Branch-and-Check with Explanations for the Vehicle Routing Problem with Time Windows

Edward Lam[1,2] and Pascal Van Hentenryck[3]

[1] CSIRO Data61, Eveleigh NSW 2015, Australia
[2] University of Melbourne, Parkville VIC 3010, Australia
[3] University of Michigan, Ann Arbor MI 48109-2117, USA

Abstract. This paper proposes the framework of branch-and-check with explanations (BCE), a branch-and-check method where combinatorial cuts are found by general-purpose conflict analysis, rather than by specialized separation algorithms. Specifically, the method features a master problem that ignores combinatorial constraints, and a feasibility subproblem that uses propagation to check the feasibility of these constraints and performs conflict analysis to derive nogood cuts. The BCE method also leverages conflict-based branching rules and strengthens cuts in a post-processing step. Experimental results on the Vehicle Routing Problem with Time Windows show that BCE is a potential alternative to branch-and-cut. In particular, BCE dominates branch-and-cut, both in proving optimality and in finding high-quality solutions quickly.

1 Introduction

Vehicle Routing Problems (VRPs) generalize the Travelling Salesman Problem (TSP). The Capacitated Vehicle Routing Problem (CVRP) is a basic variant that aims to design routes of minimal travel distance that deliver all requests from a single depot while respecting vehicle capacity constraints. The Vehicle Routing Problem with Time Windows (VRPTW) additionally requires requests to be delivered within a given time window.

VRPs have been studied extensively over the past several decades, resulting in significant computational progress (e.g., [31]). Solution techniques include constraint programming (e.g., [28,7,8]), branch-and-bound, branch-and-cut (e.g., [21,4,18]), branch-and-price (e.g., [10]), and combinations thereof (e.g., [13,3,16,25,26]). Branch-and-cut (BC) methods are of particular interest to this paper. Their key idea is to omit difficult constraints from the original formulation and to remove solutions that violate these constraints using cuts generated by separation algorithms. Separation algorithms are typically problem-specific, which limits their applicability and reuse in other problems. Furthermore, developing and implementing separation algorithms often require significant expertise, hindering their use in many applications.

This paper addresses the following research question: *Is it possible to use a general-purpose mechanism to generate cuts, and hence, avoid the difficult aspects of BC.* This paper proposes *branch-and-check with explanations* (BCE) as one

possible answer to this question. BCE divides an optimization problem into a master problem that ignores a number of difficult constraints, and a subproblem that checks the feasibility of these constraints and generates cuts using conflict analysis from constraint programming (CP) and Boolean satisfiability (SAT). More precisely, BCE uses CP for three purposes: (1) to fix variables in the master problem through propagation; (2) to generate cuts in the master problem using conflict analysis; and (3) to probe the feasibility of linear programming (LP) relaxation solutions and to derive additional cuts through conflict analysis if the probing process fails. Since the master problem does not operate on the same decision variables as the subproblem, the conflict analysis needs to continue until the variables involved in a nogood appear in the master problem.

BCE opens some interesting opportunities. First, it has the advantage of relying on a general-purpose CP engine for inference and cut separation. Second, it permits conflict-based branching rules. Finally, BCE can recognize special classes of cuts after conflict analysis and then strengthen them using well-known techniques. As a result, BCE offers a natural integration of LP, CP and SAT.

The BCE method is evaluated on the VRPTW. Experimental results indicate that BCE outperforms a BC algorithm: it proves optimality on more instances and finds significantly better solutions to instances for which BC cannot prove optimality. The results also show that a conflict-based branching rule is particularly effective in BCE and that cut strengthening produces interesting improvements to the lower bounds.

The rest of this paper is structured as follows. Section 2 reviews relevant methods for solving the VRPTW. Section 3 develops the BCE model. Section 4 discusses cut strengthening. Section 5 presents experimental results that compare the BCE model with the BC model. Section 6 discusses the limitations and potential improvements of the BCE approach for the VRPTW, as well as its relevance to branch-and-price. Section 7 concludes this paper.

2 Background

BC algorithms for VRPs are often based on a two-index flow model, which generalizes the standard formulation of the TSP. The two-index model omits the subtour elimination, vehicle capacity and time window constraints, which are added as required through cutting planes. At every node of the search tree, BC solves separation subproblems to determine if the LP relaxation solution is feasible with respect to the omitted constraints. If the solution is infeasible, the solution is discarded using a cut, forcing the solver to find another candidate solution. Branching and cutting are repeated until the search tree is explored, upon which the solver proves optimality or infeasibility.

Branch-and-Cut. BC models of the VRPTW rely on several types of cuts. The BC model in [4] inherits the capacity cuts from BC models of the CVRP. Capacity cuts generalize the subtour elimination cuts of the TSP to consider vehicle capacity. Hence, they serve the purpose of excluding both subtours and partial paths that

exceed the vehicle capacity. This model also implements infeasible path cuts to exclude partial paths that violate the time windows. Infeasible path cuts require at least one arc in an infeasible partial path to be unused. The BC algorithm from [18] uses subtour elimination constraints from the TSP instead of the capacity cuts. Vehicle capacity constraints are enforced by the same infeasible path cuts that enforce the time windows. The authors also prove that both the subtour elimination cuts and the infeasible path cuts can be strengthened using ideas conceived in [22].

Branch-and-Check. Branch-and-check [29,5] is a form of logic-based Benders decomposition [15]. The method divides a problem into a master and checking problem. The master problem is first solved to find a candidate solution, which is checked using the checking subproblem. If the checking subproblem is infeasible for a candidate solution, a constraint prohibiting this solution, and hopefully many others, is added to the master problem. Branch-and-check iterates between the master problem and the checking subproblem until a globally optimal solution is found. It has been used successfully in various applications (e.g., [14,30]). The key difference between BC and branch-and-check is that checking subproblems encompass an entire optimization problem, whereas separation subproblems only check specific aspects of the problem (i.e., they find cuts from one family).

