On the Sum Constraint: Relaxation and Applications

Tallys H. Yunes*

Graduate School of Industrial Administration
Carnegie Mellon University, Pittsburgh, PA 15213-3890, USA
tallys@cmu.edu

Abstract. The global constraint *sum* can be used as a tool to implement summations over sets of variables whose indices are not known in advance. This paper has two major contributions. On the theoretical side, we present the convex hull relaxation for the sum constraint in terms of linear inequalities, whose importance in the context of hybrid models is then justified. On the practical side, we demonstrate the applicability of the sum constraint in a scheduling problem that arises as part of the development of new products in the pharmaceutical and agrochemical industries. This problem can be modeled in two alternative ways: by using the sum constraint in a natural and straightforward manner, or by using the element constraint in a trickier fashion. With the convex hull relaxation developed earlier, we prove that the linear relaxation obtained from the former model is tighter than the one obtained from the latter. Moreover, our computational experiments indicate that the CP model based on the sum constraint is significantly more efficient as well.

1 Introduction

The global constraint *sum* can be used as a tool to implement summations over sets of variables whose indices are not known in advance. These are referred to as *variable index sets*. When studying a scheduling problem in the context of the development of new products in the agrochemical and pharmaceutical industries, we were faced with a subproblem where the sum constraint appears as a natural modeling candidate. We baptize this problem as the *Sequence Dependent Cumulative Cost Problem* (SDCCP), and we describe it in more detail in Sect. 2.1. The difficulties faced by previous attempts to solve the original main problem [5,9] indicate that a combination of Mathematical Programming and Constraint Programming methodologies might produce improved results.

The contributions of this paper are twofold. Firstly, from a hybrid modeling perspective, we give a first step toward better understanding the set of feasible solutions of the sum constraint by presenting its convex hull relaxation in terms of linear inequalities. Building on this result, we look at two alternative formulations of the SDCCP: one using the sum constraint and another using the

* Supported by the William Larimer Mellon Fellowship.

element constraint. We then show that the former is better than the latter in a well defined sense, i.e. its linear relaxation gives a tighter bound. Secondly, on the purely Constraint Programming side, our computational results for the SDCCP indicate that a model that uses the sum constraint also has the advantage of being more natural, more concise and computationally more efficient than the alternative model with the element constraint alone.

The remainder of this text is organized as follows. In Sect. 2, we describe the semantics of the sum constraint and we present a real-world example of its applicability. In Sect. 3, we analyze the importance of having a linear relaxation for the sum constraint, and for any global constraint in general, from the viewpoint of a hybrid modeling paradigm. Given this motivation, in Sect. 4 we present the best possible linear relaxation of the sum constraint, i.e. its convex hull. Within the idea of obtaining better dual bounds for optimization problems addressed by a combination of solvers, it then makes sense to assess the relative strengths of relaxations provided by alternative models of a given problem. Section 5 presents such a comparison for the SDCCP. Computational results on some randomly generated instances of that problem are given in Sect. 6. Finally, Sect. 7 summarizes our main conclusions.

2 The Sum Constraint and Its Applications

The main purpose of the sum constraint is to implement variable index sets, as defined below.

Definition 1. *Let $S_{j_1}, \ldots, S_{j_d}$ be sets of indices in $\{1, \ldots, n\}$, and let $c_1, \ldots, c_n$ be constants. If y is a variable with domain $D_y = \{j_1, \ldots, j_d\}$, the constraint sum$(y, (S_{j_1}, \ldots, S_{j_d}), (c_1, \ldots, c_n), z)$ states that*

$$z = \sum_{j \in S_y} c_j \ . \tag{1}$$

That is, the c_i's on the right hand side of (1) are not know a priori. They are function of the variable y. An extended version of this constraint exists where each constant c_i is replaced by a variable x_i. One can think of the well known element constraint as a special case of the sum constraint in which we have $S_j = \{j\}$ for all $j \in \{1, \ldots, n\}$. Algorithms for achieving hyper arc consistency for the sum constraint with constant terms, and a slightly weaker version of bounds consistency for the case with variable terms are presented in [6].

The next section introduces a scheduling problem that turns out to be a natural application of the sum constraint.

2.1 The Sequence Dependent Cumulative Cost Problem

Let us define the *Sequence Dependent Cumulative Cost Problem* (SDCCP) as follows. Suppose we are given a set of n tasks that have to be scheduled so that each task is assigned a unique and distinct position in the set $\{1, \ldots, n\}$.

