Noname manuscript No.
(will be inserted by the editor)

Jakob Puchinger · Sam Huston · Peter J. Stuckey

The Core Concept for Mixed Binary Programming

Received: date / Accepted: date

Abstract In this paper we present a new extension of the core concept for knapsack problems towards general mixed binary programs. This idea will be used to reduce problems to a core of hard to decide variables. We describe the core concept for mixed binary programs before defining an extended efficiency measure for ordering the binary variables such that a good approximate core can be devised from the solution to the LP relaxation of the MBP. Computational experiments on selected MIPLIB 2003 benchmark instances show the advantages of our approach, often yielding higher quality solutions compared to solving the original problem, given a fixed run-time. The competitiveness of our algorithm is emphasised by the fact that a new best known solution has been found for one of the benchmark problems.

Keywords Mixed Binary Programming · Primal Heuristics · Core Concept

1 Introduction

The core concept has proven to be a successful way of reducing knapsack (KP) [11,12], multidimensional knapsack (MKP) [15,16], and binary programming (BP) [9] problems to a core of interesting variables. The reduced problems are usually faster to solve than the original problem and yield high-quality solutions. Given fixed run-times, solving the reduced problems often yields better results than solving the original ones. In this paper we extend the core concept to Mixed Binary Programs (MBP), these are Mixed Integer

NICTA Victoria Laboratory
Department of Computer Science and Software Engineering,
University of Melbourne, Victoria 3010, Australia
E-mail: {jakobp|pjs}@csse.unimelb.edu.au, hustons@unimelb.edu.au

Programs (MIP) where the integer variables are restricted to 0 and 1:

$$\text{maximise} \quad \sum_{i=1}^{n} c_i x_i + \sum_{j=1}^{m} d_j y_j \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} a_{il} x_i + \sum_{j=1}^{m} b_{jl} y_i \leq e_l \qquad l = 1, \ldots, L \tag{2}$$

$$x_i \leq 1 \qquad i = 1, \ldots, n \tag{3}$$

$$x_i \geq 0, \text{ integer} \qquad i = 1, \ldots, n \tag{4}$$

$$y_j \text{ free} \qquad j = 1, \ldots, m. \tag{5}$$

We first describe related work and give some background information After describing the core concept for mixed binary programs, an extended efficiency measure is defined for ordering the binary variables such that a good approximate core can be devised from the solution to the LP relaxation of the MBP. Finally, computational experiments on selected MIPLIB 2003 benchmark instances show the advantages of our approach, often yielding higher quality solutions compared to solving the original problem, given a fixed run-time. The competitiveness of our algorithm is emphasised by the fact that a new best known solution has been found for one of the benchmark problems.

2 Related Work and Background

The work most closely related to this paper is the application of the core concept to the MKP [15,16] and BP [9]. In this paper, we extend these results to mixed binary programming, and show that the core concept can help to improve solution quality when given a fixed run-time or to obtain high-quality solutions faster.

2.1 Related Work

Recently, very interesting results have been achieved with heuristics for MBP and MIP with the goal of devising better feasible solutions earlier in the optimisation process. An extensive overview of primal heuristics for mixed integer programming can be found in [3].

Local Branching [7] combines local search and general branch-and-bound by introducing local branching constraints forcing the search to explore the neighbourhoods of current feasible solutions first.

In Relaxation Induced Neighbourhood Search (RINS) [5] subproblems for finding better feasible solutions are solved at some nodes of the branch-and-bound tree. The subproblems are obtained by fixing the variables having identical values in the current best feasible solution and in the current solution of the LP-relaxation, leaving the remaining variables free.

RINS and local branching are local-search based ideas, reducing the subproblems to certain neighbourhoods around a currently feasible solution. Our

approach requires an LP solution only, and does not make use of feasible solutions at all.

