Generic CP-Supported CMSA for Binary Integer Linear Programs

Christian Blum[1(✉)] and Haroldo Gambini Santos[2]

[1] Artificial Intelligence Research Institute (IIIA-CSIC), Campus of the UAB, Bellaterra, Spain
christian.blum@iiia.csic.es

[2] Department of Computer Science, Universidade Federal de Ouro Preto, Ouro Preto, Brazil
haroldo@ufop.edu.br

Abstract. Construct, Merge, Solve & Adapt (CMSA) is a general hybrid metaheuristic for solving combinatorial optimization problems. At each iteration, CMSA (1) constructs feasible solutions to the tackled problem instance in a probabilistic way and (2) solves a reduced problem instance (if possible) to optimality. The construction of feasible solutions is hereby problem-specific, usually involving a fast greedy heuristic. The goal of this paper is to design a problem-agnostic CMSA variant whose exclusive input is an integer linear program (ILP). In order to reduce the complexity of this task, the current study is restricted to binary ILPs. In addition to a basic problem-agnostic CMSA variant, we also present an extended version that makes use of a constraint propagation engine for constructing solutions. The results show that our technique is able to match the upper bounds of the standalone application of CPLEX in the context of rather easy-to-solve instances, while it generally outperforms the standalone application of CPLEX in the context of hard instances. Moreover, the results indicate that the support of the constraint propagation engine is useful in the context of problems for which finding feasible solutions is rather difficult.

1 Introduction

Construct, Merge, Solve & Adapt (CMSA) [6] is a hybrid metaheuristic that can be applied to any combinatorial optimization problem for which is known a way of generating feasible solutions, and whose subproblems can be solved to optimality by a black-box solver. Moreover, note that CMSA is thought for those problem instances for which the application of the standalone black-box solver is not feasible due to the problem instance size and/or difficulty. The main idea of CMSA is to generate reduced sub-instances of the original problem instances, based on feasible solutions that are constructed at each iteration, and to solve these reduced instances by means of the black-box solver. Obviously, the parameters of CMSA have to be adjusted in order for the size of the

M. J. Blesa Aguilera et al. (Eds.): HM 2019, LNCS 11299, pp. 1–15, 2019.
https://doi.org/10.1007/978-3-030-05983-5_1

reduced sub-instances to be such that the black-box solver can solve them efficiently. CMSA has been applied to several NP-hard combinatorial optimization problems, including minimum common string partition [4,6], the repetition-free longest common subsequence problem [5], and the multi-dimensional knapsack problem [15].

A possible disadvantage of CMSA is the fact that a problem-specific way of probabilistically generating solutions is used in the above-mentioned applications. Therefore, the goal of this paper is to design a CMSA variant that can be easily applied to different combinatorial optimization problems. One way of achieving this goal is the development of a solver for a quite general problem. Combinatorial optimization problems can be conveniently expressed as Integer Linear Programs (ILPs) in the format $\min \boldsymbol{c}^T\boldsymbol{x} : A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \in \mathbb{Z}^n$, where A indicates a constraints matrix, $\boldsymbol{b}$ and $\boldsymbol{c}$ are the cost and right-hand-side vectors, respectively and $\boldsymbol{x}$ is a vector of decision variables whose values are restricted to integral numbers. In this paper we propose a generic CMSA for binary integer programs (BIPs), that are obtained when $\boldsymbol{x} \in \{0,1\}^n$. This type of problem is generic enough to model a wide range of combinatorial optimization problems, from the classical traveling salesman problem [2] to protein threading problems [19] and a myriad of applications listed in the MIPLIB 2010 collection of problem instances [13]. As CMSA is an algorithm that makes use of a solution construction mechanism at each iteration, one of the challenges that we address in this paper is the fast production of feasible solutions for general BIPs. For this purpose we support the proposed generic CMSA with a constraint propagation (CP) tool for increasing the probability to generate feasible solutions.

This paper is organized as follows. The next section discusses related work. In Sect. 3, the original version of CMSA is presented, which assumes that the type of the tackled problem is known. The generic CMSA proposal for general BIPs is described in Sect. 4. Finally, an extensive experimental evaluation is presented in Sect. 5 and conclusions as well as an outlook to future work are provided in Sect. 6.

2 Related Work

The development of fast, reliable general purpose combinatorial optimization solvers is a topic that occupies operations research practitioners since many years. The main reason being that the structure of real world optimization problems usually does not remain fixed over time: constraints usually change over time and solvers optimized for a very particular problem structure may lose their efficiency in this process. Thus, a remarkable interest in integer linear programming (ILP) software packages exists, with several commercially successful products such as IBM CPLEX, Gurobi and XPRESS. This success can be attributed to the continuous improvements concerning the performance of these solvers [11,12] and the availability of high level languages such AMPL [10]. The application of these solvers to instances with very different structures creates many challenges. From a practical point of view, the most important one is, possibly, the ability of quickly providing high quality feasible solutions: even though

a complete search is executed, it is quite often the case that time limits need to be imposed and a truncated search is performed. Thus, several methods have been proposed to try to produce feasible solutions in the first steps of the search process. One of the best known approaches is the so-called *feasibility pump* [8,9].

