Branch-and-Check: A Hybrid Framework Integrating Mixed Integer Programming and Constraint Logic Programming

Erlendur S. Thorsteinsson

Graduate School of Industrial Administration; Carnegie Mellon University;
Schenley Park; Pittsburgh, PA 15213–3890; U.S.A.
E-mail: esth@erlendur.com

Abstract. We present Branch-and-Check, a hybrid framework integrating Mixed Integer Programming and Constraint Logic Programming, which encapsulates the traditional Benders Decomposition and Branch-and-Bound as special cases. In particular we describe its relation to Benders and the use of nogoods and linear relaxations. We give two examples of how problems can be modelled and solved using Branch-and-Check and present computational results demonstrating more than order-of-magnitude speedup compared to previous approaches. We also mention important future research issues such as hierarchical, dynamic and adjustable linear relaxations.

1 Introduction

The first goal of this paper is to propose a modeller/solver framework, Branch-and-Check, that not only encompasses both the traditional Benders Decomposition and Branch-and-Bound schemes of Mixed Integer Programming (MIP) as special cases of a spectrum of solution methods, but also adds an extra dimension by allowing the integration of Constraint Logic Programming (CLP) in a MIP style branching search.

In this framework we model a problem in a mixture of CLP and MIP. The CLP part of the model then adds a relaxation of itself to the MIP part (or it is added explicitely). If the two parts do not use the same variables then the model should include mapping relations between them (*shadowed* variables). The solution method is then a branching search, solving the LP relaxation of the MIP part at every node and branching on the discrete variables, but only solving the CLP part at the nodes of the branching tree where it is advantageous (or necessary), e.g., based on how difficult/large the MIP part is compared to the CLP part or how easy it is to strengthen the MIP part using the CLP solution and on the quality of those cuts.

The second goal of this paper is to identify one of the key elements for the integration of CLP and MIP that has still not been adequately addressed and to propose it as a pertinent and pressing research topic in the area of integration: *Dynamic* linear relaxations of global constraints. We will present computational results that indicate that for efficient communication between the different parts of a hybrid model some double modelling is required, i.e., the same constraint or parts of the

model must be present in both CLP and MIP form. It is also vital that the different forms of the same constraint communicate (intra-constraint communication). This is what we have previously termed *mixed* propagation of *mixed* CLP–MIP global constraints [20, 21, 24].

This holds regardless of the scheme used, be it (Tight) Cooperation [2, 22], Mixed Logical/Linear Programming (MLLP) [15, 16, 20, 21, 24], Branch-and-Check (see Sec. 3) or some other integration approach [3, 6, 7, 18, 23]. This double modelling could be explicit, but most preferably it should be implicit, i.e., mixed global constraints should post and dynamically update a linear relaxation of themselves, in addition to the classical CLP propagation on their discrete parts and mixed propagation between the discrete and continuous parts.

This extends the idea proposed by Beringer and De Backer in [2]. They argued that the standard CLP architecture is not optimal because cooperation between different solvers is only done by value propagation. In addition, they proposed, solvers, e.g., CLP and MIP solvers, should be able to communicate by exchanging information through variable bounds. Variable bounds are only a special type of linear constraints, so linear relaxations take this idea a step further and open up many more possibilities in CLP–MIP integration.

We will exemplify this based on our experiences when developing the Branch-and-Check framework and also based on our previous line of research [15, 16, 20, 21, 24]. The paper is organised as follows. This section outlined the focus of this research. Section 2 reviews the history of efforts in integrating CLP and MIP along with two classical MIP techniques, Benders Decomposition and Branch-and-Bound. In Sec. 3 we introduce the Branch-and-Check framework, discuss how it generalises Benders and Branch-and-Bound, and show how CLP can be integrated. Section 4 then gives two examples, Scheduling with Dissimilar Parallel Machines and Capacitated Vehicle Routing with Time Windows, and presents computational results demonstrating more than order-of-magnitude speedup compared to previous approaches.

2 Background

2.1 Classical MIP Techniques

Branch-and-Bound: We will assume that the reader is familiar with the classical Branch-and-Bound approach for solving MIPs. Due to different vocabulary in the two fields, however, we would like to note that this is the technique that is sometimes referred to as Branch-and-Relax [3].

Benders Decomposition: Classical Benders Decomposition exploits the fact that in some problems, fixing the values of certain difficult variables simplifies the problem tremendously. By enumerating those difficult variables, solving each resulting *subproblem* and selecting the best subproblem solution found, the original problem can be solved. Benders' method [1] is more ingenious. It solves a *master problem* to assign values to the difficult variables. Each solution to the subproblem then generates a Benders cut that is added to the master problem before resolving it. Thus each solution to the master problem must satisfy all the Benders cuts obtained so

far, avoiding searching similar regions of the solutions space again. This is similar to the role nogoods play in CLP.