Some other recent progress was made on start heuristics. The feasibility pump [6] proved to be a very promising idea for MBP. It starts by solving the LP relaxation yielding a fractional solution, which is used to derive an integer solution by rounding. An LP is then used to determine the nearest LP feasible point to this integer solution. When this point is integral, a feasible solution for the original MIP has been found. This process is repeated until a feasible solution is determined or a stopping criterion is met. The objective function was taken into account into the feasibility pump in [1] improving the quality of the generated start solutions.

2.2 Background

The core concept was first presented for the classical 0/1-knapsack problem [2] and led to very successful knapsack algorithms [11–13]. These ideas were also studied in the context of bi-criteria knapsack problems in [8]. The core concept was successfully extended to the MKP [15,16] and BP [9], leading to highly competitive heuristic algorithms.

The Knapsack Problem (KP) can be stated as

$$\text{maximise} \quad z = \sum_{j=1}^{n} p_j x_j \tag{6}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq c, \tag{7}$$

$$x_j \in \{0,1\}, \quad j \in 1,\ldots,n. \tag{8}$$

The classical greedy heuristic for KP packs the items into the knapsack in decreasing order of their *efficiencies* ($\frac{p_j}{w_j}$) as long as the knapsack constraint is not violated. The same ordering also defines the solution structure of the LP-relaxation, which consists of three parts: The first part contains all variables set to one, the second part consists of at most one *split item* (s), whose corresponding LP-value is fractional, and finally the remaining variables, which are always set to zero, form the third part.

Balas and Zemel [2] gave the following precise definition of the core of a one-dimensional 0/1-knapsack problem, based on the knowledge of an optimal integer solution x^*. Assume that the items are sorted according to decreasing efficiencies and define

$$\alpha := \min\{i | x_i^* = 0\}, \qquad \beta := \max\{i | x_i^* = 1\}. \tag{9}$$

The core is given by the items in the interval $C = \{\alpha, \ldots, \beta\}$. It is obvious that the split item is always part of the core.

The KP Core (KPC) problem is defined as

$$\text{maximise} \quad z = \sum_{j \in C} p_j x_j + \tilde{p} \tag{10}$$

$$\text{subject to} \quad \sum_{j \in C} w_j x_j \leq c - \tilde{w}, \tag{11}$$

$$x_j \in \{0, 1\}, \quad j \in C, \tag{12}$$

with $\tilde{p} = \sum_{j=1}^{a-1} p_j$ and $\tilde{w} = \sum_{j=1}^{a-1} w_j$. Obviously, the solution of KPC would suffice to compute the optimal solution of KP, which, however, has to be already partially known to determine C.

Pisinger [13] reported experimental investigations of the exact core size. He also studied the hardness of core problems, also giving a model for their expected hardness in [14].

3 The Core Concept for Mixed Binary Programming

In order to devise approximate cores for mixed binary programs, we adapt the ideas developed for the MKP [15,16]. In [9] we presented a preliminary adaptation for binary programs that is further extended here. The main goal is to be able to generate approximate cores yielding high quality near-optimal solutions.

3.1 The Core of a Mixed Binary Program

The main difference between the MKP and a MBP is that all coefficients are allowed to take positive and negative values. Furthermore non-integral variables are added to the problem. The core for a MBP consists of a subset of the binary variables, the non-binary variables are subsequently considered in the core problem.

Consider an MBP of the form (1) - (2) and assume that the binary variables x are ordered according to an efficiency measure E. Let $\{x^*, y^*\}$ be an optimal solution to the MBP, then define

$$\alpha := \min\{j \mid x_j^* = 0\} \quad \text{and} \quad \beta := \max\{j \mid x_j^* = 1\}. \tag{13}$$

We define the core of the MBP as the binary variables with indices $C = \{\alpha, \ldots, \beta\}$ and the core problem (MBPC) as:

$$\text{maximise} \quad \sum_{i \in C} c_i x_i + \sum_{j=1}^{m} d_j y_j + \tilde{c} \tag{14}$$

$$\text{subject to} \quad \sum_{i \in C} a_{il} x_i + \sum_{j=1}^{m} b_{jl} y_i \leq e_l - \tilde{a}_l \qquad l = 1, \ldots, L \tag{15}$$

$$x_i \leq 1 \qquad i \in C \tag{16}$$

$$x_i \geq 0, \text{ integer} \qquad i \in C \tag{17}$$

$$y_j \text{ free} \qquad j = 1, \ldots, m, \tag{18}$$

with $\tilde{c} = \sum_{i=1}^{\alpha-1} c_i$ and $\tilde{a}_l = \sum_{i=1}^{\alpha-1} a_{il},\ l = 1, \ldots, L$.