In the context of metaheuristics, Kochenberger et al. [14] developed a general solver for unconstrained binary quadratic programming (UBQP) problems. A whole range of important combinatorial optimization problems such as set partitioning and k-coloring can be easily modeled as special cases of the UBQP problem. Experiments showed that their general solver was able to produce high quality solutions much faster than the general purpose ILP solver CPLEX for hard problems such as the set partitioning problem. Brito and Santos [18] proposed a local search approach for solving BIPs, obtaining some encouraging results when comparing to the COIN-OR CBC Branch-and-Cut solver. In the context of constraint programming, Benoist et al. [3] proposed a fast heuristic solver (LocalSolver) based on local search. Experiments showed that LocalSolver outperformed several other solvers, especially for what concerns executions with very restricted computation times. In this paper we propose a CMSA solver for solving BIPs. This format is more restricted than the LocalSolver format, where non-linear functions can be used, but much more general than the UBQP, which can be easily modeled as a special case of binary programming. One advantage of BIPs is that several high performance solvers can be used to solve the sub-problems generated within CMSA, a feature that will be explored in the next sections.

3 Original CMSA in the Context of BIPs

As already mentioned, in this work we focus on solving BIPs. Any BIP can be expressed in the following way:

$$\min\{\boldsymbol{c}^T\boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, x_j \in \{0,1\}\ \forall j = 1,\ldots,n\} \tag{1}$$

where A is an $m \times n$ matrix, $\boldsymbol{b}$ is the right-hand-size vector of size m, $\boldsymbol{c}$ is a cost vector, and $\boldsymbol{x}$ is the vector of n binary decision variables. Note that m is the number of constraints of this BIP.

In the following we describe the original CMSA algorithm from [6]. However, instead of providing a general description as in [6], our description is already tailored for the application to BIPs. In order to clarify this fact, the algorithm described below is labeled CMSA-BIP. In general, the main idea of CMSA algorithms is to take profit from an efficient complete solver even in the context of problem instances that are too large to be solved directly. The general idea of CMSA is as follows. At each iteration, CMSA probabilistically generates solutions to the tackled problem instance. Next, the solution components that are found in these solutions are added to a sub-instance of the original problem instance. Subsequently, an exact solver is used to solve the sub-instance (if possible in the given time) to optimality.[1] Moreover, the algorithm is equipped with

[1] In the context of problems that can be modelled as BIPs, any black-box ILP solver such as, for example, CPLEX can be used for this purpose.

a mechanism for deleting seemingly useless solution components from the sub-instance. This is done such that the sub-instance has a moderate size and can be solved rather quickly to optimality.

In the context of CMSA-BIP, any combination of a variable x_j with one of its values $v \in \{0, 1\}$ is a solution component denoted by (x_j, v). Given a BIP instance, the complete set of solution components if denoted by C. Any sub-instance of the given BIP is a subset C' of C, that is, $C' \subseteq C$. Such a sub-instance $C' \subseteq C$ is feasible, if C' contains for each variable x_j $(j = 1, \ldots, n)$ at least one solution component (x_j, v), that is, either $(x_j, 0)$, or $(x_j, 1)$, or both. Moreover, a solution to the given BIP is any binary vector $\boldsymbol{s}$ that fulfills the constraints from Eq. (1). Note that a feasible solution $\boldsymbol{s}$ contains n solution components: $\{(x_j, s_j) \mid j = 1, \ldots, n\}$.

The pseudo-code of the CMSA-BIP algorithm is given in Algorithm 1. At each iteration the following is done. First, the best-so-far solution $\boldsymbol{s}^{\text{bsf}}$ is initialized to NULL, indicating that no such solution exists yet. Moreover, sub-instance C' is initialized to the empty set. Note, also, that each solution component $(x_j, v) \in C$ has a so-called age value denoted by $age[(x_j, v)]$. All these age values are initialized to zero at the start of the algorithm. Then, at each iteration, n_{a} solutions are probabilistically generated in function ProbabilisticSolutionGeneration(C); see line 6 of Algorithm 1. As mentioned above, problem-specific heuristics are generally used for this purpose. The solution components found in the constructed solutions are then added to C'. Next, an ILP solver is applied in function ApplyILPSolver(C') to find a possibly optimal solution $\boldsymbol{s}'_{\text{opt}}$ to the restricted problem instance C' (see below for a more detailed description). Note that NULL is returned in case the ILP solver cannot find any feasible solution. If $\boldsymbol{s}'_{\text{opt}}$ is better than the current best-so-far solution $\boldsymbol{s}^{\text{bsf}}$, solution $\boldsymbol{s}'_{\text{opt}}$ is taken as the new best-so-far solution. Next, sub-instance C' is adapted on the basis of solution $\boldsymbol{s}'_{\text{opt}}$ in conjunction with the age values of the solution components; see function Adapt(C', $\boldsymbol{s}'_{\text{opt}}$, age_{max}) in line 14. This is done as follows. First, the age of all solution components from C' that are not in $\boldsymbol{s}'_{\text{opt}}$ is incremented. Moreover, the age of each solution component from $\boldsymbol{s}'_{\text{opt}}$ is re-initialized to zero. Subsequently, those solution components from C' with an age value greater than age_{max}—which is a parameter of the algorithm—are removed from C'. This causes that solution components that never appear in solutions derived by the ILP solver do not slow down the solver in subsequent iterations. On the other side, components which appear in the solutions returned by the ILP solver should be maintained in C'.