Classical Benders Decomposition applies if the problem can be written

$$\begin{array}{ll} \min & cx + fy \\ \text{s.t.} & Ax + Gy \geq b, \\ & x \in D,\ y \in \mathbb{R}^n_+ . \end{array} \tag{1}$$

If x^* denotes the solution to the master problem then the subproblem is an LP,

$$\begin{array}{ll} \min & cx^* + fy \\ \text{s.t.} & Gy \geq b - Ax^*, \\ & y \in \mathbb{R}^n_+ , \end{array} \tag{2}$$

which is easily solved. The procedure is iterative, interleaving solving the master problem to optimality and the resulting subproblem. By applying duality theory to the solution of the subproblem, cuts can be generated that are added to the master problem

$$\begin{array}{lll} \min & z & \\ \text{s.t.} & z \geq u^*_q (b - Ax) + cx, & q \in Q_1, \quad (3) \\ & u^*_q (b - Ax) \leq 0, & q \in Q_2, \quad (4) \\ & x \in D, & \end{array}$$

where u^*_q is the dual solution to the subproblem when the subproblem is feasible in iterations $q \in Q_1$, and infeasible in iterations $q \in Q_2$, before resolving the master problem in the next iteration. A more detailed description of Benders Decomposition can be found in [1, 8, 11, 14, 17].

2.2 Previous Integration Schemes

Properties of a number of different problems were considered by Darby-Dowman and Little in [4, 5] and their effect on the performance of CLP and MIP approaches were presented. They reported experimental results that illustrate some key properties of the techniques: MIP is very efficient for problems with good relaxations, but it suffers when the relaxation is weak or when its restricted modelling framework results in large models. CLP, with its more expressive constraints, has smaller models that are closer to the problem description and behaves well for highly constrained problems, but it lacks the "global perspective" of relaxations.

(Tight) Cooperation: Beringer and De Backer proposed in [2] that CLP and MIP solvers can be coupled together with common (or shadowed) variables in a double modelling framework using two way communication: The MIP solver sends the CLP solver the values of the common variables that are fixed, which relies on the MIP solver being able to detect implied inequalities, and the CLP solver sends the

MIP solver strengthened bounds for the common variables. They compared solving a multi-knapsack problem as a pure CLP or a pure MIP against using the cooperation of the two solvers, and obtained favourable results.

Refalo proposed an extension to this framework in [22], where the MIP model is dynamic; it is restated when variable bounds are tightened or variables are fixed, by the CLP solver.

Mixed Logical/Linear Programming (MLLP): Hooker et al. proposed a new modelling paradigm to efficiently integrate CLP and MIP in [12, 13, 15, 16]. In that framework, constraints are in the form of conditionals that link the discrete and continuous elements of the problem. An MLLP model has the form

$$\begin{aligned} \min \quad & cx \\ \text{s.t.} \quad & h_i(y) \rightarrow A^i x \geq b^i, \quad i \in I, \\ & y \in D, \; x \in \mathbb{R}^n. \end{aligned} \tag{5}$$

The antecedents $h_i(y)$ of the conditionals are constraints that can be treated with CLP techniques. The consequents are linear inequality systems that form an LP relaxation.

An MLLP problem is solved by branching on the discrete variables. The conditionals assign roles to CLP and LP: CLP is applied to the discrete constraints to reduce the search and help determine when partial assignments satisfy the antecedents. At each node of the branching tree an LP solver minimises cx subject to the inequalities $A^i x \geq b^i$ for which $h_i(y)$ is determined to be true. This delayed posting of inequalities leads to small and lean LP problems that can be solved efficiently.

Ottosson, Thorsteinsson and Hooker, and Ottosson and Thorsteinsson extended MLLP in [21, 24] by proposing adding *mixed* global constraints that have both discrete and continuous elements within them. A mixed global constraint has a dynamically stated linear relaxation that becomes a part of the continuous part and propagates information between the discrete and continuous parts of the model. In that framework the mixed global constraints serve both as a modelling tool and a way to exploit structure in the solution process. Mixed global constraints can be written in the form (5) as conditionals, analogous to global constraints in CLP, but improve the solution process by improving the propagation.

Hybrid Decomposition: Jain and Grossmann, and Harjunkoski, Jain and Grossmann presented a scheme in [9, 18] where the problem is decomposed into two sub-parts, one handled by MIP and the other by CLP. This is demonstrated using a multi-machine scheduling problem where the assignment of tasks to machines is modelled as a MIP and the sequencing of the tasks on the assigned machines is handled using CLP. The search scheme is an iterative procedure where the assignment problem is first solved to optimality, identifying which machine to use for each task, and then a CLP feasibility problem is solved trying to sequence according to this assignment. If the sequencing fails, cutting planes are added to the MIP problem to forbid this (and subsumed) assignments and the process is iterated. This approach

has many similarities to Benders and in fact, in [11] it is shown how this problem can be written for Benders.

