alpha_{i'} - \alpha_i \le \delta_\alpha$ and $\delta_\alpha \in [0,1]$, see [2] for details.

3. Branch and cut method

In this section we present an overview of the branch and cut method for solving the MIP [7, 4]. The branch and cut method is a generization of the branch and bound method and consists of three major parts. The first part is called the enumerative part. In the enumerative part we perform the initialazation, bounding, fixing and setting of the variables, bounding, selection and fathoming. The initialization is concerned with operations such as reading in the problem data, setting of the parametes of the algorithm and etc. In the bounding part we perform some operations relating to the update of the best solution fund so far. By fixing and setting of variables we mean the assignment of a constant value to some variables of the problem, in the remainder of the branch and cut method. In the selection part we choose an active node of the branch and cut tree for processing. The selection of the next node can be perfomed according to the depth first search, the breath first search and the best-node first search criteria. In the fathoming part we stop processing a node further if the optimal value of the LP ralaxation of this node is greater than or equal to the best optimal value found so far or if the LP relaxation of the node is infeasible or the optimal solution of the LP relaxation of the node is feasible for the MIP. Another criteria for fathoming is called fathomming by contradiction. Fathomming by contradiction occures if we observe that some fixed variables must be fixed or set to another value. Finally a node is fathomed if the tailing off happenes, i.e. for certain number of successive iterations the optimal value of the LP relaxation of the corresponding nodes does not improved considerably.

The second part of the branch and cut algorithm is concened with the computation of lower bound. This part consists of the initialization of the new node, solving of the coressponding LP, separation and elimination. After initialization of the new node and solving the corresponding LP, in the separation part, we generate a valid inequality that cuts off the fractional optimal solution of the LP. In the elimination part we eliminate some valid inequaliteis that are not effective.

In the third major part of the branch and cut method, we compute an upper bound. In this part we try to find a feasible solution for the MIP and improve quality of the best solution found so far, in the every node of the branch and cut tree. The main idea of this part is that if we can improve the quality of the best solution so far, then we can decrease the width of the branch and cut tree and save a lot of computational cost. This strategy is prooved to be effective in the practice [3, 2]. One of the most efficient heuristic methods for finding a feasible solution of MIP, relating to this part is the FP method.

4. Mixed integer knapsack model

In this section we briefly describe the mixed integer programming formulation of the multidimentional mixed integer knapsack problem [1]. In the following let I and C be the set of indices of integer and continuous variables, respectively. For every $i \in I$, let x_i and c_i denote the number and the usefulness of the ith integer item, respectively.

For every $j \in C$, let w_j and d_j denote the number and the usefulness of the jth continuous item, respectively. Then, the mixed integer programming formulation of the multidimentional mixed integer knapsack problem is as follows:

$$\begin{array}{lll} max & \sum_{i\in I} c_i x_i + \sum_{j\in J} d_j w_j & \\ s.t. & \sum_{i\in I} a_{ir} x_i + \sum_{j\in J} g_{jr} w_j \le b_r, & r \in R, \\ & x_i \le u_i, \quad w_j \le v_j, & i \in I,\ j \in C, \\ & x_i \ge 0, \quad w_j \ge 0, & i \in I,\ j \in C. \end{array}$$

where, b_r is the capacity correspoding to the rth constraint and a_{ir} and g_{jr} are the space occupied in the rth constraint by the ith integer variable and the jth continuous variable, respectively. u_i and v_j denote the upper bound on the ith integer variable and the jth continuous variable, respectively. Moreover, no sign restriction is imposed on b_r, a_{ir} and g_{jr} and they can take negative values.