Finally, the BIP that is solved at each iteration in function ApplyILPSolver(C') is generated by adding the following constraints to the original BIP. For each $j = 1, \ldots, n$ the following is done. If C' only contains solution component $(x_j, 0)$, the additional constraint $x_j = 0$ is added to the original BIP. Otherwise, if C' only contains solution component $(x_j, 1)$, the additional constraint $x_j = 1$ is added to the original BIP. Nothing is added to the original BIP in case C' contains both solution components. Note that the ILP solver is applied

Algorithm 1. CMSA-BIP: CMSA for solving BIPs

given: a BIP instance, and values for the algorithm parameters
$\boldsymbol{s}^{\text{bsf}} :=$ NULL; $C' := \emptyset$
$age[(x_j, v)] := 0$ for all $(x_j, v) \in C$
while CPU time limit not reached **do**
  for $i = 1, \ldots, n_{\text{a}}$ **do**
    $\boldsymbol{s} :=$ ProbabilisticSolutionGeneration(C)
    for $j = 1, \ldots, n$ **do**
      if $(x_j, s_j) \notin C'$ **then**
        $age[(x_j, s_j)] := 0$
        $C' := C' \cup \{(x_j, s_j)\}$
      end if
    end for
  end for
  $\boldsymbol{s}'_{\text{opt}} :=$ ApplyILPSolver(C')
  if $\boldsymbol{s}'_{\text{opt}}$ is better than $\boldsymbol{s}^{\text{bsf}}$ **then** $\boldsymbol{s}^{\text{bsf}} := \boldsymbol{s}'_{\text{opt}}$ **end if**
  Adapt(C', $\boldsymbol{s}'_{\text{opt}}$, $age_{\max}$)
end while
return $\boldsymbol{s}^{\text{bsf}}$

with a computation time limit of t^{SUB} CPU seconds, which is a parameter of the algorithm.

4 Generic Way of Generating Solutions for BIPs

In those cases in which the optimization problem modeled by the given BIP is not known, we need a generic way of generating solutions to the given BIP in order to be able to apply the CMSA-BIP algorithm described in the previous section. In the following we first describe a basic solution construction mechanims, and afterwards an alternative mechanism which uses a CP tool for increasing the probability to generate feasible solutions. The first algorithm variant is henceforth denoted as Gen-Cmsa-Bip (standing for generic CMSA-BIP) and the second algorithm variant as Gen/Cp-Cmsa-Bip (standing for generic CMSA-BIP with CP support).

4.1 Basic Solution Construction Mechanism

Before starting with the first CMSA-BIP iteration, a node heuristic of the applied ILP solver might be used in order to obtain a first feasible solution. In our case, we used the node heuristic of CPLEX. If, in this way, a feasible solution can be obtained it is stored in $\boldsymbol{s}^{\text{bsf}}$. Otherwise, $\boldsymbol{s}^{\text{bsf}}$ is set to NULL. If, after this step, $\boldsymbol{s}^{\text{bsf}}$ has value NULL, the LP relaxation of the given BIP is solved. However, in order not to spend too much computation time on this step, a computation time limit of t^{LP} seconds is applied. After this, the possibly optimal solution of the LP relaxation is stored in vector $\boldsymbol{x}^{LP}$. Then, whenever function

ProbabilisticSolutionGeneration(C) is called, the following is done. First, a so-called sampling vector $\boldsymbol{x}^{\text{samp}}$ for sampling new (possibly feasible) solutions by randomized rounding is generated. If $\boldsymbol{s}^{\text{bsf}} \neq \text{NULL}$, $\boldsymbol{x}^{\text{samp}}$ is generated based on $\boldsymbol{s}^{\text{bsf}}$ and a so-called *determinism rate* $0 < d_{\text{rate}} < 0.5$ as follows:

$$x_j^{\text{samp}} = \begin{cases} d_{\text{rate}} & \text{if } s_j^{\text{bsf}} = 0 \\ 1 - d_{\text{rate}} & \text{if } s_j^{\text{bsf}} = 1 \end{cases}$$

for all $j = 1, \ldots, n$. In case $\boldsymbol{s}^{\text{bsf}} = \text{NULL}$, $\boldsymbol{x}_{\text{samp}}$ is—for all $j = 1, \ldots, n$—generated on the basis of $\boldsymbol{x}^{LP}$:

$$x_j^{\text{samp}} = \begin{cases} x_j^{LP} & \text{if } d_{\text{rate}} \leq x_j^{LP} \leq 1 - d_{\text{rate}} \\ d_{\text{rate}} & \text{if } x_j^{LP} < d_{\text{rate}} \\ 1 - d_{\text{rate}} & \text{if } x_j^{LP} > 1 - d_{\text{rate}} \end{cases}$$

After generating $\boldsymbol{x}_{\text{samp}}$, a possibly infeasible binary solutions $\boldsymbol{s}$ is generated from $\boldsymbol{x}_{\text{samp}}$ by randomized rounding. Note that this is done in the order $j = 1, \ldots, n$.