In contrast to KP, the solution of the LP-relaxation of a MBP does not consist of a single fractional split item, but up to L of its binary variables can have fractional values. They give rise to a whole *split interval* $S := \{s, \ldots, t\}$, where s and t are the first and the last index of binary variables with fractional values after sorting by efficiency E.

In order to extend the efficiency measure originally defined for the MKP, we create a tuple based measure for each binary variable x_i, by introducing ordering variables, o_i which take values representing a section of the ordering. Variables x_i are sorted in decreasing (lexicographic) order of efficiency (o_i, r_i), in other words they are first sorted by section variable, o_i, and then relative efficiency value r_i. These extensions are defined as follows:

$$(o_i, r_i) = \begin{cases} (7, \frac{c_i}{s_i}) & \text{if } c_i > 0 \wedge s_i < 0 \\ (6, c_i) & \text{if } c_i > 0 \wedge s_i = 0 \\ (5, \frac{1}{s_i}) & \text{if } c_i = 0 \wedge s_i < 0 \\ (4, \frac{c_i}{s_i}) & \text{if } c_i > 0 \wedge s_i > 0 \\ (4, \frac{s_i}{c_i}) & \text{if } c_i < 0 \wedge s_i < 0 \\ (4, 1) & \text{if } c_i = 0 \wedge s_i = 0 \\ (3, \frac{1}{s_i}) & \text{if } c_i = 0 \wedge s_i > 0 \\ (2, c_i) & \text{if } c_i < 0 \wedge s_i = 0 \\ (1, \frac{c_i}{s_i}) & \text{if } c_i < 0 \wedge s_i > 0, \end{cases} \tag{19}$$

where $s_i = \sum_{l=1}^{L} a_{il} u_l$ with u an optimal solution to the dual of the LP relaxation of the MBP.

The ordering is designed to minimise the size of the split interval, which is completely contained in section $o_i = 4$.

The sections $o_i \in \{7, 6, 5\}$ contain items that are potentially beneficial for the solution: they increase profit and/or "on average" remove weight from the optimal solution without decreasing profit. They are ordered to maximise profit and removal of weight. Similarly the sections $o_i \in \{1, 2, 3\}$ contain items that are potentially decreasing the solution quality: they decrease profit and/or add weight. Again they are ordered to maximise profit and removal of weight.

Using this efficiency measure the nature of the split interval can be characterised as follows. Let x^{LP}, y^{LP} be the optimal solution of the LP-relaxation of the MBP.

Theorem 1

$$x_i^{LP} = \begin{cases} 1 & \textit{if } r_i > 1 \textit{ or } o_i > 4, \\ \in [0, 1] & \textit{if } r_i = 1 \textit{ and } o_i = 4, \\ 0 & \textit{if } 0 \leq r_i < 1 \textit{ or } o_i < 4. \end{cases} \tag{20}$$

Proof The dual LP associated with the LP-relaxation of MBP is:

$$\text{minimise} \quad \sum_{l=1}^{L} e_l u_l + \sum_{i=1}^{n} v_i \tag{21}$$

$$\text{subject to} \quad \sum_{l=1}^{L} a_{il} u_l + v_i \geq c_i \qquad i = 1, \ldots, n \tag{22}$$

$$\sum_{l=1}^{L} b_{jl} u_l = d_j \qquad j = 1, \ldots, m \tag{23}$$

$$u_l \geq 0, \qquad l = 1, \ldots, L, \tag{24}$$

$$v_i \geq 0, \qquad i = 1, \ldots, n, \tag{25}$$

where u_l are the dual variables corresponding to the problem's constraints and each v_i corresponds to the inequality $x_i \leq 1$. For the optimal primal and dual solutions the following complementary slackness conditions hold for $i = 1, ..., n$. .

