INFORMS Journal on Computing
Vol. 00, No. 0, Xxxxx 2006, pp. 1–13
ISSN 1091-9856 | EISSN 1526-5528 | 06 | 0000 | 0001

DOI 10.1287/ijoc.1050.0168

Discrepancy-Based Additive Bounding Procedures

Andrea Lodi, Michela Milano
D.E.I.S., University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
{alodi@deis.unibo.it, mmilano@deis.unibo.it}

Louis-Martin Rousseau
Centre for Research on Transportation, Université de Montréal, CP 6128 Succ. Centre-Ville,
Montréal, Canada H3C 3J7, louism@crt.umontreal.ca

We model portions of the search tree via so-called *search constraints*. We focus on a particular kind of search constraint, the *k*-discrepancy constraint appearing in discrepancy-based search. The property that a node has an associated discrepancy *k* can be modeled (and enforced) through a linear constraint. Our key result is the exploitation of the *k*-discrepancy constraint to improve the bound given by any relaxation of a combinatorial optimization problem through the *additive bounding technique* (Fischetti and Toth 1989). We show how this simple idea can be effectively exploited to tighten relaxations in CP solvers and speed up the proof of optimality by performing a large variety of computational experiments on test problems involving the *AllDifferent* constraint. In this view, the additive bounding technique represents a non-trivial link between search and bound. Moreover, such a technique is general because it does not depend on either the *AllDifferent* constraint or the discrepancy search technique.

Key words: limited discrepancy search; additive bounding; relaxations; constraint programming
History: Accepted by John Hooker, Area Editor for Constraint Programming and Optimization; received November 2003; revised June 2005; accepted September 2005.

1. Introduction

It is widely recognized that integrating relaxations in *constraint programming* (CP) solvers leads to improved performance in optimization problems. In particular, relaxations can be used to obtain bounds on the objective-function value, for cost-based filtering (Focacci et al. 1999), and for guiding search (e.g., El Sakkout and Wallace 2000, Focacci 2001, and Milano and van Hoeve 2002, among others). This integration enables one to prune provably sub-optimal branches, thus performing optimality reasoning in CP. The efficiency of optimality reasoning depends largely on the quality of available bounds. If we have several bounding techniques for a given problem, we can simply select the tightest one. However, in this way we exploit only one relaxation structure.

The *additive bounding procedure* (ABP) partially overcomes this problem. ABP is a general technique for computing accurate bounds for combinatorial problems. It has been proposed by Fischetti and Toth (1989) and effectively applied to the asymmetric traveling salesman problem (Fischetti and Toth 1992).

Roughly speaking, given a set of bounding procedures for a problem *P*, one can apply them in sequence so that the *information* provided by a procedure is used as an input for the next one. Then, the sum of the solution values of all the procedures is a valid bound for *P*. The only requirement is that each bounding procedure must provide a reduced cost vector used as the cost vector by the next one.

In this paper we show how to exploit this remarkable technique when *limited discrepancy search* (LDS, Harvey and Ginsberg 1995) and domain splitting are used. Given a heuristic that provides a ranking of values in the variables' domain, LDS is an effective search strategy that "trusts" the heuristic. More precisely, a *discrepancy* is a branching decision that does not follow the suggestion of the heuristic. Thus, the LDS strategy explores the overall tree so that subtrees with lower discrepancy are considered first. LDS is currently available in many commercial CP solvers since it has proven to be effective in practice.

We have applied LDS together with domain splitting (e.g., Focacci 2001). At each node, a variable is selected and its domain is partitioned into two parts according to a ranking given by the heuristic: the *good* domain part containing promising values (suggested by the heuristic) and the *bad* domain part (containing the remaining values). In the resulting search tree, the property of a leaf to have an associated discrepancy value *k* can be modeled (and enforced) through a constraint, called the *k-discrepancy constraint*. Its declarative semantics state that exactly *k* variables assume a value in their corresponding bad domain.

The key idea is to use the *k*-discrepancy constraint and the additive bounding procedure together

to accelerate the proof of optimality. Such an idea can be interpreted as a *link between search and bound*, i.e., the exploitation of the structure of the search tree for tightening the bound. Such a link is generally exploited trivially: when the domain of a variable is reduced to a single value, then the bound is re-computed. In fact, we aim at showing that stronger links result in stronger propagation. In this paper, we give computational evidence that the idea is also useful in practice by considering test problems in which the widely used *AllDifferent* constraint is part of the CP model. A classical combinatorial relaxation of such a constraint is the famous linear *assignment problem* for which effective special-purpose algorithms providing reduced costs are available. However, the proposed approach is not limited to *AllDifferent*, and any combinatorial relaxation of a global constraint can be used together with LDS.

On the other hand, the additive bounding technique could be applied in CP in a variety of different ways. In particular, the CP modeling framework is based on overlapping global constraints, so it immediately comes to mind that exploitation of a sequence of combinatorial relaxations (possibly corresponding to different global constraints) through the additive bounding technique could result in stronger bounds computed in reasonable computing times. Moreover, the link between search and bound discussed here does not depend on LDS techniques but can be generalized to different search strategies whose generated sub-trees can be modeled by linear constraints. (Those issues are discussed in detail in §7.2.)

In the next section some preliminaries on reduced costs, additive bounding, and Limited Discrepancy Search are given. In §3 the link between search and bound is discussed mainly by presenting the additive bounding technique in conjunction with LDS, and by reviewing some existing methods. Section 4 focuses on the *AllDifferent* case, while §5 discusses specific test problems. Computational results are given in §6 showing the effectiveness of the proposed approach both as an enhancement in the LDS context (§6.3) and with respect to other relaxation methods from the literature (§6.4). Finally, additional remarks, extensions and conclusions are discussed in §7.

2. Preliminaries

Preliminaries on reduced costs, additive bounding, and LDS are discussed in the following two sections.

2.1. Reduced Costs and Additive Bounding

The concept of reduced cost has been proved crucial for CP, as it allows cost-based filtering (Focacci et al. 1999), provides search heuristics (Milano and van Hoeve 2002), and enables one to improve bounds, as we will show.

The reduced costs are computed as a result of the solution of a linear program. They come from duality theory and can be intuitively explained as follows. When the optimal solution of the linear program is computed, each value $\bar{c}_j$ in the reduced-cost vector can be seen as the cost that should be added to the optimal solution when the value of the corresponding variable x_j increases by one unit, i.e., variable x_j becomes part of the basis. The reduced-cost vector is such that $\bar{c}_j \geq 0$ for all variables x_j.

The additive bounding procedure (Fischetti and Toth 1989) is an interesting and effective technique for computing bounds for combinatorial optimization problems. Intuitively, an additive bounding procedure consists of solving a sequence of relaxations of a problem P, each producing an improved bound. With no loss of generality, we consider here minimization problems.

More formally, we consider a problem P whose general form is:

$$\begin{aligned} \min z &= c^T x \\ Ax &\geq b \\ x &\geq 0 \\ x &\text{ integer} \end{aligned}$$

where c is a cost vector, x is a solution vector, A is a coefficient matrix, and b is the right-hand side vector. We suppose we have a set of bounding procedures $B_1, \ldots, B_{nr}$ for P. We denote as $B_i(c)$ the ith bounding procedure when applied to an instance of problem P with cost vector c. For $k = 1, \ldots, nr$, the bounding procedure B_k returns a lower bound value LB_k and a reduced-cost vector $\bar{c}^k$. This reduced cost vector is used to "feed" the next bounding procedure B_{k+1}, i.e., $B_{k+1}(\bar{c}^k)$, as shown in Figure 1. The sum of the bounds $\sum_{k=1}^{nr} LB_k$ is a valid bound for problem P (Fischetti and Toth 1989).