4.2 CP Supported Construction Mechanism

Our algorithm makes use of the Constraint Propagation (CP) engine `cprop` that implements ideas from [1,17].[2] The support of CP is used in the following two ways. First, all constraints are processed and implications derived from the constraint set are detected and the problem is preprocessed to keep those variables fixed throughout the search process. Second, the solution construction mechanism changes in the following way. Instead of deriving values for the variables in the order $j = 1, \ldots, n$, a random order π is chosen for each solution construction. That is, at step j, instead of deriving a value for variable x_j, instead a value for variable $x_{\pi(j)}$ is derived. Then, after deciding for a value for variable $x_{\pi(j)}$, the CP tool checks if this assignment will produce an infeasible solution. If this is the case, variable $x_{\pi(j)}$ is fixed to the alternative value. If, again, the CP tool determines that this setting cannot lead to a feasible solution, the solution construction proceeds as described in Sect. 4.1, that is, finalizing the solution construction without further CP support. Otherwise—that is, if a feasible value can be chosen for the current variable—the CP might indicate possible implications consisting of further variables that, as a consequence, have to be fixed to certain values. All these implications are dealt with, before dealing with the next non-fixed variable according to π.[3]

4.3 An Additional Algorithmic Aspect

Instead of using fixed values for CMSA-BIP parameters d_{rate} and t^{SUB}, we implemented the following scheme. For both parameters we use a lower bound

[2] The used CP tool can be obtained at https://github.com/h-g-s/cprop.

[3] Note that, after fixing a value for $x_{\pi(j)}$, the value of $x_{\pi(j+1)}$ might already be fixed due to one of the implications dealt with earlier.

and an upper bound. At the start of CMSA-BIP, the values of d_{rate} and t^{SUB} are set to the lower bound. Whenever an iteration improves $\boldsymbol{s}^{\text{bsf}}$, the values of d_{rate} and t_{SUB} are set back to their respective lower bounds. Otherwise, the values of d_{rate} and t_{SUB} are increased by a factor determined by substracting the lower bound value from the upper bound value and dividing the result by 5.0. Finally, whenever the value of d_{rate}, respectively t_{SUB}, exceeds its upper bound, it is set back to the lower bound value. This procedure is inspired by variable neighborhood search (VNS) [16].

5 Experimental Evaluation

In the following we present an experimental evaluation of Gen-Cmsa-Bip and Gen/Cp-Cmsa-Bip in comparison to the standalone application of the ILP solver IBM ILOG CPLEX v12.7. Note that the same version of CPLEX was applied within both CMSA variants. Moreover, in all cases CPLEX was executed in one-threaded mode. In order to ensure a fair comparison, CPLEX was executed with two different parameter settings in the standalone mode: the default parameter settings, and with the MIP emphasis parameter set to a value of 4 (which means that the focus of CPLEX is on finding good solutions rather than on proving optimality). The default version of CPLEX is henceforth denoted by Cplex-Opt and the heuristic version of CPLEX by Cplex-Heur. All techniques were implemented in ANSI C++ (with the Concert Library of ILOG for implementing everything related to the ILP models), and using GCC 5.4.0 for compiling the software. Moreover, the experimental evaluation was performed on a cluster of PCs with Intel(R) Xeon(R) CPU 5670 CPUs of 12 nuclei of 2933 MHz and at least 40 Gigabytes of RAM.

5.1 Considered Problem Instances

The properties of the 30 selected BIPs are described in Table 1. The first 27 instances are taken from MIPLIB 2010 (http://miplib.zib.de/miplib2010.php), which is one of the best-known libraries for integer linear programming. More specifically, the ILPs on MIPLIB are classified into three hardness categories: *easy*, *hard*, and *open*. From each one of the these categories we chose (more or less randomly) 9 BIPs. In addition, we selected three instances from recent applications found in the literature:

- `mcsp-2000-4` is an instance of the minimum common string partition problem with input strings of length 2000 and an alphabet size of four [6]. The hardness of this instance is due to a massive amount of constraints.
- `rflcs-2048-3n-div-8` is an instance of the repetition-free longest common subsequence problem with two input strings of length 2048 and an alphabet size of 768 [5]. This instance is hard to solve due to the large number of variables.
- `rcjs-20testS6` is an instance of the resource constraint job scheduling problem considered in [7]. Finding feasible solutions for this problem is, for general purpose ILP solvers, rather time consuming.

Table 1. Characteristics of the 30 BIPs that were considered.

BIP instance name	# Cols/Vars	# Rows	Opt. Val.	MIPLIB status
acc-tight5	1339	3052	0.0	Easy
air04	8904	823	56137.0	Easy
cov1075	120	637	20.0	Easy
eilB101	2818	100	1216.92	Easy
ex9	10404	40962	81.0	Easy
netdiversion	129180	119589	242.0	Easy
opm2-z7-s2	2023	31798	−10280.0	Easy
tanglegram1	34759	68342	5182.0	Easy
vpphard	51471	47280	5.0	Easy
ivu52	157591	2116	481.007	Hard
opm2-z12-s14	10800	319508	−64291.0	Hard
p6b	462	5852	−63.0	Hard
protfold	1835	2112	−31.0	Hard
queens-30	900	960	−40.0	Hard
reblock354	3540	19906	−39280521.23	Hard
rmine10	8439	65274	−1913.88	Hard
seymour-disj-10	1209	5108	287.0	Hard
wnq-n100-mw99-14	10000	656900	259.0	Hard
bab1	61152	60680	Unknown	Open
methanosarcina	7930	14604	Unknown	Open
ramos3	2187	2187	Unknown	Open
rmine14	32205	268535	Unknown	Open
rmine25	326599	2953849	Unknown	Open
sts405	405	27270	Unknown	Open
sts729	729	88452	Unknown	Open
t1717	73885	551	Unknown	Open
t1722	36630	338	Unknown	Open
mcsp-2000-4	1335342	4000	Unkonwn	n.a
rflcs-2048-3n-div-8	5461	7480548	Unknown	n.a
rcjs-20testS6	273372	29032	Unknown	n.a

5.2 Parameter Setting

Both generic CMSA variants have the following parameters for which well-working values must be found: (1) the number of solution constructions per iteration (n_a), (2) the maximumm age of solution components ($age_{\max}$), (3) a computation time limit for solving the LP relaxation (t^{LP}), (4) a lower and an upper bound for the determinism rate (d_{rate} (LB) and d_{rate} (UB)), and (5) a

lower and an upper bound for the computation time limit of the ILP solver at each iteration (t^{SUB} (LB) and t^{SUB} (UB)).