5. Numerical results

In this section we describe our computational experiments. For the implementation we used a camputer with a a 2.53GHz Corei3/Windows7 with 2GB RAM using the CPLEX Version 12.6 with default settings. The 90 problem instances are taken from [1]. Table 1 is concerned with the properties of the generated test problems. In this table np denotes the problem number, $|I|$, $|C|$ and $|R|$ denote the number of integer variabes, the number of continuous variables and the number of constraints of the generated test problem, respectively. Tables 2 and 3 are devoted to the comparison of the computing time and the number of nodes of the branch and cut tree for solving the corresponding multidimentional mixed integer unbounded knapsack pronlem instance. In this table, np denotes the problem number. Moreover, $TwFP$, $ToFP$ and $TimFP$ denote the computing time of the branch and cut method without using the FP, by using the original version of the FP and by using the improved version of the FP, respectively. Finally, $NNwFP$, $NNoFP$ and $NNimFP$ denote the number of nodes of the branch and cut tree without using the FP, by using the original version of the FP and by using the improved version of the FP, respectively. In Tables 4 and 5, we compared the number of iterations and the value of the gap resulting from the branch and cut tree for each problem instance. In tables 4 and 5, $ITwFP$, $IToFP$ and $ITimFP$ denote the number of iterations of the branch and cut method without using the FP, by using the original version of the FP and by using the improved version of the FP, respectively. Moreover, $GwFP$, $GoFP$ and $GimFP$ denote the gap obtained after the application of the branch and cut method without using the FP, by using the original version of the FP and by using the improved version of the FP, respectively.

The numerical results of tables 2 and 3 show that for 27 instances the best computing times are obtained after the application of the branch and cut method using the original version of the FP. For 30 instances the best computing times are related to the branch and cut method using the improved version of the FP. For 33 problem instances the best computing times are obtained after the application of the branch and cut method without using the FP. Therefore, from 90 test problems, for 57 problem instances using the orignal or the improved version of the FP reduces the computing time of the branch and cut method.

From 90 generated prolem instances, in 31 cases the branch and cut method without using any versions of the FP has less number of nodes, while in 54 cases using either the original version or improved version of the FP results in less number of nodes of the corresponding branch and cut tree. It can also be verified that for 28 instances using the improved FP results in fewer number of nodes and for 26 instances using the original version gives better results. In 5 cases the number of nodes obtained by using all three branch and cut methods are equal.

In tables 4 and 5, for 31 problem instances the number of iterations of the branch and cut method without using the FP is less than that of the branch and cut method using at least one version of the FP. However, for 57 problem instances the branch and cut method that uses a version of the FP needs less number of iterations. In 34 cases the improved version of the FP needs less number of iterations and in 23 cases the original version of the FP needs less number of iterations. In 2 cases the number of iterations of three branch and cut methods are equal. From the 90 test problems, in 32 test instances, the branch and cut method without using the FP, gives a better value for gap than the branch and cut method using at least a version of the FP, while in 53 cases the branch and cut method using a version of FP gives a better value for the gap. For 31 cases the gap obtained by using the original version of the FP is the best and for 22 instances the improved version of the FP gives the best value for the gap. From the numerical results of tables 1, 2 and 3 we conclude that for most of the test instances, by using the original or improved version of the FP we can improve the efficiency of the resulting branch and cut method.

Table 1: Properties of the problem instances

np	$\|I\|$	$\|C\|$	$\|R\|$	np	$\|I\|$	$\|C\|$	$\|R\|$	np	$\|I\|$	$\|C\|$	$\|R\|$
1	250	5	50	31	250	20	50	61	500	10	50
2	250	5	50	32	250	20	50	62	500	10	50
3	250	5	50	33	250	20	50	63	500	10	50
4	250	5	50	34	250	20	50	64	500	10	50
5	250	5	50	35	250	20	50	65	500	10	50
6	250	5	75	36	250	20	75	66	500	10	75
7	250	5	75	37	250	20	75	67	500	10	75
8	250	5	75	38	250	20	75	68	500	10	75
9	250	5	75	39	250	20	75	69	500	10	75
10	250	5	75	40	250	20	75	70	500	10	75
11	250	5	100	41	250	20	100	71	500	10	100
12	250	5	100	42	250	20	100	72	500	10	100
13	250	5	100	43	250	20	100	73	500	10	100
14	250	5	100	44	250	20	100	74	500	10	100
15	250	5	100	45	250	20	100	75	500	10	100
16	250	10	50	46	500	5	50	76	500	20	50
17	250	10	50	47	500	5	50	77	500	20	50
18	250	10	50	48	500	5	50	78	500	20	50
19	250	10	50	49	500	5	50	79	500	20	50
20	250	10	50	50	500	5	50	80	500	20	50
21	250	10	75	51	500	5	75	81	500	20	75
22	250	10	75	52	500	5	75	82	500	20	75
23	250	10	75	53	500	5	75	83	500	20	75
24	250	10	75	54	500	5	75	84	500	20	75
25	250	10	75	55	500	5	75	85	500	20	75
26	250	10	100	56	500	5	100	86	500	20	100
27	250	10	100	57	500	5	100	87	500	20	100
28	250	10	100	58	500	5	100	88	500	20	100
29	250	10	100	59	500	5	100	89	500	20	100
30	250	10	100	60	500	5	100	90	500	20	100