Based on the definition of reduced costs, the validity of ABP is intuitive. Indeed, a single reduced-cost value $\bar{c}_j$ of a variable x_j is a (rather weak) lower bound on the increase of the objective function when the corresponding variable is forced into the basis (for the sake of simplicity) at value 1 and the linear program must be reoptimized. Thus, given the integer program

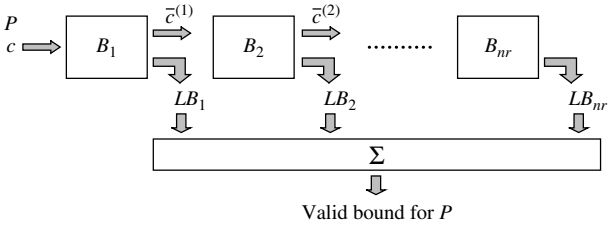

Figure 1 Additive Bounding Scheme

$\{\min c^T x \mid Ax \geq b,\ x_j = 1,\ x \geq 0,\ x \text{ integer}\}$, the solution of the initial relaxation (obtained by dropping constraints $x_j = 1$) $\{\min c^T x \mid Ax \geq b,\ x \geq 0\}$ gives as a by-product the reduced cost vector $\bar{c}$, and, in particular, $\bar{c}_j$ is also the optimal value of the linear program (obtained by dropping constraints $Ax \geq b$ and by using $\bar{c}$ as the cost vector) $\{\min \bar{c}^T x \mid x_j = 1,\ x \geq 0\}$ and can be added to the solution value of the initial relaxation to obtain a valid lower bound for the integer program. (Such a bound is an approximation of the optimal solution of $\{\min c^T x \mid Ax \geq b,\ x_j = 1,\ x \geq 0\}$.)

In the same way, after the solution of an initial relaxation $R(P)$, the reduced-cost vector $\bar{c}$ represents such a solution, and solving a different relaxation $R'(P)$ using $\bar{c}$ as the cost vector gives a lower bound on the change needed to reoptimize the previous solution to take into account this valid set of constraints.

2.2. LDS and Its Variants

An important concerns is the search strategy that we use, and it is particularly interesting when connected with additive bounding.

We restrict our attention to binary trees explored through LDS. We assume that our heuristic ranks the values in the domain variables $X_i\ \forall i \in \{1, \ldots, n\}$ that are split into two sets: a set $\mathcal{G}_i$ containing the values in the domain that are likely to correspond to *good* solutions and a set $\mathcal{B}_i$ for the values in the domain that are supposed to correspond to *bad* solutions. At each node, on the left branch we force the variable to range on the corresponding $\mathcal{G}_i$ and on the right branch on $\mathcal{B}_i$.

Then, the strategy we use is a two-step search: the first step concerns sub-problem generation where the variables are forced to range on one domain partition, while the second step concerns sub-problem solution. The tree is explored through an LDS. Thus, the first sub-problem generated has all variables ranging on the good domain part, then sub-problems are generated for increasing discrepancies. For discrepancy k, we have exactly k out of n variables that range on the bad domains, while the remaining on the good one.

This search strategy has been successfully applied to the traveling salesman problem with time windows with a reduced cost based domain partitioning (Milano and van Hoeve 2002) and is theoretically and experimentally evaluated in van Hoeve and Milano (2003). We refer to it in the following as *discrepancy-based search* (DBS).

3. Linking Search and Bound

Exploration of the search tree and the bounding procedure are usually linked during the solution of an optimization problem. Often, the information coming from the bounding procedure is exploited to guide the search heuristics. In general, branching decisions are imposed to take into account constraints violated by the relaxation. As an example, if we have a linear relaxation, the traditional branching strategy used in integer programming chooses a fractional variable and imposes two constraints removing that fractional value. Another example is to use the subtour-elimination strategy when solving an assignment problem as a relaxation of the asymmetric traveling salesman problem. This is not the only possibility. Indeed, reduced costs are often used to guide the search so promising values are considered and explored first.

Exploitation of the information in the other direction, i.e., from the search strategy to the bounding procedure, is generally trivial. In integer programming, e.g., the linear relaxation is usually enriched by the search constraint(s) and resolved. This can be clearly done in CP as well, but when the relaxation has some combinatorial structure it may be not a good idea. We exploit the search constraint associated with a discrepancy-based technique in a more sophisticated way, through additive bounding.

More precisely, suppose that we have a reference assignment $\overline{X}$ suggested by the heuristic and that we can represent the search tree at discrepancy k with a linear constraint, the k-discrepancy constraint, stating that all solutions belonging to that tree should differ from $\overline{X}$ in exactly k assignments. This linear constraint can be added to the problem model to tighten a relaxation, e.g., the linear one. In particular, we consider three alternative methods to enrich the relaxation with the discrepancy information. In §3.1 we introduce a new effective method, the additive bounding technique. In §3.2 we also present the straightforward method of adding the discrepancy constraint to the linear relaxation and the more sophisticated method of relaxing the discrepancy constraint in a Lagrangean way. The three methods are computationally compared in §6.4.

Often, practical use of discrepancy-based search imposes that, during the proof of optimality, the search method is switched to *depth first search* (DFS, see, e.g., Korf 1996 and Zhang 2000) since proving infeasibility of large discrepancy sub-problems could be time consuming. Exploiting k-discrepancy constraints enables us to improve the bound, and, provided that the search constraint is exploited within the additive bounding framework, we are allowed to stop the search as soon as such a bound guarantees that no better solution can be found. More precisely, when adding the k-discrepancy constraint, we obtain a bound worse than the current best feasible solution, and not only can we prune the tree at discrepancy k but also all trees at discrepancy greater than k. This result will be described more in detail in §3.3 where

we show that this pruning capability depends on the way in which the discrepancy constraint is exploited.

One may see the use of the AB technique herein as an enhancement to improve the capability of DBS to prove optimality (alternative uses of the additive bounding idea are discussed throughout the paper and recapped in §7.2). In a way, the hybrid method in which one switches to DFS after a few discrepancy values is another of such enhancements. Comparison of these two strengthening approaches will be discussed in §6.5.

3.1. Additive Bounding and DBS

Without loss of generality, we consider a minimization problem P defined on variables $X_1, \dots, X_n$ whose domains are denoted as $D_1, \dots, D_n$ and whose costs are $C_1, \dots, C_n$. Each time a relaxation is integrated in a CP solver, a mapping is needed that defines a correspondence between CP variables and relaxation variables. We consider here *linear programming* (LP) relaxations, and we assume that binary variables are mapped to CP domain values as follows. A classical mapping of CP variables into binary variables defines a variable x_{ij} such that

$$x_{ij} \in \{0,1\}, \quad \forall i \in \{1,\dots,n\}, \forall j \in D_i, \tag{1}$$

$$\sum_{j \in D_i} x_{ij} = 1, \quad \forall i \in \{1,\dots,n\}, \tag{2}$$

$$x_{ij} = 1 \longleftrightarrow X_i = j. \tag{3}$$

Moreover, through this mapping a cost c_{ij} is associated with each x_{ij} variable, and we consider the classical objective function

$$\min z = \sum_{i=1}^{n} \sum_{j \in D_i} c_{ij} x_{ij}. \tag{4}$$

Finally, we suppose that a relaxation $R(P)$ of problem P, defined with an integer-linear programming (ILP) model using the x variables, can be solved to optimality and returns (i) a bound LB, (ii) a vector of reduced costs $\bar{c}$, and (iii) an optimal solution x^* (which is, actually, not essential in the following).

Note that $R(P)$ is any relaxation and its solution procedure is used as the *first* bounding procedure of the additive bounding technique.

When the problem is solved through DBS we can add to problem P a constraint, called the *k-discrepancy constraint* Discrepancy_cst($[X_1, \dots, X_n]$, $[\mathcal{B}_1, \dots, \mathcal{B}_n]$, k), which holds if and only if k out of n variables range on the corresponding bad domain $\mathcal{B}$.

This constraint defined on CP variables can be expressed in terms of x variables by the set of constraints

$$\sum_{j \in \mathcal{B}_i} x_{ij} \le 1 \quad \forall i \in \{1,\dots,n\} \tag{5}$$

$$\sum_{i=1}^{n} \sum_{j \in \mathcal{B}_i} x_{ij} = k \tag{6}$$

$$x_{ij} \in \{0,1\} \tag{7}$$

where constraints (5) express that at most one value in the *bad* set of each original variable X_i must be considered, while constraint (6) forces that exactly k of the constraints (5) must be tight.