Let q_i be the position assigned to task $i \in \{1, \ldots, n\}$, and let p_j contain an arbitrary value associated to the task assigned to position j. We are interested in computing

$$v_i = \sum_{1 \leq j < q_i} p_j \ , \ \forall\, i \in \{1, \ldots, n\} \ . \tag{2}$$

Two possible ways of computing v_i are: by using variable subscripts

$$v_i = \sum_{j=1}^{n} p_{\max\{0,\, q_i - q_j\}} \qquad (\text{with } p_0 = 0) \tag{3}$$

or by using variable index sets in a more natural way

$$v_i = \sum_{j=1}^{q_i - 1} p_j \ . \tag{4}$$

Apart from these constraints, we also have upper and lower bounds on the value of q_i, for every i. In job-shop scheduling terminology, these can be thought of as release dates and due dates. Finally, we are interested in minimizing $\sum_{i=1}^{n} c_i v_i$, where c_i is the cost of performing task i.

This problem appears in the context of scheduling safety/quality tests under resource constraints. It is part of the *New Product Development Problem* (NPDP), which arises in the pharmaceutical and agrochemical industries and has been studied by many authors [3,5,9,10]. In the NPDP, a given set of products (e.g. newly created drugs) have to pass a series of tests enforced by law before being allowed to reach the market. A stochastic component is present in the sense that each test has a probability of failure. In case one of the tests for a certain product fails, all subsequent tests for that same product are canceled and rescheduling is often necessary in order to make better use of the available resources. The expected cost of test i is given by the probability of executing the test times the cost of performing it. In turn, this probability equals the product of the success probabilities of every test that finishes before test i starts. Another relevant issue is that we usually have technological precedences among the tests, which enforce an initial partial sequencing and contribute to the initial lower and upper bounds on the q_i variables, as mentioned earlier. The objective function of the NPDP seeks to minimize the total cost of performing all the required tests, while satisfying constraints on the number of available laboratories and capacitated personnel. The total cost also takes into account other factors such as a decrease in income due to delays in product commercialization, and the possibility of outsourcing the execution of some tests at a higher price. As reported in [9], the NPDP is a very hard problem to solve, even when the number of products is small.

Although the objective function of the NPDP turns out to be nonlinear, Jain and Grossmann [9] show that it can be accurately approximated by a piecewise linear function after using a logarithmic transformation. In this approximation, which we are going to consider here, one needs to calculate, for each test i, an expression of the form (2), where p_j equals the logarithm of the success probability of the test assigned to position j. As was done in [9], an immediate Mixed Integer Programming (MIP) formulation of (2) would look like

$$v_i = \sum_{j=1,\, j \neq i}^{n} y_{ji} p_j \ , \forall\, i \in \{1, \ldots, n\} \ , \tag{5}$$

where $y_{ji} = 1$ if test j precedes test i and $y_{ji} = 0$ otherwise. We are unaware of any previous work that addresses the SDCCP alone via an MIP approach. In this paper, we will only use CP to tackle the SDCCP and we will not explicitly evaluate MIP models for it.

3 The Logic-Based Modeling Paradigm

The concept of logic-based modeling is far more general and powerful than what is presented in this paper. In [6], Hooker gives a lot of illustrative examples and develops the topic to a large extent. The basic philosophy of this paradigm is very similar to the one behind the Branch-and-Infer [4], Mixed Logical/Linear Programming [7] or Branch-and-Check [11] frameworks. That is, a problem can be modeled with the Mixed Integer Programming (MIP) language of linear inequalities and also with the more expressive language of Constraint Programming (CP). Then, each type of constraint is handled by its specific solver in a collaborative effort to find an optimal solution to the original problem. That collaboration can be done in a myriad of ways and the best choice will depend on the structure of the problem being studied.

In real world situations, one can usually identify substructures or smaller parts of a problem that possess nice or well studied properties. These can even be parts of other known problems that, when put together, define the problem under consideration. When trying to write an MIP model, the modeler is aware of the structure of the problem and how its different parts are connected. However, given the limited expressive power of the vocabulary of linear inequalities, much of this structure will be lost during the translation phase. Many state-of-the-art MIP softwares have algorithms that try to identify some structure inside the linear programs in order to take advantage of them. For example, they can try to use some valid cuts that have been developed for that particular class of problems and improve the quality of the Linear Programming (LP) relaxation, or even apply a different, more specialized, algorithm such as the Network Simplex. But this is not always effective. The more localized view of the problem substructures is usually lost or becomes unrecognizable, and the solver can only work with the global picture of a linear (integer) program. Moreover, a significant portion of the cutting plane theory developed so far is not a default component of current MIP packages.