Other Approaches: Bockmayr and Kasper proposed an interesting framework in [3] for combining CLP and MIP, in which several approaches to integration or synergy are possible, by dividing the constraints for both CLP and MIP into two different categories, primitive and non-primitive. Primitive constraints are those for which there exists a polynomial time solution algorithm and non-primitive constraints are those for which this is not true.

Rodošek et al. presented in [23] a systematic approach for transforming a CLP model into a corresponding MIP model. CLP is then used along with linear relaxations in a single search tree to prune domains and establish bounds. The downside of this approach is that the systematic procedure that creates the shadow MIP model for the original CLP model includes reified arithmetic constraints, big-M constraints. A translation involving numerous big-M constraints may result in a poor MIP model, i.e., with a poor linear relaxation.

3 Branch-and-Check

3.1 Description of the General Method

Branch-and-Check builds to a certain extent on Benders Decomposition. The basic idea is to identify a part of the problem that is *basic* and a part that is *delayed*. The solution process is a branching search on the basic part where the delayed part is checked (e.g., for feasibility) as late or as seldom as possible. The rationale is that while the delayed part is necessary to check for the correctness of the solution, it may be large and computationally expensive to include in every step of the calculations and thus we want to delay looking at it as long as possible. We are going to refer to the basic part as the master problem and the delayed part as the subproblem.

This strategy can be applied to a problem of this general form:

$$\begin{aligned} \min \quad & cx + f(y) && (6)\\ \text{s.t.} \quad & Ax \leq b, && (7)\\ & H(x, y), && (8) \end{aligned}$$

where the problem is naturally split into a mixed integer linear part (7) and a non-linear part (8), e.g., (mixed) global constraints such as the `piecewise-linear` and `alldifferent` constraints. The constraints of master problems are in the top part and the constraints of the subproblems are in the lower part. The non-linear part can also include linear constraints or mappings between the x and y variables. Thus the following would be examples of problem forms this strategy can be applied to:

$$\begin{array}{lll} \min & cx + f(y) & \\ \text{s.t.} & Ax \leq b, & \\ & H(x, y), & \end{array} \qquad \begin{array}{lll} \min & cx + dy & \\ \text{s.t.} & A^1 x \leq b^1, & \\ & A^2 y \leq b^2, & \end{array} \qquad \begin{array}{lll} \min & cx + f(y) & \\ \text{s.t.} & Ax \leq b, & \\ & x \sim y, & \\ & F(y). & \end{array}$$

In the third form, $x \sim y$ represents that there is a mapping between the values of the variables x and y, e.g., a one-to-one mapping between two variables or between a variable and a set of variables such as $y \in \{1, \ldots, n\}$, $x_1, \ldots, x_n \in \{0, 1\}$ and $x_y = 1$. This second mapping is a common mapping between CLP and MIP.

Since the master problem is a relaxation of the original problem, when a solution to the master problem is found in the branching search it is not guaranteed that the solution is truly feasible nor that the objective value is correct. At those nodes in the branching tree, i.e., where all the variables in the master problem have been instantiated and the branching search is about to fathom the subtree, we solve the subproblem as well to determine if the overall solution is feasible and then what its correct objective function value is. We can solve the subproblem more often, but how often to consult the subproblem is a matter of how large or computationally expensive the subproblem is compared to the master problem.

Completely ignoring the subproblem for most of the solution process, only solving it at selected nodes, is not going to work, however, so we augment the master problem with a relaxation of the subproblem: A simpler and computationally less expensive representation of the subproblem that focuses the master problem on good candidate solutions with respect to the subproblem. For example, for the third form, the master problem would become

$$\begin{array}{ll} \min & cx + \mathcal{L}_{f(y)}(x) \\ \text{s.t.} & Ax \leq b, \\ & \mathcal{L}_{F(y)}(x). \end{array}$$

The relaxation should be hierarchical if possible, e.g., if the subproblem is a CLP then the whole relaxation should preferably be the union of the relaxations of the individual global constraints that comprise the subproblem. It should also be dynamic, i.e., as the solution process progresses it should be updated, e.g., when variables are fixed; and adjustable, i.e, it should be possible to efficiently make incremental changes, rather than have to recompute it at every node.

Whenever the subproblem is solved, cuts are added to the master problem. We add a lower bounding cut if the subproblem is feasible, bounding the objective function from below, or an infeasibility cut (a *nogood*) if the subproblem is infeasible, disallowing this solution and others similar to it. For example, for the third form, the master problem would become

$$\begin{array}{ll} \min & cx + z \\ \text{s.t.} & Ax \leq b, \\ & \mathcal{L}_{F(y)}(x), \\ & z \geq \mathcal{L}_{f(y)}(x), \\ & z \geq L(x), \\ & N(x). \end{array}$$

The subproblem in this case will be

$$\begin{array}{ll} \min & f(y) \\ \text{s.t.} & F(y), \end{array}$$

given the mapping $x \sim y$ between the variables in the master and subproblems, i.e., some of the variables in the subproblem may be fixed or have restricted values

based on the current solution of the master problem. In the examples we will look at in Sec. 4, the solution to the master problem will determine how the subproblem decomposes.