Obviously, these constraints can be can be simply added to $R(P)$, being valid for the defined subproblem. Indeed, taking into account constraints (5) and (6) in each node of the search tree would improve the lower bound obtained by considering only the constraints expressed by $R(P)$. However, if the relaxation $R(P)$ is solved through a special-purpose technique, it means that it has a special structure that is lost when the above constraints are added.

Thus, we are looking for a clean and general framework for improving the bound of $R(P)$ by taking into account constraints (5) and (6) and without relevant increase in the computational effort. This framework is indeed the additive bounding procedure, and this result is stated by the following theorem.

THEOREM 1. *Suppose that we have a relaxation $R(P)$ providing a lower bound LB and a reduced cost vector $\bar{c}$. Problem*

$$\left\{ \min z^k = \sum_{i=1}^{n} \sum_{j \in D_i} \bar{c}_{ij} x_{ij} \mid (5),\ (6),\ (7) \right\} \tag{8}$$

can be optimally solved in $O(n^2)$ time, $\tilde{z}^k$ being its optimal value, and $LB^k = LB + \tilde{z}^k$ is a valid lower bound for problem P when a k-discrepancy constraint is imposed.

PROOF. The optimal solution $\tilde{z}^k$ can be easily obtained through the following simple algorithm: (a) compute the smallest reduced cost $\min_{j \in D_i} \bar{c}_{ij}$ for each $i \in \{1,\dots,n\}$; (b) sort the n obtained reduced costs; and (c) select the k smallest ones. The time complexity of phase (a) is trivially bounded by $O(n^2)$, being the most expensive one since phase (b) (resp., (c)) requires only $O(n \log n)$ (resp., $O(n)$) time. Concerning the validity of the bound, it is trivially guaranteed by the additive argument since both $R(P)$ and the problem (4)–(7) involves (disjunct) relaxations of P, plus the k-discrepancy constraint. Finally, note that the time bound is obtained assuming that $\max_i |D_i| \le n$). □

As already pointed out, the above result is perfectly general and independent of the relaxation $R(P)$ considered, so it can be used with any other global constraint. Alternative methods using the search constraint, e.g., the linear relaxation, are discussed in the next sections. However, use of structured relaxations (and special-purpose algorithms to solve them) is often appealing (see, e.g., Focacci et al. 2003), and the additive procedure, when DBS is exploited, provides a clean and efficient improvement.

3.2. Alternative Use of the Search Constraint

The additive bounding procedure is not the only way of taking into account the k-discrepancy constraint, and, more generally, any branching constraint. Obviously, there are problem-specific ways; for example, in the asymmetric traveling salesman problem, branching on sub-tours can be efficiently handled by modifying the cost matrix accordingly (Carpaneto et al. 1995).

More generally, the k-discrepancy constraint can be taken into account rather simply if the considered relaxation is *linear* since, as shown in the previous section, such a constraint has a nice linear representation as well. However, as anticipated, when the relaxation is solved through a special-purpose algorithm exploiting its structure, addition of the k-discrepancy constraint usually destroys such a structure. This motivates the use of the additive bounding idea, but another possibility is that of using Lagrangean relaxation. To this end, we consider again the relaxation $R(P)$ introduced in the previous section by assuming that there exists a special-purpose algorithm to solve it.

In particular, we adopt a slight modification of the Lagrangean-relaxation framework proposed by Focacci et al. (2002):

(a) solve the linear relaxation involving $R(P)$ plus the k-discrepancy constraint each time a new discrepancy constraint is enforced (i.e., each time a new subproblem with increased discrepancy is considered);

(b) dualize the k-discrepancy constraint by using as a Lagrange multiplier, say λ_k, the value of the dual variable associated with the constraint (as computed by the general-purpose simplex-type LP solver);

(c) fix the value of λ_k in the sub-tree associated with k and solve, as soon as it is necessary, the relaxations $R(P)$ on the modified cost vector of step (b), with the special-purpose algorithm.

Note that the k-discrepancy constraint is the only one that needs to be dualized if and only if the combinatorial relaxation $R(P)$ implies constraints (5). This is true whenever $R(P)$ takes into account the variable mapping (1)–(3).

The above framework obviously does not depend on $R(P)$, but it can be applied easily by specializing it with any structured relaxation allowing a linear-programming representation and a special-purpose solver.

A computational comparison between the additive bounding approach and these alternative methods is reported in §6.4.

3.3. Monotonicity of the Bound

Before closing this section, which has been devoted to the various ways of using the discrepancy constraint to obtain an improved bound, an important observation is needed.

We have to take into account the linear program obtained by enhancing the relaxation $R(P)$ of our problem P at discrepancy k with constraints (5), (6), and the relaxed version $x_{ij} \in [0, 1]$ of constraints (7). Optimizing the linear objective function (4) over such a polytope gives us a valid lower bound L^k for the sub-tree defined by the k-discrepancy constraint. However, there is no dominance relation between this bound and the one obtained at discrepancy $k+1$. In other words, if, using the linear-programming relaxation, we encounter a discrepancy k such that $L^k \geq UB$, where UB is the incumbent solution value, then we are not allowed to conclude that $L^h \geq UB$, $\forall\, h = k+1, \ldots, n$. Thus, we should in principle consider all the sub-trees for $k = 1, \ldots, n$. This behavior can be improved by weakening L^k by considering equality (6) in "$\geq$" form, $\sum_{i=1}^{n} \sum_{j \in \mathcal{B}_i} x_{ij} \geq k$. It is trivial to see that this latter bound $\hat{L}^k$ is indeed monotone, and computing it at the root of each discrepancy k allows us to prune the sub-trees with $h > k$ once $\hat{L}^k \geq UB$ is verified for some value of k.

The precise algorithm if one wants to use the linear relaxation requires us to compute two bounds L^k and $\hat{L}^k$ for each discrepancy k, where the former is used to prune within the sub-tree at discrepancy k, while the latter prunes the following sub-trees.

On the other hand, this is not true for the bound LB^k computed through additive bounding. In fact, such a bound is, by construction, monotone in k since every time *only* problem (8) is solved, i.e., the reduced costs computed at the root node are added one after the other (in non-decreasing order) to the relaxation with the increase of k. (Recall, indeed, that $\bar{c}_{ij} \geq 0$, $\forall\, x_{ij}$.) Thus, as soon as $LB^k \geq UB$ for some value of k, all sub-trees with $h > k$ are immediately pruned.

4. Application to the *AllDifferent* Constraint

It was shown in the previous section how to exploit additive bounding for improving a combinatorial relaxation in the presence of a DBS constraint. To prove the effectiveness of this framework, we consider the general case of problems modeled using the *AllDifferent* constraint. It is well-known that the ILP model of the *AllDifferent* is the classical *linear assignment problem* (AP, see, e.g., Dell'Amico and Martello 1997a), provided the possible addition of dummy variables. Since the AP can be solved in polynomial time by special-purpose algorithms, using the AP as a combinatorial relaxation is particularly suited in this context.

The following is a general CP model of any problem containing the *AllDifferent* constraint and to which

is added the discrepancy constraint needed to perform DBS:

$$\min \sum_{i\in N} C_{iX_i} \tag{9}$$

$$\text{s.t. AllDifferent}(X) \tag{10}$$

$$\text{AnySide\_cst}(X) \tag{11}$$

$$\text{Discrepancy\_cst}(X, \mathcal{B}, k) \tag{12}$$

$$X_i \in D_i \quad \forall i \in N \tag{13}$$

where $D_i \subseteq N$ and $N = \{1, \ldots, n\}$.