On the other hand, the Constraint Programming (CP) community has done a lot of work on special purpose algorithms tailored to solve Constraint Satisfaction Problems [12] through local inference inside global constraints.

Looking at the CP and Operations Research (OR) worlds, it is not hard to see that they can benefit from each other if the local and global views are combined properly. In the logic-based modeling framework, the modeler is able to write the problem formulation with both linear inequalities and more expressive global constraints or logic relations that can better capture and exploit some local structure in the problem. Obviously, this hybrid formulation needs both an LP solver and a constraint solver in order to be useful. But, in principle, this would be transparent to the end-user. In essence, the whole mechanism works as follows. The linear constraints in the logic-based formulation are posted to the LP solver, as usual, and some of them may also be posted to the constraint solver. The constraint solver handles the constraints that cannot be directly posted to the LP solver (e.g. global constraints). When applicable, the continuous relaxation of some global constraints is also sent to the LP solver so as to strengthen the overall relaxation, providing better dual bounds. These linear relaxations of global constraints are sometimes referred to as *dynamic linear relaxations*, because they are supposed to change according to the way the search procedure evolves. This idea also plays a key role behind increasing the performance of hybrid decomposition procedures like Benders decomposition [2] or, more generally, Branch-and-Check, as argued by Thorsteinsson [11].

It is important to notice that some global constraints have known continuous relaxations that can be added to the set of linear inequalities that are sent to the LP solver. Nevertheless, there are also global constraints for which a continuous relaxation is either not known or too large for practical purposes. In these cases, when building the linear relaxation of the entire logic-based formulation, these global constraints are simply removed and the LP solver ignores their existence. The overall linear relaxation derived from a logic-based formulation is often smaller and weaker than what would be obtained from a pure MIP formulation. On one hand, this allows for a faster solution of the associated linear programs. On the other hand, if we make use of an implicit enumeration algorithm like branch-and-bound, the weaker bounds can result in a larger search tree. Domain reductions resulting from constraint propagation at each node of the tree may help compensate for that.

Given this motivation, the next two sections develop the strongest possible linear relaxation for the sum constraint and theoretically assess its effectiveness when applied to the SDCCP.

4 The Convex Hull Relaxation of the Sum Constraint

When dealing with linear optimization problems over disjunctions of polyhedra (see [1]), the convex hull relaxation is the best that one can hope for. More precisely, if Q is the union of a number of nonempty polyhedra and $\max\{f(x) : x \in Q\}$ is attained at an extreme point x^* of Q, there exists an extreme point x' of the convex hull of Q such that $f(x') = f(x^*)$.

From now on, we assume that $z \geq 0$, $x_i \geq 0$ for all $i \in \{1, \ldots, n\}$, and y is an integer variable whose domain has size $|D_y| = d$. For each $j \in D_y$, $S_j \subseteq$

$\{1, \ldots, n\}$ is a set of indices. We will denote by $\text{conv}(Q)$ the convex hull of an arbitrary set $Q \in \mathbb{R}^m$, for any $m \in \mathbb{N}$. Also, x and $\bar{x}$ are n-dimensional vectors whose elements are referred to by subscripts, e.g. x_i and $\bar{x}_i$.

4.1 The Constant Case

For all $i \in \{1, \ldots, n\}$, let c_i be integer constants. Then, the sum constraint

$$\text{sum}(y, (S_{j_1}, \ldots, S_{j_d}), (c_1, \ldots, c_n), z) \tag{6}$$

represents the disjunction

$$\bigvee_{j \in D_y} \left(z = \sum_{i \in S_j} c_i \right) \quad . \tag{7}$$

Proposition 1. *The convex hull of (7) is given by*

$$\min_{j \in D_y} \left\{ \sum_{i \in S_j} c_i \right\} \le z \le \max_{j \in D_y} \left\{ \sum_{i \in S_j} c_i \right\} \quad .$$

Proof. For each $j \in D_y$, let $t_j = \sum_{i \in S_j} c_i$. Notice that (7) is a one-dimensional problem in which z can take one of $|D_y|$ possible values $t_j \in \mathbb{R}$. The convex hull of these points is simply the line segment connecting all of them. □