Constraint Programming. CP with large neighborhood search was instrumental in finding many best solutions to VRPs more than a decade ago [28,7,8]. The main difficulty with CP is proving optimality since VRP objective functions are usually linear, which are known to have weak propagators. This limitations can be alleviated using the WeightedCircuit global constraint [6], for example.

Conflict Analysis. Conflict analysis has a long history in artificial intelligence and CP (e.g., [9,17]). Its popularity grew in the last two decades through the development of SAT solvers (e.g., [23,11]) and their integration in CP solvers (e.g., [24,12]). In CP solvers, propagators generate clauses that explain the inferences for an underlying SAT solver. When a propagator fails, the SAT solver performs conflict analysis, i.e., it walks the implication graph to derive a constraint, known as a nogood, that prevents the same failure from reoccurring in other parts of the search tree. Conflict analysis can also be implemented in mixed integer programming (MIP) solvers but its performance is still an open question [1].

3 The Branch-and-Check Model of the VRPTW

This section proposes the BCE model of the VRPTW. The model is organized around a MIP master problem and a CP checking subproblem.

The MIP Master Problem. The BCE model includes the traditional two-index model. Its data and decision variables are listed in Table 1. The arcs in the

Name	Description
$T > 0$	Time horizon.
$\mathcal{T} = [0, T]$	Time interval.
$Q \geq 0$	Vehicle capacity.
$\mathcal{Q} = [0, Q]$	Range of vehicle load.
$R \in \{1, \ldots, \infty\}$	Number of requests.
$\mathcal{R} = \{1, \ldots, R\}$	Set of requests.
$s = 0$	Start node.
$e = R + 1$	End node.
$\mathcal{N} = \mathcal{R} \cup \{s, e\}$	Set of all nodes.
$\mathcal{A}$	Arcs of the network. Defined in Eq. (4).
$c_{i,j} \in \mathcal{T}$	Distance cost and travel time along arc $(i, j) \in \mathcal{A}$.
$q_i \in Q$	Vehicle load demand of $i \in \mathcal{N}$.
$a_i \in \mathcal{T}$	Earliest service start time at $i \in \mathcal{N}$.
$b_i \in \mathcal{T}$	Latest service start time at $i \in \mathcal{N}$.
$x_{i,j} \in \{0, 1\}$	Decision variable indicating if a vehicle traverses $(i, j) \in \mathcal{A}$.

Table 1. Data and decision variables of the two-index flow model of the VRPTW. Sets enclosed in braces (resp. square brackets) are integer-valued (resp. real-valued).

$$\min \sum_{(i,j)\in\mathcal{A}} c_{i,j} x_{i,j} \tag{1}$$

subject to

$$\sum_{h:(h,i)\in\mathcal{A}} x_{h,i} = 1 \qquad \forall i \in \mathcal{R}, \tag{2}$$

$$\sum_{j:(i,j)\in\mathcal{A}} x_{i,j} = 1 \qquad \forall i \in \mathcal{R}. \tag{3}$$

Fig. 1. Initial constraints of the two-index model of the VRPTW.

network are given by

$$\begin{aligned} \mathcal{A} = &\{(s, i) | i \in \mathcal{R}\} \cup \\ &\{(i, j) | i, j \in \mathcal{R}, i \neq j, a_i + c_{i,j} \leq b_j, q_i + q_j \leq Q\} \cup \\ &\{(i, e) | i \in \mathcal{R}\}. \end{aligned} \tag{4}$$

The initial constraints of the model are shown in Fig. 1. The objective function, Eq. (1), minimizes the total distance cost. Constraints (2) and (3) require every request to be visited exactly once. Through its LP relaxation, the MIP master problem provides the lower bounds to the objective value and generates candidate solutions to be tested in the CP subproblem.

Name	Description
$y_{i,j} \in \{0,1\}$	Decision variable indicating if a vehicle traverses $(i,j) \in \mathcal{A}$.
$l_i \in [q_i, Q] \subseteq \mathcal{Q}$	Vehicle load after service at request $i \in \mathcal{N}$.
$t_i \in [a_i, b_i] \subseteq \mathcal{T}$	Time that a vehicle begins service at request $i \in \mathcal{N}$.

Table 2. Decision variables of the CP subproblem.

$$\bigvee_{h:(h,i)\in\mathcal{A}} y_{h,i} \qquad \forall i \in \mathcal{R}, \quad (5)$$

$$\bigvee_{j:(i,j)\in\mathcal{A}} y_{i,j} \qquad \forall i \in \mathcal{R}, \quad (6)$$

$$\neg y_{h,i} \vee \neg y_{h,j} \qquad \forall h,i,j \in \mathcal{N} : (h,i) \in \mathcal{A}, (h,j) \in \mathcal{A}, i \neq j, \quad (7)$$

$$\neg y_{h,j} \vee \neg y_{i,j} \qquad \forall h,i,j \in \mathcal{N} : (h,j) \in \mathcal{A}, (i,j) \in \mathcal{A}, h \neq i, \quad (8)$$

$$\textsc{NoSubtour}(y), \quad (9)$$

$$y_{i,j} \rightarrow l_j \geq l_i + q_j \qquad \forall (i,j) \in \mathcal{A}, \quad (10)$$

$$y_{i,j} \rightarrow t_j \geq t_i + c_{i,j} \qquad \forall (i,j) \in \mathcal{A}. \quad (11)$$

Fig. 2. Initial constraints of the CP subproblem.

The CP Checking Subproblem. The BCE model uses a CP subproblem to check the solutions found by the master problem for feasibility of the subtour elimination, vehicle capacity and time window constraints. The decision variables are listed in Table 2. Using the y binary variables, instead of the conventional successor and predecessor variables, provides a one-to-one mapping between the y variables and the x variables of the master problem. The initial constraints (without the nogoods) are presented in Fig. 2. Constraints (5) to (8) ensure that every request is visited exactly once. Constraint (9) is a global constraint that prevents subtours. Its propagator is a simple checking algorithm that prevents the head of a partial path from connecting to its tail. Constraints (10) and (11) enforce the vehicle capacity and travel time constraints.