Using the same mapping introduced in §3.1, and for the sake of simplicity defining a directed graph $G = (N, A)$ where arc $(i, j) \in A$ if and only if $j \in D_i$, problem (9)–(13), denoted by P^{gen}, maps into the following ILP formulation:

$$z(P^{\text{gen}}) = \min \sum_{i\in N}\sum_{j\in N} c_{ij}\, x_{ij} \tag{14}$$

$$\text{subject to} \quad \sum_{j\in N} x_{ij} = 1, \quad i \in N \tag{15}$$

$$\sum_{i\in N} x_{ij} = 1, \quad j \in N \tag{16}$$

$$\text{AnyLinearSide\_cst}(x) \tag{17}$$

$$\sum_{i\in N}\sum_{j\in \mathcal{B}_i} x_{ij} = k \tag{18}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A \tag{19}$$

where (5) is redundant due to (15) and (18) is an equivalent linear formulation of (11).

4.1. A First Approach

A first lower bound is easily deduced from P^{gen} since the relaxation of constraints (17) and (18) directly yields the formulation of the assignment problem. Since the AP can be efficiently and incrementally solved through the well-known *Hungarian algorithm* (e.g., Carpaneto et al. 1988), one would not like to change this structure even if one would like to take into account, at least partially, branching constraints.

In this context, we consider k-discrepancy (branching) constraints, and we use the output of the Hungarian algorithm, $(z^*(\text{AP}), \bar{c})$, to define problem P' as (8), whose optimal solution can be computed as follows:

(a) Compute the smallest reduced cost $\min_{j\in N} \bar{c}_{ij}$ $\forall i \in N$;

(b) Sort the $|N|$ obtained reduced costs; and

(c) Select the k smallest ones, say $\tilde{c}_1, \ldots, \tilde{c}_k$.

The improved bound is then $LB' = z^*(\text{AP}) + \sum_{h=1}^{k} \tilde{c}_h$, and the time complexity of this improvement is $O(n^2)$.

4.2. Better Exploiting the Structure

The method described in the previous section does not take into account the structure of the AP but only the fact that the AP can be solved through an algorithm that provides reduced costs. In fact, the structure of the AP can be efficiently exploited to improve the integration of k-discrepancy constraints. Formally, the following new relaxation, called the *k-cardinality assignment problem* (k-AP), can be defined as

$$\begin{aligned} z(k\text{-AP}) = \min & \sum_{i\in N}\sum_{j\in N} c_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{j\in N} x_{ij} \le 1, \quad i \in N \\ & \sum_{i\in N} x_{ij} \le 1, \quad j \in N \\ & \sum_{i\in N}\sum_{j\in N} x_{ij} = k \\ & x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A. \end{aligned} \tag{20}$$

The k-AP is a relaxation of P^{gen} (14)–(19) where the constraint (17) has been removed, the degree constraints of both (15) and (16) are relaxed to the "$\le$" form, and constraint (6) is relaxed to (20) by summing also on the variables x_{ij} such that $j \in \mathcal{G}_i$. The k-AP was introduced by Dell'Amico and Martello (1997b) and can be solved in polynomial time by both primal and dual algorithms (Dell'Amico et al. 2001).

Since k-AP is another relaxation of the problems we obtain at each node of the branching tree by imposing a k-discrepancy constraint, we now have two combinatorial relaxations, and we can make them co-operate through the additive bounding procedure. Specifically, we can use the AP relaxation first, and then "feed" the k-AP with the reduced-cost vector $\bar{c}$. This procedure turns out to change the k-AP we defined before: since $\bar{c}_{ij} \ge 0$, $\forall (i, j) \in A$, the variables x_{ij} such that $i \in N$, $j \in \mathcal{G}_i$ can be disregarded (i.e., set to 0) because in the original k-discrepancy constraint (6) they are not considered. This induces a sub-graph $G' = (N, A')$ where $A' = \{(i, j) \mid j \in \mathcal{B}_i\}$ and the definition of the following problem, k-AP′, which is also a valid relaxation of P^{gen} when used as a second bound:

$$\begin{aligned} z(k\text{-AP}') = \min & \sum_{(i,j)\in A'} \bar{c}_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{(i,j)\in A'} x_{ij} \le 1, \quad i \in N \\ & \sum_{(i,j)\in A'} x_{ij} \le 1, \quad j \in N \\ & \sum_{(i,j)\in A'} x_{ij} = k \\ & x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A'. \end{aligned}$$

The improved bound is then $LB' = z^*(\text{AP}) + z^*(k\text{-AP}')$. For clarity we will henceforth refer to k-AP′ simply as k-AP.

5. Test Problems

In order to validate the ideas proposed in this paper, we need to address optimization problems involving the *AllDifferent* constraint. We run experiments on three such problems, namely, the very well-known asymmetric traveling salesman problem (ATSP), and two natural and difficult extensions of the assignment problem.

5.1. Asymmetric Traveling-Salesman Problem

The ATSP is a well-known combinatorial optimization problem, and the CP models presented for this problem (except for the *k-discrepancy* constraint (23)) are generally variations of

$$\begin{aligned} \min\ & \sum_{i=1}^{n} C_{iX_i} \\ \text{s.t.}\ & \text{AllDifferent}(X) && (21)\\ & \text{NoSubTour}(X) && (22)\\ & \text{Discrepancy\_cst}(X, \mathscr{B}, k) && (23)\\ & X_i \in \{1, \ldots, n\} \quad \forall i \in \{1, \ldots, n\} \end{aligned}$$

The X variables used in this model are successor variables, which means that X_i takes the value of the node directly visited after node i. The *AllDifferent* constraint (21) is used to express conservation of flow in the network. Since the nature of the decision variables already enforces that each node has exactly one outgoing arc, we need only make sure that it also has exactly one ingoing arc. To do so, it is necessary to ensure that no two nodes have the same successor, which is the role of the *AllDifferent* constraint. The *NoSubTour* constraint (22) (see, e.g., Pesant et al. 1998) enforces connectivity of the solution by preventing cycles that are not Hamiltonian, i.e., do not involve all nodes. Constraint (23) is used to perform DBS by imposing that exactly k variables X take their values in their respective $\mathscr{B}$ sets. This problem maps into P^{gen} (14)–(19) by substituting (17) with (22).

As in many other cases in the literature, the pure ATSP is used in this context as a useful setting for comparing different techniques without the aim of improving on state-of-the-art results. However, a preliminary version of integration of DBS and additive bounding has been successfully applied to an ATSP time-constrained variant (Milano and van Hoeve 2002).

The set of ATSP instances we used were based on the ones proposed by Cirasella et al. (2001). They are 12 different classes of instances all presenting different cost structures. We used the random generator provided at http://www.research.att.com/~dsj/chtsp/.

5.2. Lower Assignment Problem

The lower assignment problem (LAP) is a simple assignment problem enriched with an additional constraint imposing a lower bound on the objective. This problem can be formulated by the following CP model:

$$\begin{aligned} \min\ & \sum_{i=1}^{n} C_{iX_i} && (24)\\ \text{s.t.}\ & \text{AllDifferent}(X) && (25)\\ & \sum_{i=1}^{n} C_{iX_i} \geq L && (26)\\ & \text{Discrepancy\_cst}(X, \mathscr{B}, k) && (27)\\ & X_i \in \{1, \ldots, n\} \quad \forall i \in \{1, \ldots, n\} \end{aligned}$$

Each assignment of a value j to a variable X_i incurs the cost C_{ij}, and the objective function (24) is the minimization of the sum of these costs over all variables. The lower bound on the objective is defined by (26), the *AllDifferent* constraint is expressed in (25), and the discrepancy-based search is imposed by (27). This problem maps onto P^{gen} (14)–(19) by substituting (17) by the linear equivalent of (26).