The master problem for the general form (6)–(8) is

$$\begin{aligned} \min \quad & cx + z && (9)\\ \text{s.t.} \quad & Ax \leq b, && (10)\\ & \mathcal{L}_{H(x,y)}(x), && (11)\\ & z \geq \mathcal{L}_{f(y)}(x), && (12)\\ & z \geq L(x), && (13)\\ & N(x), && (14) \end{aligned}$$

and the corresponding subproblem is

$$\begin{aligned} \min \quad & cx^* + f(y) && (15)\\ \text{s.t.} \quad & H(x^*, y), && (16) \end{aligned}$$

where x^* is the solution to the master problem.

3.2 Special Cases

Benders Decomposition: The correspondence to the Branch-and-Check framework is that a problem solved using classical Benders has an *empty* basic part (see (10)) and no relaxation of the subproblem (see (11)–(12)). It only has general (i.e., non-problem specific) lower bounding cuts (3) (see (13)) and nogoods (4) (see (14)) that are derived using LP duality theory.

Branch-and-Bound: Classical Branch-and-Bound,

$$\begin{aligned} \min \quad & cx \\ \text{s.t.} \quad & Ax \leq b, && (17)\\ & x \in \mathbb{R}^n, \text{ some } x_i \in \mathbb{Z}, \end{aligned}$$

is at the other extreme, it has an *empty* delayed part, and hence no relaxation of the subproblem, no lower bounding cuts and no nogoods or only the trivial nogoods that are implicit in the branching and fathoming scheme (see (11)–(16)). It only has a basic part (17) (see (10)).

3.3 Integrating MIP and CLP

It is immediately obvious that a spectrum of techniques exist between classical Benders Decomposition and Branch-and-Bound. In particular:

- In Benders the solution process might be accelerated by adding some cuts or valid inequalities a priori, i.e., adding a linear relaxation of the subproblem (11)–(12), instead of starting with an empty master problem and waiting for the Benders cuts to accumulate and start guiding the process (in the master problem) to promising candidate solutions.

– Instead of looking at the entire problem at every node of the Branch-and-Bound search tree, a part of the set of variables/constraints can be delayed and only examined when need arises. This will result in smaller problems being solved at each node, which although more nodes may be needed, may still result in overall savings.

We note, however, that in addition to this merger of Benders and Branch-and-Bound, the Branch-and-Check framework also allows for an additional dimension of flexibility. The subproblem can be of almost any form, in particular MIP and CLP can be integrated by using CLP to model and solve the subproblems. The MIP search in the master problem is still guided by the subproblem, via the relaxation (11)–(12) and the lower bounds and nogoods (13)–(14).

It is true that if the subproblem is not an LP, or more accurately if duality theory is not available, more work has to be put into deriving the lower bounds and nogoods. A survey of different duality concepts for a variety of problem classes can be found in [14]. It is not uncommon in CLP and MIP, however, to have to tailor methods for specific structures. For example, global constraints in CLP require that propagation algorithms be designed for each one and in MIP, problem specific cutting planes are widely used. In a similar fashion, when integrating CLP and MIP, work has to be put into deriving linear relaxations of mixed global constraints.

3.4 Relation to Previous Work on Decomposition Methods and Nogoods

The first key idea for extensions to the classical Benders framework was due to Jeroslow and Wang [19]. They envisioned the dual of a problem (in the case of classical Benders, an LP) as an inference problem, by showing that when LP demonstrates the unsatisfiability of a set of Horn clauses in propositional logic, the dual solution contains information about a unit resolution proof of unsatisfiability.

Hooker defined the general inference dual in [10], which was then used by Hooker and Yan in [17] for a logic-based Benders scheme in the context of logic circuit verification. There are many similarities between that paper and the paper of Jain and Grossmann [18], except that Hooker and Yan used a specialised inference algorithm rather than a general CLP package for the subproblem, and the problem was logic circuit verification rather than machine scheduling.

Benders Decomposition for Branching, generating Benders cuts from an LP subproblem while *in* the process of solving the master problem, was described by Hooker in [11]. This is the essence of Branch-and-Check, in the context of classical Benders; the examples there do not solve the subproblem with CLP. We go a step further by using a CLP solver to get the cuts for Branch-and-Check, and in addition, we give the first computational results for Branch-and-Check in a Benders context. Branch-and-Check as defined here is a form of Generalised Benders (it partitions the variables and only uses some of them in the master problem, which is the core of Benders) that generates cuts in the process of solving the master problem once.

The idea of using nogoods in branching is a standard AI technique. Branch-and-Check is different in that only a *relaxation* of the problem, rather than the full problem, is solved at each node. The full problem is consulted at only a few

nodes, and nogoods generated accordingly. In classical AI, the *full* problem would generally be checked at every node. The optimisation community has apparently never used nogoods in branching search and the constraint satisfaction community has apparently never used generalised Benders as a means to generate nogoods, although Beringer and De Backer have done related work. The integration of Benders and CLP could give new life to the idea of a nogood, which has received limited attention in practical optimisation algorithms.