Communication between the Two Models. The two models communicate in three ways: (1) variable assignments in the CP model are transmitted to the MIP model, (2) candidate solutions from the LP relaxation are probed using the CP model to determine if they are valid for the VRPTW and (3) nogoods found by conflict analysis in the CP model are translated into cuts in the MIP model.

Extended Conflict Analysis. When a failure occurs in the CP solver, conflict analysis derives a First Unique Implication Point (1UIP) nogood that is added to the CP subproblem. This constraint should also be added to the master problem but sometimes it cannot be translated into a cut for the master problem because

it contains variables that do not appear in the master problem (i.e., the load and time variables). As a result, the BCE algorithm features an extended conflict analysis that continues explaining the failure until the the nogood only contains variables in master problem. This nogood has the form

$$\bigvee_{(i,j)\in\mathcal{C}_1} y_{i,j} \vee \bigvee_{(i,j)\in\mathcal{C}_2} \neg y_{i,j},$$

where $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{A}$ are sets of arcs. This nogood can be rewritten as the cut

$$\sum_{(i,j)\in\mathcal{C}_1} x_{i,j} + \sum_{(i,j)\in\mathcal{C}_2} (1 - x_{i,j}) \geq 1.$$

It is always possible to obtain these cuts since the solver only branches on variables in the master problem. *Observe that the BCE algorithm provides a general-purpose mechanism to separate cuts in the master problem via the extended conflict analysis. These cuts, which we call MIP-1UIP nogoods, are automatically generated and do not rely on specialized separation algorithms.*

Probing the LP Relaxation. The BCE algorithm probes whether the current LP solution is feasible with respect to the subtour elimination, vehicle capacity and time constraints. It temporarily assigns every $y_{i,j}$ variable to the value of its corresponding $x_{i,j}$ variable in the LP relaxation, provided that this value is integral. The resulting tentative assignment can then be propagated by the CP solver. If a failure occurs, conflict analysis generates nogoods for both the CP and MIP models. The MIP cut will exclude the current LP solution, forcing it to find another candidate solution and improving the lower bound.

The Search Algorithm. The BCE algorithm, detailed in Fig. 3, includes the components described earlier. It blends depth-first and best-first search since best-first search is more effective for hard optimization problems, such as VRPs, but complicates the implementation of CP solvers with conflict analysis. The node selection strategy selects the node with the lowest lower bound from the set of open nodes and then explores the node subtree using depth-first search until it reaches a limit on the maximum number of open nodes per subtree. Once it reaches this limit, all unsolved siblings in the subtree are moved into the set of open nodes, and then the algorithm starts a new depth-first search from the node with the next lowest lower bound. Section 6 explains the rationale behind this search procedure.

Once a node is selected (step 1), the CP subproblem infers the implications of the decision (step 2). In the case of failure, the CP solver generates nogoods for both models and then backtracks (step 5b). If the test succeeds, the BCE algorithm checks for suboptimality using the LP relaxation (step 3). If the node is suboptimal, it backtracks (step 5b). Otherwise, the BCE algorithm checks the LP relaxation solution against the omitted constraints and separates cuts using conflict analysis if necessary (step 4). The BCE algorithm iterates between the LP relaxation and the feasibility test until no cuts are generated. Then, if the

1. **Node Selection:** Select an open node. Terminate if no open nodes remain.
2. **Feasibility Check:** Solve the CP model to determine the implications of the branching decision of the node. If propagation fails, perform conflict analysis, add the 1UIP and the MIP-1UIP nogoods to both the CP and MIP models, and go to step 5b. Otherwise, fix $x_{i,j}$ in the MIP model to the values of the $y_{i,j}$ variables.
3. **Suboptimality Check:** Solve the LP relaxation. If the objective value is worse than the incumbent solution, go to step 5b.
4. **LP Probing:** For all $x_{i,j}$ variables with a value of 0 or 1 in the LP relaxation, temporarily fix the $y_{i,j}$ variables in the CP model to the same value. Propagate the CP model. If it fails, perform conflict analysis, generate the 1UIP and the MIP-1UIP nogoods and go back to step 3.
5. **Branching and Backtracking:** If all $x_{i,j}$ variables are integral, store the LP relaxation solution as the incumbent solution and go to step 5b. Otherwise, go to step 5a because the node is fractional.
   (a) **Branching:** Create two children nodes from a fractional $x_{i,j}$ variable. Fix the variable to 0 in one child node and to 1 in the other.
   (b) **Backtracking:** If the number of nodes in the current subtree exceeds the limit or if the subtree is entirely solved, move all unsolved siblings in the subtree to the set of open nodes and go back to step 1. Otherwise, backtrack to an ancestor with an unsolved child node, select the child node and go to step 2.

Fig. 3. The BCE Search Algorithm.

node is fractional and not suboptimal, the BCE algorithm executes a branching step (step 5a). Two branching rules are implemented. The first selects the most fractional variable and the second selects the variable with the highest activity, which is defined as the number of nogoods in which the variable has previously appeared. This branching rule, known as activity-based search or variable state independent decaying sum (VSIDS) in the literature, guides the search tree towards subtrees that can be quickly pruned due to infeasibility.

Illustrating the Extended Conflict Analysis. The following discussion illustrates the extended conflict analysis procedure using the example in Figs. 4 and 5. Literals shown in a grey are fixed by the data at the root level, and hence, are always true. They are discarded in the explanations but are shown for clarity.