It is not difficult to prove the following result:

THEOREM 2. *LAP is NP-hard.*

PROOF. The proof uses a reduction of the minimization version of the classical *subset-sum* problem to LAP. The minimum subset sum is the following problem:

$$\left\{ \min \sum_{j=1}^{n} c_j x_j \;\middle|\; \sum_{j=1}^{n} c_j x_j \geq L,\ x \in \{0,1\}^n \right\},$$

and the reduction requires the construction of a bipartite graph $G = (U \cup V, W)$ with $|U| = |V| = 2n$ (i.e., $4n$ vertices numbered as $U = \{1, \ldots, 2n\}$ and $V = \{2n+1, \ldots, 4n\}$) and $|W| = 4n$ such that:

(a) $c_{j,2n+j} = c_j$, $c_{j,3n+j} = 0\ \forall j = \{1, \ldots, n\}$,
(b) $c_{j,2n+j} = 0$, $c_{j,3n+j} = 0\ \forall j = \{n+1, \ldots, 2n\}$.

Optimally solving the LAP on such a graph solves the minimum-subset-sum problem, thus proving the result. □

Depending on the value of L, LAP can be very difficult to solve even for small values of n. Indeed, the trivial combinatorial relaxation for LAP is the AP and the computational complexity lies in the fact that this bound can be very poor. More precisely, if L is more than a few percent greater than the optimal solution of the AP, then the optimality gap becomes very hard to close. Furthermore, when L increases, the quality of the information provided by the AP bound (for instance to guide the search or to perform reduced-cost-based filtering, Focacci et al. 1999, 2003) decreases. This additional constraint (26) can thus be seen as an abstraction of any set of unstructured combinatorial constraints that make solutions near the

lower bound infeasible and that cannot be treated by any special algorithm or global filtering constraint.

To generate instances of this problem, we reused the ATSP instances previously described. In the original instances, the costs associated with diagonal entries of the matrix (C_{ii}) are set to 0 since these assignments are forbidden by the ATSP definition. We modified these entries by setting them to $+\infty$ to prevent the AP optimal solution to be constituted only of self assignment ($X_i = i$) and have an objective of 0. Finally, following the previous discussion, the value of L is set by using the value of the AP relaxation. As mentioned, values of L too close to the AP value make the problem too easy to solve (and not interesting), while too-high values of L tend to generate very hard problems. For the present experimentation L was set to a value of 10% above the relaxation value, yielding difficult but computationally solvable problems.

5.3. Resource-Constrained Assignment

The resource-constrained assignment problem (RCAP) imposes over the traditional AP a constraint limiting the consumption of a certain resource. This problem is described by the following CP model:

$$\min \sum_{i=1}^{n} C_{iX_i} \tag{28}$$

$$\text{s.t. AllDifferent}(X) \tag{29}$$

$$\sum_{i=1}^{n} R_{iX_i} \le M \tag{30}$$

$$\text{Discrepancy\_cst}(X, \mathcal{B}, k) \tag{31}$$

$$X_i \in \{1, \ldots, n\} \quad \forall i \in \{1, \ldots, n\}$$

Each assignment of a value j to a variable X_i still incurs the cost C_{ij} to the objective functions (28) but also consumes a resource R_{ij}, of which there is a limited amount M (constraint (30)); the *AllDifferent* constraint is expressed in (29); and the discrepancy-based search is imposed by (31). This problem also maps into P^{gen} (14)–(19) by substituting (17) by the linear equivalent of (30). Due to the addition of a knapsack-like constraint, RCAP is obviously NP-hard.

To generate RCAP instances we again reused the ATSP instances previously described. To define the R (resource) matrix we used the distance matrix of the Solomon TSP with time-window-clustered instance `C101` (Solomon 1987). The challenge was then to set M to a value that was neither too constraining nor too easy. After a few experiments, this was achieved by first solving an AP on the R matrix and then setting M to 4 times the value of this AP optimal solution.

6. Computational Experiments

We ran two different sets of experiments. First, we compared on the test problems presented in the previous section the performance of traditional DBS with the two variants using additive bounding as an enhancement. The results are reported in §6.3. Second, we considered additional methods of taking into account the k-discrepancy constraint to improve the bound and we computationally compare them in §6.4.

Before describing the results we give some details on the implementation.

6.1. Implementation Details

The search strategies used to solve the problem are the typical branching schemes of CP. The variables are dynamically ordered using the simple *first fail* criteria of increasing domain sizes. Once a variable is selected, it is fixed to the value of its domain, which seems the most promising with respect to the objective function. For these applications, we have chosen the reduced costs as a quality criterion and we thus branch on the value j of D_i associated with the minimum reduced cost $\bar{c}_{ij}$. Reduced costs are also used to partition D_i into $\mathcal{G}_i$ and $\mathcal{B}_i$ for each X_i (see the discussion below).

Since the relaxation provides reduced costs, we are also able to perform reduced-cost filtering (see Focacci et al. 1999). This technique is employed to filter out values associated with sub-optimal solutions, which means eliminating all assignments whose reduced cost added to the current lower bound exceeds the upper bound:

$$LB + \bar{c}_{ij} > UB \implies X_i \neq j, \quad \forall i, j \in \{1, \ldots, n\}$$

where the upper bound UB is defined as one less than the value of the best feasible solution (for costs with integer values).

The CP models were implemented using the ILOG Solver 5.2 library and all experimentations were performed on a Intel 1.5 GHz Centrino laptop.

6.2. Cardinality of the Good Set

As mentioned in Milano and van Hoeve (2002), a clearly important part of the discrepancy-based search is the separation of domains into $\mathcal{G}$ and $\mathcal{B}$ sets. To choose an efficient separation one must balance two conflicting objectives: the complexity of the sub-problems and the tightness of the additive bound. If the chosen $\mathcal{G}$ sets are relatively small, then the sub-problems will have few values and will be easier to solve. However, the reduced costs associated with the values in the $\mathcal{B}$ sets will also be lower, yielding a weaker second bound and longer proof of optimality. On the other hand, generating larger $\mathcal{G}$ sets would leave larger reduced costs in the $\mathcal{B}$ sets and increase the value of the discrepancy bound. Unfortunately, this would also make each sub-problem much harder to solve. We have also noted that the difference in performance between counting the k minimum reduced costs and solving the k-AP tends to increase as the cardinality of the $\mathcal{G}$ sets decreases, which is probably

Table 1 Comparison on the ATSP

		DBS			*Count-RC*			*k*-AP		
Name	nb	Time	*BT*	*k*	Time	*BT*	*k*	Time	*BT*	*k*
amat	5	0.2	964.8	26.0	0.1	733.2	2.0	0.1	733.2	2.0
coin	5	21.2	159,642.0	26.0	10.0	75,012.4	4.0	10.0	73,958.6	4.0
crane	5	3.4	29,243.0	26.0	2.5	19,681.0	3.6	2.5	19,610.8	3.6
disk	5	0.7	9,248.4	26.0	0.8	8,671.4	11.6	0.8	8,671.4	11.6
rect	5	2.9	24,090.8	26.0	1.8	13,618.2	2.6	1.8	13,600.6	2.6
rtilt	5	4.4	37,839.6	26.0	3.2	24,063.8	3.4	4.0	23,842.4	3.4
shop	5	3.8	29,673.2	26.0	4.6	29,660.8	13.6	4.6	29,660.6	13.4
smat	5	0.2	881.8	26.0	0.1	687.0	1.8	0.1	687.0	1.8
stilt	5	3.1	22,244.8	26.0	3.2	17,402.0	3.6	3.0	17,380.6	3.4
super	5	0.1	829.6	26.0	0.2	814.2	10.6	0.2	814.2	10.6
tmat	4	85.7	784,600.5	26.0	59.9	505,721.0	6.2	53.1	437,560.7	6.0
tsmat	4	124.8	1,199,097.7	26.0	86.4	696,212.0	5.2	72.2	555,324.0	5.2

due to the presence of many small reduced costs in the $\mathcal{B}$ sets.

In this application, the sizes of the $\mathcal{G}$ sets were set to 20% of the instance size (here five values). Moreover, all values that were less than 20% greater than the largest value in each $\mathcal{G}$ were also included into $\mathcal{G}$, up to a maximum of ten values (40% of the size). Ongoing work by van Hoeve and Milano (2003) is devoted to these issues.