$$x_i \left(\sum_{l=1}^{L} a_{il} u_l + v_i - c_i \right) = 0 \tag{26}$$

$$v_i \left(x_i - 1 \right) = 0 \tag{27}$$

For more details on linear programming duality and complementary slackness conditions, refer to any textbook on linear programming, e.g. [4].

Consider the top three sections ($o_i \in \{7, 6, 5\}$). Clearly in each case we have that $c_i > s_i$. Hence satisfying equation (22) requires that $v_i > 0$. Therefore equation (27) implies that $x_i = 1$.

Consider the bottom three sections ($o_i \in \{1, 2, 3\}$). In each case $c_i < s_i$. Hence the expression $\sum_{i=1}^{m} a_{il} u_l + v_i - c_i$ or equivalently $s_i + v_i - c_i$ is greater than 0, since $v_i \geq 0$. In order to satisfy equation (26) $x_i = 0$.

For section $(4, \frac{c_i}{s_i})$ we consider two cases. If $r_i = \frac{c_i}{s_i} > 1$ then $c_i > s_i$ since $s_i > 0$ and the same reasoning as for cases $o_i \in \{7, 6, 5\}$ applies, while if $r_i < 1$ then $c_i < s_i$ and the same reasoning as cases $o_i \in \{1, 2, 3\}$ applies.

For section $(4, \frac{s_i}{c_i})$ we consider two cases. If $r_i = \frac{s_i}{c_i} > 1$ then $c_i > s_i$ since $c_i < 0$ and the same reasoning as for cases $o_i \in \{7, 6, 5\}$ applies, while if $r_i < 1$ then $c_i < s_i$ and the same reasoning as cases $o_i \in \{1, 2, 3\}$ applies.

For the remaining case, $r_i = 1$ and $o_i = 4$ there is nothing to prove.

□

The ordering within the top and bottom groups $o_i \in \{1, 2, 3\}$, and $o_i \in \{5, 6, 7\}$ is not set by the proof, these orderings are based upon maximising profit. Other possible orderings could be considered for these groups without changing our characterisation of the split interval.

An illustration of the ordering (o_i, r_i) is given in Table 1. For this example $m = 0$, that is there are no real variables. The coefficients of each variable x_i in the two constraints are given in the first two columns, and then the coefficients in the objective function. The last row gives the right hand side

of the constraints. The fourth and fifth columns show the values for s_i and the efficiency value E for each variable. Finally the last two columns give the value in the LP and IP optimal solutions of the problem. This example shows some of the sections described above. As predicted by Theorem 1, the split interval exists entirely within section $o_i = 4$.

Table 1 Example BP problem with two constraints. The 3 sections are separated based upon the IP solution. This example shows an exact core using the efficiency measure defined in Theorem 1.

a_{1i}	a_{2i}	c_i	s_i	(o_i, r_i)	LP	IP
-1	-1	1	-0.89	(7,0.89)	1.00	1
-9	12	7	-5.17	(7,0.74)	1.00	1
0	0	8	0.00	(6,8.00)	1.00	1
-2	13	12	0.24	(4,50)	1.00	1
5	0	19	3.77	(4,5.04)	1.00	1
-5	-4	-3	-4.31	(4,1.44)	1.00	1
16	21	18	14.88	(4,1.21)	1.00	1
13	15	14	11.81	(4,1.19)	1.00	**0**
-6	-10	-5	-5.87	(4,1.17)	1.00	**1**
20	-8	14	14.00	(4,1.00)	**0.71**	**1**
-6	-11	-6	-6.00	(4,1.00)	**0.85**	0
20	16	15	17.22	(4,0.87)	0.00	0
10	8	6	8.61	(4,0.70)	0.00	0
-1	-14	-4	-2.63	(4,0.66)	0.00	0
14	9	5	11.76	(4,0.43)	0.00	0
16	-3	3	11.66	(4,0.26)	0.00	0
-7	23	-9	-2.19	(4,0.24)	0.00	0
-1	10	0	0.59	(3,1.69)	0.00	0
7	0	-5	5.28	(1,0.95)	0.00	0
24	10	-9	19.43	(1,0.46)	0.00	0
20	31					