4.2 The Variable Case

The interesting case is when the list of constants is replaced by a list of variables. This other version of the sum constraint, $\text{sum}(y, (S_{j_1}, \ldots, S_{j_d}), (x_1, \ldots, x_n), z)$, represents the disjunction

$$\bigvee_{j \in D_y} \left(z = \sum_{i \in S_j} x_i \right) \quad . \tag{8}$$

Lemma 1. *Let $I = \bigcap_{j \in D_y} S_j$ and, for every $j \in D_y$, let $S'_j = S_j \setminus I$. If $I \neq \emptyset$, (8) is the projection of (9)–(10) onto the space of z and x. Moreover, the convex hull of (9)–(10) is given by (9) and the convex hull of (10), together with the non-negativity constraints $z \ge 0$, $x \ge 0$ and $w \ge 0$.*

$$z = \sum_{i \in I} x_i + w \tag{9}$$

$$\bigvee_{j \in D_y} \left(w = \sum_{i \in S'_j} x_i \right) \quad . \tag{10}$$

Proof. Clearly, for any point $(\bar{z}, \bar{x}) \in \mathbb{R}^{1+n}$ satisfying (8), it is easy to find a $\bar{w} \in \mathbb{R}$ such that $(\bar{z}, \bar{x}, \bar{w}) \in \mathbb{R}^{1+n+1}$ satisfies (9)–(10). For instance, if $(\bar{z}, \bar{x})$ satisfies the r^{th} disjunct in (8) (i.e. the one with $j = r$), take $\bar{w} = \sum_{i \in S'_r} x_i$. Conversely, let $(\bar{z}, \bar{x}, \bar{w})$ satisfy (9) and the r^{th} disjunct in (10). Then, $(\bar{z}, \bar{x})$ satisfies the r^{th} disjunct in (8). For the last part, let conv(10) denote the convex hull of (10). Also, let A be the set of points in $\mathbb{R}^{1+n+1}$ defined by (9)–(10) and let B be the set of points in $\mathbb{R}^{1+n+1}$ defined by the intersection of (9) and conv(10). We want to show that $\text{conv}(A) = B$.

$\text{conv}(A) \subseteq B$: Any point in A is built in the following way: pick a point $(\bar{x}, \bar{w})$ that satisfies one of the disjuncts in (10) and then set $\bar{z} = \sum_{i \in I} \bar{x}_i + \bar{w}$. Any point $p_3 \in \text{conv}(A)$ can be written as $p_3 = \lambda p_1 + (1 - \lambda) p_2$, for some $\lambda \in [0, 1]$ and some $p_1 = (\bar{z}_1, \bar{x}^1, \bar{w}_1)$ and $p_2 = (\bar{z}_2, \bar{x}^2, \bar{w}_2)$ from A, built as indicated before. To see that $p_3 = (\bar{z}_3, \bar{x}^3, \bar{w}_3) \in B$, notice that $p_3 \in$ conv(10) because both p_1 and p_2 satisfy (10). Finally, p_3 also satisfies (9) because $\bar{z}_3 = \lambda \bar{z}_1 + (1 - \lambda) \bar{z}_2 = \sum_{i \in I} \bar{x}^3_i + \bar{w}_3$.

$B \subseteq \text{conv}(A)$: Any point in conv(10) can be written as $(\bar{z}, \bar{x}, \bar{w}) = \lambda(\bar{z}_1, \bar{x}^1, \bar{w}_1) + (1 - \lambda)(\bar{z}_2, \bar{x}^2, \bar{w}_2)$, where $p_1 = (\bar{z}_1, \bar{x}^1, \bar{w}_1)$ and $p_2 = (\bar{z}_2, \bar{x}^2, \bar{w}_2)$ satisfy some disjuncts in (10). When $\bar{z} = \sum_{i \in I} \bar{x}_i + \bar{w}$, then $p = (\bar{z}, \bar{x}, \bar{w}) \in B$. To see that $p \in \text{conv}(A)$, simply notice that p_1 and p_2 can always be chosen with $\bar{z}_1 = \sum_{i \in I} \bar{x}^1_i + \bar{w}_1$ and $\bar{z}_2 = \sum_{i \in I} \bar{x}^2_i + \bar{w}_2$, i.e. points in A. □

Before stating the main theorem of this section, we mention an auxiliary result that can be easily proved with standard arguments from Convex Analysis.