Table 2: Comparison of the computing time and number of nodes

np	$TwFP$	$ToFP$	$TimFP$	$NNwFP$	$NNoFP$	$NNimFP$
1	0.48	0.37	0.48	0	496	0
2	0.39	0.42	0.51	0	0	0
3	0.51	0.39	0.64	814	1115	1772
4	0.38	0.36	0.36	0	0	0
5	0.38	0.33	0.36	0	0	0
6	10.4	29.45	1.61	70363	240362	7456
7	0.61	0.72	0.61	2573	4817	3104
8	0.61	0.86	0.69	2222	4604	3426
9	3.21	3.01	3.50	22984	22665	24414
10	1.56	2.64	2.42	6418	17095	12683
11	4.71	5.10	4.58	16231	22898	21290
12	0.75	0.83	0.95	3131	2815	3877
13	2.08	5.72	4.18	3159	20363	18535
14	0.55	0.78	0.55	1510	4085	1565
15	2.22	2.34	2.48	3935	5456	4996
16	0.41	0.37	0.44	475	545	0
17	0.50	0.47	0.45	1984	1716	1495
18	0.80	0.76	0.83	5165	4946	6138
19	0.47	0.47	0.55	1062	1022	2545
20	0.41	0.42	0.36	179	184	170
21	7.68	4.57	6.72	46566	22911	38360
22	3.79	11.23	3.74	13315	49190	13496
23	2.22	6.33	1.50	12552	30819	5542
24	2.84	7.58	2.95	16306	29557	15507
25	2.17	1.26	1.43	11334	4775	8146
26	2.14	2.03	2.06	3184	3567	4767
27	2.62	2.59	2.71	16139	3911	6764
28	3.09	2.40	3.96	8687	8408	13681
29	2.93	3.23	2.54	8812	7120	6903
30	3.21	8.74	3.03	3926	36751	4010
31	1.34	1.79	1.84	9548	14533	13736
32	0.81	1.03	0.81	4191	5773	5528
33	1.50	1.17	1.18	8573	5643	5353
34	1.08	1.15	0.65	5681	5867	5048
35	0.45	0.50	0.50	2950	2312	2469
36	2.79	5.16	3.26	21489	33893	21768
37	2.00	1.67	2.21	4475	4459	3777
38	2.34	3.20	4.31	8998	16305	25723
39	44.24	72.62	61.01	331734	632811	584904
40	1.67	1.75	2.36	9408	8155	11937
41	2.70	2.82	2.84	6159	5252	6584
42	3.20	3.99	7.36	9749	15483	31502
43	3.65	3.99	3.85	6675	6567	6743
44	4.85	3.01	0.51	22719	8388	13400
45	5.29	4.15	3.84	17545	8282	6160