6.3. Enhancing DBS through Additive Bounding

To evaluate the performance of the additive bounding on the DBS, we compared three different approaches: a first approach using only the first AP bound (*DBS*), the simple implementation of the additive bounding procedure described in §4.1 (*Count-RC*), and the more complex, and problem-dependent, (*k*-AP) version of this bound detailed in §4.2 (*k*-AP).

We considered 12 problem classes and generated five instances of each, for a total of 55 benchmark problems. Input data for problem generation are available on request. All problems are defined with $n = 25$ and all values reported in Tables 1–3 are averaged over the number of problems solved by all methods.

In Tables 1–3 we thus report the name of the class of instances (Name), the number of problems solved within a time limit of 3,600 CPU seconds (nb), the average computing time in seconds (Time), the average number of backtracks (*BT*), and the average maximum number of discrepancies (*k*) needed to prove optimality.

Tables 1–3 clearly indicate that discrepancy additive bounding, even in its simplest form, has significant impact on the time and the number of backtracks needed to prove optimality.

When considering all the problems, the *Count-RC* method reduces the computing time (resp. backtracks) of the search tree needed to prove optimality from a minimum of 34% (resp. 29%) to a maximum of 124% (resp. 131%) as shown in Table 4. The more sophisticated *k*-AP second bound, even though it is more time-consuming, achieves a reduction in the computing time (resp. backtracks) from a minimum of 44% (resp. 30%) to a maximum of 126% (resp. 137%), thus showing that taking advantage of a special structure

Table 2 Comparison on the LAP

		DBS			*Count-RC*			*k*-AP		
Name	nb	Time	*BT*	*k*	Time	*BT*	*k*	Time	*BT*	*k*
amat	5	0.6	2,575.0	26.0	0.3	1,274.0	2.4	0.3	1,274.0	2.4
coin	4	49.7	171,812.2	26.0	7.4	23,970.0	2.2	7.2	23,017.7	2.2
crane	5	16.4	63,501.4	26.0	4.9	18,292.8	3.4	4.9	18,042.8	3.4
disk	5	2.4	8,942.2	21.0	0.6	2,149.4	11.6	0.6	2,029.0	11.6
rect	5	230.6	747,451.2	26.0	49.0	164,336.8	4.4	47.0	155,428.6	4.2
rtilt	5	211.1	727,264.4	26.0	46.9	149,091.2	5.0	44.9	138,407.6	5.0
shop	5	18.9	66,534.2	26.0	7.1	23,264.2	4.4	6.3	20,422.8	4.4
smat	5	90.9	354,681.6	26.0	20.7	73,765.4	3.2	19.3	67,899.2	3.2
stilt	5	31.0	99,590.0	26.0	13.3	40,381.4	4.8	13.0	38,840.2	4.8
super	5	8.7	41,010.0	21.0	6.5	28,430.4	2.0	6.6	28,365.6	2.0
tmat	5	4.7	18,132.4	21.0	1.9	7,377.2	4.8	1.8	7,117.2	4.6
tsmat	5	254.6	851,003.0	26.0	252.2	829,810.2	4.4	255.7	828,933.6	4.4

Table 3 Comparison on the RCAP

		DBS			*Count-RC*			*k*-AP		
Name	nb	Time	*BT*	*k*	Time	*BT*	*k*	Time	*BT*	*k*
amat	5	0.1	145.2	21.0	0.0	125.2	1.0	0.0	125.2	1.0
coin	5	65.0	206,471.6	26.0	54.4	142,019.8	5.6	44.6	141,604.0	5.6
crane	5	29.9	267,854.2	26.0	35.4	261,936.8	5.8	35.7	261,907.4	5.8
disk	5	307.9	3,753,216.0	26.0	329.7	3,446,931.0	7.0	320.9	3,442,334.0	7.0
rect	4	60.8	212,384.2	26.0	60.9	166,730.2	7.5	49.4	164,716.7	6.7
rtilt	4	98.9	255,745.2	26.0	81.0	157,282.5	8.2	65.0	153,106.5	8.2
shop	5	1.1	3,187.6	26.0	1.3	2,756.0	3.8	1.0	2,735.2	3.8
smat	5	1.0	2,062.4	26.0	0.8	1,228.8	4.0	0.6	1,212.2	4.0
stilt	4	65.5	154,131.5	26.0	36.8	61,395.5	7.7	30.7	60,510.7	7.7
super	5	0.1	237.8	16.0	0.1	215.4	1.0	0.1	215.4	1.0
tmat	5	0.1	186.0	16.0	0.1	171.4	1.4	0.0	171.4	1.4
tsmat	1	294.7	1,020,111.0	26.0	92.0	301,073.0	9.0	95.7	286,588.0	9.0

(in this case the *AllDifferent* constraint) may be worth the effort.

If we compare the two implementations of the additive bounding idea, we notice that the approach using the *k*-AP version generally needs fewer discrepancies (k) to prove optimality. This indicates that the lower bounds obtained by this method are higher, which not only allows us to reduce the value of k but also allows a more effective pruning of each subproblem search tree (which translates in a reduced number of backtracks).

6.4. Different Ways of Coping with the *k*-Discrepancy Constraint

We compared the performance of the best algorithm of the previous section, i.e., the DBS using the *k*-AP additive relaxation, with the two different approaches discussed in §3.2: the approach taking into account the *k*-discrepancy constraint via linear relaxation (*linear* AP) and the Lagrangean approach (*Lagrangean* AP).

Tables 5–7 report the results of such a comparison on ATSP, LAP, and RCAP, respectively.

Tables 5–7 clearly show that, when the *AllDifferent* constraint is considered, the additive bounding technique to cope with the *k*-discrepancy constraint performs, in general, better in terms of computing time and number of solved instances than the competitors. As expected, the *linear*-AP approach gives the best bound, thus corresponding in the smallest values for k in all tables. Also the *Lagrangean*-AP approach gives slightly better values of k with respect to the *k*-AP approach.

However, for both *linear*-AP and *Lagrangean*-AP this improvement of the bound does not correspond to shorter computing times and to fewer backtracks. Solving LPs is in general more expensive in terms of computing time, while, with respect to the number of backtracks, the cost-based propagation associated with pure APs makes the difference by allowing many more reductions in the variables' domains. Finally, the optimal multiplier computed in step (b) of the Lagrangean procedure outlined in §3.2 unfortunately does not remain optimal during the exploration of the sub-tree associated with a fixed value of k, thus penalizing the method. (For a detailed discussion on optimality of the Lagrange multipliers, see Focacci et al. 2002.)

Table 4 Percentage Reduction of Computing Time and Backtracks Needed to Prove Optimality

	Count-RC (%)		*k*-AP (%)	
Problem	Time	*BT*	Time	*BT*
ATSP	45	65	64	94
LAP	124	131	126	137
RCAP	34	29	44	30

6.5. Hybrid DFS

As discussed in §3, the additive-bounding technique enhances the capability of DBS to prove optimality. An alternative method would be to use DBS in the early stages, i.e., with small values of k, to find good solutions quickly, and then switch to depth-first search when the incumbent solution seems to be "good enough" in the attempt to prove faster that the unique remaining sub-tree is empty. A natural question concerns the comparison of our proposed method with this hybrid DFS algorithm. Thus, we tested two versions of the latter, namely a version in which the initial algorithm is the standard DBS, say H1-DFS, and another in which the initial algorithm is instead the DFS with *k*-AP (our best approach), say H2-DFS. Concerning the threshold in which the algorithms switch to DFS, preliminary computational experiments suggested $k \geq 5$, which guarantees both a fast initial convergence to good quality solutions and the benefit of a depth first search. An adaptive method to select such a breakpoint could be effective.