The BCE solver first branches on $\neg y_{4,6}$, making it true. The travel time constraint (Constraint (11)) propagates $[\![t_6 \geq 30]\!]$ with the reason

$$\neg y_{4,6} \wedge [\![t_3 \geq 25]\!] \wedge [\![c_{3,6} = 10]\!] \wedge [\![t_5 \geq 20]\!] \wedge [\![c_{5,6} = 10]\!] \rightarrow [\![t_6 \geq 30]\!]$$

because the predecessor of request 6 must be either 3 or 5, and the earliest time to reach 6 is at time $\min(\min(t_3) + c_{3,6}, \min(t_5) + c_{5,6}) = 30$. The BCE solver then branches on $\neg y_{3,6}$. Constraint (5) requires every request to have a predecessor, which leads to the assignment of $y_{5,6}$ with the reason

$$\neg y_{3,6} \wedge \neg y_{4,6} \rightarrow y_{5,6}.$$

Constraint (8) then propagates

$$y_{5,6} \rightarrow \neg y_{5,2}$$

and

$$y_{5,6} \rightarrow \neg y_{5,7}.$$

The BCE solver then branches on $y_{0,1}$, which does not produce any inference, and then branches on $y_{6,2}$, which produces the inferences:

$$\begin{gathered} y_{6,2} \rightarrow \neg y_{6,7}, \\ \neg y_{6,7} \wedge \neg y_{5,7} \rightarrow y_{2,7}, \\ y_{6,2} \wedge [\![t_6 \geq 30]\!] \wedge [\![c_{6,2} = 10]\!] \rightarrow [\![t_2 \geq 40]\!]. \end{gathered}$$

Then, the travel time propagator fails with

$$y_{2,7} \wedge [\![t_2 \geq 40]\!] \wedge [\![c_{2,7} = 10]\!] \wedge [\![t_7 \leq 40]\!] \rightarrow \text{false}.$$

Conflict analysis deduces the following:

$$\begin{gathered} y_{2,7} \wedge [\![t_2 \geq 40]\!] \wedge [\![c_{2,7} = 10]\!] \wedge [\![t_7 \leq 40]\!] \rightarrow \text{false} \\ y_{2,7} \wedge [\![t_2 \geq 40]\!] \wedge \text{true} \wedge \text{true} \rightarrow \text{false} \\ (\neg y_{6,7} \wedge \neg y_{5,7}) \wedge (y_{6,2} \wedge [\![t_6 \geq 30]\!] \wedge [\![c_{6,2} = 10]\!]) \rightarrow \text{false} \\ y_{6,2} \wedge \neg y_{5,7} \wedge [\![t_6 \geq 30]\!] \wedge \text{true} \rightarrow \text{false} \\ y_{6,2} \wedge \neg y_{5,7} \wedge [\![t_6 \geq 30]\!] \rightarrow \text{false}. \end{gathered} \tag{12}$$

This explanation contains exactly one literal ($y_{6,2}$) at the current depth, and hence, is rewritten as the 1UIP clause

$$\neg y_{6,2} \vee y_{5,7} \vee [\![t_6 < 30]\!],$$

which is added to the CP model. Conflict analysis must continue because the nogood contains a time literal. It explains $[\![t_6 \geq 30]\!]$ in Eq. (12), which results in the MIP-1UIP explanation

$$y_{6,2} \wedge \neg y_{5,7} \wedge \neg y_{4,6} \rightarrow \text{false}.$$

This explanation is rewritten into the disjunction

$$\neg y_{6,2} \vee y_{5,7} \vee y_{4,6}, \tag{13}$$

and then into the cut

$$(1 - x_{6,2}) + x_{5,7} + x_{4,6} \geq 1.$$

Note that the literal $y_{5,7}$ was not assigned by the search.

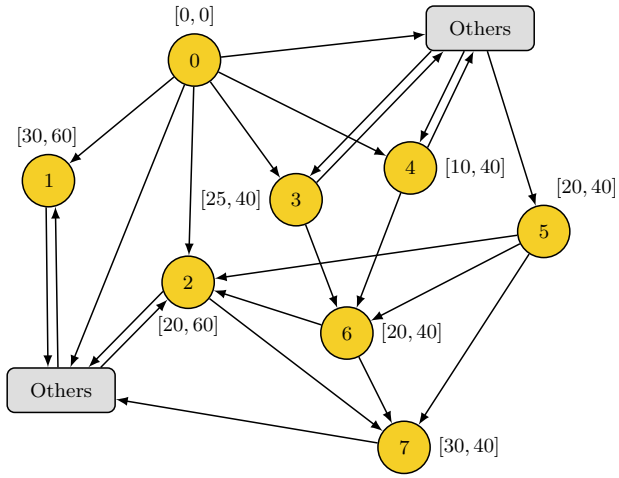

Fig. 4. Example of a network. Next to every request is its time window. The travel time across any arc is 10 units of time. The load demands are not shown as they are not relevant to the discussion.

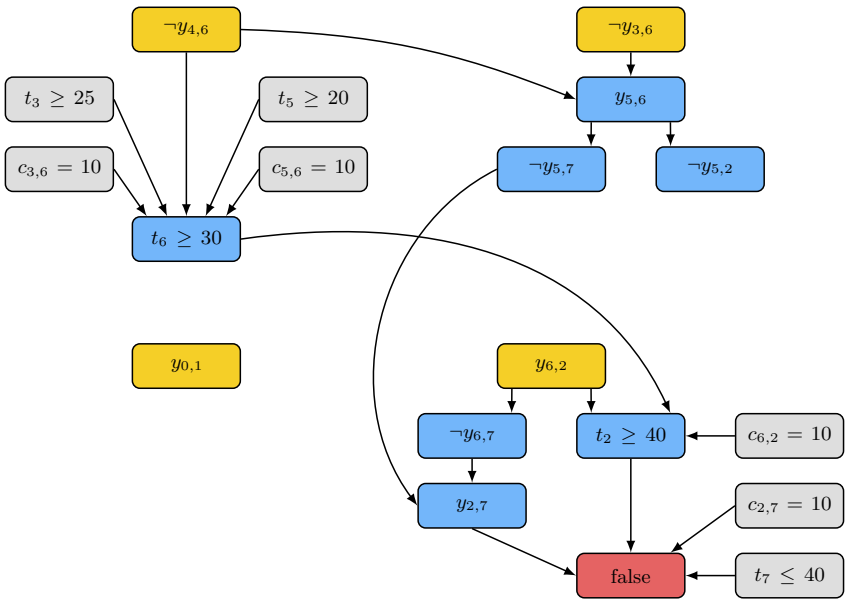

Fig. 5. Example of an implication graph after making the decisions $\neg y_{4,6}$, $\neg y_{3,6}$, $y_{0,1}$ and $y_{6,2}$ on the network in Fig. 4. Yellow literals are branching decisions. Blue literals are propagations. Grey literals are propagated at the root level, and hence, can be excluded from the nogoods since they are always true.