Concerning $age_{\max}$, it became clear during preliminary experiments that this parameter has not the same importance for Gen-Cmsa-Bip and Gen/Cp-Cmsa-Bip as it has for a problem-specific CMSA. In other words, while a setting of $n_a = 10$ and $age_{\max} = 3$ is essentially different to a setting of $n_a = 30$ and $age_{\max} = 1$ for a problem-specific CMSA, this is not the case for the generic CMSA variants. This is related to the way of constructing solutions. In a problem-specific CMSA, the greedy heuristic that is used in a probabilistic way biases the search towards a certain area of the search space. This is generally beneficial, but may have the consequence that some solution components that are needed for high-quality solutions have actually a low probability to be included in the constructed solutions. A setting of $age_{\max} > 1$ provides $age_{\max}$ opportunites—that is, applications of the ILP solver—to find high-quality solutions that incorporate such solution components. In contrast, the way of constructing solutions in the generic CMSA variants does not produce this situtaiton. Therefore, we decided for a setting of $age_{\max} = 1$ for all further experiments. Apart from $age_{\max}$, after preliminary experiments we also fixed the following parameter values:

- $n_a = 5$
- $t^{LP} = 10.0$
- The lower bound of t^{SUB} is set to 30.0 and the upper bound to 100.0

The parameter that has the strongest impact on the performance of the generic CMSA variants is d_{rate}. We noticed that both generic CMSA variants are quite sensitive to the setting of the lower and the upper bound for this parameter. However, in order to avoid a fine-tuning for each single problem instance, we decided to identify four representative parameter value configurations in order to cover the characteristics of the 30 selected problem instances. Both generic CMSA variants are then applied with all four parameter configurations to all 30 problem instances. For each problem instance we take the result of the respective best configuration as the final result (and we indicate with which configuration this result was obtained). The four utilized parameter configurations are described in Table 2.

Table 2. The four parameter configurations used for both Gen-Cmsa-Bip and Gen/Cp-Cmsa-Bip.

Parameter configuration	d_{rate} (LB)	d_{rate} (UB)
Configuration 1	0.03	0.08
Configuration 2	0.05	0.15
Configuration 3	0.1	0.3
Configuration 4	0.3	0.5

5.3 Results

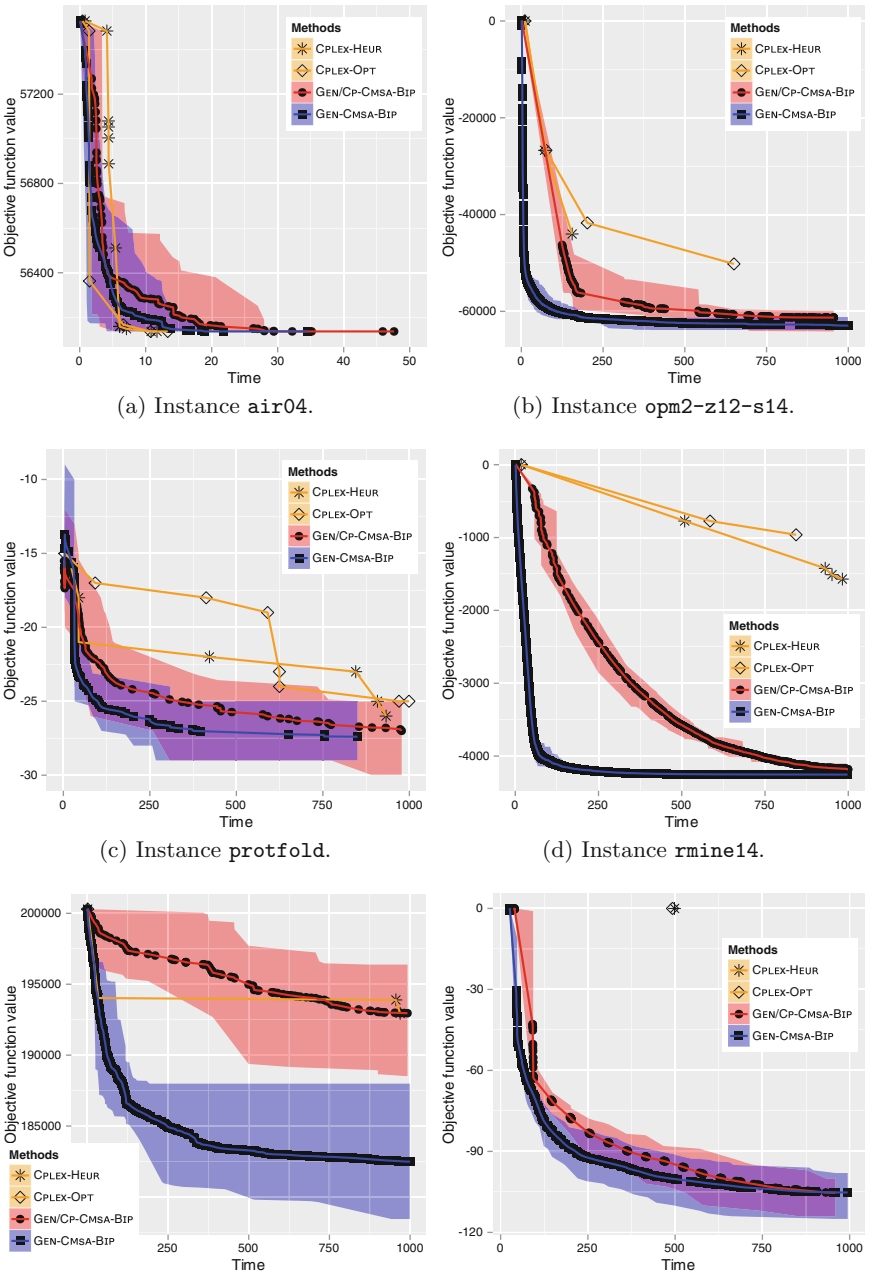
All four approaches (Gen-Cmsa-Bip, Gen/Cp-Cmsa-Bip, Cplex-Opt, and Cplex-Heur) were applied with a computation time limit of 1000 CPU seconds to each one of the 30 problem instances. However, as Gen-Cmsa-Bip and Gen/Cp-Cmsa-Bip are stochastic algorithms, they are applied 10 times to each instance, while the two CPLEX variants are applied exactly once to each instance. The numerical results are provided in Table 3, which has the following structure. The first column contains the problem instance name, and the second column provides the value of an optimal solution (if known).[4] The results of Gen-Cmsa-Bip and Gen/Cp-Cmsa-Bip are presented in three columns for each algorithm. The first column (with heading 'Best') contains the best result obtained over 10 runs, the second column (with heading 'Avg.') shows the average of the best results obtained in each of the 10 runs, and the third column indicates the configuration (out of 4) that has produced the corresponding results. Finally, the results of Cplex-Opt and Cplex-Heur are both presented in two columns. The first column shows the value of the best feasible solution produced within the allowed computation time, and the second column shows the best gap (in percent) at the end of each run. Note that a gap of 0.0 indicates that optimality was proven.
The following observations can be made:

- Concerning the BIPs classified as *easy* (see the first nine table rows), it can be noticed that Cplex-Heur always generates an optimal solution, even though optimality can not be proven in two cases. Gen/Cp-Cmsa-Bip–that is, the generic CP-supported CMSA variant—also produces an optimal solution in at least one out of 10 runs for all nine problem instances. However, in three cases, the algorithm fails to produce an optimal solution in all 10 runs per instance. The results of the basic generic CMSA variant (Gen-Cmsa-Bip) are quite similar. However, for instance `ex9` it is not able to produce any feasible solution, and for instance `netdiversion` the results are clearly inferior to those of Gen/Cp-Cmsa-Bip. Nevertheless, the support of CP also comes with a cost. This can be seen when looking at the anytime behaviour of the algorithms as shown for six exemplary cases in Fig. 1. In particular, Gen/Cp-Cmsa-Bip is often not converging as fast to good solutions as Gen-Cmsa-Bip.
- The increased difficulty of the instances labelled as *hard* (see table rows 10-18), produces more differences between the four approaches. In fact, sometimes one of the CPLEX variants is clearly better than the two CMSA versions (see, for example, instance `ivu52`), and sometimes the generic CMSA variants outperform the CPLEX versions (such as, for example, for instance `opm2-z12-s14`). As the CP-support is more costly for these instances, the results of Gen-Cmsa-Bip are generally a bit better than those of Gen/Cp-Cmsa-Bip. The effect of the increased cost of the CP support can also nicely

[4] Note that all considered BIPs are in standard form, that is, they must be minimized.

Table 3. Numerical results for the 30 considered BIPs.