Table 3: Comparison of the computing time and number of nodes

np	$TwFP$	$ToFP$	$TimFP$	$NNwFP$	$NNoFP$	$NNimFP$
46	0.76	0.69	0.69	985	1700	1164
47	0.69	0.66	0.67	814	812	835
48	1.03	1.16	0.89	3845	3846	5811
49	0.56	0.59	0.66	1366	1662	1890
50	0.42	0.39	0.42	0	0	0
51	2.39	1.93	2.04	10457	9021	7551
52	12.72	4.71	2.82	67590	17050	9799
53	1.97	1.90	1.98	7612	8134	6101
54	1.83	1.67	3.38	8520	2392	24692
55	2.00	1.87	1.97	6144	5765	6809
56	2.64	1.40	2.23	8598	5152	5492
57	12.71	3.59	3.12	55424	14773	12746
58	1.76	1.23	1.89	2778	4219	3965
59	2.62	2.67	11.14	6089	8629	46361
60	2.81	2.40		7968	6903	
61	2.04	2.17	2.26	5878	6068	5713
62	0.87	0.87	1.22	2684	2933	4819
63	1.23	0.86	1.29	5087	4177	4718
64	1.15	1.11	1.01	4798	4469	3478
65	0.52	0.51	0.64	0	0	0
66	2.34	2.03	1.86	9320	6495	5602
67	7.21	13.82	13.03	25142	78036	57225
68	3.73	45.26	3.89	19411	27870	18165
69	3.34	11.33	3.25	20131	48071	155487
70	2.37	3.15	2.84	8928	15889	13355
71	73.27	8.75	15.76	505691	153787	323815
72	2.01	1.67	1.26	5406	4156	1627
73	2.48	2.25	2.15	3469	5717	4233
74	3.07	3.28	2.76	6565	10465	6966
75	5.02	43.79	15.12	11792	272509	67032
76	1.82	2.34	5.41	1080	13394	32528
77	1.31	1.06	1.42	4536	4701	6199
78	1.34	1.20	1.47	4536	5105	5189
79	1.45	1.47	1.50	5517	5655	5559
80	0.69	0.72	0.81	868	884	1468
81	5.01	20.61	11.58	23232	202501	56874
82	3.24	4.94	9.22	11371	21813	30544
83	188.45	489.25	17.91	1322238	2963241	127115
84	14.54	15.59	3.32	90804	88455	13174
85	3.78	14.48	4.82	9217	76941	15538
86	2.51	4.15	4.09	5940	18537	16813
87	2.29	2.22	2.23	6834	5232	5223
88	2.81	2.79	1.62	4821	5043	1796
89	2.92	2.78	3.65	8293	4887	4648
90	2.03	2.14	2.07	4559	5179	4445

Table 4: Comparison of the number of iterations and gap

np	$ITwFP$	$IToFP$	$ITimFP$	$GwFP$	$GoFP$	$GimFP$
1	750	2528	793	0.25	-	3.31622
2	694	694	694	-	-	-
3	4549	4287	7769	3.039	1.329	3.010
4	618	583	591	2.475	-	0.513
5	722	722	722	-	-	-
6	485948	1726116	40445	-	4.837	4.382
7	12195	23514	14157	4.75	4.651	4.673
8	17733	30941	24321	3.526	3.908	2.811
9	22984	966356	125719	4.878	5.132	4.526
10	33725	78151	71449	4.797	5.017	4.653
11	130494	158240	136655	6.741	6.537	6.749
12	13405	11719	16134	6.404	6.440	6.599
13	17377	159901	142619	-	6.852	7.023
14	6446	15739	5462	6.014	3.839	2.936
15	18820	28337	30571	6.120	5.774	6.361
16	3132	3259	813	2.041	3.107	-
17	8737	7192	6792	1.808	1.077	0.975
18	23166	21720	24479	2.270	1.783	3.055
19	5611	5575	8492	2.508	2.348	3.307
20	1838	1882	1769	-	2.966	-
21	222602	137070	169458	4.810	4.824	4.841
22	69734	391900	93071	4.704	4.921	4.930
23	71105	177877	37672	5.118	5.056	3.258
24	97136	220591	93723	5.037	5.102	4.343
25	50092	25969	38112	4.972	4.466	4.858
26	14868	16676	22994	6.317	6.797	6.793
27	25005	18235	29694	5.343	6.304	6.197
28	61420	52317	84621	6.446	6.813	6.705
29	56060	34479	33405	5.415	7.105	6.696
30	17043	193888	17431	5.681	6.820	6.539
31	38430	60269	63205	2.403	2.682	3.240
32	21569	21960	23102	3.129	-	3.258
33	27631	14572	12833	3.209	3.100	3.046
34	22838	23210	14809	3.399	-	2.748
35	7774	5696	5029	3.081	2.333	2.948
36	102539	81969	81999	4.969	4.753	4.560
37	17051	19944	13624	4.997	4.812	4.097
38	46620	80126	143011	5.011	5.018	5.210
39	1697631	2953633	2149683	5.220	5.220	5.225
40	31955	30432	62545	4.850	4.655	5.013
41	22306	18439	25473	6.807	6.952	6.370
42	32451	58800	112860	6.561	6.263	6.813
43	24258	22410	21755	6.977	6.253	6.266
44	90375	28503	47364	7.065	7.092	7.111
45	64412	30741	23814	6.5832	6.400	6.548