Lemma 2. *Let S be an arbitrary set in $\mathbb{R}^{\ell+m}$ and let $\text{Proj}_\ell(S)$ be the projection of S onto the $\mathbb{R}^\ell$ space. Then, $\text{Proj}_\ell(\text{conv}(S)) = \text{conv}(\text{Proj}_\ell(S))$.*

Theorem 1. *Let I and S'_j, for all $j \in D_y$, be defined as in Lemma 1. Let $U = \bigcup_{j \in D_y} S'_j$. The convex hull of (8) is given by the projection of (9) and (11) onto the space of z and x, with $z \geq 0$ and $x \geq 0$.*

$$0 \leq w \leq \sum_{i \in U} x_i \quad . \tag{11}$$

Proof. By lemmas 1 and 2, it suffices to show that (11) is the convex hull of (10). Clearly, every point that satisfies (10) also satisfies (11). To complete the proof, we need to show that any point that satisfies (11) is a convex combination of points satisfying (10). Let $(\bar{x}, \bar{w})$ satisfy (11), and let $K = |U| + 1$. The role of K is to ensure that the sum of the multipliers in (12) is equal to 1. From (11), there exists $\alpha \in [0, 1]$ such that $\bar{w} = \alpha \sum_{i \in U} \bar{x}_i$. We can write $(\bar{x}, \bar{w})$ as

$$\begin{pmatrix} \bar{x} \\ \bar{w} \end{pmatrix} = \frac{\alpha}{K} \sum_{i \in U} \begin{pmatrix} K\bar{x}_i e^i \\ K\bar{x}_i \end{pmatrix} + \frac{(1 - \alpha)}{K} \sum_{i \in U} \begin{pmatrix} K\bar{x}_i e^i \\ 0 \end{pmatrix} + \frac{1}{K} \begin{pmatrix} K\bar{u} \\ 0 \end{pmatrix} \quad , \tag{12}$$

where e^i is the i^{th} unit vector, and $\bar{u}_i = \bar{x}_i$ for every $i \in \{1, \ldots, n\} \setminus U$ and $\bar{u}_i = 0$ otherwise. Notice that every point $(K\bar{x}_i e^i, K\bar{x}_i)$ satisfies the j^{th} disjunct for some j such that $i \in S'_j$. Also, since $\bigcap_{j \in D_y} S'_j = \emptyset$, for every $i \in U$ there exists a disjunct k that is satisfied by the point $(K\bar{x}_i e^i, 0)$. Namely, any k such that $i \notin S'_k$. Finally, $(K\bar{u}, 0)$ trivially satisfies any disjunct in (10) by construction.

□

After Fourier-Motzkin elimination of w, we get

$$\sum_{i \in I} x_i \le z \le \sum_{i \in I \cup U} x_i \ .$$

For the special case when $I = \emptyset$, we have $z = w$ and the previous proof shows that the convex hull of (8) is given by (11) with w replaced by z, and the non-negativity constraints $z \ge 0$ and $x \ge 0$.

5 Comparing Alternative Formulations for the SDCCP

Some parts of the NPDP described in Sect. 2.1 present clearly recognizable substructures that could be explored in the context of a logic-based modeling framework. For instance, besides the SDCCP, it is possible to model the influence of the resource limitations on the final schedule of tests by using the global constraint cumulative in a very natural way. In this paper, however, we will only concentrate on the role of the sum constraint as a tool to model the SDCCP.

Let $p_0 = 0$. We can implement the variable subscripts in (3) with (13)–(15).

$$y_{ij} = q_i - q_j + n, \ \ \forall\, j = 1, \dots, n, \ \ j \ne i \tag{13}$$

$$\text{element}(y_{ij}, [\overbrace{p_0, \dots, p_0}^{n \text{ times}}, p_1, \dots, p_{n-1}], z_{ij}), \ \ \forall\, j = 1, \dots, n, \ \ j \ne i \tag{14}$$

$$v_i = \sum_{j=1,\, j \ne i}^{n} z_{ij} \ . \tag{15}$$

From (13), the domain of y_{ij} is $D_{y_{ij}} \subseteq \{1, \dots, 2n-1\}$. The values between 1 and n represent the situations when $q_i \le q_j$, which are of no interest. That is why the first n variables in the second argument of (14) are set to zero. The variable index sets in (4) can be implemented as

$$\text{sum}(q_i, [\{1\}, \{2\}, \{2,3\}, \{2,3,4\}, \dots, \{2, \dots, n\}], [p_0, p_1, \dots, p_{n-1}], v_i) \ . \tag{16}$$

The next result states that, from the viewpoint of a Linear Programming relaxation, we only need to consider the variable index set formulation (16).