Instance	Opt. Val.	Gen-Cmsa-Bip			Gen/Cp-Cmsa-Bip			Cplex-Opt		Cplex-Heur	
		Best	**Avg.**	**Conf.**	**Best**	**Avg.**	**Conf.**	**Value**	**Gap**	**Value**	**Gap**
acc-tight5	0.0	0.0	**0.0**	4	0.0	**0.0**	4	**0.0**	0.0	**0.0**	0.0
air04	56137.0	56137.0	**56137.0**	4	56137.0	**56137.0**	4	**56137.0**	0.0	**56137.0**	0.0
cov1075	20.0	20.0	**20.0**	3	20.0	**20.0**	4	**20.0**	0.0	**20.0**	0.0
eilB101	1216.92	1216.92	**1216.92**	4	1216.92	**1216.92**	3	**1216.92**	0.0	**1216.92**	0.0
ex9	81.0	– –	– –	– –	81.0	**81.0**	2	**81.0**	0.0	**81.0**	0.0
netdiversion	242.0	308.0	386.3	3	242.0	288.0	4	276.0	15.5	**242.0**	0.0
opm2-z7-s2	-10280.0	-10280.0	**-10280.0**	4	-10280.0	**-10280.0**	3	**-10280.0**	0.0	**-10280.0**	0.0
tanglegram1	5182.0	5182.0	**5182.0**	4	5182.0	**5182.0**	4	**5182.0**	0.0	**5182.0**	0.0
vpphard	5.0	5.0	**5.0**	3	5.0	5.5	3	**5.0**	0.0	**5.0**	100.0
ivu52	481.007	657.6	884.23	3	16860.0	16860.0	3	647.05	25.77	**490.09**	2.0
opm2-z12-s14	-64291.0	-63848.0	**-62941.5**	3	-64149.0	-61286.2	3	-50200.0	78.62	-44013.0	106.52
p6b	-63.0	-63.0	-62.6	3	-63.0	-61.9	4	-62.0	9.38	**-63.0**	8.08
protfold	-31.0	-29.0	**-27.4**	3	-30.0	-27.0	3	-25.0	43.53	-26.0	54.82
queens-30	-40.0	-39.0	**-38.8**	3	-38.0	-37.7	3	-38.0	83.0	-38.0	83.56
reblock354	-39280521.23	-39261319.4	-39246477.9	4	-39271317.72	-39262125.07	4	**-39277743.28**	0.27	-39259518.82	0.53
rmine10	-1913.88	-1911.39	-1909.05	4	-1911.75	**-1910.72**	3	-1909.29	0.54	-1910.49	0.75
seymour-disj-10	287.0	287.0	**287.5**	3	287.0	287.6	4	288.0	2.08	288.0	2.17
wnq-n100-mw99-14	259.0	259.0	268.2	3	268.0	272.6	4	275.0	13.98	**267.0**	11.92
bab1	?	-218764.89	**-218764.89**	3	-218764.89	**-218764.89**	4	**-218764.89**	1.98	**-218764.89**	3.13
methanosarcina	?	2730.0	**2735.1**	3	2841.0	2918.2	2	3080.0	56.79	2772.0	56.02
ramos3	?	233.0	236.6	1	232.0	**235.3**	1	248.0	40.99	263.0	44.38
rmine14	?	-4266.31	**-4254.25**	3	-4210.99	-4182.77	3	-962.12	347.84	-1570.21	174.53
rmine25	?	-10297.66	**-8836.25**	3	-1790.65	-1130.70	4	0.0	inf	0.0	inf
sts405	?	342.0	**344.0**	2	343.0	346.2	1	349.0	50.38	348.0	60.06
sts729	?	646.0	**647.90**	1	650.0	651.4	1	661.0	61.25	729.0	65.65
t1717	?	178443.0	**182489.3**	2	188527.0	192958.1	1	200300.0	32.18	192942.0	29.92
t1722	?	120946.0	124910.0	4	119478.0	126627.0	3	119086.0	14.81	**118608.0**	15.24
mcsp-2000-4	?	514.0	**519.5**	2	527.0	527.0	3	527.0	100.0	527.0	100.0
rflcs-2048-3n-div-8	?	-115.0	-105.2	1	-114.0	**-105.3**	4	0.0	inf	0.0	inf
rcjs-20testS6	?	– –	– –	– –	11555.9	**15279.3**	1	20692.4	68.9	– –	– –



(a) Instance `air04`.

(b) Instance `opm2-z12-s14`.

(c) Instance `protfold`.

(d) Instance `rmine14`.

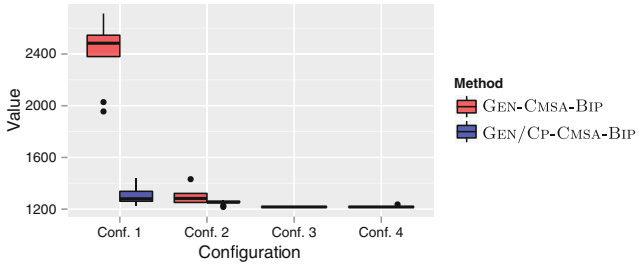
(e) Instance `t1717`.

(f) Instance `rflcs-2048-3n-div-8`.

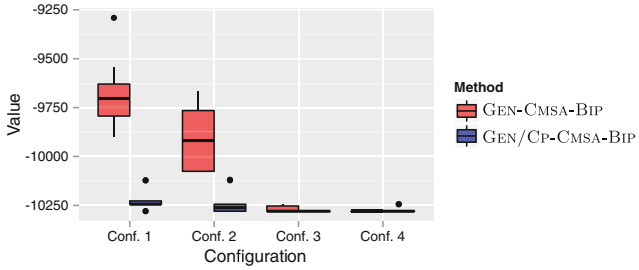
Fig. 1. Anytime performance of GEN-CMSA-BIP and GEN/CP-CMSA-BIP in comparison to the two CPLEX variants. The performance of the two generic CMSA versions is shown via the mean performance together with the confidence ribbon (based on 10 independent runs).

be observed in the anytime behaviour of the algorithms for two hard instances in Fig. 1c and b.

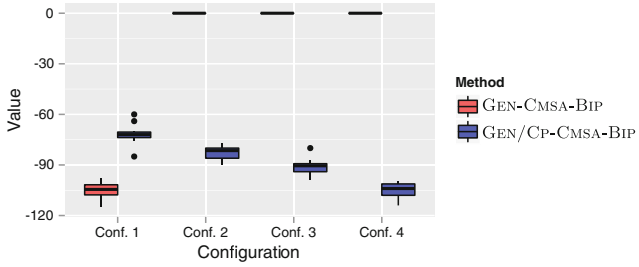
– In the context of the nine *open* instances, the generic CMSA variants clearly outperform the standalone application of CPLEX, with the exception of instance `t1722`. The same holds for the three additional, difficult problem instances (last three table rows). Note, especially, that for instance `rcjs-20testS6` the support of CP pays off again, as it is difficult to find feasible solutions for this instance.