3.2 A Fixed Core Algorithm for Mixed Binary Programming

We make use of the previously presented ideas to develop a simple primal heuristic for MBP. We determine an approximate core using the solution to the LP relaxed MBP. First the items are ordered according the efficiency value E. Second we determine an approximate core by adding δ variables to either side of the centre of the split interval. We finally solve the smaller MBP resulting of fixing the variables outside the approximate core. If the number of variables on one side of the centre of the split interval is smaller than δ, we simply add all variables on that side, but do not change the value for δ on the other side. This implies that value of δ bigger than $0.5n$ can yield approximate cores with less than n variables. Depending on the instance and on the chosen value for δ. The reduced MBP can result in an infeasible model.

4 Computational Experiments

We selected easy BP and MBP instances from MIPLIB[1] in order to evaluate the quality of the devised approximate cores. We further selected hard BP and MBP instances from MIPLIB for demonstrating the advantages of our approach for devising high-quality solutions in short run-times. We selected those instances where applying our core based approach is meaningful, i.e. where the number of columns is significantly higher than the number of rows. The following distinction between easy and hard instances was made according to the colour coding on the MIPLIB website (easy: green, hard: yellow and red).

4.1 Easy instances

We tried out different core sizes varying from $\delta = 0.3$ to $\delta = 0.7$ and the following fixed run-times: 10, 100, and 1000 seconds. If the approximate core is too small and yields an infeasible problem, we do not display it. The experiments were performed using CPLEX 10.0 on a 3.4 Ghz Intel Pentium D with 4 Gb RAM computer running Linux.

The results of our experiments are shown in Table 2. We display objective values and run-times if CPLEX found an optimal solution to the problem at hand before the time limit. An "∞" entry indicates that no solution was found in the given tun-time, whereas an "=" entry means that the result is identical to its corresponding entry with a lesser maximum run-time. A bold font indicates the best result for the given run-time, an italic font indicates that the reduced model yielded a better or equal solution compared to the original model.

In most of the cases where the original problem could not be solved to optimality within the limited run-time, the approximate core approach provides a way of obtaining better or equally good solutions.

An interesting aside: the CPLEX solver is sometimes able to devise a better solution if given a shorter run-time (see instance `set1ch`) even if run on an identical problem.

4.2 Hard instances

We tried out different core sizes varying from $\delta = 0.2$ to $\delta = 0.8$ and the following fixed run-times: 10, 100, 1000, and 3600 seconds. If the approximate core is too small and yields an infeasible problem, we do not display it. The experiments were performed using CPLEX 10.0 on a 3.4 Ghz Intel Pentium D computer with 4 Gb RAM. We also include the primal bounds achieved by the solver SCIP after one hour on a similar machine (as published in [3]).

The results of our experiments are shown in Table 3. We display the obtained objective values. A bold font indicates the best result for the given run-time, an italic font indicates that the reduced model yielded a better or equal result compared to the original model.