Theorem 2. *For each $i \in \{1, \dots, n\}$, let the initial domain of q_i be $D_i = \{1, \dots, n\}$. If we impose the constraint alldifferent$(q_1, \dots, q_n)$, the bounds on v_i given by the relaxation of (16) are at least as tight as the bounds given by the relaxation of (13)–(15).*

To prove this theorem, we need an auxiliary result that follows from the pigeon-hole principle.

Lemma 3. *Let $A = \{a_1, \dots, a_k\} \subseteq \{1, \dots, n\}$, and let $q_1, \dots, q_n$ have initial domains $D_1 = \cdots = D_n = \{1, \dots, n\}$, respectively. When we require the constraint alldifferent$(q_1, \dots, q_n)$, there exist at least k distinct variables $q_{i_1}, \dots, q_{i_k}$ such that $a_1 \in D_{i_1}, \dots, a_k \in D_{i_k}$.*

Proof of Theorem 2. The convex hull relaxation of (14) gives $0 \leq z_{ij} \leq \sum_{k \in D_{y_{ij}},\, k \geq n} p_{k-n}$, for all $j = 1, \dots, n$, $j \neq i$ (see [6] for a proof). Therefore, we can write

$$0 \leq v_i \leq \sum_{j=1}^{n} \left(\sum_{k \in D_{y_{ij}},\, k \geq n} p_{k-n} \right) . \tag{17}$$

Let $S_1 = \{0\}$ and $S_j = \{1, \dots, j-1\}$, for all $j = 2, \dots, n$. As before, let $U = \bigcup_{j \in D_i} S_j$. By Theorem 1, the convex hull relaxation of (16) is given by

$$0 \leq v_i \leq \sum_{k \in U} p_k . \tag{18}$$

We want to show that the RHS of (18) is always less than or equal to the RHS of (17). We divide the proof in 3 sub-cases.

$q_i = 1:$ Clearly, both (17) and (18) give $v_i = 0$.

$q_i = b > 1:$ In this case, the RHS of (18) reduces to $\sum_{k=1}^{b-1} p_k$. Notice that, for any j, $D_{y_{ij}}$ will contain values larger than n if and only if D_j contains values smaller than b. But, by Lemma 3, for every number $a \in \{1, \dots, b-1\}$ there exists at least one variable q_j such that $a \in D_j$. Hence, the RHS of (17) reduces to $\sum_{k=1}^{b-1} c_k p_k$, where $c_k \geq 1$.

$|D_i| = d \geq 2:$ Let $D_i = \{b_1, \dots, b_d\}$, with $b_1 < \cdots < b_d$. The RHS of (18) reduces to $\sum_{k=1}^{n} c_k p_k$, where $c_k = 1$ if $k \in U$, and $c_k = 0$ otherwise. We will show that the RHS of (17) reduces to $\sum_{k=1}^{n} \bar{c}_k p_k$, with $\bar{c}_k \geq c_k$ for every $k = 1, \dots, n$. Let us start by calculating $\bar{c}_1$, that is, the number of variables q_j for which $n+1 \in D_{y_{ij}}$. Notice that, for every j, $n+1 \in D_{y_{ij}}$ if and only if D_j contains at least one of $b_1 - 1, b_2 - 1, \dots, b_d - 1$. By Lemma 3, there exist at least d such D_j's ($d-1$ if $b_1 = 1$). Hence, $\bar{c}_1 \geq d$ ($d-1$ if $b_1 = 1$). Analogously, for $k = 2, \dots, b_1 - 1$, we want to know how many distinct D_j's contain at least one of the numbers $b_1 - k, b_2 - k, \dots, b_d - k$. This will be equal to the number of distinct $D_{y_{ij}}$'s that contain the value $n + k$. By the same reasoning, we have that $\bar{c}_k \geq d$ whenever $1 \leq k < b_1$. If $b_1 \leq k < b_2$, $b_1 - k \leq 0$ and we need only consider the $d-1$ positive numbers $b_2 - k, b_3 - k, \dots, b_d - k$. Again, there are at least $d-1$ distinct q_j variables whose domains contain at least one of these numbers. So, $\bar{c}_k \geq d-1$ for $k = b_1, \dots, b_2 - 1$. We can now repeat this argument until $k = b_d - 1$. □

To see that (18) can be strictly stronger than (17), let us consider the following example. Let $n = 3$ and $q_1, q_2, q_3 \in \{1, 2, 3\}$. Then, when we look at v_1, y_{12}, $y_{13} \in \{1, \dots, 5\}$. From (17) we get $v_1 \leq 2p_1 + 2p_2$, whereas from (18) we get the inequality $v_1 \leq p_1 + p_2$, which strictly dominates the previous one, since p_1 and p_2 are non-negative.