(a) Instance `eilB101`.



(b) Instance `opm2-z7-s2`.



(c) Instance `rflcs-2048-3n-div-8`.

Fig. 2. Boxplots showing the 10 results per algorithm configuration for Gen-Cmsa-Bip and Gen/Cp-Cmsa-Bip in the context of three of the considered problem instances.

Summarizing, with increasing problem size/difficulty, the advantage of the generic CMSA variants over the standalone application of CPLEX becomes more and more pronounced. However, as the cost of the CP support also increases with growing problem size, Gen/Cp-Cmsa-Bip is only able to outperform Gen-Cmsa-Bip when finding feasible solutions is really difficult. However, we noticed that the CP support has also an additional effect, which is shown in the graphics of Fig. 2. Each boxplot shows the final results (obtained by 10 runs per instance) of each of the four parameter configurations for both generic CMSA variants. Interestingly, the use of CP during solution construction flattens out the quality differences between the four parameter configurations. This can be seen in all three boxplots. In Fig. 2b, for example, the results of Gen-Cmsa-Bip are good with configurations 3 and 4, while they are significantly worse with configurations 1 and 2. In contrast, the results of Gen/Cp-Cmsa-Bip, while also being best with configurations 3 and 4, are only slightly worse with configurations 1 and 2. In that sense, the CP-support makes the algorithm more robust with respect to the parameter setting.

6 Conclusions

In this work, we developed a problem-agnostic CMSA algorithm for solving binary linear integer programs. The main challenge was on constructing solutions to unknown problems, in a way such that feasibility is quickly reached. For this purpose we developed—in addition to the basic approach—an algorithm variant that makes use of a constraint programming tool. Concerning the results, we were able to observe that the use of the constraint programming tool pays off for those problems for which reaching feasiblity is rather difficult. In general, with growing problem size and/or difficulty, both CMSA variants have an increasing advantage over the standalone appliaction of the ILP solver CPLEX. In a sense, our generic CMSA can be seen—in many cases—as *a better way of making use of a black-box ILP solver*. Concerning future work, we plan to extend our work towards general ILPs. Moreover, we plan to work on a mechanism for automatically adjusting the algorithm parameters at the start of a run.

References

1. Achterberg, T.: Constraint Integer Programming. Ph.D. thesis (2007)
2. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton (2007)
3. Benoist, T., Estellon, B., Gardi, F., Megel, R., Nouioua, K.: LocalSolver 1.x: a black-box local-search solver for 0–1 programming. 4OR **9**(3), 299 (2011)
4. Blum, C.: Construct, merge, solve and adapt: application to unbalanced minimum common string partition. In: Blesa, M.J., Blum, C., Cangelosi, A., Cutello, V., Di Nuovo, A., Pavone, M., Talbi, E.-G. (eds.) HM 2016. LNCS, vol. 9668, pp. 17–31. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39636-1_2

5. Blum, C., Blesa, M.J.: A comprehensive comparison of metaheuristics for the repetition-free longest common subsequence problem. J. Heuristics **24**, 551–579 (2017)
6. Blum, C., Pinacho, P., López-Ibáñez, M., Lozano, J.A.: Construct, merge, solve & adapt: a new general algorithm for combinatorial optimization. Comput. Oper. Res. **68**, 75–88 (2016)
7. Ernst, A.T., Singh, G.: Lagrangian particle swarm optimization for a resource constrained machine scheduling problem. In: 2012 IEEE Congress on Evolutionary Computation, pp. 1–8 (2012)
8. Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Math. Program. **104**(1), 91–104 (2005)
9. Fischetti, M., Salvagnin, D.: Feasibility pump 2.0. Math. Program. Comput. **1**(2), 201–222 (2009)
10. Fourer, R., Gay, D., Kernighan, B.: AMPL, vol. 117. Boyd & Fraser Danvers (1993)
11. Gamrath, G., Koch, T., Martin, A., Miltenberger, M., Weninger, D.: Progress in presolving for mixed integer programming. Math. Program. Comput. **7**, 367–398 (2015)
12. Johnson, E., Nemhauser, G., Savelsbergh, W.: Progress in linear programming-based algorithms for integer programming: an exposition. INFORMS J. Comput. **12**, 2–3 (2000)
13. Koch, T., et al.: Miplib 2010. Math. Program. Comput. **3**(2), 103 (2011)
14. Kochenberger, G., et al.: The unconstrained binary quadratic programming problem: a survey. J. Comb. Optim. **28**(1), 58–81 (2014)
15. Lizárraga, E., Blesa, M.J., Blum, C.: Construct, merge, solve and adapt versus large neighborhood search for solving the multi-dimensional knapsack problem: which one works better when? In: Hu, B., López-Ibáñez, M. (eds.) EvoCOP 2017. LNCS, vol. 10197, pp. 60–74. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55453-2_5
16. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
17. Sandholm, T., Shields, R.: Nogood learning for mixed integer programming. Technical report (2006)
18. Souza Brito, S., Gambini Santos, H., Miranda Santos, B.H.: A local search approach for binary programming: feasibility search. In: Blesa, M.J., Blum, C., Voß, S. (eds.) HM 2014. LNCS, vol. 8457, pp. 45–55. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07644-7_4
19. Xy, J., Li, M., Kim, D., Xu, Y.: RAPTOR: optimal protein threading by linear programming. J. Bioinform. Comput. Biol. **1**(1), 95–117 (2003)