[1] http://miplib.zib.de/

Table 2 Results for solving approximate cores on selected easy instances of the MIPLIB 2003 benchmark set.

		max 5 sec.		max 10 sec.		max 100 sec.	
Name	δ	Obj. Val.	Time	Obj. Val.	Time	Obj. Val.	Time
	0.4	**928.0**	3.2	*928*	3.2	928	3.2
`10teams`	0.5	∞	5.0	**924**	7.6	**924**	7.6
	0.6	*936.0*	5.0	*936*	10.0	**924**	11.1
	0.7	∞	5.0	∞	10.0	**924**	39.0
	orig	∞	5.0	∞	10.0	**924**	72.0
	0.3	1260	1.0	1260	1.1	1260	1.1
	0.4	1260	2.0	1260	2.0	1260	1.9
`aflow30a`	0.5	1205	4.1	1205	4.1	1205	4.1
	0.6	1169	5.0	1160	6.0	1160	6.0
	0.7	1171	5.0	1160	10.0	1159	10.3
	orig	**1160**	5.0	**1158**	10.0	**1158**	22.9
	0.3	∞	5.0	*56167*	10.0	**56137**	15.0
	0.4	57507	5.0	56297	10.0	**56137**	16.2
`air04`	0.5	**56138**	5.0	**56137**	10.0	**56137**	10.9
	0.6	60348	5.0	*56138*	10.0	**56137**	20.5
	0.7	*56172*	5.0	*56172*	10.0	**56137**	32.3
	orig	56287	5.0	56287	10.0	**56137**	21.2
	0.3	26600	5.0	*26458*	10.0	**26374**	12.0
	0.4	*26487*	5.0	*26468*	10.0	**26374**	16.3
`air05`	0.5	26648	5.0	26638	10.0	**26374**	15.1
	0.6	*26450*	5.0	**26374**	9.7	**26374**	9.8
	0.7	**26442**	5.0	**26374**	10.0	**26374**	10.7
	orig	26529	5.0	26469	10.0	**26374**	13.7
	0.3	**-2451377**	5.0	**-2451377**	9.8	**-2451377**	9.7
	0.4	*-2451373*	5.0	**-2451377**	10.0	**-2451377**	16.0
`cap6000`	0.5	**-2451377**	5.0	**-2451377**	10.0	**-2451377**	12.2
	0.6	*-2451373*	5.0	**-2451377**	10.0	**-2451377**	15.5
	0.7	**-2451377**	5.0	**-2451377**	10.0	**-2451377**	12.4
	orig	-2451373	5.0	**-2451377**	10.0	**-2451377**	11.2
	0.3	**181**	5.0	**177**	10.0	*176*	100.1
	0.4	*188*	5.3	**177**	10.4	**175**	100.1
`fast0507`	0.5	∞	5.1	*181*	10.6	*177*	100.1
	0.6	∞	5.1	*184*	10.9	*176*	100.0
	0.7	∞	5.1	*183*	10.4	*176*	100.2
	orig	∞	5.1	196	10.1	177	100.2
	0.3	410974.06	0.0	=	=	=	=
	0.4	410974.06	0.0	=	=	=	=
`fiber`	0.5	410974.06	0.0	=	=	=	=
	0.6	410974.06	0.1	=	=	=	=
	0.7	410974.06	0.2	=	=	=	=
	orig	**405935.18**	0.3	=	=	=	=
	0.3	5073	0.1	=	=	=	=
	0.4	4653	0.1	=	=	=	=
`fixnet`	0.5	4247	0.2	=	=	=	=
	0.6	4047	0.7	=	=	=	=
	0.7	4047	0.8	=	=	=	=
	orig	**3983**	1.2	=	=	=	=
	0.3	18	1.8	18	1.8	18	1.8
	0.4	16	5.0	16	8.3	16	8.3
`pk1`	0.5	**13**	5.0	12	10.0	12	30.4
	0.6	*15*	5.1	**11**	10.0	**11**	81.1
	0.7	*15*	5.1	**11**	10.0	**11**	81.4
	orig	15	5.0	**11**	10.0	**11**	81.6
`set1ch`	0.5	**54585.25**	5.0	**54590.50**	10.0	54541.25	100.0
	0.6	*54703.00*	5.0	*54626.25*	10.0	**54537.75**	100.0
	0.7	*54673.75*	5.0	**54590.25**	10.0	**54537.75**	100.0
	orig	54703.00	5.0	54626.25	10.0	**54537.75**	100.1

Table 3 Results for solving approximate cores on selected instances of the MIPLIB 2003 benchmark set.