4 Nogood Strengthening

The BCE algorithm presented so far uses a completely general-purpose mechanism for cut separation. Despite its generality, conflict analysis routinely discovers classical cuts. These cuts can be strengthened using proven techniques whenever they are recognized. This section presents a post-processing step that recognizes then strengthens several types of cuts.

Infeasible Path Cuts. Failure of the load or time constraints (Constraints (10) and (11)) frequently results in an infeasible partial path cut. Let $P = i_1, i_2, \ldots, i_k$, with all $i_1, \ldots, i_k \in \mathcal{N}$ distinct, be a partial path. The partial path P is infeasible with respect to the load constraint if $\sum_{u=1}^{k} q_{i_u} > Q$, and it is is infeasible with respect to the time constraint if $t_{i_k} > b_{i_k}$, where $t_{i_1} = a_{i_1}$ and $t_{i_u} = \max(a_{i_u}, t_{i_{u-1}} + c_{i_{u-1},i_u})$ for $u = 2, \ldots, k$. When a load or time window constraint fails, conflict analysis will usually produce the nogood

$$\bigvee_{(i,j)\in A(P)} \neg y_{i,j}, \tag{14}$$

where $A(P) = \{(i_1, i_2), \ldots, (i_{k-1}, i_k)\}$ is the arcs of P. This nogood requires one arc of P to be unused. It can be written equivalently as requiring at least one arc that exits P, i.e.,

$$\bigvee_{(i,j)\in\Delta^+(P)} y_{i,j},$$

where $\Delta^+(P) = \bigcup_{u=1}^{k-1}\{(i_u, j) \in \mathcal{A} | j \neq i_{u+1}\}$. This nogood can be translated into the cut

$$\sum_{(i,j)\in\Delta^+(P)} x_{i,j} \geq 1.$$

Using existing techniques [18], such a cut can be strengthened into

$$\sum_{(i,j)\in\widetilde{\Delta}^+(P)} x_{i,j} \geq 1,$$

where

$$\widetilde{\Delta}^+(P) = \bigcup_{u=1}^{k-1} \left(\left\{ (i_u, j) \in \mathcal{A} : i_u \in \mathcal{R}, j \in \mathcal{R}, j \neq i_1, \ldots, i_{u+1}, \sum_{v=1}^{u} q_{i_v} + q_j \leq Q, t_{i_u} + c_{i_u,j} \leq b_j \right\} \cup \{(i_u, e) \in \mathcal{A}\} \right)$$

is the arcs that branch off P to a feasible request. In other words, the strengthening discards arcs that are not feasible when taking into account the load and time window constraints.

Subtour Elimination Cuts. The propagator of Constraint (9) will fail if the solution contains a subtour $S = i_1, i_2, \ldots, i_k$, where $i_1 = i_k$ and all $i_1, i_2, \ldots, i_{k-1} \in \mathcal{R}$ are distinct. Conflict analysis will usually find the nogood

$$\bigvee_{(i,j)\in A(S)} \neg y_{i,j}, \tag{15}$$

where $A(S) = \{(i_1, i_2), \ldots, (i_{k-1}, i_k)\}$ is the arcs of S. Using the same reasoning as for the infeasible path cuts, this nogood can be rewritten as the cut

$$\sum_{(i,j)\in\Delta^+(S)} x_{i,j} \geq 1. \tag{16}$$

If $a_j + c_{j,i} > b_i$, then no vehicle can depart j for i while respecting the time windows. Hence, i must precede j with respect to time, written as $i \prec j$. Let $\pi(j) = \{i \in \mathcal{N} | i \prec j\}$ be the set of requests that precedes j with respect to time. Proposition 1 strengthens Constraint (16) using these precedence relations [18]. Constraint (16) can also be similarly strengthened using the precedence relations in reverse, i.e., successor relations.

Proposition 1. *Let $\bar{S} = \mathcal{N} \setminus S$ be the nodes not in a subtour S, then for any $u \in S$, Constraint (16) can be strengthened to*

$$\sum_{\substack{(i,j)\in\mathcal{A}:\\ i\in S\setminus\pi(u),\\ j\in\bar{S}\setminus\pi(u)}} x_{i,j} \geq 1.$$

Proof. Consider a subtour S and a feasible path F that visits the request u. Let $v \in \mathcal{R}$ be the last request of F visited by S. By definition, v is visited by S, i.e., $v \in S$. Furthermore, since F is a feasible path, v cannot precede u with respect to time, i.e., $v \notin \pi(u)$. Hence, $v \in S \setminus \pi(u)$. Now consider the successor of v, denoted by $\text{succ}(v) \in \mathcal{N}$*. By the definition of v,* $\text{succ}(v)$ *cannot be visited by S, i.e.,* $\text{succ}(v) \notin S$*. Again,* $\text{succ}(v)$ *cannot precede u with respect to time since F is a feasible path. Hence,* $\text{succ}(v) \in \bar{S} \setminus \pi(u)$*. Considering every request in S as v results in the proposition.*

General Cuts. Conflict analysis can derive cuts that are do not have the form of Constraint (14) nor Constraint (15). These cuts contain both true literals and false literals, such as those of Constraint (13). They originate from fixing an arc to be unused (i.e., setting $x_{i,j} = 0$ for some $(i, j) \in \mathcal{A}$), which can result in tightening the bounds of a time or load variable. Consequently, an assigned arc can become infeasible. Hence, the originating nogood will contain both true and false literals. We are not aware of VRP cuts in the literature that mix true literals and false literals. This is possibly because tightening bounds is too costly for every call to a separation algorithm. CP maintains the bounds internally as part of propagation, and hence, the bounds are readily available. Because of this, these cuts seem to be fundamentally linked to CP. It is an open research issue to understand whether these cuts can be strengthened.