4 Examples

In this next section, we will examine two problems, Scheduling with Dissimilar Parallel Machines (SDPM) and Capacitated Vehicle Routing with Time Windows (CVRTW), that benefit from using CLP to model and solve the subproblems, and demonstrate some of the issues that arise.

4.1 Scheduling with Dissimilar Parallel Machines

This problem and a decompositional method to solve it was first presented by Jain and Grossmann in [18]. The problem is described as follows: The least cost schedule has to be derived for processing a set of orders with release and due dates using a set of dissimilar parallel machines. The machines are dissimilar in the sense that there is different cost and processing time associated with each order–machine pair, but all the machines perform the same job. Jain and Grossmann modelled the problem thus:

$$\min \quad \sum_{i\in I}\sum_{m\in M} C_{im}x_{im} \tag{18}$$

$$\text{s.t.} \quad ts_i \le d_i - \sum_{m\in M} p_{im}x_{im}, \quad ts_i \ge r_i, \qquad \forall i \in I, \tag{19}$$

$$\sum_{m\in M} x_{im} = 1, \qquad \forall i \in I, \tag{20}$$

$$\sum_{i\in I} p_{im}x_{im} \le \max_i\{d_i\} - \min_i\{r_i\}, \qquad \forall m \in M, \tag{21}$$

$$\text{if } (x_{im} = 1) \text{ then } (z_i = m), \qquad \forall i \in I, \forall m \in M, \tag{22}$$

$$i.\text{start} \le d_i - p_{z_i}, \quad i.\text{start} \ge r_i, \qquad \forall i \in I, \tag{23}$$

$$i.\text{duration} = p_{z_i}, \qquad \forall i \in I, \tag{24}$$

$$i \text{ requires } t_{z_i}, \qquad \forall i \in I. \tag{25}$$

They also presented a decompositional method that solves this class of MIP problems, i.e., in which only a subset of the variables appears in the objective function. The problem decomposes into an optimisation problem (18)–(20) that is suitable for MIP (has all of the variables of the objective function and a tight relaxation), and into a feasibility problem (23)–(25) that can be solved efficiently using CLP. The variables of the two parts are linked using the mapping (22). The constraints (21)

Problem Model Size			Find and Prove Opt. Solution			
Number	Mach.	Jobs	Iter.	Nogoods	MIP sec	CLP sec
1	5	23	33	71	42.10	0.54
2	5	23	16	15	0.93	0.37
3	5	23	33	76	9.15	0.47
4	5	23	43	104	14.05	0.60
5	5	23	57	72	13.07	1.01

Table 1. Results for 5×23 problems using Jain & Grossmann's approach.

Problem Model Size			Find Opt. Solution				Prove Opt. Thereafter			
Number	Mach.	Jobs	Iter.	Nog.	MIP sec	CLP sec	Iter.	Nog.	MIP sec	CLP sec
1	5	23	8	20	2.99	0.07	7	18	6.62	0.12
2	5	23	3	2	0.09	0.07	0	0	0.00	0.00
3	5	23	19	51	3.78	0.20	0	0	0.00	0.00
4	5	23	19	55	4.05	0.19	0	0	0.00	0.00
5	5	23	17	25	1.79	0.21	6	8	1.12	0.14

Table 2. Results for 5×23 problems using Branch-and-Check.

are not necessary for the correctness of the problem, but are valid inequalities for the overall problem that are added to the MIP part.

The solution process then alternates between solving the optimisation problem to optimality and the resulting feasibility problems. If all the feasibility problems are feasible then the solution is optimal, if not then cuts are added to the optimisation problem to exclude that solution and others similar to it.

This approach bears a striking resemblance to Benders Decomposition. In fact, Hooker showed in [11] how this problem can be written for Benders. It was while studying this result that the idea of Branch-and-Check took form. We note that the correspondence with Branch-and-Check is that the function f, the subproblem part of the objective function, is identically zero (see (6)), there are no lower bounding cuts (see (12)), there is a simple relaxation of the subproblem (21) in the master problem (see (11)), and the problem is solved using multiple search trees by adding nogoods (see (14)) of the form:

$$\sum_{i \in I} a^j_{im} x_{im} \leq \sum_{i \in I} a^j_{im} - 1, \ \forall m \in M.$$

Jain and Grossmann presented very nice computational result in their paper [18], comparing against pure CLP and MIP approaches. While studying the CVRTW problem and how Branch-and-Check could be applied to that problem, we wondered what was the power of this method. First we were looking at the nogoods, but it turns out that the real power of this method lies in the linear relaxation (21). If it is removed from the formulation, problems that are solved in a matter of seconds with the relaxation, can be run for more than 24 hours without making any progress. This indicates that further research into linear relaxations of the global CLP constraints, i.e., *mixed* global constraints [21, 24], is very important.