Table 5: Comparison of the number of iterations and gap

np	$ITwFP$	$IToFP$	$ITimFP$	$GwFP$	$GoFP$	$GimFP$
46	5816	8184	6214	2.919	2.384	2.642
47	3963	3927	3960	-	-	-
48	1483	14711	23028	-	1.747	3.097
49	5852	5817	6508	3.158	1.537	1.976
50	430	429	407	-	-	-
51	72673	46329	42139	4.888	4.769	4.739
52	398626	148348	57436	4.889	4.815	4.555
53	43704	40645	38372	4.204	4.750	4.821
54	32196	27742	114320	3.688	4.113	5.157
55	38915	33085	35581	3.330	4.762	4.693
56	55131	44097	37867	5.759	6.151	6.489
57	467384	88935	84622	6.563	5.737	6.573
58	19529	30550	26879	-	5.963	2.030
59	37525	54120	378039	5.484	6.867	7.058
60	53390	53275		6.096	6.540	
61	25099	24664	22851	0.873	1.381	0.418
62	12205	14301	23259	2.193	2.749	2.257
63	18579	14928	14469	3.025	2.572	0.203
64	20051	18348	11369	3.059	1.657	3.23
65	683	736	846	1.766	2.739	-
66	60657	36264	30635	3.857	4.019	4.672
67	212605	433766	476890	4.808	4.942	4.917
68	90872	1910354	95628	4.992	5.103	5.007
69	117520	352917	87444	5.183	5.197	4.952
70	38216	70399	55249	5.026	4.995	4.838
71	2987870	153787	323819	6.926	6.919	6.919
72	67537	38813	16345	6.826	6.018	4.230
73	20075	36243	21245	2.263	5.776	4.592
74	47292	91815	46664	7.003	6.664	6.862
75	56764	1562728	474330	6.611	6.777	6.767
76	36240	48065	146609	2.770	3.291	3.193
77	15961	16799	25469	-	2.402	3.191
78	15961	19127	17103	-	2.817	2.274
79	17230	21117	21501	0.934	3.190	3.164
80	3190	3382	6151	-	1.224	3.320
81	87545	833289	286222	4.895	4.902	4.869
82	62544	143951	225584	5.030	5.017	5.012
83	10028928	24535349	609941	5.115	5.115	5.112
84	406629	487306	46243	5.251	5.268	4.978
85	34029	458821	71820	5.069	5.100	4.972
86	22881	82056	84012	6.678	6.978	6.821
87	24985	19732	20073	6.831	6.618	6.490
88	16404	19021	6976	5.801	6.034	6.953
89	32017	20043	19814	6.942	7.187	6.472
90	18980	22093	18513	4.493	6.431	5.468

Acknowledgements

The authors thank the Shiraz university of Technology for supporting this work.

References

[1] A. Atamturk, On the facets of the mixedinteger knapsack polyhedron, Mathematical Programming Serries B, 98: 145175, 2003.

[2] Boland N. L., Eberhard A.C., Engineer F. and Tsoukalas A, Improving the feasibility pump. Discrete Optimization, 4(1):77-86, 2007.

[3] Bertacco L. Fischetti M. and Lodi A, A feasibility pump heuristic for general Mixed-Integer Problems. Discrete Optimization, 4(1):63-76, 2007.

[4] M. Elf, C. Gutwenger, M. Junger and G. Rinaldi, branch and cut Algorithm for combinatorial optimization and their implementation in ABACUS,Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions., Lecture Notes in Computer Science. 2241 Springer 2001, pp. 157-222.

[5] H. Kellerer, U. Pferschy and D. Pisinger, Knapsack Problems, Springer-Verlag, Berlin, 2004.

[6] R. Weismantel, On the 0/1 knapsack polytope. Mathematical Programming, 77:4968, 1997.

[7] L. A. Wolsey, *Integer Programming*, John Wiley and Sons, New York, 1998.