Name	δ	10 sec.	100 sec.	1000 sec.	3600 sec.
aflow40b	0.2	1423.0	1423.0	1423.0	1423.0
	0.3	1413.0	1349.0	1349.0	1349.0
	0.4	1363.0	1322.0	1322.0	1322.0
	0.5	*1290.0*	1272.0	1258.0	1258.0
	0.6	*1290.0*	1236.0	1236.0	1236.0
	0.7	**1209.0**	1209.0	1209.0	1209.0
	0.8	1223.0	1206.0	1197.0	1196
	orig	1324.0	**1201.0**	**1181.0**	**1168.0**
	SCIP				1215.0
dano3mip	0.5	∞	**738.93**	720.14	**703.41**
	0.6	∞	*746.39*	**699.56**	**703.41**
	0.7	∞	*746.39*	**699.56**	**703.41**
	0.8	∞	*746.39*	704.31	**703.41**
	orig	∞	746.39	705.63	**703.41**
	SCIP				743.46
markshare1	0.2	69.0	69.0	69.0	69.0
	0.3	33.0	30.0	29.0	29.0
	0.4	**30.0**	**14.0**	11.0	*7.0*
	0.5	*32.0*	*20.0*	12.0	*9.0*
	0.6	*35.0*	*22.0*	**10.0**	*9.0*
	0.7	*35.0*	*22.0*	**10.0**	*9.0*
	0.8	*35.0*	*22.0*	**10.0**	*9.0*
	orig	35.0	22.0	**10.0**	9.0
	SCIP				**5.0**
markshare2	0.2	94.0	85.0	85.0	85.0
	0.3	69.0	36.0	29.0	29.0
	0.4	**60.0**	34.0	**21.0**	*16.0*
	0.5	73.0	**28.0**	*23.0*	**10.0**
	0.6	**60.0**	*33.0*	28.0	*20.0*
	0.7	**60.0**	*33.0*	28.0	*20.0*
	0.8	**60.0**	*33.0*	28.0	*20.0*
	orig	**60.0**	33.0	27.0	20.0
	SCIP				11.0
mkc	0.5	-535.75	-551.67	-553.65	-553.85
	0.6	**-554.71**	**-560.55**	-563.21	-563.41
	0.7	**-554.71**	*-560.21*	-563.41	*-563.73*
	0.8	**-554.71**	*-560.41*	-563.21	**-563.85**
	orig	**-554.71**	-560.21	**-563.61**	-563.73
	SCIP				-546.41
nsrand-ipx	0.2	**51520.0**	**51520.0**	**51360.0**	*51360.0*
	0.3	*51840.0*	**51520.0**	**51360.0**	*51360.0*
	0.4	*52160.0*	*51680.0*	**51360.0**	*51360.0*
	0.5	*52160.0*	*51680.0*	*51680.0*	*51360.0*
	0.6	*52160.0*	*51680.0*	**51360.0**	*51360.0*
	0.7	*52160.0*	52160.0	51680.0	51680.0
	0.8	53600.0	51200.0	51200.0	**51200.0**
	orig	53440.0	52160.0	51680.0	51360.0
	SCIP				55040.0
sp97ar	0.2	*678964760*	672730910	665182266	*662296717*
	0.3	**675605928**	*670695342*	665165087	*662538478*
	0.4	*678237748*	673707633	666699706	665127053
	0.5	*696003683*	*667312813*	*664324321*	*663792268*
	0.6	*683370608*	**666991617**	*664367677*	**661210152**
	0.7	694707937	672961265	**664167450**	664035399
	0.8	695181453	*670177624*	665347989	*66250204*
	orig	699474892	672595081	664495514	663834078
	SCIP				700910273
swath	0.5	**475.67**	**502.78**	**475.51**	*469.39*
	0.6	884.73	523.81	*503.21*	**467.41**
	0.7	1090.25	*503.67*	*492.79*	*472.16*
	0.8	800.87	539.71	*496.43*	*485.01*
	orig	788.03	522.61	515.67	508.93
	SCIP				504.00

Analysing the experiments in more detail we can observe that the overall best results are achieved with $\delta = 0.5$ for the observed test instances: they are better than the original model in 20 cases, worse for 10 cases, and identical in 1 case. We did not count the cases were no solution was found by either model.