5 Experimental Results

The Solvers. The BCE solver includes a small CP solver and calls Gurobi 6.5.2 to solve the LP relaxations. The algorithm presented in Fig. 3 has a limit of 500 nodes for the depth-first search. This number was chosen experimentally as it was superior to limits of 100, 1,000, 5,000, and 10,000 nodes. The experiments consider four versions of the solver: with and without cut strengthening, and with the two branching rules. The four versions are compared against published results of a BC model [18], as well as a pure CP model and a pure MIP model. The CP model is the standard VRPTW model based on successor variables (e.g., [19,27]), and is solved using Chuffed. The MIP model is the three-index flow model (e.g., [31]), and is solved using Gurobi. The reported results for the BC model are given an hour of CPU time on a Pentium III CPU at 600 MHz. To be fair, our solvers are run for 10 minutes on a Xeon E5-2660 V3 at 2.6 GHz.

The Results. The solvers are tested on the Solomon benchmarks with 100 requests. The results are reported in Table 3. The pure CP model failed to find any feasible solution and is omitted from the table. The pure MIP model proves optimality on only one instance and finds poor solutions to three other instances. These results were expected and are given to confirm the need for the other approaches. The rest of this section compares the BC and BCE approaches.

Upper Bounds. The four BCE methods find the same or better solutions than the BC algorithm for all instances except C204. Of the best solutions found, all but two (R201, C204) can be found using the activity-based branching rule. For the C instances, BC and BCE with activity-based search and cut strengthening are comparable since they both dominate on seven of the eight instances. For the R and RC instances, BCE with activity-based search improves upon the BC method, which generally finds solutions with costs about five times higher.

Lower Bounds. First observe that BCE with activity-based search and cut strengthening proves optimality on one more instance (RC201) than BC, which is quite remarkable. The bounds found by the BC model are superior to those from all BCE methods except for instance RC201, on which BCE with activity-based search and cut strengthening finds a tighter bound. This is not surprising since the BC algorithm implements families of cuts not present in the BCE model. These families of cuts capture logic that the constraints in the checking subproblem do not. As will be mentioned in Section 6, stronger dual bounds should be available once the BCE model is expanded with optimization constraints.

The Impact of Branching Rules. Activity-based branching performs significantly better than most-fractional branching. Without cut strengthening, activity-based branching finds solutions better than most-fractional branching on all instances except C201, on which all four BCE methods prove optimality. With cut strengthening, activity-based search performs better on 19 of the 27 instances, and worse on only one instance. This is not surprising given that branching on the most fractional variable is known to perform worse than random selection [2].

The Impact of Cut Strengthening. Cut strengthening improves the lower bounds for both branching rules. For the C instances, cut strengthening is critical for proving optimality. For the RC instances except RC208, BCE with activity-based branching and cut strengthening finds solutions better than the other methods. Cut strengthening interferes with the activity-based branching rule for about half of the R instances. The cause of this interference is not yet understood.

The results indicate that BCE is an interesting avenue for solving hard VRPs. The BCE model finds superior primal solutions despite its simplicity and the fact that it is missing many families of cuts and that the checking subproblem does not reason about optimality nor variables with fractional values in the LP relaxation solution. For practitioners without the expertise in BC, BCE provides an interesting and practically appealing alternative.

6 Future Research Directions

The BCE algorithm, presented in this paper as a proof-of-concept, can be improved in many ways. This section explores some potential improvements.

Branching. The branching rules simply assign a fractional variable to 0 in one child and 1 in the other. These branching rules make the search tree highly unbalanced, considerably degrading the performance of the solver. Future implementations should test branching on cutsets, which is the standard branching rule seen in BC models of VRPs. It would also be interesting to test branching on variables in the CP model (e.g., branching on time windows) by propagating these decisions and enforcing the implications in the MIP model.

Search Strategy. VRPs greatly benefit from best-first search. For simplicity, the BCE implementation uses depth-first search, which allows literals to be stored in a stack data structure. It is obviously possible to implement conflict analysis in best-first search but efficient implementations remain an open question today. As explained in Section 3, the BCE implementation blends depth-first search with periodic best-first selection to explore attractive parts of the search tree.

Subtour Elimination. The propagator of Constraint (9) is extremely simple and only eliminates assignments that would create a cycle. This contrasts with separation algorithms, which are able to separate cuts using fractional solutions. It would be highly desirable to study the impact of more advanced propagators and explanations for subtour elimination in CP.

Cut Strengthening. The CP model contains all the omitted constraints; namely, the subtour elimination, vehicle capacity and time window constraints. As a result, conflict analysis can deduce nogoods based on the combined infeasibility of multiple constraints. In contrast, separation algorithms only reason about one family of cuts. It is an open question whether conflict analysis can automatically strengthen the cuts by reasoning about a conjunction of constraints. This will reduce the need to develop dedicated cut strengthenings.