A further study of the results also revealed a significant difference in the time it took to solve the MIPs vs. the CLPs, up to a factor of 30 times more solving the MIPs. This indicated, and is verified by our results, see Tables 1–4, that the

Problem	Model Size		Find and Prove Opt. Solution			
Number	Mach.	Jobs	Iter.	Nogoods	MIP sec	CLP sec
1	7	30	36	80	15.15	1.06
2	7	30	96	206	90.66	2.78
3	7	30	115	225	116.87	3.42
4	7	30	71	112	34.94	2.25
5	7	30	58	97	28.25	1.92

Table 3. Results for 7×30 problems using Jain & Grossmann's approach.

Problem	Model Size		Find Opt. Solution				Prove Opt. Thereafter			
Number	Mach.	Jobs	Iter.	Nog.	MIP sec	CLP sec	Iter.	Nog.	MIP sec	CLP sec
1	7	30	10	11	0.83	0.36	0	0	0.00	0.00
2	7	30	32	62	9.92	0.98	0	0	0.00	0.00
3	7	30	8	11	0.73	0.27	0	0	0.00	0.00
4	7	30	16	27	2.55	0.46	0	0	0.00	0.00
5	7	30	8	13	0.94	0.24	0	0	0.00	0.00

Table 4. Results for 7×30 problems using Branch-and-Check.

master problem should not necessarily be solved to optimality, instead the CLP subproblems should be solved regularly throughout the tree. This result is very intuitive, as we note that the CLP subproblem decomposes into problems for each individual machine and hence are rather small, compared to the larger MIP master problem that considers all the machines at the same time. We also compared our approach on the original data given by Jain and Grossmann in [18] and obtained very favourable results. Most of those instances are, however, trivially solved using either method, so we do not include them here.

We implemented the Branch-and-Check approach for this problem thus, using OPL and OPL Script [25]: We halted the MIP master problem when a feasible solution was found and solved the CLP subproblems. If any of them were infeasible, we added nogoods to the master problem and re-solved. If all were feasible, we recorded that as a new "current-best-solution", constrained the objective function of the master problem and re-solved. This process was iterated until the master problem was infeasible, indicating that no further solutions could be found given the current bound on the objective function and the nogoods posted.

There is significant overhead with this implementation and redundant calculations: We re-start the master problem after adding cuts, instead of continuing from where we left off, and thus resolve many similar nodes of the search tree repeatedly. A better tool that would allow dynamic modifications of the master problem at each node of the search tree would obtain substantially better results.

4.2 Capacitated Vehicle Routing with Time Windows

This problem is one of visiting a set of customers using vehicles stationed at a central depot; respecting constraints such as the *capacity* of the trucks, a *time window* promised to each customer, *precedence* constraints on the customers, etc. The goal is to produce a low cost routing plan, specifying for each vehicle what customers they should visit and in what order. Cost is generally proportional to the number of vehicles, the maximum time or the total travel time.

We note that this problem decomposes. Given an assignment of trucks to routes that assigns each customer to a specific truck and obeys the capacity constraints, we have to sequence each truck by solving a Travelling Salesman Problem with Time Windows that satisfies the time window and precedence constraints and minimises our objective for each one.

Using the global `cumulative` and `count` constraints and variable index sets we can state the problem as follows for Branch-and-Check, minimising the cost of the trucks:

$$\min \sum_{i \in T} c_i y_i \tag{26}$$

$$\text{s.t. } t_j \geq R_j,\ t_j + D_j \leq S_j, \qquad \forall j \in C, \tag{27}$$

$$\sum_{j \mid (z_j = i)} w_j \leq L_i, \qquad \forall i \in T, \tag{28}$$

$$(y_i = 0) \Rightarrow \texttt{count}(i, [z_1, \ldots, z_n], =, 0), \qquad \forall i \in T, \tag{29}$$

$$\texttt{cumulative}((j \mid (z_j = i)), t_k, R_k, S_k, D_k, [d_{k_1 k_2}], \mathbf{1}, 1), \qquad \forall i \in T. \tag{30}$$

Equations (27) are the time windows, (28) are the capacity constraints and (29) ensure that if a truck is not being used, then no customers are assigned to it. The cumulative constraint (30) is imposed for each truck and schedules the customers assigned to it. The parameters are the customers assigned to the truck, the start time variables, time windows and durations of service, the transition times between all pairs of customers, a vector of all ones indicating that each customer requires one truck, and finally that there is one truck available. If z has been fixed to z^* then the subproblem for each truck i is:

$$\texttt{cumulative}((j \mid (z_j^* = i)), t'_k, R_k, S_k, D_k, [d_{k_1 k_2}], \mathbf{1}, 1), \tag{31}$$

$$t'_j \geq R_j,\ t'_j + D_j \leq S_j, \qquad \forall j \mid (z_j^* = i). \tag{32}$$