Table 5 Additional Comparison on the ATSP

	k-AP				*Linear*-AP				*Lagrangean*-AP			
Name	nb	Time	*BT*	*k*	nb	Time	*BT*	*k*	nb	Time	*BT*	*k*
amat	5	0.1	733.2	2.0	5	2.6	4,683.0	1.8	5	0.5	696.2	1.8
coin	5	10.0	73,958.6	4.0	5	270.0	552,478.8	3.4	5	61.4	93,521.6	3.4
crane	5	2.5	19,610.8	3.6	5	48.5	107,854.4	2.2	5	15.5	23,540.4	3.4
disk	5	0.8	8,671.4	11.6	5	7.3	17,783.4	11.2	3	3.0	4,437.3	1.3
rect	5	1.8	13,600.6	2.6	5	27.5	65,171.6	2.4	5	10.4	17,084.4	2.4
rtilt	5	4.0	23,842.4	3.4	5	77.3	161,117.2	2.8	5	15.3	23,444.8	2.8
shop	5	4.6	29,660.6	13.4	5	0.3	703.8	1.0	5	9.7	15,326.6	12.4
smat	5	0.1	687.0	1.8	5	2.1	3,575.0	1.8	5	0.5	555.2	1.8
stilt	5	3.0	17,380.6	3.4	5	19.7	43,087.8	3.0	5	6.2	8,400.2	3.0
super	5	0.2	814.2	10.6	5	0.0	45.0	1.0	5	0.4	616.2	9.4
tmat	4	53.1	437,560.7	6.0	4	537.5	1,142,129.2	2.7	4	375.2	607,112.2	4.5
tsmat	4	72.2	555,324.0	5.2	3	932.7	2,049,069.7	2.7	4	533.5	927,676.5	4.2

Such a method could be based on the gap between lower and upper bounds.

The results of those tests confirm that the additive-bounding approach enhances the capability of DBS to prove optimality. In particular, H1-DFS is consistently worse than our best approach *k*-AP both in terms of the number of backtrackings and computing times. The percent deterioration of the computing times is in the range [24%, 92%], proving that the remaining sub-tree that DFS has to explore is still rather large. Algorithm H2-DFS is instead comparable with our best approach in most instances, thus showing that the additive bounding idea can further hybridize such a hybrid DBS and DFS algorithm, and it is a viable way of solving combinatorial optimization problems.

7. Remarks, Extensions, and Conclusions

Some remarks concerning traditional LDS techniques, and extensions of the AB technique are discussed in the next two sections, respectively. Finally, some conclusions are drawn in the last section.

7.1. Application to Traditional LDS

The discrepancy-based additive bounding presented in this paper can also be applied to the general LDS framework, when one counts one discrepancy for each right-hand branch taken in the search tree. Given that all the values in the domains are ordered, that p_j^i is the position of value j in X_i's domain (first position being 0), and that x_{ij} takes value 1 when X_i is set to value j, the following equation can replace the discrepancy constraint (6):

$$\sum_{i \in N} \sum_{j \in B_i} p_j^i x_{ij} = k.$$

Preliminary results are disappointing, again because there are too many small reduced costs (many of value 0) in the discrepancy branches. This means that, except for very high values of k, the discrepancy lower bound is either null or very small. Since traditional LDS can be compared to DBS where the cardinality of the $\mathcal{G}$ sets is 1 (van Hoeve and Milano 2003), the preliminary tests performed confirm the issue raised discussing the cardinality of good sets in §6.

Table 6 Additional Comparison on the LAP

	k-AP				*Linear*-AP				*Lagrangean*-AP			
Name	nb	Time	*BT*	*k*	nb	Time	*BT*	*k*	nb	Time	*BT*	*k*
amat	5	0.3	1,274.0	2.4	5	1.2	2,151.6	2.2	5	1.3	1,589.6	2.2
coin	4	7.2	23,017.7	2.2	4	24.6	40,079.2	2.2	4	25.3	36,012.7	2.2
crane	5	4.9	18,042.8	3.4	5	18.0	34,054.8	3.0	5	24.9	35,042.8	3.0
disk	5	0.6	2,029.0	11.6	5	9.6	16,585.8	2.2	5	10.3	14,600.8	2.2
rect	5	47.0	155,428.6	4.2	5	293.2	497,208.4	3.8	5	558.3	747,719.8	3.8
rtilt	5	44.9	138,407.6	5.0	5	455.5	652,303.6	4.4	5	245.6	295,447.8	4.4
shop	5	6.3	20,422.8	4.4	5	45.6	70,080.2	3.4	5	47.1	56,974.8	3.4
smat	5	19.3	67,899.2	3.2	5	79.7	114,828.8	3.0	5	86.3	134,198.2	3.0
stilt	5	13.0	38,840.2	4.8	5	118.0	180,034.6	4.2	5	78.4	103,228.4	4.2
super	5	6.6	28,365.6	2.0	5	37.0	76,156.0	2.0	5	21.6	30,481.6	2.0
tmat	5	1.8	7,117.2	4.6	5	12.9	19,943.2	3.0	5	17.8	23,187.8	3.0
tsmat	5	255.7	828,933.6	4.4	5	33.2	56,666.4	3.4	5	104.6	188,109.0	3.4

Table 7 Additional Comparison on the RCAP

	k-AP				*Linear*-AP				*Lagrangean*-AP			
Name	nb	Time	*BT*	*k*	nb	Time	*BT*	*k*	nb	Time	*BT*	*k*
amat	5	0.0	125.2	1.0	5	0.1	136.0	1.0	5	0.1	151.6	1.0
coin	5	44.6	141,604.0	5.6	5	100.8	204,015.4	5.4	5	179.5	266,401.0	5.4
crane	5	35.7	261,907.4	5.8	5	135.2	297,566.6	5.2	5	239.8	373,447.2	5.2
disk	5	320.9	3,442,334.0	7.0	4	185.4	311,616.0	6.0	4	327.5	502,628.7	6.0
rect	4	49.4	164,716.7	6.7	4	196.9	344,010.0	6.5	4	358.1	518,509.0	6.5
rtilt	4	65.0	153,106.5	8.2	4	627.0	1,178,541.7	7.7	3	115.5	169,398.0	5.5
shop	5	1.0	2,735.2	3.8	5	1.6	3,172.4	3.2	5	2.3	3,099.0	3.2
smat	5	0.6	1,212.2	4.0	5	2.2	3,180.0	3.6	5	2.5	2,697.4	3.6
stilt	4	30.7	60,510.7	7.7	4	144.0	220,060.0	7.2	4	120.3	142,301.5	7.2
super	5	0.1	215.4	1.0	5	0.2	393.4	1.0	5	0.2	248.6	1.0
tmat	5	0.0	171.4	1.4	5	0.1	116.0	1.0	5	0.1	173.4	1.0
tsmat	1	95.7	286,588.0	9.0	1	2,679.1	3,232,592.0	8.0	1	3,403.2	3,718,431.0	8.0

7.2. Extensions: Global Constraints and Search Strategies

It is important to notice that application of the additive-bounding framework in the CP context is not limited to interaction between a single global constraint and the search strategy. Indeed, ABP, as outlined in Figure 1, can be naturally applied to a sequence of structured relaxations corresponding to several global constraints. In other words, the link between search and bound described in this paper is not the only context that can benefit from the additive-bounding technique.

7.2.1. Additive Bounding with More Global Constraints. It seems that the additive bounding technique should indeed be revisited in the CP context where the solution of combinatorial relaxations through special-purpose (generally incremental) algorithms is often preferred to the solution of large linear programs. For example, in the case of the ATSP, a sensible way of obtaining a strong lower bound is applying ABP to the sequence of AP and *minimum-spanning-tree* relaxations (see Fischetti and Toth 1992 for details). Such an approach proved effective in the ILP setting (Fischetti and Toth 1992) but was later outperformed by sophisticated branch-and-cut techniques exploiting polyhedral studies and the effectiveness of current LP solvers (see Fischetti et al. 2002 for details). However, CP solvers are generally designed to make very limited use of cutting planes and LP relaxations because propagation algorithms, constraint satisfaction, and clever enumeration techniques are their basic ingredients. Thus, a quick way of generating effective and efficient bounds is preferred and becomes customary when the problems to be solved involve an optimization component that is not negligible, like TSP and its variants.