6 Implementation of the Alternative Models

In order to describe the two CP models for the SDCCP, we will recall some of the notation introduced in Sect. 2.1. Let prob(i) denote the probability of success of task $i \in \{1, \ldots, n\}$. Since we have been assuming that all variables are non-negative, the variable p_j will represent $-\log(\text{prob}(\text{task assigned to position } j))$, for every $j \in \{1, \ldots, n\}$. In this fashion, we can state the SDCCP as a maximization problem with $v_i \geq 0$ for each task i. Let q_i represent the position assigned to task i and let c_i be the cost of performing it. The CP formulation of the SDCCP can be written as

$$\max \sum_{i=1}^{n} c_i v_i \tag{19}$$

$$\text{element}(q_i, [p_1, \ldots, p_n], -\log(\text{prob}(i))), \ \ \forall\, i \in \{1, \ldots, n\} \tag{20}$$

$$< \text{relationship between } p_i,\ q_i \text{ and } v_i > \tag{21}$$

$$L_i \leq q_i \leq U_i, \ \ \forall\, i \in \{1, \ldots, n\} \ \ , \tag{22}$$

where L_i and U_i are, respectively, lower and upper bounds on the value of q_i.

The two alternative models differ in the way they represent constraints (21). In the element constraint model (Model 1), we replace (21) by (13)–(15). In the sum constraint model (Model 2), we replace (21) by (16), for all $i \in \{1, \ldots, n\}$.

Using the ECL^{i}PS^e [8] constraint logic programming system version 5.3, we implemented an algorithm that achieves a slightly weaker version of bounds consistency for the sum constraint with variable terms. We also implemented a hyper arc consistency algorithm for the version of the element constraint that indexes a list of variables, as needed for (20). For the details regarding the propagation algorithms and the strategies for incremental maintenance, we refer the reader to [6]. The computational times reported in this section are given in CPU seconds of a Sun UltraSPARC-II 360MHz running SunOS 5.7.

6.1 Computational Results

To test the performance of the two models described above, we generated random instances of the SDCCP[1]. The size of the instances n, given as the number of tasks, varies from 5 to 10. The values of c_i are randomly picked from a discrete uniform distribution in the interval $[10, 100]$. To control the percentage of tasks with uncertainty in the completion, as well as the percentage of tasks with release dates and due dates, we divided the instances in four classes. Each class is characterized by two numbers a and b that denote, respectively, the percentage of tasks with prob(i) < 1, and the percentage of tasks with non-trivial release dates and due dates. The (a, b) pairs chosen for our experiments are: (33%, 33%), (50%, 33%), (50%, 50%) and (75%, 50%). The rationale behind this choice is to try to evaluate the behavior of the models under different problem configurations.

[1] The instance generator can be made available upon request.

Table 1. Comparison of the two CP models for the SDCCP (33%, 33%)

		Model 1		Model 2	
Size	Optimum	Backtracks	Time	Backtracks	Time
5	312.88	2	0.10	2	0.04
6	427.04	14	1.58	7	0.37
7	572.93	67	7.75	36	1.22
8	1,163.76	149	79.10	63	8.18
9	1,183.07	1,304	518.38	328	44.33
10	1,102.39	31,152	6,293.25	4,648	308.60

For example, for problems of size 6 in class (50%, 33%), 50% of the tasks have prob(i) chosen randomly from a uniform distribution in the interval $[0, 1]$. The other half of the prob(i) values are set to 1. Also, for 33% of the tasks, L_i and U_i are randomly chosen from a discrete uniform distribution in the interval $[1, 6]$. For the remaining 67% of the tasks, $L_i = 1$ and $U_i = 6$.

For each one of the four classes of instances, the numbers reported in tables 1 through 4 are the average values over 10 different runs for each instance size ranging from 5 to 9, and average values over 3 runs for instances of size 10.

Our results indicate that, as the instance size increases, Model 1, which is based on the element constraint, requires a significantly larger number of backtracks than Model 2, which uses the sum constraint. In terms of computational time, solving Model 2 tends to be roughly one order of magnitude faster than solving Model 1. One reason for this behavior can be attributed to the fact that Model 2, besides being more compact, also provides more efficient pruning, since Model 1 is essentially simulating the sum constraint with a number of element constraints.