If the subproblem is infeasible then nogoods can be generated to avoid that assignment and added to the master problem. Call the accumulated set of those nogoods in the l-th iteration $N_l(x_{ij})$. Then we can write the master problem thus as a MIP:

$$\min \sum_{i \in T} c_i y_i \tag{33}$$

$$\text{s.t. } t_j \geq R_j,\ t_j + D_j \leq S_j, \qquad \forall j \in C, \tag{34}$$

$$\sum_{j \in C} w_j x_{ij} \leq L_i, \qquad \forall i \in T, \tag{35}$$

$$x_{ij} \leq y_i, \qquad \forall i \in T,\ j \in C, \tag{36}$$

$$\sum_{i \in T} x_{ij} = 1, \qquad \forall j \in C, \tag{37}$$

$$N_l(x_{ij}). \tag{38}$$

We note that the 0–1 variables x_{ij} and (37) correspond to the general integer variables z_j and the index sets $\{j \mid (z_j = i)\}$.

We add a dynamic relaxation of the subproblem to the master problem by approximating the total travel time as follows: A truck will have to travel to each customer from somewhere. Thus if for each customer we find the nearest neighbour, the sum of those distances and the services times for the customers assigned to a truck is a lower bound on the actual travel time. While solving the master problem some customers will be assigned to a particular truck, through the branching. When that happens we can update the lower bound, noting that the nearest neighbour can not be among those that have been assigned to other trucks. For truck i, let A_i be the set of customers that have been assigned to truck i and let A_0 be the set of unassigned customers. For truck $i \in T$ we add

$$\sum_{j \in C} \left(D_j + \left(\min_{q \in (A_0 \cup A_i) \setminus \{j\}} d_{qj} \right) x_{ij} \right) \;\leq\; \max_{q \in A_0 \cup A_i} S_q \;-\; \min_{q \in A_0 \cup A_i} R_q.$$

to the master problem. The sets A_i and the relaxation can be updated based on what x_{ij}'s have been fixed to 1:

Set propagation: All customers start in A_0. When x_{pq} is fixed to 1, then customer q moves from A_0 to A_p and x_{pj}, $j \neq q$, can be fixed to 0.

Relaxation propagation: We calculate the $n \times n$ table of shortest distances and sort each list, before solving, so that for each customer there is a list of length $n-1$ of the other customers in increasing distance order. We then build graph of nearest neighbours. Each node has one outgoing arc, the nearest neighbour, and some incoming arcs from the nodes that consider it to be their nearest neighbour. The trigger for the propagation is when customer q moves from A_0 to A_p:

Outgoing arc propagation: Customer q may have to revise its choice for nearest neighbour q^*. If q^* is in A_k, $k \neq 0, p$, then q must look at its list and find the first customer after q^* that is in A_0 or A_p.

Incoming arc propagation: Node q must notify the nodes that consider it to be their nearest neighbour. Every such node in A_k, $k \neq 0, p$, must perform outgoing arc propagation, revising its choice for nearest neighbour by looking for the first customer on its list after q that is in A_0 or A_k.

In addition we can add various other valid inequalities to the master problem, such as symmetry breaking constraints if the trucks are identical (i.e., same cost and capacity). We can require that the first stop assigned truck i be less than or equal to the first stop assigned truck $i+1$. This can be stated in inequality form as

$$x_{i+1,n} \leq \sum_{j=1}^{m} x_{ij}, \quad \forall m, n \text{ with } n \leq m,\ \forall i \in T.$$

We can also order the trucks by adding constraints of the form $y_i \leq y_{i+1}$ (if the number of trucks is variable).

5 Conclusion

CLP and MIP are approaches that have the potential for integration to benefit the solution of combinatorial optimisation problems. In this paper we proposed a modeller/solver framework, Branch-and-Check, that encompasses both the traditional

Benders and Branch-and-Bound schemes of MIP as special cases of a spectrum of solution methods and adds an extra dimension by allowing the integration of CLP in a MIP style branching search. In particular we have described the relationship between Branch-and-Check and Benders.

We have presented the intuition behind Branch-and-Check, to delay parts of the problem, and verified with computational experiments. We have also addressed one of the key elements for the integration of CLP and MIP: Dynamic linear relaxations of global constraints. The computational results indicate that efficient communication between the different parts of a hybrid model requires some double modelling, i.e., the same constraint must be present in both CLP and MIP form. Most preferably this double modelling should be implicit, i.e., mixed global constraints should post and dynamically update a linear relaxation of themselves. This relaxation should be adjustable, i.e, it should be possible to efficiently make incremental changes, rather than recompute it at every node.

Indirectly, we have also mentioned the issue of the availability of flexible tools for testing integration ideas, or the lack thereof. We conclude that there is pressing need in this community to have access to a branching solver that is efficient but also highly customisable to allow for customisation of how each node of the search tree is processed, solved and propagated, and how the problem is modified at each node both when branching and backtracking.