	Branch-and-Check – Most-Fractional						Branch-and-Check – Activity-based											
	No Strengthening			With Strengthening			No Strengthening			With Strengthening			Branch-and-Cut			MIP		
Instance	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time
R201	1055.8	1198.0	-	1117.7	**1143.3**	-	1054.7	1177.6	-	1114.3	1149.9	-	1132.7	1155.6	-	975.6	-	-
R202	762.8	1213.0	-	852.1	1219.6	-	763.2	1133.4	-	850.7	**1109.3**	-	888.6	4980.0	-	715.3	-	-
R203	660.1	1244.8	-	709.8	1253.6	-	659.9	**1025.2**	-	707.7	1052.2	-	748.1	4980.0	-	620.3	-	-
R204	625.3	1166.7	-	639.2	1193.3	-	625.8	**858.4**	-	638.3	887.4	-	661.9	4980.0	-	584.9	-	-
R205	796.3	1222.0	-	889.6	1069.9	-	794.3	1091.3	-	876.9	**1052.5**	-	900.0	4980.0	-	732.3	-	-
R206	686.3	1171.6	-	751.4	1157.4	-	686.0	1040.1	-	745.3	**1018.9**	-	783.6	4980.0	-	644.8	-	-
R207	648.1	1187.5	-	681.5	1168.5	-	647.5	**940.7**	-	685.8	941.4	-	714.8	4980.0	-	603.1	-	-
R208	623.2	1097.4	-	633.5	1187.7	-	623.4	855.0	-	635.3	**832.5**	-	651.8	4980.0	-	577.2	-	-
R209	687.5	1238.1	-	756.6	1172.5	-	686.5	**1046.6**	-	753.1	1073.8	-	785.8	4980.0	-	648.2	-	-
R210	679.7	1225.6	-	749.9	1240.3	-	679.8	1105.6	-	750.8	**1024.9**	-	798.3	4980.0	-	636.6	-	-
R211	621.2	1335.5	-	633.0	1355.9	-	621.2	**1004.2**	-	632.1	1065.1	-	645.1	4980.0	-	577.2	4224.9	-
C201	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	11.5	589.1	**589.1**	15.2
C202	548.7	679.8	-	589.1	**589.1**	131.2	548.2	629.9	-	589.1	**589.1**	12.6	589.1	**589.1**	202.9	524.3	-	-
C203	526.5	948.3	-	563.4	672.2	-	524.7	686.5	-	565.9	**601.2**	-	586.0	632.3	-	507.3	-	-
C204	516.3	946.7	-	552.9	1086.7	-	514.7	884.5	-	555.9	660.9	-	584.4	**597.1**	-	488.3	-	-
C205	546.9	685.8	-	586.4	**586.4**	0.2	546.5	613.1	-	586.4	**586.4**	16.3	586.4	**586.4**	334.4	511.4	-	-
C206	539.9	776.9	-	586.0	**586.0**	11.8	538.2	702.6	-	586.0	**586.0**	10.6	586.0	**586.0**	419.0	504.7	4997.5	-
C207	542.7	851.1	-	585.8	**585.8**	20.6	538.3	635.2	-	585.8	**585.8**	8.3	585.8	**585.8**	527.5	503.9	-	-
C208	534.5	857.2	-	585.8	**585.8**	60.0	533.1	652.4	-	585.8	**585.8**	11.2	585.8	**585.8**	569.7	500.3	-	-
RC201	1086.5	1403.6	-	1245.8	**1261.8**	-	1081.0	1338.3	-	1261.8	**1261.8**	44.5	1250.1	1288.2	-	938.3	-	-
RC202	704.5	1465.9	-	912.7	1418.6	-	699.2	1204.2	-	916.9	**1152.3**	-	940.1	6609.4	-	641.0	-	-
RC203	615.0	1402.7	-	750.2	1359.6	-	610.8	1149.0	-	748.8	**1117.6**	-	781.6	6609.4	-	563.1	-	-
RC204	583.9	1410.2	-	657.8	1352.4	-	581.0	1007.1	-	657.0	**923.5**	-	692.7	6609.4	-	532.4	-	-
RC205	822.5	1511.6	-	1075.5	1307.0	-	818.8	1249.8	-	1055.8	**1240.9**	-	1081.7	6609.4	-	746.3	-	-
RC206	785.4	1485.4	-	964.3	1273.9	-	784.9	1270.2	-	950.7	**1202.8**	-	974.8	6609.4	-	698.2	-	-
RC207	647.3	1486.3	-	794.6	1424.5	-	642.9	1193.5	-	800.9	**1172.1**	-	832.4	6609.4	-	594.9	-	-
RC208	572.7	1629.8	-	624.0	1776.1	-	573.9	**1039.5**	-	624.3	1078.4	-	647.7	6609.4	-	527.1	5299.5	-

Table 3. Solutions to the Solomon instances with 100 requests. The table reports the lower bound, upper bound and time to prove optimality for each of the solvers. The best upper bound for each instance is shown in bold. The CP model is omitted as it is unable to find feasible solutions to any instance.

Optimization Constraints. The objective function has been omitted from the checking subproblem because propagators for linear function are known to be weak. Sophisticated propagators for the WeightedCircuit constraint should be implemented, as they may produce considerably stronger nogoods.

Application to Branch-and-Price. Branch-and-cut-and-price, which includes column generation and cut generation, is the current state-of-the-art exact method for solving classical VRPs. Preliminary experiments with an existing branch-and-price solver show that BCE is not beneficial with column generation for the VRPTW as nogoods will not be generated in step 4 of Fig. 3 because the paths already respect the time and capacity constraints. However, step 2 can fail due to incompatibility between the branching decisions of a node. This infeasibility cannot be detected by the pricing problem because it has no knowledge of the global problem, nor detected by the master problem until all paths are generated because artificial variables satisfy the constraints in the interim. Incompatible branching decisions can induce nogoods but this seldom occurs in branch-and-price because its LP relaxation bound is asymptotically tight, allowing it to discard nodes due to suboptimality much earlier than infeasibility. Hence, branch-and-price-and-check is unlikely to prove useful in solving classical VRPs. It is, however, useful for rich VRPs with inter-route constraints (e.g., [20]) because the pricing subproblem, being a shortest path problem, has no knowledge of the interactions between routes in the parent problem.

7 Conclusion

This paper proposed the framework of branch-and-check with explanations (BCE) as a step towards the grand unification of linear programming, constraint programming and Boolean satisfiability. BCE finds cuts using general-purpose conflict analysis instead of specialized separation algorithms. The method features a master problem, which ignores a number of constraints, and a checking subproblem, which uses inference to check the feasibility of the omitted constraints and conflict analysis to derive nogood cuts. It also leverages conflict-based branching rules and can strengthen cuts using traditional insights from branch-and-cut in a post-processing step.