In [10] recent results on solving MIPLIB 2003 instances using XPRESS-MP are reported. For problem `sp97ar` we were able to find a new best known solution within one hour of run time. The best known solution reported in [10] is 661670441.4 whereas we found a solution with objective 661210151.84. For the `swath` instance the proven optimal solution was also found by our core-based approach within an hour.

5 Conclusion

We presented an extension of the core concept, previously successfully used for solving different knapsack problems, for mixed binary programs. We provided a variable ordering based on an extended efficiency measure based on dual variable values. This ordering is used to compute approximate cores, which are then solved using the commercial mixed integer programming solver CPLEX. Our computational experiments show that our approach allows us to find better solutions in fixed run-times for several hard instances of the widely used MIPLIB 2003 benchmark set. We were even able to find a new best known solution to one of those instances.

Acknowledgements NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. Achterberg, T., Berthold, T.: Improving the Feasibility Pump. Discrete Optimization **4**(1), 77–86 (2007)
2. Balas, E., Zemel, E.: An algorithm for large zero-one knapsack problems. Operations Research **28**(5), 1130–1154 (1980)
3. Berthold, T.: Primal Heuristics for Mixed Integer Programs. Master's thesis, Technische Universität Berlin (2006)
4. Bertsimas, D., Tsitsiklis, J.N.: Introduction to Linear Optimization. Athena Scientific (1997)
5. Danna, E., Rothberg, E., Le Pape, C.: Exploring relaxation induced neighborhoods to improve MIP solutions. Mathematical Programming **102**, 71–90 (2005)
6. Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Mathematical Programming **104**(1), 91–104 (2005)
7. Fischetti, M., Lodi, A.: Local Branching. Mathematical Programming **98**, 23–47 (2003)
8. Gomes da Silva, C., Clímaco, J., Figueira, J.: Core problems in bi-criteria 0,1-knapsack: new developments. Tech. Rep. 12/2005, INESC-Coimbra (2005)
9. Huston, S., Puchinger, J., Stuckey, P.J.: The core concept for 0/1 integer programming. In: J. Harland, P. Manyem (eds.) Proceedings of the Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), *CRPIT*, vol. 77, pp. 39–47. ACS (2008)

10. Laundy, R., Perregaard, M., Tavares, G., Tipi, H., Vazacopoulos, A.: Solving hard mixed integer programming problems with Xpress-MP: A MIPLIB 2003 case study. INFORMS Journal on Computing (to appear)
11. Martello, S., Toth, P.: A new algorithm for the 0–1 knapsack problem. Management Science **34**, 633–644 (1988)
12. Pisinger, D.: An expanding-core algorithm for the exact 0-1 knapsack problem. European Journal of Operational Research **87**(1), 175–187 (1995)
13. Pisinger, D.: A minimal algorithm for the 0-1 knapsack problem. Operations Research **45**(5), 758–767 (1997)
14. Pisinger, D.: Core problems in knapsack algorithms. Operations Research **47**(4), 570–575 (1999)
15. Puchinger, J., Raidl, G., Pferschy, U.: The core concept for the multidimensional knapsack problem. In: Evolutionary Computation in Combinatorial Optimization - EvoCOP 2006, *LNCS*, vol. 3906, pp. 195–208. Springer (2006)
16. Puchinger, J., Raidl, G., Pferschy, U.: The multidimensional knapsack problem: Structure and algorithms. Tech. Rep. 006149, National ICT Australia, Melbourne, Australia (2007). Submitted for publication