Acknowledgements

We would like to thank Prof. John N. Hooker for his helpful comments on this paper.

References

[1] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numer. Math.*, 4:238–252, 1962.

[2] H. Beringer and B. De Backer. Combinatorial problem solving in constraint logic programming with cooperating solvers. In C. Beierle and L. Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, Studies in Computer Science and Artificial Intelligence, chapter 8, pages 245–272. Elsevier, 1995.

[3] A. Bockmayr and T. Kasper. Branch-and-infer: A unifying framework for integer and finite domain constraint programming. *INFORMS Journal on Computing*, 10(3):287–300, 1998.

[4] K. Darby-Dowman and J. Little. The significance of constraint logic programming to operational research. *Operational Research Tutorial Papers*, pages 20–45, 1995.

[5] K. Darby-Dowman and J. Little. Properties of some combinatorial optimization problems and their effect on the performance of integer programming and constraint logic programming. *INFORMS Journal on Computing*, 10(3):276–286, Summer 1998.

[6] I. R. de Farias, E. L. Johnson, and G. L. Nemhauser. A branch-and-cut approach without binary variables to combinatorial optimization problems with continuous variables and combinatorial constraints. *Knowledge Engineering Review, special issue on AI/OR, submitted*, 1999.

[7] F. Focacci, A. Lodi, and M. Milano. Cutting planes in constraint programming: An hybrid approach. In *CP-AI-OR'00 Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*, March 2000.

[8] A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization theory and Applications*, 10:237–260, 1972.

[9] I. Harjunkoski, V. Jain, and I.E. Grossmann. Hybrid mixed-integer/constraint logic programming strategies for solving scheduling and combinatorial optimization problems. *Computers and Chemical Engineering*, 24:337–343, 2000.

[10] J. N. Hooker. Logic-based methods for optimization. In Alan Borning, editor, *Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*. Springer, May 1994. (PPCP'94: Second International Workshop, Orcas Island, Seattle, USA).

[11] J. N. Hooker. *Logic-Based Methods for Optimization*. Wiley, New York, 2000.

[12] J. N. Hooker and M. A. Osorio. Mixed logical/linear programming. *Discrete Applied Mathematics*, 96–97(1–3):395–442, 1999.

[13] John N. Hooker, Hak-Jin Kim, and Greger Ottosson. A declarative modeling framework that integrates solution methods. *Annals of Operations Research, Special Issue on Modeling Languages and Approaches, to appear*, 1998.

[14] John N. Hooker and Greger Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 2000. Submitted.

[15] John N. Hooker, Greger Ottosson, Erlendur S. Thorsteinsson, and Hak-Jin Kim. On integrating constraint propagation and linear programming for combinatorial optimization. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 136–141. AAAI, The AAAI Press/The MIT Press, July 1999.

[16] John N. Hooker, Greger Ottosson, Erlendur S. Thorsteinsson, and Hak-Jin Kim. A scheme for unifying optimization and constraint satisfaction methods. *Knowledge Engineering Review, Special Issue on Artifical Intelligence and Operations Research*, 15(1):11–30, 2000.

[17] John N. Hooker and Hong Yan. Logic circuit verification by Benders decomposition. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming: The Newport Papers*, pages 267–288. MIT Press, 1995.

[18] V. Jain and I.E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. INFORMS, 2000. Presented at INFORMS Salt Lake City, paper SD32.1.

[19] R.G. Jeroslow and J. Wang. Dynamic programming, integral polyhedra, and horn clause knowledge bases. *ORSA Journal on Computing*, 1(1):7–19, 1988.

[20] Michela Milano, Greger Ottosson, Philippe Refalo, and Erlendur S. Thorsteinsson. Global constraints: When constraint programming meets operation research. *INFORMS Journal on Computing, Special Issue on the Merging of Mathematical Programming and Constraint Programming*, March 2001. Submitted.

[21] Greger Ottosson, Erlendur S. Thorsteinsson, and John N. Hooker. Mixed global constraints and inference in hybrid CLP–IP solvers. *Annals of Mathematics and Artificial Intelligence, Special Issue on Large Scale Combinatorial Optimisation and Constraints*, March 2001. Accepted for publication.

[22] Philippe Refalo. Tight cooperation and its application in piecewise linear optimization. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming*, volume 1713 of *Lecture Notes in Computer Science*. Springer, October 1999.

[23] Robert Rodošek, Mark Wallace, and Mozafar Hajian. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research, Advances in Combinatorial Optimization*, 86:63–87, 1999.

[24] Erlendur S. Thorsteinsson and Greger Ottosson. Linear relaxations and reduced-cost based propagation of continuous variable subscripts. *Annals of Operations Research, Special Issue on Integration of Constraint Programming, Artificial Intelligence and Operations Research Methods*, November 2001. Accepted for publication.

[25] P. Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, 1999.