Experimental results on the Vehicle Routing Problem with Time Windows show that BCE is a viable alternative to branch-and-cut. In particular, BCE dominates branch-and-cut, both in proving optimality (with cut strengthening) and in finding high-quality solutions.

BCE offers an interesting alternative to existing branch-and-cut approaches. By using a general-purpose constraint programming solver to derive cuts, BCE can greatly simplify the modelling of problems that traditionally use branch-and-cut. This, in turn, avoids the need for dedicated separation algorithms. BCE is also capable of identifying well-known classes of cuts and strengthening them in a post-processing step. Finally, BCE significantly benefits from conflict-based branching rules, opening further opportunities typically not available in branch-and-cut.

References

1. Achterberg, T.: Conflict analysis in mixed integer programming. Discrete Optimization 4(1), 4 – 20 (2007)
2. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Operations Research Letters 33(1), 42 – 54 (2005)
3. Baldacci, R., Mingozzi, A., Roberti, R.: New route relaxation and pricing strategies for the vehicle routing problem. Operations Research 59(5), 1269–1283 (2011)
4. Bard, J.F., Kontoravdis, G., Yu, G.: A branch-and-cut procedure for the vehicle routing problem with time windows. Transportation Science 36(2), 250–269 (2002)
5. Beck, J.C.: Checking-up on branch-and-check. In: Cohen, D. (ed.) Principles and Practice of Constraint Programming – CP 2010, Lecture Notes in Computer Science, vol. 6308, pp. 84–98. Springer Berlin Heidelberg (2010)
6. Benchimol, P., Hoeve, W.J., Régin, J.C., Rousseau, L.M., Rueher, M.: Improved filtering for weighted circuit constraints. Constraints 17(3), 205–233 (2012)
7. Bent, R., Van Hentenryck, P.: A two-stage hybrid local search for the vehicle routing problem with time windows. Transportation Science 38(4), 515–530 (2004)
8. Bent, R., Van Hentenryck, P.: A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers & Operations Research 33(4), 875–893 (2006)
9. Dechter, R.: Learning while searching in constraint-satisfaction-problems. In: Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science. pp. 178–185 (1986)
10. Desrochers, M., Desrosiers, J., Solomon, M.: A new optimization algorithm for the vehicle routing problem with time windows. Operations Research 40(2), 342–354 (1992)
11. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003, Santa Margherita Ligure, Italy, May 5-8, 2003, Selected Revised Papers, pp. 502–518. Springer Berlin Heidelberg (2004)
12. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: Gent, I.P. (ed.) Principles and Practice of Constraint Programming - CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September 20-24, 2009 Proceedings. pp. 352–366. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
13. Fukasawa, R., Longo, H., Lysgaard, J., De Aragão, M.P., Reis, M., Uchoa, E., Werneck, R.F.: Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Mathematical Programming 106(3), 491 – 511 (2006)
14. Gendron, B., Scutellà, M.G., Garroppo, R.G., Nencioni, G., Tavanti, L.: A branch-and-benders-cut method for nonlinear power design in green wireless local area networks. European Journal of Operational Research 255(1), 151–162 (2016)
15. Hooker, J.: Logic-based methods for optimization. In: Borning, A. (ed.) Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, vol. 874, pp. 336–349. Springer Berlin Heidelberg (1994)
16. Jepsen, M., Petersen, B., Spoorendonk, S., Pisinger, D.: Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research 56(2), 497 – 511 (2008)
17. Jussien, N., Barichard, V.: The palm system: explanation-based constraint programming. In: Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000,. pp. 118–133 (2000)

18. Kallehauge, B., Boland, N., Madsen, O.B.G.: Path inequalities for the vehicle routing problem with time windows. Networks 49(4), 273–293 (2007)
19. Kilby, P., Prosser, P., Shaw, P.: A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. Constraints 5(4), 389–414 (2000)
20. Lam, E., Van Hentenryck, P.: A branch-and-price-and-check model for the vehicle routing problem with location congestion. Constraints 21(3), 394–412 (2016)
21. Lysgaard, J., Letchford, A.N., Eglese, R.W.: A new branch-and-cut algorithm for the capacitated vehicle routing problem. Mathematical Programming 100(2), 423–445 – 0025–5610 (2004)
22. Mak, V.: On the Asymmetric Travelling Salesman Problem with Replenishment Arcs. Ph.D. thesis, University of Melbourne (2001)
23. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th annual Design Automation Conference. pp. 530–535. ACM (2001)
24. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints 14(3), 357–391 (2009)
25. Pecin, D., Pessoa, A., Poggi, M., Uchoa, E.: Improved Branch-Cut-and-Price for Capacitated Vehicle Routing, pp. 393–403. Springer International Publishing (2014)
26. Ropke, S., Cordeau, J.F.: Branch and cut and price for the pickup and delivery problem with time windows. Transportation Science 43(3), 267–286 (2009)
27. Rousseau, L.M., Gendreau, M., Pesant, G.: Using constraint-based operators to solve the vehicle routing problem with time windows. Journal of Heuristics 8(1), 43–58 (2002)
28. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.F. (eds.) Principles and Practice of Constraint Programming — CP98, Lecture Notes in Computer Science, vol. 1520, pp. 417–431. Springer Berlin Heidelberg (1998)
29. Thorsteinsson, E.: Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In: Walsh, T. (ed.) Principles and Practice of Constraint Programming — CP 2001, Lecture Notes in Computer Science, vol. 2239, pp. 16–30. Springer Berlin Heidelberg (2001)
30. Tran, T.T., Araujo, A., Beck, J.C.: Decomposition methods for the parallel machine scheduling problem with setups. INFORMS Journal on Computing 28(1), 83–95 (2016)
31. Vigo, D., Toth, P.: Vehicle Routing: Problems, Methods, and Applications, Second Edition. Society for Industrial and Applied Mathematics (2014)