On the other hand, the CP modeling and solving framework based on global constraints immediately recalls additive bounding techniques. Indeed, global constraints are generally overlapping because they capture different but interconnected aspects of the overall problem. It is often the case, e.g., that more than one *AllDifferent* constraint is used in the same model, and it is easy to see that our technique would use them one after the other before eventually exploiting the search constraint(s). Every time well-understood combinatorial sub-structures (generally corresponding to global constraints) can be exploited through special-purpose techniques, the additive-bounding approach can become a powerful option.

7.2.2. Additive Bounding with Different Search Techniques. Finally, the link between search and bound described in this paper is not limited to DBS. We think that, though simple, the use of AB techniques to improve a combinatorial relaxation in the presence of discrepancy constraints is the first nontrivial idea in the direction of such a link. It can be applied in all cases in which the sub-trees can be represented by linear constraints, but at the same time it is somehow limited by the fact that the additive-bounding approach is worth using only if the relaxation involving the search constraints allows a combinatorial solution, i.e., a solution method not relying on general-purpose LP methods. Two examples in this direction are *no-good recording* and *local branching* (see Dechter 1999 and Fischetti and Lodi 2003, respectively, for details).

No-goods are configurations of assignments that cannot lead to any consistent solution. Therefore, it is quite easy to express such configurations with linear constraints. For example, if the no-good contains the configuration $(X_1, X_2, X_3) = (1, 4, 6)$ involving CP variables/values, the corresponding linear constraint is $x_{11} + x_{24} + x_{36} \leq 2$. As soon as no-goods are discovered during the search tree, they can be added to the linear relaxation or considered in the additive-bounding procedure. The challenge,

however, is finding a combinatorial relaxation that contains those no-goods and can be efficiently solved.

Local branching, instead, is a complete search method that explores sub-trees representing neighborhoods of a *feasible* incumbent solution: the neighborhood is defined similarly to the k-discrepancy constraint by allowing that *at most* k assignments of the incumbent solution be changed. Again, each search constraint (i.e., each neighborhood) can be easily represented through a linear constraint, and the link between search and bound can be implemented, with the additive bounding technique being preferable if a combinatorial relaxation is available. This is trivially the case if the neighborhood "*at most k*" is explored through LDS techniques, i.e., with k "equality" constraints.

7.3. Conclusion

In this paper we have demonstrated that the additive-bounding procedure can be used to accelerate the proof of optimality significantly in a DBS framework. In this way, the additive-bounding technique has been used to build a first non-trivial link between search and bound.

In particular, the case where the *AllDifferent* is an important part of the model has been studied and two different additive bounds where proposed and computationally evaluated.

It is important to notice that application of the additive bounding framework in the CP context is not limited to interaction between a single global constraint and the search strategy. Indeed, ABP can be applied to a sequence of structured relaxations corresponding to several global constraints and eventually linked to other suitable search strategies.

An extensive use of the additive bounding framework in CP and its application to real-world and complex problems for which CP is an efficient solution method is the topic of future research.

Acknowledgments

Part of this research was carried out when the first author was Herman Goldstine Fellow of the IBM T. J. Watson Research Center, whose support is strongly acknowledged. A preliminary version of the first part of this paper appeared in the Proceedings of "Principle and Practice of Constraint Programming" (Lodi et al. 2003). Thanks are due to Matteo Fischetti, Mauro Dell'Amico, and W. J. van Hoeve for valuable comments and helpful discussions. Thanks are also due to two anonymous referees for useful comments that helped make the presentation far more general.

References

Carpaneto, G., M. Dell'Amico, P. Toth. 1995. Exact solution of large-scale asymmetric traveling salesman problems. *ACM Trans. Math. Software* **21** 394–409.

Carpaneto, G., S. Martello, P. Toth. 1988. Algorithms and codes for the assignment problem. *Ann. Oper. Res.* **13** 193–223.

Cirasella, J., D. S. Johnson, L. A. McGeoch, W. Zhang. 2001. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. A. L. Buchsbaum, J. Snoeyink, eds. *Proc. ALENEX'01, LNCS* 2153. Springer-Verlag, Berlin and Heidelberg, 32–59.

Dechter, R. 1999. Constraint satisfaction. *The MIT Encyclopedia of Cognitive Sciences* (*MITECS*). MIT Press, Cambridge, MA, 195–197.

Dell'Amico, M., S. Martello. 1997a. Linear assignment. F. Maffioli, M. Dell'Amico, S. Martello, eds. *Annotated Bibliographies in Combinatorial Optimization*. Wiley, New York, 355–371.

Dell'Amico, M., S. Martello. 1997b. The k-cardinality assignment problem. *Discrete Appl. Math.* **76** 103–121.

Dell'Amico, M., A. Lodi, S. Martello. 2001. Efficient algorithms and codes for k-cardinality assignment problems. *Discrete Appl. Math.* **110** 25–40.

El Sakkout, H., M. Wallace. 2000. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints* **5** 359–388.

Fischetti, M., A. Lodi. 2003. Local branching. *Math. Programming* **98** 23–47.

Fischetti, M., P. Toth. 1989. An additive bounding procedure for combinatorial optimization problems. *Oper. Res.* **37** 319–328.

Fischetti, M., P. Toth. 1992. An additive bounding procedure for the asymmetric traveling salesman problem. *Math. Programming* **53** 173–197.

Fischetti, M., A. Lodi, P. Toth. 2002. Exact methods for the asymmetric traveling salesman problem. G. Gutin, A. Punnen, eds. *The Traveling Salesman Problems and its Variations*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 169–205.

Focacci, F. 2001. Solving combinatorial optimization problems in constraint programming. Ph.D. thesis, Dipartimento di Ingegneria, University of Ferrara, Italy.

Focacci, F., A. Lodi, M. Milano. 1999. Cost-based domain filtering. J. Jaffar, ed. *Principle and Practice of Constraint Programming—CP 1999, LNCS* 1713. Springer-Verlag, Berlin and Heidelberg, 189–203.

Focacci, F., A. Lodi, M. Milano. 2002. A hybrid exact algorithm for the TSPTW. *INFORMS J. Comput.* **14** 403–417.

Focacci, F., A. Lodi, M. Milano. 2003. Exploiting relaxations in CP. M. Milano, ed. *Constraint and Integer Programming: Toward a Unified Methodology*. Kluwer, Boston, MA, 137–167.

Harvey, W., M. Ginsberg. 1995. Limited discrepancy search. *Proc. 14th IJCAI*. Morgan Kaufmann, San Francisco, CA, 607–615.

Korf, R. 1996. Improved limited discrepancy search. *Proc. Thirteenth National Conf. Artificial Intelligence and the Eighth Innovative Appl. Artificial Intelligence Conf.* 1. The AAAI Press, Menlo Park, CA, 286–291.

Lodi, A., M. Milano, L.-M. Rousseau. 2003. Discrepancy-based additive bounding for the AllDifferent constraint. F. Rossi, ed. *Principle and Practice of Constraint Programming—CP 2003, LNCS* 2833. Springer-Verlag, Berlin and Heidelberg, 510–524.

Milano, M., W. J. van Hoeve. 2002. Reduced cost-based ranking for generating promising subproblems. P. Van Hentenryck, ed. *Principle and Practice of Constraint Programming—CP 2002, LNCS* 2470. Springer-Verlag, Berlin and Heidelberg, 1–16.

Pesant, G., M. Gendreau, J.-Y. Potvin, J.-M. Rousseau. 1998. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Sci.* **32** 12–29.

Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Oper. Res.* **35** 254–265.

van Hoeve, W. J., M. Milano. 2003. Decomposition-based search. A theoretical and experimental evaluation. LIA technical report, LIA00203, University of Bologna, Bologna, Italy.

Zhang, W. 2000. Depth-first branch-and-bound versus local search: A case study. *Proc. Seventeenth National Conf. on Artificial Intelligence*. The AAAI Press, Menlo Park, CA, 930–935.