Table 2. Comparison of the two CP models for the SDCCP (50%, 33%)

		Model 1		Model 2	
Size	Optimum	Backtracks	Time	Backtracks	Time
5	180.39	4	0.15	3	0.05
6	367.76	18	1.07	10	0.24
7	898.01	41	6.25	24	1.06
8	1,435.72	303	52.47	91	4.96
9	1,595.03	1,803	187.27	881	19.41
10	1,505.62	67,272	8,605.18	15,345	561.90

Table 3. Comparison of the two CP models for the SDCCP (50%, 50%)

		Model 1		Model 2	
Size	Optimum	Backtracks	Time	Backtracks	Time
5	330.01	3	0.13	3	0.04
6	444.10	7	0.67	5	0.17
7	995.88	28	6.44	14	0.94
8	1,488.54	121	68.04	48	6.28
9	995.72	1,216	293.11	358	31.74
10	2,207.05	973	524.69	121	21.15

Table 4. Comparison of the two CP models for the SDCCP (75%, 50%)

		Model 1		Model 2	
Size	Optimum	Backtracks	Time	Backtracks	Time
5	670.65	5	0.20	4	0.05
6	825.83	8	0.47	4	0.10
7	1,241.32	46	4.05	23	0.59
8	1,713.76	96	24.19	46	3.14
9	2,346.90	1,943	233.54	869	19.86
10	2,512.04	4,025	1,794.19	1,311	155.76

7 Conclusions

We study a scheduling problem that appears as a substructure of a real-world problem in the agrochemical and pharmaceutical industries called the New Product Development Problem (NPDP) [5,9]. We refer to that subproblem as the Sequence Dependent Cumulative Cost Problem (SDCCP).

Given the description of the SDCCP, the sum constraint is identified as a natural tool, simplifying the modeling effort and making it more intuitive. Apart from these advantages, our computational experiments show that the sum constraint exhibits a significantly improved performance over an alternative model that uses the element constraint. From a more theoretical point of view, we study the characteristics of the set of feasible solutions of the sum constraint. We provide the representation of the convex hull of this set of feasible solutions and we prove that this linear relaxation gives tighter bounds than the linear relaxation of the aforementioned alternative model. The relevance of this result is related to hybrid modeling efforts. One promising idea is to use linear relaxations of global constraints to help in the process of solving optimization problems through a combination of solvers. These linear relaxations, or at least some of their valid inequalities, may contribute to speed up the solution process by improving the bounds on the optimal solution value.

As an extension of this work, one can try to look at different contexts in which the sum constraint also fits well as a modeling device, and then compare its performance against other modeling possibilities. Finally, the impact of modeling the SDCCP with the sum constraint while solving the more general NPDP is another topic that deserves further investigation.

Acknowledgments

The author would like to thank Professor John N. Hooker for his many helpful comments during the development of this work.

References

1. E. Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89:3–44, 1998. 84
2. J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962. 84
3. G. Blau, B. Mehta, S. Bose, J. Pekny, G. Sinclair, K. Keunker, and P. Bunch. Risk management in the development of new products in highly regulated industries. *Computers and Chemical Engineering*, 24:659–664, 2000. 82
4. 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. 83
5. S. Honkomp, G. Reklaitis, and J. Pekny. Robust planning and scheduling of process development projects under stochastic conditions. Presented at the AICHE annual meeting, Los Angeles, CA, 1997. 80, 82, 91
6. J. N. Hooker. *Logic-Based Methods for Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization, 2000. 81, 83, 88, 89
7. J. N. Hooker and M. A. Osorio. Mixed logical/linear programming. *Discrete Applied Mathematics*, 96–97(1–3):395–442, 1999. 83
8. IC-Parc, Imperial College, London. The ECL^{i}PS^e Constraint Logic Programming System. http://www.icparc.ic.ac.uk/eclipse. 89
9. V. Jain and I. Grossmann. Resource-constrained scheduling of tests in new product development. *Industrial and Engineering Chemistry Research*, 38(8):3013–3026, 1999. 80, 82, 83, 91
10. C. Schmidt and I. Grossmann. Optimization models for the scheduling of testing tasks in new product development. *Industrial and Engineering Chemistry Research*, 35(10):3498–3510, 1996. 82
11. E. S. Thorsteinsson. Branch-and-Check: A hybrid framework integrating mixed integer programming and constraint logic programming. In Toby Walsh, editor, *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, pages 16–30. Springer-Verlag, November 2001. 83, 84
12. E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993. 83